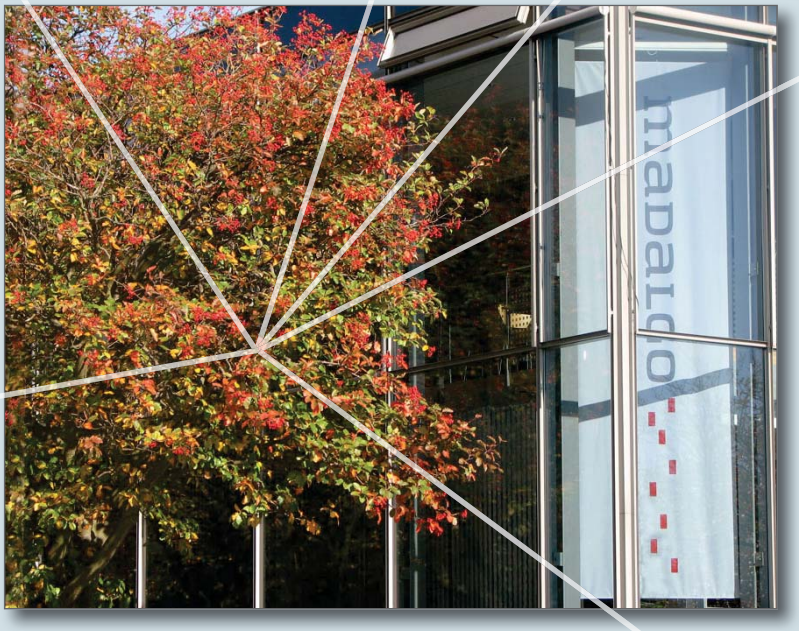
MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

AARHUS UNIVERSITY

Danmarks Grundforskningsfond
Danish National Research Foundation

FINAL REPORT

Center for Massive Data Algorithmics
2007 - 2016



When initiated in 2007, the ambitious goal of the Danish National Research Foundation Center for Massive Data Algorithmics (MADALGO) was to become a world-leading center in algorithms for handling massive data. The high-level objectives of the center were:

- To significantly advance the fundamental algorithms knowledge in the area of efficient processing of massive datasets;
- To educate the next generation of researchers in a world-leading and international environment;
- To be a catalyst for multidisciplinary collaborative research on massive dataset issues in commercial and scientific applications.

Building on the research strength at the main center site at Aarhus University in Denmark, and at the sites of the other participants at Max Planck Institute for Informatics and at Frankfurt University in Germany, and at Massachusetts Institute of Technology in the US, as well as on significant international research collaboration, multidisciplinary and industry collaboration, and in general on a vibrant international environment at the main center site, I believe we have met the goal.

Following the guidelines of the Foundation, this report contains a short reflection on the center's results, impact and international standing (section A), along with appendices with statistics on patents and spin-off companies (section A) and the statistics collected by the Foundation (section C). It also contains 10 publications reflecting the breadth and depth of center research.

Lars Arge
Center Director
May, 2016

A. Center results, impact and international standing

Center Results. A key motivation for establishment of Center for Massive Data Algorithmics was the inadequacy of traditional algorithms theory in many applications involving massive amounts of data. In particular, traditional algorithm theory use simplistic machine models (mathematical model in which algorithms are designed and analyzed) that do not take the hierarchical memory organization of modern machines into account. This translates into software inadequacies when processing massive data. Based on an ambitious research plan, center researchers have obtained a large number of results centered around four key areas that all relate to the design of algorithms in more realistic models of computations than traditional algorithm theory. Below we *briefly* highlight a few results (corresponding to the 10 selected papers).

In the area of ***I/O-efficient algorithms***, the goal is to design algorithms in a model of computation that takes the large access time of disk (compared to main memory) into account. Center results on fundamental problems in the area include results on sorting algorithms that are provably optimal in terms of both I/O and internal memory computation time (selected paper 1, which won the best paper awards at the 2013 ISAAC conference). Center researchers have also obtained results on processing of massive terrain data, including several results on modeling water flow on terrains, such as for example algorithms for computing how depressions in a terrain fills with water as it rains (paper 2, which appeared in the top computational geometry conference SoCG). Overall, we believe center work in the area has been very successful.

In the area of ***cache-oblivious algorithms,*** the aim is to develop algorithms that automatically adapt to the unknown multiple levels of modern memory hierarchies. Developing cache-oblivious algorithms is very challenging; at the start of the center, techniques for obtaining cache-oblivious algorithms was not very developed and the fundamental limitations in the area not well understood. Center researchers have obtained a good number of results in the area, and e.g. developed new fundamental techniques used to obtain an optimal cache-oblivious data structure that allows for fast search for a data element among a set of elements (paper 3, which appeared in the top algorithms conference SODA). However, overall we have not made as much progress on cache-oblivious problems as we would have liked.

In the area of ***streaming algorithms,*** the aim is to develop algorithms that only perform one sequential pass over the input data, while processing each input element as fast as possible using significantly less space than the input data size. Center researchers have obtained a large number of results in the area. For example, we have developed the first optimal algorithm for the well-studied and fundamental problem of estimating the number of distinct elements in a stream (paper 4, which won the best paper awards at the top theoretical database conference PODS in 2010). We have also obtained a large number of results in relation to the so-called linear sketching technique, and recently we showed the very first known lower bounds on how fast data stream elements can be processed (paper 5, which appeared in the top theoretical computer science conference STOC). Overall, we believe center work in the area has been very successful.

The area of ***algorithm engineering*** covers the design and analysis of practical algorithms, their efficient implementation, as well as experimentation that provides insight into their applicability and further improvement. Center researches have in particular engineered many I/O-efficient algorithms. For example, the center has had great success with engineering algorithms for flood risk analysis using detailed (and thus massive) terrain data. This work includes implementation of results on how depressions fill during a rain event mentioned above (paper 2) and recent work on modelling and computing flood risk from water rising in rivers (paper 6, which appear in the good geographic information systems conference GIScience). It also formed the basis of the center startup company SCALGO, which has very successfully marketed flood risk analysis products based on the openly available detailed terrain model of Denmark. For example, the depression risk analysis results (paper 2) formed the basis of analysis purchased by more than half of the local Danish governments. Center algorithm engineering work has also formed the basis for collaboration with several other companies, as well as for extensive collaboration with researchers in other fields, in particular biology (bioinformatics and biodiversity) researchers. The river flood risk research (paper 6) is e.g. performed with a number of biology researchers, and the collaboration has also led to a Science paper (paper 7). Overall, we believe center algorithm engineering work has been very successful.

In addition to the above four core areas, center researchers have obtained a large number of results on research questions in relation to e.g. models of ***parallel computation***, ***flash memory***, ***fault tolerance***, ***memory efficient (succinct) data structures***, as well as to fundamental problems in more traditional models of computation, in particular ***data structure*** and ***lower bound*** problems. As a few examples, center researchers have solved longstanding open problems in succinct (paper 8, which won the best student paper award at the top theoretical computer science conference FOCS in 2008) and priority queue (paper 9, which appeared in top theoretical computer science conferences STOC and is co-authored by Turing Award winner

Tarjan) data structures. Much of the center's data structure work (in various models of computation) has been on fundamental range searching problems (storing a set of d-dimensional points such that points in a query range can be identified efficiently), and we have e.g. showed very strong lower bound tradeoffs between search and update time for several variants of the problem (paper 10, which won both best paper and best student paper awards in the top theoretical computer science conferences STOC in 2012).

A key goal of the center has been to educate the next generation of researchers in a world-leading and international environment. Altogether, 28 PhD students have graduated from the center (19 from AU), and the center currently houses 14 PhD students (7 at AU). Among the 26 students at AU, 12 students are internationals. Additionally, the center has hosted 24 Post Doc (22 at AU). The center has emphasized creation of a vibrant and international environment at the main center site at AU. The center has hosted numerous research visitors, just as we have organized annual summer schools at the center. The summer school series has been very successful, with around 70 participants from 30+ institutions attending every year. The center has also hosted the top computational geometry conference (SoCG) in 2009, and is hosting the top European algorithms event ALGO (including the top European algorithms conference ESA) in 2016. The center also initiated Workshop on Massive Data Algorithmics (MASSIVE) in 2009 and the successful workshop has been held every year since in connection with SoCG or ALGO.

Center international standing and impact. Overall, we believe the center has been very successful. From a purely bibliometric point of view, this can e.g. be seen from the fact that in the last four years, center researchers have published 90+ peer-reviewed papers a year, with consistently 10+ papers in the top three general theoretical computer science conferences STOC, FOCS and SODA. According to google scholar, center papers have received 12.000+ citations (steadily increasing over the years and reaching 3000 in 2015). This is an exceptionally good record in theoretical computer science, and the center strength is confirmed by the many center best papers award, the around 30 annual invited presentations at research conferences, workshops and seminars given by center researchers, as well as by the numerous awards and recognitions received by center members. On a more subjective level, we believe that the center is among the top algorithms research centers in Europe (possibly only rivaled by the Max Planck Institute for Informatics). We believe the center is the top institution worldwide in I/O-efficient algorithms and possibly also in cache-oblivious algorithms, among the top in streaming algorithms and algorithm engineering, as well as among the top in data structures (including lower bounds) in general. This was confirmed by the international evaluation of the center in 2011, where the panel concluded that "MADALGO is a truly international research center with high visibility and respect in the worldwide computer-science community. There is no doubt that it is *the* world-leading center in the area of massive dataset algorithmics".

In terms of broader impact, we believe that the center has contributed significantly to the transformation of the algorithms field, from a very theoretical field where results are proved in elegant but somewhat unrealistic models of computation, to a field where practical applicability of theoretical results is a key focus. This general transformation has occurred worldwide during the last decade, and the center focus on new and more practically realistic models of computation has certainly contributed to the transformation. Similarly, the center's focus on understanding practical performance through algorithm engineering has contributed to the field of algorithm engineering gaining worldwide recognition. Supported by algorithm engineering work, the center has certainly also had an impact in terms of interdisciplinary and industry collaboration, including by contributing to the transformation of parts of the biological sciences to being more data driven, and by introducing innovative research based products in the marked through SCALGO (which e.g. also is involved in producing sea-charts for Greenland and contour maps for Denmark in collaboration with Danish authorities). On several occasions, the center's collaborative activities and success in transforming research into innovative products has been highlighted as examples of the broader impact of basic research, for example when the Minister for Higher Education and Science visited the center in 2013. In general, the center has prioritized outreach events and maintaining a good public visibility.
The center has also had an impact in terms of education of PhD students and Post Docs. The many center PhDs and Post Docs are now excellent ambassadors in research institutions and industry worldwide, and in general we believe that the international visibility and recognition of the center, also supported by the many international events the center has organized and participated in, will have a lasting impact on Danish algorithms research. Concretely, the center locally at AU contributed to development of the streaming algorithms area (which was not really present in Denmark at the start of the center), and is now contributing to buildup in related areas such as Machine Learning and "Big Data". For example, recent Big Data faculty candidates have specifically mentioned center strength and visibility as a reason for applying to AU.

B. Appendices

Appendix a: Patents and spin-off companies

Number of inventions reported to institution	Number of submitted patent applications	Number of granted patents	Number of mutually agreed licence, sale and option agreements	Names of spin-off companies established
				Scalable Algorithmics (SCALGO

Appendix b: Center publications

Number of publications (May 2016)	Peer-reviewed (OA)	Not peer-reviewed (OA)
Journal articles	204 (137) + 11 accepted	0
Conference proceedings	413 (226)	30 (20)
Monographs	0 (0)	2 (0)
Book chapters	2 (0)	0 (0)
Others	0 (0)	96 (24)

The 10 most prestegious conferences within the Center's research area

1. ACM Symposium on Theory of Computing (STOC)
2. IEEE Symposium on Foundations of Computer Science (FOCS)
3. ACM-SIAM Symposium on Discrete Algorithms (SODA)
4. Symposium on Computational Geometry (SoCG)
5. International Colloquium on Automata, Languages, and Programming (ICALP)
6. European Symposium on Algorithms (ESA)
7. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)
8. International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)/ International Workshop on Randomizationand Computation (RANDOM)
9. Scandinavian Workshop on Algorithm Theory (SWAT)/ Algorithms and Data Structures Symposium (WADS), previously Workshop on Algorithms and Data Structures
10. Workshop on Algorithm Engineering and Experiments (ALENEX)

The 10 most prestegious journals in the Center's research area

1. Journal of the ACM
2. SIAM Journal on Computing
3. ACM Transactions on Algorithms
4. Discrete & Computational Geometry
5. Algorithmica
6. Journal of Computer and System Sciences
7. Computational Geometry: Theory and Applications
8. ACM Journal of Experimental Algorithmics
9. Theoretical Computer Science
10. Journal of Discrete Algorithms

Bibliometric information

Distribution of center publications on 10 most prestigious conferences:

	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
STOC	1	0	0	0	3	4	1	1	4	?
FOCS	1	7	3	2	3	1	0	3	4	?
SODA	0	6	5	6	6	11	11	7	5	5
SoCG	0	5	3	3	2	3	2	2	3	5
ICALP	1	0	7	1	3	2	5	4	0	4
ESA	3	1	0	3	1	5	3	2	2	?
SPAA	3	1	0	3	1	1	0	2	0	?
APPROX/RANDOM	0	1	0	1	3	0	0	0	0	?
SWAT/WADS	1	3	6	1	3	1	2	2	5	?
ALENEX	0	0	1	0	1	0	1	2	1	1

STOC, FOCS and SODA can be rated as "best non specialized" conferences
SoCG and ALENEX can be rated as "best specialized" conferences

Center publications have been authored by 924 unique authors - 130 associated with the center and 794 not.
Only 203 center publications are by center researchers only.
Citations to center publication (according to Google scholar, which is the most reliable - but certainly not perfect - source of citation information in the area) can be found at
http://scholar.google.com/citations?user=fRowhXcAAAAJ

Conference proceedings

C1	2007	B. Escoffier, G. Moruz and A. Ribichini	Adapting Parallel Algorithms to the W-Stream Model, with Applications to Graph Problems	Proc. International Symposium on Mathematical Foundations of Computer Science (MFCS)	(PR)(CO)
C2	2007	S. Guha, P. Indyk and A. McGregor	Sketching Information Divergences	Proc. Annual Conference on Learning Theory (COLT)	(PR)(CO)
C3	2007	G. S. Brodal and A. G. Jørgensen	A Linear Time Algorithm for the k Maximal Sums Problem	Proc. International Symposium on Mathematical Foundations of Computer Science (MFCS)	(PR)(CO)
C4	2007	G. S. Brodal, L. Georgiadis, K. A. Hansen and I. Katriel	Dynamic Matchings in Convex Bipartite Graphs	Proc. International Symposium on Mathematical Foundations of Computer Science (MFCS)	(PR)(CO)
C5	2007	G. Jørgensen, G. Moruz and T. Mølhave	Resilient Priority Queues	Proc. International Workshop on Algorithms and Data Structures (WADS)	(PR)
C6	2007	G. S. Brodal, R. Fagerberg, I. Finocchi, F. Grandoni, G. Italiano, A. G. Jørgensen, G. Moruz and T. Mølhave	Optimal Resilient Dynamic Dictionaries	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)
C7	2007	P. K. Agarwal, L. Arge, A. Danner, H. Mitasova, T. Mølhave and K. Yi	TerraStream: From Elevation Data to Watershed Hierarchies	Proc. ACM International Symposium on Advances in Geographical Information Systems (ACM-GIS)	(PR)(CO)
C8	2007	M. Patrascu and Mikkel Thorup	Planning for Fast Connectivity Updates	Proc. IEEE Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)
C9	2007	G. Franceschini, S. Muthukrishnan, and M. Patrascu	Radix Sorting With No Extra Space	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)
C10	2007	E. D. Demaine, S. Mozes, B. Rossman and O. Weimann	An Optimal Decomposition Algorithm for Tree Edit Distance	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)
C11	2007	M. A. Bender, M. Farach-Colton, J. T. Fineman, Y. Fogel, B. C. Kuszmaul and J. Nelson	Cache-Oblivious Streaming B-trees	Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)	(PR)(CO)
C12	2007	E. D. Demaine, M. Ghodsi, M. Hajiaghayi, A. S. Sayedi-Roshkhar and M. Zadimoghaddam	Scheduling to Minimize Gaps and Power Consumption	Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)	(PR)(CO)
C13	2007	M. Patrascu	Lower Bounds for 2-Dimensional Range Counting	Proc. ACM Symposium on Theory of Computing (STOC)	(PR)
C14	2007	G. M. Landau, D. Tsur and O. Weimann	Indexing a Dictionary for Subset Matching Queries	Proc. Symposium on String Processing and Information Retrieval (SPIRE)	(PR)(CO)
C15	2007	T. Friedrich and D. Ajwani	Average-Case Analysis of Online Topological Ordering	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)
C16	2007	K. Chang	Multiple pass streaming algorithms for learning mixtures of distributions in R^d	Proc. Algorithmic Learning Theory (ALT)	(PR)

C17	2007	M. Westergaard, L. M. Kristensen, G. S. Brodal and L. Arge	The ComBack Method - Extending Hash Compaction with Backtracking	Proc. International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency (ICATPN)	(PR)
C18	2007	M. A. Bender, G. S. Brodal, R. Fagerberg, R. Jacob and E. Vicari	Optimal Sparse Matrix Dense Vector Multiplication in the I/O-Model	Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)	(PR)(CO)
C19	2007	A. Golynski, R. Grossi, A. Gupta, R. Raman and S. S. Rao	On the Size of Succinct Indices	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)
C20	2007	M. Olsen	Nash Stability in Additively Separable Hedonic Games is NP-hard	Proc. Conference on Computability in Europe (CiE)	(PR)
C21	2008	M. Ruzic and P. Indyk	Near-Optimal Sparse Recovery in the L1 norm	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)
C22	2008	M. Patrascu	(Data) STRUCTURES	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)
C23	2008	M. Patrascu	Succincter	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)
C24	2008	E. Demaine, S. Langerman and E. Price	Confluently Persistent Tries for Efficient Version Control	Proc. Scandinavian Workshop on Algorithm Theory (SWAT)	(PR)(CO)
C25	2008	D. Ajwani, I. Malinger, U. Meyer and S. Toledo	Characterizing the Performance of Flash Memory Storage Devices and Its Impact on Algorithm Design	Proc. Workshop on Experimental Algorithms (WEA)	(PR)(CO)
C26	2008	U. Meyer	On Dynamic Breadth-First Search in External-Memory	Proc. Symposium on Theoretical Aspects (STACS)	(PR)
C27	2008	U. Meyer	On Trade-Offs in External-Memory Diameter Approximation	Proc. Scandinavian Workshop on Algorithm Theory (SWAT)	(PR)
C28	2008	G. S. Brodal and A. G. Jørgensen	Selecting Sums in Arrays	Proc.International Symposium on Algorithms and Computation (ISAAC)	(PR)
C29	2008	L. Arge, G. S. Brodal and S. S. Rao	External Memory Planar Point Location with Logarithmic Updates	Proc. Symposium on Computational Geometry (SoCG)	(PR)
C30	2008	A. Golynski, R. Raman and S. S. Rao	On the Redundancy of Succinct Data Structures	Proc. Scandinavian Workshop on Algorithm Theory (SWAT)	(PR)(CO)
C31	2008	M. Olsen	The Computational Complexity of Link Building	Proc. International Conference on Computing and Combinatorics (COCOON)	(PR)
C32	2008	M.A. Abam, M. de Berg and J. Gudmundsson	A Simple and Efficient Kinetic Spanner	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)
C33	2008	L. Arge, M.T. Goodrich, M. Nelson and N. Sitchinava	Fundamental Parallel Algorithms for Private-Cache Chip Multiprocessors	Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA)	(PR)(CO)
C34	2008	L. Arge, T. Moelhave and N. Zeh	Cache-Oblivious Red-Blue Line Segment Intersection	Proc. European Symposium on Algorithm (ESA)	(PR)(CO)
C35	2008	P.K. Agarwal, L. Arge, T. Moelhave and B. Sadri	I/O-efficient Algorithms for Computing Contour Lines on a Terrain	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)
C36	2008	J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein and Z. Svitkina	On Distributing Symmetric Streaming Computations	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C37	2008	P. Indyk	Explicit Constructions for Compressed Sensing of Sparse Signals	Proc Symposium on Discrete Algorithms (SODA)	(PR)

C38	2008	A. Andoni, P. Indyk and R. Krauthgamer	Earth Mover Distance over High-Dimensional Spaces	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C39	2008	P. Indyk and A. McGregor	Declaring Independence via the Sketching of Sketches	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C40	2008	K. Onak and A. Sidiropoulos	Circular Partitions with Applications to Visualization and Embeddings	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)
C41	2008	J. Matousek and A. Sidiropoulos	Inapproximability for metric embeddings into R^d	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)
C42	2008	N. J. A. Harvey, J. Nelson and K. Onak	Sketching and Streaming Entropy via Approximation Theory	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)
C43	2008	A. Andoni, D. Croitoru and M. Patrascu	Hardness of Nearest Neighbor under L-infinity	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)
C44	2008	T. Chan, M. Patrascu and L. Roditty	Dynamic Connectivity: Connecting to Networks and Geometry	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)
C45	2008	S. Mozes, K. Onak and Oren Weimann	Finding an Optimal Tree Searching Strategy in Linear Time	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C46	2008	A. Chakrabarti, T.S. Jayram and M. Patrascu	Tight Lower Bounds for Selection in Randomly Ordered Streams	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C47	2008	E. Demaine, T. Ito, Ni. J. A. Harvey, C. H. Papadimitriou, M. Sideri, R. Uehara and Yushi Uno	On the Complexity of Reconfiguration Problems	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)(CO)
C48	2008	E. Demaine, G. Aloupis, S. Collette, S. Langerman, V. Sacristan and S. Wuhrer	Reconfiguration of Cube-Style Modular Robots Using O(log n) Parallel Moves	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)(CO)
C49	2008	E. Demaine, M. Buadoiu, M. Hajiaghayi, A. Sidiropoulos and M. Zadimoghaddam	Ordinal Embedding: Approximation Algorithms and Dimensionality Reduction	Proc. International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)	(PR)(CO)
C50	2008	E. Demaine, T. G. Abbott, Z. Abel, D. Charlton, M. L. Demaine and S. D. Kominers	Hinged Dissections Exist	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)
C51	2008	E. R. Hansen, S. S. Rao and P. Tiedemann	Compressing Binary Decision Diagrams	European Conference on Artificial Intelligence (ECAI)	(PR)(CO)
C52	2008	R. Berinde, P. Indyk and M. Ruzic	Practical Near-Optimal Sparse Recovery in the L1 Norm (invited paper)	Proc. Allerton Conference	(CO)
C53	2008	R. Berinde, A. Gilbert, P. Indyk, H. Karloff and M. Strauss	Combining Geometry and Combinatorics: A Unified Approach to Sparse Signal Recovery (invited paper)	Proc. Allerton Conference	(CO)
C54	2008	M.A. Abam, M. de Berg, and S-H. Poon	Fault-Tolerant Conflict-Free Coloring	Proc. Canadian Conference on Computational Geometry	(CO)
C55	2009	R. Berinde, G. Cormode, P. Indyk and M. Strauss	Space-optimal Heavyhitters with Strong Error Bounds	Proc. Symposium on Principles of Database Systems (PODS)	(PR)(CO)
C56	2009	V. Cevher, C. Hegde, P. Indyk and R. G. Baraniuk	Recovery of Clustered Sparse Signal from Compressive Measurements	Proc. International Conference on Sampling Theory and Applications (SAMPTA)	(PR)(CO)

C57	2009	E. Demaine, G. Landau and O. Weimann	On Cartesian Trees and Range Minimum Queries	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)
C58	2009	D. Hermelin, G. M. Landau, S. Landau and O. Weimann	A Unified Algorithm for Accelerating Edit-Distance Computation via Text-Compression	Proc. International Symposium on Theoretical Aspects of Computer Science (STACS)	(PR)(CO)
C59	2009	A. Kovacs, U. Meyer, G. Moruz and A. Negoescu	Online Paging for Flash Memory Devices	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)
C60	2009	G. Brodal, A. Jørgensen, G. Moruz and T. Mølhave	Counting in the Presence of Memory Faults	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)
C61	2009	D. Ajwani, A. Beckmann, R. Jacob, U. Meyer and G. Moruz	On Computational Models for Flash Memory Devices	Proc. Symposium on Experimental Algorithms (SEA)	(PR)(CO)
C62	2009	U. Meyer and V. Osipov	Design and Implementation of a Practical I/O-efficient Shortest Paths Algorithm	Proc. Workshop on Algorithm Engineering and Experiments (ALENEX)	(PR)
C63	20009	U. Meyer	Via Detours to I/O-Efficient Shortest Paths	Proc. Efficient Algorithms - Essays dedicated to Kurt Mehlhorn on the Occasion of his 60th birthday	
C64	2009	D. Ajwani, R. Dementiev, U. Meyer and V. Osipov	Breadth First Search on Massive Graphs	Proc. Ninth DIMACS Implementation Challenge: The Shortest Path Problem	(PR)
C65	2009	A. Beckmann, R. Dementiev and J. Singler	Building a Parallel Pipelined External Memory Algorithm Library	Proc. International Symposium on Parallel and Distributed Processing (IPDPS)	(PR)
C66	2009	G. S. Brodal and A. Jørgensen	Data Structures for Range Median Queries	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)
C67	2009	G. S. Brodal, R. Fagerberg, M. Greve and A. López-Ortiz	Online Sorted Range Reporting	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)(CO)
C68	2009	G. S. Brodal, A. Kaporis, S. Sioutas, K. Tsakalidis and K. Tsichlas	Dynamic 3-sided Planar Range Queries with Expected Doubly Logarithmic Time	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)(CO)
C69	2009	G. S. Brodal, A. Jørgensen and T. Mølhave	Fault Tolerant External Memory Algorithms	Proc. Algorithms and Data Structures Symposium (WADS)	(PR)
C70	2009	A. Kaporis, A.N. Papadopoulos, S. Sioutas, K. Tsakalidis and K. Tsichlas	Efficient Processing of 3-Sided Range Queries with Probabilistic Guarantees	Proc. International Conference on Database Theory (ICDT)	(PR)(CO)
C71	2009	M. Abam, M. de Berg, M. Farshi, J. Gudmundsson and M. Smid	Geometric Spanners for Weighted Point Sets	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)
C72	2009	M. Abam and M. de Berg	Kinetic Spanners in R^d	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)
C73	2009	M. Abam, P. Carmi, M. Farshi and M. Smid	On the Power of the Semi-Separated Pair Decomposition	Proc. Algorithms and Data Structures Symposium (WADS)	(PR)(CO)
C74	2009	D. Ajwani	On P-complete Problems in Memory Hierarchy Models	Proc. Workshop on Massive Data Algorithmics (MASSIVE)	
C75	2009	A. Farzan, R. Raman and S. Srinivasa Rao	Universal Succinct Representations of Trees?	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)
C76	2009	R. Pagh and S. Srinivasa Rao	Secondary Indexing in One Dimension: Beyond B-trees and Bitmap Indexes	Proc. Symposium on Principles of Database Systems (PODS)	(PR)(CO)

C77	2009	R. Grossi, A. Orlandi, R. Raman and S. Srinivasa Rao	More Haste, Less Waste: Lowering the Redundancy in Fully Indexable Dictionaries	Proc. International Symposium on Theoretical Aspects of Computer Science (STACS)	(PR)(CO)
C78	2009	J. E. Moeslund, P. K. Bøcher, J.-C. Svenning, T. Mølhave and L. Arge	Impacts of 21st Century Sea-level Rise on a Danish Major City – An Assessment Based on Fine-resolution Digital Topography and a New Flooding Algorithm	IOP Conference Series: Earth and Environmental Science 8	(PR)
C79	2009	M. de Berg and P. Hachenberger	Rotated-Box Trees: A Lightweight c-Oriented Bounding-Volume Hierarchy	Proc. International Symposium on Experimental Algorithms (SEA)	(PR)(CO)
C80	2009	P. Afshani, L. Arge and K. Dalgaard Larsen	Orthogonal Range Reporting in Three and Higher Dimensions	Proc Symposium on Foundations of Computer Science (FOCS)	(PR)
C81	2009	P. Afshani, C. Hamilton and N. Zeh	A Unified Approach for Cache-Oblivious Range Reporting and Approximate Range Counting	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)
C82	2009	P. Afshani, C. Hamilton and N. Zeh	Cache-Oblivious Range Reporting With Optimal Queries Requires Superlinear Space	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)
C83	2009	P. Afshani, J. Barbay and T. Chan	Instance-optimal Geometric Algorithms	Proc Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)
C84	2009	L. Arge, M.T. Goodrich and N. Sitchinava	Parallel External Memory Model	Proc. Workshop on Theory and Many-Cores	
C85	2009	L. Arge and M. Revsbæk	I/O-Efficient Contour Tree Simplification	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)
C86	2009	A. Andoni, P. Indyk, R. Krauthgamer and H.L. Nguyen	Approximate Line Nearest Neighbor in High Dimensions	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C87	2009	A. Andoni, P. Indyk and R. Krauthgamer	Overcoming the L1 Non-embeddability Barrier: Algorithms for Product Metrics	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C88	2009	R. Berinde and P. Indyk	Sequential Sparse Matching Pursuit	Proc. Allerton Conference	(PR)(CO)
C89	2009	A. Andoni, K. Do Ba, P. Indyk and D. Woodruff	Efficient Sketches for Earth-Mover Distance, with Applications	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)
C90	2009	A. Andoni, P. Indyk, K. Onak and R. Rubinfeld	External Sampling	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)
C91	2009	E. Demaine, M. Demaine, G. Konjevod and R. Lang	Folding a Better Checkerboard	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)(CO)
C92	2009	J. Cardinal, E. Demaine, M. Demaine, S. Imahori, S. Langerman and R. Uehara	Algorithmic Folding Complexity	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)(CO)
C93	2009	E. Demaine, M. Hajiaghayi, and D. Marx	Minimizing Movement: Fixed-Parameter Tractability	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)
C94	2009	B. Ballinger, D. Charlton, E. Demaine, M. Demaine, J. Iacono, C-H. Liu and S-H. Poon	Minimal Locked Trees	Proc. Algorithms and Data Structures Symposium (WADS)	(PR)(CO)
C95	2009	E. Demaine, D. Kane and G. Price	A Pseudopolynomial algorithm for Alexandrov's Theorem	Proc. Algorithms and Data Structures Symposium (WADS)	(PR)(CO)

C96	2009	T. Ito, M. Kaminski and E. Demaine	Reconfiguration of List Edge-Colorings in a Graph	Proc. Algorithms and Data Structures Symposium (WADS)	(PR)(CO)
C97	2009	E. Demaine, M. Hajiaghayi and K. Kawarabayashi	Approximation Algorithms via Structural Results for Apex-Minor-Free Graphs	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)
C98	2009	E. Demaine, M. Hajiaghayi and P. Klein	Node-Weighted Steiner Tree and Group Steiner Tree in Planar Graphs	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)
C99	2009	E. Demaine, G. Borradaile and S. Tazari	Polynomial-Time Approximation Schemes for Subset-Connectivity Problems in Bounded-Genus Graphs	Proc. International Symposium on Theoretical Aspects of Computer Science (STACS)	(PR)(CO)
C100	2009	E. Demaine, D. Harmon, J. Iacono, D. Kane and M. Patrascu	The Geometry of Binary Search Trees	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C101	2009	E. Demaine, K. Kawarabayashi and M. Hajiaghayi	Additive Approximation Algorithms for List-Coloring Minor-Closed Class of Graphs	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C102	2009	E. Demaine, M. Hajiaghayi, H. Mahini and M. Zadimoghaddam	The Price of Anarchy in Cooperative Network Creation Games	Proc. International Symposium on Theoretical Aspects of Computer Science (STACS)	(PR)(CO)
C103	2009	J. Cardinal, E. Demaine, S. Fiorini, G. Joret, I. Newman and O. Weimann	The Stackelberg Minimum Spanning Tree Game on Planar and Bounded-Treewidth Graphs	Proc. Workshop on Internet and Network Economics (WINE)	(PR)(CO)
C104	2009	J. McLurkin and E. Demaine	A Distributed Boundary Detection Algorithm for Multi-Robot Systems	Proc. International Conference on Intelligent Robots and Systems	(PR)(CO)
C105	2009	G. Aloupis, N. Benbernou, M. Damian, E. Demaine, R. Flatland, J. Iacono and S. Wuhrer	Efficient Reconfiguration of Lattice-Based Modular Robots	Proc. European Conference on Mobile Robots	(PR)(CO)
C106	2009	M. Ajtai, V. Feldman, A. Hassidim and J. Nelson	Sorting and Selection with Imprecise Comparisons	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)
C107	2009	R. Yuster and O. Weimann	Computing the Girth of a Planar Graph in O(n log n) time	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)
C108	2009	R. Backofen, G. Landau, M. Möhl, D. Tsur and O. Weimann	Fast RNA Structure Alignment for Crossing Input Structures	Proc. Symposium on Combinatorial Pattern Matching (CPM)	(PR)(CO)
C109	2009	P. Klein, S. Mozes and O. Weimann	Shortest Paths in Directed Planar Graphs with Negative Lengths: A Linear-Space O(n log n)-Time Algorithm	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C110	2010	K. Do Ba, P. Indyk, E. Price and D.P. Woodruff	Lower Bounds for Sparse Recovery	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C111	2010	P. Indyk, H.Q. Ngo and A. Rudra	Efficiently Decodable Non-adaptive Group Testing	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C112	2010	D.M. Kane, J. Nelson and D.P. Woodruff	An Optimal Algorithm for the Distinct Elements Problem	Proc. Symposium on Principles of Database Systems (PODS)	(PR)(CO)
C113	2010	J. Nelson and D.P. Woodruff	Fast Manhattan Sketches in Data Streams	Proc. Symposium on Principles of Database Systems (PODS)	(PR)(CO)
C114	2010	I. Diakonikolas, D.M. Kane and J. Nelson	Bounded Independence Fools Degree-2 Threshold Functions	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)

C115	2010	D.M. Kane, J. Nelson and D.P. Woodruff	On the Exact Space Complexity of Sketching and Streaming Small Norms	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C116	2010	A. Beckmann , U. Meyer, P. Sanders and J. Singler	Energy-Efficient Sorting using Solid State Disks	Proc. International IEEE Green Computing Conference	(PR)(CO)
C117	2010	M. Greve, A.G. Jørgensen, K.D. Larsen and J. Truelsen	Cell Probe Lower Bounds and Approximations for Range Mode	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)
C118	2010	M. Olsen	Maximizing PageRank with new Backlinks	Proc. International Conference on Algorithms and Complexity (CIAC)	(PR)
C119	2010	G.S. Brodal, E. Demaine, J. T. Fineman, J. Iacono, S. Langerman and J.I. Munro	Cache-Oblivious Dynamic Dictionaries with Optimal Update/Query Tradeoff	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C120	2010	A. Kaporis, A.N. Papadopoulos, S. Sioutas, K. Tsakalidis and K. Tsichlas	Efficient Processing of 3-Sided Range Queries with Probabilistic Guarantees	Proc. International Conference on Database Theory (ICDT)	(PR)(CO)
C121	2010	M.A. Abam and S. Har-Peled	New constructions of SSPDs and their applications	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)
C122	2010	M.B. Kjærgaard, H. Blunck, T. Godsk, T. Toftkjær, D.L. Christensen, and K. Grønbæk	Indoor Positioning using GPS Revisited	Proc. International Conference on Pervasive Computing (Pervasive)	(PR)
C123	2010	L. Arge, M.T. Goodrich and N. Sitchinava	Parallel external memory graph algorithms	Proc. International Parallel & Distributed Processing Symposium (IPDPS)	(PR)(CO)
C124	2010	P. Afshani, L. Arge and K.D. Larsen	Orthogonal Range Reporting: Query Lower Bounds, Optimal Structures in 3-d, and Higher Dimensional Improvements	Proc. Symposium on Computational Geometry (SoCG)	(PR)
C125	2010	P. Afshani, L. Arge and K.D Larsen	I/O-Efficient Orthogonal Range Reporting in Three and Higher Dimensions	Proc. Workshop on Massive Data Algorithmics (MASSIVE)	
C126	2010	T. Mølhave, P.K. Agarwal, L. Arge and M. Revsbæk	Scalable Algorithms for Large High-Resolution Terrain Data	Proc. International Conference on Computing for Geospatial Research & Application (COM.GEO)	(PR)(CO)
C127	2010	L. Arge, M. Revsbæk and Norbert Zeh	I/O-Efficient Computation of Water Flow Across a Terrain	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)
C128	2010	G.S. Brodal, P. Davoodi and S.S. Rao	On Space Efficient Two Dimensional Range Minimum Data Structures	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)
C129	2010	D. Ajwani, N. Sitchinava and N. Zeh	Geometric Algorithms for Private-Cache Chip Multiprocessors	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)
C130	2010	Z. Abel, N. Benbernou, M. Damian, E.D. Demaine, M.L. Demaine, R. Flatland, S. Kominers and R. Schwelle	Shape Replication Through Self-Assembly and RNase Enzymes	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C131	2010	E.D. Demaine, M. Hajiaghayi and K. Kawarabayashi	Decomposition, Approximation, and Coloring of Odd-Minor-Free Graphs	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)

C132	2010	N. Gershenfeld, D. Dalrymple, K. Chen, A. Knaian, F. Green, E.D. Demaine, S. Greenwald and P. Schmidt-Nielsen	Reconfigurable Asynchronous Logic Automata	Proc. Symposium on Principles of Programming Langauges (POPL)	(PR)(CO)
C133	2010	G. Aloupis, J. Cardinal, S. Collette, E.D. Demaine, M.L. Demaine, M. Dulieu, R. Fabila-Monroy, V. Hart, F. Hurtado, S. Langerman, M. Saumell, C. Seara and P. Taslakian	Matching Points with Things	Proc. Latin American Theoretical Informatics Symposium (LATIN)	(PR)(CO)
C134	2010	E.D. Demaine and M. Zadimoghaddam	Scheduling to Minimize Power Consumption using Submodular Functions	Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA)	(PR)
C135	2010	S. Gilbert, R. Guerraoui, F. Malakouti and M. Zadimoghaddam	Collaborative Scoring in the Presence of Malicious Players	Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA)	(PR)(CO)
C136	2010	N. Alon, E.D. Demaine, M. Hajiaghayi and T. Leighton	Basic Network Creation Games	Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA)	(PR)(CO)
C137	2010	E.D. Demaine and M. Zadimoghaddam	Minimizing the Diameter of a Network using Shortcut Edge	Proc. Scandinavian Workshop on Algorithm Theory (SWAT)	(PR)
C138	2010	M. Bateni, M.H. Hajiaghayi and M. Zadimoghaddam	Submodular Secretary Problem and Extensions	Proc. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)	(PR)(CO)
C139	2010	B. Ballinger, N. Benbernou, P. Bose, M. Damian, E.D. Demaine, V. Dujmović, R. Flatland, F. Hurtado, J. Iacono, A. Lubiw, P. Morin, V. Sacristán, D. Souvaine and R. Uehara	Coverage with k-Transmitters in the Presence of Obstacles	Proc. International Conference on Combinatorial Optimization and Applications (COCOA)	(PR)(CO)
C140	2010	E.D. Demaine and M. Zadimoghaddam	Constant Price of Anarchy in Network Creation Games via Public Service Advertising	Proc. International Workshop on Algorithms and Models for the Web-Graph	(PR)
C141	2010	G. S. Brodal, C. Kejlberg-Rasmussen and J. Truelsen	A Cache-oblivious Implicit Dictionary with the Working Set Property	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)
C142	2010	L. Arge, K. D. Larsen, T. Mølhave and F. van Walderveen	Cleaning Massive Sonar Point Clouds	Proc. International Conference on Advances in Geographic Information System (ACM-GIS)	(PR)
C143	2010	G.S Brodal, Ss.Sioutas, K. Tsichlas and C. Zaroliagis	D2-Tree: A New Overlay with Deterministic Bounds	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)(CO)
C144	2010	F. Gieseke, G. Moruz and J. Vahrenhold	Resilient kd-trees: K-means in space revisited	Proc. Conference on Data Mining (ICDM)	(PR)(CO)
C145	2010	J. Brody and E. Verbin	The Coin Problem and Pseudorandomness for Branching Programs	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)
C146	2011	H. Blunck, M. B. Kjærgaard and T. S. Toftegaard	Sensing and Classifying Impairments of GPS Reception on Mobile Devices	Proc. International Conference on Pervasive Computing (Pervasive)	(PR)(CO)
C147	2011	A. G. Jorgensen and K. G. Larsen,	Range Selection and Median: Tight Cell Probe Lower Bounds and Adaptive Data Structures	Proc. Symposium on Discrete Algorithms (SODA)	(PR)

C148	2011	P. Afshani, P. K. Agarwal, L. Arge, K. G. Larsen and J. M. Phillips	(Approximate) Uncertain Skylines	Proc. International Conference on Database Theory (ICDT)	(PR)(CO)
C149	2011	T. M. Chan, K. G. Larsen and M. Patrascu	Orthogonal Range Searching on the RAM, Revisited	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)
C150	2011	K. G. Larsen	On Range Searching in the Group Model and Combinatorial Discrepancy	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)
C151	2011	M. de Berg and C. Tsirogiannis	Exact and Approximate Computations of Watersheds on Triangulated Terrains	Proc. International Conference on Advances in Geographic Information Systems (ACM-GIS)	(PR)(CO)
C152	2011	H.Haverkort and C. Tsirogiannis	Flow on Noisy Terrains: An Experimental Evaluation	Proc. International Conference on Advances in Geographic Information Systems (ACM-GIS)	(PR)(CO)
C153	2011	D. Ajwani, N. Sitchinava and N. Zeh	I/O-Optimal Distribution Sweeping on Private-Cache Chip Multiprocessors	Proc. International Symposium on Parallel and Distributed Processing (IPDPS)	(PR)(CO)
C154	2011	M.T. Goodrich, N. Sitchinava and Q. Zhang	Sorting, Searching, and Simulation in the MapReduce Framework	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)(CO)
C155	2011	M. A. Abam, S. Daneshpajouh, L. Deleuran, S. Ehsani and M. Ghodsi	Computing Homotopic Line Simplification in a Plane	Proc. European Workshop on Computational Geometry (EuroCG)	(CO)
C156	2011	P. Afshani and N. Zeh	Improved Space Bounds for Cache-Oblivious Range Reporting	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C157	2011	P. Afshani, G.S. Brodal and N. Zeh	Ordered and Unordered Top-K Range Reporting in Large Data Sets	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C158	2011	G.S. Brodal, G. Moruz, and A. Negoescu	OnlineMin: A Fast Strongly Competitive Randomized Paging Algorithm	Proc. Workshop on Approximation and Online Algorithms (WAOA)	(PR)
C159	2011	G.S. Brodal, P. Davoodi, and S.S. Rao	Path Minima Queries in Dynamic Weighted Trees	Proc. Workshop on Algorithms and Data Structures (WADS)	(PR)(CO)
C160	2011	G.S. Brodal and K. Tsakalidis	Dynamic Planar Range Maxima Queries	Proc. International Colloquium on Automata, Languages, and Programming (ICALP)	(PR)
C161	2011	G.S. Brodal, M. Greve, V. Pandey and S.S. Rao	Integer Representations towards Efficient Counting in the Bit Probe Model	Proc. Conference on Theory and Applications of Models of Computation (TAMC)	(PR)(CO)
C162	2011	H.L. Chan, T.W. Lam, L.K. Lee, J. Pan, H.F. Ting and Q. Zhang	Edit Distance to Monotonicity in Sliding Windows	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR) (CO)
C163	2011	D. Ajwani, A. Cosgaya-Lozano and N. Zeh	Engineering a Topological Sorting Algorithm for Massive Graphs	Proc. Workshop on Algorithm Engineering and Experiments (ALENEX)	(PR)(CO)
C164	2011	S.H. Chan, T.W. Lam, L.K. Lee, C.M. Liu and H.F. Ting	Sleep management on multiple machines for energy and flow time	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR) (CO)
C165	2011	A.G. Jørgensen, M. Loffler and J. Phillips	Geometric Computations on Indecisive Points	Proc. International Workshop on Algorithms and Data Structures (WADS)	(PR)(CO)
C166	2011	P. Davoodi and S. Srinivasa Rao	Succinct Dynamic Cardinal Trees with Constant Time Operations for Small Alphabet	Proc. Theory and Applications of Models of Computation (TAMC)	(PR)(CO)

C167	2011	E. Verbin and W. Yu	The Streaming Complexity of Cycle Counting, Sorting By Reversals, and Other Problems	Proc. Symposium on Discrete Algorithms (SODA)	(PR)
C168	2011	U. Meyer, A. Negoescu and V. Weichert	New bounds for old algorithms: On the average-case behavior of classic single-source shortest path approaches	Proc. Conference on Theory and Practice of Algorithms in (Computer) Systems (TAPAS)	(PR)
C169	2011	M. Manjunath, K. Mehlhorn, K. Panagiotou and H. Sun	Approximate Counting of Cycles in Streams	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)
C170	2011	E. Price	Efficient Sketches for the Set Query Problem	Proc. Symposium on Discrete Algorithms (SODA)	(PR)
C171	2011	P. Indyk and E. Price	K-Median Clustering, Model-Based Compressive Sensing, and Sparse Recovery for Earth Mover Distance	Proc. Symposium on Theory of Computing (STOC)	(PR)
C172	2011	P. Indyk, E. Price and D. P. Woodruff	On the Power of Adaptivity in Sparse Recovery	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)
C173	2011	R. Gupta, P. Indyk, E. Price and Y. Rachlin	Compressive Sensing with Local Geometric Features	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)
C174	2011	E. Price and D. P. Woodruff	(1+eps)-approximate sparse recovery	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)
C175	2011	D. M. Kane, J. Nelson, E. Porat and D. P. Woodruff	Fast Moment Estimation in Data Streams in Optimal Space	Proc. Symposium on Theory of Computing (STOC)	(PR)(CO)
C176	2011	D. M. Kane, R. Meka and J. Nelson	Almost Optimal Explicit Johnson-Lindenstrauss Transformations	Proc. International Workshop on Randomization and Computation (RANDOM)	(PR)(CO)
C177	2011	D. B. Khanh and P. Indyk	Sparse recovery with partial support knowledge	Proc. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)	(PR)(CO)
C178	2011	K. Kawarabayashi, P. N. Klein and C. Sommer	Linear-Space Approximate Distance Oracles for Planar, Bounded-Genus, and Minor-Free Graphs	Proc. International Colloquium on Automata, Languages, and Programming (ICALP)	(PR)(CO)
C179	2011	C. Gavoille and C. Sommer	Sparse Spanners vs. Compact Routing	Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA)	(PR)(CO)
C180	2011	H. N. Djidjev and C. Sommer	Approximate Distance Queries for Weighted Polyhedral Surfaces	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)
C181	2011	D. Alistarh, J. Aspnes, K. Censor-Hillel, S. Gilbert and M. Zadimoghaddam	Optimal-Time Adaptive Tight Renaming, with Applications to Counting	Proc. Symposium on Principles of Distributed Computing (PODC)	(PR)(CO)
C182	2011	A. Karbasi and M. Zadimoghaddam	Compression with Graphical Constraints: An Interactive Browser	Proc. International Symposium on Information Theory (ISIT)	(PR)(CO)
C183	2011	B. Haeupler, V. Mirrokni and M. Zadimoghaddam	Online Stochastic Weighted Matching: Improved Approximation Algorithms	Proc. Workshop on Internet & Network Economics	(PR)(CO)
C184	2011	Z. Abel, E. D. Demaine, M. L. Demaine, S. Eisenstat, J. Lynch, T. B. Schardl and I. Shapiro-Ellowitz	Folding Equilateral Plane Graphs	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)(CO)
C185	2011	E. D. Demaine, S. Eisenstat, M. Ishaque and A. Winslow	One-Dimensional Staged Self-Assembly	Proc. International Conference on DNA Computing and Molecular Programming	(PR)(CO)

C186	2011	E. D. Demaine, M. L. Demaine, S. Eisenstat, A. Lubiw and A. Winslow	Algorithms for Solving Rubik's Cubes	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)
C187	2011	E. D. Demaine and S. Eisenstat	Flattening Fixed-Angle Chains Is Strongly NP-Hard	Proc. International Workshop on Algorithms and Data Structures (WADS)	(PR)
C188	2011	P. Christiano, E. D. Demaine and S. Kishore	Lossless Fault-Tolerant Data Structures with Additive Overhead	Proc. International Workshop on Algorithms and Data Structures (WADS)	(PR)(CO)
C189	2011	P. Berman, E. D. Demaine and M. Zadimoghaddam	O(1)-Approximations for Maximum Movement Problems	Proc. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)	(PR)(CO)
C190	2011	G. Aloupis, E. D. Demaine, M. L. Demaine, V. Dujmovic and J. Iacono	Meshes preserving minimum feature size	Proc. Spanish Meeting on Computational Geometry	(CO)
C191	2011	E. D. Demaine and A. Lubiw	A generalization of the source unfolding of convex polyhedra	Proc. Spanish Meeting on Computational Geometry	(CO)
C192	2011	E. D. Demaine, M. Hajiaghayi and K. Kawarabayashi	Contraction Decomposition in H-Minor-Free Graphs and Algorithmic Applications	Proc. Symposium on Theory of Computing (STOC)	(PR)(CO)
C193	2011	E. D. Demaine, M. J. Patitz, R. T. Schweller and S. M. Summers	Self-Assembly of Arbitrary Shapes Using RNAse Enzymes: Meeting the Kolmogorov Bound with Small Scale Factor	Proc. Symposium on Theoretical Aspects of Computer Science (STACS)	(PR)(CO)
C194	2011	E. D. Demaine and A. Schulz	Embedding Stacked Polytopes on a Polynomial-Size Grid	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)
C195	2012	P. Davoodi, M. Smid and F. van Walderveen	Two-Dimensional Range Diameter	Proc. Latin American Symposium on Theoretical Informatics (LATIN)	(PR)(CO)(OA)
C196	2012	L. Arge, M.T. Goodrich and F. van Walderveen	Computing betweenness centrality in external memory	Workshop on Massive Data Algorithmics (MASSIVE)	(CO)(OA)
C197	2012	K. G. Larsen and R. Pagh	I/O-Efficient Data Structures for Colored Range and Prefix Reporting	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C198	2012	T. M. Chan, S. Durocher, K. G. Larsen, J. Morrison and B. T. Wilkinson	Linear-Space Data Structures for Range Mode Query in Arrays	Proc. Symposium on Theoretical Aspects of Computer Science (STACS)	(PR)(CO)(OA)
C199	2012	K. G. Larsen	The Cell Probe Complexity of Dynamic Range Counting	Proc. Symposium on Theory of Computing (STOC)	(PR)(OA)
C200	2012	P. Afshani, L. Arge and K. G. Larsen	Higher-dimensional Orthogonal Range Reporting and Rectangle Stabbing in the Pointer Machine Model	Proc. Symposium on Computational Geometry (SoCG)	(PR)(OA)
C201	2012	K. G. Larsen and H. L. Nguyen	Improved Range Searching Lower Bounds	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)(OA)
C202	2012	K. G. Larsen	Higher Cell Probe Lower Bounds for Evaluating Polynomials	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(OA)
C203	2012	L. Arge, L. Deleuran, T. Mølhave, M. Revsbæk and J. Truelsen	Simplifying Massive Contour Maps	Proc. European Symposium on Algorithms (ESA)	(PR)(OA)
C204	2012	Z. Huang, K. Yi and Q. Zhang,	Randomized Algorithms for Tracking Distributed Count, Frequencies, and Ranks	Proc. Symposium on Principles of Database Systems (PODS)	(PR)(CO)(OA)

C205	2012	D.P. Woodruff and Q. Zhang	Tight Bounds for Distributed Functional Monitoring	Proc. Symposium on Theory of Computing (STOC)	(PR)(CO)(OA)
C206	2012	J. M. Phillips, E. Verbin and Q. Zhang	Lower Bounds for Number-in-Hand Multiparty Communication Complexity, Made Easy	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C207	2012	E. Verbin and Q. Zhang	Rademacher-Sketch: A Dimensionality-Reducing Embedding for Sum-Product Norms, with an Application to Earth-Mover Distance	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(OA)
C208	2012	H.L. Chan, S.H. Chan, T.W. Lam, L.K. Lee, and J. Zhu	Non-clairvoyant weighted flow time scheduling with rejection penalty	Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)	(PR)(CO)(OA)
C209	2012	G. S. Brodal, J. A. S. Nielsen and J. Truelsen	Finger search in the implicit model	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)(OA)
C210	2012	G.S. Brodal and C. Kejlberg-Rasmussen	Cache-Oblivious Implicit Predecessor Dictionaries with the Working-Set Property	Proc. Symposium on Theoretical Aspects of Computer Science (STACS)	(PR)(OA)
C211	2012	X. Sun , C. Wang and W. Yu	The Relationship between Inner Product and Counting Cycles	Proc. Latin American Theoretical Informatics Symposium (LATIN)	(PR)(CO)(OA)
C212	2012	P. Davoodi, R. Raman and S. S. Rao	Succinct Representations of Binary Trees for Range Minimum Queries	Proc. International Computing and Combinatorics Conference (COCOON)	(PR)(CO)(OA)
C213	2012	G.S. Brodal, S. Sioutas, K. Tsakalidis and K. Tsichlas	Fully Persistent B-trees	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C214	2012	G.S. Brodal, G. Lagogiannis and R.E. Tarjan.	Strict Fibonacci Heaps	Proc. Symposium on Theory of Computing (STOC)	(PR)(CO)(OA)
C215	2012	G.S Brodal, P. Davoodi, M. Lewenstein, R. Raman and S. S. Rao	Two Dimensional Range Minimum Queries and Fibonacci Lattices	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)(OA)
C216	2012	D. Ajwani, A. Beckmann, U. Meyer and D. Veith	I/O-efficient approximation of graph diameter by parallel cluster growing - a first experimental study	Proc. Workshop on Parallel Systems and Algorithms (PASA)	(PR)(CO)(OA)
C217	2012	A. Beckmann, J. Fedorowicz, J.Keller and U. Meyer	A structural analysis of the A5/1 state transition graph	Proc. Workshop on Graph Inspection and Traversal Engineering (GRAPHite)	(PR)(CO)(OA)
C218	2012	G. Moruz and A. Negoescu	Outperforming LRU via Competitive Analysis on Parametrized Inputs for Paging	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(OA)
C219	2012	G. Moruz, A. Negoescu, C. Neumann and V. Weichert	Engineering Efficient Paging Algorithms	Proc. Symposium on Experimental Algorithms (SEA)	(PR)(OA)
C220	2012	H. Hassanieh, P. Indyk, D. Katabi and E. Price	Simple and Practical Algorithm for Sparse Fourier Transform	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C221	2012	H. Hassanieh, P. Indyk, D. Katabi and E. Price	Nearly Optimal Sparse Fourier Transform	Proc. Symposium on Theory of Computing (STOC)	(PR)(CO)(OA)
C222	2012	S. Mozes and C. Sommer	Exact Distance Oracles for Planar Graphs	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C223	2012	T. Akiba, C. Sommer and K-i Kawarabayashi	Shortest-Path Queries for Complex Networks: Exploiting Low Tree-width Outside the Core	Proc. International Conference on Extending Database Technology (EDBT)	(PR)(CO)(OA)
C224	2012	S. Kreutzer and S. Tazari	Directed Nowhere Dense Classes of Graphs	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)

C225	2012	V.S. Mirrokni, S. O. Gharan and M. Zadimoghaddam	Simultaneous approximations for adversarial and stochastic online budgeted allocation	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C226	2012	D. M. Kane and J. Nelson	Sparser Johnson-Lindenstrauss Transforms	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C227	2012	C. Tsirogiannis, B. Sandel and D. Cheliotis	Efficient Computation of Popular Phylogenetic Tree Measures	Proc. Workshop on Algorithms in Bioinformatics (WABI)	(PR)(CO)(OA)
C228	2012	L. Arge, H. Haverkort and C. Tsirogiannis	Fast Generation of Multiple Resolution Instances of Raster Data Sets	Proc. International Conference on Advances in Geographic Information Systems (ACM-GIS)	(PR)(CO)(OA)
C229	2012	P. Afshani and N. Zeh	Lower Bounds for Sorted Geometric Queries in the I/O Model	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)(OA)
C230	2012	P. Afshani	Improved pointer machine and I/O lower bounds for simplex range reporting and related problems	Proc. ACM Symposium on Computational Geometry (SoCG)	(PR)(OA)
C231	2012	T. M. Chan, S. Durocher, M. Skala, and B. T. Wilkinson	Linear-Space Data Structures for Range Minority Query in Arrays	Proc. Scandinavian Workshop on Algorithm Theory (SWAT)	(PR)(CO)(OA)
C232	2012	H. Jowhari	Efficient Communication Protocols for Deciding Edit Distance	Proc. European Symposium on Algorithms (ESA)	(PR)(OA)
C233	2012	L. K. Lee, M. Lewenstein and Q. Zhang.	Parikh matching in the streaming model	Proc. International Symposium on String Processing and Information Retrieval (SPIRE)	(PR)(CO)(OA)
C234	2012	D. Belazzougui and R. Venturini	Compressed String Dictionary Look-up with Edit Distance One	Proc. Symposium on Combinatorial Pattern (CPM)	(PR)(CO)(OA)
C235	2012	B. Ammitzbøll Jurik and J.A.S. Nielsen	Audio Quality Assurance: An Application of Cross Correlation	Proc. iPRES Conference	(PR)(CO)(OA)
C236	2012	N. Sitchinava and N. Zeh	A parallel buffer tree	Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)	(PR)(CO)(OA)
C237	2012	D. Ajwani, U. Meyer and D. Veith	I/O-efficient Hierarchical Diameter Approximation	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)(OA)
C238	2012	D. Kane, K. Mehlhorn, T. Sauerwald and H. Sun	Counting Arbitrary Subgraphs in Data Streams	Proc. International Colloquium on Automata, Languages, and Programming (ICALP)	(PR)(
C239	2012	M. Wibral, P. Wollstadt, U. Meyer, N. Pampu, V.Priesemann and R. Vicente	Revisiting Wiener's principle of causality — interaction-delay reconstruction using transfer entropy and multivariate analysis on delay-weighted graphs	Proc. International Confonference in Medicine & Biology Society (EMBC)	(PR)(CO)(OA)
C240	2012	P. Indyk, R. Levi and R. Rubinfeld	Approximating and Testing k-Histogram Distributions in Sub-linear Time	Proc. Symposium on Principles of Database Systems (PODS)	(PR)(CO)(OA)
C241	2012	J. Wang, H. Hassanieh, D. Katabi and P. Indyk	Efficient and Reliable Low-Power Backscatter Networks	Proc. International Conference on Mobile Computing and Networking (SIGCOMM)	(PR)(CO)(OA)
C242	2012	H. Hassanieh, F. Adib, D. Katabi and P. Indyk	Faster GPS Via the Sparse Fourier Transform	Proc. International Conference on Mobile Computing and Networking (MOBICOM)	(PR)(CO)(OA)

C243	2012	E. Price and D. Woodruff	Applications of the Shannon-Hartley Theorem to Data Streams and Sparse Recovery	Proc. International Symposium on Information Theory (ISIT)	(PR)(CO)(OA)
C244	2012	L. Hamilton, D. Parker, C. Yu and P. Indyk	Focal Plane Array Folding for Efficient Information Extraction and Tracking	Proc. Applied Imagery Patterns Recognition Workshop (AIPR)	(PR)(CO)(OA)
C245	2013	E. Price and D. Woodruff	Lower Bounds for Adaptive Sparse Recovery	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C246	2013	A. Andoni, H. Hassanieh, P. Indyk and D. Katabi	Shift Finding in Sub-linear Time	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C247	2012	E.D. Demaine, M.L. Demaine, Y. N. Minsky, J.S.B. Mitchell, R.L. Rivest and M. Patrascu	Picture-Hanging Puzzles	Proc. International Conference on Fun with Algorithms	(CO)(OA)
C248	2012	E. D. Demaine, M.L. Demaine, J-i. Itoh, A. Lubiw, C. Nara and J. O'Rourke	Refold Rigidity of Convex Polyhedra	Proc. European Workshop on Computational Geometry	(CO)(OA)
C249	2012	S. Lim, C. Sommer, E. Nikolova and D. Rus	Practical Route Planning Under Delay Uncertainty: Stochastic Shortest Path Queries	Proc. Robotics: Science and Systems VIII	(PR)(CO)(OA)
C250	2012	C. Ratti and C. Sommer	Approximating Shortest Paths in Spatial Social Networks	Proc. International Conference on Social Computing	(PR)(CO)(OA)
C251	2012	M. Zadimoghaddam and A. Roth	Efficiently Learning from Revealed Preference	Proc. International Workshop on Internet and Network Economics	(PR)(CO)(OA)
C252	2012	C. Guo, Y. Ma, B. Yang, C. S. Jensen and M. Kaul	EcoMark: Evaluating Models of Vehicular Environmental Impact	Proc. International Conference on Advances in Geographic Information Systems (ACM-GIS)	(PR)(CO)(OA)
C253	2012	X. Li, P. Karras, L. Shi, K.-L. Tan and C. S. Jensen	Cooperative Scalable Moving Continuous Query Processing	Proc. International Conference on Mobile Data Management (MDM)	(PR)(CO)(OA)
C254	2012	D. Šidlauskas, C. S. Jensenand S. Šaltenis	A Comparison of the Use of Virtual Versus Physical Snapshots for Supporting Update-Intensive Workloads	Proc. International Workshop on Data Management on New Hardware (DaMoN)	(PR)(CO)(OA)
C255	2012	J. Rishede, M. L. Yiu and C. S. Jensen	Effective Caching of Shortest Paths for Location-Based Services	Proc. International Conference on the Management of Data (SIGMOD)	(PR)(CO)(OA)
C256	2012	D. Šidlauskas, S. Šaltenis and C. S. Jensen	Parallel Main-Memory Indexing for Moving-Object Query and Update Workloads	Proc. International Conference on the Management of Data (SIGMOD)	(PR)(CO)(OA)
C257	2012	H. Lu, X. Cao and C. S. Jensen	A Foundation for Efficient Indoor Distance-Aware Query Processing	Proc. International Conference on Data Engineering (ICDE)	(PR)(CO)(OA)
C258	2012	H. Lu and C. S. Jensen	Upgrading Uncompetitive Products Economically	Proc. International Conference on Data Engineering (ICDE)	(PR)(CO)(OA)
C259	2012	X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu and M. L. Yiu	Spatial Keyword Querying (invited paper)	Proc. International Conference on Conceptual Modeling (ER)	(CO) (OA)
C260	2013	M. Olsen and M. Revsbæk	Alliances and Bisection Width for Planar Graphs	Proc. International Workshop on Algorithms and Computation (WALCOM)	(PR)(OA)
C261	2013	K.G. Larsen and F. van Walderveen	Near-Optimal Range Reporting Structures for Categorical Data	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(OA)

C262	2013	K. Bringmann and K.G. Larsen	Succinct Sampling from Discrete Distributions	Proc. ACM Symposium on Theory of Computing (STOC)	(PR)(CO)(OA)
C263	2013	Chr. Tsirogiannis and Con. Tsirogiannis	Uncovering the Missing Routes : An Algorithmic Study on the Illicit Antiquities Trade Network	Proc. Conference on Computer Applications and Quantitative Methods in Archaeology (CAA)	(PR)(CO)(OA)
C264	2013	C. Tsirogiannis and B. Sandel	Computing the Skewness of the Phylogenetic Mean Pairwise Distance in Linear Time	Proc. Workshop on Algorithms in Bioinformatics (WABI)	(PR)(CO)(OA)
C265	2013	L. Arge, G.S. Brodal, J. Truelsen and C. Tsirogiannis	An Optimal and Practical Cache-Oblivious Algorithm for Computing Multiresolution Rasters	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)(OA)
C266	2013	L. Arge, M. de Berg and C. Tsirogiannis	Algorithms for Computing Prominence on Grid Terrains	Proc. International Conference on Advances in Geographic Information System (ACM-GIS)	(PR)(CO)(OA)
C267	2013	L. Arge and M. Thorup	RAM-Efficient External Memory Sorting	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR) (CO)(OA)
C268	2013	L. Arge, M.T. Goodrich and F. van Walderveen	Computing betweenness centrality in external memory	Proc. IEEE International Symposium on Big Data	(PR)(CO)(OA)
C269	2013	L. Arge, J. Fischer, P. Sanders and N. Sitchinava	On (Dynamic) Range Maximum Queries in External Memory	Proc. Workshop on Algorithms and Data Structures (WADS)	(PR)(CO)(OA)
C270	2013	L. Arge, F. van Walderveen and N. Zeh	Multiway simple cycle separators and I/O-efficient algorithms for planar graphs	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C271	2013	G.S. Brodal, A. Brodnik and P. Davoodi	The Encoding Complexity of Two Dimensional Range Minimum Data Structures	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)(OA)
C272	2013	A. Sand, G.S. Brodal, R. Fagerberg, C.N.S. Pedersen and T. Mailund	A practical O(n log n) time algorithm for computing the triplet distance on binary trees	Proc. Asia Pacific Bioinformatics Conference (APBC)	(PR)(CO)(OA)
C273	2013	G.S. Brodal, R. Fagerberg, C.N.S. Pedersen, T. Mailund and A. Sand	Efficient Algorithms for Computing the Triplet and Quartet Distance Between Trees of Arbitrary Degree	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C274	2013	S. Pettie and H.-H. Su	Fast Distributed Coloring Algorithms for Triangle-Free Graphs	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)(OA)
C275	2013	C. Kejlberg-Rasmussen, Y. Tao, J. Yoon, K. Tsichlas and K. Tsakalidis	I/O-Efficient Planar Range Skyline and Attrition Priority Queues	Proc. Symposium on Principles of Database Systems (PODS)	(PR)(CO)(OA)
C276	2013	Z. Wei and K. Yi	The Space Complexity of 2-Dimensional Approximate Range Counting	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C277	2013	E. Verbin and W. Yu	Data Structure Lower Bounds on Random Access to Grammar-Compressed Strings	Proc. Symposium on Combinatorial Pattern Matching (CPM)	(PR)(OA)
C278	2013	D. Belazzougui and R. Venturini	Compressed Static Functions with Applications	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C279	2013	T.M. Chan and B.T. Wilkinson	Adaptive and Approximate Orthogonal Range Counting	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)

C280	2013	S. Alamdari, P. Angelini, T.M. Chan, G. Di Battista, F. Frati, A. Lubiw, M. Patrignani, V. Roselli, S. Singla and B.T. Wilkinson	Morphing Planar Graph Drawings with a Polynomial Number of Steps	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C281	2013	T. Jurkiewicz and K. Mehlhorn	The Cost of Address Translation	Proc. Workhop on Algorithm Engineering and Experiments (ALENEX)	(PR)(OA)
C282	2013	G. Moruz and A. Negoescu	Improved space bounds for strongly competitive randomized paging algorithms	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(OA)
C283	2013	A. Beckmann, U. Meyer, and D. Veith	An Implementation of I/O-Efficient Dynamic Breadth-First Search Using Level-Aligned Hierarchical Clustering	Proc. European Symposium on Algorithms (ESA)	(PR) (CO)(OA)
C284	2013	L. Radaelli and C.S. Jensen	Towards Fully Organic Indoor Positioning	Proc. International Workshop on Indoor Spatial Awareness (ISA)	(PR)(CO)(OA)
C285	2013	X. Li, V. Ceikute, C.S. Jensen and K.-L. Tan	Trajectory Based Optimal Segment Computation in Road Network Databases	Proc. International Conference on Advances in Geographic Information Systems (ACM-GIS)	(PR)(CO)(OA)
C286	2013	M.B. Kjærgaard, M.V. Krarup, A. Stisen, T.S. Prentow, H. Blunck, K. Grønbæk and C.S. Jensen	Indoor Positioning using Wi-Fi—How Well Is the Problem Understood?	Proc. International Conference on Indoor Positioning and Indoor Navigation (IPIN)	(PR)(CO)(OA)
C287	2013	V. Ceikute and C.S. Jensen	Routing Service Quality—Local Driver Behavior Versus Routing Services	Proc. International Conference on Mobile Data Management (MDM)	(PR)(CO)(OA)
C288	2013	M. Kaul, B. Yang and C.S. Jensen	Building Accurate 3D Spatial Networks to Enable Next Generation Intelligent Transportation Systems	Proc. International Conference on Mobile Data Management (MDM)	(PR)(CO)(OA)
C289	2013	L. Radaelli, D. Sabonis, H. Lu and C.S. Jensen	Identifying Typical Movements Among Indoor Objects— Concepts and Empirical Study	Proc. International Conference on Mobile Data Management (MDM)	(PR)(CO)(OA)
C290	2013	A. Baniukevic, C.S. Jensen and H. Lu	Hybrid Indoor Positioning With Wi-Fi and Bluetooth: Architecture and Performance	Proc. International Conference on Mobile Data Management (MDM)	(PR)(CO)(OA)
C291	2013	O. Andersen, C.S. Jensen, K. Torp and B. Yang	EcoTour: Reducing the Environmental Footprint of Vehicles Using Eco-Routes	Proc. International Conference on Mobile Data Management (MDM)	(PR)(CO)(OA)
C292	2013	L.R.A. Derczynski, B. Yang and C.S. Jensen	Towards Context-Aware Search and Analysis on Social Media Data	Proc. International Conference on Extending Database Technology (EDBT)	(PR)(CO)(OA)
C293	2013	B. Yang, N. Fantini and C.S. Jensen	iPark: Identifying Parking Spaces from Trajectories	Proc. International Conference on Extending Database Technology (EDBT)	(PR)(CO)(OA)
C294	2013	E. Grant, C. Hegde and P. Indyk	Nearly Optimal Linear Embeddings into Very Low Dimensions	Proc. Global Conference on Signal and Information Processing (GlobalSIP)	(PR)(CO)(OA)
C295	2013	S. Abbar, S. Amer-Yahia, P. Indyk, S. Mahabadi and K.R. Varadarajan	Diverse near neighbor problem	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)(OA)

C296	2013	E. Grant and P. Indyk	Compressive sensing using locality-preserving matrices	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)(OA)
C297	2013	L. Schmidt, C. Hegde and P. Indyk	The Constrained Earth Movers Distance Model, with Applications to Compressive Sensing	Proc. International Conference on Sampling Theory and Applications (SampTA)	(PR)(CO)(OA)
C298	2013	S. Abbar, S. Amer-Yahia, P. Indyk and S. Mahabadi	Real-time recommendation of diverse related articles	Proc. International conference on World Wide Web (WWW)	(PR)(CO)(OA)
C299	2013	P. Indyk and I. Razenshteyn	On Model-Based RIP-1 Matrices	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)(OA)
C300	2013	B. Ghazi, H. Hassanieh, P. Indyk, D. Katabi, E. Price and L. Shi	Sample-optimal average-case sparse fourier transform in two dimensions	Proc. Allerton Conference	(PR)(CO)(OA)
C301	2013	E. Price and D. Woodruff	Lower Bounds for Adaptive Sparse Recovery	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C302	2013	A. Andoni, H. Hassanieh, P. Indyk and D. Katabi	Shift Finding in Sub-linear Time	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C303	2013	S. Har-Peled, P. Indyk and A. Sidiropoulos	Euclidean spanners in high dimensions	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C304	2013	E.D. Demaine, P. Panchekha, D. Wilson and E.Z. Yang	Blame Trees	Proc. Algorithms and Data Structures Symposium (WADS)	(PR)(CO)(OA)
C305	2013	E.D. Demaine, M.J. Patitz, T.A. Rogers, R.T. Schweller, S.M. Summers and D. Woods	The two-handed tile assembly model is not intrinsically universal	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)(OA)
C306	2013	E.D. Demaine, J. Iacono, S. Langerman and O. Ozkan	Combining Binary Search Trees	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)(OA)
C307	2013	S. Cannon, E.D. Demaine, M.L. Demaine, S. Eisenstat, M.J. Patitz, R. Schweller, S.M. Summers and A. Winslow	Two Hands Are Better Than One (up to constant factors): Self-Assembly In The 2HAM vs. aTAM	Proc. International Symposium on Theoretical Aspects of Computer Science (STACS)	(PR)(CO)(OA)
C308	2013	Z. Abel, E.D. Demaine, M.L. Demaine, S. Eisenstat, A. Lubiw, A. Schulz, D. Souvaine, G. Viglietta and A. Winslow	Algorithms for Designing Pop-Up Cards	Proc. International Symposium on Theoretical Aspects of Computer Science (STACS)	(PR)(CO)(OA)
C309	2013	E.D. Demaine and M. Zadimoghaddam	Learning Disjunctions: Near-Optimal Trade-off between Mistakes and "I Don't Know's''	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(OA)
C310	2013	A. Karbasi and M. Zadimoghaddam	Constrained Binary Identification Problem	Proc. International Symposium on Theoretical Aspects of Computer Science (STACS)	(PR)(CO)(OA)
C311	2013	M. Bateni, N. Haghpanah, B. Sivan and M. Zadimoghaddam	Revenue Maximization with Nonexcludable Goods	Proc. International Conference on Web and Internet Economics (WINE)	(PR)(CO)(OA)
C312	2013	N. Korula, V.S. Mirrokni and M. Zadimoghaddam	Bicriteria Online Matching: Maximizing Weight and Cardinality	Proc. International Conference on Web and Internet Economics (WINE)	(PR)(CO)(OA)

C313	2013	G.S. Brodal	A Survey on Priority Queues	Proc. Conference on Space Efficient Data Structures, Streams and Algorithms	(OA)
C314	2013	P. Afshani, M. Agrawal, B. Doerr, C. Doerr, K.G. Larsen, K. Mehlhorn	The Query Complexity of Finding a Hidden Permutation	Proc. Conference on Space Efficient Data Structures, Streams and Algorithms	(CO)(OA)
C315	2013	E.D. Demaine, M.L. Demaine, S. Eisenstat, T.D. Morgan and R. Uehara	Variations on Instant Insanity	Proc. Conference on Space Efficient Data Structures, Streams and Algorithms	(CO)(OA)
C316	2013	N. Sundaram, A. Turmukhametova, N. Satish, T. Mostak, P. Indyk, S. Madden and P. Dubey	Streaming Similarity Search over one Billion Tweets using Parallel Locality-Sensitive Hashing	Proc. International Conference on Very Large Data Bases (VLDB)	(PR)(CO)(OA)
C317	2014	A. Skovsgaard, D. Sidlauskas and C.S. Jensen	Scalable Top-k Spatio-Temporal Term Querying	Proc. International Conference on Data Engineering (ICDE)	(PR)(OA)
C318	2014	A. Skovsgaard, D. Sidlauskas and C.S. Jensen	A Clustering Approach to the Discovery of Points of Interest from Geo-Tagged Microblog Posts	Proc. International Conference on Mobile Data Management (MDM)	(PR)(OA)
C319	2014	S. Alstrup, E. B. Halvorsen and K.G. Larsen,	Near-Optimal Labeling Schemes for Nearest Common Ancestors	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C320	2014	K.G. Larsen, I.J. Munro, J.S. Nielsen and S.V. Thankachan	On Hardness of Several String Indexing Problems	Proc. Symposium on Combinatorial Pattern Matching (CPM)	(PR)(CO)(OA)
C321	2014	G. S. Brodal and K. G. Larsen	Optimal Planar Orthogonal Skyline Counting Queries	Proc. Scandinavian Workshop on Algorithm Theory (SWAT)	(PR)(OA)
C322	2014	P. Afshani	Fast Computation of Output-Sensitive Maxima in a Word RAM	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(OA)
C323	2014	P. Afshani and K. Tsakalidis	Optimal Deterministic Shallow Cuttings for 3D Dominance Ranges	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C324	2014	P. Afshani, C. Sheng, Y. Tao and B.T. Wilkinson	Concurrent Range Reporting in Two-Dimensional Space	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C325	2014	P. Afshani, T.M. Chan and K. Tsakalidis	Deterministic Rectangle Enclosure and Offline Dominance Reporting on the RAM	Proc. International Colloquium on Automata, Languages, and Programming (ICALP)	(PR)(CO)(OA)
C326	2014	P. Afshani and N. Sitchinava	I/O-efficient Range Minima Queries	Proc. Workshops on Algorithm Theory (SWAT)	(PR)(CO)(OA)
C327	2014	B. T. Wilkinson	Amortized bounds for dynamic orthogonal range reporting	Proc. European Symposium on Algorithms (ESA)	(PR)(OA)
C328	2014	Z. Huang and K. Yi	The Communication Complexity of Distributed epsilon-Approximations	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)(OA)
C329	2014	K-M. Chung, S. Pettie and H.-H. Su	Distributed Algorithms for the Lovász Local Lemma and Graph Coloring	Proc. Symposium on Principles of Distributed Computing (PODC)	(PR)(CO)(OA)
C330	2014	S. Gilbert, V. King, S. Pettie, E. Porat, J. Saia and M. Young	(Near) Optimal Resource-Competitive Broadcast with Jamming	Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA)	(PR)(CO)(OA)
C331	2014	H.-H. Su	Brief Announcement: A Distributed Minimum Cut Approximation Scheme	Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA)	(PR)(OA)
C332	2014	D.Belazzougui, G. S. Brodal and J. S. Nielsen	Expected Linear Time Sorting for Word Size Ω(log n loglogn)	Proc. Workshops on Algorithm Theory (SWAT)	(PR)(CO)(OA)
C333	2014	M.K. Holt, J. Johansen and G.S. Brodal	On the Scalability of Computing Triplet and Quartet Distances	Proc.Workshop on Algorithm Engineering and Experiments (ALENEX)	(PR)(OA)

C334	2014	Z. Wei and K. Yi	Equivalence between Priority Queues and Sorting in External Memory	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)(OA)
C335	2014	L Radaelli, Y. Moses and C.S. Jensen	Using Cameras to Improve Wi-Fi Based Indoor Positioning	Proc. International Symposium on Web and Wireless Geographical Information Systems	(PR)(CO)(OA)
C336	2014	B. Yang, C. Guo, C. S. Jensen, M. Kaul and S. Shang	Multi-Cost Optimal Route Planning under Time-Varying Uncertainty	Proc. International Conference on Data Engineering (ICDE)	(PR)(CO)(OA)
C337	2014	C. Silvestri, F. Lettich, S. Orlando and C.S. Jensen	GPU-based Computing of Repeated Range Queries over Moving Objects	Proc. Euromicro International Conference on Parallel, Distributed, and Network-Based Processing	(PR)(CO)(OA)
C338	2014	P. Indyk, M. Kapralov and E. Price	(Nearly) Sample-Optimal Sparse Fourier Transform	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C339	2014	A. Andoni, P. Indyk, H.L. Nguyen and I. Razenshteyn	Beyond Locality-Sensitive Hashing	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C340	2014	C. Hegde, P. Indyk and L. Schmidt	Approximation-Tolerant Model-Based Compressive Sensing	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C341	2014	Z. Abel, E. D. Demaine, M. L. Demaine, D. Eppstein, A. Lubiw and R. Uehara	Flat Foldings of Plane Graphs with Prescribed Angles and Edge Lengths	Proc. International Symposium on Graph Drawing (GD)	(PR)(CO)(OA)
C342	2014	E. D. Demaine, M. Hajiaghayi, H. Mahini, D. L. Malec, S. Raghavan, A. Sawant and M. Zadimoghaddam	How to Influence People with Partial Incentives	Proc. International World Wide Web Conference	(PR)(CO)(OA)
C343	2014	C.Tsirogiannis, B.Sandel and A. Kalvisa.	New algorithms for computing phylogenetic biodiversity.	Proc. Workshop on Algorithms in Bioinformatics (WABI)	(PR)(CO)(OA)
C344	2014	D. Sidlauskas and C. S. Jensen	Spatial Joins in Main Memory: Implementation Matters!	Proc. International Conference on Very Large Data Bases (VLDB)	(PR)(CO)(OA)
C345	2014	K. S. Bøgh, S. Chester, D. Sidlauskas and I. Assent	Hashcube: A Data Structure for Space- and Query-Efficient Skycube Compression	Proc. International Conference on Conference on Information and Knowledge Management (CIKM)	(PR)(CO)(OA)
C346	2014	A. Skovsgaard and C.S. Jensen	Top-k point of interest retrieval using standard indexes.	Proc. International Conference on Advances in Geographic Information Systems (ACM-GIS)	(PR)(OA)
C347	2014	D. Chen, C. Konrad, K. Yi, W. Yu and Q. Zhang	Robust Set Reconciliation	Proc. International Conference on Management of Data (SIGMOD)	(PR)(CO)(OA)
C348	2014	A. Grønlund and S. Pettie	Threesomes, Degenerates, and Love Triangles	IEEE Symposium on Foundations of Computer Science (FOCS 2014)	(PR)(OA)
C349	2014	D. Nanongkai and H.-H. Su	Almost-Tight Distributed Minimum Cut Algorithms	Proc. International Symposium Distributed Computing (DISC)	(PR)(CO)(OA)
C350	2014	J. I. Munro, G. Navarro, J. S. Nielsen, R. Shah and S. V. Thankachan	Top-k Term-Proximity in Succinct Space	Proc. International Symposium on Algorithms and Computation (ISAAC)	(PR)(CO)(OA)
C351	2014	P. Indyk, S.Mahabadi, M. Mahdian and V. S. Mirrokni	Composable Core-sets for Diversity and Coverage Maximization	Proc. Symposium on Principles of Database Systems (PODS)	(PR)(CO)(OA)

C352	2014	E.D. Demaine, P. Indyk, S. Mahabadi and A. Vakilian	On Streaming and Communication Complexity of the Set Cover Problem	Proc. International Symposium Distributed Computing (DISC)	(PR)(CO)(OA)
C353	2014	P. Indyk and M. Kapralov	Sample-Optimal Fourier Sampling in Any Constant Dimension	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)(OA)
C354	2014	L. Schmidt, C. Hegde, P. Indyk, J. Kane, L. Lu and D. Hohl	Automatic Fault Localization Using the Generalized Earth Mover's Distance	Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)	(PR)(CO)(OA)
C355	2014	C. Hegde, P. Indyk and L. Schmidt	A Fast Approximation Algorithm for Tree-Sparse Recovery	Proc. International Symposium on Information Theory (ISIT)	(PR)(CO)(OA)
C356	2014	C. Hegde, P. Indyk and L. Schmidt	Nearly Linear-Time Model-Based Compressive Sensing	Proc. International Colloquium on Automata, Languages, and Programming (ICALP)	(PR)(CO)(OA)
C357	2014	A. Backurs and P. Indyk	Better embeddings for planar Earth-Mover Distance over sparse sets	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)(OA)
C358	2014	H. Hassanieh, L. Shi, O. Abari, E. Hamed and D. Katabi	GHz-wide sensing and decoding using the sparse Fourier transform	Proc. INFOCOM	(PR)(CO)
C359	2014	E. D. Demaine, Y. Huang, C.-S. Liao and K. Sadakane	Canadians Should Travel Randomly	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)(OA)
C360	2014	E. D. Demaine, M. L. Demaine, S. P. Fekete, M. J. Patitz, R. T. Schweller, A. Winslow and D. Woods	One Tile to Rule Them All: Simulating Any Tile Assembly System with a Single Universal Tile	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(CO)(OA)
C361	2014	G. Aloupis, E. D. Demaine, A. Guo and G. Viglietta	Classic Nintendo Games are (NP-)Hard	Proc. International Conference on Fun with Algorithms	(CO)(OA)
C362	2014	E. D. Demaine, F. Ma and E. Waingarten	Playing Dominoes is Hard, Except by Yourself	Proc. International Conference on Fun with Algorithms	(CO)(OA)
C363	2014	K. Yamanaka, E. D. Demaine, T. Ito, J. Kawahara, M. Kiyomi, Y. Okamoto, T. Saitoh, A. Suzuki, K. Uchizawa and T. Uno	Swapping Labeled Tokens on Graphs	Proc. International Conference on Fun with Algorithms	(CO)(OA)
C364	2014	E. D. Demaine and M. L. Demaine	Fun with Fonts: Algorithmic Typography	Proc. International Conference on Fun with Algorithms	(CO)(OA)
C365	2014	Z. Abel, E. D. Demaine, M. L. Demaine, J.-I. Itoh, A. Lubiw, C. Nara and J. O'Rourke	Continuously Flattening Polyhedra Using Straight Skeletons	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)(OA)
C366	2014	B. An, S. Miyashita, M. T. Tolley, D. M. Aukes, L. Meeker, E. D. Demaine, M. L. Demaine, R. J. Wood and D. Rus	An End-To-End Approach to Making Self-Folded 3D Surface Shapes by Uniform Heating	Proc. International Conference on Robotics and Automation	(PR)(CO)(OA)
C367	2014	A. Becker, E. D. Demaine, S. Fekete and J. McLurkin	Particle Computation: Designing Worlds to Control Robot Swarms with Only Global Signals	Proc. International Conference on Robotics and Automation	(PR)(CO)(OA)
C368	2014	Y. Bachrach, O. Lev, S. Lovett, J. S. Rosenschein and M. Zadimoghaddam	Cooperative weakest link games	Proc. International Conference on Autonomous Agents and Multiagent Systems	(PR)(CO)(OA)
C369	2014	A. Termehchy, A. Vakilian, Y. Chodpathumwan and M. Winslett	Which Concepts are Worth Extracting?	Proc. International Conference on the Management of Data (SIGMOD)	(PR)(CO)(OA)

C370	2014	A. Ene and A. Vakilian	Improved Approximation Algorithms for Degree-bounded Network Design Problems with Node Connectivity Requirements	Proc. Symposium on Theory of Computing (STOC)	(PR)(CO)(OA)
C371	2014	L. Arge, J. Truelsen and J. Yang	Simplifying massive planar subdivisions	Proc. Workshop on Algorithm Engineering and Experiments (ALENEX)	(PR)(OA)
C372	2015	Z. Huang, B. Radunovic, M. Vojnovic and Q. Zhang	Communication Complexity of Approximate Maximum Matching in Distributed Graph	Proc. Symposium on Theoretical Aspects of Computer Science (STACS)	(PR)(CO)(OA)
C373	2015	G. S. Brodal, J. S. Nielsen and J. Truelsen	Strictly Implicit Priority Queues: On the Number of Moves and Worst-Case Time	Proc. Workshop on Algorithms and Data Structures (WADS)	(PR)(OA)
C374	2015	M. Elkin and S. Pettie	A Linear-Size Logarithmic Stretch Path-Reporting Distance Oracle for General Graphs	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C375	2015	S. Pettie	Sharp Bounds on Formation-free Sequences	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(OA)
C376	2015	M. Elkin, S. Pettie and H.-H. Su	(2Δ-1)-Edge Coloring is Much Easier than Maximal Matching in the Distributed Setting	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C377	2015	T. Kopelowitz, S. Pettie and E. Porat	Dynamic Set Intersection	Proc. Workshop on Algorithms and Data Structures (WADS)	(PR)(CO)(OA)
C378	2015	P. K. Agarwal, T. Mølhave, M. Revsbæk, I. Safa, Y. Wang and J. Yang.	Maintaining Contour Trees of Dynamic Terrains	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)(OA)
C379	2015	S. Chester, D. Sidlauskas, I. Assent and K. S. Bøgh	Scalable parallelization of skyline computation for multi-core processors	Proc. IEEE Conference on Data Engineering (ICDE)	(PR)(CO)(OA)
C380	2015	W. Son and P. Afshani	Streaming Algorithms for Smallest Intersecting Ball of Disjoint Balls	Proc. Conference on Theory and Applications of Models of Computation (TAMC)	(PR)(OA)
C381	2015	M. Goswami, A. Grønlund, K. G. Larsen and R. Pagh	Approximate Range Emptiness in Constant Time and Optimal Space	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C382	2015	K. G. Larsen, J. Nelson and H. L. Nguyen	Time Lower Bounds for Nonadaptive Turnstile Streaming Algorithms	Proc. Symposium on Theory of Computing (STOC)	(PR)(CO)(OA)
C383	2015	D. Ajwani, U.Meyer and D. Veith	An I/O-efficient Distance Oracle for Evolving Real-World Graphs	Proc. Workshop on Algorithm Engineering and Experiments (ALENEX)	(PR)(CO)(OA)
C384	2015	A. Andoni and I. Razenshteyn	Optimal Data-Dependent Hashing for Approximate Near Neighbors	Proc. Symposium on Theory of Computing (STOC)	(PR)(CO)(OA)
C385	2015	Z. Allen-Zhu, R. Gelashvili and I. Razenshteyn	Restricted Isometry Property for General p-Norms	Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)(OA)
C386	2015	S. Lattanzi, S. Leonardi, V. Mirrokni and I. Razenshteyn	Robust Hierarchical k-Center Clustering	Proc. Innovations in Theoretical Computer Science (ITCS)	(PR)(CO)(OA)
C387	2015	A. Andoni, R. Krauthgamer and I. Razenshteyn	Sketching and Embedding are Equivalent for Norms.	Proc. Symposium on the Theory of Computing (STOC)	(PR)(CO)(OA)
C388	2015	C. Hegde, P. Indyk and L. Schmidt	A Nearly-Linear Time Framework for Graph-Structured Sparsity	Proc. International Conference on Machine Learning (ICML)	(PR)(CO)(OA)

C389	2015	L. Schmidt, C. Hegde, P. Indyk, L. Lu, X. Chi and D. Hohl	Seismic feature extraction using steiner tree methods	Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)	(PR)(CO)(OA)
C390	2015	A. Andoni, P. Indyk, T. Laarhoven, I. P. Razenshteyn and L. Schmidt	Practical and Optimal LSH for Angular Distance	Proc. Advances in Neural Information Processing Systems (NIPS)	(PR)(CO)(OA)
C391	2015	A. Kim, E. Blais, A. G. Parameswaran, P. Indyk, S. Madden and R. Rubinfeld	Rapid Sampling for Visualizations with Ordering Guarantees	Proc. International Conference on Very Large Data Bases (VLDB)	(PR)(CO)(OA)
C392	2015	J. Acharya, I. Diakonikolas, C. Hegde, J. Z. Li and L. Schmidt	Fast and Near-Optimal Algorithms for Approximating Distributions by Histograms	Proc. Symposium on Principles of Database Systems (PODS)	(PR)(CO)(OA)
C393	2015	A. Backurs and P. Indyk	Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)	Proc. Symposium on the Theory of Computing (STOC)	(PR)(CO)(OA)
C394	2015	A. Kovacs, U. Meyer, and C. Ventre	Mechanisms with monitoring for truthful RAM allocation	Proc. Conference on Web and Internet Economics (WINE)	(PR)(CO)(OA)
C395	2015	P. Chalermsook, M. Goswami, L. Kozma, K. Mehlhorn and T. Saranurak	Self-Adjusting Binary Search Trees: What Makes Them Tick?	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)(OA)
C396	2015	P. Chalermsook, M. Goswami, L. Kozma, K. Mehlhorn and T. Saranurak	Pattern-Avoiding Access in Binary Search Trees	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)(OA)
C397	2015	P. Chalermsook, M. Goswami, L. Kozma, K. Mehlhorn and T. Saranurak	Greedy Is an Almost Optimal Deque	Proc. Workshop on Algorithms and Data Structures (WADS)	(PR)(CO)(OA)
C398	2015	L. Arge, M. Rav, S. Raza and M. Revsbæk	I/O-Efficient Event Based Depression Flood Risk	Proc. Workshop on Massive Data Algorithmics (MASSIVE)	(OA)
C399	2015	M. de Berg, C. Tsirogiannis and B.T. Wilkinson	Fast Computation of Categorical Richness on Raster Data Sets and Related Problems	Proc. International Conference on Advances in Geographic Information Systems (ACM-GIS)	(PR)(CO)(OA)
C400	2015	M. de Berg, C. Tsirogiannis and B.T. Wilkinson	Fast Computation of Categorical Richness on Raster Data Sets and Related Problems	Proc. Workshop on Massive Data Algorithmics (MASSIVE)	(CO)(OA)
C401	2015	C. Alexander, L. Arge, P. K. Bøcher, M. Revsbæk, B. Sandel, J-C. Svenning, C. Tsirogiannis and J. Yang	Computing River Floods Using Massive Terrain Data	Proc. Workshop on Massive Data Algorithmics (MASSIVE)	(OA)
C402	2015	P. Afshani and N. Sitchinava	Sorting and Permuting without Bank Conflicts on GPUs	Proc. European Symposium on Algorithms (ESA)	(PR)(CO)(OA)
C403	2015	A. Abboud, A. Backurs and V. A. Williams	If the Current Clique Algorithms are Optimal, So is Valiant's Parser	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)(OA)
C404	2015	R. Clifford, A. Grønlund and K. G. Larsen	New Unconditional Hardness Results for Dynamic and Online Problems	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)(OA)
C405	2015	E. D. Demaine, S. Fekete, C. Scheffer and A. Schmidt	New Geometric Algorithms for Fully Connected Staged Self-Assembly	Proc. International Conference on DNA Computing and Molecular Programming	(PR)(CO)(OA)

C406	2015	E. D. Demaine, T. Kaler, Q. Liu, A. Sidford and A. Yedidia	Polylogarithmic Fully Retroactive Priority Queues via Hierarchical Checkpointing	Proc. Workshop on Algorithms and Data Structures (WADS)	(PR)(CO)(OA)
C407	2015	E. D. Demaine, V. Gopal and W. Hasenplaugh	Cache-Oblivious Iterated Predecessor Queries via Range Coalescing	Proc. Workshop on Algorithms and Data Structures (WADS)	(PR)(CO)(OA)
C408	2015	A. T. Becker, E. D. Demaine, S. P. Fekete, H. M. Shad and R. Morris-Wright	Tilt: The Video -- Designing Worlds to Control Robot Swarms with Only Global Signals	Multimedia Exposition in Proc. Symposium on Computational Geometry (SoCG)	(PR)(CO)(OA)
C409	2015	H. M. Shad, R. Morris-Wright, E. D. Demaine, S. P. Fekete and A. Becker	Particle computation: Device fan-out and binary memory	Proc. IEEE International Conference on Robotics and Automation (ICRA)	(PR)(CO)(OA)
C410	2015	E. D. Demaine, S. Fekete and A. Schmidt	New Geometric Algorithms for Staged Self-Assembly	Proc. European Workshop on Computational Geometry (EuroCG)	(CO)(OA)
C411	2015	E. D. Demaine, D. Eppstein, A. Hesterberg, H. Ito, A. Lubiw, R. Uehara and Y. Uno	Folding a Paper Strip to Minimize Thickness	Proc. International Workshop on Algorithms and Computation (WALCOM)	(PR)(CO)(OA)
C412	2015	S. M. Oh, G. T. Toussaint, E. D. Demaine and M. L. Demaine	A Dissimilarity Measure for Comparing Origami Crease Patterns	Proc. International Conference on Pattern Recognition Applications and Methods (ICPRAM)	(PR)(CO)(OA)
C413	2015	A. Mehta, B. Waggoner and M. Zadimoghaddam	Online Stochastic Matching with Unequal Probabilities	Proc. Symposium on Discrete Algorithms (SODA)	(PR)(CO)(OA)
C414	2015	A. Bhaskara, A. Theertha and M. Zadimoghaddam	Sparse Solutions to Nonnegative Linear Systems and Applications	Proc. International Conference on Artificial Intelligence and Statistics (AISTATS)	(PR)(CO)(OA)
C415	2015	A. Abboud, A. Backurs and V. A. Williams	Tight Hardness Results for LCS and Other Sequence Similarity Measures.	Proc. Symposium on Foundations of Computer Science (FOCS)	(PR)(CO)(OA)
C416	2016	U. Meyer and M. Penschuck	Generating Massive Scale-Free Networks under Resource Constraints	Proc. Workshop on Algorithm Engineering and Experiments (ALENEX)	(PR)(OA)
C417	2016	S. Ashkiani, A. Davidson, U. Meyer and J. Owens	GPU Multisplit	Proc. Symposium on Principles and Practice of Parallel Programming (PPoPP)	(PR) (CO)(OA)
C418	2016	R. Duan, J. Garg and K. Mehlhorn	An Improved Combinatorial Polynomial Algorithm for the Linear Arrow-Debreu Market	Proc. Symposium on Discrete Algorithms (SODA)	(PR) (CO)(OA)
C419	2016	Z. Abel, E.D. Demaine, M.L. Demaine, S. Eisenstat, J. Lynch and T.B. Schardl	Who Needs Crossings? Hardness of Plane Graph Rigidity	Proc. International Conference on Computational Geometry (SoCG)	(PR) (CO)(OA)
C420	2016	E.D. Demaine, G. Viglietta and A. Williams	Super Mario Bros. is Harder/Easier than We Thought	Proc. International Conference on Fun with Algorithms	(CO)(OA)
C421	2016	E.D. Demaine, F. Ma, E. Waingarten, A. Schvartzman and S. Aaronson	The Fewest Clues Problem	Proc. International Conference on Fun with Algorithms	(CO)(OA)
C422	2016	E.D. Demaine, J. Lynch, G.J. Mirano and N. Tyagi	Energy-Efficient Algorithms	Proc. ACM Conference on Innovations in Theoretical Computer Science	(PR) (CO)(OA)
C423	2016	K.G. Larsen and J. Nelson	The Johnson-Lindenstrauss Lemma is Optimal for Linear Dimensionality Reduction	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR) (CO)(OA)

C424	2016	A. Grønlund and K.G. Larsen	Towards Tight Lower Bounds for Range Reporting on the RAM	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(OA)
C425	2016	C. Baum, I. Damgård, K.G. Larsen and M. Nielsen	How to prove knowledge of small secrets	Proc. International Cryptology Conference (CRYPTO)	(PR)(CO)
C426	2016	P. Afshani and J.A.S. Nielsen	Data Structure Lower Bounds for Document Indexing Problems	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR)(OA)
C427	2016	P. Afshani, E. Berglin, I. van Duijn and J.A.S. Nielsen	Applications of incidence bounds in point covering problems	Proc. International Conference on Computational Geometry (SoCG)	(PR)(OA)
C428	2016	P. Afshani, D. R. Sheehy and Y. Stein	Approximating the Simplicial Depth in High Dimensions	Proc. European Workshop on Computational Geometry (EuroCG)	(CO)(OA)
C429	2016	J. Yon, S.W. Bae, S-W. Cheng, O. Cheong and B.T. Wilkinson	Approximating Convex Shapes with respect to Symmetric Difference under Homotheties	Proc. International Conference on Computational Geometry (SoCG)	(PR) (CO)(OA)
C430	2016	C. Tsirogiannis and B. Sandel	Fast Phylogenetic Biodiversity Computations Under a Non-Uniform Random Distribution	Proc. International Conference on Research in Computational Molecular Biology (RECOMB)	(PR)(OA)
C431	2016	C. Alexander, L. Arge, P.K. Bøcher, M. Revsbæk, B. Sandel, J.-C. Svenning, C. Tsirogiannis, and J. Yang	Computing River Floods Using Massive Terrain Data	Proc. International Conference on Geographic Information Science (GIScience)	(PR)(OA)
C432	2016	M. Löffler, F. Staals and J. Urhausen	New Results on Trajectory Grouping under Geodesic Distance	Proc. European Workshop on Computational Geometry (EuroCG)	(CO)(OA)
C433	2016	M. van Kreveld, M. Löffler, F. Staals and L. Wiratma	A Refined Definition for Groups of Moving Entities and its Computation	Proc. European Workshop on Computational Geometry (EuroCG)	(CO)(OA)
C434	2016	A. van Goethem, M. van Kreveld, M. Löffler, B. Speckmann and F. Staals	Grouping Time-varying Data for Interactive Exploration	Proc. Symposium on Computational Geometry (SoCG)	(PR) (CO)(OA)
C435	2016	I. Kostitsnya, M. Löffler, V. Polishchuk and F. Staals	On the complexity of minimum-link path problems	Proc. Symposium on Computational Geometry (SoCG)	(PR) (CO)(OA)
C436	2016	T. Kopelowitz, S. Pettie and E. Porat	Higher Lower Bounds from the 3SUM Conjecture	Proc. Symposium on Discrete Algorithms (SODA)	(PR) (CO)(OA)
C437	2016	G.S. Brodal	External Memory Three-Sided Range Reporting and Top-k Queries with Sublogarithmic Updates	Proc. Symposium on Theoretical Aspects of Computer Science (STACS)	(PR)(OA)
C438	2016	Z. Huang and P. Peng	Dynamic Graph Stream Algorithms in o(n) Space	Proc. International Colloquium on Automata, Languages and Programming (ICALP)	(PR) (CO)(OA)
C439	2016	P. Brandes, Z. Huang, H.-H. Su and R. Wattenhofer	Clairvoyant Mechanisms for Online Auctions	Proc. International Conference on Computing and Combinatorics (COCOON)	(PR) (CO)(OA)
C440	2016	A. Backurs, P. Indyk, I. Razenshteyn and D.P. Woodruff	Nearly-optimal bounds for sparse recovery in generic norms, with applications to k-median sketching	Proc. Symposium on Discrete Algorithms (SODA)	(PR) (CO)(OA)

C441	2016	M. Cheraghchi and P. Indyk	Nearly Optimal Deterministic Algorithm for Sparse Walsh-Hadamard Transform	Proc. Symposium on Discrete Algorithms (SODA)	(PR) (CO)(OA)
C442	2016	S. Har-Peled, P. Indyk, S. Mahabadi and A. Vakilian	Towards Tight Bounds for the Streaming Set Cover Problem	Proc. Symposium on Principles of Database Systems (PODS)	(PR) (CO)(OA)
C443	2016	A. Abboud, A. Backurs, T.D. Hansen, V.V. Williams and O. Zamir	Subtree Isomorphism Revisited	Proc. Symposium on Discrete Algorithms (SODA)	(PR) (CO)(OA)

Journals

J1	2007	G. S. Brodal, R. Fagerberg and G. Moruz	On the Adaptiveness of Quicksort	ACM Journal of Experimental Algorithmics, 12	(PR) (CO)
J2	2008	D. Ajwani, T. Friedrich and U. Meyer	An O(n^{2.75}) Algorithm for Incremental Topological Ordering	ACM Transactions on Algorithms, 4(4)	(PR)
J3	2008	M. Stissing, T. Mailund, C. N. S. Pedersen, G. S. Brodal and R. Fagerberg	Computing the All-Pairs Quartet Distance on a set of Evolutionary Trees	Journal of Bioinformatics and Computational Biology, 6(1)	(PR)(CO)
J4	2008	L. Arge, M. de Berg, H. J. Haverkort and K. Yi	The Priority R-Tree: A Practically Efficient and Worst-Case Optimal R-Tree	ACM Transactions on Algorithms, 4(1)	(PR)(CO)
J5	2009	M. Olsen	Nash Stability in Additively Separable Hedonic Games and Community Structures	Theory of Computing Systems, 45(4)	(PR)
J6	2009	M. Abam, M. de Berg, M. Farshi and J. Gudmundsson	Region-Fault Tolerant Geometric Spanners	Discrete & Computational Geometry, 41(4)	(PR)(CO)
J7	2009	M. Abam, M. de Berg and B. Speckmann	Kinetic kd-Trees and Longest-Side kd-Trees	SIAM Journal of Computing, 39(4)	(PR)(CO)
J8	2009	L. Arge, V. Samoladas and K. Yi	Optimal External-Memory Planar Point Enclosure	Algorithmica, 54(3)	(PR)(CO)
J9	2009	L. Arge, M. de Berg and H. Haverkort	Cache-Oblivious R-Trees	Algorithmica, 53(1)	(PR)(CO)
J10	2009	H. Iben, J. O'Brien and E. Demaine	Refolding Planar Polygons	Discrete & Computational Geometry, 41(3)	(PR)(CO)
J11	2009	E. Demaine, M. Hajiaghayi, H. Mahini, A. Sayedi-Roshkhar, S. Oveisgharan and M. Zadimoghaddam	Minimizing Movement	ACM Transactions on Algorithms, 5(3)	(PR)(CO)
J12	2009	E. Demaine, M. Hajiaghayi and K. Kawarabayashi	Algorithmic Graph Minor Theory: Improved Grid Minor Bounds and Wagner's Contraction	Algorithmica, 54(2)	(PR)(CO)
J13	2009	T. Abbott, M. Burr, T. Chan, E. Demaine, M. Demaine, J. Hugg, D. Kane, S. Langerman, J. Nelson, E. Rafalin, K. Seyboth and V. Yeung	Dynamic Ham-Sandwich Cuts in the Plane	Computational Geometry: Theory and Applications, 42(5)	(PR)(CO)
J14	2009	E.D. Demaine, M. Hajiaghayi, H. Mahini and M. Zadimoghaddam	The Price of Anarchy in Network Creation Games	ACM SIGECOM Exchanges, 8(2)	(PR)(CO)
J15	2009	E.D. Demaine, M.L. Demaine, J. Iacono and S. Langerman	Wrapping Spheres with Flat Paper	Computational Geometry: Theory and Applications, 42(8)	(PR)(CO)
J16	2010	P. Indyk and A. Gilbert	Sparse Recovery Using Sparse Matrices	Proceedings of the IEEE June 2010	(PR)(CO)
J17	2010	E.D. Demaine, S.Langerman and E. Price	Confluently Persistent Tries for Efficient Version Control	Algorithmica 57(3)	(PR)(CO)

J18	2010	M.A. Abam, M. de Berg, P. Hachenberger and A. Zarei	Streaming Algorithms for Line Simplification	Discrete & Computational Geometry 43(3)	(PR)(CO)
J19	2010	M.A. Abam, M. de Berg and J. Gudmundsson	A Simple and Efficient Kinetic Spanner	Computational Geometry: Theory and Applications 43(3)	(PR)(CO)
J20	2010	D. Ajwani and T. Friedrich	Average-case Analysis of Incremental Topological Ordering	Discrete Applied Mathematics 158	(PR)(CO)
J21	2010	H. Blunck and J. Vahrenhold	In-Place Algorithms for Computing (Layers of) Maxima	Algorithmica 57(1)	(PR)(CO)
J22	2010	P. Indyk, Z. Syed, C. Stultz, M. Kellis and J. Guttag	Motif discovery in physiological datasets: A methodology for inferring predictive elements	ACM Transactions on Knowledge Discovery in Data 4(1)	(PR)(CO)
J23	2010	E. Hawkes, B. An, N. M. Benbernou, H. Tanaka, S. Kim, E.D. Demaine, D. Rus and R.J. Wood	Programmable matter by folding	Proceedings of the National Academy of Sciences of the United States of America 107(28)	(PR)(CO)
J24	2010	J.L. Bredin, E.D. Demaine, M. Hajiaghayi and D. Rus	Deploying Sensor Networks with Guaranteed Fault Tolerance	IEEE/ACM Transactions on Networking 18(1)	(PR)(CO)
J25	2010	E.D. Demaine, J. Iacono and S. Langerman	Grid Vertex-Unfolding Orthostacks	International Journal of Computational Geometry and Applications 20(3)	(PR)(CO)
J26	2010	E.D. Demaine, S.P. Fekete, G. Rote, N. Schweer, D. Scymura and M. Zelke	Integer Point Sets Minimizing Average Pairwise L_1 Distance: What is the Optimal Shape of a Town?	Computational Geometry: Theory and Applications 44(2)	(PR)(CO)
J27	2010	R. Connelly, E.D. Demaine, M.L. Demaine, S. Fekete, S. Langerman, J. S. B. Mitchell, A. Ribó and G. Rote	Locked and Unlocked Chains of Planar Shapes	Discrete & Computational Geometry 44(2)	(PR)(CO)
J28	2010	P.K. Agarwal, L. Arge and K. Yi	I/O-Efficient Batched Union-Find and Its Applications to Terrain Analysis	ACM Transactions on Algorithms 7(1)	(PR)(CO)
J29	2010	P. Afshani, C. Hamilton and N. Zeh	A General Approach for Cache-Oblivious Range Reporting and Approximate Range Counting	Computational geometry: Theory and applications 43(8)	(PR)(CO)
J30	2010	J. Katajainen and S. S. Rao	A compact data structure for representing a dynamic multiset	Information Processing Letters 110(23)	(PR)(CO)
J31	2010	M.A. Bender, G.S. Brodal, R. Fagerberg, R. Jacob and E. Vicari	Optimal Sparse Matrix Dense Vector Multiplication in the I/O-Model	Theory of Computing Systems 47(4)	(PR)(CO)
J32	2010	C. Demetrescu, B. Escoffier, G. Moruz and A. Ribichini	Adapting Parallel Algorithms to the W-Stream Model, with Applications to Graph Problems	Theoretical Computer Science 411(44-46)	(PR)(CO)
J33	2011	J. E. Moeslund, L. Arge, P. K. Bøcher, B. Nygaard and J.-C. Svenning	Geographically Comprehensive Assessment of Salt-Meadow Vegetation-Elevation Relations Using LiDAR	Wetlands 31(3)	(PR)(CO)
J34	2011	B. Sandel, L. Arge, B. Dalsgaard, R. Davies, K. Gaston, W. Sutherland and J.-C. Svenning	The influence of Late Quaternary climate-change velocity on species endemism	Science 334	(PR)(CO)

J35	2011	B. Dalsgaard, E. Magård, J. Fjeldså, A.M. Martín González, C. Rahbek, J. Olesen, J. Ollerton, R. Alarcón, A.C. Araujo, P.A. Cotton, C. Lara, C.G. Machado, I. Sazima, M. Sazima, A. Timmermann, S. Watts, B. Sandel, W. Sutherland and J.-C. Svenning	Specialization in Plant-Hummingbird Networks Is Associated with Species Richness, Contemporary Precipitation and Quaternary Climate-Change Velocity	PLoS ONE 6	(PR)(CO)
J36	2011	B. Sandel, M. Krupa and J. Corbin	Using plant functional traits to guide restoration: A case study in California coastal grassland	Ecosphere 2	(PR)(CO)
J37	2011	P. Afshani, C. Hamilton and N. Zeh	Cache-Oblivious Range Reporting With Optimal Queries Requires Superlinear Space	Discrete & Computational Geometry 45(4)	(PR)(CO)
J38	2011	G.S. Brodal, B. Gfeller, A.G. Jørgensen and P. Sanders	Towards Optimal Range Medians	Theoretical Computer Science 412(24)	(PR)(CO)
J39	2011	M. Kutz, G.S: Brodal, K. Kaligosi and I. Katriel	Faster Algorithms for Computing Longest Common Increasing Subsequences	Journal of Discrete Algorithms 9(4)	(PR)(CO)
J40	2011	M.A. Bender, G.S. Brodal, R. Fagerberg, D. Ge, S. He, H. Hu, J. Iacono and A. López-Ortiz	The Cost of Cache-Oblivious Searching	Algorithmica 61(2)	(PR)(CO)
J41	2011	H.L. Chan, T.W. Lam, L.K. Lee and H.F. Ting	Approximating frequent items in asynchronous data stream over a sliding window	Algorithmica 4(3)	(PR) (CO)
J42	2011	C. Daskalakis, R. M. Karp, E. Mossel, S. Riesenfeld and E. Verbin	Sorting and Selection in Posets	SIAM Journal of Computing	(PR)(CO)
J43	2011	M. A. Abam and M. de Berg	Kinetic Spanners in R^d	Discrete & Computational Geometry 45(4)	(PR)(CO)
J44	2011	M. A. Abam, M. de Berg, M. Farshi, J. Gudmundsson and M. H. M. Smid	Geometric Spanners for Weighted Point Sets	Algorithmica 61(1)	(PR)(CO)
J45	2011	M. A. Abam, P. K. Agarwal, M. de Berg and H. Yu	Out-of-Order Event Processing in Kinetic Data Structures	Algorithmica 60(2)	(PR)(CO)
J46	2011	J. Freixas, X. Molinero, M. Olsen and M. J. Serna	On the Complexity of Problems on Simple Games	RAIRO - Operations Research 45(4)	(PR)(CO)
J47	2011	A. Beckman, U. Meyer, P. Sanders and J. Singler	Energy-Efficient Sorting using Solid State Disks	Sustainable Computing: Informatics and Systems 1(2)	(PR)(CO)
J48	2011	E. D. Demaine, S. P. Fekete, G. Rote, N. Schweer, D. Schymura and M. Zelke	Integer Point Sets Minimizing Average Pairwise L1 Distance: What is the Optimal Shape of a Town?	Computational Geometry: Theory and Applications 44(2)	(PR)(CO)
J49	2011	B. An, N. Benbernou, E. D. Demaine and D. Rus	Planning to Fold Multiple Objects from a Single Self-Folding Sheet	Robotica 29(1)	(PR)(CO)

J50	2011	G. Aloupis, S. Collette, M. Damian, E. D. Demaine, R. Flatland, S. Langerman, J. O'Rourke, V. Pinciu, S. Ramaswami, V. Sacristan and S. Wuhrer	Efficient constant-velocity reconfiguration of crystalline robots	Robotica 29(1)	(PR)(CO)
J51	2011	E. D. Demaine, M. L. Demaine, V. Hart, G. N. Price and T. Tachi	(Non)existence of Pleated Folds: How Paper Folds Between Creases	Graphs and Combinatorics 27(3)	(PR)(CO)
J52	2011	E. D. Demaine, M. L. Demaine, V. Hart, J. Iacono, S. Langerman and J. O'Rourke	Continuous Blooming of Convex Polyhedra	Graphs and Combinatorics 27(3)	(PR)(CO)
J53	2011	J. Cardinal, E. D. Demaine, M. L. Demaine, S. Imahori, T. Ito, M. Kiyomi, S. Langerman, R. Uehara and T. Uno	Algorithmic Folding Complexity	Graphs and Combinatorics 27(3)	(PR)(CO)
J54	2011	K. C. Cheung, E. D. Demaine, J. Bachrach and S. Griffith	Programmable Assembly With Universally Foldable Strings (Moteins)	IEEE Transactions on Robotics 27(4)	(PR)(CO)
J55	2011	G. Aloupis, P. Bose, E. D. Demaine, S. Langerman, H. Meijer, M. Overmars and G. T. Toussaint	Computing Signed Permutations of Polygons	International Journal of Computational Geometry and Applications 21(1)	(PR)(CO)
J56	2011	T. Ito, E. D. Demaine, N. J. A. Harvey, C. H. Papadimitriou, M. Sideri, R. Uehara and Y. Uno	On the Complexity of Reconfiguration Problems	Theoretical Computer Science 412(12-14)	(PR)(CO)
J57	2011	H. Ahn, S. Bae, E. D. Demaine, M. L. Demaine, S. Kim, M. Korman, I. Reinbacher and W. Son	Covering points by disjoint boxes with outliers	Computational Geometry: Theory and Applications 44(3)	(PR)(CO)
J58	2011	J. Cardinal, E. D. Demaine, S. Fiorini, G. Joret, S. Langerman, I. Newman and O. Weimann	The Stackelberg Minimum Spanning Tree Game	Algorithmica 59(2)	(PR)(CO)
J59	2011	H. Haverkort and F. van Walderveen	Four-Dimensional Hilbert Curves for R-Trees	Journal of Experimental Algorithmics 16	(PR)(CO)
J60	2012	B. Sandel and J. Corbin	Scale and diversity following manipulation of productivity and disturbance in Californian coastal grasslands.	Journal of Vegetation Science 23(5)	(PR)(CO)(OA)

J61	2012	M. Schleuning, J. Fründ, A-M. Klein, S. Abrahamczyk, R. Alarcón, M. Albrecht, G.K.S. Andersson, S. Bazarian, K. Böhning-Gaese, R. Bommarco, B. Dalsgaard, D.M. Dehling, A. Gotlieb, M. Hagen, T. Hickler, A. Holzschuh, C.N. Kaiser-Bunbury, H. Kreft, R.J. Morris, B. Sandel, W.J. Sutherland, J-C. Svenning, T. Tscharntke, S. Watts, C.N. Weiner, M. Werner, N.M. Williams, C. Winqvist, C.F. Dormann and N. Blüthgen	Specialization of Mutualistic Interaction Networks Decreases toward Tropical Latitudes	Current Biology 22(20)	(PR)(CO)(OA)
J62	2012	B. Sandel and E. Dangremond	Climate change and the invasion of California by grasses	Global Change Biology 18(1)	(PR)(CO)(OA)
J63	2012	P. Afshani, P. K. Agarwal, L. Arge, K. G. Larsen and J. M. Phillips	(Approximate) Uncertain Skylines	Theory of Computing Systems 52(3)	(PR)(CO)(OA)
J64	2012	P. K. Agarwal, L. Arge, H. Kaplan, E. Molad, R. E. Tarjan and K Yi	An Optimal Dynamic Data Structure for Stabbing-Semigroup Queries	SIAM Journal on Computing 41(1)	(PR)(CO)(OA)
J65	2012	L. Arge, G.S. Brodal and S. S. Rao	External memory planar point location with logarithmic updates	Algorithmica 63(1-2)	(PR)(CO)(OA)
J66	2012	G. S. Brodal, P. Davoodi and S. S. Rao	On Space Efficient Two Dimensional Range Minimum Data Structures	Algorithmica 63(4)	(PR)(CO)(OA)
J67	2012	G.S. Brodal, G. Moruz and A. Negoescu	OnlineMin: A Fast Strongly Competitive Randomized Paging Algorithm	Theory of Computing Systems 56(1)	(PR)(OA)
J68	2012	H.L. Chan, T.W. Lam, L.K. Lee and H.F. Ting	Continuous monitoring of distributed data streams over a time-based sliding window	Algorithmica 62(3-4)	(PR)(CO)(OA)
J69	2012	G. Cormode, S. Muthukrishnan, K. Yi and Q. Zhang	Continuos sampling from distributed streams	Journal of the ACM 59(2)	(PR)(CO)(OA)
J70	2012	U. Meyer and N. Zeh	I/O-efficient shortest path algorithms for undirected graphs with random and bounded edge lengths	ACM Transactions on algorithms 8(3)	(PR)(CO)(OA)
J71	2012	F. Gieseke, G. Moruz and J. Vahrenhold	Resilient K-d Trees: K-Means in Space Revisited.	Frontiers of Computer Science 6(2)	(PR)(CO)(OA)
J72	2012	E.D. Demaine, M. Hajiaghayi, H. Mahini and M. Zadimoghaddam	The Price of Anarchy in Network Creation Games	ACM Transactions on Algorithms 8(2)	(PR)(CO)(OA)
J73	2012	O. Aichholzer, F. Aurenhammer, E.D. Demaine, F. Hurtado, P. Ramos and J. Urrutia	On k-convex polygons	Computational Geometry: Theory and Applications 45(3)	(PR)(CO)(OA)
J74	2012	T.G. Abbott, Z. Abel, D. Charlton, E.D. Demaine, M.L. Demaine and S.D. Kominers	Hinged Dissections Exist	Discrete & Computational Geometry 47(1)	(PR)(CO)(OA)

J75	2012	M. Greve, A.M. Lykke, C.W. Fagg, J. Bogaert, I. Friis, R. Marchant, A.R. Marshall, J. Ndayishimiye, B. Sandel, C. Sandom, M. Schmidt, J.R. Timberlake, J.J. Wieringa, G. Zizka and J.-C. Svenning	Continental-scale variability in browser diversity is a major driver of diversity patterns in acacias across Africa	Journal of Ecology 100	(PR)(CO)(OA)
J76	2012	R. Gupta, P. Indyk, E. Price and Y. Rachlin	Compressive Sensing with Local Geometric Features	International Journal of Computational Geometry and Applications 22(4)	(PR)(CO)(OA)
J77	2012	D. Charlton, E.D. Demaine, M.L. Demaine, V. Dujmovic, P. Morin and R. Uehara	Ghost Chimneys	International Journal of Computational Geometry and Applications 22(3)	(PR)(CO)(OA)
J78	2012	E.D. Demaine, M.L. Demaine and R. Uehara	Any Monotone Boolean Function Can Be Realized by Interlocked Polygons	Algorithms 5(1)	(PR)(CO)(OA)
J79	2012	E.D. Demaine and M. Zadimoghaddam	Constant Price of Anarchy in Network-Creation Games via Public-Service Advertising	Internet Mathematics 8(1-2)	(PR)(OA)
J80	2012	L. Moll, S. Tazari and M. Thurley	Computing hypergraph width measures exactly	Information Processing Letters 112(6)	(PR)(CO)(OA)
J81	2012	S. Tazari	Faster approximation schemes and parameterized algorithms on (odd-) H-minor-free graphs	Theoretical Computer Science 417	(PR)(OA)
J82	2012	D. Wu, G. Cong and C. S. Jensen	A Framework for Efficient Spatial Web Object Retrieval	VLDB Journal 21(6)	(PR)(CO)(OA)
J83	2012	D. Wu, M. L. Yiu, G. Cong and C. S. Jensen	Joint Top-K Spatial Keyword Query Processing	IEEE Transaction on Knowledge and Data Engineering 24(10)	(PR)(CO)(OA)
J84	2012	X. Cao, G. Cong, B. Cui, C. S. Jensen and Q. Yuan	Approaches to Exploring Category Information for Question Retrieval in Community Question Answer Archives	ACM Transactions on Information Systems 30(2)	(PR)(CO)(OA)
J85	2012	M. Yiu, L., I. Assent, C. S. Jensen and P. Kalnis	Outsourced Similarity Search on Metric Data Assets	IEEE Transactions on Knowledge and Data Engineering 24(2)	(PR)(CO)(OA)
J86	2012	X. Cao, G. Cong, C. S. Jensen, J. J. Ng, B. C. Ooi, N.-T. Phan and and D. Wu	SWORS: A System for the Efficient Retrieval of Relevant Spatial Web Objects	Proceedings of the VLDB Endowment 5(12)	(PR)(CO)(OA)
J87	2013	J.E. Moeslund, L. Arge, P.K. Bøcher, T. Dalgaard, R. Ejrnæs, M.V. Odgaard and J.-C. Svenning	Topographically controlled soil moisture drives plant diversity patterns within grasslands	Biodiversity and Conservation 22(10)	(PR)(CO)(OA)
J88	2013	J.E. Moeslund, L. Arge, P.K. Bøcher, T. Dalgaard and J.-C. Svenning	Topography as a driver of local terrestrial vascular plant diversity patterns	Nordic Journal of Botany 31(2)	(PR)(CO)(OA)
J89	2013	J.E. Moeslund, L. Arge, P.K. Bøcher, T. Dalgaard, M.V. Odgaard, B. Nygaard and J.-C. Svenning	Topographically controlled soil moisture is the primary driver of local vegetation patterns across a lowland region	Ecosphere 4(7)	(PR)(CO)(OA)
J90	2013	C. Alexander, J.E. Moeslund, P.K. Bøcher, L. Arge and J.-C. Svenning	Airborne laser scanner (LiDAR) proxies for understory light conditions	Remote Sensing of Environment 134	(PR)(CO)(OA)

J91	2013	C. Sandom, L. Dalby, C. Fløjgaard, W.D. Kissling, J. Lenoir, B. Sandel, K. Trøjelsgaard, R. Ernæs and J.-C. Svenning	Mammal predator and prey species richness are strongly linked at macroscales	Ecology 94(5)	(PR)(CO)(OA)
J92	2013	B. Dalsgaard, K. Trøjelsgaard, A.M. Martin González, D. Nogués-Bravo, J. Ollerton, T. Petanidou, B. Sandel, M. Schleuning, Z. Wang, C. Rahbek, W.J. Sutherland, J.-C. Svenning and J.M. Olesen	Historical climate-change influences modularity of pollination networks	Ecography 36(12)	(PR)(CO)(OA)
J93	2013	J.-C. Svenning and B. Sandel	Disequilibrium vegetation dynamics under future climate change	American Journal of Botany	(PR)(OA)
J94	2013	A.B. Smith, B. Sandel, N.J.B. Kraft and S. Carey	Characterizing scale-dependent community assembly using the functional-diversity-area relationship	Ecology 94(11)	(PR)(CO)(OA)
J95	2013	B. Sandel and J.-C. Svenning	Human impacts drive a global topographic signature in tree cover	Nature Communications 4	(PR)(OA)
J96	2013	M. Olsen and M. Revsbæk	Alliance Partitions and Bisection Width for Planar Graphs	Journal of Graph Algorithms and Applications 17(6)	(PR)(OA)
J97	2013	P.K. Agarwal, L. Arge, S. Govindarajan, J. Yang and K. Yi	Efficient external memory structures for range-aggregate queries	Computational Geometry: Theory and Application 46(3)	(PR)(CO)(OA)
J98	2013	A. Sand, G.S. Brodal, R. Fagerberg, C.N.S. Pedersen and T. Mailund	A practical O(n log n) time algorithm for computing the triplet distance on binary trees	BMC Bioinformatics 14	(PR)(CO)(OA)
J99	2013	G.S. Brodal, M. Greve, V. Pandey and S.S. Rao	Integer Representations towards Efficient Counting in the Bit Probe Model	Journal of Discrete Algorithms	(PR)(CO)(OA)
J100	2013	A. Sand, M.K. Holt, J. Johansen, R. Fagerberg, G.S. Brodal, C.N.S. Pedersen and T. Mailund	Algorithms for Computing the Triplet and Quartet Distances for Binary and General Trees	MDPI Biology - Special Issue on Developments in Bioinformatic Algorithms 2(4)	(PR)(CO)(OA)
J101	2013	K. Yi and Q. Zhang	Optimal Tracking of Distributed Heavy Hitters and Quantiles	Algorithmica 65(1)	(PR)(CO)(OA)
J102	2013	P.K. Agarwal, G. Cormode, Z. Huang, J.M. Phillips, Z. Wei and K. Yi.	Mergeable Summaries	ACM Transactions on Database Systems 38(4)	(PR)(CO)(OA)
J103	2013	R. Pagh, Z. Wei, K. Yi and Q. Zhang	Cache-Oblivious Hashing	Algorithmica	(PR)(CO)(OA)
J104	2013	U. Meyer and V. Weichert	Algorithm Engineering für moderne Hardware	Informatik-Spektrum	(PR)(OA)
J105	2013	X. Li, V. Ceikute, C.S. Jensen and K.-L. Tan	Effective Online Group Discovery in Trajectory Databases	IEEE Transactions on Knowledge and Data Engineering 25(12)	(PR)(CO)(OA)
J106	2013	M. Kaul, R.C.-W. Wong, B. Yang and C.S. Jensen	Finding Shortest Paths on Terrains by Killing Two Birds with One Stone	Proceedings of the VLDB Endowment 7(1)	(PR)(CO)(OA)
J107	2013	K. S. Bøgh, A. Skovsgaard and C.S. Jensen	GroupFinder: A New Approach to Top-K Point-of-Interest Group Retrieval	Proceedings of the VLDB Endowment 6(12)	(PR)(CO)(OA)

J108	2013	B. Yang, C. Guo and C.S. Jensen	Travel Cost Inference from Sparse, Spatio-Temporally Correlated Time Series Using Markov Models	Proceedings of the VLDB Endowment 6(9)	(PR)(CO)(OA)
J109	2013	D. Wu, M.L. Yiu and C.S. Jensen	Moving Spatial Keyword Queries: Formulation, Methods, and Analysis	ACM Transactions on Database Systems 38(1)	(PR)(CO)(OA)
J110	2013	L. Chen, G. Cong, C.S. Jensen and D. Wu	Spatial Keyword Query Processing: An Experimental Evaluation	Proceedings of the VLDB Endowment 6(3)	(PR)(CO)(OA)
J111	2013	K. Tzoumas, A. Deshpande and C.S. Jensen	Efficiently Adapting Graphical Models for Cardinality Estimation	The VLDB Journal 22(1)	(PR)(CO)(OA)
J112	2013	B. Ballinger, N. Benbernou, P. Bose, M. Damian, E.D. Demaine, V. Dujmovic, R. Flatland, F. Hurtado, J. Iacono, A. Lubiw, P. Morin, V. Sacristan, D. Souvaine and R. Uehara	Coverage with k-Transmitters in the Presence of Obstacles	Journal of Combinatorial Optimization 25(2)	(PR)(CO)(OA)
J113	2013	S. Butler, E.D. Demaine, R. Graham and T. Tachi	Constructing Points through Folding and Intersection	International Journal of Computational Geometry and Applications 23(1)	(PR)(CO)(OA)
J114	2013	G. Barequet, N. Benbernou, D. Charlton, E.D. Demaine, M.L. Demaine, M. Ishaque, A. Lubiw, A. Schulz, D.L. Souvaine, G.T. Toussaint and A. Winslow	Bounded-Degree Polyhedronization of Point Sets	Computational Geometry: Theory and Applications 46(2)	(PR)(CO)(OA)
J115	2013	J. Cardinal, E.D. Demaine, S. Fiorini, G. Joret, I. Newman and O. Weimann	The Stackelberg Minimum Spanning Tree Game on Planar and Bounded-Treewidth Graphs	Journal of Combinatorial Optimization 25(1)	(PR)(CO)(OA)
J116	2013	G. Aloupis, J. Cardinal, S. Collette, E.D. Demaine, M.L. Demaine, M. Dulieu, R. Fabila-Monroy, V. Hart, F. Hurtado, S. Langerman, M. Saumell, C. Seara and P. Taslakian	Non-crossing matchings of points with geometric objects	Computational Geometry: Theory and Applications 46(1)	(PR)(CO)(OA)
J117	2013	N. Alon, E.D. Demaine, M. Hajiaghayi and T. Leighton	Basic Network Creation Games	SIAM Journal on Discrete Mathematics 27(2)	(PR)(CO)(OA)
J118	2013	E.D. Demaine, S. Eisenstat, M. Ishaque and A. Winslow	One-Dimensional Staged Self-Assembly	Natural Computing 12(2)	(PR)(CO)(OA)
J119	2013	E.D. Demaine, M.L. Demaine, J. Itoh, A. Lubiw, C. Nara and J. O'Rourke	Refold Rigidity of Convex Polyhedra	Computational Geometry: Theory and Applications 46(8)	(PR)(CO)(OA)
J120	2013	G. Aloupis, N. Benbernou, M. Damian, E.D. Demaine, R. Flatland, J. Iacono and S. Wuhrer	Efficient Reconfiguration of Lattice-Based Modular Robots	Computational Geometry: Theory and Applications 46(8)	(PR)(CO)(OA)
J121	2013	Z. Abel, E.D. Demaine, M.L. Demaine, S. Eisenstat, J. Lynch and T.B. Schardl	Finding a Hamiltonian Path in a Cube with Specified Turns is Hard	Journal of Information Processing 21(3)	(PR)(CO)(OA)

J122	2013	E.D. Demaine, M. Ghodsi, M. Hajiaghayi, A.S. Sayedi-Roshkhar and M. Zadimoghaddam	Scheduling to Minimize Gaps and Power Consumption	Journal of Scheduling 16(2)	(PR)(CO)(OA)
J123	2013	Z. Abel, E.D. Demaine, M.L. Demaine, S. Eisenstat, J. Lynch, T.B. Schardl and I. Shapiro-Ellowitz	Folding Equilateral Plane Graphs	International Journal of Computational Geometry and Applications 23(2)	(PR)(CO)(OA)
J124	2013	M. Bateni, M. Hajiaghayi and M. Zadimoghaddam	Submodular secretary problem and extensions	ACM Transactions on Algorithms 9(4)	(PR)(CO)(OA)
J125	2013	A. Elmasry, A. Farzan and J. Iacono	On the hierarchy of distribution-sensitive properties for data structures	Acta Informatica 50(4)	(PR)(CO)(OA)
J126	2013	P. Afshani	Improved pointer machine and I/O lower bounds for simplex range reporting and related problems	International Journal of Computational Geometry and Applications 23 (4-5)	(PR)(OA)
J127	2014	C. Alexander, P.K. Bøcher, L. Arge and J.-C. Svenning	Regional-scale mapping of tree cover, height and main phenological tree types using airborne laser scanning data	Remote Sensing of Environment 147	(PR)(CO)(OA)
J128	2014	M. Schleuning, L. Ingmann, R. Strauß, S. Fritz, B. Dalsgaard, D.M. Dehling, M. Plein, F.V. Saavedra, B. Sandel, J.-C. Svenning, K. Böhning-Gaese and C.F. Cormann	Ecological, historical and evolutionary determinants of modularity in weighted seed-dispersal networks	Ecology Letters 17(4)	(PR)(CO)(OA)
J129	2014	G. Feng, X.C. Mi, P.K. Bøcher, L.F. Mao, B. Sandel, M. Cao, W.H. Ye, Z.Q. Hao, H.D. Gong, Y.T. Zhang, X.H. Zhao, G.Z. Jin, K.P. Ma and J.-C. Svenning	Relative roles of local disturbance, current climate and palaeoclimate in determining phylogenetic and functional diversity in Chinese forests	Biogeosciences 11	(PR)(CO)(OA)
J130	2014	R. Ø. Pedersen, B. Sandel and J.-C. Svenning	Macroecological evidence for competitive regional-scale interactions between the two major clades of mammal carnivores (Feliformia and Caniformia)	PLoS ONE 9(6)	(PR)(CO)(OA)
J131	2014	Dalsgaard, B., D.W. Carstensen, J. Fjeldså, P.K. Maruyama, C. Rahbek, B. Sandel, J. Sonne, J.-C. Svenning, Z. Wang and W.J. Sutherland	Does geography, current climate or historical climate determine bird species richness, endemism and island network roles in Wallacea and the West Indies?	Ecology and Evolution 4(20)	(PR)(CO)(OA)
J132	2014	J.Y. Barnagaud, W.D. Kissling, B. Sandel, W. Eiserhardt, H. Balslev, C.H. Sekercioglu, B. Enquist, C. Tsirogiannis and J.-C. Svenning	Ecological traits influence the phylogenetic structure of bird species co-occurrences worldwide	Ecology Letters 17(7)	(PR)(CO)(OA)

J133	2014	T. Amano, B. Sandel, H. Eager, E. Bulteau, J.-C. Svenning, B. Dalsgaard, C. Rahbek, R.G. Davies and W.J. Sutherland	Global distribution and drivers of language extinction risk	Proceedings of the Royal Academy of Science B: Biological Sciences 281(1793)	(PR)(CO)(OA)
J134	2014	C. Lamanna, B. Blonder, C. Violle, N.J.B. Kraft, B. Sandel, I. Simova, J. Donoghue, J.-C. Svenning, B. McGill, B. Boyle, S. Dolins, P.M. Jørgensen, A. Marcuse-Kubitza, N. Morueta-Holme, R.K. Peet, W.H. Piel, J. Regetz, M. Schildhauer, N. Spencer, B. Theirs, S.K. Wiser and B.J. Enquist	The latitudinal species richness gradient does not arise from a larger functional trait space in the tropics	Proceedings of the National Academy of Sciences 111(38)	(PR)(CO)(OA)
J135	2014	C. Tsirogiannis and B. Sandel	Computing the Skewness of the Phylogenetic Mean Pairwise Distance in Linear Time	Algorithms for Molecular Biology 9(15)	(PR)(OA)
J136	2014	D. Sidlauskas, S. Saltenis and C.S. Jensen	Processing of Extreme Moving-Object Update and Query Workloads in Main Memory	The VLDB Journal 23(5)	(PR)(CO)(OA)
J137	2014	T.M. Chan, S. Durocher, K.G. Larsen, J. Morrison and B.T. Wilkinson	Linear-Space Data Structures for Range Mode Query in Arrays	Theory of Computing Systems 55(4)	(PR)(CO)(OA)
J138	2014	K.G. Larsen	On Range Searching in the Group Model and Combinatorial Discrepancy	SIAM Journal on Computing 43(2)	(PR)(OA)
J139	2014	G. Cormode and H. Jowhari	A second look at counting triangles in graph streams	Theoretical Computer Science 552	(PR)(CO)(OA)
J140	2014	G. S. Brodal, A. C. Kaporis, A. N. Papadopoulos, S. Sioutas, K. Tsakalidis and K. Tsichlas	Dynamic 3-sided planar range queries with expected doubly-logarithmic time	Theoretical Computer Science 526	(PR)(CO)(OA)
J141	2014	A. Sand, M.K. Holt, J. Johansen, G.S. Brodal, T. Mailund and C.N.S. Pedersen	tqDist: A Library for Computing the Quartet and Triplet Distances Between Binary or General Trees	Bioinformatics 30(14)	(PR)(CO)(OA)
J142	2014	K. Yi, L. Wang and Z. Wei	Indexing for Summary Queries: Theory and Practice	ACM Transactions on Database Systems	(PR)(CO)(OA)
J143	2014	M. Abam, S. Daneshpajouh, L. Deleuran, S. Ehsani and M. Ghodsi	Computing Homotopic Line Simplification in a Plane	Computational Geometry: Theory and Applications 47(7)	(PR)(CO)
J144	2014	S. Shang, R. Ding, K. Zheng, C.S. Jensen, P. Kalnis and X. Zhou	Personalized Trajectory Matching in Spatial Networks	The VLDB Journal 23(3)	(PR)(CO)
J145	2014	B. Yang, B., M. Kaul and C.S. Jensen	Using Incomplete Information for Complete Weight Annotation of Road Networks	IEEE Transactions on Knowledge and Data Engineering 26(5)	(PR)(CO)(OA)
J146	2014	G. Moruz, A. Negoescu, C.Neumann and V. Weichert	Engineering Efficient Paging Algorithms	Journal of Experimental Algorithmics 19	(PR)(CO)(OA)

J147	2014	E. D. Demaine, M. Hajiaghayi and D. Marx	Minimizing Movement: Fixed-Parameter Tractability	ACM Transactions on Algorithms 11(2)	(PR)(CO)(OA)
J148	2014	E. D. Demaine, M. L. Demaine, R. Uehara, T. Uno and Y. Uno	UNO is hard, even for a single player	Theoretical Computer Science 521	(PR)(CO)(OA)
J149	2014	M. Damian, E. D. Demaine and R. Flatland	Unfolding Orthogonal Polyhedra with Quadratic Refinement: The Delta-Unfolding Algorithm	Graphs and Combinatorics 30(1)	(PR)(CO)(OA)
J150	2014	G. Borradaile, E. D. Demaine and S.Tazari	Polynomial-Time Approximation Schemes for Subset-Connectivity Problems in Bounded-Genus Graphs	Algorithmica 68(2)	(PR)(CO)(OA)
J151	2014	G. S. Brodal, M. Greve, V. Pandey and S. S. Satti	Integer representations towards efficient counting in the bit probe model	Journal of Discrete Algorithms 26	(PR)(CO)(OA)
J152	2014	Sandom, C., S. Faurby, B. Sandel, J.-C. Svenning	Global Late Quaternary megafauna extinctions linked to humans, not climate change.	Proceedings of the Royal Society B: Biological Sciences 281(1787)	(PR)(CO)(OA)
J153	2014	Kissling, D., L. Dalby, C. Fløjgaard, J. Lenoir, B. Sandel, C. Sandom, K. Trøjelsgaard, J.-C. Svenning.	Establishing macroecological trait datasets: digitalization, extrapolation and validation of diet preferences in terrestrial mammals worldwide	Ecology and Evolution 4(14)	(PR)(CO)(OA)
J154	2014	H.-K. Ahn, H.-S. Kim, S.-S. Kim and W. Son	Computing k Centers over Streaming Data for Small k	International Journal of Computational Geometry and Applications 24(2)	(PR)(CO)(OA)
J155	2014	T. Jurkiewicz and K. Mehlhorn	On a Model of Virtual Address Translation	Journal of Experimental Algorithmics 19	(PR)(CO)(OA)
J156	2014	A. C. Gilbert, P. Indyk, M. Iwen and L. Schmidt	Recent Developments in the Sparse Fourier Transform: A compressed Fourier transform for big data	Signal Processing Magazine 31(5)	(PR)(CO)(OA)
J157	2014	S. Felton, M. Tolley, E. Demaine, D. Rus and R. Wood	A method for building self-folding machines	Science 345(6197)	(PR)(CO)(OA)
J158	2014	E. D. Demaine, M. L. Demaine, Y. N. Minsky, J. S. B. Mitchell, R. L. Rivest and M. Patrascu	Picture-Hanging Puzzles	Theory of Computing Systems 54(4)	(PR)(CO)(OA)
J159	2014	O. Aichholzer, G. Aloupis, E. D. Demaine, M. L. Demaine, S. P. Fekete, M. Hoffmann, A. Lubiw, J. Snoeyink and A. Winslow	Covering Folded Shapes	Journal of Computational Geometry 5(1)	(PR)(CO)(OA)
J160	2014	E. D. Demaine, Y. Okamoto, R. Uehara and Y. Uno	Computational complexity and an integer programming model of Shakashaka	IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 97-A(6)	(PR)(CO)(OA)
J161	2014	Z. Abel, E. D. Demaine, M. L. Demaine, T. Horiyama and R. Uehara	Computational Complexity of Piano-Hinged Dissections	IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 97-A(6)	(PR)(CO)(OA)
J162	2014	T. Ito and E. D. Demaine	Approximability of the Subset Sum Reconfiguration Problem	Journal of Combinatorial Optimization 28(3)	(PR)(CO)(OA)
J163	2015	W. Son, H.-K. Ahn and S. W. Bae	Group Nearest-Neighbor Queries in the L_1 Plane	Theoretical Computer Scienc 592	(PR)(CO)(OA)

J164	2015	G. S. Brodal, S. Sioutas, K. Tsichlas and C. Zaroliagis	D^2-Tree: A New Overlay with Deterministic Bounds	Algorithmica 72(3)	(PR)(CO)(OA)
J165	2015	S. Pettie and H.-H. Su	Distributed Coloring Algorithms for Triangle-Free Graphs	Information and Computation 243	(PR)(OA)
J166	2015	S. Pettie	Sensitivity Analysis of Minimum Spanning Trees in Sub-Inverse-Ackermann Time	Journal of Graph Algorithms and Applications 19(1)	(PR)(OA)
J167	2015	S. Pettie	Three Generalizations of Davenport-Schinzel Sequences	SIAM Journal on Discrete Mathematics 29(4)	(PR)(OA)
J168	2015	E. Sebastián-González, B. Sandel, B. Dalsgaard and P. Guimarães	Macroecological trends in nestedness and modularity of seed-dispersal networks.	Global Ecology and Biogeography 24(3)	(PR)(CO)(OA)
J169	2015	K. Engemann, B. Enquist, B. Sandel, B. Boyle, P. Jørgensen, N. Morueta-Holme, R. Peet, C. Violle and J.-C. Svenning	Limited sampling hampers 'big data' estimation of species richness in a tropical biodiversity hotspot.	Ecology and Evolution, 5(3)	(PR)(CO)(OA)
J170	2015	B. Sandel	Towards a taxonomy of spatial scale-dependence	Ecography 38(4)	(PR)
J171	2015	T. M. Chan, S. Durocher, M. Skala and B. T. Wilkinson	Linear-Space Data Structures for Range Minority Query in Arrays	Algorithmica 72(4)	(PR)(CO)(OA)
J172	2015	K. G. Larsen, J. I. Munro, J. S. Nielsen and S. Thankachan	On Hardness of Several String Indexing Problems	Theoretical Computer Science 582	(PR)(CO)(OA)
J173	2015	A. Kovacs, U. Meyer, G. Moruz and A. Negoescu	The optimal structure of algorithms for alpha-paging.	Information Processing Letters, 115(12)	(PR)(OA)
J174	2015	A. Termehchy, A. Vakilian, Y. Chodpathumwan and M. Winslett	Cost-Effective Database Design For Information Extraction	ACM Transactions on Database Systems 40(2)	(PR)(CO)(OA)
J175	2015	E. D. Demaine, J. Iacono and S. Langerman	Worst-Case Optimal Tree Layout in External Memory	Algorithmica 72(2)	(PR)(CO)(OA)
J176	2015	B. Blonder, D. Nogués-Bravo, M.K. Borregaard, J. Donoghue II, P.M. Jørgensen, N.J.B. Kraft, J.-P. Lessard, N. Morueta-Holme, B. Sandel, J.-C. Svenning, C. Violle, C. Rahbek and B.J. Enquist	Linking environmental filtering and disequilibrium to biogeography with a community climate framework	Ecology 96(4)	(PR)(CO)
J177	2015	B. Sandel, A.G. Gutiérrez, P.B. Reich, F. Schrodt, J. Dickie and J. Kattge	Estimating the missing species bias in plant trait measurements	Journal of Vegetation Science 26(5)	(PR)(CO)
J178	2015	Sandel, B., A.G. Gutiérrez, P.B. Reich, F. Schrodt, J. Dickie and J. Kattge	Late Cenozoic climate change and the phylogenetic structure of regional conifer floras worldwide	Global Ecology and Biogeography 24(10)	(PR)(CO)
J179	2015	J-C. Svenning, W. Eiserhardt, S. Normand, A. Ordonez and B. Sandel	The influence of paleoclimate on present-day patterns in biodiversity and ecosystems.	Annual Reviews in Ecology, Evolution and Systematics 46	(PR)(CO)(OA)
J180	2015	M. Pasgaard, N. Strange, B. Dalsgaard, P.K.M. Mendonça and B. Sandel	Geographical imbalances and divides in the scientific production of climate change knowledge	Global Environmental Change 35	(PR)(CO)(OA)
J181	2015	L. Arge and M. Thorup	RAM-efficient External Memory Sorting	Algoritmica 73(4)	(PR)(CO)(OA)

J182	2015	G. S. Brodal, G. Moruz and A. Negoescu	OnlineMin: A Fast Strongly Competitive Randomized Paging Algorithm,	Theory of Computing Systems 56(1)	(PR)(OA)
J183	2015	J. Brody and K. G. Larsen	Adapt or Die: Polynomial Lower Bounds for Non-Adaptive Dynamic Data Structures	Theory of Computing 11(19)	(PR)(CO)(OA)
J184	2015	P. Wollstadt, U. Meyer, and M. Wibral	A Graph Algorithmic Approach to Separate Direct from Indirect Neural Interactions	PLoS ONE, 10(10)	(PR)(CO)(OA)
J185	2015	C. Hegde, P. Indyk and L. Schmidt	Approximation Algorithms for Model-Based Compressive Sensing	IEEE Transactions of Information Theory 61(9)	(PR)(CO)(OA)
J186	2015	E. D. Demaine, F. Ma, M. Susskind and E. Waingarten	You Should Be Scared of German Ghost	Journal of Information Processing 23(3)	(PR)(CO)(OA)
J187	2015	G. Aloupis, E. D. Demaine, A. Guo and G. Viglietta	Classic Nintendo Games are (Computationally) Hard	Theoretical Computer Science 586	(PR)(CO)(OA)
J188	2015	K. Yamanaka, E. D. Demaine, T. Ito, J. Kawahara, M. Kiyomi, Y. Okamoto, T. Saitoh, A. Suzuki, K. Uchizawa and T. Uno	Swapping Labeled Tokens on Graphs	Theoretical Computer Science 586	(PR)(CO)(OA)
J189	2015	A. B. Adcock, E. D. Demaine, M. L. Demaine, M. P. O'Brien, F. Reidl, F. S. Villaamil and B. D. Sullivan	Zig-Zag Numberlink is NP-Complete	Journal of Information Processing 23(3)	(PR)(CO)(OA)
J190	2015	E. D. Demaine and M. L. Demaine	Fun with Fonts: Algorithmic Typography	Theoretical Computer Science 586	(PR)(OA)
J191	2015	E. D. Demaine, M. L. Demaine, E. Fox-Epstein, D. A. Hoang, T. Ito, H. Ono, Y. Otachi, R. Uehara and T. Yamada	Linear-time algorithm for sliding tokens on trees	Theoretical Computer Science 600	(PR)(CO)(OA)
J192	2016	E.D. Demaine, M.J. Patitz, T.A. Rogers, R.T. Schweller, S.M. Summers and D. Woods	The Two-Handed Tile Assembly Model is not Intrinsically Universal	Algorithmica 74(2)	(PR)(CO)
J193	2016	E.D. Demaine, D. Eppstein, A. Hesterberg, H. Ito, A. Lubiw, R. Uehara and Y. Uno	Folding a paper strip to minimize thickness	Journal of Discrete Algorithms 36	(PR)(CO)
J194	2016	J-C. Svenning, P.B.M. Pedersen, C.J. Donlan, R. Ejrnaes, S. Faurby, M. Galetti, D.M. Hansen, B. Sandel, C.J. Sandom, J.W. Terborgh and F.W.M. Vera	Science for a wilder Anthropocene - synthesis and future directions for rewilding research	Proceedings of the National Academy of Sciences, 113	(PR) (CO)(OA)
J195	2016	J-C. Svenning, P.B.M. Pedersen, C.J. Donlan, R. Ejrnaes, S. Faurby, M. Galetti, D.M. Hansen, B. Sandel, C.J. Sandom, J.W. Terborgh and F.W.M. Vera	Reply to Rubenstein and Rubenstein: Time to move on from ideological debates on rewilding	Proceedings of the National Academy of Sciences, USA 113	(PR) (CO)(OA)

J196	2016	J. Sonne, A.M. Martin González, P.K. Maruyama, B. Sandel, S. Abrahamczyk, R. Alarcon, A.C. Araujo, F.P. Araujo, S.M. de Azevedo Jr., A.C. Baquero, D. Nogues-Bravo, P.A. Cotton., T.T. Ingversen, G. Kohler, C. Lara, F.M.G. Las-Casas, A.O. Machado, C.G. Machado, M.A. Maglianesi, A.C. Moura, G.M. Olivera, P.E.Oliveira, J.F. Ornelas, L. da Cruz Rodrigues, L. Rosero-Lasprilla, A.M. Rui, M. Sazima, M. Schleuning, A. Timmermann, I.G. Varassin, J. Vinzentin-Bugoni, Z. Wang, S. Watts, J. Fjeldså, J.-C. Svenning, C. Rahbek and B. Dalsgaard	High proportion of smaller ranged hummingbird species coincides with ecological specialization across the Americas	Proceedings of the Royal Society B,. 283	(PR) (CO)(OA)
J197	2016	G. Feng, L. Mao, B. Sandel, N.G. Swenson and J.-C. Svenning	High plant endemism in China is partially linked to reduced glacial-interglacial climate change	Journal of Biogeography 43	(PR) (CO)(OA)
J198	2016	C. Doughty, A. Wolf, N. Morueta-Holme, P.M. Jørgensen, B. Sandel, C. Violle, B. Boyle, N.J.B. Kraft, R.K. Peet, B.J. Enquist, J.-C. Svenning, S. Blake and M. Galetti	Megafauna extinction, tree species range reduction, and carbon storage in Amazonian forests	Ecography 39	(PR) (CO)(OA)
J199	2016	D. N. Karger, A. Cord, M. Kessler, H. Kreft, I. Kühn, S. Pompe, B. Sandel, J. Sarmento-Cabral, A. Smith, J.-C. Svenning, H. Tuomisto, P. Weigelt and K. Wesche	Delineating probabilistic species pools in ecology and biogeography	Global Ecology and Biogeography 25	(PR) (CO)(OA)
J200	2016	K. Engemann, B. Sandel, B.J. Enquist, P.M. Jørgensen, N.J.B. Kraft, A. Marcuse-Kubitza, B. McGill, N. Morueta-Holme, R.K. Peet, C. Violle, S. Wiser and J.-C. Svenning	Patterns and drivers of plant functional group dominance across the Western Hemisphere - a macroecological re-assessment based on a massive novel botanical dataset	Botanical Journal of the Linnean Society 180	(PR) (CO)(OA)
J201	2016	Y. Li, X. Li, B. Sandel, D. Blank, Z. Liu, X. Liu and S. Yan.	Climate and topography explain range sizes of terrestrial vertebrates	Nature Climate Change 6	(PR) (CO)(OA)
J202	2016	K. Milton, D.A. Nolin, K. Ellis, J. Lozier, B. Sandel and E. Lacey	Genetic, spatial, and social relationships among adults in a group of Howler Monkeys (Alouatta palliata) from Barro Colorado Island, Panama	Primates 57	(PR) (CO)(OA)

J203	2016	Z. Ma, B. Sandel and J.-C. Svenning	Phylogenetic assemblage structure of North American trees is more strongly shaped by glacial-interglacial climate variability in gymnosperms than in angiosperms	Ecology and Evolution 6	(PR) (CO)(OA)
J204	2016	D. Arroyuelo, P. Davoodi and S.S. Rao	Succinct Dynamic Cardinal Trees	Algorithmica 74(2)	(PR) (CO)(OA)
-	To Appear	C. Tsirogiannis and B. Sandel	PhyloMeasures: A package for computing phylogenetic biodiversity measures and their statistical moments	Ecography	(PR)(CO)
-	To Appear	N. Morueta-Holme, B. Blonder, B. Sandel, B. McGill, R.K. Peet, J. Ott, C. Violle, B. Enquist, P.M. Jørgensen and J.-C. Svenning	A network approach for inferring species associations from co-occurrence data	Ecography	(PR)(CO)
-	To Appear	T. K. Nielsen, B.M. Benito, J.-C. Svenning, B. Sandel, L. McKerracher, F. Reide and P.C. Kjaergaard	Investigating Neanderthal dispersal above 55N in Europe during Marine Isotope Stage 5	Quaternary International	(PR)(CO)
-	To Appear	G.R. Goldsmith, N. Morueta-Holme, B. Sandel, E.D. Fitz, S.D. Fitz, B. Boyle, N. Casler, R. Condit, S. Dolins, J. Donoghue, K. Engemann, P.M. Jørgensen, N.J.B. Kraft, A. Marcuse-Kubitza, B. McGill, R.K. Peet, W. Piel, J. Regetz, M. Schildhauer, N. Spencer, J.-C. Svenning, B. Thiers, C. Violle, S. Wiser and B. Enquist	Plant-O-Matic: A dynamic and mobile field guide to all plants of the Americas	Methods in Ecology and Evolution	(PR)(CO)
-	To Appear	B. Sandel, A.-C. Monnet and M. Vorontsova	Multidimensional structure of grass functional traits among species and assemblages	Journal of Vegetation Science	(PR)(CO)
-	To Appear	J.S. Ku and E.D. Demaine	Folding Flat Crease Patterns With Thick Materials	Journal of Mechanisms and Robotics	(PR)
-	To Appear	T.M. Chan and B.T. Wilkinson	Adaptive and Approximate Orthogonal Range Counting	ACM Transactions on Algorithms	(PR)(CO)
-	To Appear	B. Sandel and C. Tsirogiannis	Species Introductions and the Phylogenetic and Functional Structure of California's Grasses	Ecology	(PR)
-	To Appear	B. Sandel and C. Tsirogiannis	Fast Computation of Measures of Phylogenetic Beta Diversity	PLoS ONE	(PR)
-	To Appear	A. Kalvisa, C. Tsirogiannis, I. Silamikelis, G. Skenders, L. Broka, A. Zirnitis, D. Bandere, I. Jansone and R. Ranka	MIRU-VNTR genotype diversity and indications of homoplasy in M. avium strains isolated from humans and slaughter pigs in Latvia	Journal of Infection, Genetics and Evolution	(PR) (CO)
-	To Appear	L. Barenboim, M. Elkin, S. Pettie and J. Schneider	The Locality of Distributed Symmetry Breaking	Journal of the ACM	(PR)(CO)

Thesis

T1	2007	I. Brudaru	Heuristics for Average Diameter Approximation with External Memory Algorithms	MPI	MS Thesis
T2	2007	G. Moruz	Hardware-Aware Algorithms and Data Structures	AU	PhD Thesis
T3	2008	M. Patrascu	Lower Bound Techniques for Data Structures	MIT	PhD Thesis
T4	2008	A. Sidiropoulos	Computational metric embeddings	MIT	PhD Thesis
T5	2008	D. Ajwani	Traversing large graphs in realistic settings	MPI	PhD Thesis
T6	2008	K. Do Ba	Testing closeness of distributions under the EMD metric	MIT	MS Thesis
T7	2008	K. Lai	Complexity of Union-Split-Find Problems	MIT	MS Thesis
T8	2008	J. M. Larsen og M. Nielsen	En undersøgelse af algoritmer til løsning af generalized movers problem i 3D	AU	MS Thesis
T9	2008	C. Andersen	An optimal minimum spanning tree algorithm	AU	MS Thesis
T10	2008	M. Revsbæk	I/O-efficient Algorithms for Batched Union-Find with Dynamic Set Properties and its Applications to Hydrological Conditioning	AU	MS Thesis
T11	2008	A. H. Jensen	I/O-efficient Processing of LIDAR Data	AU	MS Thesis
T12	2009	M. Olsen	Link Building	AU	PhD Thesis
T13	2009	T. Mølhave	Handling Massive Terrains and Unreliable Memory, AU	AU	PhD Thesis
T14	2009	H. B. Kirk	Searching with Dynamic Optimality: In Theory and Practice	AU	MS Thesis
T15	2009	K. Piatkowski	Implementering og udvikling af maksimum delsum algoritmer	AU	MS Thesis
T16	2009	O. Weimann	Accelerating Dynamic Programming	MIT	PhD Thesis
T17	2009	V. Weichert	Radiation parameterization of the climate model COSMO/CLM in CUDA	FRA	MS Thesis
T18	2009	R. Berinde	Advances in Sparse Signal Recovery Methods	MIT	MS Thesis
T19	2009	P. Davoodi	Two Dimensional Range Minimum Queries	AU	MS Thesis
T20	2009	K. Tsakalidis	External Memory 3-sided Planar Range Reporting and Persistent B-Trees	AU	MS Thesis
T21	2009	L. Deleuran	Polygonal Line Simplification	AU	MS Thesis
T22	2010	A. G. Jørgensen	Data Structures: Sequence Problems, Range Queries, and Fault Tolerance	AU	PhD Thesis
T23	2010	J. Moeslund	Fine-resolution geospatial modelling of contemporary and potential future plant diversity in Denmark	AU	MS Thesis
T24	2010	J. Truelsen	Working Set Implicit Dictionaries and Range Mode Lower Bounds and Approximations	AU	MS Thesis

T25	2010	M. Greve	Online Sorted Range Reporting and Approximating the Mode	AU	MS Thesis
T26	2010	D. Kjær	Range Media Algorithms	AU	MS Thesis
T27	2010	J. Suhr Christensen	Experimental Study of Kinetic Geometric t-Spanner Algorithms	AU	MS Thesis
T28	2011	K. G. Larsen	Optimal Orthogonal Range Reporting in 3-d	AU	MS Thesis
T29	2011	C. Kejlberg-Rasmussen	On Implicit Dictionaries with the Working-Set Property and Catenable Priority Queues with Attrition	AU	MS Thesis
T30	2011	P. Davoodi	Data Structures: Range Queries and Space Efficiency	AU	PhD Thesis
T31	2011	K. Tsakalidis	Dynamic Data Structures: Orthogonal Range Queries and Update Efficiency	AU	PhD Thesis
T32	2011	J. Nelson	Sketching and Streaming High-Dimensional Vectors	MIT	PhD Thesis
T33	2012	J. E. Moeslund	The role of topography in determining local plant diversity patterns across Denmark	AU	PhD Thesis
T34	2012	F. van Walderveen	External Memory Graph Algorithms and Range Searching Data Structures	AU	PhD Thesis
T35	2012	L. Deleuran	Homotopic Polygonal Line Simplification	AU	PhD Thesis
T36	2012	C. Neumann	Practical Paging Algorithms	FRA	MS Thesis
T37	2012	D. Veith	Implementation of an External-Memory Diameter Approximation	FRA	MS Thesis
T38	2012	M. Sturmann	k-Dimensionale Orthogonale Bereichsanfragen für GPUs auf großen Instanzen	FRA	MS Thesis
T39	2012	P. Wollstadt	A Graph Algorithmic Approach to Separate Direct from Indirect Neural Interactions by Identifying Alternative Paths with Similar Weights	FRA	BS Thesis
T40	2012	E. Deza	An efficient implementation of the optimal paging algorithm	FRA	BS Thesis
T41	2012	T. Morgan	Map Folding	MIT	MS Thesis
T42	2012	R. Gupta	A Compressive Sensing Algorithm for Attitude Determination	MIT	MS Thesis
T43	2012	A. Koefoed-Hansen	Representations for Path Finding in Planar Environments	AU	MS Thesis
T44	2013	K.G. Larsen	Models and Techniques for Proving Data Structure Lower Bounds	AU	PhD Thesis (OA)
T45	2013	C. Kejlberg-Rasmussen	Dynamic Data Structures: The Interplay of Invariants and Algorithm	AU	PhD Thesis (OA)
T46	2013	J. Fogh	Engineering a Fast Fourier Transform	AU	MS Thesis (OA)
T47	2013	M. Holt and J. Johansen	Computing Triplet and Quartet Distances	AU	MS Thesis (OA)
T48	2013	J. Schou	Range Minimum Data Structures	AU	MS Thesis

T49	2013	A. Negoescu	Design of Competitive Paging Algorithms with Good Behaviour in Practice	FRA	PhD Thesis (OA)
T50	2013	D. Pick	Effiziente Algorithmen auf Eingebetteten Plattformen	FRA	MS Thesis
T51	2013	T. Timmer	I/O-effiziente Durchmesser-Approximierung auf gewichteten Graphen	FRA	MS Thesis
T52	2013	D. Frascaria	Improved results for (h,k)-paging	FRA	MS Thesis
T53	2013	S. Försch	An efficient implementation of PARTITION2	FRA	MS Thesis
T54	2013	A. Kehlenbach	Interaktive Stadtplanungsmaßnahmen auf der GPU	FRA	MS Thesis
T55	2013	S. Bechtold	Shortest Paths with Multiple Constraints in Flight Networks	FRA	MS Thesis
T56	2013	V. Ceikute	Inferring Groups of Objects, Preferred Routes, and Facility Locations from Trajectories	AU	PhD Thesis (OA)
T57	2013	E. Price	Sparse Recovery and Fourier Sampling	MIT	PhD Thesis
T58	2013	L. Schmidt	Model-Based Compressive Sensing with Earth Mover's Distance Constraints	MIT	MS Thesis
T59	2013	S. Mahabadi	Approximate Nearest Neighbor And Its Many Variants	MIT	MS Thesis
T60	2013	F. Mogensen	Locating Points of Interest Based on Geo- tagged Tweets	AU	MS Thesis
T61	2013	C.W. Schmidt	Indicering af spatio-tekstuelle data – et empirisk studie	AU	MS Thesis
T62	2014	J.M. Friis and S.B. Olesen	An Experimental Comparison of Max Flow Algorithms	AU	MS Thesis (OA)
T63	2014	D. W. Petersen	Orthogonal Range Skyline Counting Queries	AU	MS Thesis (OA)
T64	2014	J. Kunert	Hashing and Random Graphs	AU	MS Thesis (OA)
T65	2014	B. Mortensen	Algorithms for Computing Convex Hulls Using Linear Programming	AU	MS Thesis (OA)
T66	2014	M. Revsbæk	Handling Massive and Dynamic Terrain Data	AU	MS Thesis (OA)
T67	2014	A. Skovsgaard	Indexing, Query Processing, and Clustering of Spatio-Temporal Text Objects	AU	PhD Thesis
T68	2014	Samir van de Sand	Eine adaptive Tabusuche für das Vehicle Routing Problem mit Zeitfenstern	FRA	MS Thesis
T69	2014	Morteza Zadimoghaddam	Online Allocation Algorithms with Applications in Computational Advertising	MIT	PhD Thesis
T70	2015	J. Truelsen	Space Efficient Data Structures and External Terrain Algorithms	AU	PhD Thesis

T71	2015	J. Yang	Efficient Algorithms for Handling Massive Terrains	AU	PhD Thesis
T72	2015	J. S. Nielsen	Implicit Data Structures, Sorting, and Text Indexing	AU	PhD Thesis
T73	2015	C. Jespersen	Monte Carlo Evaluation of Financial Options Using a GPU	AU	MS Thesis (OA)
T74	2015	B. Mortensen	Algorithms for Computing Convex Hulls Using Linear Programming	AU	MS Thesis (OA)
T75	2015	K. V. Ebbesen	On the Practicality of Data-Oblivious Sorting	AU	MS Thesis (OA)
T76	2015	J. H. Knudsen and R. L. Pedersen	Engineering Rank and Select Queries on Wavelet Trees	AU	MS Thesis (OA)
T77	2015	J. C. C. Jensen	Event Detection in Soccer using Spatio-Temporal Data	AU	MS Thesis (OA)
T78	2015	M. E. Hougaard	On the Complexity of Red-Black Trees for Higher Dimensions	AU	MS Thesis (OA)
T79	2015	S. N. Madsen and R. Hallenberg-Larsen	Computing Set Operations on Simple Polygons Using Binary Space Partition Trees	AU	MS Thesis (OA)
T80	2015	L. Walther	Intersection of Convex Objects in the Plane	AU	MS Thesis (OA)
T81	2015	A. S-H. Vinther and M. S-H. Vinther	Pathfinding in Two-dimensional Worlds	AU	MS Thesis (OA)
T82	2015	T. Sandholt and M. B. Christensen	Geometric Measures of Depth	AU	MS Thesis (OA)
T83	2015	M. Ravn	Orthogonal Range Searching in 2D with Ball Inheritance	AU	MS Thesis (OA)
T84	2015	H. Hassanieh	The Sparse Fourier Transform: Theory & Practice	MIT	PhD Thesis
T85	2015	N. Schepsen	STXXL - Paging-Algorithmen	FRA	BS Thesis
T86	2015	D. Roß	Cohesion in scientific collaboration and citation networks	FRA	MS Thesis
T87	2016	J. Landbo and C. Green	Range Mode Queries in Arrays	AU	MS Thesis (OA)
T88	2016	H.K. Christensen	Algorithms for Finding Dominators in Directed Graphs	AU	MS Thesis (OA)

Other

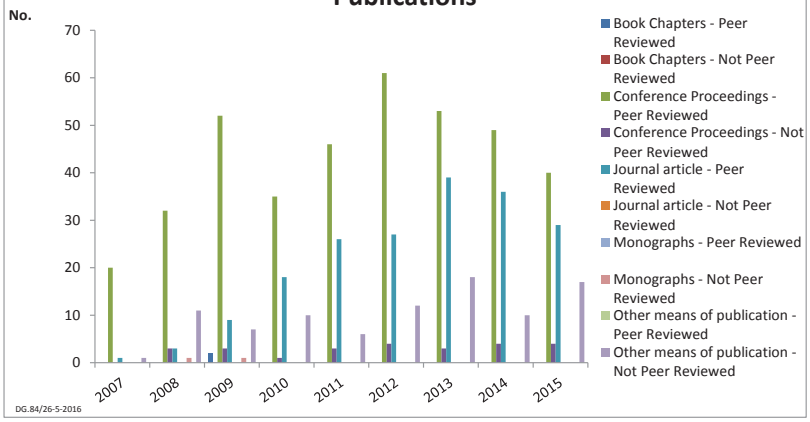
O1	2008	E. Demaine, B. Gassend, J. O'Rourke, and G. T. Toussaint	All Polygons Flip Finitely ... Right?	In "Surveys on Discrete and Computational Geometry: Twenty Years Later", Contemporary Mathematics 453	(CO)
O2	2008	A. Andoni and P. Indyk	Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions	Communications of the ACM, 51(1)	(CO)
O3	2008	K. Mehlhorn and P. Sanders	Algorithms and Data Structures: The Basic Toolbox	Springer Verlag	(CO)
O4	2009	D. Ajwani and U. Meyer	Design and Engineering of External Memory Traversal Algorithms for general graphs	In Algorithmic of Large and Complex Networks, Springer Verlag	(PR)
O5	2009	L. Arge and N. Zeh	External-memory Algorithms and Data Structures	In Algorithms and Theory of Computation Handbook, CRC Press	(PR)(CO)

O6	2009	R. Hearn and E. Demaine	Games, Puzzles, and Computation	A.K. Peters	(CO)
O7	2010	D. Ajwani and H. Meyerhenke	Realistic Computer Models	In Algorithm Engineering. Bridging the Gap Between Algorithm Theory and Practice, Springer Verlag	(CO)
O8	2011	H. Balslev, L. Arge, J.-C. Svenning, M. H. Schierup and C. S. Jensen	Abstracts of Royal Danish Academy of Sciences Symposium on Biodiversity in the Silicon Age		(CO)
O9	2012	L. Arge and K. G. Larsen	I/O-Efficient Spatial Data Structures for Range Queries	Invited abstract in SIGSPATIAL Special, July, 2012.	
O10	2012	B. Sandel, L. Arge, B. Dalsgaard, R.G. Davies, K.J. Gaston, W.J. Sutherland and J.-C. Svenning	Response - Global endemism needs spatial integration	Science 335	(CO)
O11	2014	U. Meyer and N. Zeh	I/O-model	Encyclopedia of Algorithms, second edition	(CO)
O12	2014	U. Meyer and N. Zeh	List-Ranking	Encyclopedia of Algorithms, second edition	(CO)

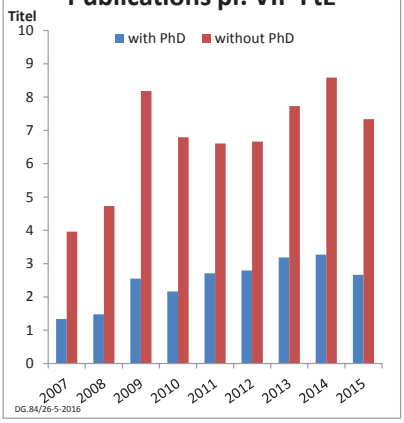
C. Center statistics

In the following, the center statistics collected by the foundation is enclosed. Note that since the same statistics is collected for all centers, not all of it is particularly relevant or meaningful for MADALGO. For example, since the tradition in the algorithms field is that only researcher that contributed very substantially to a result is a co-author on the paper with the results, and since authors are (almost) always listed alphabetically, statistics on “centerleder as co-author” and “centerleader is either first or last author” are not very relevant. Also, note that publication and citation statistics is also given in Section B.

Publications



Publications pr. VIP FtE



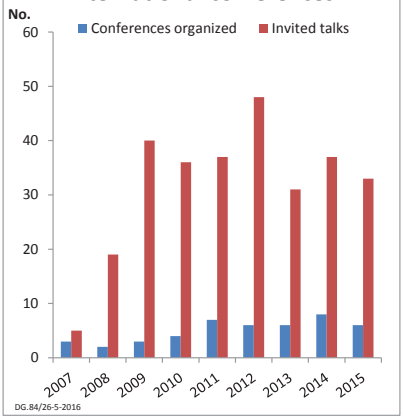
No journal impact factor data available.

DG.84/26-5-2016

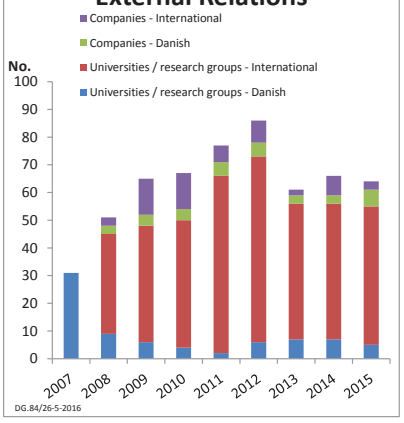
No citation data available.

DG.84/26-5-2016

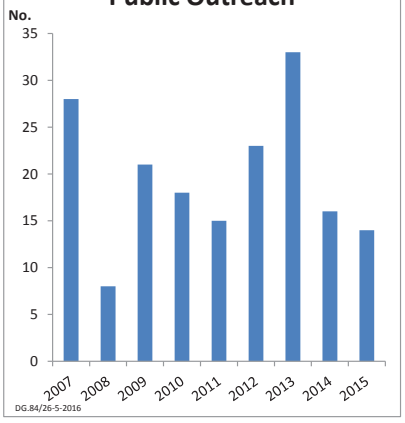
International conferences



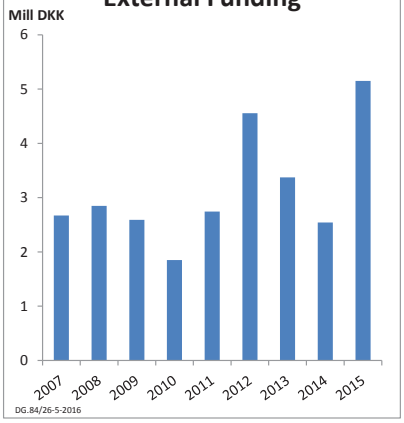
External Relations



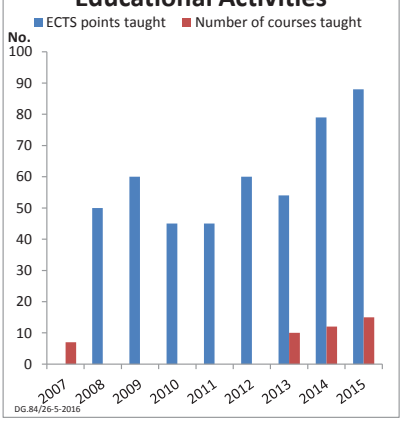
Public Outreach



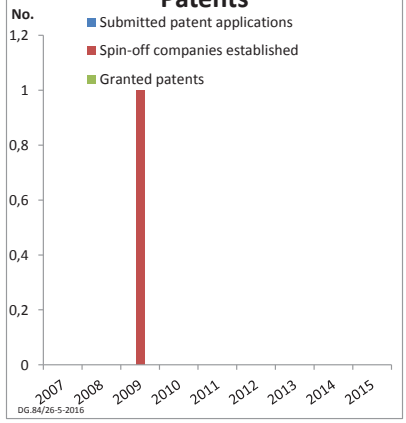
External Funding



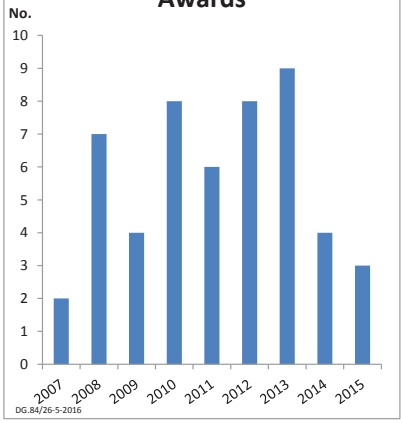
Educational Activities

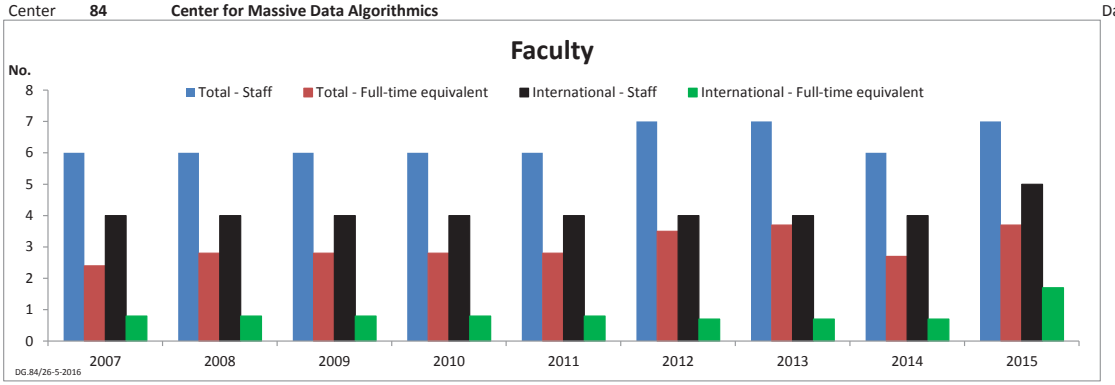


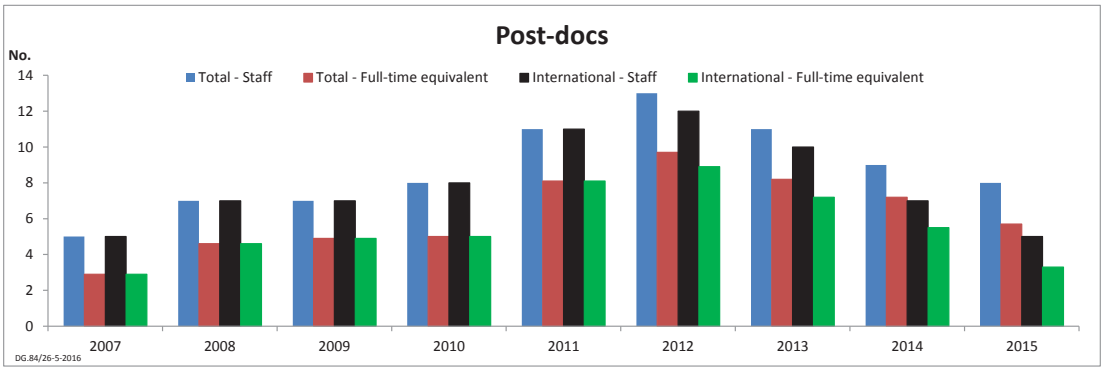
Patents

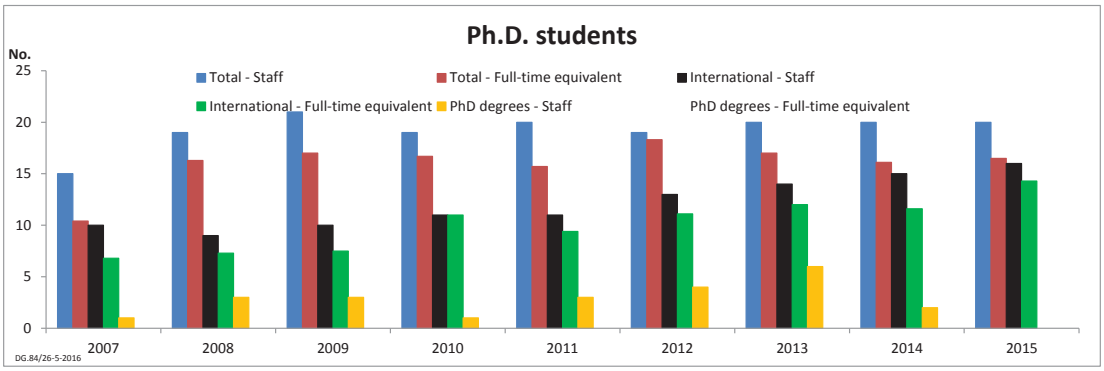


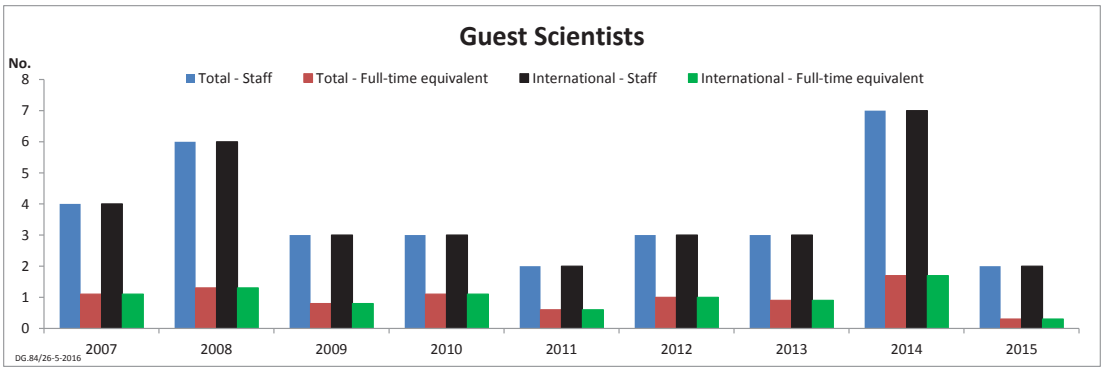
Awards

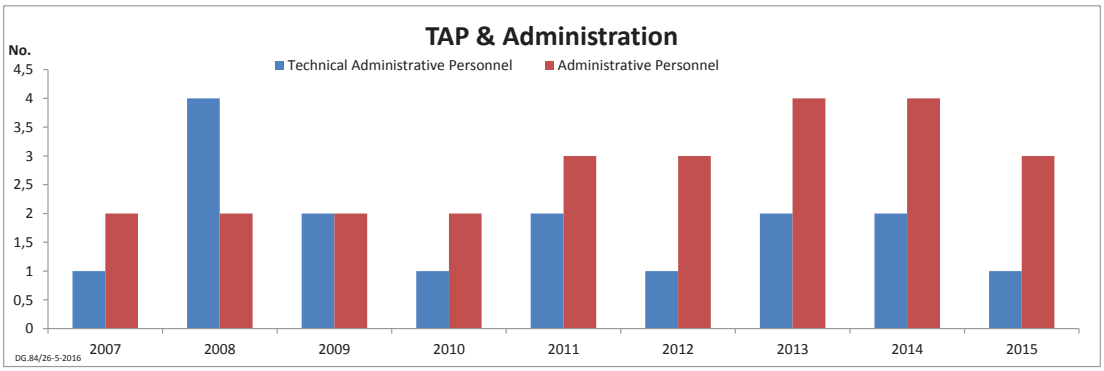


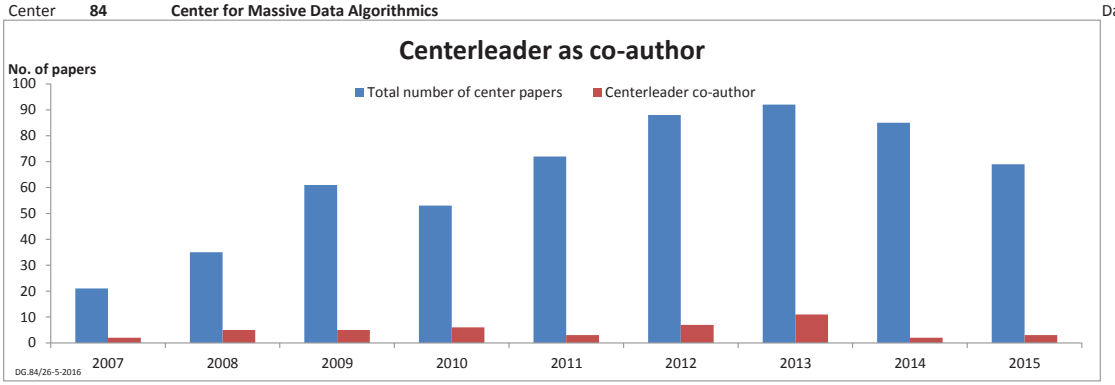

Faculty
No.
Total - Staff
Total - Full-time equivalent
International - Staff
International - Full-time equivalent
2007
2008
2009
2010
2011
2012
2013
2014
2015
DG.84/26-5-2016

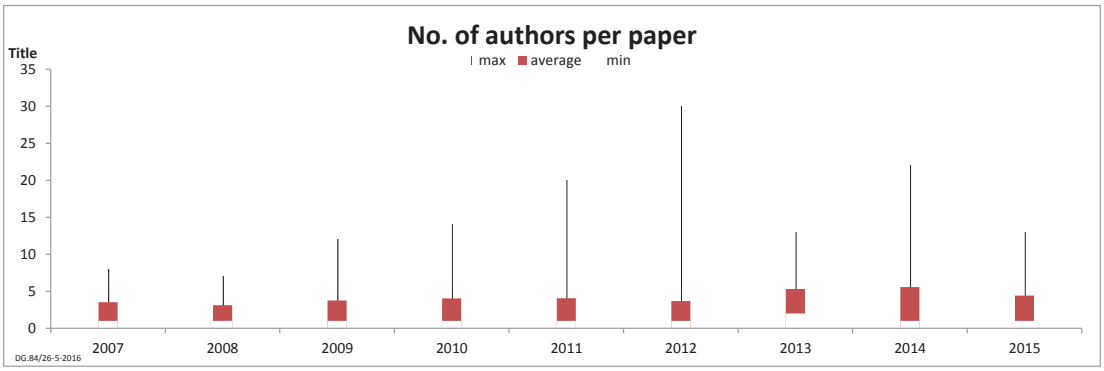

Post-docs
No.
Total - Staff
Total - Full-time equivalent
International - Staff
International - Full-time equivalent
2007
2008
2009
2010
2011
2012
2013
2014
2015
DG.84/26-5-2016

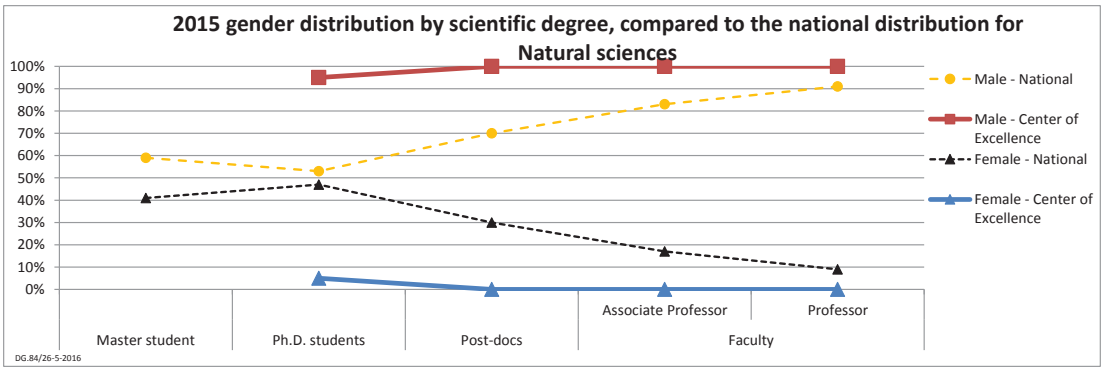

Ph.D. students
No.
Total - Staff
Total - Full-time equivalent
International - Staff
International - Full-time equivalent
PhD degrees - Staff
PhD degrees - Full-time equivalent
2007
2008
2009
2010
2011
2012
2013
2014
2015
DG.84/26-5-2016


Guest Scientists
No.
Total - Staff
Total - Full-time equivalent
International - Staff
International - Full-time equivalent
2007
2008
2009
2010
2011
2012
2013
2014
2015
DG.84/26-5-2016


TAP & Administration
No.
Technical Administrative Personnel
Administrative Personnel
2007
2008
2009
2010
2011
2012
2013
2014
2015
DG.84/26-5-2016







Number of scientists included: Ph.D. students: 20 Post-docs: 8 Associate Professors: 1 Professors: 6
Not included: 0

Selected publications

The below ten papers are selected in an attempt to both cover the breadth of the center's research and to showcase some of the main center results. This is obviously not a simple task, and many important and interesting papers could not be selected. The number of each paper in the center publication list is listed below each paper, along with a few keywords describing the paper.

1. L. Arge and M. Thorup
   RAM-Efficient External Memory Sorting
   Proc. International Symposium on Algorithms and Computation (ISAAC), 2013
   C267 (I/O and internal memory optimality – best paper award)

2. L. Arge, M. Revsbæk and N. Zeh
   I/O-Efficient Computation of Water Flow Across a Terrain
   Proc. Symposium on Computational Geometry (SoCG), 2010
   C127 (I/O and terrain flood risk)

3. G.S. Brodal, E. Demaine, J.T. Fineman, J. Iacono, S. Langerman and J.I. Munro
   Cache-Oblivious Dynamic Dictionaries with Optimal Update/Query Tradeoff
   Proc. Symposium on Discrete Algorithms (SODA), 2010
   C119 (Cache-oblivious data structures)

4. D.M. Kane, J. Nelson and D.P. Woodruff
   An Optimal Algorithm for the Distinct Elements Problem
   Proc. Symposium on Principles of Database Systems (PODS), 2010
   C112 (Streaming algorithms – best paper award)

5. K. G. Larsen, J. Nelson and H. L. Nguyen
   Time Lower Bounds for Nonadaptive Turnstile Streaming Algorithms
   Proc. Symposium on Theory of Computing (STOC), 2015
   C382 (Streaming algorithms)

6. C. Alexander, L. Arge, P.K. Bøcher, M. Revsbæk, B. Sandel, J.-C. Svenning,
   C. Tsirogiannis, and J. Yang
   Computing River Floods Using Massive Terrain Data
   Proc. International Conference on Geographic Information Science (GIScience), 2016
   C431 (I/O, algorithm engineering and terrain flood risk)

7. B. Sandel, L. Arge, B. Dalsgaard, R. Davies, K. Gaston, W. Sutherland and J.-C. Svenning
   The influence of Late Quaternary climate-change velocity on species endemism
   Science 334
   J34 (Interdisciplinary collaboration and algorithm engineering)

8. M. Patrascu
   Succincter
   Proc. Symposium on Foundations of Computer Science (FOCS), 2008
   C23 (Succinct data structures – best student paper award)

9. G.S. Brodal, G. Lagogiannis and R.E. Tarjan
   Strict Fibonacci Heaps
   Proc. Symposium on Theory of Computing (STOC), 2012
   C214 (Classical data structures)

10. K. G. Larsen
    The Cell Probe Complexity of Dynamic Range Counting
    Proc. Symposium on Theory of Computing (STOC), 2012
    C199 (Lower bounds – best paper and best student paper award)

RAM-Efficient External Memory Sorting

Lars Arge[1,⋆] and Mikkel Thorup[2,⋆⋆]

[1] MADALGO⋆⋆⋆, Aarhus University, Aarhus, Denmark
[2] University of Copenhagen,† Copenhagen, Denmark

Abstract. In recent years a large number of problems have been considered in external memory models of computation, where the complexity measure is the number of blocks of data that are moved between slow external memory and fast internal memory (also called I/Os). In practice, however, internal memory time often dominates the total running time once I/O-efficiency has been obtained. In this paper we study algorithms for fundamental problems that are simultaneously I/O-efficient and internal memory efficient in the RAM model of computation.

1 Introduction

In the last two decades a large number of problems have been considered in the external memory model of computation, where the complexity measure is the number of blocks of elements that are moved between external and internal memory. Such movements are also called I/Os. The motivation behind the model is that random access to external memory, such as disks, often is many orders of magnitude slower than random access to internal memory; on the other hand, if external memory is accessed sequentially in large enough blocks, then the cost per element is small. In fact, disk systems are often constructed such that the time spent on a block access is comparable to the time needed to access each element in a block in internal memory.

Although the goal of external memory algorithms is to minimize the number of costly blocked accesses to external memory when processing massive datasets, it is also clear from the above that if the internal processing time per element in a block is large, then the practical running time of an I/O-efficient algorithm is dominated by internal processing time. Often I/O-efficient algorithms are in fact not only efficient in terms of I/Os, but can also be shown to be internal memory efficient in the comparison model. Still, in many cases the practical running time of I/O-efficient algorithms is dominated by the internal computation time. Thus both from a practical and a theoretical point of view it is interesting to investigate

⋆ Supported in part by the Danish National Research Foundation and the Danish National Advanced Technology Foundation.
⋆⋆ Supported in part by an Advanced Grant from the Danish Council for Independent Research under the Sapere Aude research career program.
⋆⋆⋆ Center for Massive Data Algorithmics—a center of the Danish National Research Foundation.
† Part of this work was done while the author was at AT&T Labs–Research.

L. Cai, S.-W. Cheng, and T.-W. Lam (Eds.): ISAAC2013, LNCS 8283, pp. 491–501, 2013.

how internal-memory efficient algorithms can be obtained while simultaneously ensuring that they are I/O-efficient. In this paper we consider algorithms that are both I/O-efficient and efficient in the RAM model in internal memory.

Previous results. We will be working in the standard external memory model of computation, where M is the number of elements that fit in main memory and an I/O is the process of moving a block of B consecutive elements between external and internal memory [1]. We assume that $N \geq 2M$, $M \geq 2B$ and $B \geq 2$. Computation can only be performed on elements in main memory, and we will assume that each element consists of one word. We will sometime assume the comparison model in internal memory, that is, that the only computation we can do on elements are comparisons. However, most of the time we will assume the RAM model in internal memory. In particular, we will assume that we can use elements for addressing, e.g. trivially implementing permuting in linear time. Our algorithms will respect the standard so-called *indivisibility assumption*, which states that at any given time during an algorithm the original N input elements are stored somewhere in external or internal memory. Our internal memory time measure is simply the number of performed operations; note that this includes the number of elements transferred between internal and external memory.

Aggarwal and Vitter [1] described sorting algorithms using $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os. One of these algorithms, external merge-sort, is based on $\Theta(M/B)$-way merging. First $O(N/M)$ sorted runs are formed by repeatedly sorting M elements in main memory, and then these runs are merged together $\Theta(M/B)$ at a time to form longer runs. The process continues for $O(\log_{M/B}\frac{N}{M})$ phases until one is left with one sorted list. Since the initial run formation and each phase can be performed in $O(N/B)$ I/Os, the algorithm uses $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os. Another algorithm, external distribution-sort, is based on $\Theta(\sqrt{M/B})$-way splitting. The N input elements are first split into $\Theta(\sqrt{M/B})$ sets of roughly equal size, such that the elements in the first set are all smaller than the elements in the second set, and so on. Each of the sets are then split recursively. After $O(\log_{\sqrt{M/B}}\frac{N}{M}) = O(\log_{M/B}\frac{N}{M})$ split phases each set can be sorted in internal memory. Although performing the split is somewhat complicated, each phase can still be performed in $O(N/B)$ I/Os. Thus also this algorithm uses $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os.

Aggarwal and Vitter [1] proved that external merge- and distribution-sort are I/O-optimal when the comparison model is used in internal memory, and in the following we will use $sort_E(N)$ to denote the number of I/Os *per block of elements* of these optimal algorithms, that is, $sort_E(N) = O(\log_{M/B}\frac{N}{B})$ and external comparison model sort takes $\Theta(\frac{N}{B}sort_E(N))$ I/Os. (As described below, the I/O-efficient algorithms we design will move $O(N \cdot sort_E(N))$ elements between internal and external memory, so $O(sort_E(N))$ will also be the per element internal memory cost of obtaining external efficiency.) When no assumptions other than the indivisibility assumption are made about internal memory computation (i.e. covering our definition of the use of the RAM model in internal memory), Aggarwal and Vitter [1] proved that permuting N elements according to a given permutation requires $\Omega(\min\{N, \frac{N}{B}sort_E(N)\})$ I/Os. Thus this is also a lower

bound for RAM model sorting. For all practical values of N, M and B the bound is $\Omega(\frac{N}{B}sort_E(N))$. Subsequently, a large number of I/O-efficient algorithms have been developed. Of particular relevance for this paper, several priority queues have been developed where insert and deletemin operations can be performed in $O(\frac{1}{B}sort_E(N))$ I/Os amortized [2,4,8]. The structure by Arge [2] is based on the so-called buffer-tree technique, which uses $O(M/B)$-way splitting, whereas the other structures also use $O(M/B)$-way merging.

In the RAM model the best known sorting algorithm uses $O(N \log\log N)$ time [6]. Similar to the I/O-case, we use $sort_I(N) = O(\log\log N)$ to denote the *per element* cost of the best known sorting algorithm. If randomization is allowed then this can be improved to $O(\sqrt{\log\log n})$ expected time [7]. A priority queue can also be implemented so that the cost per operation is $O(sort_I(N))$ [9].

Our results. In Section 2 we first discuss how both external merge-sort and external distribution-sort can be implemented to use optimal $O(N \log N)$ time if the comparison model is used in internal memory, by using an $O(N \log N)$ sorting algorithm and (in the merge-sort case) an $O(\log N)$ priority queue. We also show how these algorithms can relatively easily be modified to use

$$O(N \cdot (sort_I(N) + sort_I(M/B) \cdot sort_E(N))) \text{ and}$$

$$O(N \cdot (sort_I(N) + sort_I(M) \cdot sort_E(N)))$$

time, respectively, if the RAM model is used in internal memory, by using an $O(N \cdot sort_I(N))$ sorting algorithm and an $O(sort_I(N))$ priority queue.

The question is of course if the above RAM model sorting algorithms can be improved. In Section 2 we discuss how it seems hard to improve the running time of the merge-sort algorithm, since it uses a priority queue in the merging step. By using a linear-time internal-memory splitting algorithm, however, rather than an $O(N \cdot sort_I(N))$ sorting algorithm, we manage to improve the running time of external distribution-sort to

$$O(N \cdot (sort_I(N) + sort_E(N))).$$

Our new *split-sort* algorithm still uses $O(\frac{N}{B}sort_E(N))$ I/Os. Note that for small values of M/B the $N \cdot sort_E(N)$-term, that is, the time spent on moving elements between internal and external memory, dominates the internal time. Given the conventional wisdom that merging is superior to splitting in external memory, it is also surprising that a distribution algorithm outperforms a merging algorithm.

In Section 3 we develop an I/O-efficient RAM model priority queue by modifying the buffer-tree based structure of Arge [2]. The main modification consists of removing the need for sorting of $O(M)$ elements every time a so-called buffer-emptying process is performed. The structure supports insert and deletemin operations in $O(\frac{1}{B}sort_E(N))$ I/Os and $O(sort_I(N) + sort_E(N))$ time. Thus it can be used to develop another $O(\frac{N}{B}sort_E(N))$ I/O and $O(N \cdot (sort_I(N) + sort_E(N)))$ time sorting algorithm.

Finally, in Section 4 we show that when $\frac{N}{B}sort_E(N) = o(N)$ (and our sorting algorithms are I/O-optimal), any I/O-optimal sorting algorithm must transfer

a number of elements between internal and external memory equal to $\Theta(B)$ times the number of I/Os it performs, that is, it must transfer $\Omega(N \cdot sort_E(N))$ elements and thus also use $\Omega(N \cdot sort_E(N))$ internal time. In fact, we show a lower bound on the number of I/Os needed by an algorithm that transfers $b \leq B$ elements on the average per I/O, significantly extending the lower bound of Aggarwal and Vitter [1]. The result implies that (in the practically realistic case) when our split-sort and priority queue sorting algorithms are I/O-optimal, they are in fact also CPU optimal in the sense that their running time is the sum of an unavoidable term and the time used by the best known RAM sorting algorithm. As mentioned above, the lower bound also means that the time spent on moving elements between internal and external memory resulting from the fact that we are considering I/O-efficient algorithms can dominate the internal computation time, that is, considering I/O-efficient algorithms implies that less internal-memory efficient algorithms can be obtained than if not considering I/O-efficiency. Furthermore, we show that when $B \leq M^{1-\varepsilon}$ for some constant $\varepsilon > 0$ (the *tall cache assumption*) the same $\Omega(N \cdot sort_E(N))$ number of transfers are needed for any algorithm using less than $\varepsilon N/4$ I/Os (even if it is not I/O-optimal).

To summarize our contributions, we open up a new area of algorithms that are both RAM-efficient and I/O-efficient. The area is interesting from both a theoretical and practical point of view. We illustrate that existing algorithms, in particular multiway merging based algorithms, are not RAM-efficient, and develop a new sorting algorithm that is both efficient in terms of I/O and RAM time, as well as a priority queue that can be used in such an efficient algorithm. We prove a lower bound that shows that our algorithms are both I/O and internal-memory RAM model optimal. The lower bound significantly extends the Aggarwal and Vitter lower bound [1], and shows that considering I/O-efficient algorithms influences how efficient internal-memory algorithms can be obtained.

2 Sorting

External merge-sort. In external merge-sort $\Theta(N/M)$ sorted runs are first formed by repeatedly loading M elements into main memory, sorting them, and writing them back to external memory. In the first merge phase these runs are merged together $\Theta(M/B)$ at a time to form longer runs. The merging is continued for $O(\log_{M/B} \frac{N}{M}) = O(sort_E(N))$ merge phases until one is left with one sorted run. It is easy to realize that M/B runs can be merged together in $O(N/B)$ I/Os: We simply load the first block of each of the runs into main memory, find and output the B smallest elements, and continue this process while loading a new block from the relevant run every time all elements in main memory from that particular run have been output. Thus external merge-sort uses $O(\frac{N}{B} \log_{M/B} \frac{N}{M}) = O(\frac{N}{B} sort_E(N))$ I/Os.

In terms of internal computation time, the initial run formation can trivially be performed in $O(N/M \cdot M \log M) = O(N \log M)$ time using any $O(N \log N)$ in-

ternal sorting algorithm. Using an $O(\log(M/B))$ priority queue to hold the minimal element from each of the M/B runs during a merge, each of the $O(\log_{M/B} \frac{N}{M})$ merge phases can be performed in $O(N \log \frac{M}{B})$ time. Thus external merge-sort can be implemented to use $O(N \log M + \log_{M/B} \frac{N}{M} \cdot N \log \frac{M}{B}) = O(N \log M + N \log \frac{N}{M}) = O(N \log N)$ time, which is optimal in the comparison model.

When the RAM model is used in internal memory, we can improve the internal time by using a RAM-efficient $O(M \cdot sort_I(M))$ algorithm in the run formation phase and by replacing the $O(\log(M/B))$ priority queue with an $O(sort_I(M/B))$ time priority queue [9]. This leads to an $O(N \cdot (sort_I(M) + sort_I(M/B) \cdot sort_E(N))$ algorithm. There seems no way of avoiding the extra $sort_I(M/B)$-term, since that would require an $O(1)$ priority queue.

External distribution-sort. In external distribution-sort the input set of N elements is first split into $\sqrt{M/B}$ sets $X_0, X_1, \ldots, X_{\sqrt{M/B}-1}$ defined by $s = \sqrt{M/B} - 1$ split elements $x_1 < x_2 < \ldots < x_s$, such that all elements in X_0 are smaller than x_1, all elements in $X_{\sqrt{M/B}-1}$ are larger than or equal to x_s, and such that for $1 \leq i \leq \sqrt{M/B} - 2$ all elements in X_i are larger than or equal to x_i and smaller than x_{i+1}. Each of these sets is recursively split until each set is smaller than M (and larger than $M/(M/B) = B$) and can be sorted in internal memory. If the s split elements are chosen such that $|X_i| = O(N/s)$ then there are $O(\log_s \frac{N}{B}) = O(\log_{M/B} \frac{N}{B}) = O(sort_E(N))$ split phases. Aggarwal and Vitter [1] showed how to compute a set of s split elements with this property in $O(N/B)$ I/Os. Since the actual split of the elements according to the split elements can also be performed in $O(N/B)$ I/Os (just like merging of M/B sorted runs), the total number of I/Os needed by distribution-sort is $O(\frac{N}{B} sort_E(N))$.

Ignoring the split element computation it is easy to implement external distribution-sort to use $O(N \log N)$ internal time in the comparison model: During a split we simply hold the split elements in main memory and perform a binary search among them with each input element to determine to which set X_i the element should go. Thus each of the $O(\log_{M/B} \frac{N}{B})$ split phases uses $O(N \log \sqrt{M/B})$ time. Similarly, at the end of the recursion we sort $O(N/M)$ memory loads using $O(N \log M)$ time in total. The split element computation algorithm of Aggarwal and Vitter [1], or rather its analysis, is somewhat complicated. Still it is easy to realize that it also works in $O(N \log M)$ time as required to obtain an $O(N \log N)$ time algorithm in total. The algorithm works by loading the N elements a memory load at a time, sorting them and picking every $\sqrt{M/B}/4$'th element in the sorted order. This obviously requires $O(N/M \cdot M \log M) = O(N \log M)$ time and results in a set of $4N/\sqrt{M/B}$ elements. Finally, a linear I/O and time algorithm is used $\sqrt{M/B}$ times on this set of elements to obtain the split elements, thus using $O(N)$ additional time.

If we use a RAM sorting algorithm to sort the memory loads at the end of the split recursion, the running time of this part of the algorithm is reduced to $O(N \cdot sort_I(M))$. Similarly, we can use the RAM sorting algorithm in the split element computation algorithm, resulting in an $O(N \cdot sort_I(M))$ algorithm and

consequently a $sort_I(M)$-term in the total running time. Finally, in order to avoid the binary search over $\sqrt{M/B}$ split elements in the actual split algorithm, we can modify it to use sorting instead: To split N elements among s splitting elements stored in s/B blocks in main memory, we allocate a buffer of one block in main memory for each of the $s+1$ output sets. Thus in total we require $s/B + (s+1)B < M/2$ of the main memory for split elements and buffers. Next we repeatedly bring $M/2$ elements onto main memory, sort them, and distribute them to the $s+1$ buffers, while outputting the B elements in a buffer when it runs full. Thus this process requires $O(N \cdot sort_I(M))$ time and $O(N/B)$ I/Os like the split element finding algorithm. Overall this leads to an $O(N \cdot (sort_I(M) + sort_I(M) \cdot sort_E(N)))$ time algorithm.

Split-sort. While it seems hard to improve the RAM running time of the external merge-sort algorithm, we can actually modify the external distribution-sort algorithm further and obtain an algorithm that in most cases is optimal both in terms of I/O and time. This *split-sort* algorithm basically works like the distribution-sort algorithm with the split algorithm modification described above. However, we need to modify the algorithm further in order to avoid the $sort_I(M)$-term in the time bound that appears due to the repeated sorting of $O(M)$ elements in the split element finding algorithm, as well as in the actual split algorithm.

First of all, instead of sorting each batch of $M/2$ elements in the split algorithm to split them over $s = \sqrt{M/B} - 1 < \sqrt{M/2}$ split elements, we use a previous result that shows that we can actually perform the split in linear time.

Lemma 1 (Han and Thorup [7]). *In the RAM model N elements can be split over $N^{1-\varepsilon}$ split elements in linear time and space for any constant $\varepsilon > 0$.*

Secondly, in order to avoid the sorting in the split element finding algorithm of Aggarwal and Vitter [1], we design a new algorithm that finds the split elements on-line as part of the actual split algorithm, that is, we start the splitting with no split elements at all and gradually add at most $s = \sqrt{M/B} - 1$ split elements one at a time. An online split strategy was previously used by Frigo et al [5] in a cache-oblivious algorithm setting. More precisely, our algorithm works as follows. To split N input elements we, as previously, repeatedly bring $M/2$ elements onto main memory, distribute them to buffers using the current split elements and Lemma 1, while outputting the B elements in a buffer when it runs full. However, during the process we keep track of how many elements are output to each subset. If the number of elements in a subset X_i becomes $2N/s$ we pause the split algorithm, compute the median of X_i and add it to the set of splitters, and split X_i at the median element into two sets of size N/s. Then we continue the splitting algorithm.

It is easy to see that the above splitting process results in at most $s+1$ subsets containing between N/s and $2N/s - 1$ elements each, since a set is split when it has $2N/s$ elements and each new set (defined by a new split element) contains at least N/s elements. The actual median computation and the split of X_i can be performed in $O(|X_i|) = O(N/s)$ time and $O(|X_i|/B) = O(N/sB)$ I/Os [1]. Thus if we charge this cost to the at least N/s elements that were inserted in X_i

since it was created, each element is charged $O(1)$ time and $O(1/B)$ I/Os. Thus each distribution phase is performed in linear time and $O(N/B)$ I/Os, leading to an $O(N \cdot (sort_I(M) + sort_E(N)))$ time algorithm.

Theorem 1. *The split-sort algorithm can be used to sort N elements in $O(N \cdot (sort_I(M) + sort_E(N)))$ time and $O(\frac{N}{B} sort_E(N))$ I/Os.*

Remarks. Since $sort_I(M) + sort_E(N) \geq sort_I(N)$ our split-sort algorithm uses $\Omega(N \cdot sort_I(N))$ time. In Section 4 we prove that the algorithm in some sense is optimal both in terms of I/O and time. Furthermore, we believe that the algorithm is simple enough to be of practical interest.

3 Priority Queue

In this section we discuss how to implement an I/O- and RAM-efficient priority queue by modifying the I/O-efficient buffer tree priority queue [2].

Structure. Our external priority queues consists of a fanout $\sqrt{M/B}$ B-tree [3] T over $O(N/M)$ leaves containing between $M/2$ and M elements each. In such a tree, all leaves are on the same level and each node (except the root) has fan-out between $\frac{1}{2}\sqrt{M/B}$ and $\sqrt{M/B}$ and contains at most $\sqrt{M/B}$ splitting elements defining the element ranges of its children. Thus T has height $O(\log_{\sqrt{M/B}} \frac{N}{M}) = O(sort_E(N))$. To support insertions efficiently in a "lazy" manner, each internal node is augmented with a *buffer* of size M and an *insertion buffer* of size at most B is maintained in internal memory. To support deletemin operations efficiently, a RAM-efficient priority queue [9] supporting both deletemin and deletemax,[1] called the *mini-queue*, is maintained in main memory containing the up to $M/2$ smallest elements in the priority queue.

Insertion. To perform an insertion we first check if the element to be inserted is smaller than the maximal element in the mini-queue, in which case we insert the new element in the mini-queue and continue the insertion process with the currently maximal element in the mini-queue. Next we insert the element to be inserted in the insertion buffer. When we have collected B elements in the insertion buffer we insert them in the buffer of the root. If this buffer now contains more than $M/2$ elements we perform a *buffer-emptying process* on it, "pushing" elements in the buffer one level down to buffers on the next level of T: We load the $M/2$ oldest elements into main memory along with the less than $\sqrt{M/B}$ splitting elements, distribute the elements among the splitting elements, and finally output them to the buffers of the relevant children. Since the splitting and buffer elements fit in memory and the buffer elements are distributed to $\sqrt{M/B}$ buffers one level down, the buffer-emptying process is performed in $O(M/B)$ I/Os. Since we distribute $M/2$ elements using $\sqrt{M/B}$ splitters the process can

[1] A priority queue supporting both deletemin and deletemax can easily be obtained using two priority queues supporting deletemin and delete as the one by Thorup [9].

be performed in $O(M)$ time (Lemma 1). After emptying the buffer of the root some of the nodes on the next level may contain more than $M/2$ elements. If they do we perform recursive buffer-emptying processes on these nodes. Note that this way buffers will never contain more than M elements. When (between 1 and $M/2$) elements are pushed down to a leaf (when performing a buffer-emptying process on its parent) resulting in the leaf containing more than M (and less than $3M/2$) elements we split it into two leaves containing between $M/2$ and $3M/4$ elements each. We can easily do so in $O(M/B)$ I/Os and $O(M)$ time [1]. As a result of the split the parent node v gains a child, that is, a new leaf is inserted. If needed, T is then balanced using node splits as a normal B-tree, that is, if the parent node now has $\sqrt{M/B}$ children it is split into two nodes with $1/2\sqrt{M/B}$ children each, while also distributing the elements in v's buffer among the two new nodes. This can easily be accomplished in $O(M/B)$ I/Os and M time. The rebalancing may propagate up along the path to the root (when the root splits a new root with two children is constructed).

During buffer-emptying processes we push $\Theta(M)$ elements one level down the tree using $O(M/B)$ I/Os and $O(M)$ time. Thus each element inserted in the root buffer pays $O(1/B)$ I/Os and $O(1)$ time amortized, or $O(\frac{1}{B}\log_{M/B}\frac{N}{B}) = O(\frac{1}{B}sort_E(N))$ I/Os and $O(\log_{M/B}\frac{N}{B}) = O(sort_E(N))$ time amortized on buffer-emptying processes on a root-leaf path. When a leaf splits we may use $O(M/B)$ I/Os and $O(M)$ time in each node of a leaf-root path of length $O(sort_E(N))$. Amortizing among the at least $M/4$ elements that were inserted in the leaf since it was created, each element is charged and additional $O(\frac{1}{B}sort_E(N))$ I/Os and $O(sort_E(N))$ time on insertion in the root buffer. Since insertion of an element in the root buffer is always triggered by an insertion operation, we can charge the $O(\frac{1}{B}sort_E(N))$ I/Os and $O(sort_E(N))$ time cost to the insertion operation.

Deletemin. To perform a deletemin operation we first check if the mini-queue contains any elements. If it does we simply perform a deletemin operation on it and return the retrieved element using $O(sort_I(M))$ time and no I/Os. Otherwise we perform buffer-emptying processes on all nodes on the leftmost path in T starting at the root and moving towards the leftmost leaf. After this the buffers on the leftmost path are all empty and the smallest elements in the structure are stored in the leftmost leaf. We load the between $M/2$ and M elements in the leaf into main memory, sort them and remove the smallest $M/2$ elements and insert them in the mini-queue in internal memory. If this results in the leaf having less than $M/2$ elements we insert the elements in a sibling and delete the leaf. If the sibling now has more than M elements we split it. As a result of this the parent node v may lose a child. If needed T is then rebalanced using node fusions as a normal B-tree, that is, if v now has $1/2\sqrt{M/B}$ children it is fused with its sibling (possibly followed by a split). As with splits after insertion of a new leaf, the rebalancing may propagate up along the path to the root (when the root only has one leaf left it is removed). Note that no buffer merging is needed since the buffers on the leftmost path are all empty.

If buffer-emptying processes are needed during a deletemin operation we spend $O(\frac{M}{B}\log_{M/B}\frac{N}{B}) = O(\frac{M}{B}sort_E(N))$ I/Os and $O(M\log_{M/B}\frac{N}{B}) = O(M \cdot sort_E(N))$ time on such processes that are not paid by buffers running full (containing more than $M/2$ elements). We also use $O(M/B)$ I/Os and $O(M \cdot sort_I(M))$ time to load and sort the leftmost leaf, and another $O(M \cdot sort_I(M))$ time is used to insert the $M/2$ smallest elements in the mini-queue. Then we may spend (M/B) I/Os and $O(M)$ time on each of at most $O(\log_{M/B}\frac{N}{B})$ nodes on the leftmost path that need to be fused or split. Altogether the filling up of the mini-queue requires $O(\frac{M}{B}sort_E(N))$ I/Os and $O(M\cdot(sort_I(M)+sort_E(N)))$ time. Since we only fill up the mini-queue when $M/2$ deletemin operations have been performed since the last fill up, we can amortize this cost over these $M/2$ deletemin operations such that each deletemin is charged $O(\frac{1}{B}sort_E(N))$ I/Os and $O(sort_E(N) + sort_I(M))$ time.

Theorem 2. *There exists a priority queue supporting an insert operation in $O(\frac{1}{B}sort_E(N))$ I/Os and $O(sort_E(N))$ time amortized and a deletemin operation in $O(\frac{1}{B}sort_E(N))$ I/Os and $O(sort_I(M) + sort_E(N))$ time amortized.*

Remarks. Our priority queue obviously can be used in a simple $O(\frac{N}{B}sort_E(N))$ I/O and $O(N \cdot (sort_I(M) + sort_E(N)))$ time sorting algorithm. Note that it is essential that a buffer-emptying process does not require sorting of the elements in the buffer. In normal buffer-trees [2] such a sorting is indeed performed, mainly to be able to support deletions and (batched) rangesearch operations efficiently. Using a more elaborate buffer-emptying process we can also support deletions without the need for sorting of buffer elements.

4 Lower Bound

Assume that $\frac{N}{B}sort_E(N) = o(N)$ and for simplicity also that B divides N. Recall that under the indivisibility assumption we assume the RAM model in internal memory but require that at any time during an algorithm the original N elements are stored somewhere in memory; we allow copying of the original elements. The internal memory contains at most M elements and the external memory is divided into N *blocks* of B elements each; we only need to consider N blocks, since we are considering algorithms doing less than N I/Os. During an algorithm, we let X denote the set of original elements (including copies) in internal memory and Y_i the set of original elements (including copies) in the i'th block; an I/O transfers *up to* B elements between an Y_i and X. Note that in terms of CPU time, an I/O can cost anywhere between 1 and B (transfers).

In the external memory permuting problem, we are given N elements in the first N/B blocks and want to rearrange them according to a given permutation; since we can always rearrange the elements within the N/B blocks in $O(N/B)$ I/Os, a permutation is simply given as an assignment of elements to blocks (i.e. we ignore the order of the elements within a block). In other words, we start with a distribution of N elements in $X, Y_1, Y_2, \ldots Y_N$ such that $|Y_1| = |Y_2| = \ldots = |Y_{N/B}| = B$ and $X = Y_{(N/B)+1} = Y_{(N/B)+2} = \ldots = Y_N = \emptyset$,

and should produce another given distribution of the same elements such that $|Y_1| = |Y_2| = \ldots = |Y_{N/B}| = B$ and $X = Y_{(N/B)+1} = Y_{(N/B)+2} = \ldots = Y_N = \emptyset$.

To show that any permutation algorithm that performs $O(\frac{N}{B} sort_E(N))$ I/Os has to transfer $\Omega(N \cdot sort_E(N))$ elements between internal and external memory, we first note that at any given time during a permutation algorithm we can identify a distribution (or more) of the original N elements (or copies of them) in $X, Y_1, Y_2, \ldots Y_N$. We then first want to bound the number of distributions that can be created using T I/Os, given that b_i, $1 \leq i \leq T$, is the number of elements transferred in the i'th I/O; any correct permutation algorithm needs to be able to create at least $\frac{N!}{B!^{N/B}} = \Omega((N/B)^N)$ distributions.

Consider the i'th I/O. There are at most N possible choices for the block Y_j involved in the I/O; the I/O either transfers $b_i \leq B$ elements from X to Y_j or from Y_j to X. In the first case there are at most $\binom{M}{b_i}$ ways of choosing the b_i elements, and each element is either moved or copied. In the second case there are at most most $\binom{B}{b_i}$ ways of choosing the elements to move or copy. Thus the I/O can at most increase the number of distributions that can be created by a factor of

$$N \cdot \left(\binom{M}{b_i} + \binom{B}{b_i} \right) \cdot 2^{b_i} < N(2eM/b_i)^{2b_i}.$$

Now the T I/Os can thus at most create $\prod_{i=1}^{T} N(2eM/b_i)^{2b_i}$ distributions. That this number is bounded by $\left(N(2eM/b)^{2b}\right)^T$, where b is the average of the b_i's, can be seen by just considering two values b_1 and b_2 with average b. In this case we have

$$N(2eM/b_1)^{2b_1} \cdot N(2eM/b_2)^{2b_2} \leq \frac{N^2(2eM)^{2(b_1+b_2)}}{b^{2(b_1+b_2)}} \leq \left(N(2eM/b)^{2b}\right)^2.$$

Next we consider the number of distributions that can be created using T I/Os for all possible values of b_i, $1 \leq i \leq T$, with a given average b. This can trivially be bounded by multiplying the above bound by B^T (since this is a bound on the total number of possible sequences $b_1, b_2, \ldots, b_T$). Thus the number of distributions is bounded by $B^T \left(N(2eM/b)^{2b}\right)^T = ((BN)(2eM/b)^{2b})^T$. Since any permutation algorithm needs to be able to create $\Omega((N/B)^N)$ distributions, we get the following lower bound on the number of I/Os $T(b)$ needed by an algorithm that transfers $b \leq B$ elements on the average per I/O:

$$T(b) = \Omega\left(\frac{N \log(N/B)}{\log N + b \log(M/b)}\right).$$

Now $T(B) = \Omega(\min\{N, \frac{N}{B} sort_E(N)\})$ corresponds to the lower bound proved by Aggarwal and Vitter [1]. Thus when $\frac{N}{B} sort_E(N) = o(N)$ we get $T(B) = \Omega(\frac{N}{B} sort_E(N)) = \Omega\left(\frac{N \log(N/B)}{B \log(M/B)}\right)$. Since $1 \leq b \leq B \leq M/2$, we have $T(b) = \omega(T(B))$ for $b = o(B)$. Thus any algorithm performing optimal $O(\frac{N}{B} sort_E(N))$ I/Os must transfer $\Omega(N \cdot sort_E(N))$ elements between internal and external memory.

Reconsider the above analysis under the tall cache assumption $B \leq M^{1-\varepsilon}$ for some constant $\varepsilon > 0$. In this case, we have that the number of distributions any permutation algorithm needs to be able to create is $\Omega((N/B)^N) = \Omega(N^{\varepsilon N})$. Above we proved that with T I/Os transferring an average number of b keys an algorithm can create at most $(BN(2eM/b)^{2b})^T < N^{2T}M^{2bT}$ distributions. Thus we have $M^{2bT} \geq N^{\varepsilon N - 2T}$. For $T < \varepsilon N/4$, we get $M^{2bT} \geq N^{\varepsilon N/2}$ and thus that the number of transferred elements bT is $\Omega(N \log_M N)$. Since the tall cache assumption implies that $\log(N/B) = \Theta(\log N)$ and $\log(M/B) = \Theta(\log M)$ we have that $N \log_M N = \Theta(N \log_{M/B}(N/B)) = \Theta(N \cdot sort_E(N))$. Thus any algorithm using less than $\varepsilon N/4$ I/Os must transfer $\Omega(N \cdot sort_E(N))$ elements between internal and external memory.

Theorem 3. *When $B \leq \frac{1}{2}M$ and $\frac{N}{B}sort_E(N) = o(N)$, any I/O-optimal permuting algorithm must transfer $\Omega(N \cdot sort_E(N))$ elements between internal and external memory under the indivisibility assumption.*

When $B \leq M^{1-\varepsilon}$ for some constant $\varepsilon > 0$ any, permuting algorithm using less than $\varepsilon N/4$ I/Os must transfer $\Omega(N \cdot sort_E(N))$ elements between internal and external memory under the indivisibility assumption.

Remark. The above means that in practice where $\frac{N}{B}sort_E(N) = o(N)$ our $O(\frac{N}{B}sort_E(N))$ I/O and $O(N \cdot (sort_I(N) + sort_E(N))$ time split-sort and priority queue sort algorithms are not only I/O-optimal but also CPU optimal in the sense that their running time is the sum of an unavoidable term and the time used by the best known RAM sorting algorithm.

References

1. Aggarwal, A., Vitter, J.S.: The Input/Output complexity of sorting and related problems. Communications of the ACM 31(9), 1116–1127 (1988)
2. Arge, L.: The buffer tree: A technique for designing batched external data structures. Algorithmica 37(1), 1–24 (2003)
3. Comer, D.: The ubiquitous B-tree. ACM Computing Surveys 11(2), 121–137 (1979)
4. Fadel, R., Jakobsen, K.V., Katajainen, J., Teuhola, J.: Heaps and heapsort on secondary storage. Theoretical Computer Science 220(2), 345–362 (1999)
5. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proc. IEEE Symposium on Foundations of Computer Science, pp. 285–298 (1999)
6. Han, Y.: Deterministic sorting in $O(n \log\log n)$ time and linear space. In: Proc. ACM Symposium on Theory of Computation, pp. 602–608 (2002)
7. Han, Y., Thorup, M.: Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In: Proc. IEEE Symposium on Foundations of Computer Science, pp. 135–144 (2002)
8. Kumar, V., Schwabe, E.: Improved algorithms and data structures for solving graph problems in external memory. In: Proc. IEEE Symp. on Parallel and Distributed Processing, pp. 169–177 (1996)
9. Thorup, M.: Equivalence between priority queues and sorting. Journal of the ACM 54(6), Article 28 (2007)

I/O-Efficient Computation of Water Flow Across a Terrain

Lars Arge*
MADALGO
University of Aarhus
Aarhus, Denmark
large@madalgo.au.dk

Morten Revsbæk*
MADALGO
University of Aarhus
Aarhus, Denmark
mrevs@madalgo.au.dk

Norbert Zeh†
Faculty of Computer Science
Dalhousie University
Halifax, Canada
nzeh@cs.dal.ca

ABSTRACT

Consider rain falling at a uniform rate onto a terrain $\mathcal{T}$ represented as a triangular irregular network. Over time, water collects in the basins of $\mathcal{T}$, forming lakes that spill into adjacent basins. Our goal is to compute, for each terrain vertex, the time this vertex is flooded (covered by water). We present an I/O-efficient algorithm that solves this problem using $O(\text{sort}(X)\log(X/M) + \text{sort}(N))$ I/Os, where N is the number of terrain vertices, X is the number of pits of the terrain, $\text{sort}(N)$ is the cost of sorting N data items, and M is the size of the computer's main memory. Our algorithm assumes that the volumes and watersheds of the basins of $\mathcal{T}$ have been precomputed using existing methods.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms

Algorithms, Theory

Keywords

Terrains, geographical information systems, I/O-efficient algorithms

*Supported in part by the Danish National Research Foundation, the Danish Strategic Research Council, and by the US Army Research office. MADALGO is the Center for Massive Data Algorithmics, a center of the Danish National Research Foundation.

†Supported in part by the Natural Sciences and Engineering Research Council of Canada and the Canada Research Chairs programme. Part of this work was done while on sabbatical at MADALGO.

SCG'10, June 13–16, 2010, Snowbird, Utah, USA.

1. INTRODUCTION

An important problem in terrain analysis is the prediction of water flow across a terrain. Traditional approaches focus on computing the river network of the terrain under the assumption that water does not collect in the basins of the terrain. In reality, water *does* collect in the terrain's basins, particularly during heavy rainfall. This may cause basins to spill into adjacent basins, changing the river network of the terrain as a result. Thus, to model the flow of water across a terrain over time, it is necessary to compute the times at which the basins of the terrain spill. In this paper, we solve the more general problem of computing, for every vertex v of a terrain $\mathcal{T}$, the time t_v at which it is flooded. We assume the terrain $\mathcal{T}$ is represented as a triangular irregular network (TIN) and the amount of rain is uniform across the terrain.

The accuracy of predictions of natural phenomena, such as flooding, depends on the precision of the data used in these predictions. High-resolution elevation models even of fairly small geographic regions often exceed the size of a computer's main memory. Current GIS tools cannot handle data sets of this size efficiently and therefore need to work with models of lower resolutions. This sacrifices accuracy because lower-resolution models do not capture all terrain features. For example, in experiments we conducted to predict the flooding of coastal regions of Denmark due to rising sea levels, an elevation model with one data point every 10m failed to capture a dike around an island, and the island was predicted to be flooded if the sea rises by 2m, while the dike can in fact withstand 2m higher sea levels. Using a higher-resolution model with one data point every 2m, we obtained the correct prediction, but storing elevation data for Denmark alone requires 500GB of space at this resolution, an input size well beyond the size of main memory and beyond the reach of current GIS tools.

To process data sets beyond the size of main memory efficiently, it is necessary to develop I/O-efficient algorithms, that is, algorithms that focus on minimizing the number of disk accesses they perform to swap data between disk and internal memory, as a disk access is orders of magnitude slower than an internal-memory computation step. In this paper, we propose an I/O-efficient algorithm for computing the flooding times of all vertices of a terrain $\mathcal{T}$.

The core of the problem is computing the spill times of all basins of $\mathcal{T}$. A simple method to compute these spill times is to simulate the entire sequence of spill events of the basins of $\mathcal{T}$ and maintain the watersheds of all basins that yet have to spill, as well as predicted spill times for these basins based on their current watersheds. An internal-memory solution

based on this idea has been presented in [14]. In Section 3.2, we discuss an $O(N \log N)$ time implementation of this approach using a priority queue and a union-find structure. This simulation approach does not translate into an I/O-efficient algorithm, as the only known I/O-efficient union-find structure [1] requires the sequence of UNION operations to be known in advance. In simulating the sequence of spill events, the *set* of UNION operations is known, but their order depends on the spill times of the basins. Using an internal-memory union-find structure and an I/O-efficient priority queue (e.g., [3, 12]), a cost of $O(X\alpha(X) + \text{sort}(N))$ disk accesses can be achieved, where $\alpha(\cdot)$ is the inverse of Ackermann's function, $\text{sort}(N) \ll N$ is the I/O complexity of sorting N data items (see next section), and X is the number of pits of the terrain. In contrast, the algorithm presented in this paper achieves an I/O complexity of $O(\text{sort}(X) \log(X/M) + \text{sort}(N))$ disk accesses.

In the remainder of this section, we formally define the computational model we use, discuss previous work, and give an overview of our algorithm. Section 2 introduces the terminology and notation we use throughout the paper. Section 3 discusses our algorithm for computing the spill times of the terrain's basins and the flooding times of all terrain vertices. This algorithm makes use of an I/O-efficient meldable priority queue, which we discuss in Section 4.

1.1 I/O Model

We use the *I/O model* with one (logical) disk [2] to design and analyze our algorithm. In this model, the computer is equipped with a two-level memory hierarchy consisting of an *internal memory* and a (disk-based) *external memory*. The internal memory is capable of holding M data items (vertices, edges, etc.), while the external memory is of conceptually unlimited size. All computation has to happen on data in internal memory. Data is transferred between internal and external memory in blocks of B consecutive data items. Such a transfer is referred to as an *I/O operation* or *I/O*. The cost of an algorithm is the number of I/Os it performs. The number of I/Os required to read N contiguous items from disk is $\lceil N/B \rceil$. The number of I/Os required to sort N items is $\text{sort}(N) = \Theta((N/B) \log_{M/B}(N/B))$ [2]. For all realistic values of N, M, and B, we have $N/B < \text{sort}(N) \ll N$.

1.2 Related Work

Due to its importance, the problem of computing water flow across a terrain (usually in the form of a river network) has been studied extensively. Most existing methods for computing river networks assume that once water flows into a small basin of the terrain, it never flows out—in other words, basins do not spill. Therefore, to avoid water getting caught in small local basins, most flow modelling approaches first remove all basins by *flooding* the terrain, that is, conceptually pouring water onto the terrain until all basins are filled [4, 13, 15]. However, this often leads to unrealistic flow patterns, since many important geographical features are removed. Recent papers [1, 5] suggested *partial flooding* algorithms that flood only "small" basins, where the size of a basin can be defined using different measures, such as height, volume or area. Agarwal, Arge and Yi [1] described an $O(\text{sort}(N))$ I/O partial flooding algorithm that removes all basins of small height. This is done by computing the *topological persistence* [10, 11] of each basin and removing the ones with persistence below a small threshold. The key to obtaining an $O(\text{sort}(N))$ I/O solution to this problem is an algorithm that can process a sequence of N UNION and FIND operations using $O(\text{sort}(N))$ I/Os, assuming the entire sequence of operations is provided in advance. Arge and Revsbæk [5] extended the result of [1] to removing basins based on different geometric measures, including volume and area.

Partial flooding methods provide a basis only for approximate solutions to flow modelling, as the underlying assumption is that all "small" basins are flooded at a certain time, while all "big" basins are not. This assumption may not be true, as the spill time of a basin depends on its volume and its watershed, and the watershed of a basin may grow over time as a result of other basins spilling into it; in particular, the watershed of a big basin may grow to a size far exceeding the size of the watershed of a small basin and, thus, the big basin may spill *before* the small basin. To model the flow network at time t accurately, it is necessary to compute the times the basins of $\mathcal{T}$ fill and remove the basins that are full at time t. This is much harder than the partial flooding approaches discussed above, as the above methods are based on local measures associated with each basin, while the computation of the actual spill time of each basin β depends on the spill times of other basins that spill into β. As mentioned in the introduction, Liu and Snoeyink [14] presented an internal-memory algorithm for computing actual spill times and used it for flow prediction. While the paper does not present the algorithm in detail, we discuss an efficient implementation using a union-find structure and a priority queue in Section 3.2; this implementation is needed as part of our I/O-efficient algorithm.

We also require a tree structure that represents the nesting of the basins of $\mathcal{T}$. This tree has been termed the *merge tree* of $\mathcal{T}$ in [7,8] (also see Section 2) and can be computed using $O(\text{sort}(N))$ I/Os using the topological persistence algorithm of [1]. This algorithm is easily augmented to compute the lowest saddle on the boundary of each basin and, as shown in [5], the volume of each basin.

Computing the watershed sizes of all basins of $\mathcal{T}$ is harder. The watershed sizes of all basins of $\mathcal{T}$ are easily computed from the watershed sizes of all pits by summing watershed sizes bottom up in the merge tree. However, the only I/O-efficient algorithm for computing the watersheds of all pits of a terrain exactly is the one of [9]. This algorithm performs $O(\text{sort}(N + S))$ I/Os for fat terrains, where S is the size of the terrain's strip map. The size of the strip map of a fat terrain is $\Theta(N^2)$ in the worst case [9], in which case the exact computation of watersheds becomes infeasible. An approach often taken in practice [16–18] is to assume that water flows only along terrain edges, allowing the computation of watersheds using $O(\text{sort}(N))$ I/Os. While this may lead to poor approximations of the watersheds for irregular terrains [9], TIN's derived from LIDAR data, for example, are rather regular, and the computed watersheds match the real watersheds rather closely.

1.3 New Result

We present an algorithm that computes the flooding times of all terrain vertices using $O(\text{sort}(X) \log(X/M) + \text{sort}(N))$ I/Os, where N is the size of the terrain and X is the number of pits. This assumes that the watershed sizes and volumes of all basins are given and that every vertex is labelled with the pit whose watershed contains it. The cost of computing

this information has been discussed in the previous section. Our algorithm operates on the merge tree $\mathcal{M}$ of the given terrain $\mathcal{T}$. Given a node β of $\mathcal{M}$, we employ a recursive strategy to compute the spill times of all basins represented by descendants of β. If there are at most M such descendants, we use an augmented version of the internal-memory algorithm mentioned in the introduction to compute the spill times of their corresponding basins using $O(\text{sort}(Y))$ I/Os, where Y is the number of descendants of β plus the number of basins that spill into β. Otherwise we consider an appropriate path P from β to a descendant leaf and prove that we can compute the spill times of all basins whose parents belong to P using $O(\text{sort}(Y))$ I/Os, where Y is defined as above. The path P is chosen so that every subtree attached to P contains at most half of β's descendants, and we invoke our algorithm recursively on the root of each such subtree. This ensures that $\log(X/M)$ levels of recursion suffice to break $\mathcal{M}$ into subtrees of size at most M, to which the above internal-memory algorithm can be applied. The cost per level of recursion is $O(\text{sort}(X))$ I/Os. Hence, the total cost of the algorithm is $O(\text{sort}(X)\log(X/M))$ I/Os. This gives only the spill times of the basins. To compute the flooding times of *all* terrain vertices, we use a post-processing step that can be seen as a simpler version of the recursive step of our algorithm and performs $O(\text{sort}(N))$ I/Os.

The intuition behind the computation in each recursive step is the following. In general, there are two directions water can flow across any saddle of the terrain $\mathcal{T}$. Given the flow direction for each saddle, spill times could be computed rather easily, but the direction for each saddle is determined by the spill times of the two basins that merge at this saddle. For the saddles between the basins corresponding to the path P in each recursive steps, we can characterize each flow direction as either "confluent" (toward the basin represented by the leaf of P) or "diffluent" (away from the leaf), and we can show that confluent spill events that influence other spill events—we call these *watershed events*—can themselves depend only on confluent watershed events. This allows us to perform a sweep in the confluent direction first to compute the times of all confluent watershed events. In a second phase, we sweep in the diffluent direction and use the times computed in the first phase to compute the correct times for *all* spill events.

2. PRELIMINARIES

In this section, we introduce the basic terminology used throughout this paper. In the following, we make some assumptions about the structure of the terrain that simplify the exposition in the rest of the paper. All these assumptions can be removed using standard perturbation techniques.

Terrains, pits, and basins. We consider the terrain $\mathcal{T}$ to be represented as a triangular irregular network, which is a planar triangulation each of whose vertices v has an associated elevation $\mathcal{T}(v)$. The elevation of a point interior to a triangle is a linear interpolation of the elevations of the triangle vertices. In this manner, the terrain is represented as a continuous piecewise linear surface, and we use $\mathcal{T}(p)$ to refer to the elevation of the point $p \in \mathbb{R}^2$. We assume that no two adjacent vertices have the same elevation. A *pit* of $\mathcal{T}$ is a local minimum, that is, a terrain vertex all of whose neighbours have higher elevations. A *saddle point* of $\mathcal{T}$ is a vertex v with four vertices w_1, w_2, w_3, w_4 among its neighbours that satisfy $\max(\mathcal{T}(w_1), \mathcal{T}(w_3)) < \mathcal{T}(v) < \min(\mathcal{T}(w_2), \mathcal{T}(w_4))$ and appear in the order w_1, w_2, w_3, w_4 clockwise around v.

A *basin* is a maximal connected set of points $\beta \subseteq \mathbb{R}^3$ such that $\mathcal{T}((x,y)) \leq z \leq h_\beta$, for all $(x,y,z) \in \beta$, where h_β is a fixed elevation chosen as the upper boundary of β. A basin β is *maximal* if there exists a saddle point p_β on the boundary of β such that $\mathcal{T}(p_\beta) = h_\beta$. The point p_β is called the *spill point* of basin β, since water poured into β spills over p_β into an adjacent basin once β becomes full. We assume p_β is unique, that is, there are no two saddles with elevation h_β on the boundary of β. We also assume exactly two basins meet in each spill point. Throughout this paper, we are interested almost exclusively in maximal basins and refer to them simply as basins. Every basin contains at least one pit, and for every pit there exists a unique (maximal) basin that contains only this pit. We call such a basin *elementary*.

Trickle paths, watersheds, and tributaries. The *trickle path* of a point $q \in \mathcal{T}$ is the path that starts at q, continues in the direction of steepest descent for every point it visits, and ends either in a pit p or at the boundary of $\mathcal{T}$. In the former case, water falling onto q collects in the elementary basin corresponding to p; in the latter, it flows off the edge of the terrain. The *watershed* of a pit p is the set of points whose trickle paths end in p. The watershed W^0_β of a basin β is the union of the watersheds of all pits contained in β. More generally, we use W^t_β to denote the watershed of basin β at time t, which is the area such that water falling onto W^t_β at time t collects in basin β. A *tributary* of β is a basin τ such that $\tau \cap \beta = \emptyset$, τ spills before β, and water falling onto $W^{t_\tau}_\tau$ at the time t_τ when τ spills collects in β; that is, τ spills into β and the watershed of τ at this time becomes part of the watershed of β. Note that τ is usually a tributary of more than one basin. In particular, if β is the smallest basin that has τ as a tributary, then τ is a tributary of every basin that contains β but not τ.

Merge tree and flow tree. The basins of $\mathcal{T}$ form a hierarchy that is easily represented using a rooted tree, the *merge tree* $\mathcal{M}$ of $\mathcal{T}$. The leaves of $\mathcal{M}$ are the elementary basins of $\mathcal{T}$. A basin β_1 is the parent of a basin β_2 if and only if $\beta_2 \subset \beta_1$ and there exists no basin β_3 such that $\beta_2 \subset \beta_3 \subset \beta_1$. Under the assumptions we made about $\mathcal{T}$, $\mathcal{M}$ is a binary tree. For a subset S of nodes of $\mathcal{M}$, let $\mathcal{M}(S)$ be the subgraph of $\mathcal{M}$ induced by S. For a node α of $\mathcal{M}$, $\mathcal{M}_\alpha$ denotes the subtree of $\mathcal{M}$ induced by α and its descendants. We do not distinguish between merge tree nodes and their corresponding basins. For a basin β, we use V_β to denote β's volume, W^t_β to denote β's watershed at time t or its size (which will be clear from the context), E_β to denote the event that β spills into an adjacent basin, and t_β to denote the time of event E_β. We call E_β the *spill event* associated with β. See Figure 1 for an illustration of these definitions.

Now consider a subset S of nodes of $\mathcal{M}$ such that $\mathcal{M}(S)$ is a tree. The flow paths between the basins in S form a *flow tree* $\mathcal{F}(S)$ defined as follows. Let S' be the set of leaves of $\mathcal{M}(S)$. The elements of S' are the vertices of $\mathcal{F}(S)$. There is an edge between two such vertices α_1 and α_2 if their watersheds touch in the common spill point of two basins $\beta_1, \beta_2 \in S$. See Figure 2.

3. COMPUTING FLOODING TIMES

Our algorithm for computing the flooding times of all terrain vertices has two phases. The first phase (Sections 3.1–3.3) computes the spill times t_β of all basins of $\mathcal{T}$ using

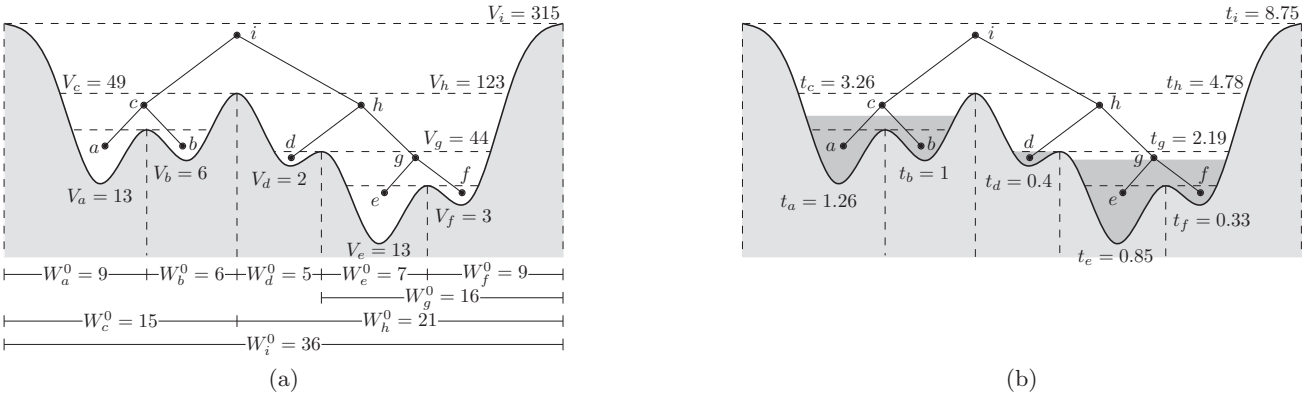


Figure 1: (a) A 1-d terrain with its corresponding merge tree and volumes and initial watersheds of its basins. (b) The spill times of the basins and the water levels in the basins at time $t = 2$.

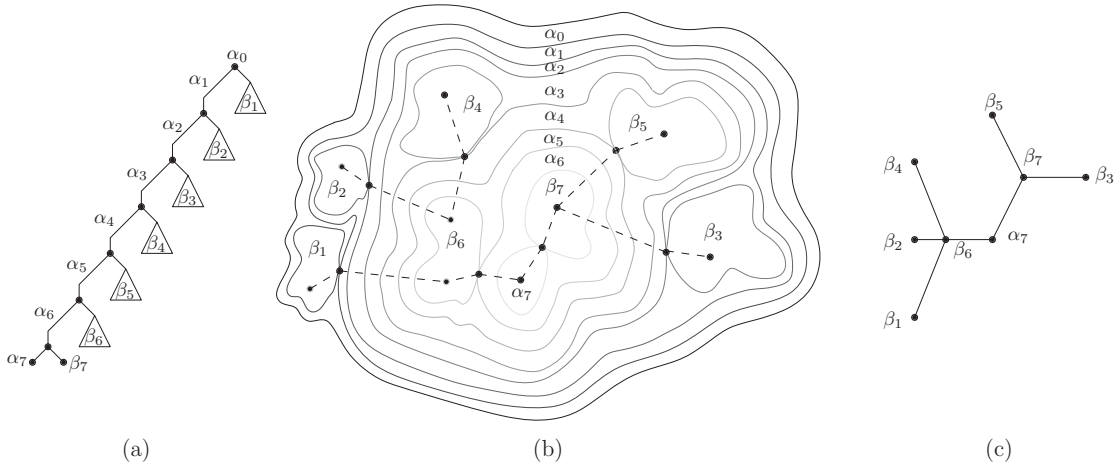


Figure 2: Figure (a) shows the merge tree $\mathcal{M}(S)$ of a set $S = \{\alpha_0, \alpha_1, \beta_1, \ldots, \alpha_7, \beta_7\}$ of basins. Figure (b) illustrates the nesting of these basins using contour lines through their spill points. Figure (c) shows the flow tree of basins $\alpha_7, \beta_1, \beta_2, \ldots, \beta_7$. Every edge corresponds to a spill point and joins the two basins containing the endpoints of the trickle paths starting at this spill point (shown as dashed lines in Figure (b)). Note that the trickle paths corresponding to different edges incident to a flow tree node β_i may end in different pits inside the basin β_i. This is the case for basin β_6 in Figure (b).

$O(\text{sort}(X)\log(X/M))$ I/Os. The second phase (Section 3.4) then computes the flooding times of all vertices of $\mathcal{T}$ using $O(\text{sort}(N))$ I/Os.

3.1 Computing Spill Times

We assume the merge tree $\mathcal{M}$ is given and every node $\beta \in \mathcal{M}$ stores V_β, W_β^0, as well as identifiers of the two elementary basins $\lambda_f(\beta)$ and $\lambda_r(\beta)$ such that $\lambda_f(\beta) \in \mathcal{M}_\beta$ and $W^0_{\lambda_f(\beta)}$ and $W^0_{\lambda_r(\beta)}$ touch in p_β; $\lambda_f(\beta)$ is needed for some data structure queries in our algorithm, while $\lambda_r(\beta)$ is the basin where water spilling from β would collect if we poured water only into β. We further assume the elementary basins of $\mathcal{T}$ have been numbered using a preorder traversal of the flow tree $\mathcal{F}$ of $\mathcal{T}$ starting at an arbitrary root, and every elementary basin stores the preorder interval of its descendants. This information can be computed from the input of our algorithm using $O(\text{sort}(X))$ I/Os using standard techniques. Details appear in the full paper.

Our algorithm for computing the spill times of all basins is recursive. Every recursive call takes a node $\beta \in \mathcal{M}$ and a list $\mathcal{R}_\beta$ of all tributaries of β as input. Each tributary $\tau \in \mathcal{R}_\beta$ stores its spill time t_τ, its watershed $W_\tau := W_\tau^{t_\tau}$ at time t_τ, and the preorder number of an elementary basin it contains. The task of the recursive call on a node β is to compute the spill times of all proper descendants of β in $\mathcal{M}$. The top-level invocation takes the root ρ of $\mathcal{M}$ and an empty list of tributaries as input (since ρ has no tributaries). We distinguish two cases for each recursive call.

If $|\mathcal{M}_\beta| \le M$, we use the algorithm in Section 3.2 below to solve the problem using $O(\text{sort}(X))$ I/Os, where X is the total input size of the invocation, that is, $X := |\mathcal{M}_\beta| + |\mathcal{R}_\beta|$.

If $|\mathcal{M}_\beta| > M$, we compute a *heaviest path* P in $\mathcal{M}_\beta$. This path consists of a sequence of nodes $\alpha_0, \alpha_1, \ldots, \alpha_k$, where $\alpha_0 = \beta$, α_k is a leaf, and, for $1 \le i \le k$, α_i is the child of α_{i-1} with the bigger subtree $\mathcal{M}_{\alpha_i}$ among the two children of α_{i-1}. We use β_i to denote the other child of α_{i-1}. See

Figure 2(a). The first step of the algorithm is to compute the spill times t_{α_i} and t_{β_i} and the set $\mathcal{R}_{\beta_i}$ of tributaries of β_i, for all $1 \leq i \leq k$. To finish the computation of spill times, we recursively invoke the algorithm on each node β_i, $1 \leq i \leq k$. As we show in Section 3.3, the computation of the spill times t_{α_i} and t_{β_i}, for $1 \leq i \leq k$, and of the lists of tributaries of the basins $\beta_1, \beta_2, \ldots, \beta_k$ takes $\mathrm{O}(\mathrm{sort}(k + |\mathcal{R}_\beta|)) = \mathrm{O}(\mathrm{sort}(X))$ I/Os, where $X := |\mathcal{M}_\beta| + |\mathcal{R}_\beta|$. The path P can be computed using $\mathrm{O}(\mathrm{sort}(X))$ I/Os using the Euler tour technique and list ranking [6]. This gives the following result.

THEOREM 1. *The spill times of all basins of a terrain $\mathcal{T}$ can be computed using $\mathrm{O}(\mathrm{sort}(X)\log(X/M))$ I/Os, where X is the number of elementary basins of $\mathcal{T}$.*

PROOF. We prove in Section 3.2 that, given the tributaries of β, we compute the correct spill times for all basins in $\mathcal{M}_\beta$ in the case $|\mathcal{M}_\beta| \leq M$. In Section 3.3, we prove that, in the case $|\mathcal{M}_\beta| > M$, we compute the spill times of basins α_i and β_i and the tributary lists $\mathcal{R}_{\beta_i}$, for $1 \leq i \leq k$, correctly. The correctness of the algorithm then follows by induction. The I/O complexity of an invocation of the algorithm with total input size $X := |\mathcal{M}_\beta| + |\mathcal{R}_\beta|$ and merge tree size $Y := |\mathcal{M}_\beta|$ is given by the recurrence

$$T(X,Y) = \begin{cases} \mathrm{O}(\mathrm{sort}(X)) & Y \leq M \\ \mathrm{O}(\mathrm{sort}(X)) + \sum_{i=1}^{k} T(X_i, Y_i) & Y > M \end{cases},$$

where $Y_i := |\mathcal{M}_{\beta_i}|$ and $X_i := Y_i + |\mathcal{R}_{\beta_i}|$. By the definition of a tributary, every basin τ with $\tau \in \mathcal{M}_\beta$ or $\tau \in \mathcal{R}_\beta$ can be the tributary of at most one basin β_i. Hence, we have $\sum_{i=1}^{k} X_i \leq X$. Furthermore, the choice of the path P as a heaviest path ensures that $Y_i \leq Y/2$, for all $1 \leq i \leq k$. Together, these two facts imply that the recurrence solves to $T(X,Y) = \mathrm{O}(\mathrm{sort}(X)\log(Y/M))$. For the root of $\mathcal{M}$, we have $X = Y$, that is, the overall complexity of the algorithm is $\mathrm{O}(\mathrm{sort}(X)\log(X/M))$, as claimed. □

3.2 Small Basins

To solve the case $|\mathcal{M}_\beta| \leq M$, we provide an implementation of the algorithm of [14] using a union-find structure and a priority queue and extend it to take the tributaries of the basin β into account when computing the spill times of all sub-basins of β. Before running the actual algorithm, we compute, for each tributary τ of β, the elementary sub-basin $\lambda_r(\tau)$ of β that τ spills into; that is, τ spills into β across a saddle on the boundary of $W^0_{\lambda_r(\tau)}$. These basins can be computed from the preorder numbers stored with the tributaries of β and the preorder intervals of the elementary sub-basins of β using $\mathrm{O}(\mathrm{sort}(X))$ I/Os. Details appear in the full paper.

The algorithm maintains a set of *active* basins in $\mathcal{M}_\beta$, which are the basins that have not spilled yet but whose children have already spilled. A basin that has spilled is *finished*, and a basin with at least one unfinished child is *inactive*. The set of active basins always contains the next basin in $\mathcal{M}_\beta$ to spill. We maintain the set of active basins using two data structures: a priority queue Q and a union-find structure U. The priority queue stores the active basins with priorities equal to their predicted spill times—these times decrease over time as we discover more tributaries of active basins. The union-find structure stores the elementary basins (leaves) in $\mathcal{M}_\beta$ and allows us to find, for each such basin α, the active basin α' such that W^0_α is currently part of $W_{\alpha'}$. More precisely, a $\textsc{Find}(\alpha)$ operation returns a representative elementary sub-basin of α', and it is easy to ensure that at all times, the representative of α' stores the ID of α'. Initially, all elementary basins are active and all other basins are inactive; the predicted spill time of an elementary basin α is $t'_\alpha := V_\alpha / W^0_\alpha$. To allow the updating of predicted spill times, each active basin α also stores the time u_α when its watershed changed last—that is, the spill time of its most recent tributary—as well as its residual volume V^r_α at time u_α, which is the portion of V_α left to be filled at this time. Initially, $V^r_\alpha = V_\alpha$ and $u_\alpha = 0$, for every elementary basin α of $\mathcal{T}$.

After initializing the algorithm's data structures as just described, we process the events in Q and $\mathcal{R}_\beta$ by increasing time until Q contains only the basin β. The details of each iteration are as follows. Let τ be the next tributary in $\mathcal{R}_\beta$ to be processed, and let α be the active basin with minimum priority in Q. If $t_\tau < t'_\alpha$, we remove τ from $\mathcal{R}_\beta$ and process E_τ as a *watershed event* with time t_τ, as described below. If $t'_\alpha < t_\tau$, we remove α from Q and consider its sibling σ. If σ is not finished, we process E_α as a *watershed event* with time t'_α; otherwise we process it as a *basin event* with time t'_α.

Watershed event: A watershed event occurs when a basin α spills into a non-full basin α', thereby increasing the watershed of α'. To process a watershed event E_α with time t, we find the active basin α' that α spills into using a $\textsc{Find}(\lambda_r(\alpha))$ operation on U. We update the information for α' as $V^r_{\alpha'} := V^r_{\alpha'} - W_{\alpha'}(t - u_{\alpha'})$, $W_{\alpha'} := W_{\alpha'} + W_\alpha$, $u_{\alpha'} := t$, and $t'_{\alpha'} := t + V^r_{\alpha'}/W_{\alpha'}$, and then update the priority of α' in Q accordingly. If $\alpha \in \mathcal{R}_\beta$, this finishes the processing of E_α. If $\alpha \in \mathcal{M}_\beta$, we need to update U to reflect that water falling or flowing onto W_α after time t flows into α'. We do this by performing a $\textsc{Union}(\lambda_f(\alpha), \lambda_r(\alpha))$ operation on U and ensuring that the representative of the resulting set of elementary basins points to α'. We also label the node α in $\mathcal{M}_\beta$ as finished.

Basin event: A basin event occurs when the second of two sibling basins becomes full, leaving its parent basin in $\mathcal{M}_\beta$ to fill. When processing a basin event E_α with time t, for some $\alpha \in \mathcal{M}_\beta$, both α and its sibling σ in $\mathcal{M}_\beta$ are full at time t. Thus, we label α as finished in $\mathcal{M}_\beta$ and mark its parent γ as active. Water that flowed into α before time t now collects in γ. Thus, we set $W_\gamma := W_\alpha$ and ensure that the representative of α now points to γ. Since both α and σ are full at time t, we set $V^r_\gamma := V_\gamma - V_\alpha - V_\sigma$, $u_\gamma := t$, and $t'_\gamma := t + V^r_\gamma / W_\gamma$. Then we insert γ into Q with priority t'_γ.

The following lemma states the correctness and I/O complexity of the above procedure.

LEMMA 1. *Let β be a basin with at most M sub-basins, and let $X := |\mathcal{M}_\beta| + |\mathcal{R}_\beta|$. The spill times of all basins $\alpha \in \mathcal{M}_\beta$ can be computed using $\mathrm{O}(\mathrm{sort}(X))$ I/Os.*

PROOF. To bound the I/O complexity of the procedure, we observe that Q, U, and $\mathcal{M}_\beta$ fit in memory and that scanning the sorted list of tributaries $\mathcal{R}_\beta$ requires us to keep only one buffer block of size B in memory. Thus, apart from sorting the tributaries in $\mathcal{R}_\beta$ by their spill times using $\mathrm{O}(\mathrm{sort}(X))$ I/Os, the algorithm only loads $\mathcal{M}_\beta$ into memory and scans $\mathcal{R}_\beta$, which takes $\mathrm{O}(X/B)$ I/Os. All other processing happens in memory.

To prove the correctness of the algorithm, we consider the sequence of events $E_1, E_2, \ldots, E_X$ affecting basin β and its sub-basins, sorted by their times $t_1, t_2, \ldots, t_X$. That is, every event E_i is an event E_α with $\alpha \in \mathcal{M}_\beta$ or such that $\alpha \notin \mathcal{M}_\beta$ and α is a tributary of a basin $\alpha' \in \mathcal{M}_\beta$, in which case α is also a tributary of β and belongs to $\mathcal{R}_\beta$.

We use t'_i to denote the "current" predicted time of the event E_i. Consider the basin α such that $E_i = E_\alpha$. Then we define $t'_i := t_i$ if α is a tributary of β and $t'_i := \infty$ if $\alpha \in \mathcal{M}_\beta$ and α is inactive; if $\alpha \in \mathcal{M}_\beta$ and α is active, we define t'_i to be its priority in Q. We use induction on i to prove that $t'_i = t_i$ when event E_i is processed. To do so, we prove that the following holds after processing event E_i:

(i) Events $E_1, E_2, \ldots, E_i$ have been processed. No other events have been processed.

(ii) $W_\alpha = W_\alpha^{t_i}$, for all active basins $\alpha \in \mathcal{M}_\beta$.

(iii) $t'_{i+1} = t_{i+1}$, while $t'_j \geq t_j$, for all $j > i+1$.

The base case is $i = 0$, setting $t_0 = 0$. Property (i) holds because no events have been processed yet. Property (ii) holds because initially $W_\alpha = W_\alpha^0$, for every elementary basin in $\mathcal{M}_\beta$, which is exactly the set of active basins at the beginning of the procedure. To see that (iii) holds, observe that $t'_j = t_j$ if $E_j = E_\tau$, for some tributary τ of β. If $E_j = E_\alpha$, for $\alpha \in \mathcal{M}_\beta$, and α is inactive, then $t'_j = \infty$; if α is active, then $t'_j = V_\alpha / W_\alpha^0 \geq t_j$. Moreover, if $E_1 = E_\alpha$, for some $\alpha \in \mathcal{M}_\beta$, then α has no tributaries and $t_1 = V_\alpha / W_\alpha^0 = t'_1$.

For the inductive step, consider $i > 0$ and assume the claim holds for all $j < i$. By part (i) of the inductive hypothesis, E_{i-1} is the $(i-1)$st event to be processed. By part (iii) of the inductive hypothesis, we have $t'_i = t_i$ and $t'_j \geq t_j > t_i$, for all $j > i$, after processing E_{i-1}. Hence, E_i is the ith event to be processed. This establishes (i).

To prove (ii), we consider the two possible types of event E_i separately. Let $E_i = E_\alpha$. If E_i is a watershed event, part (ii) of the inductive hypothesis implies that we identify the correct watershed $W_{\alpha'}$ into which α spills at time t_i and, hence, that we update $W_{\alpha'}$ to $W_{\alpha'}^{t_i}$; all other watersheds remain unchanged. If E_i is a basin event, α and its sibling are full at time t_i, while α's parent γ has to fill. Furthermore, all water flowing into γ immediately before time t_i collects in α because α's sibling is already full. Therefore, $W_\gamma^{t_i} = W_\alpha^{t_i}$, and we correctly set $W_\gamma := W_\alpha$.

Finally, consider (iii). We consider only events $E_j = E_\alpha$ with α an active basin in $\mathcal{M}_\beta$ because, by definition, $t'_j = t_j$, for every spill event E_j of a tributary of β, and $t'_j = \infty$, for every spill event E_j of an inactive basin in $\mathcal{M}_\beta$. It is easily verified that we update t'_α correctly whenever we increase W_α. By (ii), each event E_h with $h \leq i$ updates W_α if and only if $E_h = E_\tau$, for some tributary τ of α. Thus, $t'_j \geq t_j$, for all $j \geq i$. For $j = i+1$, the spill events of the tributaries of α are a subset of $E_1, E_2, \ldots, E_i$, as they have to happen before $E_\alpha = E_{i+1}$. Thus, we have $t'_{i+1} = t_{i+1}$ after E_i is processed. □

3.3 Basin Sets With Linear Merge Trees

Now consider a path $P = \langle \beta = \alpha_0, \alpha_1, \ldots, \alpha_k \rangle$ from a merge tree node β to a descendant leaf α_k, and let β_i be the sibling of α_i, for all $1 \leq i \leq k$. The basic step of our solution for the case $|\mathcal{M}_\beta| > M$ computes the spill times of $\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_k, \beta_k$, as well as the lists of tributaries $\mathcal{R}_{\beta_1}, \mathcal{R}_{\beta_2}, \ldots, \mathcal{R}_{\beta_k}$ of the basins $\beta_1, \beta_2, \ldots, \beta_k$. Our algorithm for this problem operates on the flow tree $\mathcal{F} := \mathcal{F}(S)$ of the set S of basins in P. Note that every edge in $\mathcal{F}$ corresponds to the spill point connecting two basins α_i and β_i, for $1 \leq i \leq k$, and that $\lambda_f(\alpha_i) = \lambda_r(\beta_i)$ and $\lambda_f(\beta_i) = \lambda_r(\alpha_i)$ in this case. So the tree is easy to construct using a constant number of sorting and scanning steps of S. Details appear in the full paper.

We call a spill event E_{β_i} *confluent*, as the water spilling from β_i at time t_{β_i} spills towards α_k. Similarly, we call a spill event E_{α_i} *diffluent*, as the water spilling from α_i at time t_{α_i} spills away from α_k (α_k is a sub-basin of α_i). We define $\mathcal{F}_{\beta_i}$ to be the subtree of $\mathcal{F}$ rooted in β_i, assuming α_k is chosen as the root of $\mathcal{F}$. Our algorithm proceeds in two phases. The *confluent phase* computes times $t'_{\beta_1}, t'_{\beta_2}, \ldots, t'_{\beta_k}$. These times satisfy $t'_{\beta_i} = t_{\beta_i}$ if E_{β_i} is a watershed event and $t'_{\beta_i} \geq t_{\beta_i}$ if E_{β_i} is a basin event. In addition, this phase constructs a list L_{β_i}, for each $1 \leq i \leq k$, which is a superset of those tributaries of β_i that belong to $\mathcal{F}_{\beta_i}$ or spill into β across a saddle on the boundary of a basin $\beta_j \in \mathcal{F}_{\beta_i}$. The second, *diffluent phase* computes times $t''_{\alpha_1}, t''_{\alpha_2}, \ldots, t''_{\alpha_k}$ and $t''_{\beta_1}, t''_{\beta_2}, \ldots, t''_{\beta_k}$ from the times and potential tributary lists computed in the confluent phase, and we prove that $t''_{\alpha_i} = t_{\alpha_i}$ and $t''_{\beta_i} = t_{\beta_i}$, for all $1 \leq i \leq k$. In the process, the diffluent phase computes the list $\mathcal{R}_{\beta_i}$ of tributaries of β_i, for every $1 \leq i \leq k$.

Intuitively, this two-phase approach works because the confluent phase ensures that the spill times of all basins β_i that are tributaries of other basins are computed correctly (since β_i can be a tributary only if its spill event is a watershed event). A basin α_i can have only confluent tributaries, since basins $\alpha_1, \alpha_2, \ldots, \alpha_k$ are nested. A basin β_i can have only one diffluent tributary, namely α_i, because β_i is a sub-basin of α_j, for all $j < i$, and α_i is a super-basin of α_h, for all $h > i$. Thus, by processing the basins "outwards" from α_k—that is, by decreasing index i—in the diffluent phase, we can ensure that the spill times of all tributaries of a basin are known by the time the basin is processed.

3.3.1 From Tributaries to Spill Times

Both phases of our algorithm use the same elementary procedure to compute a predicted spill time for a basin. To avoid duplication, we describe this procedure here and refer to it as procedure FindSpillTime later. The input of this procedure is a basin α, a time u_α, the residual volume V_α^r of α at time u_α, and the watershed $W_\alpha = W_\alpha^{u_\alpha}$ of α at time u_α. In addition, we are given a priority queue Q containing a set of potential tributaries of α; each entry $\tau \in Q$ satisfies $t'_\tau > u_\alpha$, where t'_τ denotes the priority of τ. The output of the procedure is a predicted spill time t'_α of α and a list L of entries removed from Q while computing t'_α.

Initially, we set $t'_\alpha := u_\alpha + V_\alpha^r / W_\alpha$. Then we repeat the following steps to update t'_α until either Q is empty or the minimum entry τ in Q satisfies $t'_\tau \geq t'_\alpha$. We remove the minimum entry τ (which satisfies $t'_\tau < t'_\alpha$) from Q and append it to L. Then we set $V_\alpha^r := V_\alpha^r - W_\alpha(t'_\tau - u_\alpha)$, $W_\alpha := W_\alpha + W_\tau$, $u_\alpha := t'_\tau$, and $t'_\alpha := u_\alpha + V_\alpha^r / W_\alpha$.

Lemma 2. *Assume at the beginning of procedure* FindSpillTime, *W_α and V_α^r reflect the actual watershed and residual volume of α at time u_α, Q contains only entries τ with $t'_\tau \geq u_\alpha$, and $t'_\tau < t_\alpha$ if and only if τ is a tributary of α. Assume further that every tributary $\tau \in Q$ of α satisfies $t'_\tau = t_\tau$. Then $t'_\alpha \geq t_\alpha$ when the procedure terminates. If Q*

contains every *tributary* τ *of* α *with* $t_\tau \geq u_\alpha$*, then* $t'_\alpha = t_\alpha$*, and* $L = \{\tau \in \mathcal{R}_\alpha \mid t_\tau \geq u_\alpha\}$.

PROOF. First assume for the sake of contradiction that $t'_\alpha < t_\alpha$ when the algorithm terminates. Then the last element $\tau \in Q$ processed by procedure FINDSPILLTIME satisfies $t'_\tau < t'_\alpha$, as the update of t'_α when processing an entry τ does not decrease t'_α below t'_τ. Hence, every processed entry τ satisfies $t'_\tau < t_\alpha$. This implies that all processed elements are tributaries of α that satisfy $u_\alpha < t'_\tau = t_\tau < t_\alpha$, and we process them in order. Thus, after processing every tributary τ, we have $W_\alpha \leq W_\alpha^{t_\tau}$, which implies that $t'_\alpha \geq t_\alpha$, a contradiction.

Now assume Q contains all tributaries of α and $t'_\alpha > t_\alpha$ when procedure FINDSPILLTIME finishes. This is possible only if we do not process all tributaries of α. An unprocessed tributary τ would have to remain in Q by the end of the procedure and, thus, satisfies $t'_\tau > t'_\alpha$. However, $t'_\tau = t_\tau < t_\alpha < t'_\alpha$, again a contradiction.

Finally, observe that, if $t'_\alpha = t_\alpha$, we process exactly the elements $\tau \in Q$ with $t'_\tau < t_\alpha$, and place them into L. These are exactly the tributaries of α that satisfy $t_\tau \geq u_\alpha$. □

3.3.2 Confluent Phase

To implement the confluent phase of the algorithm, we root the flow tree $\mathcal{F}$ in α_k and process its nodes in postorder. With every node $\alpha \in \mathcal{F}$ we associate a list $\mathcal{S}_\alpha \subseteq \mathcal{R}_\beta$ containing all tributaries τ of β that spill into β across a saddle on the boundary of W_α^0. These lists are easy to compute using $O(\text{sort}(X))$ I/Os from the preorder numbers of the elementary basins associated with the tributaries, assuming that every basin in $\mathcal{M}$ stores the largest preorder interval of all its elementary sub-basins, which is easily computed in a preprocessing step using a bottom-up traversal in $\mathcal{M}$. Details appear in the full paper.

During the traversal of $\mathcal{F}$, we ensure that visiting a node β_i produces a priority queue Q_{β_i} that contains all basins τ that satisfy (i) $\tau \in \{\beta_j\} \cup \mathcal{S}_{\beta_j}$, for some $\beta_j \in \mathcal{F}_{\beta_i}$, and (ii) $t'_\tau \geq t'_{\beta_h}$, for all β_h on the path from β_j to β_i in $\mathcal{F}$, inclusive. At any point during the postorder traversal, there is a set of *active* nodes, which are nodes that have been visited already but whose parents in $\mathcal{F}$ have not. We maintain the priority queues of all active nodes in a sequence sorted in the order these nodes are visited and represent this sequence of priority queues using a data structure that supports INSERT and DELETEMIN operations on the last priority queue in the sequence, the creation of a new priority queue at the end of the sequence, as well as a MELD operation, which replaces the last two priority queues in the sequence with their union. In Section 4, we show that the external heap of Fadel et al. [12] can be extended to process a sequence of N INSERT, DELETEMIN, CREATE, and MELD operations using $O(\text{sort}(N))$ I/Os.

The processing of a node β_i distinguishes two cases. If β_i is a leaf, we create a new priority queue Q_{β_i} at the end of the sequence of priority queues of active nodes. If β_i is an internal node with children $\beta_{j_1}, \beta_{j_2}, \ldots, \beta_{j_h}$, the corresponding priority queues $Q_{\beta_{j_1}}, Q_{\beta_{j_2}}, \ldots, Q_{\beta_{j_h}}$ are at the end of the current sequence of priority queues, and we construct Q_{β_i} by melding these priority queues. In both cases, we continue by inserting all elements $\tau \in \mathcal{S}_{\beta_i}$ into Q_{β_i}, with priority $t'_\tau = t_\tau$. Then we use procedure FINDSPILLTIME to compute t'_{β_i} and L_{β_i}; the input to the procedure is β_i, Q_{β_i}, $u_{\beta_i} := 0$, $W_{\beta_i} := W^0_{\beta_i}$, and $V^r_{\beta_i} := V_{\beta_i}$. Once this procedure terminates, we insert β_i into Q_{β_i}, with priority t'_{β_i}.

The confluent phase terminates when the traversal reaches the root α_k of $\mathcal{F}$. Since E_{α_k} is a diffluent spill event, we do not compute its time in this phase of the algorithm. We only construct a list L_{α_k} by collecting all elements in $Q_{\beta_{j_1}}, Q_{\beta_{j_2}}, \ldots, Q_{\beta_{j_h}}$ and $\mathcal{S}_{\alpha_k}$, where $\beta_{j_1}, \beta_{j_2}, \ldots, \beta_{j_h}$ are the children of α_k in $\mathcal{F}$.

To establish the correctness of the confluent phase, we require a number of technical lemmas. The first one characterizes the spill events of basins on the flow path between a basin and its tributary. For three basins $\beta_j, \beta_h \in \mathcal{F}$ and a basin $\alpha \in \{\alpha_i, \beta_i\}$, we say β_h is on the path from β_j to α in $\mathcal{F}$ if β_h is not a sub-basin of α but belongs to the path from β_j to every sub-basin $\alpha' \in \mathcal{F}$ of α.

LEMMA 3. *Consider a tributary* τ *of a basin* $\alpha \in \{\alpha_i, \beta_i\}$*, and assume that* $\tau \in \{\beta_j\} \cup \mathcal{S}_{\beta_j}$*, for some* j*. Then* E_{β_h} *is a watershed event, for every basin* β_h *on the path from* β_j *to* α *in* $\mathcal{F}$ *that is not a sub-basin of* α_i.

PROOF. For $j > i$, the lemma holds vacuously because β_j is a sub-basin of α_i in this case and $\alpha \in \{\alpha_i, \beta_i\}$; this implies that every basin β_h on the path from β_j to α is a sub-basin of α_i. For $j \leq i$, assume there exists a basin β_h on the path from β_j to α that is not a sub-basin of α_i and such that E_{β_h} is a basin event. Since β_h is not a sub-basin of α_i, we have $h \leq i$. If $\alpha = \alpha_i$, this implies that α is a sub-basin of α_h and $t_\alpha \leq t_{\alpha_h} \leq t_{\beta_h}$. If $\alpha = \beta_i$, then $h < i$ because $\beta_h \neq \alpha$. Thus, α is again a sub-basin of α_h; in particular, $t_\alpha \leq t_{\beta_h}$. However, since β_j is a tributary of α, we have $t_{\beta_j} < t_\alpha$ and, therefore, $t_{\beta_j} < t_{\beta_h}$ in both cases. For β_j to be a tributary of α, there has to exist a down-hill path from β_j to α at time t_{β_j}. By the choice of β_h, any path from β_j to α has to pass through a point interior to β_h first and then through p_{β_h}. Since β_h is not full at time t_{β_j}, no such path is down-hill at time t_{β_j}, a contradiction. □

Our next lemma shows that, if a basin $\tau \in \{\beta_j\} \cup \mathcal{R}_{\beta_j}$ is a tributary of a basin β_i with $\beta_j \in \mathcal{F}_{\beta_i}$, then $\tau \in L_{\beta_i}$, that is, τ is processed when computing t'_{β_i}; otherwise, if it is a tributary of $\alpha \in \{\alpha_i, \beta_i\}$, then it is in L_γ, for some sub-basin γ of α_i. The first part is used to prove in Lemma 5 below that the confluent phase computes the correct spill times for all confluent watershed events. The second part is used to establish the correctness of the diffluent phase, discussed in Section 3.3.3.

LEMMA 4. *Consider a basin* $\tau \in \{\beta_j\} \cup \mathcal{S}_{\beta_j}$ *with* $\tau \in L_\gamma$*. Assume further that every basin* β_h *in* $\mathcal{F}_\gamma$ *satisfies* $t'_{\beta_h} \geq t_{\beta_h}$*, with equality if* E_{β_h} *is a watershed event. If* τ *is a tributary of* β_i *and* $\beta_j \in \mathcal{F}_{\beta_i}$*, then* $\gamma = \beta_i$*. Otherwise, if* τ *is a tributary of a basin* $\alpha \in \{\alpha_i, \beta_i\}$*, then* γ *is a sub-basin of* α_i.

PROOF. First assume τ is a tributary of β_i and $\beta_j \in \mathcal{F}_{\beta_i}$. Then E_τ is a watershed event and either $\tau \in \mathcal{S}_{\beta_j}$ or $\tau = \beta_j$. In both cases, we have $t'_\tau = t_\tau$, in the latter case by the assumption that $t'_{\beta_h} = t_{\beta_h}$ for every watershed event on the path from β_j to γ. If γ belongs to the path from β_j to β_i and $\gamma \neq \beta_i$, then we have $t'_\gamma = t_\gamma < t_\tau$ because τ is a tributary of β_i and, hence, E_γ is a watershed event. However, $\gamma \neq \alpha_k$ in this case, and every element τ in L_γ is removed from Q_γ while computing t'_γ. Thus, $t'_\tau < t'_\gamma$, a contradiction. This shows that γ is an ancestor of β_i in $\mathcal{F}$. This, however, implies that $t'_{\beta_i} \geq t_{\beta_i} > t_\tau = t'_\tau$. Thus, the computation of t'_{β_i} removes τ from Q_{β_i}, and $\tau \in L_{\beta_i}$.

Now assume that τ is a tributary of a basin $\alpha \in \{\alpha_i, \beta_i\}$ and, if $\alpha = \beta_i$, that $\beta_j \notin \mathcal{F}_{\beta_i}$. Then, if γ belongs to the path from β_j to α_i, we obtain $t'_\gamma = t_\gamma < t_\tau = t'_\tau$ on the one hand, because τ is a tributary of α and, by Lemma 3, E_γ is a watershed event; on the other hand, $t'_\gamma > t'_\tau$ because otherwise $\tau \notin L_\gamma$. Thus, we obtain a contradiction and γ is a sub-basin of α_i. □

LEMMA 5. *For all $1 \le i \le k$, we have $t'_{\beta_i} \ge t_{\beta_i}$. If E_{β_i} is a watershed event, equality holds.*

PROOF. By induction on $|\mathcal{F}_{\beta_i}|$. The base case is $|\mathcal{F}_{\beta_i}| = 0$, in which case there is nothing to prove. So consider a node β_i and assume the claim holds for all its descendants. Then, by Lemma 4, every tributary $\tau \in \{\beta_j\} \cup \mathcal{S}_{\beta_j}$, for some descendant β_j of β_i, is inspected when computing t'_{β_i}, and each such tributary satisfies $t'_\tau = t_\tau$ by the inductive hypothesis. In particular, each such tributary belongs to Q_{β_i} when invoking procedure FINDSPILLTIME to compute t'_{β_i}. We prove that any other element $\alpha \in Q_{\beta_i}$ satisfies $t'_\alpha \ge t_{\beta_i}$. By Lemma 2, this implies that $t'_{\beta_i} \ge t_{\beta_i}$, with $t'_{\beta_i} = t_{\beta_i}$ if Q_{β_i} contains *all* tributaries of β_i, which is the case if E_{β_i} is a watershed event. Indeed if there was a tributary τ such that $\tau \in \{\beta_j\} \cup \mathcal{R}_{\beta_j}$, for some $\beta_j \notin \mathcal{F}_{\beta_i}$, the water spilling from τ into β_i would have to flow through a point interior to α_i and then through $p_{\alpha_i} = p_{\beta_i}$. Since $t_\tau < t_{\beta_i}$ and E_{β_i} is a watershed event, α_i is not full at time t_τ, that is, the path from τ to β_i cannot be down-hill at time t_τ, a contradiction.

So consider an element $\alpha \in Q_{\beta_i}$, and assume that $\alpha \in \{\beta_j\} \cup \mathcal{S}_{\beta_j}$, for some $\beta_j \in \mathcal{F}_{\beta_i}$, and $t'_\alpha < t_{\beta_i}$. We prove that α is a tributary of β_i. We have $t'_\alpha \ge t'_{\beta_h}$, for all β_h on the path from β_j to β_i, excluding β_i, because otherwise β_h would have removed α from Q_{β_h} when computing t'_{β_h}. Since, by the inductive hypothesis, $t_{\beta_h} \le t'_{\beta_h}$, this implies that $t_{\beta_h} < t_{\beta_i}$. Also by the inductive hypothesis, we have $t'_\alpha \ge t_\alpha$. If $t_\alpha \ge t_{\beta_h}$, for all β_h on the path from β_j to β_i, then α is a tributary of β_i. If $t_\alpha < t_{\beta_h} \le t'_\alpha$, for some β_h, then, by the inductive hypothesis, E_α is not a watershed event. In particular, $\alpha = \beta_j$, for some $j < i$, and $t_{\beta_i} < t_{\alpha_j} \le t_{\beta_j} = t_\alpha$ because β_i is a sub-basin of α_j. This contradicts our assumption that $t_\alpha \le t'_\alpha < t_{\beta_i}$. □

3.3.3 Diffluent Phase

In the diffluent phase, we process the basins in $\mathcal{M}_\beta$ in the order $\alpha_k, \beta_k, \alpha_{k-1}, \beta_{k-1}, \ldots, \alpha_1, \beta_1$. We initialize a priority queue Q to contain all events in L_{α_k}. Then we repeat the following steps for $i = k, k-1, \ldots, 1$:

First we compute t''_{α_i}. If $i = k$, we do this by invoking procedure FINDSPILLTIME with arguments α_k, Q, $u_{\alpha_k} := 0$, $W_{\alpha_k} := W^0_{\alpha_k}$, and $V^r_{\alpha_k} := V_{\alpha_k}$. If $i < k$, we invoke the procedure with arguments α_i, Q, $u_{\alpha_i} := \max(t''_{\alpha_{i+1}}, t''_{\beta_{i+1}})$, $W_{\alpha_i} := \max(W_{\alpha_{i+1}}, W_{\beta_{i+1}})$, and $V^r_{\alpha_i} := V_{\alpha_i} - V_{\alpha_{i+1}} - V_{\beta_{i+1}}$. We place the elements of Q processed by procedure FINDSPILLTIME into a list $\mathcal{R}''_{\alpha_i}$.

Then we compute t''_{β_i}. To do so, we insert α_i (with priority t''_{α_i}) and the events in L_{β_i} into Q and invoke procedure FINDSPILLTIME with arguments β_i, Q, $u_{\beta_i} := 0$, $W_{\beta_i} := W^0_{\beta_i}$, and $V^r_{\beta_i} := V_{\beta_i}$. We place the elements of Q processed by procedure FINDSPILLTIME into a list $\mathcal{R}''_{\beta_i}$.

The computation of t''_{α_i} and t''_{β_i} may not process all elements in Q, and some of these elements may be sub-basins of α_i. These elements should be ignored in subsequent computations, as they cannot be tributaries of any α_j or β_j with $j < i$. To ensure this, we augment the above procedure to ignore in iteration i all elements α_j or β_j in Q with $j > i$. (Note that we ignore only these basins, not the elements of $\mathcal{S}_{\alpha_k}$ or $\mathcal{S}_{\beta_j}$ with $j > i$.)

LEMMA 6. *For all $1 \le i \le k$, we have $t_{\alpha_i} = t''_{\alpha_i}$, $t_{\beta_i} = t''_{\beta_i}$, $\mathcal{R}_{\beta_i} = \mathcal{R}''_{\beta_i}$, and $\mathcal{R}_{\alpha_i} = \mathcal{R}''_{\alpha_k} \cup \bigcup_{j=i}^{k-1} \mathcal{R}''_{\beta_j}$.*

PROOF. By induction on i. For $i = k$, Lemmas 4 and 5 imply that every tributary τ of α_k belongs to L_{α_k} and satisfies $t'_\tau = t_\tau$. Any other basin $\alpha \in L_{\alpha_k}$ satisfies $t'_\alpha \ge t_\alpha$, and the same arguments as in the proof of Lemma 5 show that $t'_\alpha \ge t_{\alpha_k}$ in this case. Hence, by Lemma 2, the first iteration computes $t''_{\alpha_k} = t_{\alpha_k}$ and processes only tributaries of α_k, that is, $\mathcal{R}''_{\alpha_k} = \mathcal{R}_{\alpha_k}$.

Now consider β_k. By Lemmas 4 and 5, every tributary of β_k belongs to $\{\alpha_k\} \cup L_{\alpha_k} \cup L_{\beta_k}$, and we have just argued that the computation of α_k processes only tributaries of α_k. Hence, Q contains all tributaries of β_k. We have just argued that $t''_{\alpha_k} = t_{\alpha_k}$ and, by Lemma 5, any other tributary τ of β_k satisfies $t'_\tau = t_\tau$. For an element $\alpha \in Q$ that is not a tributary of β_k, the same arguments as in the proof of Lemma 5 show that $t'_\alpha \ge t_{\beta_k}$. (However, we have to distinguish the cases $\beta_j \in \mathcal{F}_{\beta_k}$ and $\beta_j \notin \mathcal{F}_{\beta_k}$, where $\alpha \in \{\beta_j\} \cup \mathcal{R}_{\beta_j}$.) Thus, by Lemma 2, we compute $t''_{\beta_k} = t_{\beta_k}$ and $\mathcal{R}''_{\beta_k} = \mathcal{R}_{\beta_k}$.

Now assume that $i < k$ and that the inductive hypothesis holds for all $j > i$. By Lemmas 4 and 5, every tributary τ of α_i is contained in $L := L_{\alpha_k} \cup \bigcup_{j=i+1}^{k} L_{\beta_j} = Q \cup \bigcup_{j=i+1}^{k} (\mathcal{R}''_{\alpha_j} \cup \mathcal{R}''_{\beta_j})$ and satisfies $t'_\tau = t_\tau$. Using the arguments from the proof of Lemma 5 again, every $\alpha \in L$ that is not a tributary of α_i satisfies $t'_\alpha \ge t_{\alpha_i}$. Moreover, the elements of $L \setminus Q$ are exactly the tributaries of α_{i+1} and β_{i+1}, that is, the tributaries τ of α_i that satisfy $t_\tau < u_{\alpha_i}$. Hence, by Lemma 2, we compute $t''_{\alpha_i} = t_{\alpha_i}$ and $\mathcal{R}''_{\alpha_i}$ to contain exactly the tributaries τ of α_i that satisfy $t_\tau \ge u_{\alpha_i}$. Thus, $\mathcal{R}_{\alpha_i} = \mathcal{R}''_{\alpha_i} \cup \bigcup_{j=i+1}^{k} (\mathcal{R}''_{\alpha_j} \cup \mathcal{R}''_{\beta_j})$. The correctness proof for the computation of t''_{β_i} and $\mathcal{R}''_{\beta_i}$ is identical to that for β_k. □

LEMMA 7. *The computation of the spill times t_{α_i} and t_{β_i} and of the tributary lists $\mathcal{R}_{\beta_i}$, for all $1 \le i \le k$, takes $O(\mathrm{sort}(X))$ I/Os.*

PROOF. The correctness of the algorithm is established by Lemma 6. Its I/O-complexity is established as follows: Both phases of the algorithm perform $O(X)$ priority queue operations, as every event is inserted and removed from a priority queue only once in each of the two phases and the confluent phase performs at most X MELD operations. By Theorem 3 in Section 4, these priority queue operations cost $O(\mathrm{sort}(X))$ I/Os. Apart from this, the algorithm traverses trees $\mathcal{F}$ and $\mathcal{M}(S)$ once each and scans lists associated with the tree nodes of total length $O(X)$. After arranging the tree nodes and list elements in the right order, using a sorting step, the scanning of these lists takes $O(X/B)$ I/Os. □

3.4 Computing the Flooding Times of All Terrain Vertices

The terrain vertices that are not spill points of basins can be seen as partitioning the basins of $\mathcal{T}$ into sub-basins as follows. For each terrain vertex v, let α_v be the smallest basin that contains v. For a basin β, let $V(\beta)$ be the set of terrain vertices with $\alpha_v = \beta$. Each vertex $v \in V(\beta)$ defines a sub-basin $\beta_{\mathcal{T}(v)}$ of β containing the portion of β below elevation $\mathcal{T}(v)$. Let $v_1, v_2, \ldots, v_k$ be the vertices in $V(\beta)$ sorted by

increasing elevation. Then the water from every tributary in $\mathcal{R}''_{\beta_i}$ first collects in $\beta_{\mathcal{T}(v_1)}$; once $\beta_{\mathcal{T}(v_1)}$ is full, the water collects in $\beta_{\mathcal{T}(v_2)}$, and so on. This is just a simplified version of the diffluent flow computation in Section 3.3.3, and computing the spill times of basins $\beta_{\mathcal{T}(v_1)}, \beta_{\mathcal{T}(v_2)}, \ldots, \beta_{\mathcal{T}(v_k)}$—that is, the flooding times of vertices $v_1, v_2, \ldots, v_k$—from the list $\mathcal{R}''_\beta$ takes $\mathrm{O}(\mathrm{sort}(N))$ I/Os in total for all basins of $\mathcal{T}$. The computation of the sets $V(\beta)$ can be incorporated in the computation of the basins of $\mathcal{T}$ without increasing the I/O complexity of that phase. Details appear in the full paper. Thus, we have the following result.

THEOREM 2. *The flooding times of all vertices of a terrain $\mathcal{T}$ with N vertices and X pits can be computed using $\mathrm{O}(\mathrm{sort}(X)\log(X/M) + \mathrm{sort}(N))$ I/Os, given the watersheds and volumes of all basins of $\mathcal{T}$.*

4. A MELDABLE PRIORITY QUEUE

In this section, we discuss how to extend the external heap of Fadel et al. [12] to obtain a meldable priority queue that can be used in the procedure in Section 3.3.2. This structure maintains a sequence of priority queues $Q_1, Q_2, \ldots, Q_k$ under CREATE, INSERT, DELETEMIN, and MELD operations. Given the current sequence $Q_1, Q_2, \ldots, Q_k$, a CREATE operation creates a new, empty priority queue Q_{k+1} and appends it to the end of the sequence; an INSERT(x) operation inserts element x into Q_k; a DELETEMIN operation deletes and returns the minimum element in Q_k; a MELD operation replaces Q_{k-1} and Q_k with a new priority queue $Q'_{k-1} := Q_{k-1} \cup Q_k$ containing the elements from Q_{k-1} and Q_k. We prove the following result.

THEOREM 3. *There exists a linear-space data structure that uses $\mathrm{O}(\mathrm{sort}(N))$ I/Os to process a sequence of N* CREATE, INSERT, DELETEMIN, *and* MELD *operations.*

4.1 The Structure

Each priority queue Q_i is represented as an *external heap* similar to the one presented in [12]. The underlying structure is a rooted tree whose internal nodes have between $a = M/(4B)$ and $b = M/B$ children. There are two types of leaves: *regular* leaves are on the lowest level of the tree, which we call the *leaf level*; leaves above the leaf level are *special*.

Every node v has an associated buffer X_v. If v is an internal node, X_v has size M. If v is a leaf, there is no bound on the size of X_v. The elements in X_v are sorted, and the buffer contents of adjacent nodes satisfy the *heap property*: for every node v with parent u and every pair of elements $x \in X_u$ and $y \in X_v$, we have $x \leq y$.

To support INSERT and DELETEMIN operations efficiently, every priority queue Q_i is equipped with an *insert/delete buffer* or *I/D buffer* for short. This buffer is capable of holding up to M elements and stores the minimum elements in Q_i as well as newly inserted elements. To distinguish newly inserted elements in Q_i's I/D buffer from minimum elements, Q_i has an associated priority p_i, which is the minimum priority of the elements stored in the external part of Q_i.

The I/D buffers of priority queues $Q_1, Q_2, \ldots, Q_k$ are kept on a *buffer stack*, with the buffer of Q_1 at the bottom and the buffer of Q_k at the top. The I/D buffers of consecutive priority queues are separated by special *marker elements*. We always keep the topmost M elements of this stack in memory, which ensures in particular that the I/D buffer of Q_k is in memory.

4.2 Operations

Create. To create a new priority queue Q_{k+1}, a CREATE operation pushes a new marker element onto the buffer stack.

Insert. An INSERT operation on Q_k adds the inserted element x to Q_k's I/D buffer. If the buffer now contains M elements, we FLUSH the buffer as described below. Then we FILL the I/D buffer with the $M/2$ minimum elements in Q_k and update p_k accordingly.

DeleteMin. A DELETEMIN operation on Q_k returns the minimum element in Q_k's I/D buffer. Before doing so, however, it checks whether the I/D buffer of Q_k is empty or its minimum element is greater than p_k. If so, it FLUSHes the I/D buffer and then FILLs it with the $M/2$ minimum elements in Q_k.

Meld. A MELD operation behaves differently depending on the structure of the two involved priority queues, Q_{k-1} and Q_k. During different stages of its life time, a priority queue may have no external portion because all its elements fit in the I/D buffer. If at least one of the two priority queues, say Q_k, has no external portion, we destroy its I/D buffer, and INSERT its elements into Q_{k-1}. If both priority queues have external portions, we FLUSH and destroy their I/D buffers, MERGE their two trees as described below, and then FILL the I/D buffer of the merged priority queue $Q'_{k-1} := Q_{k-1} \cup Q_k$ with the $M/2$ minimum elements in Q'_{k-1}.

Flush. A FLUSH operation of an I/D buffer containing K elements creates a new leaf l at the leaf level and stores these K elements in l. Then it applies a HEAPIFY operation to the path from l to the root to restore the heap property. If the addition of l increased the degree of l's parent p to $M/B + 1$, we split p into two nodes p' and p'', each with half of p's children. We associate the buffer of p with p' and populate the buffer of p'' with the M smallest elements in its subtree using a FILL operation. If this split increases the degree of p's parent to $M/B + 1$, we apply this rebalancing procedure recursively until we reach the root. If the root has degree greater than M/B, we split it into two nodes with a new parent.

Fill. A FILL operation applied to a node v repeatedly takes the smallest element stored in the buffers of v's children, removes it from the corresponding child's buffer and stores it in v's buffer. This continues until M elements have been collected in v's buffer or there are no elements left in the buffers of v's children. Whenever a child buffer runs empty while filling v's buffer, its buffer is filled recursively before continuing to fill v's buffer. Once there are no more elements left in a node v's subtree, we mark v as *exhausted*, in order to avoid repeatedly trying to fill v with elements. More precisely, a node is marked as exhausted once its buffer becomes empty and all its children are exhausted. To fill the I/D buffer of a priority queue Q_k with the $M/2$ smallest elements in Q_k, we transfer the $M/2$ smallest elements from the root of Q_k into the I/D buffer, FILLing the root's buffer whenever it runs empty. If the root becomes exhausted in the process, we delete the entire external portion of the priority queue.

Merge. A MERGE operation between two trees T_1 and T_2 of heights $h_1 > h_2$ proceeds as follows. We locate a node v at height $h_2 + 1$ in T_1 and make the root r_2 of T_2 a child of v. We also create a new special leaf node l that is a child of v. Now let $v = v_0, v_1, \ldots, v_h$ be the ancestors of v, by increasing distance from v, let K_i be the number of elements in X_{v_i}, and let $K = \sum_{i=0}^{h} K_i$. First we collect the elements in $X_{v_0}, X_{v_1}, \ldots, X_{v_h}$ in sorted order and store them in l. Then we repeatedly remove the minimum element from X_{r_2} and X_l, refilling X_{r_2} as necessary, until we have collected the K smallest elements in the subtrees rooted at r_2 and l. We store these elements in buffers $X_{v_0}, X_{v_1}, \ldots, X_{v_h}$, sorted top-down and so that each node v_i receives K_i elements. We clear the "exhausted" labels of all nodes among $v_0, v_1, \ldots, v_h$. If v has degree greater than $M/2$ as a result of gaining two children, r_2 and l, we rebalance the tree using node splits starting at v similar to the rebalancing done after a FLUSH operation.

Heapify. The final operation to discuss is the HEAPIFY operation. Given a node v and its path $v = v_0, v_1, \ldots, v_h$ to the root, a HEAPIFY operation ensures that the elements stored on the path satisfy the heap property. It assumes that the elements in $v_1, v_2, \ldots, v_h$ already do so, but some elements in v_0 may be less than elements stored at higher nodes. To restore the heap property, we sort the elements in v_0 and collect the elements in $v_1, v_2, \ldots, v_h$ in sorted order. Then we merge the two sorted sequences to obtain a sorted sequence of the elements stored in $v_0, v_1, \ldots, v_h$ and distribute these elements over the buffers of nodes $v_0, v_1, \ldots, v_h$, assigning the same number of elements to each node as it had before the operation and storing the elements sorted top-down on the path. If some of the nodes on the path were marked as exhausted before this operation, we unmark them.

4.3 Analysis

To prove the correctness of all priority queue operations, we need to verify that the heap property is maintained at all times. This is little more than an exercise and is therefore omitted.

It is easy to see that every CREATE, INSERT, DELETEMIN, and MELD operation makes only $O(1)$ changes to the buffer stack and thus has an amortized cost of $O(1/B)$ I/Os, excluding the manipulations performed by the FLUSH, FILL, MERGE, and HEAPIFY operations they trigger. The following two lemmas bound the cost of all FLUSH, FILL, MERGE, and HEAPIFY operations performed during a sequence of N priority queue operations. Due to lack of space, their proofs are omitted.

LEMMA 8. *During a sequence of N priority queue operations, at most $O(N/M)$* FLUSH, FILL, MERGE, *and* HEAPIFY *operations are performed.*

LEMMA 9. *The amortized cost per* FLUSH, FILL, MERGE *or* HEAPIFY *operation is $O((M/B)\log_{M/B}(N/M))$ I/Os.*

5. REFERENCES

[1] P. Agarwal, L. Arge, and K. Yi. I/O-efficient batched union-find and its applications to terrain analysis. In *Proc. 22nd SCG*, pp. 167–176, 2006.

[2] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Comm. of the ACM*, 31(9):1116–1127, 1988.

[3] L. Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003.

[4] L. Arge, J. Chase, P. Halpin, L. Toma, D. Urban, J. S. Vitter, and R. W. and. Flow computation on massive grid terrains. In *Proc. 9th ACM GIS*, pp. 82–87, 2001.

[5] L. Arge and M. Revsbæk. I/O-efficient contour tree simplification. In *Proc. 20th ISAAC*, pp. 1155–1165, 2009.

[6] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter. External-memory graph algorithms. In *Proc. 6th SODA*, pp. 139–149, 1995.

[7] A. Danner. *I/O Efficient Algorithms and Applications in Geographic Information Systems*. PhD thesis, Dept. of Comp. Sci., Duke University, 2006.

[8] A. Danner, K. Yi, T. Mølhave, P. K. Agarwal, L. Arge, and H. Mitasova. TerraStream: From elevation data to watershed hierarchies. Manuscript, 2007.

[9] M. de Berg, O. Cheong, H. Haverkort, J.-G. Lim, and L. Toma. I/O-efficient flow modelling on fat terrains. In *Proc. 10th WADS*, pp. 239–250, 2007.

[10] H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical morse complexes for piecewise linear 2-manifolds. In *Proc. 17th SCG*, pp. 70–79, 2001.

[11] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. In *Proc. 41st FOCS*, pp. 454–463, 2000.

[12] R. Fadel, K. V. Jakobsen, J. Katajainen, and J. Teuhola. Heaps and heapsort on secondary storage. *Theor. Comp. Sci.*, 220(2):345–362, 1999.

[13] S. Jenson and J. Domingue. Extracting topographic structure from digital elevation data for geographic information system analysis. *Photogrammetric Eng. and Remote Sensing*, 54(11):1593–1600, 1988.

[14] Y. Liu and J. Snoeyink. Flooding triangulated terrain. In *Proceedings of the 11th International Symposium on Spatial Data Handling*, pages 137–148, 2005.

[15] J. F. O'Callaghan and D. M. Mark. The extraction of drainage networks from digital elevation data. *Computer Vision, Graphics and Image Proc.*, 28(3):323–344, 1984.

[16] O. Palacios-Velez, W. Gandoy-Bernasconi, and B. Cuevas-Renaud. Geometric analysis of surface runoff and the computation order of unit elements in distributed hydrological models. *J. Hydrology*, 211:266–274, 1998.

[17] D. M. Theobald and M. F. Goodchild. Artifacts of TIN-based surface flow modeling. In *Proc. GIS/LIS'90*, pp. 955–964, 1990.

[18] G. Tucker, S. Lancaster, N. Gasparini, and S. Rybarczyk. An object-oriented framework for hydrology and geomorphic modeling using triangulated irregular networks. *Computers and Geosciences*, 27(8):959–973, 2001.

Cache-Oblivious Dynamic Dictionaries with Update/Query Tradeoffs

Gerth Stølting Brodal[*†] Erik D. Demaine[‡†] Jeremy T. Fineman[‡§] John Iacono[¶]
Stefan Langerman[‖] J. Ian Munro[**]

Abstract

Several existing cache-oblivious dynamic dictionaries achieve $O(\log_B N)$ (or slightly better $O(\log_B \frac{N}{M})$) memory transfers per operation, where N is the number of items stored, M is the memory size, and B is the block size, which matches the classic B-tree data structure. One recent structure achieves the same query bound and a sometimes-better amortized update bound of $O\left(\frac{1}{B^{\Theta(1/(\log\log B)^2)}}\log_B N + \frac{1}{B}\log^2 N\right)$ memory transfers. This paper presents a new data structure, the *xDict*, implementing predecessor queries in $O(\frac{1}{\varepsilon}\log_B \frac{N}{M})$ worst-case memory transfers and insertions and deletions in $O\left(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B \frac{N}{M}\right)$ amortized memory transfers, for any constant ε with $0 < \varepsilon < 1$. For example, the xDict achieves subconstant amortized update cost when $N = M\,B^{o(B^{1-\varepsilon})}$, whereas the B-tree's $\Theta(\log_B \frac{N}{M})$ is subconstant only when $N = o(MB)$, and the previously obtained $\Theta\left(\frac{1}{B^{\Theta(1/(\log\log B)^2)}}\log_B N + \frac{1}{B}\log^2 N\right)$ is subconstant only when $N = o(2^{\sqrt{B}})$. The xDict attains the optimal tradeoff between insertions and queries, even in the broader external-memory model, for the range where inserts cost between $\Omega(\frac{1}{B}\lg^{1+\varepsilon} N)$ and $O(1/\lg^3 N)$ memory transfers.

1 Introduction

This paper presents a new data structure, the xDict, which is the asymptotically best data structure for the dynamic-dictionary problem in the cache-oblivious model.

Memory models. The *external-memory (or I/O) model* [1] is the original model of a two-level memory hierarchy. This model consists of an internal memory of size M and a disk storing all remaining data. The algorithm can transfer contiguous blocks of data of size B to or from disk at unit cost. The textbook data structure in this model is the B-tree [2], a dynamic dictionary that supports inserts, deletes, and predecessor queries in $O(\log_B N)$ memory transfers per operation.

The *cache-oblivious model* [9, 10] arose in particular from the need to model multi-level memory hierarchies. The premise is simple: analyze a data structure or algorithm just as in the external-memory model, but the data structure or algorithm is not explicitly parametrized by M or B. Thus the analysis holds for an arbitrary M and B, in particular all the M's and B's encountered at each level of the memory hierarchy. The algorithm could not and fortunately does not have to worry about the block replacement strategy because the optimal strategy can be simulated with constant overhead. This lack of parameterization has let algorithm designers develop elegant solutions to problems by finding the best ways to enforce data locality.

Comparison of cache-oblivious dictionaries. Refer to Table 1. In the cache-oblivious model, Prokop's static search structure [10] was the first to support predecessor searches in $O(\log_B N)$ memory transfers, but it does not support insertion or deletion. Cache-oblivious B-trees [3, 4, 7] achieve $O(\log_B N)$ memory transfers for insertion, deletion, and search. The shuttle tree [5] supports insertions and deletions in amortized $O\left(\frac{1}{B^{\Theta(1/(\log\log B)^2)}}\log_B N + \frac{1}{B}\log^2 N\right)$ memory transfers, which is an improvement over $\Theta(\log_B N)$ for $N = 2^{o(B/\log B)}$, while preserving the $O(\log_B N)$ query bound.[1] Our xDict reduces the insertion and deletion bounds further to $O\left(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B \frac{N}{M}\right)$, for any constant $0 < \varepsilon \le 1$, under the *tall-cache assumption* (common to many cache-oblivious algorithms) that $M = \Omega(B^2)$. For all of these data structures, the query bounds are worst case and the update bounds are amortized.

[*]Department of Computer Science, Aarhus University, IT-parken, Åbogade 34, DK-8200 Århus N, Denmark, gerth@cs.au.dk

[†]Supported in part by MADALGO — Center for Massive Data Algorithmics — a Center of the Danish National Research Foundation.

[‡]MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, edemaine@mit.edu, jfineman@cs.cmu.edu

[§]Supported in part by NSF Grants CCF-0541209 and CCF-0621511, and Computing Innovation Fellows.

[¶]Department of Computer Science and Engineering, Polytechnic Institute of New York University, 5 MetroTech Center, Brooklyn, NY 11201, USA, http://john.poly.edu

[‖]Maître de recherches du F.R.S.-FNRS, Computer Science Department, Université Libre de Bruxelles, CP212, Boulevard du Triomphe, 1050 Bruxelles, Belgium, stefan.langerman@ulb.ac.be

[**]Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, imunro@uwaterloo.ca

[1]For these previous data structures, the $\log_B N$ terms may well reduce to $\log_B \frac{N}{M}$ terms, but only $\log_B N$ was explicitly proven.

Data Structure	Search	Insert/Delete
static search [10]	$O(\log_B N)$	not supported
B-trees [3, 4, 7]	$O(\log_B N)$	$O(\log_B N)$
shuttle tree [5]	$O(\log_B N)$	$O\left(\frac{1}{B^{\Theta(1/(\log\log B)^2)}}\log_B N + \frac{1}{B}\log^2 N\right)$
lower bound [6]	$O(\frac{1}{\varepsilon}\log_B \frac{N}{M})$	$\Longrightarrow \quad \Omega\left(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B \frac{N}{M}\right)$
xDict [this paper]	$O(\frac{1}{\varepsilon}\log_B \frac{N}{M})$	$O\left(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B \frac{N}{M}\right)$

Table 1: Summary of known cache-oblivious dictionaries. Our xDict uses the tall-cache assumption that $M = \Omega(B^2)$.

Lower bounds. The tradeoff between insertion and search costs was partly characterized in the external-memory model [6]. Because any algorithm in the cache-oblivious model is also an algorithm in the external-memory model using the same number of memory transfers, lower bounds for the external-memory model carry over to the cache-oblivious model. Brodal and Fagerberg [6] proved that any structure supporting insertions in $I = O\left(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B(N/M)\right)$ amortized memory transfers, when I is between $\Omega(\frac{1}{B}\lg^{1+\varepsilon} N)$ and $O(1/\lg^3 N)$ and when $N \geq M^2$, must use $\Omega(\frac{1}{\varepsilon}\log_B(N/M))$ worst-case memory transfers for search. They also produced a data structure achieving this tradeoff, but their structure is not cache-oblivious, using knowledge of the parameters B and M. The xDict structure achieves the same tradeoff in the cache-oblivious model, and is therefore optimal for this range of insertion cost I. Slightly outside this range, the optimal bounds are not known, even in the external-memory model.

2 Introducing the x-box

Our xDict dynamic-dictionary data structure is built in terms of another structure called the x-box. For any positive integer x, an *x-box* supports a batched version of the dynamic-dictionary problem (defined precisely later) in which elements are inserted in batches of $\Theta(x)$. Each x-box will be defined recursively in terms of y-boxes for $y < x$, and later we build the overall xDict data structure in terms of x-boxes for x increasing doubly exponentially. Every x-box uses a global parameter, a real number $\alpha > 0$, affecting the insertion cost, with lower values of α yielding cheaper insertions. This parameter is chosen globally and remains fixed throughout.

As shown in Figure 1, an x-box is composed of three buffers (arrays) and many $\sqrt{x}$-boxes, called *subboxes*. The three buffers of an x-box are the *input buffer* of size x, the *middle buffer* of size $x^{1+\alpha/2}$, and the *output buffer* of size $x^{1+\alpha}$. The $\sqrt{x}$-subboxes of an x-box are divided into two levels: the *upper level* consists of at most $\frac{1}{4}x^{1/2}$ subboxes, and the *lower level* consists of at most $\frac{1}{4}x^{1/2+\alpha/2}$ subboxes. Thus, in total, there are fewer than $\frac{1}{2}x^{1/2+\alpha/2}$ subboxes. See Table 2 for a table of buffer counts and sizes. As a base case, an $O(1)$-box consists of a single array that acts as both the input and output buffers, with no recursive subboxes or middle buffer.

Logically, the upper-level subboxes are children of the input buffer and parents of the middle buffer. Similarly, the lower-level subboxes are children of the middle buffer and parents of the output buffer. However, the buffers and subboxes do not necessarily form a tree structure. Specifically, for an x-box D, there are pointers from D's input buffer to the input buffers of its upper-level subboxes. Moreover, there may be many pointers from D's input buffers to a single subbox. There are also pointers from the output buffers of the upper-level subboxes to D's middle buffer. Again, there may be many pointers originating from a single subbox. Similarly, there are pointers from D's middle buffer to its lower-level subboxes' input buffers, and from the lower-level subboxes' output buffers to D's output buffers.

The number of subboxes in each level has been chosen to match the buffer sizes. Specifically, the total size of the input buffers of all subboxes in the upper level is at most $\frac{1}{4}x^{1/2} \cdot x^{1/2} = \frac{1}{4}x$, which is a constant factor of the size of the x-box's input buffer. Similarly, the total size of the upper-level subboxes' output buffers is at most $\frac{1}{4}x^{1/2} \cdot (x^{1/2})^{1+\alpha} = \frac{1}{4}x^{1+\alpha/2}$, which matches the size of the x-box's middle buffer. The total size of the lower-level subboxes' input and output buffers are at most $\frac{1}{4}x^{1/2+\alpha/2} \cdot x^{1/2} = \frac{1}{4}x^{1+\alpha/2}$ and $\frac{1}{4}x^{1/2+\alpha/2} \cdot (x^{1/2})^{1+\alpha} = \frac{1}{4}x^{1+\alpha}$, which match the sizes of the x-box's middle and output buffers, respectively, to within a constant factor.

An x-box D organizes elements as follows. Suppose that the keys of elements contained in D range from $[\kappa_{\min}, \kappa_{\max})$. The elements located in the input buffer occur in sorted order, as do the elements located in the middle buffer and the elements located in the output buffer. All three of these buffers may contain elements having any keys between $\kappa_{\min}$ and $\kappa_{\max}$. The upper-level subboxes, however, partition the key space. More precisely, suppose that there are r upper-level subboxes. Then there exist keys $\kappa_{\min} = \kappa_0 < \kappa_1 < \cdots < \kappa_r = \kappa_{\max}$ such that each subbox contains elements in a distinct range $[\kappa_i, \kappa_{i+1})$. Similarly, the lower-level subboxes partition the key space. There is, however, no relationship between the partition imposed by the upper-level subboxes and that of the lower-level subboxes; the subranges are entirely unrelated. What this setup

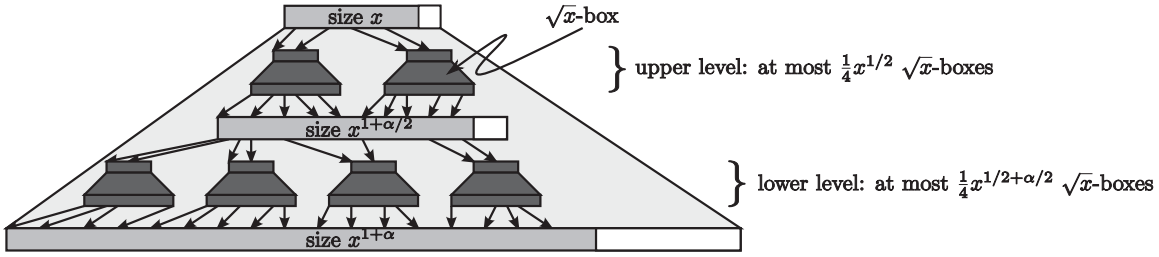


Figure 1: The recursive structure of an x-box. The arrows represent lookahead pointers. These lookahead pointers are evenly distributed in the target buffer, but not necessarily in the source buffer. Additional pointers (not shown) allow us to find the nearest lookahead pointer(s) in $O(1)$ memory transfers. The white space at the right of the buffers indicates empty space, allowing for future insertions.

Buffer	**Size per buffer**	**Number of buffers**	**Total size**
Top buffer	x	1	x
Top buffer	$x^{1/2}$	$\frac{1}{4}x^{1/2}$	$\frac{1}{4}x$
Middle buffer	$(x^{1/2})^{1+\alpha/2}$	$\frac{1}{4}x^{1/2}$	$\frac{1}{4}x^{1+\alpha/4}$
Bottom buffer	$(x^{1/2})^{1+\alpha}$	$\frac{1}{4}x^{1/2}$	$\frac{1}{4}x^{1+\alpha/2}$
Middle buffer	$x^{1+\alpha/2}$	1	$x^{1+\alpha/2}$
Top buffer	$x^{1/2}$	$\frac{1}{4}x^{1/2+\alpha/2}$	$\frac{1}{4}x^{1+\alpha/2}$
Middle buffer	$(x^{1/2})^{1+\alpha/2}$	$\frac{1}{4}x^{1/2+\alpha/2}$	$\frac{1}{4}x^{1+3\alpha/4}$
Bottom buffer	$(x^{1/2})^{1+\alpha}$	$\frac{1}{4}x^{1/2+\alpha/2}$	$\frac{1}{4}x^{1+\alpha}$
Bottom buffer	$x^{1+\alpha}$	1	$x^{1+\alpha}$

Table 2: Sizes of buffers in an x-box. This table lists the sizes of the three buffers in an x-box and the sizes and number of buffers in its recursive $x^{1/2}$-boxes, expanding just one level of recursion.

means is that an element with a particular key may be located in the input buffer or the middle buffer or the output buffer or a particular upper-level subbox or a particular lower-level subbox. Our search procedure will thus look in all five of these locations to locate the element in question.

Before delving into more detail about the x-boxes, let us first give a rough sketch of insertions into the data structure. When an element is inserted into an x-box D, it is first inserted into D's input buffer. As the input buffer stores elements in sorted order, elements in the input buffer must move to the right to accommodate newly inserted elements. When D's input buffer becomes full (or nearly full), the elements are inserted recursively into D's upper-level subboxes. When an upper-level subbox becomes full enough, it is "split" into two subboxes, with one subbox taking the half of the elements with smaller keys and the other taking the elements with larger keys. When the maximum number of upper-level subboxes is reached, all elements are moved from the upper-level subboxes to D's middle buffer (which stores all elements in sorted order). When the middle buffer becomes full enough, elements are moved to the lower-level subboxes in a similar fashion. When the maximum number of lower-level subboxes is reached, all elements are moved from the lower-level subboxes to D's output buffer.

To aid in pushing elements down to the appropriate subboxes, we embed in the input (and middle) buffers a pointer to each of the upper-level (and lower-level) subboxes. These *subbox pointers* are embedded into the buffers by associating with them the minimum key stored in the corresponding subbox, and then storing them along with the buffer's elements in sorted order by key.

To facilitate searches, we employ the technique of *fractional cascading* [8], giving an x-box's input buffer (and middle buffer) a sample of the elements of the upper-level subboxes' (and lower-level subboxes') input buffers. Specifically, let U be an upper-level subbox of the x-box D. Then a constant fraction of the keys stored in U's input buffer are also stored in D's input buffer. This sampling is performed deterministically by placing every sixteenth element in U's input buffer into D's input buffer. The sampled element (in D's input buffer) has a pointer to the corresponding element in U's input buffer. This type of pointer also occurs in [5], where they are called "lookahead pointers." We adopt the same term here. This sampling also occurs on the output buffers. Specifically, the output buffers of the upper-level (and lower-level) subboxes contain a similar sample of the

elements in D's middle buffer (and output buffer).[2]

The advantage of lookahead pointers is roughly as follows. Suppose we are looking for a key κ in some buffer A. Let s be the multiple of 16 (i.e., a sampled element) such that $A[s] \leq \kappa < A[s+16]$. Then our search procedure will scan slots s to $s+16$ in the buffer A. If κ is not located in any of these slots, then it is not in the buffer. Ideally, this scan would also provide us with a good starting point to search within the next buffer.

As the lookahead pointers may be irregularly distributed in D's input (or middle) buffer, a stretch of sixteen elements in D's input (or middle) buffer may not contain a lookahead pointer to an upper (or lower)-level subboxes. To remedy this problem, we associate with each element in D's input (or middle) buffer a pointer to the nearest lookahead or subbox pointers preceding and following it.

Techniques. While fractional cascading has been employed before [5], we are not aware of any previous cache-oblivious data structures that are both recursive and that use fractional cascading. Another subtler difference between the x-box and previous cache-oblivious structures involves the sizing of subboxes and our use of the middle buffer. If the x-box matched typical structures, then an x-box with an input buffer of size x and an output buffer of size $x^{1+\alpha}$ would have a middle buffer of size $x^{\sqrt{1+\alpha}}$, not the $x^{1+\alpha/2}$ that we use. That is to say, it is natural to size the buffers such that a size-x input buffer is followed by a size-x^δ middle buffer for some δ, and a size-$y = x^\delta$ middle buffer is followed by a size-$y^\delta = x^{\delta^2}$ output buffer. Our choice of sizes causes the data structure to be more topheavy than usual, a feature which facilitates obtaining our query/update tradeoff.

3 Sizing an x-box

An x-box stores the following fields, in order, in a fixed contiguous region of memory.

1. A counter of the number of real elements (not counting lookahead pointers) stored in the x-box.
2. The top buffer.
3. Array of booleans indicating which upper-level subboxes are being used.
4. Array of upper-level subboxes, in an arbitrary order.
5. The middle buffer.
6. Array of booleans indicating which lower-level subboxes are being used.
7. Array of lower-level subboxes, in an arbitrary order.
8. The bottom buffer.

In particular, the entire contents of each $\sqrt{x}$-subbox are stored within the x-box itself. In order for an x-box to occupy a fixed contiguous region of memory, we need a upper bound on the maximum possible space usage of a box.

LEMMA 3.1. *The total space usage of an x-box is at most $c\,x^{1+\alpha}$ for some constant $c > 0$.*

Proof. The proof is by induction. An x-box contains three buffers of total size $c'(x+x^{1+\alpha/2}+x^{1+\alpha}) \leq 3c'x^{1+\alpha}$, where c' is the constant necessary for each array entry (including information about lookahead pointers, etc.). The boolean arrays use a total of at most $\frac{1}{2}x^{1/2+\alpha/2} \leq c'x^{1+\alpha}$ space, giving us a running total of $4c'x^{1+\alpha}$ space. Finally, the subboxes by assumption use a total of at most $c(x^{1/2})^{1+\alpha} \cdot \frac{1}{2}x^{1/2+\alpha/2} = \frac{c}{2}x^{1+\alpha}$ space. Setting $c \geq 8c'$ yields a total space usage of at most $cx^{1+\alpha}$. □

4 Operating an x-box

An x-box D supports two operations:

1. BATCH-INSERT$(D, e_1, e_2, \ldots, e_{\Theta(x)})$: Insert $\Theta(x)$ keyed elements $e_1, e_2, \ldots, e_{\Theta(x)}$, given as a sorted array, into the x-box D. BATCH-INSERT maintains lookahead pointers as previously described.
2. SEARCH(D, s, κ): Return a pointer to an element with key κ in the x-box D if such an element exists. If no such element exists, return a pointer to κ's predecessor in D's output buffer, that is, the element located in D's output buffer with the largest key smaller than κ. We assume that we are given the nearest lookahead pointer s preceding κ pointing into D's input buffer: that is, s points to the sampled element (i.e., having an index that is a multiple of 16) in D's input buffer that has the largest key not exceeding κ.

We treat an x-box as *full* or *at capacity* when it contains $\frac{1}{2}x^{1+\alpha}$ real elements. A BATCH-INSERT is thus only allowed when the inserted elements would not cause the x-box to contain more than $\frac{1}{2}x^{1+\alpha}$ elements. Our algorithm does not continue to insert into any recursive x-box when this number of elements is exceeded. The constant can be tuned to waste less space, but we choose $\frac{1}{2}$ here to simplify the presentation. Recall that the output buffer of an x-box has size $x^{1+\alpha}$, which is twice the size necessary to accommodate all the real elements in the x-box. We allow the other

[2] Any sufficiently large sampling constant suffices here. We chose 16 as an example of one such constant for concreteness. The constant must be large enough that, when sampling D's output buffer, the resulting lookahead pointers occupy only a constant fraction of the lower-level subboxes' output buffers. To make the description more concise, the constant fraction we allow is $\frac{1}{4}$, but in fact any constant would work. As the lower-level subboxes' output buffers account for only $\frac{1}{4}$ of the space of D's output buffer, these two constants are multiplied to get that only $\frac{1}{16}$ of the elements in D's output buffer may be sampled.

$\frac{1}{2}x^{1+\alpha}$ space in the output buffer to store external lookahead pointers (i.e., lookahead pointers into a buffer in the containing x^2-box).

4.1 SEARCH. Searches are easiest. The operation SEARCH(D, s, κ) starts by scanning D's input buffer at slot s and continues until reaching an element with key κ or until reaching slot s' where the key at s' is larger than κ. By assumption on lookahead pointers, this scan considers $O(1)$ array slots. If the scan finds an element with key κ, then we are done. Otherwise, we consider the nearest lookahead or subbox pointer preceding slot $s' - 1$. We follow this pointer and search recursively in the corresponding subbox. That recursive search returns a pointer in the subbox's output buffer. We again reference the nearest lookahead pointer and jump to a point in the middle buffer. This process continues, scanning a constant-size region in middle buffer, searching recursively in the lower-level subbox, and scanning a constant-size region in the output buffer.

LEMMA 4.1. *For $x > B$, a* SEARCH *in an x-box costs $O((1+\alpha)\log_B x)$ memory transfers.*

Proof. The search considers a constant-size region in the three buffers, for a total of $O(1)$ memory transfers, and performs two recursive searches. Thus, the cost of a search can be described by the recurrence $S(x) = 2S(\sqrt{x}) + O(1)$. Once $x^{1+\alpha} = O(B)$, or equivalently $x = O(B^{1/(1+\alpha)})$, an entire x-box fits in a block, and no further recursions incur any other cost. Thus, we have a base case of $S(O(B^{1/(1+\alpha)})) = 0$.

The recursion therefore proceeds with a nonzero cost for $\lg\lg x - \lg\lg O(B^{1/(1+\alpha)})$ levels, for a total of $2^{\lg\lg x - \lg\lg O(B^{1/(1+\alpha)})} = (1+\alpha)\lg x / \lg O(B) = O((1+\alpha)\log_B x)$ memory transfers. □

4.2 BATCH-INSERT overview. For clarity, we decompose BATCH-INSERT into several operations, including two new auxiliary operations:

1. FLUSH(D): After this operation, all k elements in the x-box D are located in the first $\Theta(k)$ slots of D's output buffer. These elements occur in sorted order. All other buffers and recursive subboxes are emptied, temporarily leaving the D without any internal lookahead pointers (to be fixed later by a call SAMPLE-UP). The $\Theta()$ arises because of the presence of lookahead pointers directed from the output buffer. The FLUSH operation is an auxiliary operation used by the BATCH-INSERT.

2. SAMPLE-UP(D): This operation may only be invoked on an x-box that is entirely empty except for its output buffer (as with one that has just been FLUSHed). The sampling process is employed from the bottom up, creating subboxes as necessary, and placing the appropriate lookahead pointers. The SAMPLE-UP operation is an auxiliary operations used by BATCH-INSERT.

4.3 FLUSH. To FLUSH an x-box, first flush all the subboxes. Now consider the result. Elements can live in only five possible places: the input buffer, the middle buffer, the output buffer, the upper-level subboxes' output buffers, and the lower-level subboxes' output buffers. The elements are in sorted order in all of the buffers. Moreover, as the upper-level (and lower-level) subboxes partition the key space, the collection of upper-level subboxes' output buffers form a fragmented sorted list of elements. Thus, after flushing all subboxes, moving all elements to the output buffer requires just a 5-way merge into the output buffer. A constant-way merge can be performed in a linear number of memory transfers in general, but here we have to deal with the fact that upper-level and lower-level subboxes represent fragmented lists, requiring random accesses to jump from the end of one output buffer to the beginning of another.

When merging all the elements into the output buffer, we merge only real elements, not lookahead pointers (except for the lookahead pointers that already occur in the output buffer). This step breaks the sampling structure of the data structure, which we will later resolve with the SAMPLE-UP procedure.

When the flush completes, the input and middle buffers are entirely empty, and all subboxes are deleted.[3]

LEMMA 4.2. *For $x^{1+\alpha} > B$, a* FLUSH *in an x-box costs $O(x^{1+\alpha}/B)$ memory transfers.*

Proof. We can describe the flush by the recurrence

$$F(x) = O(x^{1+\alpha}/B) + O(x^{1/2+\alpha/2}) + \frac{1}{2}x^{1/2+\alpha/2}F(\sqrt{x}) ,$$

where the first term arises due to scanning all the buffers (i.e., the 5-way merge), the second term arises from the random accesses both to load the first block of any of the subboxes and to jump when scanning the concatenated list of output buffers, and the third term arises due to the recursive flushing. When $x^{1+\alpha} = O(M)$, the second term disappears as the entire x-box fits in memory, and we thus need only pay for loading each block once. When applying a tall-cache assumption that $M = \Omega(B^2)$, it follows that the second term only occurs when $x^{1/2+\alpha/2} = \Omega(B)$, and hence when $x^{1/2+\alpha/2} = x^{1+\alpha}/x^{1/2+\alpha/2} = x^{1+\alpha}/\Omega(B) = O(x^{1+\alpha}/B)$. We thus have a total cost of

$$F(x) \le c_1 x^{1+\alpha}/B + \frac{1}{2}x^{1/2+\alpha/2}F(\sqrt{x}) ,$$

[3] In fact, the subboxes use a fixed memory footprint, so they are simply marked as deleted.

where c_1 is a constant hidden by the order notation.

We next prove that $F(x) \leq cx^{1+\alpha}/B$ by induction. As a base case, when $y^{1+\alpha}$ fits in memory, the cost is $F(y) = cy^{1+\alpha}/B$ as already noted, to load each block into memory once. Applying the inductive hypothesis that $F(y) \leq cy^{1+\alpha}/B$ for some sufficiently large constant c and $y < x$, we have $F(x) \leq c_1 x^{1+\alpha}/B + \frac{1}{2}x^{1/2+\alpha/2}(cx^{1/2+\alpha/2}/B) = c_1 x^{1+\alpha}/B + \frac{1}{2}cx^{1+\alpha}/B$. Setting $c > 2c_1$ completes the proof. □

4.4 SAMPLE-UP. When invoking a SAMPLE-UP, we assume that the only elements in the x-box live in the output buffer. In this state, there are no lookahead pointers to facilitate searches. The SAMPLE-UP recomputes the lookahead pointers, allowing future searches to be performed efficiently.

The SAMPLE-UP operates as follows. Suppose that the output buffer contains $k < x^{1+\alpha}$ elements. Then we create $(k/16)/(x^{1/2+\alpha/2}/2) = k/8x^{1/2+\alpha/2} \leq x^{1/2+\alpha/2}/8$ new lower-level subboxes. Recall that this is half the number of available lower-level subboxes. We then assign to each of these subboxes $x^{1/2+\alpha/2}/2$ of contiguous sampled pointers (filling half of their respective output buffers), and recursively call SAMPLE-UP in the subboxes. Then, we sample from the lower-level subboxes' input buffers to the middle buffer, and we sample the middle-buffer to upper-level subboxes in a similar fashion. Finally, we sample the upper-level subboxes up to the input buffer.

LEMMA 4.3. *A* SAMPLE-UP *in an x-box, for $x^{1+\alpha} > B$, costs $O(x^{1+\alpha}/B)$.*

Proof. The proof is virtually identical to the proof for FLUSH. The recurrence is the same (in fact, it is better here because we can guarantee that the subboxes are, in fact, contiguous).
□

4.5 BATCH-INSERT. The BATCH-INSERT operation takes as input a sorted array of elements to insert. In particular, when inserting into an x-box D, a BATCH-INSERT inserts $\Theta(x)$ elements. For conciseness, let us say that the constant hidden by the theta notation is $1/2$. First, merge the inserted elements into D's input buffer, and increment the counter of elements contained in D by $(1/2)x$. For simplicity, also remove the lookahead pointers during this step. We will readd them later.

Then, (implicitly) partition the input buffer according to the ranges owned by each of the upper-level subboxes. For any partition containing at least $(1/2)\sqrt{x}$ elements, repeatedly remove $(1/2)\sqrt{x}$ elements from D's input buffer and insert them recursively into the appropriate subbox until the partition contains less than $(1/2)\sqrt{x}$ elements. If performing a recursive insert would cause the number of (real) elements in the subbox to exceed $(1/2)\sqrt{x}^{1+\alpha}$, first "split" the subbox. A "split" entails first FLUSHing the subbox, creating a new subbox, moving the larger half of the elements from the old subbox's output buffer to the new subbox's output buffer (involving two scans), and calling SAMPLE-UP on both subboxes, and updating the counter recording the number of elements in each subbox to reflect the number of real elements in each.

Observe that after all the recursions occur, the number of real elements in D's input buffer is at most $(1/2)\sqrt{x} \cdot (1/4)\sqrt{x} = (1/8)x$, as otherwise more elements would have moved down to a subbox.

After moving elements from the input buffer to the upper-level subboxes, resample from the upper-level subboxes' input buffers into D's input buffer. Note that the number of lookahead pointers introduced into the input buffer is at most $\frac{1}{16}\sqrt{x} \cdot \frac{1}{4}\sqrt{x}$. When combining the number of lookahead pointers with the number of real elements, we see that D's input buffer is far less than half full, and hence it can accommodate the next insertion.

When a split causes the last available subbox to be allocated, we abort any further recursive inserts and instead merge all of D's input-buffer and upper-level subbox elements into D's middle buffer. This merge entails first FLUSHing all the upper-level subboxes, and then merging into the middle buffer (similar to the process for the full FLUSH). We then perform an analogous movement from the middle buffer to the lower-level subboxes, matching the movement from the upper-level subboxes to the middle buffer. (If the last lower-level subbox is allocated, we move all elements from D's middle buffer and lower-level subboxes to D's output buffer and then call SAMPLE-UP on D.) When insertions into the lower-level subboxes complete, we allocate new upper-level subboxes (as in SAMPLE-UP) and sample from D's middle buffer into the upper-level suboxes' output buffers, call SAMPLE-UP recursively on these subboxes, and finally sample from the subboxes' input buffers into D's input buffer.

Observe that the upper-level subboxes' output buffers collectively contain at most $\frac{1}{16}x^{1+\alpha/2}$ lookahead pointers. Moreover, after elements are moved into the middle buffer, these are the only elements in the upper-level subboxes, and they are spread across at most half $(x^{1/2}/8)$ the upper-level subboxes, as specified for SAMPLE-UP. Hence, there must be at least $x^{1/2}/8$ subbox splits between moves into the middle buffer. Because subboxes splits only occur when the two resulting subboxes contain at least $(1/4)x^{1+\alpha/2}$ real elements, it follows that there must be at least $(1/4)x^{1+\alpha/2} \cdot (1/8)x^{1+\alpha/2} = \Omega(x^{1+alpha/2})$ insertions into D between moves into the middle buffer. A similar argument shows that there must be at least $\Omega(x^{1+\alpha})$ insertions into D between insertions into the output buffer.

THEOREM 4.1. *A* BATCH-INSERT *into an x-box, with $x > B$, costs an amortized $O((1+\alpha)\log_B(x)/B^{1/(1+\alpha)})$ memory transfers per element.*

Proof. An insert has several costs per element. First, there is the cost of merging into the input array, which is simply $O(1/B)$ per element. Next, each element is inserted recursively into a top-level subbox. These recursive insertions entail random accesses to load the first block of each of the subboxes. Then these subboxes must be sampled, which is dominated by the cost of the aforementioned random accesses and scans. An element may also contribute to a split of a subbox, but each split may be amortized against $\Omega(x^{1/2+\alpha/2})$ insertions. Then we must also consider the cost of moving elements from the upper-level subboxes to the middle buffer, but this movement may be amortized against the $\Omega(x^{1+\alpha/2})$ elements being moved. Finally, there are similar costs among the lower-level subboxes.

Let us consider the cost of the random accesses more closely. If all of the upper-level subboxes fit into memory, then the cost of random accesses is actually the minimum of performing the random accesses or loading the entire upper-level into memory. For the upper-level, we denote this value by $\mathit{UpperRA}(x)$. We thus have $\mathit{UpperRA}(x) = O(\frac{1}{4}x^{1/2})$ if $x^{1+\alpha/2} = \Omega(M)$, and $\mathit{UpperRA}(x) = O(\min\left\{\frac{1}{4}x^{1/2}, x^{1+\alpha/2}/B\right\})$ if $x^{1+\alpha/2} = O(M)$. In fact, we are really concerned with $\mathit{UpperRA}(x)/x$, as the random accesses can be amortized against the x elements inserted. We analyze the two cases separately, and we assume the tall-cache assumption that $M > B^2$.

1. Suppose that $x^{1+\alpha/2} = \Omega(M)$. Then we have $x^{1+\alpha/2} = \Omega(B^2)$ by the tall-cache assumption, and hence $x^{1/2} = \Omega(B^{2/(2+\alpha)})$. It follows that $\mathit{UpperRA}(x)/x = O(1/\sqrt{x}) = O(1/B^{2/(2+\alpha)})$.

2. Suppose that $x^{1+\alpha/2} = O(M)$. We have two subcases here. If $x > B^{2/(1+\alpha)}$, then we have a cost of at most $\mathit{UpperRA}(x)/x = O(x^{1/2}/x) = O(1/B^{1/(1+\alpha)})$. If, on the other hand, $x < B^{2/(1+\alpha)}$, then we have $\mathit{UpperRA}(x)/x = O(x^{1+\alpha/2}/Bx) = O(x^{\alpha/2}/B) = O(B^{\alpha/(1+\alpha)}/B) = O(1/B^{1/(1+\alpha)})$.

Because $1/B^{2/(2+\alpha)} < 1/B^{1/(1+\alpha)}$, we conclude that $\mathit{UpperRA}(x)/x = O(1/B^{1/(1+\alpha)})$.

We must also consider the cost of random accesses into the lower-level subboxes, which can be amortized against the $x^{1+\alpha/2}$ elements moved. A similar case analysis shows that $\mathit{LowerRA}(x)/x^{1+\alpha/2} = O(1/B^{1/(1+\alpha)})$.

We thus have a total insertion cost of

$$\begin{aligned} I(x) &= O\left(\frac{x/B}{x}\right) + O\left(\frac{\mathit{UpperRA}(x)}{x}\right) + I(\sqrt{x}) \\ &\quad + O\left(\frac{F(\sqrt{x})}{x^{1/2+\alpha/2}}\right) + O\left(\frac{\frac{1}{4}x^{1/2}F(\sqrt{x})}{x^{1+\alpha/2}}\right) \\ &\quad O\left(\frac{x^{1+\alpha/2}/B}{x^{1+\alpha/2}}\right) + O\left(\frac{\mathit{LowerRA}(x)}{x^{1+\alpha/2}}\right) \\ &\quad + I(\sqrt{x}) + O\left(\frac{F(\sqrt{x})}{x^{1/2+\alpha/2}}\right) \\ &\quad + O\left(\frac{\frac{1}{4}x^{1/2+\alpha/2}F(\sqrt{x})}{x^{1+\alpha}}\right) + O\left(\frac{x^{1+\alpha}/B}{x^{1+\alpha}}\right) \\ &= O(1/B) + O\left(\frac{\mathit{UpperRA}(x)}{x}\right) \\ &\quad + O\left(\frac{\mathit{LowerRA}(x)}{x^{1+\alpha/2}}\right) + O\left(\frac{F(\sqrt{x})}{x^{1/2+\alpha/2}}\right) \\ &\quad + 2\,I(\sqrt{x}) \\ &= O(1/B) + O(1/B^{1/(1+\alpha)}) + O(1/B^{1/(1+\alpha)}) \\ &\quad + O\left(\frac{x^{1/2+\alpha/2}/B}{x^{1/2+\alpha/2}}\right) + 2\,I(\sqrt{x}) \\ &= O(1/B^{1/(1+\alpha)}) + 2\,I(\sqrt{x}) \end{aligned}$$

As we charge loading the first block of a subbox to an insert into the parent, we have a base case of $I(O(B^{1/(1+\alpha)})) = 0$, i.e., when the x-box fits into a single block. Solving the recurrence, we get a per element cost of $O(1/B^{1/(1+\alpha)})$ at $\lg\lg x - \lg\lg O(B^{1/(1+\alpha)})$ levels of recursion, and hence a total cost of $O(2^{\lg\lg x - \lg\lg O(B^{1/(1+\alpha)})}/B^{1/(1+\alpha)}) = O\left(\frac{(1+\alpha)\lg x}{B^{1/(1+\alpha)}\lg O(B)}\right) = O((1+\alpha)\log_B(x)/B^{1/(1+\alpha)})$. □

5 Building a dictionary out of x-boxes

The xDict data structure consists of $\log_{1+\alpha}\log_2 N + 1$ x-boxes of doubly increasing size, where α is the same parameter for the underlying x-boxes. Specifically, for $0 \le i \le \log_{1+\alpha}\log_2 N$, the ith box has $x = 2^{(1+\alpha)^i}$. The x-boxes are linked together by incorporating into the ith box's output buffer the lookahead pointers corresponding to a sample from the $(i+1)$st box's input buffer.

We can define the operations on an xDict in terms of x-box operations. To insert an element into an xDict, we simply insert it into the smallest x-box ($i = 0$), which has $x = \Theta(1)$ so supports individual element insertions. When the ith box reaches capacity (containing $2^{(1+\alpha)^{i+1}}/2$ elements), we FLUSH it, insert all of its elements (contained in its output buffer) into the $(i+1)$st box, and empty the ith box's output buffer. This process terminates after performing a batch insert into the jth box if the jth box is the first box that can accommodate the elements (having not yet reached capacity). At this point, all boxes preceding the jth box are entirely empty. We next rebuild the lookahead starting from the $(j-1)$st box down to the 0th box by sampling from the $(i+1)$st box's input buffer into the ith box's output buffer and then calling SAMPLE-UP on the ith box. As elements are first inserted into the 0th box and eventually

move through all boxes, our insertion analysis accounts for an insertion into each box for each element inserted. The cost of the FLUSH and SAMPLE-UP incurred by each element as it moves through each of the boxes is dominated by the cost of the insertion.

To search in the xDict, we simply search in each of the x-boxes, and return the closest match. Specifically, we search the boxes in order from smallest to largest. If the element is not found in the ith box, we have a pointer into the ith box's output buffer. We use this pointer to find an appropriate lookahead pointer into the $(i+1)$st box's input buffer, and begin the search from that point.

The performance of the xDict is essentially a geometric series:

THEOREM 5.1. *The xDict supports searches in $O(\frac{1}{\alpha}\log_B \frac{N}{M})$ memory transfers and (single-element) inserts in $O(\frac{1}{\alpha}(\log_B \frac{N}{M})/B^{1/(1+\alpha)})$ amortized memory transfers, for $0 < \alpha \le 1$.*

Proof. A simple upper bound on the search cost is

$$\begin{aligned}
&\sum_{i=0}^{\log_{1+\alpha}\log_2 N} O((1+\alpha)\log_B(2^{(1+\alpha)^i}) \\
&= O\left(\frac{1+\alpha}{\lg B}\sum_{i=0}^{\log_{1+\alpha}\log_2 N}(1+\alpha)^i\right) \\
&= O\left(\frac{(1+\alpha)\lg N}{\lg B}\sum_{i=0}^{\infty}\frac{1}{(1+\alpha)^i}\right) \\
&= O\left(\frac{(1+\alpha)^2}{\alpha}\log_B N\right) \\
&= O\left(\frac{1}{\alpha}\log_B N\right).
\end{aligned}$$

The last step of the derivation follows from the assumption that $\alpha \le 1$ and hence that $(1+\alpha)^2 = O(1)$.

The above analysis, however, exploits only a constant number of cache blocks. If we assume that the memory already holds all x-boxes smaller than $O(M^{1/(1+\alpha)})$,[4] the first $O(\frac{1}{\alpha}\log_B M^{1/(1+\alpha)}) = O(\frac{1}{\alpha}\log_B M)$ transfers are free, resulting in a search cost of $O(\frac{1}{\alpha}\log_B \frac{N}{M})$. The cache-oblivious model assumes an optimal paging strategy, and using half the memory to store the smallest x-boxes is no better than optimal.

The analysis for insertions is identical except that all costs are multiplied by $O(1/B^{1/(1+\alpha)})$. □

COROLLARY 5.1. *For any ε with $0 < \varepsilon < 1$, there exists a setting of α such that the xDict supports searches in $O(\frac{1}{\varepsilon}\log_B \frac{N}{M})$ memory transfers and supports insertions in $O(\frac{1}{\varepsilon}\log_B(\frac{N}{M})/B^{1-\varepsilon})$ amortized memory transfers.*

Proof. First off, we consider only $\varepsilon < \frac{1}{2}$ (rolling up a particular constant into the big-O notation), as larger ε only hurt the performance of inserts.

Choose $\alpha = \varepsilon/(1-\varepsilon)$, which gives $B^{1/(1+\alpha)} = B^{1-\varepsilon}$. Because $\varepsilon < \frac{1}{2}$, we have $\alpha < 1$, and we can apply Theorem 5.1. The $1/\alpha$ term solves to $1/\alpha = (1-\varepsilon)/\varepsilon = O(1/\varepsilon)$ to complete the proof. □

6 Final notes

We did not address deletion in detail, but claim that it can be handed using standard techniques. For example, to delete an element we can insert an anti-element with the same key value. In the course of an operation, should a key value and its antivalue be discovered, they annihilate each other while releasing potential which is used to remove them from the buffers they are in. Rebuilding the whole structure when the number of deletions since the last rebuild is half of the structure ensures that the total size does not get out of sync with the number of not-deleted items currently stored.

Another detail is that, to hold n items, the xDict may create an n-box, which occupies up to $\Theta(n^{1+\alpha})$ of address space. However, only $\Theta(n)$ space of the xDict will ever be occupied, and the layout ensures that the unused space is at the end. Therefore the xDict data structure can use optimal $\Theta(n)$ space.

Acknowledgments

We would like to thank the organizers of the MADALGO Summer School on Cache-Oblivious Algorithms, held in Aarhus, Denmark, on August 18–21, 2008, for providing the opportunity for the authors to come together and create and analyze the structure presented in this paper.

References

[1] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, September 1988.

[2] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189, February 1972.

[3] Michael A. Bender, Erik D. Demaine, and Martin Farach-Colton. Cache-oblivious B-trees. *SIAM Journal on Computing*, 35(2):341–358, 2005.

[4] Michael A. Bender, Ziyang Duan, John Iacono, and Jing Wu. A locality-preserving cache-oblivious dynamic dictionary. *Journal of Algorithms*, 53(2):115–136, 2004.

[5] Michael A. Bender, Martin Farach-Colton, Jeremy T. Fineman, Yonatan R. Fogel, Bradley C. Kuszmaul, and Jelani Nelson. Cache-oblivious streaming B-trees. In *Proceedings of*

[4] All x-boxes with size at most $O(M^{1/(1+\alpha)})$ fit in $O(M)$ memory as the x-box sizes increase supergeometrically.

the 19th Annual ACM Symposium on Parallel Algorithms and Architectures, pages 81–92, San Diego, CA, June 2007.

[6] Gerth Stølting Brodal and Rolf Fagerberg. Lower bounds for external memory dictionaries. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 546–554, Baltimore, MD, January 2003.

[7] Gerth Stølting Brodal, Rolf Fagerberg, and Riko Jacob. Cache oblivious search trees via binary trees of small height. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 39–48, San Francisco, CA, January 2002.

[8] Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(2):133–162, 1986.

[9] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 285–297, New York, NY, 1999.

[10] Harald Prokop. Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1999.

PODS 2010 Best Paper Award

An Optimal Algorithm for the Distinct Elements Problem

Daniel M. Kane *
Harvard University
One Oxford Street
Cambridge, MA 02138
dankane@math.harvard.edu

Jelani Nelson †
MIT CSAIL
32 Vassar Street
Cambridge, MA 02139
minilek@mit.edu

David P. Woodruff
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
dpwoodru@us.ibm.com

ABSTRACT

We give the first optimal algorithm for estimating the number of distinct elements in a data stream, closing a long line of theoretical research on this problem begun by Flajolet and Martin in their seminal paper in FOCS 1983. This problem has applications to query optimization, Internet routing, network topology, and data mining. For a stream of indices in $\{1, \ldots, n\}$, our algorithm computes a $(1 \pm \varepsilon)$-approximation using an optimal $O(\varepsilon^{-2} + \log(n))$ bits of space with $2/3$ success probability, where $0 < \varepsilon < 1$ is given. This probability can be amplified by independent repetition. Furthermore, our algorithm processes each stream update in $O(1)$ worst-case time, and can report an estimate at any point midstream in $O(1)$ worst-case time, thus settling both the space and time complexities simultaneously.

We also give an algorithm to estimate the Hamming norm of a stream, a generalization of the number of distinct elements, which is useful in data cleaning, packet tracing, and database auditing. Our algorithm uses nearly optimal space, and has optimal $O(1)$ update and reporting times.

Categories and Subject Descriptors: F.2.0 [Analysis of Algorithms and Problem Complexity]: General; H.2.m [Database Management]: Miscellaneous

General Terms: Algorithms, Theory

Keywords: distinct elements, streaming, query optimization, data mining

1. INTRODUCTION

Estimating the number of distinct elements in a data stream is a fundamental problem in network traffic monitoring, query optimization, data mining, and several other database areas. For example, this statistic is useful for selecting a minimum-cost query plan [33], database design [18], OLAP [30, 34], data integration [10, 14], and data warehousing [1].

In network traffic monitoring, routers with limited memory track statistics such as distinct destination IPs, requested URLs, and source-destination pairs on a link. Distinct elements estimation is also useful in detecting Denial of Service attacks and port scans [2, 17]. In such applications the data is too large to fit at once in main memory or too massive to be stored, being a continuous flow of data packets. This makes small-space algorithms necessary. Furthermore, the algorithm should process each stream update (i.e., packet) quickly to keep up with network speeds. For example, Estan *et al* [17] reported packet header information being produced at .5GB per hour while estimating the spread of the Code Red worm, for which they needed to estimate the number of distinct Code Red sources passing through a link.

Yet another application is to data mining: for example, estimating the number of distinct queries made to a search engine, or distinct users clicking on a link or visiting a website. Distinct item estimation was also used in estimating connectivity properties of the Internet graph [32].

We formally model the problem as follows. We see a stream $i_1, \ldots, i_m$ of indices $i_j \in [n]$, and our goal is to compute $F_0 = |\{i_1, \ldots, i_m\}|$, the number of distinct indices that appeared, using as little space as possible. Since it is known that exact or deterministic computation of F_0 requires linear space [3], we settle for computing a value $\widetilde{F}_0 \in [(1-\varepsilon)F_0, (1+\varepsilon)F_0]$ for some given $0 < \varepsilon < 1$ with probability $2/3$, over the randomness used by the algorithm. This probability can be amplified by independent repetition.

The problem of space-efficient F_0-estimation is well-studied, beginning with the work of Flajolet and Martin [20], and continuing with a long line of research, [3, 4, 5, 6, 9, 12, 16, 17, 19, 23, 24, 26, 36]. In this work, we finally settle both the space- and time-complexities of F_0-estimation by giving an algorithm using $O(\varepsilon^{-2} + \log(n))$ bits of space, with worst-case update and reporting times $O(1)$. By update time, we mean the time to process a stream token, and by reporting time, we mean the time to output an estimate of F_0 at any point in the stream. Our space upper bound matches the known lower bounds [3, 26, 36] up to a constant factor, and the $O(1)$ update and reporting times are clearly optimal. A detailed comparison of our results to those in previous work is given in Figure 1. There is a wide spectrum of time/space tradeoffs but the key points are that none of the previous

*Supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

†Supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship, and in part by the Center for Massive Data Algorithmics (MADALGO) - a center of the Danish National Research Foundation. Part of this work was done while the author was at the IBM Almaden Research Center.

PODS'10, June 6–11, 2010, Indianapolis, Indiana, USA.

Paper	Space	Update Time	Notes
[20]	$O(\log n)$	-	Assumes random oracle, constant ε
[3]	$O(\log n)$	$O(\log n)$	Only works for constant ε
[24]	$O(\varepsilon^{-2}\log n)$	$O(\varepsilon^{-2})$	
[5]	$O(\varepsilon^{-3}\log n)$	$O(\varepsilon^{-3})$	
[4]	$O(\varepsilon^{-2}\log n)$	$O(\log(\varepsilon^{-1}))$	Algorithm I in the paper
[4]	$O(\varepsilon^{-2}\log\log n + \text{poly}(\log(\varepsilon^{-1}), \log\log n)\log n)$	$\varepsilon^{-2}\,\text{poly}(\log\log n + \log(\varepsilon^{-1}))$	Algorithm II in the paper
[4]	$O(\varepsilon^{-2}(\log(\varepsilon^{-1}) + \log\log n) + \log n)$	$O(\varepsilon^{-2}(\log(\varepsilon^{-1}) + \log\log n))$	Algorithm III in the paper
[16]	$O(\varepsilon^{-2}\log\log n + \log n)$	-	Assumes random oracle, additive error
[17]	$O(\varepsilon^{-2}\log n)$	-	Assumes random oracle
[6]	$O(\varepsilon^{-2}\log n)$	$O(\log(\varepsilon^{-1}))$	
[19]	$O(\varepsilon^{-2}\log\log n + \log n)$	-	Assumes random oracle, additive error
This work	$O(\varepsilon^{-2} + \log n)$	$O(1)$	Optimal

Figure 1: Comparison of our algorithm to previous algorithms on estimating the number of distinct elements in a data stream.

algorithms achieved our optimal $O(\varepsilon^{-2}+\log n)$ bits of space, and the only ones to achieve optimal $O(1)$ update and/or reporting time had various restrictions, e.g., the assumption of access to a random oracle (that is, a truly random hash function) and/or a small constant additive error in the estimate. The best previous algorithms without any assumptions are due to Bar Yossef *et al* [4], who provide algorithms with various tradeoffs (Algorithms I, II, and III in Figure 1).

We also give a new algorithm for estimating L_0, also known as the *Hamming norm* of a vector [13], with optimal running times and near-optimal space. This problem is a generalization of F_0-estimation to the case when items can be removed from the stream. While F_0-estimation is useful for a single stream or for taking unions of streams if there are no deletions, L_0-estimation can be applied to a pair of streams to measure the number of unequal item counts. This makes it more flexible than F_0, and can be used in applications such as maintaining ad-hoc communication networks amongst cheap sensors [25]. It also has applications to data cleaning to find columns that are mostly similar [14]. Even if the rows in the two columns are in different orders, streaming algorithms for L_0 can quickly identify similar columns. As with F_0, L_0-estimation is also useful for packet tracing and database auditing [13].

Formally, in this problem there is a vector $x = (x_1, \ldots, x_n)$ which starts off as the 0 vector, and receives m updates of the form $(i, v) \in [n] \times \{-M, \ldots, M\}$ in a stream (M is some positive integer). The update (i, v) causes the change $x_i \leftarrow x_i + v$. At the end of the stream, we should output $(1\pm\varepsilon)L_0$ with probability at least 2/3, where $L_0 = |\{i : x_i \neq 0\}|$. Note that L_0-estimation is a generalization of F_0-estimation, since in the latter case an index i in the stream corresponds to the update $(i, 1)$ in an L_0-estimation problem.

We give an L_0-estimation algorithm with $O(1)$ update and reporting times, using $O(\varepsilon^{-2}\log(n)(\log(1/\varepsilon)+\log\log(mM)))$ bits of space, both of which improve upon the previously best known algorithm of Ganguly [22], which had $O(\log(1/\varepsilon))$ update time and required $O(\varepsilon^{-2}\log(n)\log(mM))$ space. Our update and reporting times are optimal, and the space is optimal up to the $\log(1/\varepsilon) + \log\log(mM)$ term due to known lower bounds [3, 27]. Furthermore, unlike with Ganguly's algorithm, our algorithm does not require that $x_i \geq 0$ for each i to operate correctly.

1.1 Overview of our algorithms

Our algorithms build upon several techniques given in previous works, with added twists to achieve our stated performance. In for example [4], it was observed that if one somehow knows ahead of time a value $R = \Theta(F_0)$, refining to $(1 \pm \varepsilon)$-approximation becomes easier. For example, [4] suggested a "balls and bins" approach to estimating F_0 given such an R. The key intuition is that when hashing A balls randomly into K bins, the number of bins hit by at least one ball is highly concentrated about its expectation, and treating this expectation as a function of A then inverting provides a good approximation to A with high probability for $A = \Theta(K)$. Then if one subsamples each index in $[n]$ with probability $2^{-\log(R/K)}$, in expectation the number of distinct items surviving is $\Theta(K)$, at which point the balls-and-bins approach can be simulated by hashing indices (the "balls") into entries of a bitvector (the "bins").

Following the above scheme, an estimate of F_0 can be obtained by running a constant-factor approximation in parallel to obtain such an R at the end of the stream, and meanwhile performing the above scheme for geometrically increasing guesses of R, one of which must be correct to within a constant factor. Thus, the bits tracked can be viewed as a bitmatrix: rows corresponding to $\log(n)$ levels of subsampling, and columns corresponding to entries in the bitvector. At the end of the stream, upon knowing R, the estimate from the appropriate level of subsampling is used. Such a scheme with $K = \Theta(1/\varepsilon^2)$ works, and gives $O(\varepsilon^{-2}\log(n))$ space, since there are $\log(n)$ levels of subsampling.

It was then first observed in [16] that, in fact, an estimator can be obtained without maintaining the full bitmatrix above. Specifically, for each column they gave an estimator that required only maintaining the deepest row with its bit set to 1. This allowed them to collapse the bitmatrix above to $O(\varepsilon^{-2}\log\log(n))$ bits. Though, their estimator and analysis required access to a purely random hash function.

Our F_0 algorithm is inspired by the above two algorithms of [4, 16]. We give a subroutine RoughEstimator using $O(\log(n))$ space which with high probability, simultaneously provides a constant-factor approximation to F_0 at all times in the stream. Previous subroutines gave a constant factor approximation to F_0 at any *particular* point in the stream with probability $1-\delta$ using $O(\log(n)\log(1/\delta))$ space; a good approximation at all times then required setting $\delta = 1/m$ to apply a union bound, thus requiring $O(\log(n)\log(m))$ space. The next observation is that if $R = \Theta(F_0)$, the largest row index with a 1 bit for any given column is heavily concen-

trated around the value $\log(F_0/K)$. Thus, if we *bitpack* the K counters and store their offsets from $\log(R/K)$, we expect to only use $O(K)$ space for all counters combined. Whenever R changes, we update all our offsets.

There are of course obvious obstacles in obtaining $O(1)$ running times, such as the occasional need to decrement all K counters (when R increases), or to locate the starting position of a counter in a bitpacked array when reading and writing entries. For the former, we use a "variable-bit-length array" data structure [7], and for the latter we use an approach inspired by the technique of deamortization of global rebuilding (see [29, Ch. 5]). Furthermore, we analyze our algorithm without assuming a truly random hash function, and show that a combination of fast k-wise independent hash functions [35] and uniform hashing [31] suffice to have sufficient concentration in all probabilistic events we consider.

Our L_0-estimation algorithm also uses subsampling and a balls-and-bins approach, but needs a different subroutine for obtaining the value R, and for representing the bitmatrix. Specifically, if one maintains each bit as a counter and tests for the counter being non-zero, frequencies of opposite sign may cancel to produce 0 and give false negatives. We instead store the dot product of frequencies in each counter with a random vector over a suitably large finite field. We remark that Ganguly's algorithm [22] is also based on a balls-and-bins approach, but on viewing the number of bins hit by *exactly* one ball (and not at least one ball), and the source of his algorithm's higher complexity stems from technical issues related to this difference.

1.2 Preliminaries

Throughout this paper, all space bounds are given in bits. We always use m to denote stream length and $[n]$ to denote the universe (the notation $[n]$ represents $\{1, \ldots, n\}$). Without loss of generality, we assume n is a power of 2, and $\varepsilon \leq \varepsilon_0$ for some fixed constant $\varepsilon_0 > 0$. In the case of L_0, M denotes an upper bound on the magnitude of updates to the x_i. We use the standard word RAM model, and running times are measured as the number of standard machine word operations (integer arithmetic, bitwise operations, and bitshifts). We assume a word size of at least $\Omega(\log(nmM))$ bits to be able to manipulate counters and indices in constant time.

For reals $A, B, \varepsilon \geq 0$, we use the notation $A = (1 \pm \varepsilon)B$ to denote that $A \in [(1-\varepsilon)B, (1+\varepsilon)B]$. We use $\mathrm{lsb}(x)$ to denote the (0-based index of) the least significant bit of a nonnegative integer x when written in binary. For example, $\mathrm{lsb}(6) = 1$. We define $\mathrm{lsb}(0) = \log(n)$. All our logarithms are base 2 unless stated otherwise. We also use $\mathcal{H}_k(U, V)$ to denote some k-wise independent hash family of functions mapping U into V. Using known constructions [11], a random $h \in \mathcal{H}_k(U, V)$ can be represented in $O(k \log(|U| + |V|))$ bits when $|U|, |V|$ are powers of 2, and computed in the same amount of space. Also, henceforth, whenever we discuss picking an $h \in \mathcal{H}_k(U, V)$, it should be understood that h is being chosen as a random element of $\mathcal{H}_k(U, V)$.

When discussing F_0, for $t \in [m]$ we use $I(t)$ to denote $\{i_1, \ldots, i_t\}$, and define $F_0(t) = |I(t)|$. We sometimes use I to denote $I(m)$ so that $F_0 = F_0(m) = |I|$. In the case of L_0-estimation, we use $I(t)$ to denote the i with $x_i \neq 0$ at time t. For an algorithm which outputs an estimate $\widetilde{F}_0$ of F_0, we let $\widetilde{F}_0(t)$ be its estimate after only seeing the first t updates (and similarly for L_0). More generally, for any variable y kept as part of the internal state of any of our algorithms, we use $y(t)$ to denote the contents of that variable at time t.

Lastly, we analyze our algorithm without any idealized assumptions, such as access to a cryptographic hash function, or to a hash function which is truly random. Our analyses all take into account the space and time complexity required to store and compute the types of hash functions we use.

2. BALLS AND BINS WITH LIMITED INDEPENDENCE

In the analysis of the correctness of our algorithms, we require some understanding of the balls and bins random process with limited independence. We note that [4] also required a similar analysis, but was only concerned with approximately preserving the expectation under bounded independence whereas we are also concerned with approximately preserving the variance. Specifically, consider throwing a set of A balls into K bins at random and wishing to understand the random variable X being the number of bins receiving at least one ball. This balls-and-bins random process can be modeled by choosing a random hash function $h \in \mathcal{H}_A([A], [K])$, i.e. h acts fully independently on the A balls, and letting $X = |i \in [K] : h^{-1}(i) \neq \emptyset|$. When analyzing our F_0 algorithm, we require an understanding of how X behaves when $h \in \mathcal{H}_k([A], [K])$ for $k \ll A$.

Henceforth, we let X_i denote the random variable indicating that at least one ball lands in bin i under a truly random hash function h, so that $X = \sum_{i=1}^{K} X_i$.

The following fact is standard.

FACT 1. $\mathbf{E}[X] = K\left(1 - \left(1 - \frac{1}{K}\right)^A\right)$

The proof of the following lemma is deferred to the full version due to space constraints.

LEMMA 1. *If* $100 \leq A \leq K/20$, *then* $\mathbf{Var}[X] < 4A^2/K$.

We now state a lemma that k-wise independence for small k suffices to preserve $\mathbf{E}[X]$ to within $1 \pm \varepsilon$, and to preserve $\mathbf{Var}[X]$ to within an additive ε^2. We note that item (1) in the following lemma was already shown in [4, Lemma 1] but with a stated requirement of $k = \Omega(\log(1/\varepsilon))$, though their proof actually seems to only require $k = \Omega(\log(1/\varepsilon)/\log\log(1/\varepsilon))$. Our proof of item (1) also only requires this k, but we require dependence on K in our proof of item (2). The proof of the following lemma is in Section A.1, and is via approximate inclusion-exclusion.

LEMMA 2. *There exists some constant* ε_0 *such that the following holds for* $\varepsilon \leq \varepsilon_0$. *Let* A *balls be mapped into* K *bins using a random* $h \in \mathcal{H}_{2(k+1)}([A], [K])$, *where* $k = c\log(K/\varepsilon)/\log\log(K/\varepsilon)$ *for a sufficiently large constant* $c > 0$. *Suppose* $1 \leq A \leq K$. *For* $i \in [K]$, *let* X_i' *be an indicator variable which is* 1 *if and only if there exists at least one ball mapped to bin* i *by* h. *Let* $X' = \sum_{i=1}^{K} X_i'$. *Then the following hold:*

(1). $|\mathbf{E}[X'] - \mathbf{E}[X]| \leq \varepsilon\mathbf{E}[X]$

(2). $\mathbf{Var}[X'] - \mathbf{Var}[X] \leq \varepsilon^2$

We now give a consequence of the above lemma.

LEMMA 3. *There exists a constant ε_0 such that the following holds. Let X' be as in Lemma 2, and also assume $100 \le A \le K/20$ with $K = 1/\varepsilon^2$ and $\varepsilon \le \varepsilon_0$. Then*

$$\mathbf{Pr}[|X' - \mathbf{E}[X]| \le 8\varepsilon\mathbf{E}[X]] \ge 4/5$$

· PROOF. Observe that

$$\begin{aligned}\mathbf{E}[X] &\ge (1/\varepsilon^2)\left(1 - \left(1 - A\varepsilon^2 + \binom{A}{2}\varepsilon^4\right)\right)\\ &= (1/\varepsilon^2)\left(A\varepsilon^2 - \binom{A}{2}\varepsilon^4\right)\\ &\ge (39/40)A,\end{aligned}$$

since $A \le 1/(20\varepsilon^2)$.

By Lemma 2 we have $\mathbf{E}[X'] \ge (1-\varepsilon)\mathbf{E}[X] > (9/10)A$, and additionally using Lemma 1 we have that $\mathbf{Var}[X'] \le \mathbf{Var}[X] + \varepsilon^2 \le 5\varepsilon^2A^2$. Set $\varepsilon' = 7\varepsilon$. Applying Chebyshev's inequality,

$$\begin{aligned}\mathbf{Pr}[|X' - \mathbf{E}[X']| \ge (10/11)\varepsilon'\mathbf{E}[X']] &\le \mathbf{Var}[X']/((10/11)^2(\varepsilon')^2\mathbf{E}^2[X'])\\ &\le 5\cdot A^2\varepsilon^2/((10/11)^2(\varepsilon')^2(9/10)^2A^2)\\ &< (13/2)\varepsilon^2/(10\varepsilon'/11)^2\\ &< 1/5\end{aligned}$$

Thus, with probability at least 1/5, by the triangle inequality and Lemma 2 we have $|X' - \mathbf{E}[X]| \le |X' - \mathbf{E}[X']| + |\mathbf{E}[X'] - \mathbf{E}[X]| \le 8\varepsilon\mathbf{E}[X]$. □

3. F_0 ESTIMATION ALGORITHM

In this section we describe our F_0-estimation algorithm. Our algorithm requires, in part, a constant-factor approximation to F_0 at every point in the stream in which F_0 is sufficiently large. We describe a subroutine ROUGHESTIMATOR in Section 3.1 which provides this, using $O(\log(n))$ space, then we give our full algorithm in Section 3.2.

We remark that the algorithm we give in Section 3.2 is space-optimal, but is not described in a way that achieves $O(1)$ worst-case update and reporting times. In Section 3.4, we describe modifications to achieve optimal running times while preserving space-optimality.

We note that several previous algorithms could give a constant-factor approximation to F_0 with success probability 2/3 using $O(\log(n))$ space. To understand why our guarantees from ROUGHESTIMATOR are different, one should pay particular attention to the quantifiers. In previous algorithms, it was guaranteed that there exists a constant $c > 0$ such that at any *particular* point t in the stream, with probability at least $1-\delta$ the output $\widetilde{F}_0(t)$ is in $[F_0(t), cF_0(t)]$, with the space used being $O(\log(n)\log(1/\delta))$. To then guarantee $\widetilde{F}_0(t) \in [F_0(t), cF_0(t)]$ *for all* $t \in [m]$ with probability 2/3, one should set $\delta = 1/(3m)$ to then union bound over all t, giving an overall space bound of $O(\log(n)\log(m))$. Meanwhile, in our subroutine ROUGHESTIMATOR, we ensure that with probability 2/3, $\widetilde{F}_0(t) \in [F_0(t), cF_0(t)]$ for all $t \in [m]$ simultaneously, and the overall space used is $O(\log(n))$.

3.1 RoughEstimator

We now show that ROUGHESTIMATOR (Figure 2) with probability $1 - o(1)$ (as $n \to \infty$) outputs a constant-factor approximation to $F_0(t)$ for every t in which $F_0(t)$ is sufficiently large. That is, if our estimate of $F_0(t)$ is $\widetilde{F}_0(t)$,

$$\mathbf{Pr}[\forall t \in [m] \text{ s.t. } F_0 \ge K_{\mathrm{RE}},\ \widetilde{F}_0(t) = \Theta(F_0(t))] = 1 - o(1),$$

where K_{RE} is as in Figure 2.

THEOREM 1. *With probability $1 - o(1)$, the output $\widetilde{F}_0$ of* ROUGHESTIMATOR *satisfies $F_0(t) \le \widetilde{F}_0(t) \le 8F_0(t)$ for every $t \in [m]$ with $F_0(t) \ge K_{\mathrm{RE}}$ simultaneously. The space used is $O(\log(n))$.*

PROOF. We first analyze space. The counters in total take $O(K_{\mathrm{RE}}\log\log(n)) = O(\log(n))$ bits. The hash functions h_1^j, h_2^j each take $O(\log(n))$ bits. The hash functions h_3^j take $O(K_{\mathrm{RE}}\log(K_{\mathrm{RE}})) = O(\log(n))$ bits.

We now analyze correctness.

LEMMA 4. *For any fixed point t in the stream with $F_0(t) \ge K_{\mathrm{RE}}$, and fixed $j \in [3]$, with probability $1 - O(1/K_{\mathrm{RE}})$ we have $F_0(t) \le \widetilde{F}_0^j(t) \le 4F_0(t)$.*

PROOF. The algorithm ROUGHESTIMATOR of Figure 2 can be seen as taking the median output of three instantiations of a subroutine, where each subroutine has K_{RE} counters $C_1, \ldots, C_{K_{\mathrm{RE}}}$, hash functions h_1, h_2, h_3, and defined quantities $T_r(t) = |\{i : C_i(t) \ge r\}|$, where $C_i(t)$ is the state of counter C_i at time t. We show that this subroutine outputs a value $\widetilde{F}_0(t) \in [F_0(t), 4F_0(t)]$ with probability $1 - O(1/K_{\mathrm{RE}})$.

Define $I_r(t) \subseteq I(t)$ as the set of $i \in I(t)$ with $\mathrm{lsb}(h_1(i)) \ge r$. Note $|I_r(t)|$ is a random variable, and

$$\mathbf{E}[|I_r(t)|] = \frac{F_0(t)}{2^r},\ \mathbf{Var}[|I_r(t)|] = \frac{F_0(t)}{2^r} - \frac{F_0(t)}{2^{2r}} \le \mathbf{E}[|I_r(t)|],$$

with the latter using 2-wise independence of h_1. Then by Chebyshev's inequality,

$$\mathbf{Pr}\left[\left||I_r(t)| - \frac{F_0(t)}{2^r}\right| \ge q\cdot\frac{F_0(t)}{2^r}\right] \le \frac{1}{q^2\cdot\mathbf{E}[|I_r(t)|]}. \quad (1)$$

Since $F_0(t) \ge K_{\mathrm{RE}}$, there exists an $r' \in [0, \log n]$ such that $K_{\mathrm{RE}}/2 \le \mathbf{E}[|I_{r'}(t)|] < K_{\mathrm{RE}}$. We condition on the event $\mathcal{E}$ that $K_{\mathrm{RE}}/3 \le I_{r'}(t) \le 4K_{\mathrm{RE}}/3$, and note

$$\mathbf{Pr}[\mathcal{E}] = 1 - O(1/K_{\mathrm{RE}})$$

by Eq. (1). We also condition on the event $\mathcal{E}'$ that for all $r'' > r' + 1$, $I_{r''}(t) \le 7K_{\mathrm{RE}}/24$. Applying Eq. (1) with $r = r' + 2$ and using that $I_{r+1}(t) \subseteq I_r(t)$,

$$\mathbf{Pr}[\mathcal{E}'] \ge 1 - O(1/K_{\mathrm{RE}}).$$

We now define two more events. The first is the event $\mathcal{E}''$ that $T_{r'}(t) \ge \rho K_{\mathrm{RE}}$. The second is the event $\mathcal{E}'''$ that $T_{r''}(t) < \rho K_{\mathrm{RE}}$ for all $r'' > r' + 1$. Note that if $\mathcal{E}'' \wedge \mathcal{E}'''$ holds, then $F_0(t) \le \widetilde{F}_0(t) \le 4F_0(t)$. We now show that these events hold with large probability.

Define the event $\mathcal{A}$ that the indices in $I_{r'}(t)$ are perfectly hashed under h_2, and the event $\mathcal{A}'$ that the indices in $I_{r'+2}(t)$ are perfectly hashed under h_2. Then

$$\mathbf{Pr}[\mathcal{A} \mid \mathcal{E}] \ge 1 - O(1/K_{\mathrm{RE}}).$$

and similarly for $\mathbf{Pr}[\mathcal{A}' \mid \mathcal{E}']$.

Note that, conditioned on $\mathcal{E} \wedge \mathcal{A}$, $T_{r'}(t)$ is distributed exactly as the number of non-empty bins when throwing $|I_{r'}(t)|$ balls uniformly at random into K_{RE} bins. This is

1. Set $K_{\mathrm{RE}} = \max\{8, \log(n)/\log\log(n)\}$.
2. Initialize $3K_{\mathrm{RE}}$ counters $C_1^j, \ldots, C_{K_{\mathrm{RE}}}^j$ to -1 for $j \in [3]$.
3. Pick random $h_1^j \in \mathcal{H}_2([n], [0, n-1])$, $h_2^j \in \mathcal{H}_2([n], [{K_{\mathrm{RE}}}^3])$, $h_3^j \in \mathcal{H}_{2K_{\mathrm{RE}}}([{K_{\mathrm{RE}}}^3], [K_{\mathrm{RE}}])$ for $j \in [3]$.
4. **Update(i):** For each $j \in [3]$, set $C^j_{h_3^j(h_2^j(i))} \leftarrow \max\left\{C^j_{h_3^j(h_2^j(i))}, \mathrm{lsb}(h_1^j(i))\right\}$.
5. **Estimator:** For integer $r \geq 0$, define $T_r^j = |\{i : C_i^j \geq r\}|$.
   For the largest $r = r^*$ with $T_r^j \geq \rho K_{\mathrm{RE}}$, set $\widetilde{F}_0^j = 2^{r^*} K_{\mathrm{RE}}$. If no such r exists, $\widetilde{F}_0^j = -1$.
   Output $\widetilde{F}_0 = \mathrm{median}\{\widetilde{F}_0^1, \widetilde{F}_0^2, \widetilde{F}_0^3\}$.

Figure 2: RoughEstimator pseudocode. With probability $1 - o(1)$, $\widetilde{F}_0(t) = \Theta(F_0(t))$ at every point t in the stream for which $F_0(t) \geq K_{\mathrm{RE}}$. The value ρ is $.99 \cdot (1 - e^{-1/3})$.

because, conditioned on $\mathcal{E} \wedge \mathcal{A}$, there are no collisions of members of $I_{r'}(t)$ under h_2, and the independence of h_3 is larger than $|I_{r'}(t)|$. Thus,

$$\mathbf{E}[T_{r'}(t) \mid \mathcal{E} \wedge \mathcal{A}] = \left(1 - \left(1 - \frac{1}{K_{\mathrm{RE}}}\right)^{|I_{r'}(t)|}\right) K_{\mathrm{RE}}.$$

The same argument applies for the conditional expectation $\mathbf{E}[T_{r''}(t) \mid \mathcal{E}' \wedge \mathcal{A}']$ for $r'' > r' + 1$. Call these conditional expectations E_r. Then since $(1-1/n)/e \leq (1-1/n)^n \leq 1/e$ for all real $n \geq 1$ (see Proposition B.3 of [28]), we have that E_r/K_{RE} lies in the interval

$$\left[\left(1 - e^{-\frac{|I_r(t)|}{K_{\mathrm{RE}}}}\right), \left(1 - e^{-\frac{|I_r(t)|}{K_{\mathrm{RE}}}}\left(1 - \frac{1}{K_{\mathrm{RE}}}\right)^{\frac{|I_r(t)|}{K_{\mathrm{RE}}}}\right)\right]$$

Thus for $r'' > r' + 1$,

$$E_{r''} \leq \left(1 - e^{-7/24}\left(1 - \frac{1}{K_{\mathrm{RE}}}\right)^{7/24}\right) K_{\mathrm{RE}}, \text{ and}$$

$$E_{r'} \geq \left(1 - e^{-1/3}\right) K_{\mathrm{RE}}.$$

A calculation shows that $E_{r''} < .99E_{r'}$ since $K_{\mathrm{RE}} \geq 8$.

By negative dependence in the balls and bins random process (see [15]), the Chernoff bound applies to $T_r(t)$ and thus

$$\mathbf{Pr}\left[|T_{r'}(t) - E_{r'}| \geq \epsilon E_{r'} \mid \mathcal{E} \wedge \mathcal{A}\right] \leq 2e^{-\epsilon^2 E_{r'}/3}$$

for any $\epsilon > 0$, and thus by taking ϵ a small enough constant,

$$\mathbf{Pr}[\mathcal{E}'' \mid \mathcal{E} \wedge \mathcal{A}] \geq 1 - e^{-\Omega(K_{\mathrm{RE}})}.$$

We also have, for $r'' > r' + 1$,

$$\mathbf{Pr}[\mathcal{E}''' \mid \mathcal{E}' \wedge \mathcal{A}'] = \mathbf{Pr}\left[T_{r''}(t) \geq \rho K_{\mathrm{RE}} \mid \mathcal{E}' \wedge \mathcal{A}'\right] \geq 1 - e^{-\Omega(K_{\mathrm{RE}})}.$$

Thus, overall,

$$\begin{aligned}
\mathbf{Pr}[\mathcal{E}'' \wedge \mathcal{E}'''] &\geq \mathbf{Pr}[\mathcal{E}'' \wedge \mathcal{E}''' \wedge \mathcal{E} \wedge \mathcal{E}' \wedge \mathcal{A} \wedge \mathcal{A}'] \\
&\geq \mathbf{Pr}[\mathcal{E}'' \wedge \mathcal{E} \wedge \mathcal{A}] + \mathbf{Pr}[\mathcal{E}''' \wedge \mathcal{E}' \wedge \mathcal{A}'] - 1 \\
&= \mathbf{Pr}[\mathcal{E}'' \mid \mathcal{E} \wedge \mathcal{A}] \cdot \mathbf{Pr}[\mathcal{A} \mid \mathcal{E}] \cdot \mathbf{Pr}[\mathcal{E}] \\
&\quad + \mathbf{Pr}[\mathcal{E}''' \mid \mathcal{E}' \wedge \mathcal{A}'] \cdot \mathbf{Pr}[\mathcal{A}' \mid \mathcal{E}'] \cdot \mathbf{Pr}[\mathcal{E}'] \\
&\quad - 1 \\
&\geq 1 - O(1/K_{\mathrm{RE}})
\end{aligned}$$

□

Now, note that for any $t \in [m]$, if $\widetilde{F}_0^j(t)$ is a 4-approximation to $F_0(t)$ for at least two values of j, then $\widetilde{F}_0(t) \in [F_0(t), 4F_0(t)]$. Thus by Lemma 4, $\widetilde{F}_0(t) \in [F_0(t), 4F_0(t)]$ with probability $1 - O(1/{K_{\mathrm{RE}}}^2)$. Let t_r be the first time in the stream when $F_0(t_r) = 2^r$ (if no such time exists, let $t_r = \infty$). Then by a union bound, our estimate of $\widetilde{F}_0(t_r)$ is in $[F_0(t_r), 4F_0(t_r)]$ at all t_r for $r \in [0, \log n]$ with probability $1 - O(\log(n)/{K_{\mathrm{RE}}}^2) = 1 - o(1)$. Now, observe that our estimates $\widetilde{F}_0(t)$ can only ever increase with t. Thus, if our estimate is in $[F_0(t_r), 4F_0(t_r)]$ at all points t_r, then it is in $[F_0(t), 8F_0(t)]$ for all $t \in [m]$. This concludes our proof.

3.2 Full algorithm

In this section we analyze our main algorithm (Figure 3), which $(1 \pm O(\varepsilon))$-approximates F_0 with 11/20 probability. We again point out that the implementation described in Figure 3 is not our final algorithm which achieves $O(1)$ update and reporting times; the final optimal algorithm is a modification of Figure 3, described in Section 3.4. We assume throughout that $F_0 \geq K/32$ and deal with the case of small F_0 in Section 3.3. The space used is $O(\varepsilon^{-2} + \log(n))$ bits. Note that the 5/8 can be boosted to $1 - \delta$ for arbitrary $\delta > 0$ by running $O(\log(1/\delta))$ instantiations of our algorithm in parallel and returning the median estimate of F_0. Also, the $O(\varepsilon)$ term in the error guarantee can be made ε by running the algorithm with $\varepsilon' = \varepsilon/C$ for a sufficiently large constant C. Throughout this section we without loss of generality assume n is larger than some constant n_0, and $1/\varepsilon^2 \geq C\log(n)$ for a constant C of our choice, and is a power of 2. If one desires a $(1 \pm \varepsilon)$-approximation for $\varepsilon > 1/\sqrt{C\log(n)}$, we simply run our algorithm with $\varepsilon = 1/\sqrt{C\log(n)}$, which worsens our promised space bound by at most a constant factor.

The algorithm of Figure 3 works as follows. We maintain $K = 1/\varepsilon^2$ counters $C_1, \ldots, C_K$ as well as three values A, b, est. Each index is hashed to some level between 0 and $\log(n)$, based on the least significant bit of its hashed value, and is also hashed to one of the counters. Each counter maintains the deepest level of an item that was hashed to it. Up until this point, this information being kept is identical as in the LogLog [16] and HyperLogLog [19] algorithms (though our analysis will not require that the hash functions be truly random). The value A keeps track of the amount of storage required to store all the C_i, and our algorithm fails if this value ever becomes much larger than a constant times K (which we show does not happen with large probability).

1. Set $K = 1/\varepsilon^2$.
2. Initialize K counters $C_1, \ldots, C_K$ to -1.
3. Pick random $h_1 \in \mathcal{H}_2([n], [0, n-1])$, $h_2 \in \mathcal{H}_2([n], [K^3])$, $h_3 \in \mathcal{H}_k([K^3], [K])$ for $k = \Omega(\log(1/\varepsilon)/\log\log(1/\varepsilon))$.
4. Initialize $A, b, \text{est} = 0$.
5. Run an instantiation RE of RoughEstimator.
6. **Update(i):** Set $x \leftarrow \max\{C_{h_3(h_2(i))}, \text{lsb}(h_1(i)) - b\}$.
   Set $A \leftarrow A - \lceil \log(2 + C_{h_3(h_2(i))}) \rceil + \lceil \log(2 + x) \rceil$.
   If $A > 3K$, Output FAIL.
   Set $C_{h_3(h_2(i))} \leftarrow x$. Also feed i to RE.
   Let R be the output of RE.

   if $R > 2^{\text{est}}$:
   (a) $\text{est} \leftarrow \log(R)$, $b_{\text{new}} \leftarrow \max\{0, \text{est} - \log(K/32)\}$.
   (b) For each $j \in [K]$, set $C_j \leftarrow \max\{-1, C_j + b - b_{\text{new}}\}$
   (c) $b \leftarrow b_{\text{new}}$, $A \leftarrow \sum_{j=1}^{K} \lceil \log(C_j + 2) \rceil$.
7. **Estimator:** Define $T = |\{j : C_j \geq 0\}|$. Output $\widetilde{F}_0 = 2^b \cdot \frac{\ln(1 - \frac{T}{K})}{\ln(1 - \frac{1}{K})}$.

Figure 3: F_0 algorithm pseudocode. With probability $11/20$, $\widetilde{F}_0 = (1 \pm O(\varepsilon))F_0$.

The value est is such that 2^{est} is a $\Theta(1)$-approximation to F_0, and is obtained via RoughEstimator, and b is such that we expect $F_0(t)/2^b$ to be $\Theta(K)$ at all points t in the stream. Each C_i then actually holds the *offset* (from b) of the deepest level of an item that was hashed to it; if no item of level b or deeper hashed to C_i, then C_i stores -1. Furthermore, the counters are bitpacked so that C_i only requires $O(1 + \log(C_i))$ bits of storage (Section 3.4 states a known data structure which allows the bitpacked C_i to be stored in a way that supports efficient reads and writes).

Theorem 2. *The algorithm of Figure 3 uses $O(\varepsilon^{-2} + \log(n))$ space.*

Proof. The hash functions h_1, h_2 each require $O(\log(n))$ bits to store. The hash function h_3 takes $O(k \log(K)) = O(\log^2(1/\varepsilon))$ bits. The value b takes $O(\log\log n)$ bits. The value A never exceeds the total number of bits to store all counters, which is $O(\varepsilon^{-2}\log(n))$, and thus A can be represented in $O(\log(1/\varepsilon) + \log\log(n))$ bits. The counters C_j never in total consume more than $O(1/\varepsilon^2)$ bits by construction, since we output FAIL if they ever would. □

Theorem 3. *The algorithm of Figure 3 outputs a value which is $(1 \pm O(\varepsilon))F_0$ with probability at least $11/20$ as long as $F_0 \geq K/32$.*

Proof. Let $\widetilde{F}_0^{\text{RE}}(t)$ be the estimate of F_0 offered by RE at time t. Throughout this proof we condition on the event $\mathcal{E}$ that $F_0(t) \leq \widetilde{F}_0^{\text{RE}}(t) \leq 8F_0(t)$ for all $t \in [m]$, which occurs with probability $1 - o(1)$ by Theorem 1.

We first show that the algorithm does not output FAIL with large probability. Note A is always $\sum_{i=1}^{K} \lceil \log(C_i + 2) \rceil$, and we must thus show that with large probability this quantity is at most $3K$ at all points in the stream. Let $A(t)$ be the value of A at time t (before running steps (a)-(c)), and similarly define $C_j(t)$. We condition on the randomness used by RE, which is independent from the remaining parts of the algorithm. Let $t_1, \ldots, t_{r-1}$ be the points in the stream where the output of RE changes, i.e $\widetilde{F}_0^{\text{RE}}(t_j - 1) \neq \widetilde{F}_0^{\text{RE}}(t_j)$ for all $j \in [r-1]$, and define $t_r = m$. We note that $A(t)$ takes on its maximum value for $t = t_j$ for some $j \in [r]$, and thus it suffices to show that $A(t_j) \leq 3K$ for all $j \in [r]$. We furthermore note that $r \leq \log(n) + 3$ since $\widetilde{F}_0^{\text{RE}}(t)$ is weakly increasing, only increases in powers of 2, and is always between 1 and $8F_0 \leq 8n$ given that $\mathcal{E}$ occurs. Now,

$$\begin{aligned} A(t) &\leq K + \sum_{i=1}^{K} \log(C_i(t) + 2) \\ &\leq K + K \cdot \log\left(\frac{\sum_{i=1}^{K} C_i(t)}{K} + 2\right) \end{aligned}$$

with the last inequality using concavity of the logarithm and Jensen's inequality. It thus suffices to show that, with large probability, $\sum_{i=1}^{K} C_i(t_j) \leq 2K$ for all $j \in [r]$.

Fix some $t = t_j$ for $j \in [r]$. For $i \in I(t)$, let $X_i(t)$ be the random variable $\max\{-1, \text{lsb}(h_1(i)) - b\}$, and let $X(t) = \sum_{i \in I(t)} X_i(t)$. Note $\sum_{i=1}^{K} C_i(t) \leq X(t)$, and thus it suffices to lower bound $\mathbf{Pr}[X(t) \leq 2K]$.

We have that $X_i(t)$ equals s with probability $1/2^{b+s+1}$ for $0 \leq s < \log(n) - b$, equals $\log(n) - b$ with probability $1/n$, and equals -1 with the remaining probability mass. Thus

$$\mathbf{E}[X(t)] \leq F_0(t) \cdot \left(1/n + \sum_{s=0}^{\log(n)-b-1} 2^{-(b+s+1)}\right) = F_0(t)/2^b.$$

Furthermore, by choice of h_1 the $X_i(t)$ are pairwise independent, and thus $\mathbf{Var}[X(t)] \leq \mathbf{E}[X(t)]$ since $\mathbf{Var}[X_i(t)] \leq \mathbf{E}[X_i(t)]$. Then by Chebyshev's inequality,

$$\mathbf{Pr}[X(t) > 2K] < \frac{F_0(t)}{2^b \cdot (2K - F_0(t)/2^b)^2}$$

Conditioned on $\mathcal{E}$, $K/256 \leq F_0(t)/2^b \leq K/32$, implying the above probability is at most $1/(32K)$. Then by a union bound over all t_j for $j \in [r]$, we have that $X(t) \leq 2K$ for all $j \in [r]$ with probability at least $1 - r/(32K) \geq 1 - 1/32$ by our assumed upper bound on ε, implying we output FAIL with probability at most $1/32$.

We now show that the output from Step 7 in Figure 3 is

$(1 \pm O(\varepsilon))F_0$ with probability 11/16. Let $\mathcal{A}$ be the algorithm in Figure 3, and let $\mathcal{A}'$ be the same algorithm, but without the third line in the update rule (i.e., $\mathcal{A}'$ never outputs FAIL). We first show that the output $\widetilde{F}_0$ of $\mathcal{A}'$ is $(1 \pm O(\varepsilon))F_0$ with probability 5/8. Let I_b be the set of indices $i \in I$ such that $\text{lsb}(i) \geq b$. Then $\mathbf{E}[|I_b|] = F_0/2^b$, and $\mathbf{Var}[|I_b|] \leq \mathbf{E}[|I_b|]$, with the last inequality in part using pairwise independence of h_1. We note that conditioned on $\mathcal{E}$, we have

$$K/256 \leq \mathbf{E}[|I_b|] \leq K/32$$

Let $\mathcal{E}'$ be the event that $K/300 \leq |I_b| \leq K/20$. Then by Chebyshev's inequality,

$$\mathbf{Pr}[\mathcal{E}' \mid \mathcal{E}] \geq 1 - O(1/K) = 1 - o(1).$$

Also, if we let $\mathcal{E}''$ be the event that I_b is perfectly hashed under h_2, then pairwise independence of h_2 gives

$$\mathbf{Pr}[\mathcal{E}'' \mid \mathcal{E}'] \geq 1 - O(1/K) = 1 - o(1).$$

Now, conditioned on $\mathcal{E}' \wedge \mathcal{E}''$, we have that T is a random variable counting the number of bins hit by at least one ball under a k-wise independent hash function, where there are $B = |I_b|$ balls, K bins, and $k = \Omega(\log(K/\varepsilon)/\log\log(K/\varepsilon))$. Then by Lemma 3, $T = (1 \pm 8\varepsilon)(1-(1-1/K)^B)K$ with 4/5 probability, in which case

$$\ln(1 - T/K) = \ln((1-1/K)^B \pm 8\varepsilon(1-(1-1/K)^B))$$

Conditioned on $\mathcal{E}'$, $(1-1/K)^B = \Theta(1)$, and thus the above is $\ln((1 \pm O(\varepsilon))(1-1/K)^B) = B\ln(1-1/K) \pm O(\varepsilon)$ since $\ln(1+x) = O(|x|)$ for $|x| < 1/2$, and thus

$$\widetilde{F}_0 = B \cdot 2^b \pm O(\varepsilon \cdot 2^b K). \tag{2}$$

Conditioned on $\mathcal{E}$, we have that $2^b \leq 256F_0/K$, and thus the error term in Eq. (2) is $O(\varepsilon F_0)$. Also, $\mathbf{E}[B] = F_0/2^b$, which is at least $K/256$ conditioned on $\mathcal{E}$. Thus by pairwise independence of h_1, Chebyshev's inequality implies

$$\mathbf{Pr}[|B - \mathbf{E}[B]| \geq c/\sqrt{K}] \leq \frac{\mathbf{E}[B]}{(c^2/K)\cdot\mathbf{E}[B]^2} \leq \left(\frac{16}{c}\right)^2$$

since $\mathbf{Var}[B] \leq \mathbf{E}[B]$, which we can make an arbitrarily small constant by setting c to be a large constant. Note that $1/\sqrt{K}$ is just ε, and thus we have that $B = (1 \pm O(\varepsilon))F_0/2^b$ with arbitrarily large constant probability.

Putting everything together, we have that, conditioned on $\mathcal{E} \wedge \mathcal{E}' \wedge \mathcal{E}''$, $\widetilde{F}_0 = (1 \pm O(\varepsilon))F_0$ with probability at least $4/5 - \delta$ for any constant $\delta > 0$ of our choice, e.g. $\delta = 1/5$. Since $\mathbf{Pr}[\mathcal{E} \wedge \mathcal{E}' \wedge \mathcal{E}''] \geq 1 - o(1)$, we thus have

$$\mathbf{Pr}[\widetilde{F}_0 = (1 \pm O(\varepsilon))F_0] \geq 3/5 - o(1).$$

Note our algorithm in Figure 3 succeeds as long as (1) we do not output FAIL, and (2) $\widetilde{F}_0 = (1 \pm O(\varepsilon))F_0$, and thus overall we succeed with probability at least $1 - \frac{2}{5} - o(1) - \frac{1}{32} > \frac{11}{20}$. $\square$

3.3 Handling small F_0

In Section 3.2, we assumed that $F_0 = \Omega(K)$ for $K = 1/\varepsilon^2$ (specifically, $F_0 \geq K/32$). In this subsection, we show how to deal with the case that F_0 is small, by running a similar (but simpler) algorithm to that of Figure 3 in parallel.

The case $F_0 < 100$ can be dealt with simply by keeping the first 100 distinct indices seen in the stream in memory, taking $O(\log(n))$ space.

For the case $F_0 \geq 100$ we can apply Lemma 3 as was done in the proof of Theorem 3. We maintain $K' = 2K$ bits $B_1, \ldots, B_{K'}$ in parallel, initialized to 0. When seeing an index i in the stream, in addition to carrying out Step 6 of Figure 3, we also set $B_{h_3(h_2(i))}$ to 1 (h_3 can be taken to have range $K' = 2K$, and its evaluation can be taken modulo K when used in Figure 3 to have a size-K range). Let t_0 be the smallest $t \in [m]$ with $F_0(t) = K'/64$, and t_1 be the smallest $t \in [m]$ with $F_0(t) = K'/32$ (if no such t_i exist, set them to ∞). Define $T_B(t) = |\{i : B_i(t) = 1\}|$, and define $\widetilde{F}_0^B(t) = \ln(1 - T_B(t)/K')/\ln(1 - 1/K')$. Then by similar calculations as in Theorem 3 and a union bound over t_0, t_1, $\mathbf{Pr}[\widetilde{F}_0^B(t_i) = (1 \pm O(\varepsilon))F_0(t_i)$ for $i \in \{0,1\}]$ is at least $1 - 2 \cdot (1/5) - o(1) = 3/5 - o(1)$. Noting that $\widetilde{F}_0^B(t)$ monotonically increases with t, we can do the following: for t with $\widetilde{F}_0^B(t) \geq K'/32 = K/16$, we output the estimator from Figure 3; else, we output $\widetilde{F}_0^B(t)$. We summarize this section with the following theorem.

THEOREM 4. *Let $\delta > 0$ be any fixed constant, and $\varepsilon > 0$ be given. There is a subroutine requiring $O(\varepsilon^{-2} + \log(n))$ space which with probability $1 - \delta$ satisfies the property that there is some $t' \in [m]$ satisfying: (1) for any fixed $t < t'$, $(1 \pm O(\varepsilon))F_0$ is output, and (2) for any $t \geq t'$ the subroutine outputs* LARGE*, and we are guaranteed $F_0(t) \geq 1/(16\varepsilon^2)$.'*

3.4 Running time

In this subsection we discuss an implementation of our F_0 algorithm in Figure 3 with $O(1)$ update and reporting times. We first state a few theorems from previous works.

THEOREM 5 (BRODNIK [8], FREDMAN AND WILLARD [21]). *The least and most significant bits of an integer fitting in a machine word can be computed in constant time.*

The next two theorems give hash families which have strong independence properties while only requiring $O(1)$ evaluation time (recall that the k-wise independent hash functions of Carter and Wegman require $\Theta(k)$ evaluation time).

THEOREM 6 (PAGH AND PAGH [31, THEOREM 1.1]). *Let $S \subseteq U = [u]$ be a set of $z > 1$ elements, and let $V = [v]$, with $1 < v \leq u$. Suppose the machine word size is $\Omega(\log(u))$. For any constant $c > 0$ there is word RAM algorithm that, using time $\log(z)\log^{O(1)}(v)$ and $O(\log(z) + \log\log(u))$ bits of space, selects a family $\mathcal{H}$ of functions from U to V (independent of S) such that:*

1. *With probability $1 - O(1/z^c)$, $\mathcal{H}$ is z-wise independent when restricted to S.*
2. *Any $h \in \mathcal{H}$ can be represented by a RAM data structure using $O(z\log(v))$ bits of space, and h can be evaluated in constant time after an initialization step taking $O(z)$ time.*

The following is a corollary of Theorem 2.16 in [35].

THEOREM 7 (SIEGEL [35]). *Let $U = [u]$ and $V = [v]$ with $u = v^c$ for some constant $c \geq 1$, where the machine word size is $\Omega(\log(v))$. Suppose one wants a $k(v)$-wise independent hash family $\mathcal{H}$ of functions mapping U to V for $k(v) = v^{o(1)}$. For any constant $\epsilon > 0$ there is a randomized procedure for constructing such an $\mathcal{H}$ which succeeds*

with probability $1 - 1/v^\epsilon$*, taking* v^ϵ *bits of space. A random* $h \in \mathcal{H}$ *can be selected using* v^ϵ *bits of random seed, and* h *can be evaluated in* $O(1)$ *time.*

We now describe a fast version of RoughEstimator.

Lemma 5. RoughEstimator *can be implemented with* $O(1)$ *worst-case update and reporting times, at the expense of only giving a* 16*-approximation to* $F_0(t)$ *for every* $t \in [m]$ *with* $F_0(t) \geq K_{\text{RE}}$*, for* K_{RE} *as in Figure 2.*

Proof. We first discuss update time. We replace each h_3^j with a random function from the hash family $\mathcal{H}$ of Theorem 6 with $z = 2K_{\text{RE}}$, $u = {K_{\text{RE}}}^3$, $v = K_{\text{RE}}$. The constant c in Item 1 of Theorem 6 is chosen to be 1, so that each h_3^j is uniform on any given subset of z items of $[u]$ with probability $1-O(1/K_{\text{RE}})$. Note that the proof of correctness of RoughEstimator (Theorem 1) only relied on the h_3^j being uniform on some unknown set of $4K_{\text{RE}}/3 < 2K_{\text{RE}}$ indices with probability $1 - O(1/K_{\text{RE}})$ (namely, those indices in $I_{r'}(t)$). The space required to store any $h \in \mathcal{H}$ is $z \log(v) = O(\log(n))$, which does not increase our space bound for RoughEstimator. Updates then require computing a least significant bit, and computing the h_1^j, h_2^j, h_3^j, all taking constant time.

For reporting time, in addition to the information maintained in Figure 2, we also maintain three sets of counters $A_0^j, A_1^j, A_2^j, A_3^j, A_4^j$ for $j \in [3]$. For a fixed j, the A_i^j store T_{r+i}^j for an r we now specify. Roughly speaking, for the values of t where $F_0(t) \geq K_{\text{RE}}$, r will be such that, conditioned on RoughEstimator working correctly, 2^r will always be in $[F_0(t)/2, 8F_0(t)]$. We then alter the estimator of RoughEstimator to being 2^{r+1}.

Note that, due to Theorem 4, the output of RoughEstimator does not figure into our final F_0-estimator until $F_0(t) \geq (1 - O(\varepsilon))/(32\varepsilon^2)$, and thus the output of the algorithm is irrelevant before this time. We start off with $r = \log(1/(32\varepsilon^2))$. Note that $A_0^j, A_1^j, A_2^j, A_3^j, A_4^j$ can be maintained in constant time during updates. At some point t_1, the estimator from Section 3.3 will declare that $F_0(t_1) = (1 \pm O(\varepsilon))/(32\varepsilon^2)$, at which point we are assured $F_0(t_1) \geq 1/(64\varepsilon^2) \geq \log(n)$ (assuming ε is smaller than some constant, and assuming that $1/\varepsilon^2 \geq 64\log(n)$). Similarly, we also have $F_0(t_1) \leq 1/(16\varepsilon^2) \leq 4\log(n)$. Thus, by our choice of r and conditioned on the event that RoughEstimator of Figure 2 succeeds (i.e., outputs a value in $[F_0(t), 8F_0(t)]$ for all t with $F_0(t) \geq K_{\text{RE}}$), we can determine the median across the j of the largest r^* such that $T_r^j \geq \rho K_{\text{RE}}$ from the A_i^j and set $r(t_1) = r^*$ so that $2^{r(t_1)}$ is in $[F_0(t_1), 8F_0(t_1)]$.

Our argument henceforth is inductive: conditioned on the output of RoughEstimator from Figure 2 being correct (always in $[F_0(t), 8F_0(t)]$), $2^{r(t)}$ will always be in $[F_0(t)/2, 8F_0(t)]$ for all $t \geq t_1$, which we just saw is true for $t = t_1$. Note that conditioned on RoughEstimator being correct, its estimate of F_0 cannot jump by a factor more than 8 at any given point in the stream. Furthermore, if this happens, we will detect it since we store up to A_4^j. Thus, whenever we find that the estimate from RoughEstimator changed (say from $2^{r'}$ to $2^{r''}$), we increment r by $r'' - r'$ and set each A_i^j to $A_{i+r''-r'}^j$ for $i \leq 4 + r' - r''$ For $4 + r' - r'' < i \leq 4$, we recompute A_i^j from scratch, by looping over the K_{RE} counters C_i. This requires $O(K_{\text{RE}})$ work, but note that since $t \geq t_1$, there must be at least K_{RE} updates before $F_0(t)$ doubles, and thus we can afford to do $O(1)$ work toward this looping per update. In the meantime 2^r cannot fall below $F_0/2$. □

We will use the following "variable-bit-length array" data structure to implement the array C of counters in Figure 3, which has entries whose binary representations may have unequal lengths. Specifically, in Figure 3, the bit representation of C_i requires $O(1 + \log(C_i + 2))$ bits.

Definition 1 (Blandford, Blelloch [7]). *A variable-bit-length array (VLA) is a data structure implementing an array* $C_1, \ldots, C_n$ *supporting the following operations: (1)* **update**(i, x) *sets the value of* C_i *to* x*, and (2)* **read**(i) *returns* C_i*. Unlike in standard arrays, the* C_i *are allowed to have bit-representations of varying lengths, and we use* $\text{len}(C_i)$ *to represent the length of the bit-representation of* C_i*.*

Theorem 8 (Blandford and Blelloch [7]). *There is a VLA data structure using* $O(n + \sum_i \text{len}(C_i))$ *space to store* n *elements, supporting worst-case* $O(1)$ *updates and reads, under the assumptions that (1)* $\text{len}(C_i) \leq w$ *for all* i*, and (2)* $w \geq \log(\mathcal{M})$*. Here* w *is the machine word size, and* $\mathcal{M}$ *is the amount of memory available to the VLA.*

We now give a time-optimal version of Figure 3.

Theorem 9. *The algorithm of Figure 3 can be implemented with* $O(1)$ *worst-case update and reporting times.*

Proof. For update time, we select h_3 from the hash family of Theorem 7, which requires $O(1/\varepsilon^\epsilon)$ space for arbitrarily small $\epsilon > 0$ of our choosing (say, $\epsilon = 1$), and thus this space is dominated by other parts of the algorithm. We then can evaluate h_1, h_2, h_3 in constant time, as well as compute the required least significant bit in constant time. Updating A requires computing the ceiling of a base-2 logarithm, but this is just a most significant bit computation which we can do in $O(1)$ time. We can also read and write the C_j in constant time whilst using the same asymptotic space by Theorem 8.

What remains is to handle the **if** statement for when $R > 2^{\text{est}}$. Note that a naïve implementation would require $O(K)$ time. Though this **if** statement occurs infrequently enough that one could show $O(1)$ amortized update time, we instead show the stronger statement that an implementation is possible with $O(1)$ worst case update time. The idea is similar to that in the proof of Lemma 5: when b_{new} changes, it cannot change more than a constant number of times again in the next $O(K)$ updates, and so we can spread the $O(K)$ required work over the next $O(K)$ stream updates, doing a constant amount of work each update.

Specifically, note that b_{new} only ever changes for times t when $R(t) > 2^{\text{est(t)}} \geq K/16$, conditioned on the subroutine of Theorem 4 succeeding, implying that $F_0(t) \geq K/256$, and thus there must be at least $K/256$ updates for $F_0(t)$ to double. Since RoughEstimator always provides an 8-approximation, est can only increase by at most 3 in the next $K/256$ stream updates. We will maintain a primary and secondary instantiation of our algorithm, and only the primary receives updates. Then in cases where $R > 2^{\text{est}}$ and b_{new} changes from b, we copy a sufficiently large constant number of the C_j (specifically, $3 \cdot 256$) for each of the next $K/256$ updates, from the primary to secondary structure, performing the update $C_j \leftarrow \max\{-1, C_j + b - b_{\text{new}}\}$ in the secondary structure. If RoughEstimator fails and est changes by more than 3 in the next $K/256$ updates, we output FAIL. Meanwhile, during this copy phase, we

process new stream updates in both the primary and secondary structures, and we answer updates from the primary structure. The analysis of correctness remains virtually unchanged, since the value 2^b corresponding to the primary structure still remains a constant-factor approximation to F_0 during this copy phase.

For reporting time, note we can maintain $T = |\{i : C_i \geq 0\}|$ during updates, and thus the reporting time is the time to compute a natural logarithm, which can be made $O(1)$ via a small lookup table (see Section A.2). □

4. L_0 ESTIMATION ALGORITHM

Here we give an algorithm for estimating L_0, the Hamming norm of a vector updated in a stream.

Our L_0 algorithm is based on the approach to F_0 estimation in Figure 4. In this approach, we maintain a $\lg(n) \times K$ bit-matrix A, and upon receiving an update i, we subsample i to the row determined by the lsb of a hash evaluation, then evaluate another hash function to tell us a column and set the corresponding bit of A to 1. Note that our algorithm from Section 3 is just a space-optimized implementation of this approach. Specifically, in Figure 3 we obtained a c-approximation R to F_0 via RoughEstimator for $c = 8$. The value b we maintained was just $\max\{0, \lg(32R/K)\}$. Then rather than explicitly maintaining A, we instead maintained counters C_j which allowed us to deduce whether $A_{b,j} = 1$ (specifically, $A_{b,j} = 1$ iff $C_j = 0$).

The proof of correctness of the approach in Figure 4 is thus essentially identical to that of Theorem 3 (in fact simpler, since we do not have to upper bound the case of outputting FAIL), so we do not repeat it here. Thus, we need only show that the approach in Figure 4 can be implemented for some constant $c \geq 1$ in the context of L_0-estimation. Specifically, we must show that (a) the bit-matrix A can be maintained (with large probability), and (b) we can implement the oracle in Step 4 of Figure 4 to give a c-approximation to L_0 for some constant $c \geq 1$.

We first show (a), that we can maintain the bit-matrix A with large probability. In fact, note our estimate of L_0 only depends on one particular row $i^* = \log(16R/K)$ of A, so we need only ensure that we maintain row i^* with large constant probability. We first give two facts.

Fact 2. *Let $t, r > 0$ be integers. Pick $h \in \mathcal{H}_2([r], [t])$. For any $S \subset [r]$, $\mathbf{E}[\sum_{i=1}^{s} \binom{|h^{-1}(i) \cap S|}{2}] \leq |S|^2/(2t)$.*

Proof. Write $|S| = s$. Let $X_{i,j}$ indicate $h(i) = j$. By linearity of expectation, the desired expectation is then

$$t \sum_{i<i'} \mathbf{E}[X_{i,1}]\mathbf{E}[X_{i',1}] = t\binom{s}{2}\frac{1}{t^2} \leq \frac{s^2}{2t}.$$

□

Fact 3. *Let $\mathbb{F}_q$ be a finite field and $v \in \mathbb{F}_q^d$ be a non-zero vector. Then, a random $w \in \mathbb{F}_q^d$ gives $\mathbf{Pr}_w[v \cdot w = 0] = 1/q$, where $v \cdot w$ is the inner product over $\mathbb{F}_q$.*

Proof. The set of vectors orthogonal to v is a linear subspace $V \subset \mathbb{F}_q^d$ of dimension $d-1$ and thus has q^{d-1} points. Thus, $\mathbf{Pr}[w \in V] = 1/q$. □

Lemma 6. *There is a scheme which represents each $A_{i,j}$ using $O(\log(1/\varepsilon) + \log\log(mM))$ bits such that, for $i^* = \log(16R/K)$, the (i^*)th row of A can be recovered with probability $2/3$. Furthermore, the update time and time to recover any $A_{i,j}$ are both $O(1)$.*

Proof. We represent each $A_{i,j}$ as a counter $B_{i,j}$ of $O(\log(K) + \log\log(mM))$ bits. We interpret $A_{i,j}$ as being the bit "1" if $B_{i,j}$ is non-zero; else we intrepret $A_{i,j}$ as 0. The details are as follows. We choose a prime p randomly in $[D, D^3]$ for $D = 100K\log(mM)$. Notice that for mM larger than some constant, by standard results on the density of primes there are at least $K^2\log^2(mM)$ primes in the interval $[D, D^3]$. Since every frequency x_i is at most mM in magnitude and thus has at most $\log(mM)$ prime factors, non-zero frequencies remain non-zero modulo p with probability $1 - O(1/K^2)$, which we condition on occurring. We also randomly pick a vector $\mathbf{u} \in \mathbb{F}_p^K$ and $h_4 \in \mathcal{H}_2([K^3], [K])$. Upon receiving an update (i, v), we increment $B_{\mathrm{lsb}(h_1(i)), h_3(h_2(i))}$ by $v \cdot \mathbf{u}_{h_4(h_2(i))}$, then reduce modulo p.

Define $I_{i^*} = \{i \in I : \mathrm{lsb}(i) = i^*\}$. Note that conditioned on $R \in [L_0, cL_0]$, we have $\mathbf{E}[I_{i^*}] \leq K/32$, and thus $\mathbf{Pr}[|I_{i^*}| \leq K/20] = 1 - O(1/K) = 1 - o(1)$ by Chebyshev's inequality. We condition on this event occurring. Also, since the range of h_2 is of size K^3, the indices in I_{i^*} are perfectly hashed with probability $1 - O(1/K) = 1 - o(1)$, which we also condition on occurring.

Let $\mathcal{Q}$ be the event that p does not divide any $|x_j|$ for $j \in I_{i^*}$. Then by a union bound, $\mathbf{Pr}[\mathcal{Q}] = 1 - O(1/K)$.

Let $\mathcal{Q}'$ be the event that $h_4(h_2(j)) \neq h_4(h_2(j'))$ for distinct $j, j' \in I_{i^*}$ with $h_3(h_2(j)) = h_3(h_2(j'))$.

Henceforth, we also condition on both $\mathcal{Q}$ and $\mathcal{Q}'$ occurring, which we later show holds with good probability. Define J as the set of $j \in [K]$ such that $h_3(h_2(i)) = j$ for at least one $i \in I_{i^*}$, so that to properly represent the $A_{i^*,j}$ we should have $B_{i^*,j}$ non-zero iff $j \in J$. For each $j \in J$, $B_{i^*,j}$ can be viewed as maintaining the dot product of a non-zero vector $\mathbf{v}$, the frequency vector x restricted to coordinates in I_{i^*} which hashed to j, with a random vector $\mathbf{w}$, namely, the projection of $\mathbf{u}$ onto coordinates in I_{i^*} that hashed to j. The vector $\mathbf{v}$ is non-zero since we condition on $\mathcal{Q}$, and $\mathbf{w}$ is random since we condition on $\mathcal{Q}'$.

Now, let $X_{i,j}$ be a random variable indicating that $h_3(h_2(j)) = h_3(h_2(j'))$ for distinct $j, j' \in I_{i^*}$. Let $X = \sum_{j<j'} X_{j,j'}$. By Fact 2 with $r = K^3$, $t = K$, and $s = |I_{i^*}| < K/20$, we have that $\mathbf{E}[X] \leq K/800$. Let $Z = \{\{j, j'\} \in \binom{I_{i^*}}{2} : h_3(h_2(j)) = h_3(h_2(j'))\}$. For $(j, j') \in Z$ let $Y_{j,j'}$ be a random variable indicating $h_4(h_2(j)) = h_4(h_2(j'))$, and let $Y = \sum_{(j,j')\in Z} Y_{j,j'}$. Then by pairwise independence of h_4, and the fact that we conditioned on I_{i^*} being perfectly hashed under h_2, we have

$$\mathbf{E}[Y] = \sum_{(j,j')\in Z} \mathbf{Pr}[h_4(h_2(j)) = h_4(h_2(j'))] = |Z|/K.$$

Note $|Z| = X$. Conditioned on $X \leq 20\mathbf{E}[X] \leq K/40$, which happens with probability at least $19/20$ by Markov's inequality, we have that $\mathbf{E}[Y] \leq |Z|/K \leq 1/40$, so that $\mathbf{Pr}[Y \geq 1] \leq 1/40$. Thus, $\mathcal{Q}'$ holds with probability at least $(19/20) \cdot (39/40) > 7/8$.

Finally, by Fact 3 with $q = p$, and union bounding over all K counters $B_{i^*,j}$, no $B_{i^*,j}$ for $j \in J$ is 0 with probability $1 - K/p \geq 99/100$. Thus, our scheme overall succeeds with probability $(7/8) \cdot (99/100) - o(1) > 2/3$. □

We next show (b) in Section A.3, i.e. give an algorithm providing an $O(1)$-approximation to L_0 with $O(1)$ update and reporting times. The space used is $O(\log(n)\log\log(mM))$.

1. Set $K = 1/\varepsilon^2$.
2. Instantiate a $\lg(n) \times K$ bit-matrix A, initializing each $A_{i,j}$ to 0.
3. Pick random $h_1 \in \mathcal{H}_2([n], [0, n-1])$, $h_2 \in \mathcal{H}_2([n], [K^3])$, $h_3 \in \mathcal{H}_k([K^3], [K])$ for $k = \Omega(\lg(1/\varepsilon)/\lg\lg(1/\varepsilon))$.
4. Obtain a value $R \in [F_0, cF_0]$ from some oracle, for some constant $c \geq 1$.
5. **Update(i):** Set $A_{\mathrm{lsb}(h_1(i)),h_3(h_2(i))} \leftarrow 1$.
6. **Estimator:** Define $T = |\{j \in [K] : A_{\log(16R/K),j} = 1\}|$. Output $\widetilde{F_0} = \frac{32R}{K} \cdot \frac{\ln(1-\frac{T}{K})}{\ln(1-\frac{1}{K})}$.

Figure 4: An algorithm skeleton for F_0 estimation.

Note that, as with our F_0 algorithm, we also need to have an algorithm which provides a $(1 \pm \varepsilon)$-approximation when $L_0 \ll 1/\varepsilon^2$. Just as in Section 3.3, this is done by handling the case of small L_0 in two cases separately: detecting and estimating when $L_0 \leq 100$, and $(1 \pm \varepsilon)$-approximating L_0 when $L_0 > 100$. In the former case, we can compute L_0 exactly with large probability by perfect hashing (see Lemma 8 in Section A.3). In the latter case, we use the same scheme as in Section 3.3, but using Lemma 6 to represent our bit array.

Putting everything together, we have the following.

THEOREM 10. *There is an algorithm for $(1\pm\varepsilon)$-approximating L_0 using space $O(\varepsilon^{-2}\log(n)(\log(1/\varepsilon) + \log\log(mM)))$, with $2/3$ success probability, and with $O(1)$ update and reporting times.*

Acknowledgments

We thank Nir Ailon, Erik Demaine, Piotr Indyk, T.S. Jayram, Swastik Kopparty, Rasmus Pagh, Mihai Pătraşcu, and the authors of [6] for valuable discussions and references.

5. REFERENCES

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua approximate query answering system. In *SIGMOD*, pages 574–576, 1999.

[2] A. Akella, A. Bharambe, M. Reiter, and S. Seshan. Detecting DDoS attacks on ISP networks. In *Proc. MPDS*, 2003.

[3] N. Alon, Y. Matias, and M. Szegedy. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[4] Z. Bar-Yossef, T.S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proc. RANDOM*, pages 1–10, 2002.

[5] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. SODA*, pages 623–632, 2002.

[6] K. S. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *SIGMOD*, pages 199–210, 2007.

[7] D. K. Blandford and G. E. Blelloch. Compact dictionaries for variable-length keys and data with applications. *ACM Trans. Alg.*, 4(2), 2008.

[8] A. Brodnik. Computation of the least significant set bit. In *Proc. ERK*, 1993.

[9] J. Brody and A. Chakrabarti. A multi-round communication lower bound for gap hamming and some consequences. In *Proc. CCC*, pages 358–368, 2009.

[10] P. Brown, P. J. Haas, J. Myllymaki, H. Pirahesh, B. Reinwald, and Y. Sismanis. Toward automated large-scale information integration and discovery. In *Data Management in a Connected World*, pages 161–180, 2005.

[11] L. Carter and M. N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[12] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.

[13] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). *IEEE Trans. Knowl. Data Eng.*, 15(3):529–540, 2003.

[14] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *SIGMOD*, pages 240–251, 2002.

[15] D. P. Dubhashi and D. Ranjan. Balls and bins: A study in negative dependence. *Random Struct. Algorithms*, 13(2):99–124, 1998.

[16] M. Durand and P. Flajolet. Loglog counting of large cardinalities (extended abstract). In *Proc. ESA*, pages 605–617, 2003.

[17] C. Estan, G. Varghese, and M. E. Fisk. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Trans. Netw.*, 14(5):925–937, 2006.

[18] S. J. Finkelstein, M. Schkolnick, and P. Tiberio. Physical database design for relational databases. *ACM Trans. Database Syst.*, 13(1):91–128, 1988.

[19] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. *Disc. Math. and Theor. Comp. Sci.*, AH:127–146, 2007.

[20] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[21] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993.

[22] S. Ganguly. Counting distinct items over update streams. *Theor. Comput. Sci.*, 378(3):211–222, 2007.

[23] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB*, pages 541–550, 2001.

[24] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proc. SPAA*, pages 281–291, 2001.

[25] P. Indyk. Algorithms for dynamic geometric problems over data streams. In *Proc. STOC*, pages 373–380, 2004.

[26] P. Indyk and D. P. Woodruff. Tight lower bounds for the distinct elements problem. In *Proc. FOCS*, pages 283–, 2003.

[27] D. M. Kane, J. Nelson, and D. P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proc. SODA*, pages 1161–1178, 2010.

[28] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[29] M. H. Overmars. *The Design of Dynamic Data Structures*. Springer, 1983.

[30] S. Padmanabhan, B. Bhattacharjee, T. Malkemus, L. Cranston, and M. Huras. Multi-dimensional clustering: A new data layout scheme in db2. In *SIGMOD*, pages 637–641, 2003.

[31] A. Pagh and R. Pagh. Uniform hashing in constant time and optimal space. *SIAM J. Comput.*, 38(1):85–96, 2008.

[32] C. R. Palmer, G. Siganos, M. Faloutsos, and C. Faloutsos. The connectivity and fault-tolerance of the internet topology. In *NRDM Workshop*, 2001.

[33] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD*, pages 23–34, 1979.

[34] A. Shukla, P. Deshpande, J. F. Naughton, and K. Ramasamy. Storage estimation for multidimensional aggregates in the presence of hierarchies. In *Proc. VLDB*, pages 522–531, 1996.

[35] A. Siegel. On universal classes of extremely random

constant-time hash functions. *SIAM J. Computing*, 33(3):505–543, 2004.

[36] D. P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proc. SODA*, pages 167–175, 2004.

APPENDIX

A. APPENDIX

A.1 Expectation and variance analyses for balls and bins with limited independence

The following is a proof of Lemma 2. Below, X corresponds to the random variable which is the number of bins receiving at least one ball when tossing A balls into K independently at random.

Lemma 2 (restatement). *There exists some constant ε_0 such that the following holds for $\varepsilon \leq \varepsilon_0$. Let A balls be mapped into K bins using a random $h \in \mathcal{H}_{2(k+1)}([A],[K])$, where $k = c\log(K/\varepsilon)/\log\log(K/\varepsilon)$ for a sufficiently large constant $c > 0$. Suppose $1 \leq A \leq K$. For $i \in [K]$, let X'_i be an indicator variable which is 1 if and only if there exists at least one ball mapped to bin i by h. Let $X' = \sum_{i=1}^{K} X'_i$. Then the following hold:*

(1). $|\mathbf{E}[X'] - \mathbf{E}[X]| \leq \varepsilon\mathbf{E}[X]$

(2). $\mathbf{Var}[X'] - \mathbf{Var}[X] \leq \varepsilon^2$

PROOF. Let A_i be the random variable number counting the number of balls in bin i when picking $h \in \mathcal{H}_{2(k+1)}([A],[K])$. Define the function:

$$f_k(n) = \sum_{i=0}^{k} (-1)^i \binom{n}{i}$$

We note that $f_k(0) = 1$, $f_k(n) = 0$ for $1 \leq n \leq k$ and $|f_k(n)| \leq \binom{n}{k+1}$ otherwise. Let $f(n) = 1$ if $n = 0$ and 0 otherwise. We now approximate X_i as $1 - f_k(A_i)$. We note that this value is determined entirely by k-wise independence of h. We note that this is also

$$1 - f(A_i) \pm O\left(\binom{A_i}{k+1}\right) = X_i \pm O\left(\binom{A_i}{k+1}\right).$$

The same expression holds for the X'_i, and thus both $\mathbf{E}[X'_i]$ and $\mathbf{E}[X_i]$ are sandwiched inside an interval of size bounded by twice the expected error. To bound the expected error we can use $(k+1)$-independence. We have that the expected value of $\binom{A_i}{k+1}$ is $\binom{A}{k+1}$ ways of choosing $k+1$ of the balls times the product of the probabilities that each ball is in bin i. This is

$$\binom{A}{k+1} K^{-(k+1)} \leq \left(\frac{eA}{K(k+1)}\right)^{k+1} \leq \frac{A}{K} \cdot (e(k+1))^{-(k+1)},$$

with the last inequality using that $A \leq K$. Thus, $|\mathbf{E}[X_i] - \mathbf{E}[X'_i]| \leq \varepsilon^2 A/K$ for $k = c\log(1/\varepsilon)/\log\log(1/\varepsilon)$ for sufficiently large constant c. In this case $|\mathbf{E}[X] - \mathbf{E}[X']| \leq \varepsilon^2 A \leq \varepsilon\mathbf{E}[X]$ for ε smaller than some constant since $\mathbf{E}[X] = \Omega(A)$ for $A \leq K$.

We now analyze $\mathbf{Var}[X']$. We approximate X_iX_j as $(1 - f_k(A_i))(1 - f_k(A_j))$. This is determined by $2k$-independence of h and is equal to

$$\left(1 - f(A_i) \pm O\left(\binom{A_i}{k+1}\right)\right)\left(1 - f(A_j) \pm O\left(\binom{A_j}{k+1}\right)\right)$$
$$= X_iX_j \pm O\left(\binom{A_i}{k+1} + \binom{A_j}{k+1} + \binom{A_i}{k+1}\binom{A_j}{k+1}\right)$$

We can now analyze the error using $2(k+1)$-wise independence. The expectation of each term in the error is calculated as before, except for products of the form

$$\binom{A_i}{k+1}\binom{A_j}{k+1}.$$

The expected value of this is

$$\binom{A}{k+1, k+1} K^{-2(k+1)} \leq \binom{A}{k+1}^2 K^{-2(k+1)} \leq \left(\frac{eA}{K(k+1)}\right)^{2(k+1)}.$$

Thus, for $k = c'\log(K/\varepsilon)/\log\log(K/\varepsilon)$ for sufficiently large $c' > 0$, each summand in the error above is bounded by $\varepsilon^3/(6K^2)$, in which case $|\mathbf{E}[X_iX_j] - \mathbf{E}[X_iX_j]| \leq \varepsilon^3/K^2$. We can also make c' sufficiently large so that $|\mathbf{E}[X] - \mathbf{E}[X']| \leq \varepsilon^3/K^2$. Now, we have

$$\begin{aligned}
&\mathbf{Var}[X'] - \mathbf{Var}[X] \\
&\quad\leq |(\mathbf{E}[X] - \mathbf{E}[X']) + 2\sum_{i<j}(\mathbf{E}[X_iX_j] - \mathbf{E}[X'_iX'_j]) \\
&\qquad\qquad - (\mathbf{E}^2[X] - \mathbf{E}^2[X'])| \\
&\quad\leq |\mathbf{E}[X] - \mathbf{E}[X']| \\
&\qquad + K(K-1)\cdot\max_{i<j}|\mathbf{E}[X_iX_j] - \mathbf{E}[X'_iX'_j]| \\
&\qquad\qquad + |\mathbf{E}^2[X] - \mathbf{E}^2[X']| \\
&\quad\leq \varepsilon^3/K^2 + \varepsilon^3 + \mathbf{E}^2[X](2\varepsilon^3/K^2 + (\varepsilon^3/K^2)^2) \\
&\quad\leq 5\varepsilon^3
\end{aligned}$$

which is at most ε^2 for ε sufficiently small. □

A.2 A compact lookuptable for the natural logarithm

LEMMA 7. *Let $K > 4$ be a positive integer, and write $\gamma = 1/\sqrt{K}$. It is possible to construct a lookup table requiring $O(\gamma^{-1}\log(1/\gamma))$ bits such that $\ln(1 - c/K)$ can then be computed with relative accuracy γ in constant time for all integers $c \in [4K/5]$.*

PROOF. We set $\gamma' = \gamma/15$ and discretize the interval $[1, 4K/5]$ geometrically by powers of $(1+\gamma')$. We precompute the natural algorithm evaluated at $1 - \rho/K$ for all discretization points ρ, with relative error $\gamma/3$, creating a table A taking space $O(\gamma^{-1}\log(1/\gamma))$. We answer a query $\ln(1 - c/K)$ by outputting the natural logarithm of the closest discretization point in A. First, we argue that the error from this output is small enough. Next, we argue that the closest discretization point can be found in constant time.

For the error, the output is up to $(1 \pm \gamma/3)$,

$$\begin{aligned}\ln(1-(1\pm\gamma')c/K) &= \ln(1-c/K \pm \gamma' c/K)\\ &= \ln(1-c/K) \pm 5\gamma' c/K\\ &= \ln(1-c/K) \pm \gamma c/(3K).\end{aligned}$$

Using the fact that $|\ln(1-z)| \geq z/(1-z)$ for $0 < z < 1$, we have that $|\ln(1-c/K)| \geq c/(K-c) \geq c/K$. Thus,

$$\begin{aligned}(1\pm\gamma/3)(\ln(1-c/(3K)) \pm \gamma c/K) &= (1\pm\gamma/3)^2 \ln(1-c/K)\\ &= (1\pm\gamma)\ln(1-c/K).\end{aligned}$$

Now, for finding the discretization point, note we need to look up $A[\lceil \log_{1+\gamma'}(c)\rceil] = A[\lceil \log(c)/(a\gamma')\rceil]$, where $a\gamma' = \log(1+\gamma')$ (note, we can compute $\log(1+\gamma') = a\gamma'$ in preprocessing). Now, write $c = d \cdot 2^k$ where $k = \lfloor \log(c)\rfloor$ and thus $1 \leq d < 2$. We can compute k in $O(1)$ time since it is the most significant bit of c. We know $\log_{1+\gamma'}(c) = \log(d\cdot 2^k)/(a\gamma') = k/(a\gamma') + \log(d)/(a\gamma')$. Now, the derivative of the log function in the range $[1,2)$ is sandwiched between two constants. Thus, if we discretize $[1,2)$ evenly into $O(\gamma'^{-1})$ buckets and store the log of a representative of each bucket in a lookup table B, we can additively $O(\gamma')$-approximate $\log(d)$ by table lookup of $B[\lfloor (d-1)/\gamma'\rfloor]$. So now we have computed

$$\begin{aligned}&k/(a\gamma') + (\log(d) + O(\gamma'))/(a\gamma')\\ &\quad = k/(a\gamma') + \log(d)/(a\gamma') \pm O(1).\end{aligned}$$

This $O(1)$ can be taken to be arbitrarily small, say at most $1/3$, by tuning the constant in the discretization. So we know the correct index to look at in our index table A up to $\pm 1/3$; since indices are integers, we are done. □

A.3 A Rough Estimator for L_0-estimation

We describe here a subroutine RoughL0Estimator which gives a constant-factor approximation to L_0 with probability $9/16$. First, we need the following lemma which states that when L_0 is at most some constant c, it can be computed exactly in small space. The lemma follows by picking a random prime $p = \Theta(\log(mM)\log\log(mM))$ and pairwise independently hashing the universe into $[\Theta(c^2)]$ buckets. Each bucket is a counter which tracks the sum of frequencies modulo p of updates to universe items landing in that bucket. The estimate of L_0 is then the total number of non-zero counters, and the maximum estimate after $O(\log(1/\eta))$ trials is finally output. This gives the following.

Lemma 8. *There is an algorithm which, when given the promise that $L_0 \leq c$, outputs L_0 exactly with probability at least $1-\eta$ using $O(c^2 \log\log(mM))$ space, in addition to needing to store $O(\log(1/\eta))$ independently chosen pairwise independent hash functions mapping $[n]$ into $[c^2]$. The update and reporting times are $O(1)$.*

Now we describe RoughL0Estimator. We pick a function $h : [n] \to [n]$ at random from a pairwise independent family. For each $0 \leq j \leq \log(n)$ we create a substream $\mathcal{S}^j$ consisting of those $x \in [n]$ with $\mathrm{lsb}(h(x)) = j$. Let $L_0(\mathcal{S})$ denote L_0 of the substream $\mathcal{S}$. For each $\mathcal{S}^j$ we run an instantiation B_j of Lemma 8 with $c = 141$ and $\eta = 1/16$. All instantiations share the same $O(\log(1/\eta))$ hash functions $h^1, \ldots, h^{O(\log(1/\eta))}$.

To obtain our final estimate of L_0 for the entire stream, we find the largest value of j for which B^j declares $L_0(\mathcal{S}^j) > 8$. Our estimate of L_0 is $\tilde{L}_0 = 2^j$. If no such j exists, we estimate $\tilde{L}_0 = 1$.

Theorem 11. RoughL0Estimator *with probability at least $9/16$ outputs a value $\tilde{L}_0$ satisfying $L_0 \leq \tilde{L}_0 \leq 110 L_0$. The space used is $O(\log(n)\log\log(mM))$, and the update and reporting times are $O(1)$.*

Proof. The space to store h is $O(\log n)$. The $\Theta(\log(1/\eta))$ hash functions h^i in total require $O(\log(1/\eta)\log n) = O(\log n)$ bits to store since $1/\eta = O(1)$. The remaining space to store a single B^j for a level is $O(\log\log(mM))$ by Lemma 8, and thus storing all B^j across all levels requires $O(\log(n)\log\log(mM))$ space.

As for running time, upon receiving a stream update (x, v), we first hash x using h, taking time $O(1)$. Then, we compute $\mathrm{lsb}(h(x))$, also in constant time. Now, given our choice of η for B^j, we can update B^j in $O(1)$ time by Lemma 8.

To obtain $O(1)$ reporting time, we again use the fact that we can compute the least significant bit of a machine word in constant time. We maintain a single machine word z of at least $\log(n)$ bits and treat it as a bit vector. We maintain that the jth bit of z is 1 iff $L_0(\mathcal{S}^j)$ is reported to be greater than 8 by B^j. This property can be maintained in constant time during updates. Constant reporting time then follows since finding the deepest level j with greater than 8 reported elements is equivalent to computing $\mathrm{lsb}(z)$.

Now we prove correctness. Observe that $\mathbf{E}[L_0(\mathcal{S}^j)] = L_0/2^{j+1}$ when $j < \log n$ and $\mathbf{E}[L_0(\mathcal{S}^j)] = L_0/2^j = L_0/n$ when $j = \log n$. Let j^* be the largest j satisfying $\mathbf{E}[L_0(\mathcal{S}^j)] \geq 1$ and note that $1 \leq \mathbf{E}[L_0(\mathcal{S}^{j^*})] \leq 2$. For any $j > j^*$, $\mathbf{Pr}[L_0(\mathcal{S}^j) > 8] < 1/(8\cdot 2^{j-j^*-1})$ by Markov's inequality. Thus, by a union bound, the probability that any $j > j^*$ has $L_0(\mathcal{S}^j) > 8$ is at most $(1/8)\cdot\sum_{j-j^*=1}^{\infty} 2^{-(j-j^*-1)} = 1/4$. Now, let $j^{**} < j^*$ be the largest j such that $\mathbf{E}[L_0(\mathcal{S}^j)] \geq 55$, if such a j exists. Since we increase the j by powers of 2, we have $55 \leq \mathbf{E}[L_0(\mathcal{S}^{j^{**}})] < 110$. Note that h is pairwise independent, so $\mathbf{Var}[L_0(\mathcal{S}^{j^{**}})] \leq \mathbf{E}[L_0(\mathcal{S}^{j^{**}})]$. For this range of $\mathbf{E}[L_0(\mathcal{S}^{j^{**}})]$, we then have by Chebyshev's inequality that

$$\mathbf{Pr}\left[|L_0(\mathcal{S}^{j^{**}}) - \mathbf{E}[L_0(\mathcal{S}^{j^{**}})]| \geq 3\sqrt{\mathbf{E}[L_0(\mathcal{S}^{j^{**}})]}\right] \leq 1/9.$$

If $|L_0(\mathcal{S}^{j^{**}}) - \mathbf{E}[L_0(\mathcal{S}^{j^{**}})]| < 3\sqrt{\mathbf{E}[L_0(\mathcal{S}^{j^{**}})]}$, then

$$32 < 55 - 3\sqrt{55} < L_0(\mathcal{S}^{j^{**}}) < 110 + 3\sqrt{110} < 142$$

since $55 \leq \mathbf{E}[L_0(\mathcal{S}^{j^{**}})] < 110$.

So far we have shown that with probability at least $3/4$, $L_0(\mathcal{S}^j) \leq 8$ for all $j > j^*$. Thus, for these j the B^j will estimate L_0 of the corresponding substreams to be at most 8, and we will not output $\tilde{L}_0 = 2^j$ for $j > j^*$. On the other hand, we know for j^{**} (if it exists) that with probability at least $8/9$, $\mathcal{S}^{j^{**}}$ will have $32 < L_0(\mathcal{S}_i^{j^{**}}) < 142$. By our choice of $c = 141$ and $\eta = 1/16$ in the B^j, $B^{j^{**}}$ will output a value $\tilde{L}_0(\mathcal{S}_i^{j^{**}}) \geq L_0(\mathcal{S}_i^{j^{**}})/4 > 8$ with probability at least $1 - (1/9 + 1/16) > 13/16$ by Lemma 8. Thus, with probability at least $1-(3/16+1/4) = 9/16$, we output $\tilde{L}_0 = 2^j$ for some $j^{**} \leq j \leq j^*$, which satisfies $110\cdot 2^j < L_0 \leq 2^j$. If such a j^{**} does not exist, then $L_0 < 55$, and 1 is a 55-approximation in this case. □

Time Lower Bounds for Nonadaptive Turnstile Streaming Algorithms

Kasper Green Larsen*
Aarhus University
larsen@cs.au.dk

Jelani Nelson†
Harvard University
minilek@seas.harvard.edu

Huy L. Nguyễn
Simons Institute
hlnguyen@cs.princeton.edu

ABSTRACT

We say a turnstile streaming algorithm is *non-adaptive* if, during updates, the memory cells written and read depend only on the index being updated and random coins tossed at the beginning of the stream (and not on the memory contents of the algorithm). Memory cells read during *queries* may be decided upon adaptively. All known turnstile streaming algorithms in the literature, except a single recent example for a particular *promise problem* [7], are non-adaptive. In fact, even more specifically, they are all linear sketches.

We prove the first non-trivial update time lower bounds for both randomized and deterministic turnstile streaming algorithms, which hold when the algorithms are non-adaptive. While there has been abundant success in proving space lower bounds, there have been no non-trivial turnstile update time lower bounds. Our lower bounds hold against classically studied problems such as heavy hitters, point query, entropy estimation, and moment estimation. In some cases of deterministic algorithms, our lower bounds nearly match known upper bounds.

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General

General Terms

Theory, Algorithms

Keywords

streaming; time lower bounds; cell probe model; heavy hitters; point query

*Supported by Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation, grant DNRF84.

†Supported by NSF grant IIS-1447471 and NSF CAREER award CCF-1350670, ONR grant N00014-14-1-0632, and a Google Faculty Research Award.

STOC'15, June 14–17, 2015, Portland, Oregon, USA.

http://dx.doi.org/10.1145/2746539.2746542.

1. INTRODUCTION

In the turnstile streaming model of computation [36] there is some vector $v \in \mathbb{R}^n$ initialized to $\vec{0}$, and we must provide a data structure that processes coordinate-wise updates to v. An update of the form (i, Δ) causes the change $v_i \leftarrow v_i + \Delta$, where $\Delta \in \{-M, \ldots, M\}$. Occasionally our data structure must answer queries for some function of v. In many applications n is extremely large, and thus it is desirable to provide a data structure with space consumption much less than n, e.g. polylogarithmic in n. For example, n may be the number of valid IP addresses ($n = 2^{128}$ in IPv6), and v_i may be the number of packets sent from source IP address i on a particular link. A query may then ask for the support size of v (the "distinct elements" problem [14]), which was used for example to estimate the spread of the Code Red worm after filtering the packet stream based on the worm's signature [13, 35]. Another query may be the ℓ_2 norm of v [2], which was used by AT&T as part of a packet monitoring system [31, 42]. In some examples there is more than one possible query to be asked; in "point query" problems a query is some $i \in [n]$ and the data structure must output v_i up to some additive error, e.g. $\varepsilon\|v\|_p$ [6, 12]. Such point query data structures are used as subroutines in the heavy hitters problem, where informally the goal is to output all i such that v_i is "large". If the data structure is linearly composable (meaning that data structures for v and v' can be combined to form a data structure for $v - v'$), heavy hitters data structures can be used for example to detect trending topics in search engine query streams [20, 21, 6]. In fact the point query structure [6] has been implemented in the log analysis language Sawzall at Google [41].

Coupled with the great success in providing small-space data structures for various turnstile streaming problems has been a great amount of progress in proving space lower bounds, i.e. theorems which state that *any* data structure for some particular turnstile streaming problem must use space (in bits) above some lower bound. For example, tight or nearly tight space lower bounds are known for the distinct elements problem [2, 43, 44, 25], ℓ_p norm estimation [2, 3, 5, 43, 29, 22, 24, 26, 44, 25], heavy hitters [27], entropy estimation [4, 29], and several other problems.

While there has been much previous work on understanding the space required for solving various streaming problems, much less progress has been made regarding time complexity: the time it takes to process an update in the stream and the time it takes to answer a query about the stream. This is despite strong motivation, since in several applications the data stream may be updated at an extremely fast

rate, so that in fact update time is often arguably *more important* than space consumption; for example, [42] reported in their application for ℓ_2-norm estimation that their system was constrained to spend only 130 nanoseconds per packet to keep up with high network speeds. If for example n is the number of IP addresses 2^{128}, then certainly the (computer connected to the) router has much more than $\lg n$ bits of memory or even cache, which is a regime much previous streaming literature has focused on. Arguably this primary focus on space was not because of the greater *practical* interest in space complexity, but simply because up until this point there was a conspicuous absence of understanding concerning the tradeoff between update time and memory consumption (especially from the perspective of lower bounds). In short, many previous research targets were defined by what our tools allowed us to prove. In fact, in high-speed applications such as in networking, one could easily imagine preferring a solution using n^ϵ memory (as long as it fit in cache) with a very fast update time than, say, polylogarithmic memory with worse update time.

Of course, without any space constraint achieving fast update and query times is trivial: store v and its norm in memory explicitly and spend constant time to add Δ to v_i and change the stored norm after each update. Thus an interesting data structural issues arises: how can we simultaneously achieve small space and low time for turnstile streaming data structures? As mentioned, surprisingly very little is understood about this question for any turnstile streaming problem. For some problems we have very fast algorithms (e.g. constant update time for distinct elements [30], and also for ℓ_2 estimation [42]), whereas for others we do not (e.g. super-constant time for ℓ_p estimation for $p \neq 2$ [28] and heavy hitters problems [6, 12]), and we do not have proofs precluding the possibility that a fast algorithm exists. Indeed, the only previous time lower bounds for streaming problems are those of Clifford and Jalsenius [8] and Clifford, Jalsenius, and Sach [9, 10] for streaming multiplication, streaming edit distance and streaming Hamming distance computation. These problems are significantly harder than e.g. ℓ_p estimation (the lower bounds proved apply even for super-linear space usage) and there appears to be no way of extending their approach to obtain lower bounds for the problems above. Importantly, the problems considered in [8, 9, 10] are not in the turnstile model.

A natural model for upper bounds in the streaming literature (and also for data structures in general), is the word RAM model: basic arithmetic operations on machine words of w bits each cost one unit of time. In the data structure literature, one of the strongest lower bounds that can be proven is in the cell probe model [34, 15, 45], where one assumes that the data structure is broken up into S words each of size w bits and cost is only incurred when the user reads a memory cell from the data structure (S is the space). Lower bounds proven in this model hold against any algorithm operating in the word RAM model. In recent years there has been much success in proving data structure lower bounds in the cell probe model (see for example [39, 33]) using techniques from communication complexity and information theory. Can these techniques be imported to obtain time lower bounds for clasically studied turnstile streaming problems?

QUESTION 1. *Can we use techniques from cell probe lower bound proofs to lower bound the time complexity for classical streaming problems?*

Indeed the lower bounds for streaming multiplication and Hamming distance computation were proved using the *information transfer* technique of Pătraşcu and Demaine [40], but this approach does not seem effective against the turnstile streaming problems above.

There are a couple of obvious problems to attack for Question 1. For example, for many streaming problems one can move from, say, 2/3 success probability to $1-\delta$ success probability by running $\Theta(\lg(1/\delta))$ instantiations of the algorithm in parallel then outputting the median answer. This is possible whenever the output of the algorithm is numerical, such as for example distinct elements, moment estimation, entropy estimation, point query, or several other problems. Note that doing so increases both the space and update time complexity of the resulting streaming algorithm by a $\Theta(\lg(1/\delta))$ factor. Is this necessary? It was shown that the blowup in *space* is necessary for several problems by Jayram and Woodruff [26], but absolutely nothing is known about whether the blowup in time is required. Thus, any non-trivial lower bound in terms of δ would be novel.

Another problem to try for Question 1 is the heavy hitters problem. Consider the ℓ_1 heavy hitters problem: a vector v receives turnstile updates, and during a query we must report all i such that $|v_i| \geq \varepsilon\|v\|_1$. Our list is allowed to contain some false positives, but not "too false", in the sense that any i output should at least satisfy $|v_i| \geq (\varepsilon/2)\|v\|_1$. Algorithms known for this problem using non-trivially small space, both randomized [12] and deterministic [19, 37], require $\tilde{\Omega}(\lg n)$ update time. Can we prove a matching lower bound?

Our Results.

In this paper we present the first update time lower bounds for a range of classical turnstile streaming problems. The lower bounds apply to a restricted class of randomized streaming algorithms that we refer to as randomized *non-adaptive* algorithms. We say that a randomized streaming algorithm is *non-adaptive* if:

- Before processing any elements of the stream, the algorithm may toss some random coins.
- The memory words read/written to (henceforth jointly referred to as probed) on any update operation (i, Δ) are completely determined from the index i and the initially tossed random coins.

Thus a non-adaptive algorithm may not decide which memory words to probe based on the current contents of the memory words. Note that in this model of non-adaptivity, the random coins can encode e.g. a hash function chosen (independent of the stream) from a desirable family of hash functions and the update algorithm can choose what to probe based on the hash function. It is only the input-specific contents of the probed words that the algorithm may not use to decide which words to probe. To the best of our knowledge, all the known algorithms for classical turnstile streaming problems are indeed *non-adaptive*, in particular *linear sketches* (i.e. maintaining Πv in memory for some $r \times n$ matrix Π, $r \ll n$). One exception we are aware of is an

adaptive algorithm given for a *promise problem* in [7] (it is also worth mentioning that for the non-promise version of the problem, the algorithm given in the same work is again non-adaptive). We further remark that our lower bounds only require the update procedure to be non-adaptive and still apply if adaptivity is used to answer queries.

REMARK 2. One common technique in turnstile streaming (e.g. see [28]) is to batch some number of updates then process them all more time-efficiently once the batch is large enough. In all examples we are aware of using this technique, a time savings is gained solely because batching allows for faster evaluation of the hash functions involved (e.g. using fast multipoint evaluation of polynomials). Thus, such techniques have only ever found application to decreasing word RAM time complexity, but have thus far never been effective in decreasing cell probe complexity, which is what our current work here lower bounds. Thus, in all these cases, there is a non-adaptive algorithm achieving the best known cell probe complexity.

For the remainder of the paper, for ease of presentation we assume that the update increments are (possibly negative) integers bounded by some $M \leq \text{poly}(n)$ in magnitude, and that the number of updates in the stream is also at most $\text{poly}(n)$. We further assume the trans-dichotomous model [16, 17], i.e. that the machine word size w is $\Theta(\lg n)$ bits. This is a natural assumption, since typically the streaming literature assumes that basic arithmetic operations on a value $|v_i|$, the index of the current position in the stream, or an index i into v can be performed in constant time.

We prove lower bounds for the following types of queries in turnstile streams. For each problem listed, the query function takes no input (other than point query, which takes an input $i \in [n]$). Each query below is accompanied by a description of what the data structure should output.

- ℓ_1 **heavy hitter:** return an index i such that $|v_i| \geq \|v\|_\infty - \|v\|_1/2$.
- **Point query:** given i at query time, returns $v_i \pm \|v\|_1/2$.
- ℓ_p/ℓ_q **norm estimation ($1 \leq q \leq p \leq \infty$):** returns $\|v\|_p \pm \|v\|_q/2$.
- **Entropy estimation.** returns a 2-approximation of the entropy of the distribution which assigns probability $|v_i|/\|v\|_1$ to i for each $i \in [n]$.

The lower bounds we prove for non-adaptive streaming algorithms are as follows ($n^{-O(1)} \leq \delta \leq 1/2 - \Omega(1)$ is the failure probability of a query):

- Any randomized non-adaptive streaming algorithm for point query, ℓ_p/ℓ_q estimation with $1 \leq q \leq p \leq \infty$, and entropy estimation, must have worst case update time $t_u = \Omega(\frac{\lg(1/\delta)}{\sqrt{\lg n \lg(eS/t_u)}})$.

  We also show that any *deterministic* non-adaptive streaming algorithm for the same problems must have worst case update time $t_u = \Omega(\lg n/\lg(eS/t_u))$.

- Any randomized non-adaptive streaming algorithm for ℓ_1 heavy hitters, must have worst case update time $t_u = \Omega(\min\{\sqrt{\frac{\lg(1/\delta)}{\lg(eS/t_u)}}, \frac{\lg(1/\delta)}{\sqrt{\lg t_u \cdot \lg(eS/t_u)}}\})$. Any deterministic and non-adaptive streaming algorithm for ℓ_1 heavy hitters must have worst case update time $t_u = \Omega(\frac{\lg n}{\lg(eS/t_u)})$.

REMARK 3. The deterministic lower bound above for point query matches two previous upper bounds for point query [37], which use error-correcting codes to yield deterministic point query data structures. Specifically, for space $S = O(\lg n)$, our lower bound implies $t_u = \Omega(\lg n)$, matching an upper bound based on random codes. For space $O((\lg n/\lg\lg n)^2)$, our lower bound is $t_u = \Omega(\lg n/\lg\lg n)$, matching an upper bound based on Reed-Solomon codes. Similarly, the deterministic bound above for deterministic ℓ_2/ℓ_1 norm estimation matches the previous upper bound for this problem [37], showing that for the optimal space $S = \Theta(\lg n)$, the fastest query time of non-adaptive algorithms is $t_u = \Theta(\lg n)$.

These deterministic upper bounds are also in the cell probe model. In particular, the point query data structure based on random codes and the norm estimation data structure require access to combinatorial objects that are shown to exist via the probabilistic method, but for which we do not have explicit constructions. The point query structure based on Reed-Solomon codes can be implemented in the word RAM model with $t_u = \tilde{O}(\lg n)$ using fast multipoint evaluation of polynomials. This is because performing an update, in addition to accessing $O(\lg n/\lg\lg n)$ memory cells of the data structure, requires evaluating a degree-$O(\lg n/\lg\lg n)$ polynomial on $O(\lg n/\lg\lg n)$ points to determine which memory cells to access (see [37] for details).

REMARK 4. The best known randomized upper bounds are $S = t_u = O(\lg(1/\delta))$ for point query [12] and ℓ_p/ℓ_p estimation for $p \leq 2$ [42, 28]. For entropy estimation the best upper bound has $S = t_u = \tilde{O}((\lg n)^2 \lg(1/\delta))$ [23]. For ℓ_1 heavy hitters the best known upper bound (in terms of S and t_u) has $S = t_u = O(\lg(n/\delta))$.

In addition to being the first non-trivial turnstile update time lower bounds, we also managed to show that the update time has to increase polylogarithmically in $1/\delta$ as the error probability δ decreases, which is achieved with the typical reduction using $\lg(1/\delta)$ independent copies of a data structure with constant error probability.

Our lower bounds can also be viewed in another light. If one is to obtain constant update time algorithms for the above problems, then one has to design algorithms that are *adaptive*. Since all known upper bounds have non-adaptive updates, this would require a completely new strategy to designing turnstile streaming algorithms. Note that for randomized algorithms, our lower bounds do not rule out constant update time with constant error probability. If however the error probability is upper bounded by $2^{-\omega(\sqrt{\lg n})}$, then adaptivity is necessary to achieve constant update time.

We also show a new cell probe *upper bound* for ℓ_1 point query which outperforms the CountMin sketch [12] in terms of *query time*, while matching it in both space and update time (see Section 4.2). Our new algorithm is inspired by the solution to the hard instance for our above lower bounds and we believe this upper bound provides evidence that despite the importance of the problems, the effort on understanding the time complexity of them has been insufficient and

perhaps better algorithms are achievable. In fact, our upper bound even demonstrates the $\lg(1/\delta)/\sqrt{\lg n}$ behaviour of our lower bounds, further supporting the hypothesis that faster algorithms exist. Our upper bound is randomized, non-adaptive and in the cell probe model, meaning that we assume computation is free of charge and that we have access to input independent truly random hash functions that can be evaluated free of charge. Clearly our lower bounds apply to this setting. It would be interesting to find a fast implementation of our algorithm in the word-RAM model.

Technique.

As suggested by Question 1, we prove our lower bounds using recent ideas in cell probe lower bound proofs. More specifically, we use ideas from the technique now formally known as *cell sampling* [18, 38, 32]. This technique derives lower bounds based on one key observation: if a data structure/streaming algorithm probes t memory words on an update, then there is a set C of t memory words such that at least m/S^t updates probe only memory words in C, where m is the number of distinct updates in the problem (for data structure lower bound proofs, we typically consider queries rather than updates, and we obtain tighter lower bounds by forcing C to have near-linear size).

We use this observation in combination with the standard one-way communication games typically used to prove streaming space lower bounds. In these games, Alice receives updates to a streaming problem and Bob receives a query. Alice runs her updates through a streaming algorithm for the corresponding streaming problem and sends the resulting Sw bit memory footprint to Bob. Bob then answers his query using the memory footprint received from Alice. By proving communication lower bounds for *any* communication protocol solving the one-way communication game, one obtains space lower bounds for the corresponding streaming problem.

At a high level, we use the cell sampling idea in combination with the one-way communication game as follows: if Alice's non-adaptive streaming algorithm happens to "hash" her updates such that they all probe the same t memory cells, then she only needs to send Bob the contents of those t cells. If $t < S$, this gives a communication saving over the standard reduction above. We formalize this as a general sketch-compression theorem, allowing us to compress the memory footprint of any non-adaptive streaming algorithm at the cost of increasing the error probability. This general theorem has the advantage of allowing us to re-use previous space lower bounds that have been proved using the standard reduction to one-way communication games, this time however obtaining lower bounds on the update time. We demonstrate these ideas in Section 2 and also give a more formal definition of the classic one-way communication game.

2. SKETCH COMPRESSION

In the following, we present a general theorem for compressing non-adaptive sketches. Consider a streaming problem in which we are to maintain an n-dimensional vector v. Let U be the *update domain*, where each element of U is a pair $(i, \Delta) \in [n] \times \{-M, \ldots, M\}$ for some $M = \text{poly}(n)$. We interpret an update $(i, \Delta) \in U$ as having the effect $v[i] \leftarrow v[i] + \Delta$. Initially all entries of v are 0. We also define the *query domain* $Q = \{q_1, \ldots, q_r\}$, where each $q_i \in Q$ is a function $q_i : \mathbb{Z}^n \to \mathbb{R}$. With one-way communication games in mind, we define the input to a streaming problem as consisting of two parts. More specifically, the *pre-domain* $D_{pre} \subseteq U^a$ consists of sequences of a update operations. The *post-domain* $D_{post} \subseteq \{U \cup Q\}^b$ consists of sequences of b updates and/or queries. Finally, the *input domain* $D \subseteq D_{pre} \times D_{post}$ denotes the possible pairings of a initial updates followed by b intermixed queries and updates. The set D defines a streaming problem P_D.

We say that a randomized streaming algorithm for a problem P_D uses S words of space if the maximum number of memory words used when processing any $d \in D$ is S. Here a memory word consists of $w = \Theta(\lg n)$ bits. The worst case update time t_u is the maximum number of memory words read/written to upon processing an update operation for any $d \in D$. The error probability δ is defined as the maximum probability over all $d \in D$ and queries $q_i \in d \cap Q$, of returning an incorrect results on query q_i after the updates preceding it in the sequence d.

A streaming problem P_D of the above form naturally defines a one-way communication game: on an input $(d_1, d_2) \in D$, Alice receives d_1 (the first a updates) and Bob receives d_2 (the last b updates and/or queries). Alice may now send a message to Bob based on her input and Bob must answer all queries in his input as if streaming through the concatenated sequence of operations $d_1 \circ d_2$. The error probability of a communication protocol is defined as the maximum over all $d \in D$ and $q_i \in \{d \cap Q\}$, of returning an incorrect results on q_i when receiving d as input.

Traditionally, the following reduction is used:

THEOREM 5. *If there is a randomized streaming algorithm for P_D with space usage S and error probability δ, then there is a private coin protocol for the corresponding one-way communication game in which Alice sends Sw bits to Bob and the error probability is δ.*

Proof. Alice simply runs the streaming algorithm on her input and sends the memory image to Bob. Bob continues the streaming algorithm and outputs the answers. ■

Recall from Section 1 that a randomized streaming algorithm is *non-adaptive* if:

- Before processing any elements of the stream, the algorithm may toss some random coins.
- The memory words read/written to (henceforth jointly referred to as probed) on any update operation (i, Δ) is completely determined from i and the initially tossed random coins.

We show that for non-adaptive algorithms, one can efficiently reduce the communication by increasing the error probability. We require some additional properties of the problem however: we say that a streaming problem P_D is *permutation invariant* if for any *permutation* $\pi : [n] \to [n]$, it holds that $\pi(q_i(v)) = (\pi(q_i)(\pi(v)))$ for all $q_i \in Q$. Here $\pi(v)$ is the n-dimensional vector with value $v[i]$ in entry $\pi[i]$, $\pi(q_i)$ maps all indices (if any) in the definition of the query q_i wrt. π and $\pi(q_i(v))$ maps all indices in the answer $q_i(v)$ (if any) wrt. π.

Observe that point query, ℓ_p estimation, entropy estimation and heavy hitters all are permutation invariant problems. For point query, we have $\pi(q_i(v)) = q_i(v)$ since answers contain no indices, but $\pi(q_i)$ might differ from q_i

since queries are defined from indices. For ℓ_p estimation and entropy estimation, we simply have $\pi(q_i(v)) = q_i(v)$ and $\pi(q_i) = q_i$ since neither queries or answers involve indices. For heavy hitters we have $\pi(q_i) = q_i$ (there is only one query), but we might have that $\pi(q_i(v)) \neq q_i(v)$ since the answer to the one query is an index. We now have the following:

THEOREM 6. *If there is a randomized non-adaptive streaming algorithm for a permutation invariant problem P_D with $a \leq \sqrt{n}$, having space usage S, error probability δ, and worst case update time $t_u \leq (1/2)(\lg n/\lg(eS/t_u))$, then there is a private coin protocol for the corresponding one-way communication game in which Alice sends at most $a\lg e + t_u a \lg(eS/t_u) + \lg a + \lg\lg(en/a) + t_u w + 1$ bits to Bob and the error probability is $2e^a \cdot (eS/t_u)^{t_u a}\delta$.*

Before giving the proof, we present the two main ideas. First observe that once the random choices of a non-adaptive streaming algorithm have been made, there must be a large collection of indices $I \subseteq [n]$ for which all updates (i, Δ), where $i \in I$, probe the same small set of memory words (there are at most $\binom{S}{t_u}$ distinct sets of t_u words to probe). If all of Alice's updates probed only the same set of t_u words, then Alice could simply send those words to Bob and we would have reduced the communication to $t_u w$ bits. To handle the case where Alice's updates probe different sets of words, we make use of the permutation invariant property. More specifically, we show that Alice and Bob can agree on a collection of k permutations of the input indices, such that one of these permutes all of Alice's updates to a new set of indices that probe the same t_u memory cells. Alice can then send this permutation to Bob and they can both alter their input based on the permutation. Therefore Alice and Bob can solve the communication game with $\lg k + t_u w$ bits of communication. The permutation of indices unfortunately increases the error probability as we shall see below. With these ideas in mind, we now give the proof of Theorem 6.

Proof (of Theorem 6). Alice and Bob will permute the indices in their communication problem and use the randomized non-adaptive streaming algorithm on this transformed instance to obtain an efficient protocol for the original problem.

By Yao's principle, we have that the randomized complexity of a private coin one-way communication game with error probability δ equals the complexity of the best deterministic algorithm with error probability δ over the worst distribution. Hence we show that for any distribution μ on $D \subseteq D_{pre} \times D_{post}$, there exists a deterministic one-way communication protocol with $t_u a \lg S + \lg a + \lg\lg n + t_u w$ bits of communication and error probability $2e^a \cdot (eS/t_u)^{t_u a}\delta$. We let μ_1 denote the marginal distribution over D_{pre} and μ_2 the marginal distribution over D_{post}.

Let $\mu = (\mu_1, \mu_2)$ be a distribution on D. Define a new distribution $\gamma = (\gamma_1, \gamma_2)$: pick a uniform random permutation π of $[n]$. Now draw an input d from μ and permutate all indices of updates (and queries if defined for such) using the permutation π. The resulting sequence $\pi(d)$ is given to Alice and Bob as before, which defines the new distribution $\gamma = (\gamma_1, \gamma_2)$. We use $A \sim \mu_1$ to denote the r.v. providing Alice's input drawn from distribution μ_1 and $\pi(A) \sim \gamma_1$ denotes the random variable providing Alice's transformed input. We define $B \sim \mu_2$ and $\pi(B) \sim \gamma_2$ symmetrically.

Recall we want to solve the one-way communication game on A and B. To do this, first observe that by fixing the random coins, the randomized non-adaptive streaming algorithm gives a non-adaptive and deterministic streaming algorithm that has error probability δ and space usage S under distribution γ. Before starting the communication protocol on A and B, Alice and Bob both examine the algorithm (it is known to both of them). Since it is non-adaptive and deterministic, they can find a set of t_u memory words C, such that at least $n/\binom{S}{t_u} > n/(eS/t_u)^{t_u} \geq \sqrt{n} \geq a$ indices $i \in [n]$ satisfy that any update (i, Δ) probes only memory words in C. We let I^C denote the set of all such indices (again, I^C is known to both Alice and Bob). Alice and Bob also agree on a set of permutations $\{\rho_1, \ldots, \rho_k\}$ (we determine a value for k later), such that for any set of at most a indices, I', that can occur in Alice's updates, there is at least one permutation ρ_i where:

- $\rho_i(j) \in I^C$ for all $j \in I'$
- Let $I(A)$ denote the (random) indices of the updates in A. Then the probability that the non-adaptive and deterministic protocol errs on input $\rho_i(A)$, conditioned on $I(A) = I'$, is at most $2e^a \cdot (eS/t_u)^{t_u a} \cdot \varepsilon_{I(A)=I'}$ where $\varepsilon_{I(A)=I'}$ is the error probability of the deterministic and non-adaptive streaming algorithm on distribution γ, conditioned on $I(A) = I'$.

Again, this set of permutations is known to both players. The protocol is now simple: upon receiving A, Alice finds the index i of a permutation ρ_i satisfying the above for the indices $I(A)$. She then sends this index to Bob and runs the deterministic and non-adaptive algorithm on $\rho_i(A)$. She forwards the addresses and contents of all memory words in C as well. This costs a total of $\lg k + |C| \cdot w \leq \lg k + t_u(n)w$ bits. Note that no words outside C are updated during Alice's updates. Bob now remaps his input B according to ρ_i and runs the deterministic and non-adaptive streaming algorithm on his updates and queries. Observe that for each query $q_j \in B$, Bob will get the answer $\rho_i(q_j)(\rho_i(v))$ if the algorithm does not err, where v is the "non-permuted" vector after processing all of Alice's updates A and all updates in B preceeding the query q_j. For each such answer, he computes $\rho_i^{-1}(\rho_i(q_j)(\rho_i(v)))$. Since P_D is permutation invariant, we have $\rho_i^{-1}(\rho_i(q_j)(\rho_i(v))) = \rho_i^{-1}(\rho_i(q_j(v))) = q_j(v)$. The final error probability (over μ) is hence at most $2e^a \cdot (eS/t_u)^{t_u a} \cdot \delta$ since $\mathbb{E}_{I'}\, \varepsilon_{I(A)=I'} = \delta$.

We only need a bound on k. For this, fix one set I' of at most a indices in $[n]$ and consider drawing k uniform random permutations. For each such random permutation Γ, note that $\Gamma(I')$ is distributed as $I(\pi(A))$ conditioned on $I(A) = I'$. Hence, the expected error probability when using the map Γ (expectation over choice of Γ) is precisely $\varepsilon_{I(A)=I'}$. We also have

$$\mathbb{P}(\Gamma(I') \subseteq I^C) = \frac{\binom{|I^C|}{|I'|}}{\binom{n}{|I'|}} \geq \left(\frac{|I^C|}{en}\right)^{|I'|} \geq e^{-a} \cdot \left(\frac{eS}{t_u}\right)^{-t_u a}$$

By Markov's inequality and a union bound, we have both $\Gamma(I') \subseteq I^C$ and error probability at most $2e^a \cdot (eS/t_u)^{t_u a} \cdot \varepsilon_{I(A)=I'}$ with probability at least $e^{-a} \cdot (eS/t_u)^{-t_u a}/2 \stackrel{\text{def}}{=} p$ over the choice of Γ. Thus if we pick $k = (1/p)a\lg(en/a) > (1/p) \cdot \lg\binom{n}{a}$ and use that $1 + x \leq e^x$ for all real x, then setting the probability that all permutations Γ chosen fail to have the desired failure probability and $\Gamma(I') \subseteq I^C$ is at

most $(1-p)^k < 1/\binom{n}{a}$. Thus by a union bound over all $\binom{n}{a}$ size-a subsets of indices, we can conclude that the desired set of permutations exists. ■

3. IMPLICATIONS

In this section, we present the (almost) immediate implications of Theorem 6. All these results follow by reusing the one-way communication game lower bounds originally proved to obtain space lower bounds of heavy hitters [27]. Consider the generic protocol in Figure 1. For different streaming problems, we will show the different ways to implement the function $Check(t, j)$ using the updates and queries of the problems, where $Check(t, j)$ is supposed to be able to tell if t is equal to i_j with failure probability δ. We show first that if this is the case, we obtain a lower bound on the update time t_u. The lower bounds then follow from the implementation of $Check(t, j)$.

1: Alice chooses a indices $i_1, \ldots, i_a$ randomly without replacement in $[n]$.
2: Alice performs updates $v[i_j] \leftarrow v[i_j] + C^j$ for $j = 1, \ldots, a$, where C is a large constant.
3: Alice sends her memory state to Bob.
4: **for** j from a down to 1 **do** ▷ Bob decodes $i_a, \ldots, i_1$ from Alice's message.
5: **for all** $t \in [n]$ **do**
6: **if** Check(t,j) **then**
7: Bob declares $i_j = t$.
8: Bob performs the update $v[i_j] \leftarrow v[i_j] - C^j$
9: **end if**
10: **end for**
11: **end for**

Figure 1: The communication protocol for Alice to send a random numbers in $[n]$ to Bob.

The proofs of the following two theorems follow easily from Theorem 6:

THEOREM 7. *Any randomized non-adaptive streaming algorithm that can implement all the $Check(t, j)$ calls using no more than $k \leq n^{O(1)}$ queries with failure probability $n^{-\Omega(1)} \leq \delta \leq 1/2 - \Omega(1)$ each, must have worst case update time $t_u = \Omega\left(\min\left\{\sqrt{\frac{\lg 1/\delta}{\lg(eS/t_u)}}, \frac{\lg 1/\delta}{\sqrt{\lg k \lg(eS/t_u)}}\right\}\right)$.*

Proof. We first prove the lower bound for the case where $\sqrt{\frac{\lg 1/\delta}{\lg(eS/t_u)}}$ is the smaller of the two terms. This is the case at least from $n^{-\Omega(1)} \leq \delta \leq k^{-3}$. Assume for contradiction that such a randomized non-adaptive algorithm exists with $t_u = o(\sqrt{\lg(1/\delta)/\lg(eS/t_u)})$. Set $a = c_0 t_u$ for a sufficiently large constant c_0. We invoke Theorem 6 to conclude that Alice can send her memory state to Bob using $a \lg e + t_u a \lg(eS/t_u) + \lg a + \lg\lg(en/a) + t_u w + 1$ bits while increasing the failure probability of Bob's k queries to

$$2e^a \cdot (eS/t_u)^{t_u a} \delta \leq (eS/t_u)^{c_0 t_u^2} \delta^{1-o(1)} \leq \delta^{1-o(1)} \leq k^{-2}$$

each. By a union bound, the answer to all Bob's queries are correct with probability at least $1 - 1/k \geq 9/10$, so Bob can recover $i_1, \ldots, i_a$ with probability 9/10. By Fano's inequality, Alice must send Bob at least $\Omega(H(i_1, \ldots, i_a)) = \Omega(a \lg(n/a)) \geq c_0 c_1 t_u \lg n$ bits for some constant c_1 (where c_1 does not depend on c_0). But the size of her message was

$$\begin{aligned} a \lg e + t_u a \lg(eS/t_u) + \lg a + \lg\lg(en/a) + t_u w + 1 &\leq \\ c_0 t_u^2 \lg(eS/t_u) + t_u w + o(t_u \lg n). \end{aligned}$$

We assumed

$$t_u = o\left(\sqrt{\lg(1/\delta)/\lg(eS/t_u)}\right) = o\left(\sqrt{\lg n/\lg(eS/t_u)}\right)$$

and thus the above is bounded by $t_u w + o(t_u \lg n)$. Setting c_0 high enough, we get that $w \geq (c_0 c_1 \lg n)/2$ and thus we have reached a contradiction.

For the case $k^{-3} < \delta < 1/2 - \Omega(1)$, observe that we can decrease the failure probability to k^{-3} by increasing the space, update time and query time by a factor $\alpha = \Theta(\lg_{1/\delta} k)$: simply run α independent copies of the streaming algorithm and return the majority answer on a query. Hence the lower bound becomes

$$\Omega\left(\frac{\sqrt{\frac{\lg k}{\lg(eS\alpha/(t_u\alpha))}}}{\alpha}\right) = \Omega\left(\frac{\lg 1/\delta}{\sqrt{\lg k \lg(eS/t_u)}}\right).$$

■

THEOREM 8. *Any deterministic non-adaptive streaming algorithm that can implement $Check(t, j)$ must have worst case update time $t_u = \Omega(\frac{\lg n}{\lg(eS/t_u)})$.*

Proof. The proof is similar to that of Theorem 7. Assume for contradiction there is a deterministic non-adaptive streaming algorithm with $t_u = o(\lg n/\lg(eS/t_u))$. Choose $a = c_0 t_u$ for sufficiently large constant c_0. By Theorem 6, Alice can send her memory state to Bob using

$$\begin{aligned} a \lg e + t_u a \lg(eS/t_u) + \lg a + \lg\lg(en/a) + t_u w + 1 &\leq \\ c_0 t_u^2 \lg(eS/t_u) + t_u w + o(t_u \lg n) &\leq \\ t_u w + o(t_u \lg n) \end{aligned}$$

bits and Bob can still answer every query correctly. Since Bob can recover $i_1, \ldots, i_a$, Alice must send Bob at least $H(i_1, \ldots, i_a) = \Omega(a \lg(n/a)) = c_0 c_1 t_u \lg n$ for some constant c_1 (independent of c_0). Setting c_0 large enough gives $w \geq (c_0 c_1 \lg n)/2$, thus deriving the contradiction. ■

3.1 Applications to specific problems

Point query.

We can implement each $Check(t, j)$ by simply querying for the value of $v[t]$ and check if the returned value is at least $C^j/3$. By the guarantee of the algorithm, if it does not fail, the returned value is within a factor 3 from the right value and thus, $Check(t, j)$ can correctly tell if $t = v_j$. Thus we run a total of $na = n^{O(1)}$ queries to implement all $Check(t, j)$'s.

ℓ_p/ℓ_q estimation.

We can implement $Check(t, j)$ as follows.

- $v[t] \leftarrow v[t] - C^j$
- Check if $\|v\|_p$ is at most $C^j/3$
- $v[t] \leftarrow v[t] + C^j$

By the guarantee of the algorithm, if it does not fail, the estimated ℓ_p/ℓ_q norm is at most $3C^{j-1} < C^j/3$ if $t = i_j$ and it is greater than $C^j/3$ if $t \neq i_j$. Thus, $Check(t, j)$ can correctly tell if $t = v_j$. Again we used $n^{O(1)}$ queries in total.

We can also implement $Check(t, j)$ for entropy estimation.

Entropy estimation.

We implement $Check(t, j)$ as follows.

- $v[t] \leftarrow v[t] - C^j$
- Check if the entropy is at most $1/3$
- $v[t] \leftarrow v[t] + C^j$

Consider the case the algorithm does not fail. First, if $t = i_j$ then the entropy is at most

$$\begin{aligned}
&\sum_{i=1}^{j} \frac{C^i(C-1)}{C^{j+1}-C} \lg \frac{C^{j+1}-C}{C^i(C-1)} \\
&\quad \leq \sum_{i=1}^{j-1} \frac{C^i(C-1)}{C^{j+1}-C} \lg 2C^{j-i} + \frac{C^j(C-1)}{C^{j+1}-C} \lg \frac{C^{j+1}-C}{C^j(C-1)} \\
&\quad \leq O\left(\frac{\lg C}{C-1}\right) + \frac{C^j(C-1)}{C^{j+1}-C} \cdot \frac{C^j-C}{C^j(C-1)} \\
&\quad \leq O\left(\frac{\lg C}{C-1}\right) \\
&\quad \leq 1/10.
\end{aligned}$$

On the other hand, if $t \neq i_j$ then after the first operation, $\|v\|_1 \leq \frac{2C^{j+1}-C^j-C}{C-1} < 3C^j$ so the entropy is at least the binary entropy of whether the index i_j is picked, which is greater than $H(1/3) > 0.9$. Thus, by the guarantee of the algorithm, $Check(t, j)$ correctly tells if $t = v_j$ using $n^{O(1)}$ queries.

COROLLARY 9. *Any randomized non-adaptive streaming algorithm for ℓ_1 heavy hitters with failure probability $n^{-O(1)} \leq \delta \leq 1/2 - \Omega(1)$ must have worst case update time $t_u = \Omega(\min\{\sqrt{\frac{\lg(1/\delta)}{\lg(eS/t_u)}}, \frac{\lg(1/\delta)}{\sqrt{\lg t_u \cdot \lg(eS/t_u)}}\})$. Any deterministic non-adaptive streaming algorithm must have worst case update time $t_u = \Omega\left(\frac{\lg n}{\lg(eS/t_u)}\right)$.*

Proof. We use a slightly different decoding procedure for Bob. Instead of running $Check(t, j)$ all indices $t \in [n]$, in iteration j, Bob can simply query for the heavy hitter to find i_j. Note that in iteration j, we have $\|v\|_1 = \frac{C^{j+1}-C}{C-1} < 2C^j = 2v[i_j]$ so if the algorithm is correct, Bob will find the index i_j. We have thus implemented all the $Check(t, j)$'s using only a queries. Recall from the proof of Theorem 7 that $a = O(t_u)$. ■

4. UPPER BOUNDS

In this section we first present an upper bound for the concrete hard instance used in Section 3 to derive our update time lower bounds. The upper bound nearly matches our lower bound and in particular exhibits the $\lg(1/\delta)/\sqrt{\lg n}$ behaviour with optimal space usage. In Section 4.2 we present an algorithm for ℓ_1 point query which outperforms the Count-Min sketch in terms of *query time*, while matching it in both space and update time. Our new algorithm is inspired by the solution to the hard instance from Section 3 and we believe this upper bound provides evidence that it might actually be the known upper bounds and not the lower bounds we present that are sub-optimal. Both our upper bounds are randomized, non-adaptive and in the cell probe model, meaning that we assume computation is free of charge and that we have access to truly random hash functions that can be evaluated free of charge. Clearly our lower bounds apply to this setting. Obtaining similar upper bounds in the word-RAM seems to require more ideas. We discuss this in further detail in Section 4.1 and Section 4.2.

4.1 Solving the Hard Instance

Recall from Section 3 that the hard instance used to derive the lower bounds has the following form: We have k distinct indices $i_1, \ldots, i_k$ and k values $\Delta_1, \ldots, \Delta_k \in V$ where $V = \{1, \ldots, n^{O(1)}\}$. We perform the updates $v_{i_j} \leftarrow v_{i_j} + \Delta_j$ for $j = 1, \ldots, k$. Following that, we perform the updates $v_{i_j} \leftarrow v_{i_j} - \Delta_j$ for $j = k, \ldots, 1$ possibly intermixed with queries asking whether a given index i satisfies $v_i = \Gamma_i$ for a query value $\Gamma_i \in V$. We present a randomized and non-adaptive algorithm for this problem, where we assume the availability of truly random hash functions. Our solution uses optimal space $\Theta(k \lg(1/\delta)/\lg n)$ words, each word of $w = \Theta(\lg n)$ bits. The query time and update time are both $\Theta(\lg(1/\delta)/\sqrt{\lg n})$ for any $\delta \leq \exp(-\Theta(\sqrt{\lg n}))$.

Algorithm.

Construct a table T with t rows and r columns. We denote the rows by $T_1, \ldots, T_t$. Every entry of the table stores a w bit word, which is initialized to 0. We have t truly random hash functions $h_1, \ldots, h_t : [n] \to [r]$ mapping indices to uniform random table cells. We also have t truly random hash functions $\sigma_1, \ldots, \sigma_t : [n] \times V \to \binom{w}{\sqrt{w}}$ mapping indices and a value to uniform random length w bit strings with precisely $\sqrt{w}$ 1-bits. Upon an update $v_i \leftarrow v_i + \Delta$ or $v_i \leftarrow v_i - \Delta$ for some $\Delta \in V$, we do a bitwise XOR of $\sigma_j(i, \Delta)$ onto the word stored in entry $T_j[h_j(i)]$ for $j = 1, \ldots, t$.

To answer whether an index i satisfies $v_i = \Gamma$ for some $\Gamma \in V$, we compute the standard inner product $\langle T_j[h_j(i)], \sigma_j(i, \Gamma)\rangle$ for all $j = 1, \ldots, t$ (interpreting the words as $\{0,1\}$-vectors in $\mathbb{R}^w$). Note that such an inner product is simply a count of how many 1's $T_j[h_j(i)]$ and $\sigma_j(i, \Gamma)$ have in common. Since $\sigma_j(i, \Gamma)$ has only $\sqrt{w}$ 1's, this number of always bounded by $\sqrt{w}$. If the majority of the $T_j[h_j(i)]$'s have

$$\langle T_j[h_j(i)], \sigma_j(i, \Gamma)\rangle \geq \sqrt{w}/2,$$

we return that v_i equals Γ and otherwise that $v_i \neq \Gamma$.

Analysis.

In the cell probe model, we measure query time and update time only as the number of memory words read/updated. Thus the query time and update time is t and the space usage is tr words. Given a desired error probability $\delta \leq \exp(-\Theta(\sqrt{\lg n}))$, we set $t = \Theta(\lg(1/\delta)/\sqrt{\lg n})$ and let $r = ck/\sqrt{\lg n}$ for some constant $c > 0$. This gives optimal space and an update and query time of $O(\lg(1/\delta)/\sqrt{\lg n})$. What remains is to show that the error probability is bounded by δ. For this, consider a query asking whether $v_i = \Gamma$ for some $\Gamma \in V$. Since the hash functions used for the different rows are independent, we restrict our attention to one row T_j. Let $K_j(i, \Gamma)$ be set of pairs $(i', \Delta_{i'})$ such that $\Delta_{i'} \in V$, $v_{i'} = \Delta_{i'}$ and either $i' \neq i$ or $\Delta_{i'} \neq \Gamma$. Note that $\Delta_{i'} \in V$

implies $\Delta_{i'} \neq 0$ and thus $|K_j(i,\Gamma)| \in \{k-1,k\}$ depending on whether $\Gamma = v_i$ or not. From $K_j(i,\Gamma)$ we also define $C_j(i,\Gamma) \subseteq K_j(i,\Gamma)$ as the pairs $(i',\Delta_{i'}) \in K_j(i,\Gamma)$ satisfying $h_j(i') = h_j(i)$, i.e. $C_j(i,\Gamma)$ is the *conflict list* of pairs hashing to the same word as index i in table T_j.

Let Σ denote the bitwise XOR of $\sigma_j(i',\Delta_{i'})$ for all $(i',\Delta_{i'})$ in $C_j(i,\Gamma)$. Then if $v_i = \Gamma$ we have $T_j[h_j(i)] = (\Sigma \text{ XOR } \sigma_j(i,\Gamma))$ and if $v_i \neq \Gamma$ we have $T_j[h_j(i)] = \Sigma$. It follows that if $\langle \Sigma, \sigma_j(i,\Gamma)\rangle < \sqrt{w}/2$, then $\langle T_j[h_j(i)], \sigma_j(i,\Gamma)\rangle > \sqrt{w}/2$ iff $v_i = \Gamma$. Hence we let E_{err} denote the event that $\langle \Sigma, \sigma_j(i,\Gamma)\rangle \geq \sqrt{w}/2$. We wish to bound $\mathbb{P}(E_{\text{err}})$.

For this, let E_{few} denote the event $|C_j(i,\Gamma)| \leq \sqrt{w}/4$. Now observe that conditioned on E_{few}, there are no more than $w/4$ bits in Σ that are 1. Since σ_j is truly random and $(i,\Gamma) \notin C_j(i,\Gamma)$, we have that $\sigma_j(i,\Gamma)$ is independent of these set bits and

$$\mathbb{E}[\langle \Sigma, \sigma_j(i,\Gamma)\rangle] \leq \sqrt{w}/4.$$

It follows that

$$\mathbb{P}(E_{\text{err}} \mid E_{\text{few}}) \leq \exp(-\Omega(\sqrt{w})) = \exp(-\Omega(\sqrt{\lg n})).$$

Next observe that

$$\mathbb{E}[|C_j(i,\Gamma)|] \leq k/r = \sqrt{\lg n}/c.$$

For big enough constant c, this is less than $\sqrt{w}/8$ and we get from a Chernoff bound that

$$\mathbb{P}(\neg E_{\text{few}}) = \mathbb{P}(|C_j(i,\Gamma)| > \sqrt{w}/4) < \exp(-\Omega(\sqrt{w})) = \exp(-\Omega(\sqrt{\lg n})).$$

We conclude

$$\begin{aligned}\mathbb{P}(E_{\text{err}}) &= \\ \mathbb{P}(E_{\text{err}} \mid E_{\text{few}})\,\mathbb{P}(E_{\text{few}}) + \mathbb{P}(E_{\text{err}} \mid \neg E_{\text{few}})\,\mathbb{P}(\neg E_{\text{few}}) &= \\ \exp(-\Omega(\sqrt{\lg n})).\end{aligned}$$

The probability that we fail in most of the t rows is thus bounded by $\exp(-\Omega(t\sqrt{\lg n})) \leq \delta$ which completes the analysis.

Discussion.

There are two places in the above algorithm where we make use of free computation in the cell probe model: Computing $\langle T_j[h_j(i)], \sigma_j(i,\Gamma)\rangle$ and in the efficient truly random hash functions. There seems to be hope of getting around the true randomness by resorting to $\Theta(\lg n)$-wise independent hash functions and batching several evaluations to exploit algorithms such as fast multipoint evaluation of polynomials to obtain close-to-constant amortized evaluation time, see e.g. [28]. Hashing only to bit strings with exactly $\sqrt{w}$ 1's in near-constant time might need some additional ideas.

4.2 Faster ℓ_1 Point Query in the Cell Probe Model

Below we present our improved algorithm for ℓ_1 point query. In this problem we must support updates $v_i \leftarrow v_i + \Delta$ for $\Delta \in \{-n^{O(1)}, \ldots, n^{O(1)}\}$. Given a query index i, we must return v_i up to an additive error of $\varepsilon\|v\|_1$ with probability at least $1-\delta$. For the reader familiar with the classic CountMin sketch, our basic idea is to use the word-packing approach from Section 4.1 to pack roughly $\lg n$ counters in one word. The difficulty here is that Δ requires $\Theta(\lg n)$ bits and not just one bit as in Section 4.1, thus no more than one counter fits in a word. We get around this by storing a *summary word* for groups of roughly $\lg n$ counters. The summary words store $(1+\varepsilon)$-approximations of the values of the counters and hence several approximations can be packed in one word. When answering queries, it suffices to use the approximate counters and thus we have effectively reduced the number of words that must be read. Our final result, in the cell probe model, is $O(\varepsilon^{-1}\lg(1/\delta))$ words of space, update time $O(\lg(1/\delta))$, and query time $O(1 + \varepsilon\lg(1/\delta) + \frac{\lg(1/\delta)}{\sqrt{\lg n}} \cdot \sqrt{\lg\lg n + \lg(1/\varepsilon)})$. The space and update time match the CountMin sketch, whereas the query time strictly outperforms it for $1/n^{o(1)} \leq \varepsilon \leq o(1)$ and $\delta = o(1)$. The details are as follows.

Algorithm.

Let $k = \Theta(\min\{\lg^{-2}(1/\delta), \varepsilon^{-2}, \lg n/\lg\lg_{1+\varepsilon} n\})$. We store a table T with t rows denoted $T_1, \ldots, T_t$. Each row T_j is partitioned into r blocks of k entries each. We use $T_j[h,\ell]$ to denote the ℓ'th entry of the h'th block in T_j for any $(h,\ell) \in [r] \times [k]$. Each of the trk entries stores a $\Theta(\lg n)$-bit counter which is initialized to 0. In addition to T, we store a table A with t rows and r columns. The rows of A are denoted $A_1, \ldots, A_t$ and entry $A_j[h]$ stores a $(1+\varepsilon/4)$-approximation of $T_j[h,\ell]$ for each $\ell \in [k]$ and thus an entry of A takes $O(k\lg\lg_{1+\varepsilon} n) \leq O(\lg n)$ bits.

We have t truly random hash functions $h_1, \ldots, h_t : [n] \to [r]$ mapping indices to blocks and t truly random hash functions $\sigma_1, \ldots, \sigma_t : [n] \to \binom{k}{\sqrt{k}}$ mapping indices to a subset of $\sqrt{k}$ counters inside a block.

Upon an update $v_i \leftarrow v_i + \Delta$, we add Δ to all counters $T_j[h_j(i),\ell]$ for $j = 1, \ldots, t$ and $\ell \in \sigma_j(i)$. We also update the corresponding $(1+\varepsilon/4)$-approximations of the counters. These are stored in $A_j[h_j(i)]$ for $j = 1, \ldots, t$.

To answer a query for index i, we read $A_j[h_j(i)]$ for $j = 1, \ldots, t$. For each j, we extract from $A_j[h_j(i)]$ a $(1+\varepsilon/4)$-approximations of $T_j[h_j(i),\ell]$ for each $\ell \in \sigma_j(i)$. We finally return the median of these approximations as our estimate of v_i.

Analysis.

We first analyse the resource requirements. The space usage is $O(trk)$ words. In the cell probe model, the query time and update time is defined as the number of words read/updated and thus the algorithm has update time $O(t\sqrt{k})$ and query time $O(t)$.

We fix the parameters in the following. Given a desired error probability δ, we set $t = 1 + O(\lg(1/\delta)/\sqrt{k})$. To obtain asymptotically optimal space, we set $r = c(\varepsilon^{-1}\lg(1/\delta))/(tk)$ for a large constant $c > 0$. Note that by our choice of k, we have $r \geq c(\varepsilon^{-1}\lg(1/\delta))/(k + O(\lg(1/\delta)\sqrt{k})) \geq 1$ if c is sufficiently large. What remains is to show that this gives the desired error probability on a query for index i. What makes the analysis more cumbersome than for the standard CountMin sketch is mainly the dependencies introduced by the blocking.

For the analysis, first observe that if we consider a counter $T_j[h_j(i),\ell]$ for an $\ell \in \sigma_j(i)$ and let β denote the sum of absolute values of all other indices contributing to $T_j[h_j(i),\ell]$, i.e. $\beta = \sum_{i' \neq i : h_j(i) = h_j(i') \wedge \ell \in \sigma_j(i')} |v_{i'}|$, then if $\beta \leq \varepsilon\|v\|_1/2$ we have

$$(v_i - \varepsilon\|v\|_1/2)/(1+\varepsilon/4) \leq A_j[h_j(i)] \leq (v_i + \varepsilon\|v\|_1/2)(1+\varepsilon/4).$$

Since $v_i \leq \|v\|_1$, this implies

$$\begin{aligned} A_j[h_j(i)] &\geq (1-\varepsilon/4)(v_i - \varepsilon\|v\|_1/2) \\ &\geq v_i - \varepsilon\|v\|_1/2 - \varepsilon\|v\|_1/4 - \varepsilon\|v\|_1/8 \\ &\geq v_i - \varepsilon\|v\|_1. \end{aligned}$$

and

$$\begin{aligned} A_j[h_j(i)] &\leq (v_i + \varepsilon\|v\|_1/2)(1+\varepsilon/4) \\ &\leq v_i + \varepsilon\|v\|_1/2 + \varepsilon\|v\|_1/4 + \varepsilon\|v\|_1/8 \\ &\leq v_i + \varepsilon\|v\|_1. \end{aligned}$$

It follows that our median estimate is correct if more than half of the counters $T_j[h_j(i), \ell]$ satisfy

$$\sum_{i' \neq i : h_j(i) = h_j(i') \wedge \ell \in \sigma_j(i')} |v_{i'}| \leq \varepsilon\|v\|_1/2.$$

To bound the probability this happens, let C_j denote the subset of indices $i' \neq i$ such that $h_j(i') = h_j(i)$. We say that the event E_{err} happens if there are more than $\sqrt{k}/4$ indices $\ell \in \sigma_j(i)$ for which $\sum_{i' \in C_j : \ell \in \sigma_j(i')} |v_{i'}| > \varepsilon\|v\|_1/2$. We wish to bound $\mathbb{P}(E_{\text{err}})$. For this, partition C_j into two sets H_j and L_j. The set H_j contains the *heavy* indices $i' \in C_j$ such that $|v_{i'}| \geq \varepsilon\|v\|_1/2$. L_j contains the remaining *light* indices. We define two auxiliary events. Let E_{errH} be the event $|H_j| \geq \sqrt{k}/16$ and let E_{errL} be the event $\sum_{i' \in L_j} |v_{i'}| \geq \varepsilon\|v\|_1\sqrt{k}/16$. If we condition on $\neg E_{\text{errH}}$ and $\neg E_{\text{errL}}$, then there can be no more than $k/16 + k/8 < k/5$ indices $\ell \in [k]$ for which $\sum_{i' \in C_j : \ell \in \sigma_j(i')} |v_{i'}| > \varepsilon\|v\|_1/2$. From this it follows from a Chernoff bound that $\mathbb{P}(E_{\text{err}} \mid \neg E_{\text{errH}} \wedge \neg E_{\text{errL}}) = \exp(-\Omega(\sqrt{k}))$.

Next observe that there can be no more than $2\varepsilon^{-1}$ indices i' for which $|v_{i'}| \geq \varepsilon\|v\|_1/2$. For each such index i', we have $\mathbb{P}(i' \in H_j) = 1/r$ and we get $\mathbb{E}[|H_j|] \leq 2\varepsilon^{-1}/r = 2tk/(c\lg(1/\delta)) = 2k/(c\lg(1/\delta)) + O(\sqrt{k}/c) = O(\sqrt{k}/c)$. Setting c high enough, this is no more than $\sqrt{k}/32$. It follows from a Chernoff bound that $\mathbb{P}(E_{\text{errH}}) = \exp(-\Omega(\sqrt{k}))$.

For analysing $\mathbb{P}(E_{\text{errL}})$, observe that

$$\begin{aligned} \mathbb{E}[\sum_{i' \in L_j} |v_{i'}|] &\leq \|v\|_1/r \\ &= \varepsilon\|v\|_1 tk/(c\lg(1/\delta)) \\ &= O(\varepsilon\|v\|_1\sqrt{k}/c). \end{aligned}$$

For large enough c, this is bounded by $\varepsilon\|v\|_1\sqrt{k}/32$ and thus E_{errL} is an event in which $\sum_{i' \in L_j} |v_{i'}|$ exceeds its expectation by at least a factor 2. Since all i' considered have $|v_{i'}| \leq \varepsilon\|v\|_1/2$, it follows that the probability of E_{errL} is maximized when the mass (of $\|v\|_1$) is distributed evenly on $2\varepsilon^{-1}$ coordinates. Thus E_{errL} becomes the event that at least $\sqrt{k}/8$ of these coordinates fall in L_j. By the arguments we used to bound $\mathbb{P}(E_{\text{errH}})$, we conclude $\mathbb{P}(E_{\text{errL}}) = \exp(-\Omega(\sqrt{k}))$. Combining the pieces, we conclude $\mathbb{P}(E_{\text{err}}) = \exp(-\Omega(\sqrt{k}))$.

Finally observe that for the median estimate to be incorrect, the event E_{err} must happen for $\Omega(t)$ rows and the hash functions for these rows are independent. It finally follows by a Chernoff bound that the error probability is bounded by $\exp(-\Omega(t\sqrt{k})) \leq \delta$. By our choice of parameters, this gives optimal space of $O(\varepsilon^{-1}\lg(1/\delta))$ words, update time $O(t\sqrt{k}) = O(\sqrt{k} + \lg(1/\delta)) = O(\lg(1/\delta))$ and query time $O(t) = O(1 + \lg(1/\delta)/\sqrt{k})$ which is bounded by

$$O\left(1 + \varepsilon\lg(1/\delta) + \frac{\lg(1/\delta)}{\sqrt{\lg n}} \cdot \sqrt{\lg\lg n + \lg(1/\varepsilon)}\right)$$

This strictly outperforms the CountMin sketch for any ε and δ satisfying $1/n^{o(1)} \leq \varepsilon \leq o(1)$ and $\delta = o(1)$.

Discussion.

In addition to the true randomness, the median computation also prevents the above algorithm from being efficiently implemented in word RAM. If one finds an efficient way of extracting the relevant $\sqrt{k}$ approximations from each $A_j[h_j(i)]$, one can most likely use the standard randomized median selection algorithm [11] to find the median efficiently using word-level parallelism (basically running an external memory model [1] median selection algorithm).

5. REFERENCES

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31:1116–1127, 1988.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[3] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004.

[4] A. Chakrabarti, G. Cormode, and A. McGregor. A near-optimal algorithm for estimating the entropy of a stream. *ACM Transactions on Algorithms*, 6(3), 2010.

[5] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *IEEE Conference on Computational Complexity*, pages 107–117, 2003.

[6] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

[7] R. H. Chitnis, G. Cormode, M. T. Hajiaghayi, and M. Monemizadeh. Parameterized streaming algorithms for vertex cover. *CoRR*, abs/1405.0093, 2014.

[8] R. Clifford and M. Jalsenius. Lower bounds for online integer multiplication and convolution in the cell-probe model. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 593–604, 2011.

[9] R. Clifford, M. Jalsenius, and B. Sach. Tight cell-probe bounds for online hamming distance computation. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 664–674, 2013.

[10] R. Clifford, M. Jalsenius, and B. Sach. Cell-probe bounds for online edit distance and other pattern matching problems. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015. To appear.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Mass., 1990.

[12] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[13] C. Estan, G. Varghese, and M. E. Fisk. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Trans. Netw.*, 14(5):925–937, 2006.

[14] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[15] M. L. Fredman. Observations on the complexity of generating quasi-Gray codes. *SIAM J. Comput.*, 7:134–146, 1978.

[16] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993.

[17] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.

[18] A. Gál and P. B. Miltersen. The cell probe complexity of succinct data structures. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming*, pages 332–344, 2003.

[19] S. Ganguly and A. Majumder. CR-precis: A deterministic summary structure for update data streams. In *ESCAPE*, pages 48–59, 2007.

[20] Google. Google Trends. `http://www.google.com/trends`.

[21] Google. Google Zeitgeist. `http://www.google.com/press/zeitgeist.html`.

[22] A. Gronemeier. Asymptotically optimal lower bounds on the NIH-multi-party information complexity of the AND-function and disjointness. In *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 505–516, 2009.

[23] N. J. A. Harvey, J. Nelson, and K. Onak. Sketching and streaming entropy via approximation theory. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 489–498, 2008.

[24] T. S. Jayram. Hellinger strikes back: A note on the multi-party information complexity of AND. In *Proceedings of the 12th International Workshop on Randomization and Computation (RANDOM)*, pages 562–573, 2009.

[25] T. S. Jayram, R. Kumar, and D. Sivakumar. The one-way communication complexity of Hamming distance. *Theory of Computing*, 4(1):129–135, 2008.

[26] T. S. Jayram and D. P. Woodruff. Optimal bounds for Johnson-Lindenstrauss transforms and streaming problems with subconstant error. *ACM Transactions on Algorithms*, 9(3):26, 2013.

[27] H. Jowhari, M. Saglam, and G. Tardos. Tight bounds for Lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 49–58, 2011.

[28] D. M. Kane, J. Nelson, E. Porat, and D. P. Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 745–754, 2011.

[29] D. M. Kane, J. Nelson, and D. P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1161–1178, 2010.

[30] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 41–52, 2010.

[31] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: methods, evaluation, and applications. In *Internet Measurement Comference*, pages 234–247, 2003.

[32] K. G. Larsen. Higher cell probe lower bounds for evaluating polynomials. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 293–301, 2012.

[33] K. G. Larsen. *Models and Techniques for Proving Data Structure Lower Bounds*. PhD thesis, Aarhus University, 2013.

[34] M. Minsky and S. Papert. *Perceptrons*. MIT Press, 1969.

[35] D. Moore, 2001. `http://www.caida.org/research/security/code-red/`.

[36] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[37] J. Nelson, H. L. Nguyễn, and D. P. Woodruff. On deterministic sketching and streaming for sparse recovery and norm estimation. *Lin. Alg. Appl.*, 441:152–167, Jan. 2014. Preliminary version in RANDOM 2012.

[38] R. Panigrahy, K. Talwar, and U. Wieder. Lower bounds on near neighbor search via metric expansion. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 805–814, 2010.

[39] M. Pătraşcu. *Lower bound techniques for data structures*. PhD thesis, Massachusetts Institute of Technology, 2008.

[40] M. Pătraşcu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006.

[41] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[42] M. Thorup and Y. Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM J. Comput.*, 41(2):293–331, 2012.

[43] D. P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 167–175, 2004.

[44] D. P. Woodruff. *Efficient and Private Distance Approximation in the Communication and Streaming Models*. PhD thesis, Massachusetts Institute of Technology, 2007.

[45] A. C.-C. Yao. Should tables be sorted? *J. ACM*, 28:615–628, 1981.

DRAFT (submitted) version

Computing River Floods Using Massive Terrain Data

Cici Alexander
Dynamiques de l'Environnement Côtier
IFREMER, France
calexand@ifremer.fr

Lars Arge
MADALGO*
Aarhus University, Denmark
large@madalgo.au.dk

Peder Klith Bøcher
Department of Bioscience
Aarhus University, Denmark
peder.bocher@biology.au.dk

Morten Revsbæk
SCALGO
Aarhus, Denmark
morten@scalgo.com

Brody Sandel
Department of Bioscience and MADALGO*
Aarhus University, Denmark
brody.sandel@bios.au.dk

Jens-Christian Svenning
Department of Bioscience
Aarhus University, Denmark
svenning@biology.au.dk

Constantinos Tsirogiannis
Department of Bioscience and MADALGO*
Aarhus University, Denmark
constant@madalgo.au.dk

Jungwoo Yang
MADALGO*
Aarhus University, Denmark
jungwoo@madalgo.au.dk

ABSTRACT

Many times river floods have resulted in huge catastrophes. To reduce the negative outcome of such floods, it is important to predict their extent before they happen. For this reason, scientists nowadays use algorithms that model river floods on digital terrains. Yet, all existing algorithms of this kind have a major drawback; they cannot efficiently process massive terrain datasets, which have become widely available during the last years.

In this paper, we describe two algorithms that provide high-quality river flood modelling and, unlike any previous approach, efficiently handle massive terrain data. More specifically, given a raster terrain and a subset of its cells representing a river network, we describe two algorithms that for each cell in the raster estimate the height that the river should rise for the cell to get flooded. One of the proposed algorithms is a redesign of a European Union approved method that is used by authorities in Denmark for modelling river floods. We show how this algorithm can be adapted to efficiently handle massive terrain data. The other algorithm is a novel method that we introduce for modelling river floods. For an input raster that consists of N cells, and which is so large that it can only be stored in the hard disk of a computer, each of the proposed algorithms can produce its output with only $O(\text{sort}(N))$ transfers of data blocks between the disk and the main memory. Here $\text{sort}(N)$ denotes the minimum number of data transfers that are needed for sorting a set of N elements stored on disk. We have implemented both algorithms, and compared their output with data that were acquired from a real flood event. We show that both algorithms produce an output that models the actual event quite accurately. In fact, the new algorithm that we introduce produces more accurate results than the existing popular method. We evaluated the efficiency of our algorithms in practice by conducting experiments on massive datasets. We show that the two algorithms perform efficiently even for datasets of approximatelly 268 GB size.

*Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

1. INTRODUCTION

Throughout history, river floods have caused large disasters. Usually induced by heavy rainfall, such floods can lead to casualties and huge financial damage for the local communities. A recent example is the catastrophic flood of the Indus river in Pakistan that took place in 2010 [6]. This flood claimed approximatelly two thousand lives, and about one fifth of the total area of the country ended up covered by water. Society wants to predict such floods, so that measures can be taken in advance to reduce the harm done. Therefore, it is important for people to know which regions around a river have the highest risk of getting flooded when the level of the river rises.

Today, hydrologists use computers to model river floods; they use specialised software to simulate flood events based on digital representations of terrains and rivers. Such terrain representations are widely known as Digital Elevation Models (DEMs). The most popular type of DEMs is the so-called *grid* or *raster* DEMs. In a raster DEM the domain of the terrain is divided into square cells of equal size, and each cell is associated with an elevation value.

One method for modelling river floods on DEMs is the method introduced by Berg Sonne [11]; let $\mathcal{G}$ be a raster terrain and let $R(\mathcal{G})$ be the set of cells in $\mathcal{G}$ that represents the region covered by a river network in this terrain. Also, let x be a positive real. Given $\mathcal{G}$, $R(\mathcal{G})$ and x, the method

estimates which cells in $\mathcal{G}$ will get flooded if the level of the river $R(\mathcal{G})$ rises uniformly by x meters. Of course, a flood is a very complex phenomenon and is influenced by many factors, some of which are difficult to determine. Therefore, we cannot expect that a flood can be modelled precisely by the output of any method, no matter how involved this method is. However, the method proposed by Berg Sonne is today considered a quite accurate tool for modelling river floods. Hence, after approval by the European Union it is used by the state authorities of Denmark [11].

However, Berg Sonne's method has a major drawback; it cannot process massive DEMs. Recent advances in Lidar technology have made it possible to produce detailed and huge DEM datasets. In many cases, such a dataset is so large that it cannot fit in the main memory of a standard computer. Hence, the dataset has to be stored mainly on disk. Since the computer's processor can only handle data that appear in the main memory, blocks of data have to be transfered between the disk and the memory in order to process the dataset. We call a transfer of a single block of data between the disk and the memory an *I/O-operation*, or an I/O for short. The problem here is that a single I/O is an extremely slow operation; it can take about the same time as a million CPU operations. Therefore, when it comes to processing huge amounts of data, it is important to process the dataset in a way that we minimize the number of data transfers between the disk and the memory. Otherwise, the whole process becomes practically infeasible. To handle this issue, Aggarwal and Vitter introduced the so called *I/O-model*, which takes into account the number of I/Os between the disk and the main memory [3]. The performance of an algorithm in the I/O-model is measured as the number of I/Os that take place during its execution. This measure of performance is called the *I/O-efficiency* of the algorithm. To describe the I/O-efficiency we need three parameters; the size N of the input data, the size of the internal memory M, and the size B of a single block of data that can be transfered from and to the disk. Two basic processes that take place during the execution of most algorithms is scanning and sorting. We can scan a set of N elements stored in the disk with $O(\text{scan}(N))$ I/Os, where $\text{scan}(N) = N/B$. We can also sort a set of N records in an I/O-efficient manner with $O(\text{sort}(N))$ I/Os, where $\text{sort}(N) = N/B \log_{M/B} N/B$ [3].

Standard algorithms are often designed based on the assumption that all input data fit in the main memory. Hence, usually they cannot handle massive datasets. This is also the case for algorithms that are used to model river floods; to the best of our knowledge, there does not exist any I/O-efficient algorithm for this problem. Therefore, the users of up-to-date hydrological software are forced to choose between two approaches. In the first approach, the resolution of the input DEM is reduced (so it fits entirely in the main memory). Thus, a large amount of detail in terrain data is thrown away. Important features on the landscape, such as ditches and levees, may not be depicted anymore on the resulting terrain. When it comes to modelling a river flood, this results in wrong estimations. In the other approach, users divide the massive DEM into smaller tiles and each tile is processed independently; in this way, when processing a single tile, we do not take into account how the rest of the landscape affects the flood in that region. Therefore, there is a need for developing algorithms that, on one hand model river floods accuratelly, and on the other hand handle massive terrain datasets efficiently.

Our Results.

Inspired by the above, we designed two I/O-efficient algorithms that can be used for modelling river floods. The first algorithm is an adaptation of Berg Sonne's method that can handle massive raster terrains. The second algorithm is a novel method that we introduce for modelling river floods. For each of these algorithms, the input is a raster $\mathcal{G}$, and a subset $R(\mathcal{G})$ of cells in $\mathcal{G}$ representing the area covered by a river network. Each of our algorithms returns for each cell $c \in \mathcal{G}$ a value $\text{f}(c)$, indicating the minimum number of meters the river level should rise before c gets flooded. We call this value the *resistance* value of c. Given the resistance values $\text{f}(c)$ for every $c \in \mathcal{G}$, and a positive integer x, we can then easily extract the part of the terrain that is flooded if the river level rises uniformly by x meters. Each of the algorithms that we propose uses different criteria for computing resistance values, hence they produce different outputs. In the algorithm by Berg Sonne, the resistance value of each cell is computed in two stages; in the first stage, for each cell c the elevation difference is computed between c and its closest river cell (in the xy-plane). We call this elevation difference the *obstruction* value of c. In the second stage, the resistance value of c is computed based on the obstruction values of the cells that appear on any path connecting c with the river network. In our new algorithm, we compute the resistance values by taking into account how water flows on the terrain surface. In particular, we consider a model according to which water can flow from a cell to potentially more than one of its neighbours. Based on this model, we compute the resistance value of each cell $c \in \mathcal{G}$ as the elevation difference between c and the highest river cell where water from c can reach.

We have designed both of our algorithms using the I/O-model of Agarwal and Vitter [3]. To compute the resistance values on a raster that has N cells, each of our algorithms require $O(\text{sort}(N))$ I/Os in the worst case. We have implemented both algorithms and measured their efficiency in practice. We show that the two algorithms perform efficiently even for raster datasets of approximatelly 268 gigabytes size; to process a dataset of this size on a standard workstation, our adaptation for Berg Sonne's method required roughly 24 hours, and our new algorithm roughly 31 hours. We also conducted experiments to evaluate that our algorithms can model adequately real flood events. To do this, we worked as follows; we used as reference a vector dataset which outlines the river flood that took place in Pakistan in 2010 [6]. We also used each of our algorithms to compute the resistance values on a grid that models the terrain in Pakistan. Then we selected a large number of pairs of cells from this grid; each pair was selected such that one cell in the pair is covered by the flooded region in the vector dataset, and the other cell falls outside this region. On the output of each algorithm, we measured the percentage of pairs where the flooded cell of the pair scores a lower resistance value than the non-flooded cell. This percentage was 87% for Berg Sonne's method and 92% for our algorithm. We repeated the same measurements many times, each time sampling cell pairs within a different small region overlapping with the flooded area. We observed that the percentage scored by each method depends on the size of the sampling region, and the topographic heterogeneity within this region.

The lowest percentages were observed for the sampling regions of the smallest size that we considered; these regions were squares of 20 km dimension. For such regions, the mean percentage measured for Berg Sonne's method is 61%, and the mean percentage for our algorithm is 71%. For all region sizes that we used, our algorithm provided on average more accurate results than Berg Sonne's method. To understand the reasons behind this difference in the output quality, we used both algorithms to model river floods on a massive raster that represents the terrain in Denmark. Among other artefacts, the method by Berg Sonne produced flooded regions that had larger size than the ones calculated by our algorithm. As we explain, one reason for this is that Berg Sonne's method produces very small resistance values for areas along the entire coastline of the terrain.

2. PROBLEM DEFINITION AND NOTATION

Let $\mathcal{G}$ be a grid terrain that consists of N cells. For every cell $c \in \mathcal{G}$ we use $h(c)$ to indicate the elevation of the terrain at this cell. We denote the cell that appears at the i-th row and j-th column of $\mathcal{G}$ by $\mathcal{G}(i, j)$. We assume without loss of generality that the center of grid cell $\mathcal{G}(i, j)$ has xy-coordinates (j, i). For any cell $c \in G$, we denote this center point by $p(c)$. We call the *xy-distance*, or simply the *distance*, between two cells in $\mathcal{G}$ the 2D Euclidean distance between their cell centers on the xy-domain of $\mathcal{G}$. Let C be a set of cells in $\mathcal{G}$ and let c be a cell that belongs to this set. We say that c is the *closest cell* in C to another cell c' if c has the smallest xy-distance to c' compared to any other cell in C.

We use $R(\mathcal{G})$ to denote a subset of the cells in $\mathcal{G}$ that belong to a river network of the terrain. We call these cells the *river cells* of $\mathcal{G}$. The cells in $R(\mathcal{G})$ represent the river network in $\mathcal{G}$ when there is no flood. This implies that the elevation value of each cell in $R(\mathcal{G})$ approximates the average height of the river level at this location when no flood occurs. For the algorithms that we present, we assume that $R(\mathcal{G})$ is provided as part of the input.

Let h_{rise} be a positive real. We say that there is a *river rise* of h_{rise} meters, or that *the river rises* by h_{rise} meters, when for each cell $c \in R(\mathcal{G})$ the river level rises to elevation $h(c) + h_{\text{rise}}$. We call h_{rise} the *rise value*. Thus, when a river rise takes place we assume that the level of the river increases by the same amount at all river cells.

We study the following problem. Given a terrain $\mathcal{G}$ and its river network $R(\mathcal{G})$, we want to compute for every cell $c \in \mathcal{G}$ a value f(c) that estimates the minimum value h_{rise} such that c gets flooded when the river rises by h_{rise} meters. We call this value the *resistance* value of c. Each of the two algorithms that we present in this paper defines these resistance values in a different way; hence, for the same input grid the output between the two algorithms may differ substantially. For each algorithm. we provide a detailed definition for the resistance of a grid cell in the description of the algorithm. For both algorithms, it is assumed that all river cells are flooded by default. Therefore, for both approaches we imply that the resistance value of every river cell is set to zero.

3. DESCRIPTION OF THE ALGORITHMS

3.1 Adaptation of Berg Sonne's Method

The first algorithm that we describe is based on the flood modelling method introduced by Berg Sonne [11]. Originally, this method was designed to solve a more simple problem than the one that we examine. In particular, the input of the original method is a raster $\mathcal{G}$, the river network $R(\mathcal{G})$, and a rise value h_{rise}. Instead of computing flood resistance values, the method outputs the cells in $\mathcal{G}$ that are considered to get flooded when $R(\mathcal{G})$ rises by h_{rise} meters. We call this version of the method *ProximityFlood*. Below, we first explain how *ProximityFlood* calculates the flooded cells in $\mathcal{G}$ for a given rise value h_{rise}. Then, we show how we can use this method to design an I/O-efficient algorithm that computes a flood resistance value for each input cell[1].

ProximityFlood consists of two steps. In the first step, every cell $c \in \mathcal{G} \setminus R(\mathcal{G})$ gets associated with a single river cell in $R(\mathcal{G})$; this is the river cell from which we consider that c can potentially get flooded. We call this cell the *source* cell of c, and we denote this by source(c). The source cell for every $c \in \mathcal{G} \setminus R(\mathcal{G})$ is defined as the river cell $c' \in R(\mathcal{G})$ that has the smallest xy-distance from c. After calculating source(c) for every non-river cell c, the height difference between source(c) and c is computed and stored together with c. We call this value the *obstruction* value obst(c) of c.

In the second step, we extract the cells in $\mathcal{G}$ that are considered to get flooded when the river rises by h_{rise} meters. More specifically, we extract any cell c that a) has an obstruction value obst[c] $\leqslant h_{\text{rise}}$ *and* b) there exists a path of cells between c and a river cell c_R such that any non-river cell c' in this path has an obstruction value obst(c') $\leqslant h_{\text{rise}}$. Notice that, in this way, not all cells with obstruction $\leqslant h_{\text{rise}}$ are flooded.

Method *ProximityFlood* can be used to model a single flood event at a time. On the other hand, if we want to study which regions get flooded for different rise values then we have to run this method many times, once for each distinct rise value h_{rise}. To avoid this, we instead choose to compute for each cell c the minimum rise value h_{rise} for which c gets flooded according to method *ProximityFlood*. Below we describe our new I/O-efficient algorithm that does this, which we call *ProximityResistance*.

As with *ProximityFlood*, the new algorithm consists of two main steps. In the first step, we compute for each cell $c \in \mathcal{G} \setminus R(\mathcal{G})$ the source cell source(c) and the obstruction obst(c). In the second step, we calculate the flood resistance values of all cells in $\mathcal{G} \setminus R(\mathcal{G})$.

Computing the source cells and obstruction values.

For the first step, the main task is to compute the source cell for each non-river cell c; given this cell, it is straightforward to compute the obstruction obst(c). Calculating the source cells in $\mathcal{G}$ is similar to computing a Voronoi di-

[1]Some implementations of *ProximityFlood* include an extra preprocessing step where the heights of the river cells are adjusted to make it consistent with the rest of the terrain data. This is useful when the river dataset is acquired from a different source than the terrain raster. In that case, projecting the river data on the raster may create artefacts (such as rivers that flow upstream). In the description that we provide for *ProximityFlood* we do not include this preprocessing step; we consider that this step has to do more with configuring the datasets rather than with the method itself. Yet, the preprocessing step can be also handled in an I/O-efficient manner, given a realistic assumption on the memory size.

agram on the xy-domain of $\mathcal{G}$; the sites of the Voronoi diagram are the center-points of the river cells in $\mathcal{G}$ and for any cell $c \in \mathcal{G} \setminus R(\mathcal{G})$ it holds that source$(c) = c'$ if the center of c falls in the Voronoi region of $p(c')$. Computing the Voronoi diagram of the river cells can be done in $O(\text{sort}(N))$ I/Os [10, 2]. Then, we sweep simultaneously the diagram and grid $\mathcal{G}$. During the sweep, we maintain the diagram edges that intersect the sweep line, sorted according to the x-coordinate of their intersection point with this line. For every row of $\mathcal{G}$ that we encounter, we scan the edges that intersect the sweep line to determine the Voronoi region (and therefore the corresponding source cell) where each cell in the row belongs to. Notice that the number of edges in the sweep line cannot be more than two times the number of cells in a row. This is because there cannot be more than two river cells on a single column whose Voronoi regions intersect the same horizontal line. Therefore, scanning the raster and updating the sweep line can be done efficiently in $O(\text{sort}(N))$ I/Os in total. From this we conclude that computing the source cells on the raster can be performed in $O(\text{sort}(N))$ I/Os. But we can do this more efficiently; in the appendix we present an algorithm that computes the source cells using $O(\text{scan}(N))$ I/Os.

Computing the flood resistance values.

In the second step of method *ProximityResistance* we compute for each cell $c \in \mathcal{G}$ its flood resistance f(c). Recall that for every cell c this resistance value is equal to the minimum rise value h_{rise} such that obst$(c) \leqslant h_{\text{rise}}$ and c is connected to the river by a path of cells with obstruction $\leqslant h_{\text{rise}}$. Based on this definition, we can reduce the computation of the flood resistance values to the problem of computing the *raise elevations* on a terrain, that was described by Arge *et al.* as part of their partial flooding algorithm [8]. This problem is defined as follows; let $\mathcal{G}$ be a raster and let $\zeta_1, \ldots, \zeta_k$ be a set of cells in $\mathcal{G}$ that we call *sinks*. For any path of cells path in $\mathcal{G}$ the height of path is defined as the height of the highest cell on this path. The *raise elevation* of a cell $c \in \mathcal{G}$ is the minimum height among all paths that connect c to ζ_i for any $1 \leqslant i \leqslant k$. Arge *et al.* provide an algorithm that computes the raise elevations for all the cells on the terrain in $O(\text{sort}(N))$ I/Os [8].

We can reduce the problem of computing the flood resistance values of the cells in $\mathcal{G}$ to an instance of the raise elevation problem as follows; we create a raster $\mathcal{G}'$ that has the same number of rows and columns as $\mathcal{G}$. For any river cell $\mathcal{G}(i,j) \in R(\mathcal{G})$ we let the corresponding cell $\mathcal{G}'(i,j)$ to be a sink. For any non-river cell $\mathcal{G}(i,j)$ we let cell $\mathcal{G}'(i,j)$ have elevation equal to the obstruction value of $\mathcal{G}(i,j)$. It is know easy to see that the raise value of any cell $\mathcal{G}'(i,j)$ is equal to the flood resistance value that we want to compute for $\mathcal{G}(i,j)$. By applying the I/O-efficient algorithm of Arge *et al.* on $\mathcal{G}'$ we can compute the described flood resistance values in $O(\text{sort}(N))$ I/Os.

THEOREM 3.1. Let $\mathcal{G}$ be a raster terrain that consists of N cells, and let $R(\mathcal{G})$ be the set of all river cells in this raster. *ProximityResistance* computes the flood resistance values of all cells in $\mathcal{G}$ using $O(\text{sort}(N))$ I/Os.

3.2 Our New Method

In *ProximityResistance*, a cell c can only get flooded from the closest river cell source(c) in the xy-plane. Intuitively, this is very unnatural since the flow of water on the terrain is obviously influenced by the terrain topography. Therefore, we introduce a novel method which instead chooses source(c) based on a model that represents how water flows on the terrain. We refer to this new method as *UpstreamResistance*. Below, we first describe how source(c) is chosen in *UpstreamResistance*, and then we show how we can compute this I/O-efficiently.

For a raster $\mathcal{G}$ let $\mathcal{F}(\mathcal{G}) = (V, E)$ be the graph such that for each cell $c \in \mathcal{G}$ there exists exactly one vertex $v(c)$ in V, and there exists a directed edge in E from $v(c)$ to $v(c')$ if cells $c, c' \in \mathcal{G}$ are adjacent and $h(c) > h(c')$. We call this graph the *flow graph* of $\mathcal{G}$. For now let us assume that no adjacent cells in $\mathcal{G}$ have the same elevation value. Hence, there exists exactly one directed edge in $\mathcal{F}(\mathcal{G})$ for each pair of adjacent cells in $\mathcal{G}$, and $\mathcal{F}(\mathcal{G})$ is a DAG. The concept of the flow graph was introduced in previous works to model how water flows between cells on a DEM. It is naturally assumed that water on a cell can flow only to neighbour cells with lower height; that is modelled with a directed edge in the flow graph.

For any cell $c \in G$ water from c may flow following different routes on the raster until reaching one or more cells on the boundary of river $R(\mathcal{G})$. In method *UpstreamResistance* we choose source(c) to be one of these cells on the river boundary, that is, the river cells where the water from c reaches. More formally, let c be a cell in $\mathcal{G}$. Consider a path in $\mathcal{F}(\mathcal{G})$ that starts from vertex $v(c)$ and ends at a vertex $v(c')$ where c' is a river cell, such that the path does not contain a vertex corresponding to any other river cell. We call such a path a *downstream path* of c. Let $DC(c)$ denote the set of all river cells that belong to some downstream path of c. In method *UpstreamResistance*, source(c) is the cell in $DC(c)$ with the highest elevation value. The flood resistance of c is then defined as the height difference $h(c) - h(\text{source}(c))$.

When it comes to implementing *UpstreamResistance* I/O-efficiently, the two key tasks for computing the flood resistances are constructing the flow graph $\mathcal{F}(\mathcal{G})$, and computing the source cell for every cell in $\mathcal{G}$. If no flat areas exist on $\mathcal{G}$, we can construct $\mathcal{F}(\mathcal{G})$ straightforwardly in $O(\text{scan}(N))$ I/Os. As for computing the source cells, observe that for any cell c it holds that source$(c) =$ source(c') for some c' such that there exists an edge in $\mathcal{F}(\mathcal{G})$ from $v(c)$ to $v(c')$. Therefore, we can compute source(c) by first computing the source cells for those neighbours of c that appear downstream in $\mathcal{F}(\mathcal{G})$, and then use these to infer source(c). Arge *et al.* describe an I/O-efficient algorithm that computes the number of upstream cells for every cell on a raster in $O(\text{sort}(N))$ I/Os [4]. Their algorithm can be easily modified for computing the source cells in $\mathcal{G}$. Therefore, we can perform this computation in $O(\text{sort}(N))$ I/Os.

Handling flat areas.

Terrain datasets often contain large connected regions of cells that have exactly the same elevation. Given raster $\mathcal{G}$ that contains such flat areas, we have to perform two extra steps; first, we have to outline all distinct flat areas in $\mathcal{G}$, and then we have to model how water flows on each such area. For the second step, we will have to modify the definition of the flow graph so as to represent flow between cells in a flat area. To outline the flat areas in $\mathcal{G}$ we have to compute the connected components of cells in the raster that have the same elevation. This can be done I/O-efficiently in $O(\text{sort}(N))$ I/Os. Let $A \subseteq \mathcal{G}$ be a flat area in $\mathcal{G}$, and

let c be a cell on the boundary of A. We say that c is a *spill point* of A if c is adjacent to at least one cell that has elevation lower than $h(c)$. When modelling water flow on A the goal is to route flow so that every cell in A drains to at least one spill point of this area (if such a spill point exists). Let A be a flat area that has at least one spill point. Every cell $c \in A$ can be routed to all the spill points in A. Therefore, all cells in A should have exactly the same source cell and also the same flood resistance. To represent this appropriatelly, we modify slightly the way we construct the flow graph such that instead of representing each cell in A by exactly one vertex, we use a single vertex to represent the entire flat area. We denote this vertex by $v(A)$. The in-edges of $v(A)$ connect this vertex with all vertices $v(c)$ such that c is a cell adjacent to A, and c has a higher elevation than A. Similarly, the out-edges of $v(A)$ connect to all lower elevation vertices in the graph representing cells adjacent to A.

After building the flow graph in this manner, we proceed with the computation of the resistance values. We processing a vertex that represents a flat region A, we distinguish two cases depending on whether A contains river cells or not. In the case that A does not contain any river cell, we find the highest river cell c that appears on a downstream path from $v(A)$, and for every cell in A we set the flood resistance to the elevation difference between A and c. In the case that A contains river cells, we consider that all cells in this area are flooded by default. Hence, for every cell in A we set the flood resistance value to zero. In that case, vertex $v(A)$ is treated in the flow graph in the same way as a vertex that represents a single river cell; for each cell c such that $v(A)$ appears in a downstream path from c we use the elevation of A to determine the flood resistance of c, as if A was a single river cell.

Note that not all flat areas have a spill point. In that case, a flat area is a region of locally minimum elevation in $\mathcal{G}$. Let A be such an area in $\mathcal{G}$. If A does not contain any river cell then we consider that A corresponds to a spurious pit. We remove all such pits by raising the elevation of the terrain within and around this region. The removal of the spurious pits can be done as a preprocessing step before constructing the actual flow graph on $\mathcal{G}$. We can do this I/O-efficiently in $O(\text{sort}(N))$ I/Os, using the partial flooding algorithm described by Danner *et al.* [8]. On the other hand, if A contains river cells, then we process this area in the same way as we did with flat areas that contain both river cells and spill points; the flood resistance of all cells in A are set to zero, and the entire area is represented by a single vertex $v(A)$ in the flow graph.

From the above description, we conclude that constructing the flow graph for a raster $\mathcal{G}$ boils down to computing the connected components of flat regions in $\mathcal{G}$, and then adding in the described way the graph edges between the vertices of the flat regions and the adjacent cells. We can compute the connected components of flat regions on $\mathcal{G}$ using the batched union-find algorithm of Agarwal *et al.* [1] which uses $O(\text{sort}(N))$ I/Os. Constructing the modified flow graph, and processing the vertices that represent flat areas can be done straightforwardly in $O(\text{sort}(N))$ I/Os.

THEOREM 3.2. Let $\mathcal{G}$ be a raster terrain that consists of N cells, and let $R(\mathcal{G})$ be the set of river cells in this raster. *UpstreamResistance* computes the flood resistance values of all cells in $\mathcal{G}$ using $O(\text{sort}(N))$ I/Os.

4. IMPLEMENTATIONS AND EXPERIMENTS

We implemented both algorithms described in Section 3 in order to evaluate how fast they perform in practice, as well as how good they model real flood events.

4.1 Description of Implementations

We implemented both algorithms in C++, using the open source library TPIE that provides I/O-efficient algorithms for sorting and scanning data [13]. We used the GNU g++ compiler (version 4.8.2), and the experiments were ran on a Linux Ubuntu operating system (release 14.04).

When implementing *ProximityResistance* we made two modifications compared to the description in Section 3. First, when computing the source cells on $\mathcal{G}$ using a sweepline approach, we used the $O(\text{scan}(N))$ approach (described in the appendix) and we made the practically realistic assumption that a constant number of rows in $\mathcal{G}$ can fit in main memory. Thus, instead of performing an external scan of each row and maintaining an I/O-efficient stack during the sweep, we simply store the two last rows that we swept in memory and perform all computations internally. Second, when computing the raise elevations we did not use the $O(\text{sort}(N))$ batched union-find algorithm by Agarwal *et al.* [1] (that is quite involved), but instead a much simpler $O(\text{sort}(N)\log(N/M))$ algorithm also proposed by Agarwal *et al.*. Agarwal *et al.* [1] and Danner *et al.* [8] have showed that this simple union-find algorithm performs very well in practice.

When implementing *UpstreamResistance* we accurately followed the description in Section 3. The only difference was that we again used the practical union-find algorithm of Agarwal *et al.* (that requires $O(\text{sort}(N)\log(N/M))$ I/Os), this time for computing the connected components of flat areas in $\mathcal{G}$, and for removing flat areas that correspond to spurious pits.

4.2 Measuring I/O-Efficiency in Practice

To measure the practical efficiency of each method, we ran our implementations on a massive raster dataset that represents the terrain surface of the entire country of Denmark. Publicly available through the website of the Danish Ministry of Environment [7], this raster consists of roughly 66.4 billion cells, arranged in 287500 rows and 231250 columns. Each cell represents a region of 1.6×1.6 meters on the terrain and is assigned an elevation value which is a 4-byte floating point number. The total size of the uncompressed dataset is 268 gigabytes. We refer to this dataset as `denmark`.

Raster `denmark` does not include any river data, and therefore we had to extract the river cells before conducting the experiments. To do so, we first preprocessed the raster by removing all shallow pits. Then, we selected the river cells based on the size of their upstream area. For this reason, we computed the flow graph of `denmark` as described in Section 3 except that for each cell c we included at most one outgoing edge. This outgoing edge points to the vertex $v(c')$ such that c' is a neighbour of c and the vector from $p(c)$ to $p(c')$ has the steepest downward slope. Then, we computed for each cell c the size of its upstream area; this is the area that is covered by all cells c' such that there exists a path from $v(c')$ to $v(c)$ in the flow graph. We extracted the river cells by selecting all cells whose upstream area was larger than 12.5 km^2. We picked this threshold since the resulting river network resembles better the actual shape of the rivers in Denmark, according to available orthophotos.

We ran both of our algorithms on the `denmark` raster and the extracted cells, using a workstation that has a Xeon CPU (W3565), a four-core processor with 3.2GHz per core. The workstation had 48 Gigabytes of main memory, and a raid (redundant array of independent disks) that consists of nineteen disks, with 3 Terrabytes capacity in total. The maximum amount of main memory that was available at any point during the execution of our implementations was 22 Gigabytes. The total time taken by the implementation of *ProximityResistance* was roughly 24.2 hours; only 2.4 hours was used for computing the source cell for each non-river cell, and the rest 21.8 hours were spent on computing the resistance values. For the implementation of *UpstreamResistance*, the total execution time was approximatelly 31.1 hours. The first stage of this method, where the flow graph of the input raster is computed, took 12.5 hours. The rest 18.6 hours were spent for delineating the flat areas on the terrain, and computing the resistance values.

From the above, it is clear that the implementations of both methods have a very good performance even for an enormous dataset such as `denmark`. Each method took less than one and a half day to process this dataset, using an amount of main memory which corresponds to roughly 8% of the datasets total size.

4.3 Evaluating the Quality of Flood Modelling

In the second set of experiments we used an actual flood event to evaluate the quality of the output produced by the two methods, namely the catastrophic flood of the Indus river that took place in Pakistan in 2010 [6].

For the experiments we used a raster terrain extracted from the SRTM grid, a DEM that represents the earth surface from 60° North to 56° South [12]. The extracted raster covers a square region of approximately 2160×2160 kilometers and includes the entire Indus river basin–see Fig. 1. The raster consists of 24,000×24,000 cells, and the dimension of each square cell is approximately 90 meters. We refer to this dataset as `indus`.

Since the `indus` DEM does not contain any river data, we extracted the river cells based on the upstream area of each cell, in the same way as we did for the `denmark` dataset in Section 4.2. In this case we used an upstream area of 300 km^2 since it produces a visual result that matches the shape of the local river network, as it appears in orthophotos acquired when there was no flood in the region.

To evaluate the ability of our algorithms to accurately model floods, we used a vector dataset that shows the actual flooded regions around the river during the Indus river flood. This dataset was released by the Dartmouth Flood Observatory, and contains data acquired with MODIS (Moderate-resolution Imaging Spectroradiometer) technology [9]. We refer to this dataset as `flood`. The `flood` dataset was constructed based on several satellite photos of the Indus region, acquired during the period from the 1st to the 5th of August of 2010. It represents with polygons all the regions that were flooded in at least one day during this period. The bounding box of `flood` covers a rectangular region that spans approximatelly 1118 and 911 kilometers on the langitudinal and the longitudinal axes respectively. It contains 4294 polygons, and the total area covered by these polygons is approximatelly 30483 km^2. Refer to Fig. 1.

We ran the implementations of *ProximityResistance* and *UpstreamResistance* algorithms on the `indus` DEM and the extracted river cells, and we evaluated the output of each algorithm using a method that resembles the *Area-Under-the-Curve* (also known as AUC) measure, which is one of the most popular measures for model testing [5]. In particular, we overlayed `flood` with `indus` and extracted the cells in `indus` whose centers lie in the interior of a polygon in `flood`. We refer to these cells as the *flooded* cells of `indus`. In total, we identified 4045544 flooded cells. Next we selected at random a large set of pairs of cells. Each pair was selected so that it consists of one flooded cell and one non-flooded cell. We denote this set of pairs by $\mathcal{P}$. For each of our methods, we determined for each pair $pr \in \mathcal{P}$ if the flooded cell in pr scores a higher resistance value than the non-flooded cell, and calculated the percentage of the pairs in $\mathcal{P}$ for which this condition holds. We call this percentage the *output quality* of the method. The value of the output quality is an estimation of the AUC measure; the output quality value is equal to the AUC if $\mathcal{P}$ consists of all possible pairs of flooded/non-flooded cells in the region of interest. For our study, we chose 10^5 pairs, considering that this is a sufficient number for estimating the value of the AUC. For method *ProximityResistance* the output quality is 87%, while for *UpstreamResistance* the output quality is 92%. This shows clearly that both of the methods produce flood resistances that are highly consistent with the actual event.

To measure how the two methods perform on a more local scale, we calculated their output quality within several smaller regions. More specifically, within the xy-region covered by `flood` we extracted three sets of square windows, each set consisting of windows of certain size. In the first set each window is a square with dimension 20 km, in the second set each window has dimension 40 km, and the third set consists of windows of 80 km dimension. The windows of each set were picked in the following way. Within the region covered by `flood` we extracted at random 500 windows of the same size. Then we used a greedy algorithm to select a subset of these windows, so that there is no pair of windows in the subset that overlap with each other, and so that each window contains at least 500 flooded and at least 500 non-flooded cells. Thus, we ended up with a subset of 119 windows for the first set, and forty-five and twenty-two windows for the second and third set respectively. From each window, we selected 10^5 cell pairs, again so that each pair contains one flooded and one non-flooded cell. We then calculated the output quality of our methods for each window. Figure 4.3 shows the results for the windows of 20 km dimension, where the mean output quality was 61% for *ProximityResistance* and 71% for *UpstreamResistance*. For windows of 40 km dimension, *ProximityResistance* attained mean output quality 69% and *UpstreamResistance* mean output quality 81%. For the third set of windows, the values were 76% and 85%, respectively.

Therefore, for each window size *UpstreamResistance* has higher mean output quality than *ProximityResistance*. For both methods the output quality increases as the window size becomes larger. Yet, we observed that for all examined window sizes, there exist windows were at least one of the methods has an output quality value of less than 50%.

To examine the above further, we investigated if there is a correlation between the output quality values and the two following factors: heterogeneity of the terrain (variability of elevation values) and the number of flooded cells inside

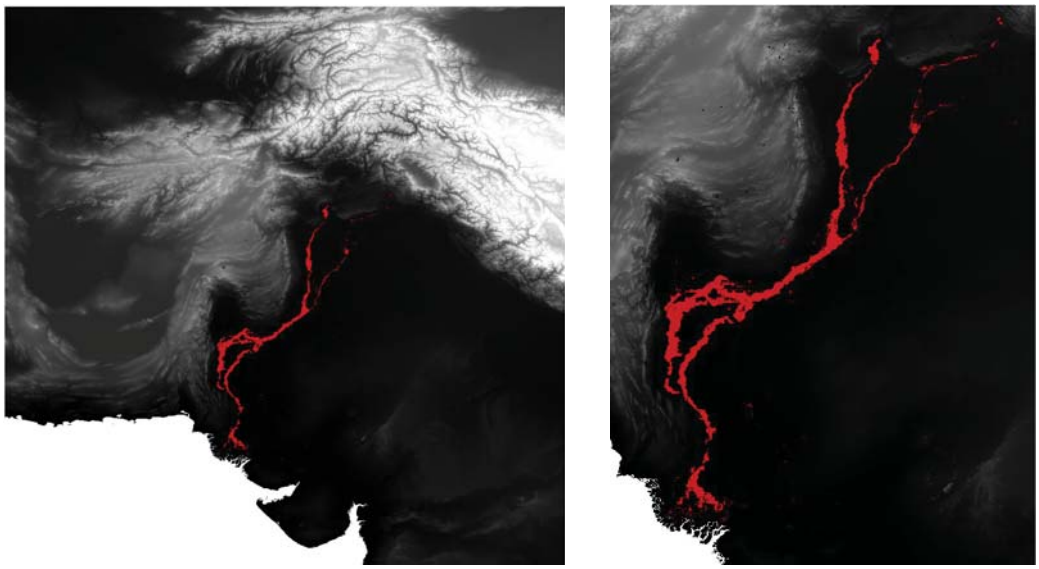


Figure 1: (a) An illustration of the `indus` DEM together with the `flood` vector dataset. The cells of the DEM appear in grayscale colours, shaded according to their elevation values; cells of higher elevation are indicated by lighter shades. The polygons of the `flood` dataset appear in red colour. (b) A closer view of the flooded regions.

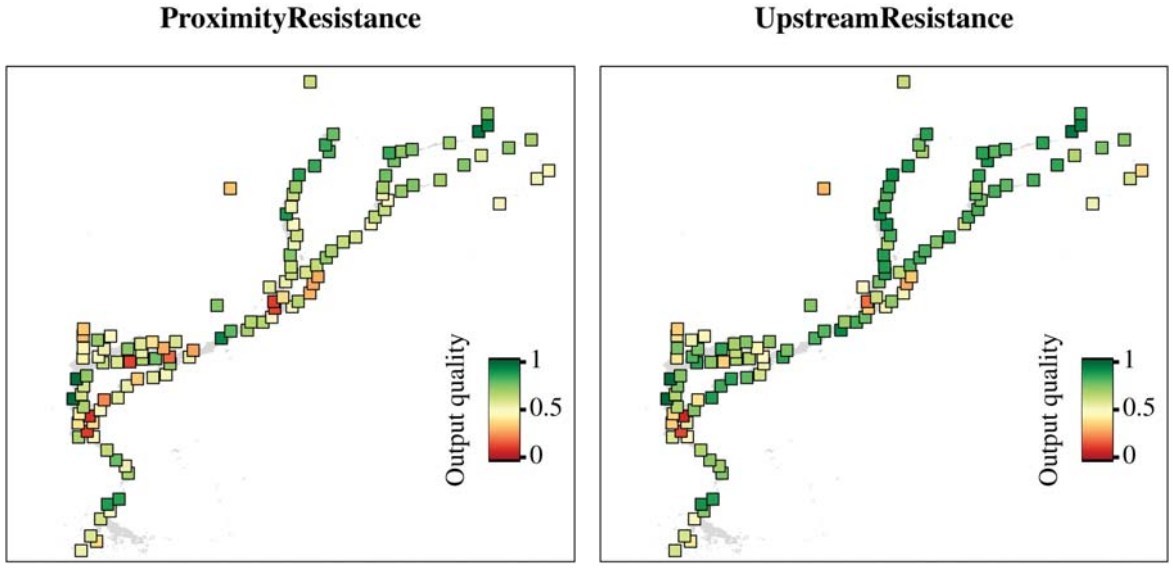


Figure 2: The locations for the selected windows of 20 km dimension. In each subfigure, a window is represented by a colored box. Each box is colored according to the output quality value achieved by one of the methods for the corresponding window. The relative size of the boxes in the figure is larger than the size of the original windows. We did this to make the color of each box more visible. The xy-regions of the original windows do not overlap with each other. Left: boxes colored based on the output quality values for `ProximityResistance`. Right: boxes colored according to the output quality values of `UpstreamResistance`.

each window. To measure the heterogeneity of the terrain within each window w, we computed the logarithm of the standard deviation for the elevations of the cells in w. We call this value the *topographic heterogeneity* of w. In order to examine visually the relation between the output quality and the topographic heterogeneity among the different windows, we created a scatter plot for each method. Each scatter plot contains a 2-dimensional point $p(w)$ for every window w; the horizontal coordinate of $p(w)$ is equal to the topographic heterogeneity of w, and the vertical coordinate of this point is equal to the output quality of the method for w. Figure 4.4 shows the scatter plots that we produced for windows of 20 km dimension. In a similar way, we created a scatter plot for each method where the horizontal coor-

dinates of the presented points are equal to the number of flooded cells in the windows that we examine. These scatter plots appear also in Figure 4.4. It becomes evident that both of the methods score higher output quality values for windows of intermediate topographic heterogeneity. Most of the low output quality values appear on windows of small heterogeneity. Regions that consist mainly of flat areas belong to this category. Also, there does not appear to be any relation between the output quality of the methods and the number of flooded cells within each window. The visualisations that we produced for the windows of larger size showed similar patterns.

4.4 Comparing the Output of the Methods

To gain more insight about *ProximityResistance* and *UpstreamResistance*, we visually examined the output that the two methods produced for the `denmark` dataset. During this examination, for various rise values ρ we extracted the regions in the output of each method which consisted of all cells with flood resistance $\leqslant \rho$. The first observation we made was that for the same rise value the size of the flooded area that appears in the output of *ProximityResistance* is larger than in the output produced by *UpstreamResistance*. Refer to Figure 4(a) and Figure 4(b). One of the reasons that contributes to this difference is how the two methods estimate river floods around coastlines; in the output of *ProximityResistance*, almost the entire coastline of the terrain appears flooded even for very small rise values. Refer to Figure 4(c). We can explain this as follows. Recall that with *ProximityResistance* a cell c gets flooded for a rise value ρ if a) this cell has a height difference $\leqslant \rho$ from the closest river cell on the xy-domain (the obstruction value), and b) if there is a path from c to any river cell such that the obstruction values of all cells in the path is $\leqslant \rho$. The terrain cells which appear close to the coastline have low height values, since they lie almost on the sea level. Therefore, for each such cell the height difference from the closest river cell is either very small or negative, which means that the coastline constitutes a path of cells that connects to the river and all cells in this path have very low obstruction values. As a consequence, even for small rise values all cells in this path are flooded when using *ProximityResistance*. On the other hand, in the output of the *UpstreamResistance* method, coastlines do not appear flooded even for large rise values. The reason is that for a coastline cell there is usually no flow path that connects this cell with a river cell. Hence, cell c can not get flooded whichever the river-rise value. We also observed one more artefact in the output of method *ProximityResistance*; in some cases, this method produces flooded regions with long linear boundaries that do not correspond to actual obstacles on the elevation profile of the terrain. Refer to Figure 4(d). These artefacts are the result of assigning obstruction values to non-river cells based on the Voronoi diagram of the river cells on the xy-domain of the terrain. In an area that extends between two different river streams, this step may produce two regions of cells that have a large difference in their obstruction values. The boundary between these two regions follows the boundaries between Voronoi regions of river cells that belong to different streams. As a consequence, for certain rise values there appear flooded areas in the output whose boundary follows the boundary between the Voronoi regions of the river cells.

5. REFERENCES

[1] P. K. Agarwal, L. Arge, and K. Yi. I/O-Efficient Batched Union-Find and its Applications to Terrain Analysis. In *Proc. 22nd Annual Symposium on Computational Geometry*, pages 167–176, 2006.

[2] P. K. Agarwal, L. Arge, and K. Yi. I/O-Efficient Construction of Constrained Delaunay Triangulations. In *Proc. 13th Annual European Symposium on Algorithms*, pages 355–366, 2005.

[3] A. Aggarwal and J. S. Vitter. The Input/Output Complexity of Sorting and Related Problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[4] L. Arge, L. Toma, and J. S. Vitter. I/O-Efficient Algorithms for Problems on Grid-Based Terrains. *ACM Journal on Experimental Algorithmics*, 6(1), 2001.

[5] U. Brefeld and T. Scheffer. AUC maximizing support vector learning. In *Proc. ICML Workshop on ROC Analysis in Machine Learning*, 2005.

[6] Encyclopaedia Britannica Webpage, Pakistan Floods of 2010. `http://www.britannica.com/EBchecked/topic/1731329/Pakistan-Floods-of-2010`.

[7] Elevation Model of Denmark, Geodata Agency of the Danish Ministry of Environment. `http://gst.dk/emner/frie-data/hvilke-data-er-omfattet/hvilke-data-er-frie/dhm-danmarks-hoejdemodel/`.

[8] A. Danner, T. Mølhave, K. Yi, P.K. Agarwal, L. Arge, and H. Mitasova. TerraStream: From Elevation Data to Watershed Hierarchies. In *Proc. 15th ACM International Symposium on Advances in Geographic Information Science (ACM GIS)*, pages 212–219, 2007.

[9] The Dartmouth Flood Observatory Webpage. `http://floodobservatory.colorado.edu/`.

[10] M.T. Goodrich, J.J. Tsay, D.E. Vengroff and J.S. Vitter. External-Memory Computational Geometry. In *Proc. 34th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 714–723, 1993.

[11] B. Kronvang, M. Søndergaard, C.C. Hoffmann, H. Thodsen, N. Bering Ovesen, M. Stjernholm, C. B. Nielsen, C. Kjærgaard, B. Schønfeldt, and B. Levesen. Etablering af P-Ådale. Technical report 840, National Environmental Research Institute of Denmark, 2011.

[12] E. Rodriguez, C.S. Morris, and J.E. Belz. (2006). A Global Assessment of the SRTM Performance. *Photogrammetric Engineering & Remote Sensing*, 72(3):249–260, 2006.

[13] TPIE, the Templated Portable I/O Environment. `http://www.madalgo.au.dk/tpie/`.

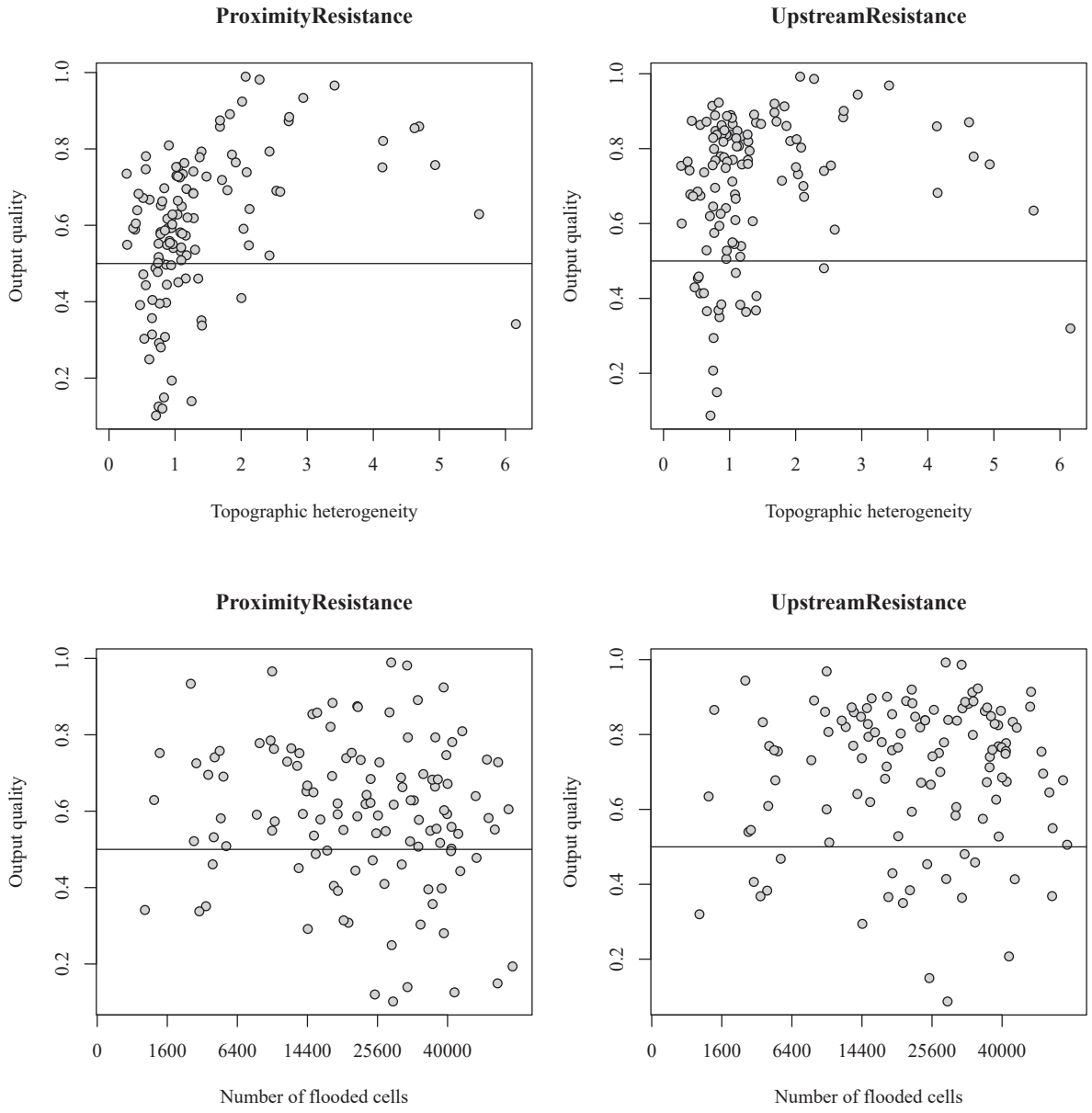
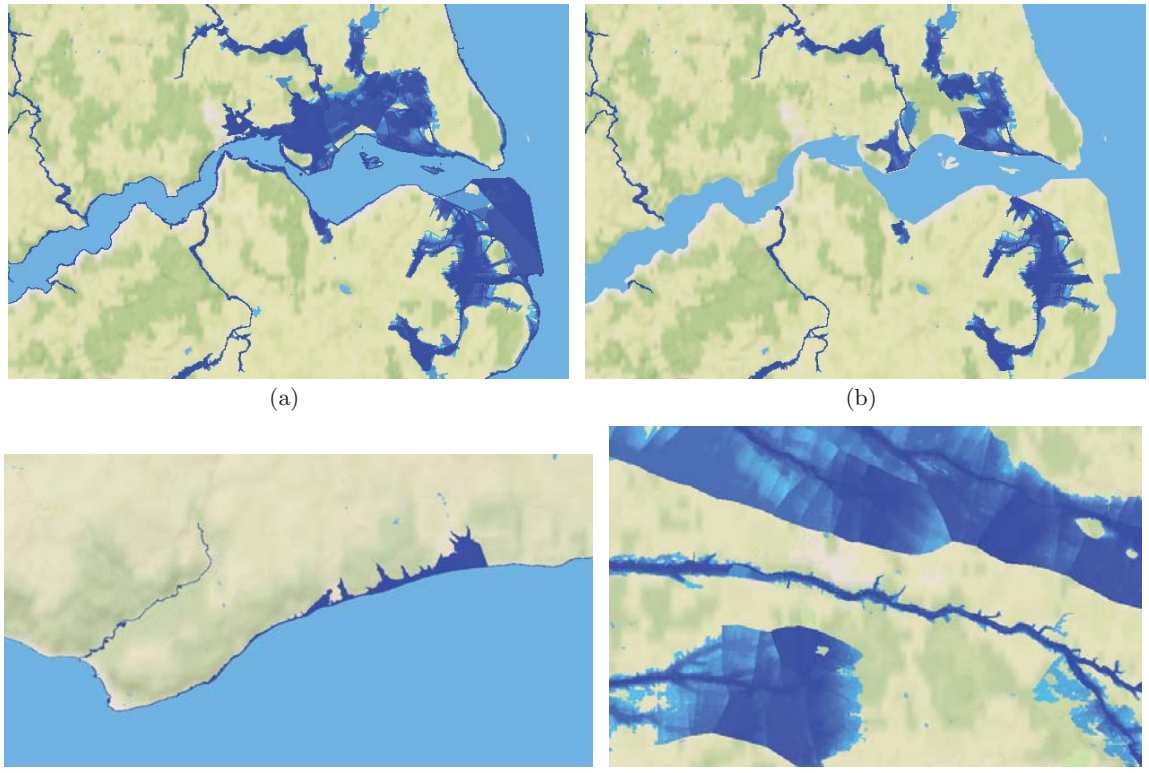


Figure 3: Scatter plots that show the relation between the output quality of each method and two features of the examined windows. Each point corresponds to a window of 20 km dimension. Top: The relation between output quality and topographic heterogeneity. Bottom: The relation between output quality and the number of flooded cells in each window.



(a) (b)

(c) (d)

Figure 4: An illustration of the outputs of *ProximityResistance* and *UpstreamResistance* for the `denmark` dataset. Flooded regions are indicated by dark blue color. (a) The output of the *ProximityResistance* around Hadsund town (north-east Jutland) for a rise value of half a meter. (b) The output of *UpstreamResistance* for the same region and rise value. (c) The output of *ProximityResistance* close to Vejle city with a rise value of just one milimeter. The entire coast appears flooded, with wide flooded areas at certain places. (d) The output of the *ProximityResistance* on a region with several river streams, for a rise value of 2.8 meters.

APPENDIX

In Section 3.1 we claimed that, given a grid $\mathcal{G}$ and the set of river cells $R(\mathcal{G})$ on this grid, we can compute for every cell $c \in \mathcal{G}$ its source cell using only $O(\text{scan}(N))$ I/Os. Recall that the source cell of c is the cell in $R(\mathcal{G})$ that has the closest xy-distance to c. Next we describe in detail how we can do this computation while achieving the claimed performance bound.

Instead of computing explicitly the Voronoi diagram of centers of the cells in $R(\mathcal{G})$, we use a simpler approach. For every cell $c = \mathcal{G}(i, j)$ we compute three cells: The closest river cell $\mathcal{G}(k, l)$ such that $k < i$, the closest river cell $\mathcal{G}(k', l')$ such that $k' > i$, and the closest river cell $\mathcal{G}(k'', l'')$ such that $k'' = i$. We indicate these cells by $\text{source}_<(c)$, $\text{source}_>(c)$, and $\text{source}_=(c)$, respectively. Obviously, one of these three cells is the source cell of c; given those cells we can determine $\text{source}(c)$ by simply comparing their xy-distances from c. We can calculate these three cells for every cell in $\mathcal{G}$ in $O(\text{scan}(N))$ I/Os using a sweep-line technique. Next we show how we can do this for $\text{source}_<(c)$ and $\text{source}_=(c)$; the process for computing $\text{source}_>(c)$ is quite similar to the one for $\text{source}_<(c)$.

Let $\text{row}(i)$ denote the i-th row in $\mathcal{G}$, where $\text{row}(1)$ is the bottom row in the raster. Let $\mathcal{VD}(i)$ indicate the Voronoi diagram on the xy-domain of $\mathcal{G}$ where the sites are the centers of all river cells $\mathcal{G}(k, l)$ such that $k < i$. For every cell $c \in \text{row}(i)$, the cell $\text{source}_<(c)$ corresponds to the site in $\mathcal{VD}(i)$ whose Voronoi region contains the center of c. Based on this observation, to compute $\text{source}_<(c)$ for every cell $c \in \mathcal{G}$ we scan $\mathcal{G}$ row by row, starting from the bottom row of the raster. During this process, we maintain a sweepline l_s parallel to the y-axis. When we process $\text{row}(i)$, we consider that the sweepline has the same y-coordinate as the centers of the cells of this row. We use $l_s(i)$ to indicate the sweepline at that moment. Line $l_s(i)$ intersects the Voronoi regions of certain sites in $\mathcal{VD}(i)$ and is thus subdived into horizontal segments, each contained in a different region. To represent this we compute a list of points L_i; these are the intersection points between $l_s(i)$ and the voronoi boundaries of the sites that appear below $l_s(i)$. With every point $p \in L_i$ we also store at most two river cells $rc_{\text{left}}(p)$ and $rc_{\text{right}}(\text{p})$. Cell $rc_{\text{left}}(p)$ is the river cell whose Voronoi region intersects l_s to the left of p, while $rc_{\text{right}}(p)$ is the river cell whose Voronoi region intersects l_s to the right of this point. The points in L_i are stored in order of increasing x-coordinate. Given the list L_i, it is easy to compute the source cells for every cell c in $\text{row}(i)$; we scan this row and L_i simultaneously, while keeping track of the cell centers in $\text{row}(i)$ that fall inside the Voronoi region delimited by two consecutive points in L_i.

Computing L_i for each row is a bit more involved. We can do this efficiently as follows; to calculate the elements in L_i we look at the list L_{i-1} that we computed for the previous row. First, we construct an intermediate list L_{temp} that has a structure similar to L_i except that it stores the intersections points between $l_s(i)$ and the Voronoi regions in $\mathcal{VD}(i-1)$. Then, we update this intermediate list so that also the regions of the sites in $\text{row}(i-1)$ are taken into account.

To construct L_{temp} we need to find where the boundaries of the Voronoi regions of $\mathcal{VD}(i-1)$ intersect $l_s(i)$. To do so for every region boundary that intersects $l_s(i-1)$ we could simply compute where this boundary intersects $l_s(i)$. However, not all region boundaries in $\mathcal{VD}(i-1)$ that intersect $l_s(i-1)$ intersect also $l_s(i)$; this is the case when there exist regions in $\mathcal{VD}(i-1)$ whose y-span ends somewhere between $l_s(i)$ and $l_s(i-1)$. This can be handled in the following way; we scan list L_{i-1} and we maintain an I/O-efficient stack ST that stores are the bisectors between Voronoi sites in $\mathcal{VD}(i-1)$ whose regions potentially intersect $l_s(i)$. More specifically, let p_{curr} be the element that is currently scanned in L_{i-1}. We maintain the invariant that ST stores the bisectors for only those sites in $\mathcal{VD}(i-1)$ whose Voronoi regions: a) intersect $l_s(i-1)$ to the left of p_{curr}, and b) would intersect $l_s(i)$ if we did not consider the sites in $\mathcal{VD}(i-1)$ whose regions intersect $l_s(i-1)$ to the right of p. We also maintain that for any two lines l_1 and l_2 in the stack, line l_1 is stored below l_2 if and ony if l_1 crosses $l_s(i)$ to the left of l_2. With each line l that we store in ST we also maintain the two sites $rc_{\text{left}}(l)$ and $rc_{\text{right}}(l)$ for which l is the bisector. Initially ST is empty. Recall that p_{curr} is the intersection point between the sweepline $l_s(i-1)$ and the bisector line of two neighbouring sites $rc_{\text{left}}(p)$ and $rc_{\text{right}}(p)$. Let l_{bisect} be this line for p_{curr}, the element currently processed in L_{i-1}. Let $\text{top}(ST)$ indicate the bisector currently stored at the top of the stack ST. We compute the intersection point between l_{bisect} and $\text{top}(ST)$; if this point has a y-coordinate that falls above $l_s(i)$, or below $l_s(i-1)$ we push l_{bisect} into the stack and we continue with the next element in L_{i-1}. Otherwise, we compute the bisector between $rc_{\text{left}}(\text{top}(ST))$ and $rc_{\text{right}}(p)$ and we set l_{bisect} to represent this line. Then, we remove the line at the top of the stack and compute the intersection point between the current l_{bisect} and the line that is now stored $\text{top}(ST)$. We repeat this process until either l_{bisect} and $\text{top}(ST)$ intersect at a point above $l_s(i)$ or below $l_s(i-1)$, or ST becomes empty. It is easy to prove that after processing all the elements in L_{i-1}, stack ST stores the support lines of exactly those Voronoi boundaries in $\mathcal{VD}(i-1)$ that intersect $l_s(i)$.

Given the data stored in ST we can easily construct the intermediate list L_{temp}; recall that this is the list that stores the intersection points between $l_s(i)$ and the bisectors of the Voronoi regions in $\mathcal{VD}(i-1)$. To construct L_i, we then process L_{temp} together with the river cells in $\text{row}(i-1)$. First, we compute the Voronoi diagram of the river cells in $\text{row}(i-1)$; this diagram is trivial to compute since all sites have the same y-coordinate, and the bisectors between the Voronoi regions are lines parallel to the y-axis. For each river cell c in $\text{row}(i-1)$ we compute the interval which corresponds to the x-span of the Voronoi region of c in this diagram. We store each such interval together with corresponding cell in a list L_{int}, in increasing order of the x-coordinates of their endpoints. Next, we use the intervals in L_{int} and the information stored in L_{temp} in order to compute the elements of L_i, the intersection points between the river cells in $\mathcal{VD}(i)$ and $l_s(i)$. For this, we need to partition $l_s(i)$ into intervals, such that each interval corresponds to the intersection of $l_s(i)$ with a Voronoi region in $\mathcal{VD}(i)$. Each such interval is either a subset of an interval in L_{int}, or a subset of an interval represented by two consecutive elements in L_{temp}. Therefore, we can compute L_i by simultaneously scanning L_{temp} and L_{int}, and substituting, inserting, or deleting points from L_{temp} using the information stored in L_{int}.

Let k be the number of cells in a single row of $\mathcal{G}$. From the above description, we conclude that constructing each list L_i requires $O(\text{scan}(k))$ I/Os since we need to scan $\text{row}(i-1)$,

list L_{i-1}, and L_{temp} a constant number of times. We also have to insert and extract at most $O(k)$ elements to and from the I/O-efficient stack ST. We can insert or extract a single element from such a stack in $O(1/B)$ I/Os, which sums up to $O(\text{scan}(k))$ I/Os for processing all cells in a row.

Processing L_i to determine the values $\text{source}_<$ also requires one scan, which adds up in total to $O(\text{scan}(N))$ I/Os for handling the corresponding lists for all rows in $\mathcal{G}$.

Computing $\text{source}_=(c)$ is a simple task; let c be any cell in $\mathcal{G}$, and let r be the row in $\mathcal{G}$ which contains c. Cell $\text{source}_=(c)$ is either the closest river cell on r that appears on the left side of c, or the closest river cell from the right side of c. Therefore, to determine $\text{source}_=(c)$ we scan each row of $\mathcal{G}$ independently, and for each cell that we are currently scanning we keep track of the nearest river cell from each side on this row. Hence, we can compute cell $\text{source}_=(c)$ for every cell $c \in \mathcal{G}$ in $O(\text{scan}(N))$ I/Os. From the above description, we conclude that we can compute $\text{source}_>(c)$, $\text{source}_<(c)$, $\text{source}_=(c)$, and therefore the source cell $\text{source}(c)$ for every cell $c \in \mathcal{G} \setminus R(\mathcal{G})$ in $O(\text{scan}(N))$ I/Os.

The Influence of Late Quaternary Climate-Change Velocity on Species Endemism

B. Sandel,[1,2]* L. Arge,[2] B. Dalsgaard,[3] R. G. Davies,[4] K. J. Gaston,[5] W. J. Sutherland,[3] J.-C. Svenning[1]

The effects of climate change on biodiversity should depend in part on climate displacement rate (climate-change velocity) and its interaction with species' capacity to migrate. We estimated Late Quaternary glacial-interglacial climate-change velocity by integrating macroclimatic shifts since the Last Glacial Maximum with topoclimatic gradients. Globally, areas with high velocities were associated with marked absences of small-ranged amphibians, mammals, and birds. The association between endemism and velocity was weakest in the highly vagile birds and strongest in the weakly dispersing amphibians, linking dispersal ability to extinction risk due to climate change. High velocity was also associated with low endemism at regional scales, especially in wet and aseasonal regions. Overall, we show that low-velocity areas are essential refuges for Earth's many small-ranged species.

Anthropogenic climate change is a major threat to Earth's biodiversity and the ecosystem services it provides (*1*). As climate changes, the conditions suitable for local persistence of a particular species move across the surface of the Earth, driving species responses both to recent warming (*2*–*4*) and to long-term natural climate cycles (*5*–*8*). Species with strong dispersal abilities inhabiting relatively stable climates may track climate fairly closely. Conversely, species with weak dispersal abilities relative to the rate of climate change may fail fully to occupy climatically suitable areas (*9*–*14*) and may even go extinct, despite appropriate habitat being present elsewhere (*15*–*17*).

Climate-change velocity is a measure of the local rate of displacement of climatic conditions over Earth's surface (*18*). It integrates macroclimatic shifts with local spatial topoclimate gradients and is calculated by dividing the rate of climate change through time by the local rate of climate change across space, yielding a local instantaneous velocity measure. By describing the local rate at which species must migrate to track changing climate, climate-change velocity is more biologically relevant than are traditional macroclimatic anomalies (*13*, *16*, *19*, *20*). Furthermore, because climate-change velocity incorporates fine-scale topoclimate gradients it captures the important buffering effect of topographic heterogeneity on climate change (*21*). For example, a given temperature change should have very different biological consequences depending on topographic context: In flat areas, considerable movement is required to track a 1°C increase in temperature, whereas a short shift uphill could be sufficient in mountains. Thus, species distributed in topographically homogenous landscapes will experience higher climate-change velocities and therefore require stronger dispersal abilities to track climate change than those of species in heterogeneous landscapes.

High climate-change velocities are likely to be associated with incomplete range filling and species extinctions (*22*). Not all species, however,

[1]Ecoinformatics and Biodiversity Group, Department of Bioscience, Aarhus University, Aarhus 8000 C, Denmark. [2]Center for Massive Data Algorithmics (MADALGO), Department of Computer Science, Aarhus University, Aarhus 8000 C, Denmark. [3]Conservation Science Group, Department of Zoology, University of Cambridge, Cambridge CB2 3EJ, UK. [4]School of Biological Sciences, University of East Anglia, Norwich NR4 7TJ, UK. [5]Environment and Sustainability Institute, University of Exeter, Cornwall TR10 9EZ, UK.

*To whom correspondence should be addressed. E-mail: brody.sandel@biology.au.dk

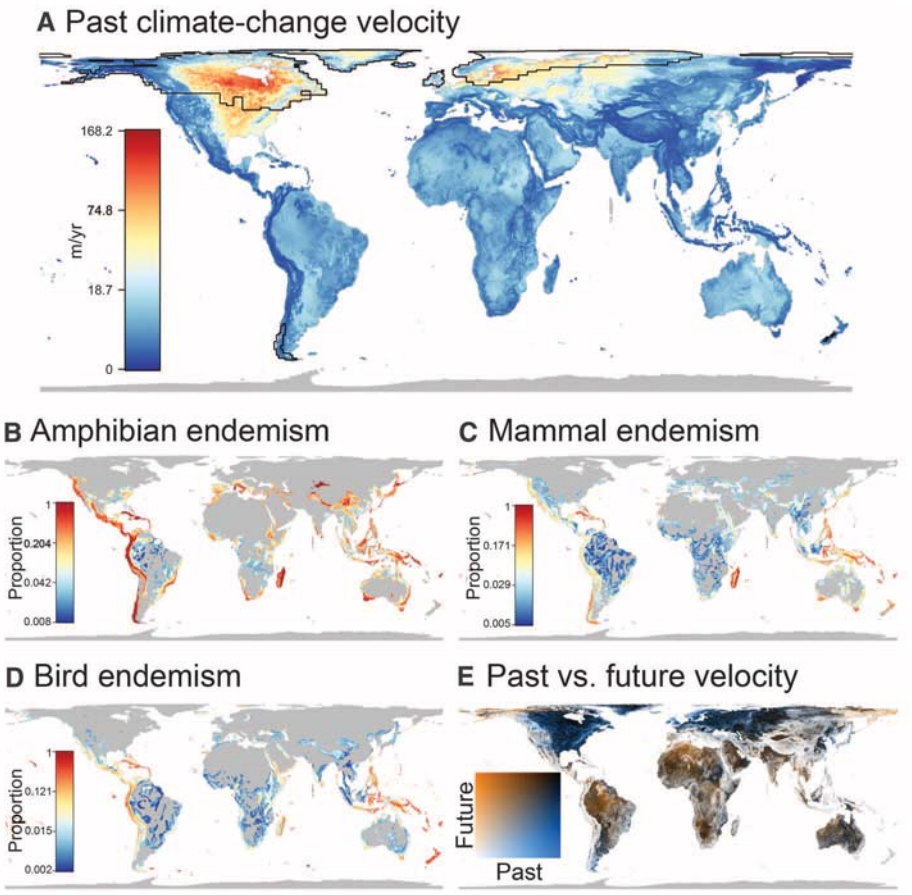


Fig. 1. Global maps of (**A**) climate-change velocity since the Last Glacial Maximum (21,000 years ago); proportional endemism of (**B**) amphibians, (**C**) mammals, and (**D**) birds; and (**E**) relationships of past and predicted future climate-change velocity. Velocity is highest in northeastern North America and north-central Eurasia; these same areas display low or no endemism (the black line delimits areas that were glaciated at LGM). Mountain ranges and other low-velocity regions exhibit higher endemic richness of amphibians, mammals, and birds. Rank-transformed velocity since the LGM and rank-transformed velocities until 2080 show similar spatial patterns, but there are areas of important mismatch. Orange areas, where velocities in the past have been low but predicted future velocities are expected to be high, are a particular conservation concern.

are at equal risk from high velocity (*20*). Strong dispersers should be most able to maintain distributional equilibrium with climate conditions and are therefore likely to occupy more of their potential range and avoid extinction. Species with small ranges are at particular risk of extinction (*20*); they often have small population sizes and densities (*23*) and are less likely to occupy refuges that remain suitable during climate oscillations (*15*, *16*, *24*). Range size may also reflect other species characteristics that influence resilience to climate fluctuation, with widespread species tending to have broad climatic tolerances (*25*), generalist strategies (*23*), and strong dispersal capabilities (*23*). Hence, high-velocity regions should have fewer small-ranged species and fewer species with poor dispersal ability (*16*).

To test these hypotheses, we used reconstructions of mean annual temperature at the Last Glacial Maximum (LGM; 21,000 years ago) to calculate a global map of climate-change velocity from the LGM to the present (*26*) and tested the effects of velocity on patterns of range size and species richness of amphibians, mammals, and birds (*27*). The difference between the LGM and the present is one of the strongest climatic shifts in all of the Quaternary (*28*), and its spatial pattern is probably similar to the spatial patterns of earlier climate cycles (*16*). Our analysis revealed that the LGM-to-present climate-change velocity exhibits marked geographic variation, with peaks in northeastern North America and north-central Eurasia (Fig. 1A). Velocities tended to be lower in the Southern Hemisphere and in mountainous areas.

Globally, small-ranged (<250,000 km^2; hereafter termed "endemic") amphibians, mammals, and birds are concentrated where climate-change velocity is low (Fig. 1, B to D) [results were not sensitive to the particular definition of small ranges (*27*)]. In high-velocity northeastern North America and north-central Eurasia, endemic species are nearly absent, whereas low-velocity areas often harbor highly endemic faunas. For all species groups, low-velocity sites are more likely to harbor endemic species than are high-velocity sites (Fig. 2, insets; logistic regression with spatial filters: $n = 2000$ grid cells, all $P < 0.0001$). In addition, among regions (10° × 10° equivalents) containing at least one endemic species, velocity is strongly and negatively related to the proportion of amphibian [bivariate regression; $n = 188$ regions, coefficient of determination (r^2) = 0.283, $P < 0.001$], mammal ($n = 231$ regions, $r^2 = 0.328$, $P < 0.001$), and bird ($n = 240$ regions, $r^2 = 0.385$, $P < 0.001$) species that are endemic (Fig. 2). We excluded glaciated areas (Fig. 1A) from this analysis, but investigations showed that results were not sensitive to this decision.

It is widely accepted that modern climate influences species distributions and diversity, whereas the role of historical determinants is less clear (*23*). We therefore examined models that incorporated descriptions of modern climate, the spatial pattern of modern climate conditions, and climate-change velocity. The spatial pattern of modern climate conditions was summarized by calculating, for each grid cell, the total area of land within a 1000-km radius that had a mean annual temperature (MAT) within 1°C and mean annual precipitation (MAP) within 100 mm of that focal cell (*24*). The extent of analogous modern climate exerted strong influences on endemism, reflecting the fact that regionally rare climates are likely to contain small-ranged species (Table 1) (*24*). Endemism of all three groups was negatively related to productivity and to two measures of seasonality. In models that considered all variables together while controlling for spatial autocorrelation [simultaneous autoregressive models (SARs)], velocity was a highly significant negative predictor of endemism for all groups, and for the amphibians was second in importance only to the extent of analogous climate. We examined the effect of adding terms to this model to describe the interaction of velocity and modern climate descriptors. Including these terms generally had small effects on other coefficient estimates, and only one such interaction was significant (velocity × precipitation seasonality for amphibians), so these interactions were not considered further. We examined all subsets of the full SAR models and compared them using Akaike's Information Criterion (AIC). For all groups, models incorporating velocity were always strongly preferred over equivalent models without velocity (mean AIC improvement: amphibians, 29.6; mammals, 38.5; and birds, 39.0). Across the full model set with and without climate-change velocity, there was high support for velocity as part of the best model (summed Akaike weights for velocity > 0.989 for all groups). These results show that past climate-change velocity and modern climate act together to determine global patterns of endemism.

Because velocity incorporates fine-scale topographic effects on climate stability, it should also contribute to within-region variation in endemism. To test this, we divided the world into regions (10° × 10° equivalents) and asked whether velocity was correlated with endemism within these regions. For all three species groups, high-velocity areas within regions were associated with low endemism (global means of within-region correlation coefficients: amphibians, $r = -0.160$; mammals, $r = -0.157$; and birds, $r = -0.091$, all $P < 0.01$) (Fig. 3). This overall pattern is consistent with the loss of small-ranged species in high-velocity regions, but the local importance of velocity showed strong geographic variation. Velocity should have the strongest impact where the climate is sometimes highly suitable for a given group, potentially favoring the generation and maintenance of endemic diversity (*29*). In contrast, regions characterized by generally unfavorable conditions are expected to contain few endemics, whether or not velocity is low. In addition, the ability of species to cope with temperature fluctuations is thought to vary spatially, with species in highly seasonal

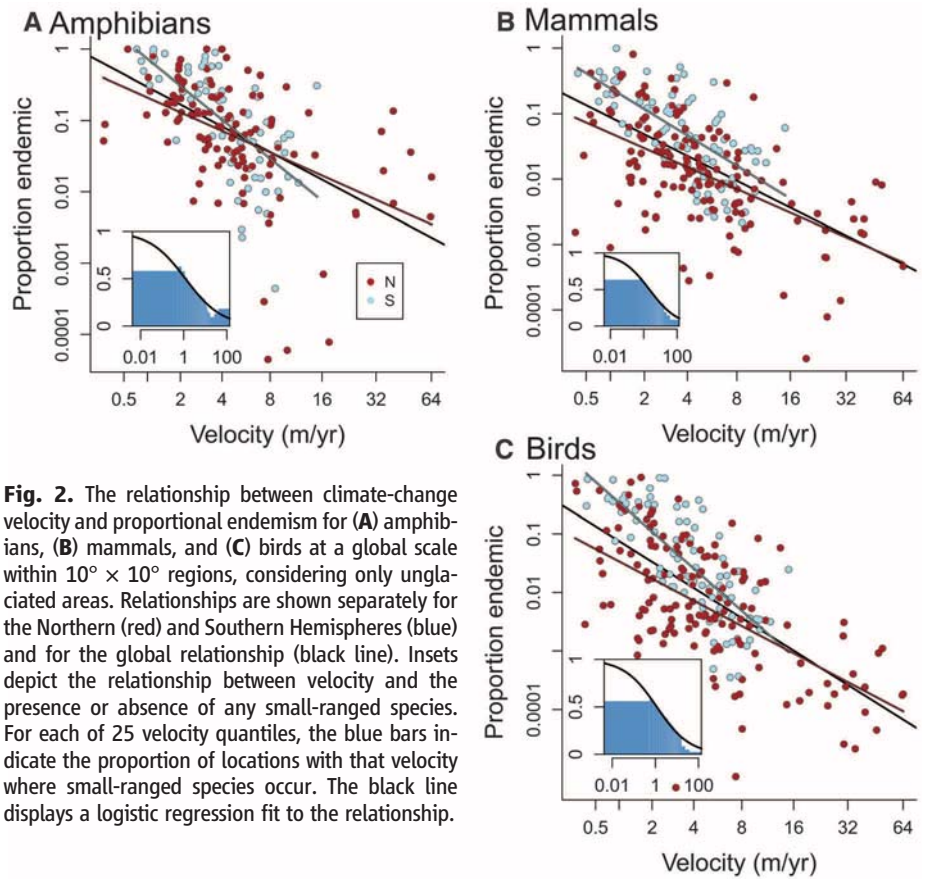


Fig. 2. The relationship between climate-change velocity and proportional endemism for (**A**) amphibians, (**B**) mammals, and (**C**) birds at a global scale within 10° × 10° regions, considering only unglaciated areas. Relationships are shown separately for the Northern (red) and Southern Hemispheres (blue) and for the global relationship (black line). Insets depict the relationship between velocity and the presence or absence of any small-ranged species. For each of 25 velocity quantiles, the blue bars indicate the proportion of locations with that velocity where small-ranged species occur. The black line displays a logistic regression fit to the relationship.

areas tolerating a wider temperature range (*30*). We tested these hypotheses by identifying the modern climatic variables that most strongly control the within-region correlation of velocity and endemism patterns (table S9). For all three species groups, the models with the lowest AIC scores contained a single significant predictor and were consistent with the above predictions; velocity was particularly important for amphibians and mammals where precipitation was high [amphibians, n = 125 regions, standardized regression coefficient (β) = –0.185, P = 0.0465; mammals, n = 149 regions, β = –0.259, P = 0.0018) and for birds where temperature seasonality was low (n = 135 regions, β = 0.334, P = 0.0102) (Fig. 3).

Although a comprehensive test has thus far been lacking, the importance of climate-change velocity for a group should depend on the dispersal abilities of its species. Hence, of the taxa analyzed, birds should be best at tracking high-velocity changes, whereas amphibians should be worst (*9*). Indeed, the importance of velocity in determining global patterns of endemism decreased from amphibians (standardized β = –0.288) to mammals (standardized β = –0.216) to birds (standardized β = –0.183) (Table 1). Furthermore, velocity was most tightly correlated with patterns of within-region endemism for amphibians and least correlated for birds. Lending additional support to the importance of dispersal ability, velocity was also more tightly associated with patterns of nonvolant mammal endemism (standardized global β = –0.166, P = 0.0058, mean local r = –0.165) than with bat endemism (standardized global β = –0.135, P = 0.0670, mean local r = –0.034). Global patterns of endemism were well-correlated among the three groups, suggesting that overall they respond to similar drivers. However, regions with low amphibian endemism relative to avian endemism (which are correlated, r = 0.681) tended to have high velocity, indicating that amphibians respond more strongly than do birds to climate change (multiple regression of amphibian endemism against velocity and avian endemism, n = 183 regions, standardized $\beta_{velocity}$ = –0.281, P = 0.0010). Similar results were obtained for amphibian endemism relative to mammalian endemism (r = 0.693, n = 185 regions, $\beta_{velocity}$ = –0.283, P = 0.0010) and mammalian endemism relative to avian endemism (r = 0.819, n = 212 regions, $\beta_{velocity}$ = –0.175, P = 0.0028). Richness of the lowest three range-size quartiles in amphibians is low relative to the equivalent avian richness (n = 175 regions, $\beta_{velocity}$ = –0.247, P = 0.0031) and mammal richness (n = 173 regions, $\beta_{velocity}$ = –0.333, P < 0.0001) in high-velocity regions. As expected given their low dispersal ability, amphibian assemblages in high-velocity regions were thus particularly depauperate of endemic species relative to bird and mammal assemblages.

The focal relationship between velocity and endemism is corroborated by other patterns in the distributions of all three groups. At both global and regional spatial scales, high velocity was associated with larger median range sizes, lower variation in range size within assemblages, reduced richness of species with range size below the median, and weaker, inconsistent, and sometimes positive relationships for richness of species with larger range sizes (figs. S1 to S6 and tables S1 to S6 and S9).

High velocities have been proposed to be associated with incomplete range filling and species extinctions (*10*, *17*, *22*). Although we found no evidence for reduced range sizes with higher velocities, the decline in endemism and inferred importance of dispersal ability are consistent with the extinctions hypothesis. However, it may be that velocity per se is not driving endemism but simply correlates with other variables that do have a direct mechanistic link. Alternatively, a direct mechanistic link between velocity and endemism may not require species extinctions. We considered several such alternative hypotheses and show that none are consistent with the data.

High-velocity areas may coincide with those where analogous climate conditions have most expanded since the LGM; low endemism in these areas may occur because species in such areas have expanded their range size accordingly. Climate expansion since the LGM was highly correlated with the extent of modern analogous conditions (r = 0.794) but was a weaker predictor of endemism in all cases (compare Table 1 with table S12). Furthermore, the range expansion hypothesis predicts that the groups with strongest dispersal ability (birds) should have expanded their ranges most and therefore should show the strongest relationships to velocity, which is contrary to our results. It is also possible that velocity appears to be an important predictor, not because of a mechanistic link but only because it is derived from another variable that does have a direct link. However, climate-change velocity was a better predictor than either of its components

Table 1. Relationships of seven predictor variables to global patterns of proportional endemism for amphibians, mammals, and birds. GVI, generalized vegetation index; MAT, mean annual temperature; MAP, mean annual precipitation; TS, temperature seasonality; PS, precipitation seasonality; Extent, the regional extent of analogous climate; and Velocity, Late Quaternary climate-change velocity. Five comparative measures were used: the coefficient of determination from bivariate regressions (r^2), standardized regression coefficients estimated from ordinary least-squares multiple regression (OLS), simultaneous autoregressive models using all predictors (SAR), the reduced SAR model with the lowest AIC score ($SAR_{reduced}$), and Akaike weights based on SAR models. Blank cells indicate variables that were not included in the reduced model. Figures in text are based on the full SAR models, which were most consistently successful at removing residual spatial autocorrelation. The effect of the change in MAT between LGM and present (Anomaly) and topographic heterogeneity (TH) were tested by replacing velocity in all models with each of these variables. *P < 0.05, **P < 0.01, ***P < 0.001.

	r^2	OLS	SAR	$SAR_{reduced}$	$Weight_{AIC}$
Amphibians					
GVI	0.227***	–0.185**	–0.196**	–0.198**	0.989
MAT	0.002	0.010	0.008		0.294
MAP	0.090***	–0.125	–0.115	–0.11	0.460
TS	0.068***	–0.018	0.001		0.312
PS	0.073***	–0.233***	–0.244***	–0.241***	0.997
Extent	0.415***	–0.439***	–0.414***	–0.411***	1.000
Velocity	0.283***	–0.252***	–0.288***	–0.289***	1.000
Anomaly	0.082***	–0.203**	–0.213**	–0.207***	0.979
TH	0.134***	0.230**	0.260***	0.285***	0.998
Mammals					
GVI	0.253***	–0.169***	–0.160***	–0.160***	0.993
MAT	0.158***	0.228**	0.216*	0.216*	0.882
MAP	0.203***	–0.215**	–0.190*	–0.190*	0.736
TS	0.313***	–0.180	–0.182	–0.182	0.597
PS	0.039**	–0.194***	–0.173***	–0.173***	0.929
Extent	0.538***	–0.490***	–0.467***	–0.467***	1.000
Velocity	0.328***	–0.199***	–0.216***	–0.216***	0.998
Anomaly	0.187***	–0.155**	–0.158**	–0.158**	0.975
TH	0.023*	0.092	0.106*	0.106*	0.646
Birds					
GVI	0.297***	–0.237***	–0.212***	–0.213***	1.000
MAT	0.102***	0.065	0.021		0.284
MAP	0.137***	–0.317***	–0.303***	–0.302***	0.999
TS	0.348***	–0.500***	–0.549***	–0.565***	1.000
PS	0.036**	–0.183***	–0.120*	–0.114*	0.838
Extent	0.446***	–0.369***	–0.323***	–0.320***	1.000
Velocity	0.385***	–0.194***	–0.183***	–0.180***	0.989
Anomaly	0.222***	–0.148***	–0.121*	–0.119*	0.733
TH	0.015	0.074	0.086*	0.081*	0.726

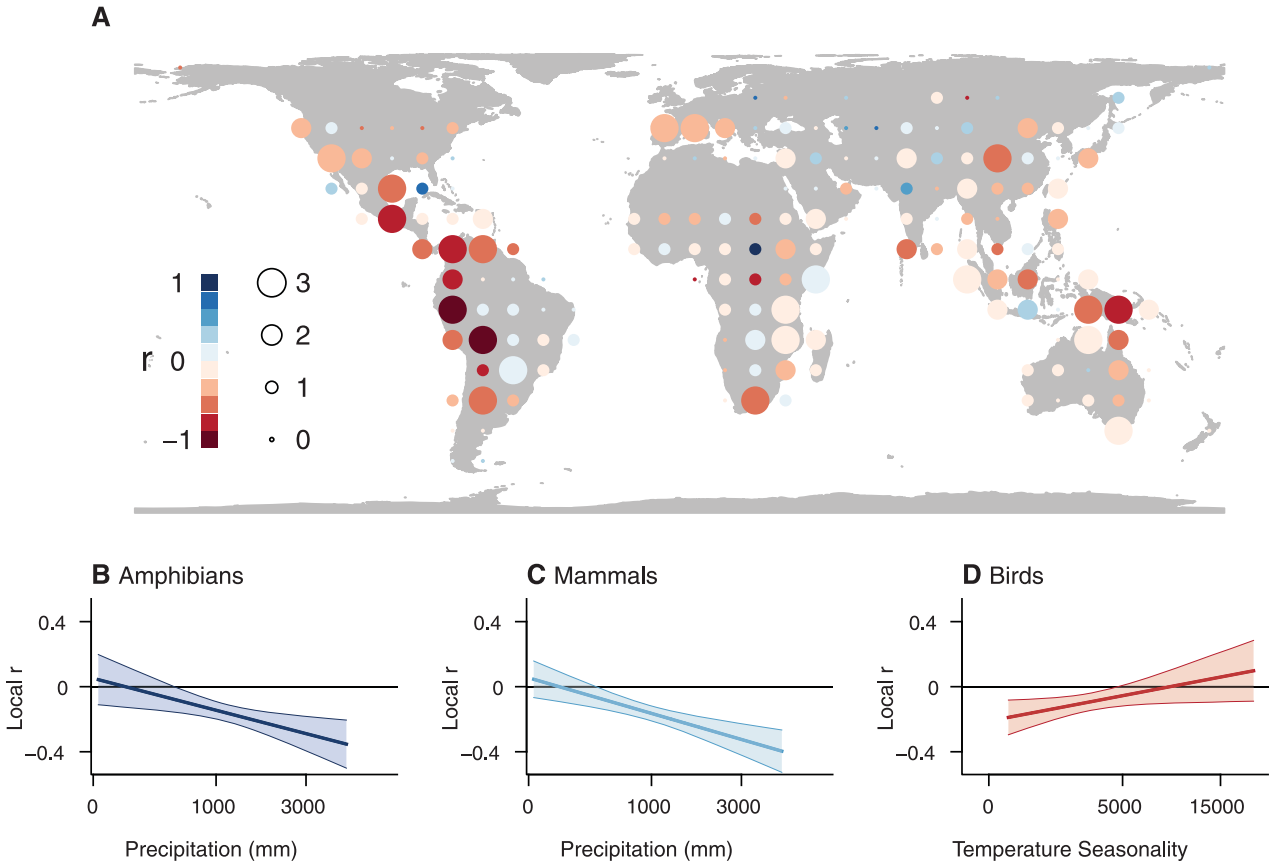


Fig. 3. The global pattern of within-region fit of climate-change velocity to patterns of endemism. (**A**) Within each 10° × 10° region, circle size indicates the number of species groups for which velocity was significantly ($P < 0.05$) associated with endemism. The color of the circle corresponds to the mean correlation coefficient across all groups present within that region. (**B** to **D**) Globally, strong within-region correlations between velocity and endemism were associated with high precipitation (amphibians and mammals) and low temperature seasonality (birds). Envelopes around each line show the 90% confidence interval.

(topographic heterogeneity and macroclimate anomaly) and was the only one with consistent predictive power across scales (Table 1), which suggests that topographic heterogeneity and anomaly both contain important and complementary information. High endemism and low velocity occur not only in mountains but also in macroclimatically stable, relatively flat regions (such as portions of the Amazon basin and central Africa). At the same time, high-velocity mountain areas harbor low endemism (such as the Altai mountains of northern Mongolia and some mountains in the western United States).

Does a direct role of velocity depend on extinctions, and are alternatives consistent with the data? Dynesius and Jansson (*15*) proposed three interrelated mechanisms linking climate stability and patterns of richness and range size: (i) Instability may select for increased dispersal ability and generalism, leading to large ranges; (ii) gradual speciation rates may be reduced in unstable areas, producing a lack of young, small-ranged species; and (iii) small-ranged species may have gone extinct in unstable areas. Underlying all of these hypotheses is the process of lineage extinction across a range of evolutionary scales, from selection acting on within-species lineages to the extinction of newly diverging or full species (*31*). In principal, reduced speciation rates [explanation (ii)] might instead be due to high rates of gene flow among populations in unstable, shifting climates. However, this mixing should be most pronounced for strongly dispersing species. In contrast, lineage extinctions at all levels should have the strongest effects on weak dispersers, which is consistent with our results. Thus, elevated extinction rates—likely across a range of evolutionary scales (*31*)—appear to best explain the association of low endemism with high velocity (*17*, *22*).

Our results have important implications for conservation in a world that is increasingly experiencing elevated climate-change velocities (*18*). Areas that have experienced high velocities in the past are on average also expected to experience high velocities over the next century (Fig. 1E and fig. S7). As we have shown, these areas are already missing small-ranged species, suggesting that most of the remaining species may cope well with future changes. However, there are important mismatches between the spatial patterns of past and future climate change; areas with low velocities in the past, high concentrations of endemic species, and high velocities into the future are a particular conservation concern. These areas include western Amazonia, where concentrations of endemic species that have experienced relatively low-velocity changes in the past may be faced with rapid climate shifts in the near future.

Taken together, these results indicate that past climate changes have left important legacies in contemporary range size and species richness patterns, supplemented by the influences of modern climate and its spatial pattern. Small-ranged species constitute most of Earth's species diversity (*23*); our findings show that these species, especially those from less vagile groups, are sensitive to climate movements and are concentrated in areas where possibilities for tracking past climate changes have been greatest. This conclusion also suggests that small-ranged, weakly dispersing species in previously stable regions experiencing high future climate-change velocities will be at greatest extinction risk from anthropogenic climate change.

References and Notes

1. C. Parmesan, G. Yohe, *Nature* **421**, 37 (2003).
2. R. K. Colwell, G. Brehm, C. L. Cardelús, A. C. Gilman, J. T. Longino, *Science* **322**, 258 (2008).
3. J. Lenoir, J. C. Gégout, P. A. Marquet, P. de Ruffray, H. Brisse, *Science* **320**, 1768 (2008).
4. C. Moritz *et al.*, *Science* **322**, 261 (2008).
5. B. Huntley, T. Webb III, *J. Biogeogr.* **16**, 5 (1989).
6. P. K. Schoonmaker, D. R. Foster, *Bot. Rev.* **57**, 204 (1991).
7. M. S. McGlone, *Global Ecol. Biogeogr. Lett.* **5**, 309 (1996).
8. M. B. Davis, R. G. Shaw, *Science* **292**, 673 (2001).
9. M. B. Araújo, R. G. Pearson, *Ecography* **28**, 693 (2005).
10. J.-C. Svenning, F. Skov, *Ecol. Lett.* **7**, 565 (2004).
11. J.-C. Svenning, F. Skov, *Glob. Ecol. Biogeogr.* **16**, 234 (2007).

12. J.-C. Svenning, F. Skov, *Ecol. Lett.* **10**, 453 (2007).
13. M. B. Araújo *et al*., *Ecography* **31**, 8 (2008).
14. J.-C. Svenning, S. Normand, F. Skov, *Ecography* **31**, 316 (2008).
15. M. Dynesius, R. Jansson, *Proc. Natl. Acad. Sci. U.S.A.* **97**, 9115 (2000).
16. R. Jansson, *Proc. Biol. Sci.* **270**, 583 (2003).
17. J.-C. Svenning, *Ecol. Lett.* **6**, 646 (2003).
18. S. R. Loarie *et al*., *Nature* **462**, 1052 (2009).
19. R. Jansson, T. J. Davies, *Ecol. Lett.* **11**, 173 (2008).
20. T. J. Davies, A. Purvis, J. L. Gittleman, *Am. Nat.* **174**, 297 (2009).
21. D. Scherrer, C. Körner, *J. Biogeogr.* **38**, 406 (2010).
22. D. Nogués-Bravo, R. Ohlemüller, P. Batra, M. B. Araújo, *Evolution* **64**, 2442 (2010).
23. K. J. Gaston, *The Structure and Dynamics of Geographic Ranges* (Oxford Univ. Press, New York, 2003).
24. R. Ohlemüller *et al*., *Biol. Lett.* **4**, 568 (2008).
25. G. C. Stevens, *Am. Nat.* **133**, 240 (1989).
26. This measure does not account for abrupt or transient changes within the time interval. Over periods of decades or centuries, relatively rapid changes may produce velocities considerably higher than those obtained by using just the LGM and present.
27. Materials and methods are available as supporting material on *Science* Online.
28. W. F. Ruddiman, *Earth's Climate: Past and Future* (W.H. Freeman and Company, New York, 2001).
29. D. J. Currie *et al*., *Ecol. Lett.* **7**, 1121 (2004).
30. C. K. Ghalambor, R. B. Huey, P. R. Martin, J. J. Tewksbury, G. Wang, *Integr. Comp. Biol.* **46**, 5 (2006).
31. A. C. Carnaval, M. J. Hickerson, C. F. B. Haddad, M. T. Rodrigues, C. Moritz, *Science* **323**, 785 (2009).

Acknowledgments: We thank the Aarhus University Research Foundation for financial support. This study was also supported in part by MADALGO–Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation. B.D. is supported by the Danish Council for Independent Research–Natural Sciences, and W.J.S. is funded by Arcadia. We thank international climate modeling groups for providing their data for analysis, the Laboratoire des Sciences du Climat et de l'Environnement for collecting and archiving the paleoclimate model data, the International Union for Conservation of Nature and Natural Resources for making the amphibian and mammal range data available, and the Natural Environment Research Council–funded Avian Diversity Hotspots Consortium (NER/O/S/2001/01230) for the use of the bird range data. We thank four anonymous reviewers whose constructive comments improved this manuscript. Data are archived at Dryad (http://dx.doi.org/10.5061/dryad.b13j1).

Supporting Online Material

www.sciencemag.org/cgi/content/full/science.1210173/DC1
Materials and Methods
Figs. S1 to S12
Tables S1 to S12
References (*32*–*54*)

22 June 2011; accepted 19 August 2011
Published online 6 October 2011;
10.1126/science.1210173

Long-Term Change in the Nitrogen Cycle of Tropical Forests

Peter Hietz,[1]* Benjamin L. Turner,[2] Wolfgang Wanek,[3] Andreas Richter,[4] Charles A. Nock,[5] S. Joseph Wright[2]

Deposition of reactive nitrogen (N) from human activities has large effects on temperate forests where low natural N availability limits productivity but is not known to affect tropical forests where natural N availability is often much greater. Leaf N and the ratio of N isotopes ($\delta^{15}N$) increased substantially in a moist forest in Panama between ~1968 and 2007, as did tree-ring $\delta^{15}N$ in a dry forest in Thailand over the past century. A decade of fertilization of a nearby Panamanian forest with N caused similar increases in leaf N and $\delta^{15}N$. Therefore, our results indicate regional increases in N availability due to anthropogenic N deposition. Atmospheric nitrogen dioxide measurements and increased emissions of anthropogenic reactive N over tropical land areas suggest that these changes are widespread in tropical forests.

Anthropogenic N fixation has approximately doubled atmospheric deposition of reactive N in terrestrial ecosystems globally, with regional variation resulting from differences in the intensity of agriculture, the burning of fossil fuels, and biomass burning (*1*). Many temperate and boreal ecosystems are N limited; in these regions, atmospheric N deposition has caused the acidification of soils and waters, loss of soil cations, a switch from N to P limitation, a decline in the diversity of plant communities adapted to low N availability, and increases in carbon uptake and storage (*2*, *3*). Natural N availability is much greater in many tropical forests than in most temperate forests due to high rates of N fixation by heterotrophic soil microbes and rhizobia associated with legumes, which are abundant in many tropical forests (*4*). Nitrogen deposition is increasing in the tropics, and this region may see the most dramatic increases in the coming decades (*1*). It has been hypothesized that this will acidify soils, deplete soil nutrients, reduce tree growth and carbon storage, and negatively affect biodiversity in tropical forests (*5*, *6*). Yet despite extensive speculation, there remains no direct evidence for changes in the N cycle in tropical forests.

The ratio of stable N isotopes ($\delta^{15}N$) reflects the nature of the N cycle in ecosystems, with higher values indicating greater N availability and a more open N cycle (*7*, *8*). In temperate ecosystems where N deposition is low, leaf N concentrations and the $\delta^{15}N$ of leaves and wood decreased during the 20th century, indicating progressive N limitation in response to changes in land use (*9*) and increasing atmospheric CO_2 concentrations (*10*). In contrast, wood $\delta^{15}N$ values have increased in temperate forests with high rates of N deposition or a history of recent disturbance, suggesting more open N cycles under such conditions (*11*, *12*).

We compared leaves from herbarium specimens (158 species) collected ~40 years ago (~1968) from a tropical moist forest on Barro Colorado Island (BCI), Republic of Panama, with sun and shade leaves (340 species) collected in 2007. Over four decades, leaf $\delta^{15}N$ increases averaged 1.4 ± 0.16 per mil (‰) (SEM) and 2.6 ± 0.1‰ when comparing 1960s leaves to conspecific 2007 shade and sun leaves, respectively. Based on their leaf mass per area, 1960s leaves included a mixture of both sun and shade leaves (*13*). The increase in leaf $\delta^{15}N$ occurred in both legumes (Fabaceae) and nonlegumes (Fig. 1, A and B). Foliar N concentrations in nonlegumes increased by 7.7 ± 1.9% and 15.2 ± 2.5% when comparing 1960s leaves to 2007 sun and shade leaves, respectively (Fig. 1C). Legumes had substantially greater foliar N concentrations than nonlegumes, and there was no overall change in their foliar N concentration between the 1960s and 2007 (Fig. 1D).

To assess whether the changes detected on BCI are representative of tropical forests more broadly, we determined $\delta^{15}N$ in tree rings from three nonleguminous tree species in the Huai Kha Khaeng Reserve, a remote monsoon forest near the Thailand-Myanmar border. Significant increases in $\delta^{15}N$ during the past century were detected in all three species (Fig. 2). Similar changes were reported previously for tree rings in two Amazonian rainforest tree species (*14*).

A forest N addition experiment conducted 1 km from BCI provides perspective on the changes in foliar N composition (*15*). Foliar $\delta^{15}N$ increased by 0.3 to 1.5‰ in four tree species and by ~0.5 to 1.2‰ in fine litter (*15*), and the N concentration in litterfall increased by 7% (*16*) after 8 to 9 years of fertilization with 125 kg N $ha^{-1}year^{-1}$. The observed increase in leaf $\delta^{15}N$ did not reflect the signal of the N fertilizer, which had a lower $\delta^{15}N$ (–2.2‰) than leaves of nonfertilized trees in control plots (*15*) and therefore should have resulted in a decline rather than an increase in foliar $\delta^{15}N$. Nitrogen fertilization also increased NO_3 leaching (from 0.01 to 0.93 mg N $liter^{-1}$), NO flux (from 70 to 196 μg N m^{-2} day^{-1}), and N_2O flux (from 448 to 1498 μg N m^{-2} day^{-1}) (*15*), confirming that the increase in leaf $\delta^{15}N$ after N fertilization was associated with a more

[1]Institute of Botany, University of Natural Resources and Life Sciences, Gregor Mendel-Straße 33, 1180 Vienna, Austria. [2]Smithsonian Tropical Research Institute, Apartado 0843-03092, Balboa, Ancón, Republic of Panama. [3]Department of Chemical Ecology and Ecosystem Research, University of Vienna, Althanstrasse 14, A-1090 Vienna, Austria. [4]Institute of Environmental Physics, University of Bremen, Otto-Hahn-Allee 1, D-28359 Bremen, Germany. [5]Centre d'Étude de la Forêt, Département des Sciences Biologiques, Université du Québec à Montréal, Post Office Box 8888 Centre-ville Station, H3C 3P8 Montreal, Canada.

*To whom correspondence should be addressed. E-mail: peter.hietz@boku.ac.at

The Influence of Late Quaternary Climate-Change Velocity on Species Endemism

B. Sandel, L. Arge, B. Dalsgaard, R. G. Davies, K. J. Gaston, W. J. Sutherland and J.-C. Svenning (October 6, 2011)

Science **334** (6056), 660-664. [doi: 10.1126/science.1210173] originally published online October 6, 2011

Editor's Summary

Science (print ISSN 0036-8075; online ISSN 1095-9203) is published weekly, except the last week in December, by the American Association for the Advancement of Science, 1200 New York Avenue NW, Washington, DC 20005.

2008 49th Annual IEEE Symposium on Foundations of Computer Science

Succincter

Mihai Pătraşcu*
MIT

Abstract

We can represent an array of n values from $\{0, 1, 2\}$ using $\lceil n \log_2 3 \rceil$ bits (arithmetic coding), but then we cannot retrieve a single element efficiently. Instead, we can encode every block of t elements using $\lceil t \log_2 3 \rceil$ bits, and bound the retrieval time by t. This gives a linear trade-off between the redundancy of the representation and the query time.

In fact, this type of linear trade-off is ubiquitous in known succinct data structures, and in data compression. The folk wisdom is that if we want to waste one bit per block, the encoding is so constrained that it cannot help the query in any way. Thus, the only thing a query can do is to read the entire block and unpack it.

We break this limitation and show how to use recursion to improve redundancy. It turns out that if a block is encoded with two (!) bits of redundancy, we can decode a single element, and answer many other interesting queries, in time logarithmic in the block size.

Our technique allows us to revisit classic problems in succinct data structures, and give surprising new upper bounds. We also construct a locally-decodable version of arithmetic coding.

1 Introduction

1.1 Motivation

Can we represent data close to the information-theoretic minimum space, and still answer interesting queries efficiently? Two basic examples can showcase the antagonistic nature of compression and fast queries:

1. Suppose we want to store a bit-vector $A[1 \,..\, n]$, and answer partial sums queries: $\textsc{Rank}(k)$, which asks for $\sum_{i=1}^{k} A[i]$; and $\textsc{Select}(k)$, which asks for the index of the k-th one in the array.

One the one hand, we must store some summaries (e.g. partial sums at various points) to support fast queries. On the other hand, a summary is, almost by definition, redundant. If we store a summary for every block of t bits, it would appear that the query needs to spend time proportional to t, because no guiding information is available inside the block.

2. Suppose we want to represent an array $A[1 \,..\, n]$ of "trits" ($A[i] \in \{0, 1, 2\}$), supporting fast access to single elements $A[i]$. We can encode the entire array as a number in $\{0, \ldots, 3^n - 1\}$, but the information about every trit is "smeared" around, and we cannot decode one trit without decoding the whole array.

Generalizing trits to symbols drawn independently from a distribution with entropy H, we can use arithmetic coding to achieve $n \cdot H + 1$ bits on average, but information about each element is spread around in the encoding. At the other extreme, we can use Huffman coding to achieve $n \cdot \big(H + O(1)\big)$ bits[1] of storage, which represents every element in "its own" memory bits, using a prefix-free code.

The natural solutions to these problems gravitate towards a linear trade-off between redundancy and query time. We can store a summary for every block of t elements (in problem 1.), or we can "round up" the entropy of every block of t symbols to an integral number of bits (in problem 2.). In both cases, we are introducing redundancy of roughly $\frac{n}{t}$, and the query time will be proportional to t.

It is not hard to convince oneself that a linear trade-off is the best possible. If we store t elements with at most $O(1)$ bits of redundancy, we need a super-efficient encoding that is essentially fixed due the entropy constraint. Because the encoding is so constrained, it would appear that it cannot be useful beyond representing the data itself. Then, the only way to work with such a super-efficient encoding is to decode it entirely, forcing query time proportional to t.

In this paper, we show that this intuition is false: we can use recursion even inside a super-efficient encoding (if we are allowed *two* bits of redundancy). Instead of decoding t elements to get to one trit, local decoding can be supported in logarithmic time. This extends to storing an entire augmented tree succinctly, so we can solve RANK/SELECT in

*Supported by a Google Research Award and by MADALGO - Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

[1] More precisely, Gallager [11] showed that the redundancy is at most $p_{\max} + 0.086$ bits per element, where $p_{\max}$ is the maximum probability of an element.

0272-5428/08 $25.00 © 2008 IEEE
DOI 10.1109/FOCS.2008.83

logarithmic time. At every node of the tree, we can achieve something nontrivial (store the sum of its subtree), while introducing just $\frac{1}{t}$ bits of redundancy.

1.2 Succinct(er) Data Structures

In the field of succinct data structures, the goal is to construct data structures that use space equal to the information theoretic minimum plus some redundancy $\mathcal{R}$, while supporting various types of queries. The field has been expanding at a remarkable rate in the past decade, exploring a wide variety of problems and queries.

All of these structures, however, exhibit a linear trade-off between redundancy and query time. Typically the results are stated for constant query time, and achieve a fixed redundancy close to linear, most often around $O(\frac{n}{\lg n})$. At a high enough level of abstraction, this comes from storing $\varepsilon \lg n$ elements by an optimal encoding, and using a precomputed lookup table of size $O(n^{\varepsilon})$ to decode them in constant time. It can be seen that for any constant query time, the redundancy will not improve asymptotically (we can only look at a constant number of blocks).

For many natural problems in the field, our technique can achieve redundancy $O(n/\mathrm{poly}\log n)$ with constant running times, for any desired $\mathrm{poly}\log$. This suggests we have to rethink our expectations and approaches when designing succinct data structures.

Surprisingly, our constructions are often easier than the fixed-redundancy data structures they are replacing. It is not uncommon for succinct data structures to group elements into chunks, group chucks in superchunks, and finally group superchunks in megachunks. Each level has different lookup tables, and different details in the implementation. By having to do recursion for an unbounded number of levels, we are forced to discover a clean and uniform way to do it.

The following are a few results that follow from our technique. We believe however that the main value of this paper is to demonstrate that redundancy can be improved through recursion. These particular results are illustrations.

Locally decodable arithmetic codes. Our toy problem of storing ternary values can be generalized to representing an array of n elements with zeroth-order entropy H. Matching zeroth-order entropy is needed at the bottom of almost any technique attempting to match higher order entropy (including, for example, LZ77, the Burrows-Wheeler transform, JPEG, and MP3). Arithmetic coding can have a notable advantage over Huffman coding at low rates. For example, Shannon estimated the entropy of English to be 1.3 bits/letter, a rate at which a constant redundancy per letter can increase the encoding size by a significant percentage.

Mitzenmacher [17] uses arithmetic coding to compress Bloom filters, an application that critically requires local decodability. He asks for "a compression scheme that also provided random access," noting that "achieving random access, efficiency, and good compression simultaneously is generally difficult."

Our results give a version of locally-decodable arithmetic codes, in which the trade-off between local access time and redundancy is exponential:

Theorem 1. *Consider an array of n elements from an alphabet Σ, and let $f_\sigma > 0$ be the number of occurrences of letter σ in the array. On a RAM with cells of $\Omega(\lg n)$ bits, we can represent the array with:*

$$O(|\Sigma| \lg n) \;+\; \sum_{\sigma\in\Sigma} f_\sigma \log_2 \frac{n}{f_\sigma} \;+\; n/\Big(\frac{\lg n}{t}\Big)^t \;+\; \tilde{O}(n^{3/4})$$

bits of memory, supporting single-element access in $O(t)$ time.

Observe that $\sum f_\sigma \log_2 \frac{n}{f_\sigma}$ is the empirical entropy of the array. Thus, if the elements are generated by a memoryless process with entropy H, the expected space is $n \cdot H$, plus redundancy decreasing exponentially in t, plus $O(|\Sigma|)$ words needed to represent the distribution.

Bit-vectors with RANK/ SELECT. The problem of supporting RANK and SELECT on bit vectors is the bread-and-butter of succinct data structures, finding use in most other data structures (for representing trees, graphs, suffix trees / suffix arrays etc). Thus, the redundancy needed for this problem has come under quite a bit of attention.

The seminal papers of Jacobson [15] from FOCS'89, and Clark and Munro [5] from SODA'96 gave the first data structures using space $n + o(n)$ and constant query time. These results were later improved [19, 21, 26].

In several applications, the set of ones is not dense in the array. Thus, the problem was generalized to storing an array $A[1\,..\,u]$, containing n ones and $u - n$ zeros. The optimal space is $B = \lg \binom{u}{n}$. Note that the problem can solve predecessor search, by running $\textsc{Select}(\textsc{Rank}(i))$. From the lower bounds of [25], it follows that constant running times are only possible when the universe is not too much larger than n, in particular, $u = n \cdot \mathrm{poly}\log n$. Thus, succinct data structures have focused on this regime of parameters.

Pagh [23] achieved space $B + O(n \cdot \frac{(\lg\lg n)^2}{\lg n})$ for this sparse problem. Recently, Golynski et al. [13] achieved $B + O(n \cdot \frac{\lg\lg u}{\lg^2 n})$. Finally, Golynski et al. [14] have achieved space $B + O(n \cdot \frac{\lg\lg n \cdot \lg(u/n)}{\lg^2 n})$. That paper conjectures that their redundancy is the best possible in constant time.

Here, we disprove their conjecture, and show that any redundancy $O(n/\mathrm{poly}\log n)$ is achievable.

Theorem 2. *On a RAM with cells of $\Omega(\lg u)$ bits, we can represent an array $A[1\,..\,u]$ with n ones and $u-n$ zeros using $\log_2 \binom{u}{n} + \frac{u}{(\lg u\ /t)^t} + \tilde{O}(u^{3/4})$ bits of memory, supporting* RANK *and* SELECT *queries in $O(t)$ time.*

The RANK/SELECT problem has also seen a lot of work in lower bounds [10, 16, 14, 12], particularly bounds applying to "systematic encodings." In this model, the bit vector is stored separately in plain form, and the succinct data structure consists of a sublinear size index on the side. Under this requirement, the best achievable redundancy with query time t is $\frac{n}{t \cdot \text{poly} \lg n}$, i.e. the linear trade-off between redundancy and query time is the best possible. Our results demonstrate the significant power of not storing data in plain form.

Dictionaries. The dictionary problem is to store a set S of n elements from a universe u, and answer membership queries (is $x \in S$?) efficiently. Improving space has long been a primary goal in the study of dictionaries. For example, classic works put a lot of effort into analyzing linear probing when the table is close to capacity (if a $1-\varepsilon$ fraction of the cells are used, how does the "constant" running time degrade with ε?). Another trend is to study weaker versions of dictionaries, in the hope of saving space. Examples include the well known Bloom filters [1, 3, 22], and dictionaries that support retrieval only, sometimes called "Bloomier filters" [6, 4, 18].

The famous FKS dictionaries [8] were the first to solve the dictionary problem in linear space, in the sense of using $O(n)$ cells of size $\lg u$, while supporting queries in $O(1)$ time in the worst-case. Many data structures with similar performance have been suggested; in particular, cuckoo hashing [24] uses $(2+\varepsilon)n$ memory words.

Brodnik and Munro [2] were the first to approach the entropy bound, $B = \lg \binom{u}{n}$. Their data structure used space $B + O(B/\lg\lg\lg u)$, and they suggested that getting significantly more sublinear bounds might not be possible without "a more powerful machine model."

Pagh [23] gave the best known bound, achieving $B + O(n\frac{(\lg\lg n)^2}{\lg n})$. In fact, his algorithm is reduction to the RANK problem in vectors of size $u = n \cdot \text{poly} \log n$. Thus, our results immediately imply dictionaries with redundancy $B + O(n/\lg^c n)$, for any constant c.

Balanced parentheses. Our techniques imply results identical to RANK/SELECT for the problem of storing a string of $2n$ balanced parentheses, and answering two queries:

MATCH(k) : find the parenthesis that matches the one on position k.

ENCLOSING(k) : find the open parenthesis that encloses most tightly the one on position k.

The optimal space is given by the Catalan number: $\lg\left(\frac{1}{n+1}\binom{2n}{n}\right) = 2n - O(\lg n)$. Due to the natural association between tree structures and balanced parentheses, this problem has been the crucial building block in most succinct tree representations.

A generic transformation. In fact, all of our results are shown via a generic transformation, converting a broad class of data structures based on augmented search trees into succinct data structures. It seems likely that such a broad result will have applications beyond the ones explored in this paper.

The reader is referred to §4 for formal details about the class of augmented search trees handled by our transformation.

1.3 Technical Discussion

Our goal is to improve redundancy through recursion, as opposed to a linear scan. A naïve view for how recursion might work for the trits problem is the following. We take w trits, which have entropy $w\log_2 3$, and "extract" some M bits of information (e.g. $M = \lfloor w\log_2 3 \rfloor$), which we store. Then, the remaining $\delta = w\log_2 3 - M$ bits of information are passed to the second level of the recursion. At the second level, we aggregate w blocks, for which we must store $w\delta$ bits of information. We store some M' bits (e.g. $M' = \lfloor w\delta \rfloor$), and pass $\delta' = w\delta - M'$ bits to the second level, etc.

Since we are not willing to waste a bit of redundancy per block, δ may not be an integer. Unfortunately, "passing some fractional bits of information" is an intuition that does not render itself to any obvious implementation.

Our solution for passing a fractional number of entropy bits is elegant and, in hindsight, quite natural. We will approximate δ by $\log_2 K$, where K is an integer. This means that passing δ bits of information is almost the same as passing a number in $\{0, \ldots, K-1\}$. This approach introduces redundancy ("unused entropy") of $\log_2 K - \delta$ bits, a quantity that depends on how close δ is to a logarithm of an integer. Note however, that if K is the best approximation, δ is sandwiched between $\log_2(K-1)$ and $\log_2 K$. Thus, if we choose δ large enough, there is always some K that gives a good approximation.

At the second level of recursion, the problem will be to represent an array of n/w values from $\{0, \ldots, K-1\}$. This is just a generalization of the ternary problem to an arbitrary universe, so the same ideas apply recursively.

It should be noted that the basic technique of storing a certain number of bits and passing the "spill" to be stored separately, is not new. It was originally introduced by Munro et al. [20], and is the basis of the logarithmic improvement in redundancy of Golynski et al. [13] (see the

"informative encodings" of that paper). Unfortunately, the techniques in these papers do no allow recursive composition, which is the key to our results.

In §2, we formally define the concept of *spill-over representations*, and uses it to prove Theorem 4 for representing an array of trits.

For less trivial applications, we need to compose variable-length representations without losing entropy. For example, we may want to store the sum of n numbers, and then the numbers themselves, by an efficient encoding that doesn't store the sum redundantly. With the right view, we can give a very usable lemma performing this composition; see §3. Finally, §4 uses this lemma to derive our main results in succinct data structures.

2 Spill-Over Representations

Assume we want to represent a value from a set $\mathcal{X}$. Any representation in a sequence of memory bits will use at least $\lceil \log_2 |\mathcal{X}| \rceil$ bits in the worst case, so it will have a redundancy of almost one bit if $|\mathcal{X}|$ is not close to a power of two. To achieve a redundancy much smaller than one for *any* value of $|\mathcal{X}|$, we must first define a model of "representation" where this is possible.

A *spill-over* representation consists of M memory bits (stored in our random-access memory), plus a number in $\{0, \ldots, K-1\}$ that is stored by some outside entity. This number is called the *spill*, and K is called the *spill universe*. Observe that the spill-over representation is capable of representing $K \cdot 2^M$ distinct values. When using such a representation to store an element in $\mathcal{X}$, with $|\mathcal{X}| < K \cdot 2^M$, we define the redundancy of the representation to be $\log_2(K \cdot 2^M) - \log_2 |\mathcal{X}| = M + \log_2 K - \log_2 |\mathcal{X}|$. We can think of this as entropy wasted by the representation.

When talking about algorithms that access a spill-over representation, we assume the spill is provided by the outside entity for free, and the algorithm may access the memory bits by random access in the word memory.

As the most fundamental example of a spill-over representation, we have the following:

Lemma 3. *Consider an arbitrary set $\mathcal{X}$, and fix $r \le |\mathcal{X}|$. We can represent an element of $\mathcal{X}$ by a spill-over encoding with a spill universe K satisfying $r \le K \le 2r$, and redundancy at most $\frac{2}{r}$ bits.*

Proof. We must choose M and K carefully to achieve our redundancy bound. Specifically, we choose M to satisfy $2^M \cdot r \le |\mathcal{X}| \le 2^{M+1} \cdot r$. This fixes the spill universe to be $K = \lceil \frac{|\mathcal{X}|}{2^M} \rceil$; observe that $r < K \le 2r$.

Any injective map of $\mathcal{X}$ into $\{0,1\}^M \times \{0, \ldots, K-1\}$ defines a spill-over scheme. For example, we can divide an index in $\mathcal{X}$ by 2^M, store the remainder as M memory bits, and pass the quotient as the spill. This is decodable by $O(1)$ arithmetic operations.

Our spill-over scheme is capable of representing $2^M \cdot K$ different values, as opposed to the $|\mathcal{X}|$ values required. Thus, we have a redundancy of:

$$\begin{aligned} \log_2 \left(\tfrac{K \cdot 2^M}{|\mathcal{X}|} \right) &= \log_2 \left(\tfrac{\lceil \frac{|\mathcal{X}|}{2^M} \rceil \cdot 2^M}{|\mathcal{X}|} \right) \\ &\le \log_2 \left(\tfrac{(\frac{|\mathcal{X}|}{2^M} + 1) \cdot 2^M}{|\mathcal{X}|} \right) \\ &= \log_2 \left(1 + \tfrac{2^M}{|\mathcal{X}|} \right) \le \log_2 \left(1 + \tfrac{1}{r} \right) \le \tfrac{2}{r} \quad \square \end{aligned}$$

2.1 Application: Storing Trits

We now show how to compose spill-over representations from Lemma 3 recursively, yielding the following bounds for our toy problem of storing n ternary values:

Theorem 4. *On a RAM with w-bit cells, we can represent an array $A[1..n]$ with $A[i] \in \{0,1,2\}$ using $\lceil n \log_2 3 \rceil + \frac{n}{(w/t)^t} + \operatorname{poly} \log n$ bits of memory, while supporting single-element accesses in $O(t)$ time.*

Proof. We first group elements into blocks of w. A block takes values in a set of size 3^w. We apply Lemma 3 with some parameter r to be determined, and store each block as M_0 memory bits and a spill in a universe $K_0 \in [r, 2r]$. Given the spill, we can read the $O(w)$ memory bits and decode in constant time:

- first assemble the spill and the memory bits, multiplying the spill by 2^{M_0}, and adding them together.
- now extract the ith trit, dividing by 3^i, and taking the number modulo 3.

In practice, we want to avoid division by precomputing a table with the multiplicative inverses of 3^i. Note that we do not need explicit pointers to each block, since the offset at which the memory bits of a block are stored is equal to M_0 times the block index.

Now let $B = \Theta(\frac{w}{\lg r})$, with $B \ge 2$. At the second level of recursion, we store the spills of B consecutive blocks. There are $K_0^B \le (2r)^B = 2^{O(w)}$ choices for the bottom-level spills. Using Lemma 3, we can represent this data as $M_1 \le \log_2(K_0^B) = O(w)$ memory bits, and a spill in universe $K_1 \in [r, 2r]$. Note that the assumption $r \le |\mathcal{X}|$ made by the lemma is indeed satisfied, because $|\mathcal{X}| = K_0^B \ge r^2$. Since Lemma 3 is applied in identical conditions, K_1 and M_1 will be identical for all level-1 blocks.

We can continue to apply this scheme recursively, storing B spills in universe K_1, generating a spill in universe $K_2 \in [r, 2r]$, etc. We do this for t levels, supporting access queries in $O(t)$ word probes. At the end of the recursion, we store each of the final spills with $\lceil \log_2 K_t \rceil$ bits, paying one bit of redundancy per spill. Note that the size of the final scheme

is a telescoping sum of the sizes at intermediate levels of recursion, i.e. the final redundancy is simply the sum of the redundancies introduced at each step. This gives:

$$\begin{aligned}\mathcal{R} &= O\left(\frac{n/w}{r} + \frac{n/(Bw)}{r} + \cdots + \frac{n/(B^t w)}{r} + \frac{n}{B^t w}\right) \\ &= O\left(\frac{n}{wr} + \frac{n}{w \cdot B^t}\right)\end{aligned}$$

To balance the two terms (the redundancy of the spill-over representations, versus the final redundancy of one bits per spill), let $r = B^t$. By our choice of B, we have $B = O(\frac{w}{\lg r}) = O(\frac{w}{t \lg B}) = O(\frac{w/t}{\lg(w/t)})$. We thus have $r = B^t = \left(\frac{w}{t}\right)^{\Omega(t)}$. Adjusting constants, we have a redundancy of $\mathcal{R} \leq \frac{n}{(w/t)^t}$, and query time $O(t)$.

As a header of the data structure, we need to store the values K_i and M_i for every level of the recursion, which are needed to navigate the data structure. This adds $O(\lg^2 n)$ bits to the redundancy. □

3 Composing Variable-Length Encodings

As we have just seen, spill-over encodings of fixed length can be composed easily in a recursive fashion. However, in less trivial applications, we need to compose variable-length encodings without losing entropy. For an illustration of the concept, assume we want to store an array $A[1\,..\,n]$ of bits, such that we can query both any element $A[i]$, and the sum of the entire array: $X = \sum_{j=1}^{n} A[j]$. If we choose the trivial encoding as n bits, querying the sum will take linear time.

Conceptually, we must first store the sum, followed by the array itself, represented by an *efficient* encoding that uses knowledge of the sum (i.e. does not contain any redundant information about the sum). This should not lose entropy, since $H(X) + H(A|X) = n$.

The trouble is that the bound $H(X)$ can only be achieved in expectation, by some kind of arithmetic code. Furthermore, since $H(A|X)$ is a function of X, the parameters of a spill-over representation must also vary with X. Thus, we need to piece together some kind of arithmetic code for X, with a variable-length spill-over representation of A whose size depends on X. In the end, however, we should obtain a fixed-length encoding, since the overall entropy is n bits.

The following lemma formalizes this intuition, in a statement that has been crafted carefully for ease of use in applications:

Lemma 5. *Assume we have to represent a variable $x \in \mathcal{X}$, and a pair $(y_M, y_K) \in \{0,1\}^{M(x)} \times \{0, \ldots, K(x) - 1\}$. Let $p(x)$ be a probability density function on $\mathcal{X}$, and $K(\cdot), M(\cdot)$ be non-negative functions on $\mathcal{X}$ satisfying:*

$$(\forall) x \in \mathcal{X}: \quad \log_2 \frac{1}{p(x)} + M(x) + \log_2 K(x) \leq H \quad (1)$$

We can design a spill-over representation for x, y_M and y_K with the following parameters:

- *the spill universe is $K_\star$ with $K_\star \leq 2r$, and the memory usage is $M_\star$ bits.*
- *the redundancy is at most $\frac{4}{r}$ bits, i.e. $M_\star + \log_2 K_\star \leq H + \frac{4}{r}$.*
- *if the word size is $w = \Omega\big(\lg |\mathcal{X}| + \lg r + \lg \max K(x)\big)$, x and y_K can be decoded with $O(1)$ word probes. The input bits y_M can be read directly from memory, but only after y_K is retrieved.*
- *given a precomputed table of $O(|\mathcal{X}|)$ words that only depends on the input functions K, M and p, decoding x and y_K takes constant time on the word RAM.*

In §3.1, we describe the main details of the construction. To make the construction implementable, we need to tweak the distribution $p(\cdot)$ to avoid pathologically rare events; this is detailed in §3.2. Finally, in §3.3 we describe the constant-time decoding procedure, which relies on a cute algorithmic trick. Note that a table of size $O(|\mathcal{X}|)$ is optimal, since the lemma is instantiated for three arbitrary functions on $\mathcal{X}$.

3.1 The Basic Construction

The design of our data structure is remarkably simple. First, let $M_{\min} = \min_{x \in \mathcal{X}} M(x)$. We can decrease each $M(x)$ to $M_{\min}$, by encoding $M(x) - M_{\min}$ bits of memory into the spill. This has the technical effect that we may not access y_M prior to decoding y_K, as stated in the lemma. From now on, we assume y_M always has $M_{\min}$ bits, and y_K comes from a universe of $K'(x) = K(x) \cdot 2^{M(x) - M_{\min}}$.

Now let Z be the set of possible pairs (x, y_K).

Claim 6. *We have $\log_2 |Z| + M_{\min} \leq H$.*

Proof. We will show $|Z| \leq \max_{x \in \mathcal{X}} \frac{K'(x)}{p(x)}$. Since $\log_2 \frac{1}{p(x)} + \log_2 K'(x) + M_{\min} \leq H$ for all x, it follows that $\log_2 |Z| + M_{\min} \leq H$.

Suppose for contradiction that $|Z| > \frac{K'(x)}{p(x)}$ for all $x \in \mathcal{X}$. Then, $K'(x) < |Z| \cdot p(x)$, for all x. We have $|Z| = \sum_{x \in \mathcal{X}} K'(x) < \sum_{x \in \mathcal{X}} \big(|Z| \cdot p(x)\big) = |Z|$, which gives a contradiction. □

Though the proof of this lemma looks like a technicality, the intuition behind it is quite strong. The space of encodings Z is partitioned into equivalence classes $(x, \star)$, giving each element x a fraction equal to $K(x)/|Z|$. But $\log_2 \frac{1}{p(x)} + \log_2 K(x) \approx H - M_{\min}$, so $K(x)/p(x)$ is roughly *fixed*. Thus, the fraction of space assigned to x is roughly $p(x)$, which is exactly how arithmetic coding operates (partitioning the real interval $[0, 1]$).

To complete the representation, we apply Lemma 3 to the set Z. The resulting spill is passed along as the spill of

our data structure, and the memory bits are stored at the beginning of the data structure, followed by the $M_{\min}$ bits of y_M. The only redundancy introduced is by this application of Lemma 3, i.e. $\frac{2}{r}$ bits.

For this construction to be efficiently decodable, we must at the very least be able to manipulate a value of Z in constant time, i.e. have a word size of $w = \Omega(\lg |Z|)$. To understand this requirement, we bound $\log_2 |Z| \leq H - \min M(x)$. For every x, we are free to increase $M(x)$, padding with zero bits, up to the maximum integer satisfying $M(x) \leq H - \log_2 K(x) - \log_2 \frac{1}{p(x)}$. Thus, $M(x) > H - \log_2 \frac{K(x)}{p(x)} - 1$. This implies the bound $\log_2 |Z| \leq \log_2 \frac{\max K(x)}{\min p(x)} + 1$, i.e. $|Z| = O(\frac{\max K(x)}{\min p(x)})$. The statement of the lemma already assumes $w = \Omega(\lg \max K(x))$, since $K(x)$ is the universe of the input spill. Thus, it remains to ensure that $w = \Omega(\lg \frac{1}{\min p(x)})$.

3.2 Handling Rare Events

Unfortunately, some values $x \in \mathcal{X}$ can be arbitrarily rare, and in fact, $\min p(x)$ is prohibitively small even for natural applications. Remember our example of representing an array $A[1\,..\,n]$ of bits, together with its sum $x = \sum_{j=1}^{n} A[i]$. We have $\min p(x) = p(n) = 2^{-n}$, which means our algorithm wants to manipulate spills of $\Omega(n)$ bits. A word size of $\Theta(n)$ bits is an unrealistic assumption.

Our strategy will be to tweak the distribution $p(\cdot)$ slightly, increasing $\min p(x)$ towards $\frac{1}{|\mathcal{X}|}$, while not losing too much entropy if we code according to this false distribution. (In information-theoretic terms, the Kullback-Leibler divergence between the distributions must be small.)

Formally, we tweak the distribution by adding $\frac{1}{|X| \cdot r}$ to every probability, and then normalizing:

$$p'(x) = \Big(p(x) + \frac{1}{|X| \cdot r}\Big) \Big/ \Big(1 + \frac{1}{r}\Big)$$

Observe that $\sum_{x \in X} p'(x) = \big(\sum_{x \in X} p(x) + |X| \cdot \frac{1}{|X| \cdot r}\big) \big/ \big(1 + \frac{1}{r}\big) = 1$, so $p'(\cdot)$ indeed defines a distribution.

We now have $\min p'(x) \geq \frac{1}{|\mathcal{X}| \cdot r} \big/ (1 + \frac{1}{r}) \geq \frac{1}{2r \cdot |\mathcal{X}|}$. This requires the word size to be $w = \Omega(\lg r + \lg |\mathcal{X}|)$, a reasonable assumption made by the lemma. (In our example with an n-bit array, $|\mathcal{X}| = n+1$, so we require $w = \Omega(\lg r + \lg n)$, instead of $\Omega(n)$ are before.)

Finally, we must show that the entropy bound in (1) is not hurt too much if we code according to $p'(\cdot)$ instead of $p(\cdot)$. Note that:

$$\begin{aligned} \log_2 \tfrac{1}{p'(x)} &\leq \log_2 \tfrac{1}{p(x)/(1+1/r)} \\ &= \log_2 \tfrac{1}{p(x)} + \log_2 \big(1 + \tfrac{1}{r}\big) \leq \log_2 \tfrac{1}{p(x)} + \tfrac{2}{r} \end{aligned}$$

We can replace (1) with the following weaker guarantee:

$$\begin{aligned} &M(x) + \log_2 K(x) + \log_2 \tfrac{1}{p'(x)} \\ &\quad \leq M(x) + \log_2 K(x) + \log_2 \tfrac{1}{p(x)} + \tfrac{2}{r} \leq H + \tfrac{2}{r} \end{aligned}$$

Combining with the construction from §3.1, which introduced another $\frac{2}{r}$ bits of redundancy, we have lost $\frac{4}{r}$ bits of redundancy overall.

3.3 Algorithmic Decoding

The heart of the decoding problem lies in recovering x and y_K based on an index in Z (output by Lemma 3). We could do this with a lookup in a precomputed table of size $|Z|$. Remember that we bounded $|Z| = O(\frac{\max K(x)}{\min p(x)}) = O(|\mathcal{X}| \cdot r \cdot \max K(x))$, which is not a strong enough bound for some applications. We now show how to use a table of size just $O(|\mathcal{X}|)$.

From the point of view of x, the space Z is partitioned into pieces of cardinality $K(x)$, and the query is to find the piece containing a given codeword. We are free to design the partition to make decoding efficient. First, we assign to each x a contiguous interval of Z. Let z_x be the left boundary of the interval assigned to x. Decoding x is equivalent to a predecessor search, locating the codeword among the values $\{z_1, \ldots, z_{|\mathcal{X}|}\}$. Decoding y_K simply subtracts z_x from the codeword.

Unfortunately, the optimal bounds for predecessor search [25] are superconstant in the worst case. To achieve constant time, we must leverage our ability to choose the encoding: we must arrange the intervals in an order that makes predecessor search easy! While this sounds mysterious, it turns out that sorting the intervals by increasing length suffices.

Claim 7. *If intervals are sorted by length, i.e. $z_{i+1} - z_i \geq z_i - z_{i-1}$, predecessor search among the z_i's can be supported in constant time by a data structure of $O(|\mathcal{X}|)$ words.*

Proof. Let $f(\tau)$ be the smallest interval of length τ, i.e. $f(\tau) = \min\{i \mid z_{i+1} - z_i \geq \tau\}$. Consider the set of $z_{f(\tau)}$, for every τ a power of two. This set has $O(w)$ values, so we can store it in a fusion tree [9], and support predecessor search in constant time.

Say we have located the query between $z_{f(\tau)}$ and $z_{f(2\tau)}$. All intervals in this range have width between τ and 2τ, i.e. we have constant spread. In this case, predecessor search can be solved easily in constant time [7]: break the universe into buckets of size τ, which ensures at most one value per bucket, and at most three buckets to inspect until the predecessor is found. □

4 Applications to Succinct Data Structures

4.1 Augmented Trees

As mentioned already, our results are based on a generic transformation of augmented B-trees to succinct data structures. For some $B \geq 2$, we define a class of data structures, *aB-trees*, as follows:

- The data structure represents an array $A[1\,..\,n]$ of elements from some alphabet Σ, where n is a power of B. The data structure is a B-ary tree with the elements of A in the leaves.
- Every node is augmented with a value from some alphabet Φ. The value of a leaf is a function of its array element, and the value of an internal node is a function $\mathcal{A}$ of the values of its B children, and the size of the subtree.
- The query algorithm examines the values of the root's children, decides which child to recurse to, examines all values of that node's children, recurses to one of them, etc. When a leaf is examined, the algorithm outputs the query answer. We assume the query algorithm spends constant time per node, if all values of the children are given packed in a word.

For instance, the RANK/SELECT problem has a standard solution through an aB-tree. The alphabet Σ is simply $\{0,1\}$. Every internal node counts the sum of the leaves in its subtree (equivalently, the sum of its children), so $\Phi = \{0,\ldots,n\}$. The queries can be solved in constant time per node if the values of all children are given packed in a word. This uses very standard ideas from word-level parallelism that we omit.

We aim to compress an aB-tree. A natural goal for the space an aB-tree should use is given by $\mathcal{N}(n,\varphi)$, defined as the number of instances of $A[1\,..\,n]$ such that the root is labeled with $\varphi \in \Phi$. Observe that we can write the following recursion for $\mathcal{N}(B\cdot n,\varphi)$:

$$\mathcal{N}(B\cdot n,\varphi) = \sum_{\varphi_1,\ldots,\varphi_B:\mathcal{A}(\varphi_1,\ldots,\varphi_B,n)=\varphi} \mathcal{N}(n,\varphi_1)\cdots\mathcal{N}(n,\varphi_B)$$

Indeed, any instance for the first half is valid, as long as its aggregate value combines properly with the aggregate of the second half.

To develop intuition for $\mathcal{N}$, observe that in the RANK/SELECT example, $\mathcal{N}(n,\varphi) = \binom{n}{\varphi}$, because φ was just the number of ones in the array. Our recursion becomes the following obvious identity:

$$\mathcal{N}(B\cdot n,\varphi) = \sum_{\varphi_1+\cdots+\varphi_B=\varphi} \mathcal{N}(n,\varphi_1)\cdots\mathcal{N}(n,\varphi_B)$$

We will show the following general result:

Theorem 8. *Let $B = O(\frac{w}{\lg(n+|\Phi|)})$. We can store an aB-tree of size n with root value φ using $\log_2 \mathcal{N}(n,\varphi)+2$ bits. The query time is $O(\log_B n)$, assuming precomputed look-up tables of $O(|\Sigma|+|\Phi|^{B+1}+B\cdot|\Phi|^B)$ words, which only depend on n, B and the aB-tree algorithm.*

Essentially, this result compresses the entire aB-tree with only two 2 bits of redundancy. The additional space of the look-up tables will not matter too much, since we construct many data structures that share them.

Application to RANK/SELECT. Say we want to solve RANK and SELECT in time $O(t)$, for an array of size U with N ones. As mentioned already, RANK and SELECT queries can be supported by an aB-tree, so Theorem 8 applies. If the aB-tree has size r, we have $|\Sigma| = 2$ and $|\Phi| = r+1$.

Choose $B \geq 2$ such that $B\lg B = \frac{\varepsilon \lg U}{t}$, and let $r = B^t = \left(\frac{\lg U}{t}\right)^{\Theta(t)}$. We break the array into buckets of size r, rounding up the size of the last bucket. Each bucket is stored as a succinct aB-tree. Supporting RANK and SELECT inside such an aB-tree requires time $O(\log_B r) = O(t)$.

For each bucket, we store the the index in memory of the bucket's memory bits. Let $N_1, N_2, \ldots$ be number of ones in each subarray. We store a partial sums vector for these values (to aid RANK), and a predecessor structure on the partial sums (to aid SELECT). We have at most U/r values from a universe of U, so the predecessor structure can support query time $O(t)$ using space $\frac{U}{r}\cdot r^{\Omega(1/t)} \leq \frac{U}{r}\cdot B = U/B^{t-1}$ words; see [25]. A query begins by examining these auxiliary structures, and then performing a query in the right aB-tree.

The components of the memory consumption are:

1. a pointer to each bucket, and the partial sums for the array $N_1, N_2, \ldots$. These occupy $O(\frac{U}{r}\lg U) = O\big(\frac{U\lg U}{B^t}\big)$ bits.
2. the predecessor structure, occupying $O(U/B^{t-1})$ words. This dominates item 1., and is $U/B^{\Theta(t)}$ bits.
3. the succinct aB-trees, which occupy:

$$\begin{aligned}\textstyle\sum_i \left[\log_2\binom{r}{N_i}+2\right] &\leq \log_2\textstyle\prod_i\binom{r}{N_i} + O\big(\frac{U}{r}\big)\\ &\leq \log_2\tbinom{U+r-1}{N} + O\big(\tfrac{U}{r}\big) \leq \log_2\tbinom{U}{N} + O\big(r+\tfrac{U}{r}\big)\end{aligned}$$

   bits. The redundancy can be rewritten as $r+\frac{U}{r} = O(\max\{\frac{U}{r},\sqrt{U}\})$.
4. the look-up tables, of size $O((r+1)^{B+1}+(r+1)^B\cdot B) = 2^{O(tB\lg B)} = 2^{O(\varepsilon\lg U)} = U^{O(\varepsilon)}$. Setting ε a small enough constant, this contributes negligibly to the redundancy. The only limitation is $B \geq 2$, so we cannot reduce the lookup tables below $O(r^3)$ words. A redundancy of $\frac{U}{r}+O(r^3)$ can be written as $\max\{\frac{U}{r}, U^{3/4}\}$.

To summarize, we obtain a redundancy of $U/B^{\Theta(t)} + O(U^{3/4})$. Readjusting constants in t, the redundancy is $U/\left(\frac{\lg U}{t}\right)^t + O(U^{3/4})$.

Application to arithmetic coding. In this application, Σ is the alphabet that we wish to encode. Intuitively, a letter $\sigma \in \Sigma$ has $\log_2 \frac{n}{f_\sigma}$ bits of entropy. We round these values up to multiples of $\frac{1}{r}$, which only adds redundancy $\frac{n}{r}$ over all symbols.

We construct an aB-tree, in which internal nodes are augmented to store the entropy of the symbols in their subtree. If the aB-tree has size at most r, the total entropy is at most $O(r \lg n)$, so $|\Phi| = O(\lg r + \lg\lg n)$. The query algorithm is trivial: it just traverses the tree down to the leaf that it wants to retrieve, ignoring all nodes along the way.

By this definition, $\mathcal{N}(n, \varphi)$ is the number of n-letter sequences with total entropy exactly φ. But there are at most 2^φ such sequences, by an immediate packing argument. Thus, an aB-tree of size r having value φ at the root can be compressed to space $\varphi + 2$. We now proceed as in the previous example, breaking the array into n/r buckets of size r, and performing the same calculations for the space occupied by auxiliary structures.

4.2 Proof of Theorem 8

The proof is by induction on powers of B, aggregating B spill-over representations for aB-trees of size n/B into one for an aB-tree of size n. Let $K(n, \varphi)$ be the spill universe used for a data structure of size n and root label φ. Let $M(n, \varphi)$ be the memory bits used by such a representation.

Let r to be determined. We guarantee inductively that:

$$K(n, \varphi) \le 2r; \tag{2}$$

$$M(n, \varphi) + \log_2 K(n, \varphi) \le \log_2 \mathcal{N}(n, \varphi) + 4\frac{2n-1}{r}. \tag{3}$$

Consider the base case, $n = 1$. The alphabet Σ is partitioned into sets Σ_φ of array elements for which the leaf is labeled with φ. We have $\mathcal{N}(1, \varphi) = |\Sigma_\varphi|$. We use a spill-over encoding as in Lemma 3 to store an index into Σ_φ. The encoding will use a spill universe $K(1, \varphi) \le 2r$ and $M(1, \varphi)$ bits of memory, such that $M(1, \varphi) + \log_2 K(1, \varphi) \le \log_2 |\Sigma_\varphi| + \frac{2}{r}$. We store a look-up table that determines the array value based on the value φ and the index into Σ_φ. These look-up tables (for all φ) require space $O(|\Phi| + |\Sigma|)$.

For the induction step, we break the array into B subarrays of size n/B. Let $\widetilde{\varphi} = (\varphi_1, \ldots, \varphi_B)$ denote the values at the root of each of the B subtrees. We recursively represent each subarray using $M(\frac{n}{B}, \varphi_i)$ bits of memory and spill universe $K(\frac{n}{B}, \varphi_i)$.

Then, all memory bits from the children are concatenated into a bit vector of size $M' = \sum_i M(\frac{n}{B}, \varphi_i)$, and the spills are combined into a superspill from the universe $K' = \prod_i K(\frac{n}{B}, \varphi_i)$. Since $\lg K' \le \lg\left((2r)^B\right) = O(B \lg r)$, we require that $B = O(w/\lg r)$, so that this superspill fits in a constant number of words. For every possible $\widetilde{\varphi}$, we precompute the partial sums of $M(\frac{n}{B}, \varphi_i)$, so that we know where the ith child begins in constant time. We also precompute the constant needed to extract the ith spill from the superspill. These tables require $O(B \cdot |\Phi|^B)$ words.

Summing the recursive guarantee (3) of every child, we have:

$$\log_2 K' + M' \;\le\; \log_2 \prod_i \mathcal{N}\Big(\frac{n}{B}, \varphi_i\Big) \;+\; 4 \cdot \frac{2n - B}{r}$$

Let $\varphi = \mathcal{A}(\varphi_1, \ldots, \varphi_i, n)$ be the value at the root. Let $p(\cdot)$ be the distribution of $\widetilde{\varphi}$ given this value of φ, that is:

$$p(\widetilde{\varphi}) \;=\; \prod_i \mathcal{N}\Big(\frac{n}{B}, \varphi_i\Big) \Big/ \mathcal{N}(n, \varphi)$$

But then:

$$\log_2 K' + M' \;\le\; \log_2 \mathcal{N}(n, \varphi) - \log_2 \frac{1}{p(\widetilde{\varphi})} + 4 \cdot \frac{2n - B}{r}$$

This satisfies the entropy condition (1) of Lemma 5. We apply the lemma to represent $\widetilde{\varphi}$, the superspill, and the memory bits of the subarrays. We obtain a representation with spill universe $K_\star \le 2r$ and $M_\star$ memory bits, such that:

$$\begin{aligned} M_\star + \log_2 K_\star \;&\le\; \log_2 \mathcal{N}(n, \varphi) \;+\; 4\frac{2n - B}{r} \;+\; \frac{4}{r} \\ &\le\; \log_2 \mathcal{N}(n, \varphi) + 4 \cdot \frac{2n - 1}{r} \end{aligned}$$

The precomputed table required by Lemma 5 is linear in the support of $p(\cdot)$, which is $|\Phi|^B$. Such a table is stored for every distinct φ, giving space $|\Phi|^{B+1}$. We must have $B = O(\frac{w}{\lg |\Phi|})$.

This completes the induction step. To prove Theorem 8, we construct the above representation for the required size n, using the value $r = \frac{1}{8n}$. Note that this requires $B \le O\big(\frac{w}{\lg \max\{|\Phi|, r}\}\big)$.

At the root, the final spill is stored explicitly at the beginning of the data structure. Thus, the space is:

$$\lceil \log_2 K(n, \varphi) \rceil + M(n, \varphi) \;\le\; \log_2 \mathcal{N}(n, \varphi) + 2$$

The queries are easy to support. First, we read the final spill at the root. Then, we decode $\widetilde{\varphi}$ and the superspill from the representation of Lemma 5. The aB-tree query algorithm decides which child to follow recursively based on $\widetilde{\varphi}$. We extract the spill of that child from the superspill, and recurse. The constants needed to extract the spill and the position in memory of the child were stored in look-up tables.

5 Open Problems

It is an important open problem to establish whether our exponential trade-off between redundancy and query time is optimal. We conjecture that it is. Unfortunately, proving this seems beyond the scope of current techniques. The only lower bound for succinct data structures (without the systematic assumption) is via a rather simple idea of Gál and Miltersen [10], which requires that the data structure have an intrinsic error-correcting property. Such a property is not characteristic of our problems.

Even if the exponential trade-off cannot be improved, it would be interesting to establish where this trade-off "bottoms." Due to our need for large precomputed tables, the smallest redundancy that we can achieve is some $O(n^\alpha)$ bits, where α is a constant close to one (for instance, $\alpha = 3/4$ for RANK/SELECT). Can this redundancy be reduced to, say, $O(\sqrt{n})$, or hopefully even $O(n^\varepsilon)$?

Acknowledgments. This work was initiated while the author was visiting the Max Planck Institut für Informatik, Saarbrücken, in June 2005. I wish to thank Seth Pettie for telling me about the problem on that occasion, and for initial discussions. I also wish to thank Rasmus Pagh and Rajeev Raman for helping me understand previous work.

References

[1] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[2] Andrej Brodnik and J. Ian Munro. Membership in constant time and almost-minimum space. *SIAM Journal on Computing*, 28(5):1627–1640, 1999. See also ESA'94.

[3] Larry Carter, Robert Floyd, John Gill, George Markowsky, and Mark Wegman. Exact and approximate membership testers. In *Proc. 10th ACM Symposium on Theory of Computing (STOC)*, pages 59–65, 1978.

[4] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. The Bloomier filter: an efficient data structure for static support lookup tables. In *Proc. 15th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 30–39, 2004.

[5] David R. Clark and J. Ian Munro. Efficient suffix trees on secondary storage. In *Proc. 7th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 383–391, 1996.

[6] Erik D. Demaine, Friedhelm Meyer auf der Heide, Rasmus Pagh, and Mihai Pătraşcu. De dictionariis dynamicis pauco spatio utentibus (lat. on dynamic dictionaries using little space). In *Proc. Latin American Theoretical Informatics (LATIN)*, pages 349–361, 2006.

[7] Erik D. Demaine, Thouis Jones, and Mihai Pătraşcu. Interpolation search for non-independent data. In *Proc. 15th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–523, 2004.

[8] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *Journal of the ACM*, 31(3):538–544, 1984. See also FOCS'82.

[9] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993. See also STOC'90.

[10] Anna Gál and Peter Bro Miltersen. The cell probe complexity of succinct data structures. In *Proc. 30th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 332–344, 2003.

[11] Robert G. Gallager. Variations on a theme by huffman. *IEEE Transactions on Information Theory*, 24(6):668–674, 1978.

[12] Alexander Golynski. Optimal lower bounds for rank and select indexes. *Theoretical Computer Science*, 387(3):348–359, 2007. See also ICALP'06.

[13] Alexander Golynski, Roberto Grossi, Ankur Gupta, Rajeev Raman, and S. Srinivasa Rao. On the size of succinct indices. In *Proc. 15th European Symposium on Algorithms (ESA)*, pages 371–382, 2007.

[14] Alexander Golynski, Rajeev Raman, and S. Srinivasa Rao. On the redundancy of succinct data structures. In *Proc. 11th Scandinavian Workshop on Algorithm Theory (SWAT)*, 2008.

[15] Guy Jacobson. Space-efficient static trees and graphs. In *Proc. 30th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 549–554, 1989.

[16] Peter Bro Miltersen. Lower bounds on the size of selection and rank indexes. In *Proc. 16th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 11–12, 2005.

[17] Michael Mitzenmacher. Compressed bloom filters. *IEEE/ACM Transactions on Networking*, 10(5):604–612, 2002. See also PODC'01.

[18] Christian Worm Mortensen, Rasmus Pagh, and Mihai Pătraşcu. On dynamic range reporting in one dimension. In *Proc. 37th ACM Symposium on Theory of Computing (STOC)*, pages 104–111, 2005.

[19] J. Ian Munro. Tables. In *Proc. 16th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 37–40, 1996.

[20] J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct representations of permutations. In *Proc. 30th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 345–356, 2003.

[21] J. Ian Munro, Venkatesh Raman, and S. Srinivasa Rao. Space efficient suffix trees. *Journal of Algorithms*, 39(2):205–222, 2001. See also FSTTCS'98.

[22] Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal Bloom filter replacement. In *Proc. 16th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 823–829, 2005.

[23] Rasmus Pagh. Low redundancy in static dictionaries with constant query time. *SIAM Journal on Computing*, 31(2):353–363, 2001. See also ICALP'99.

[24] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004. See also ESA'01.

[25] Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th ACM Symposium on Theory of Computing (STOC)*, pages 232–240, 2006.

[26] Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proc. 13th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, 233–242, 2002.

Strict Fibonacci Heaps

Gerth Stølting Brodal
MADALGO*
Dept. of Computer Science
Aarhus University
Åbogade 34, 8200 Aarhus N
Denmark
gerth@cs.au.dk

George Lagogiannis
Agricultural University
of Athens
Iera Odos 75, 11855 Athens
Greece
lagogian@aua.gr

Robert E. Tarjan†
Dept. of Computer Science
Princeton University
and HP Labs
35 Olden Street, Princeton
New Jersey 08540, USA
ret@cs.princeton.edu

ABSTRACT

We present the first pointer-based heap implementation with time bounds matching those of Fibonacci heaps in the worst case. We support make-heap, insert, find-min, meld and decrease-key in worst-case $O(1)$ time, and delete and delete-min in worst-case $O(\lg n)$ time, where n is the size of the heap. The data structure uses linear space.

A previous, very complicated, solution achieving the same time bounds in the RAM model made essential use of arrays and extensive use of redundant counter schemes to maintain balance. Our solution uses neither. Our key simplification is to discard the structure of the smaller heap when doing a meld. We use the pigeonhole principle in place of the redundant counter mechanism.

Categories and Subject Descriptors

E.1 [**Data Structures**]: Trees; F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*

General Terms

Algorithms

Keywords

Data structures, heaps, meld, decrease-key, worst-case complexity

*Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

†Partially supported by NSF grant CCF-0830676, US-Israel Binational Science Foundation Grant 2006204, and the Distinguished Visitor Program of the Stanford University Computer Science Department. The information contained herein does not necessarily reflect the opinion or policy of the federal government and no official endorsement should be inferred.

STOC.12, May 19–22, 2012, New York, New York, USA.

1. INTRODUCTION

Williams in 1964 introduced binary heaps [25]. Since then the design and analysis of heaps has been thoroughly investigated. The most common operations supported by the heaps in the literature are those listed below. We assume that each item stored contains an associated key. No item can be in more than one heap at a time.

makeheap() Create a new, empty heap and return a reference to it.

insert(H, i) Insert item i, not currently in a heap, into heap H, and return a reference to where i is stored in H.

meld(H_1, H_2) Return a reference to a new heap containing all items in the two heaps H_1 and H_2 (H_1 and H_2 cannot be accessed after meld).

find-min(H) Return a reference to where the item with minimum key is stored in the heap H.

delete-min(H) Delete the item with minimum key from the heap H.

delete(H, e) Delete an item from the heap H given a reference e to where it is stored.

decrease-key(H, e, k) Decrease the key of the item given by the reference e in heap H to the new key k.

There are many heap implementations in the literature, with a variety of characteristics. We can divide them into two main categories, depending on whether the time bounds are worst case or amortized. Most of the heaps in the literature are based on heap-ordered trees, i.e. tree structures where the item stored in a node has a key not smaller than the key of the item stored in its parent. Heap-ordered trees give heap implementations that achieve logarithmic time for all the operations. Early examples are the implicit binary heaps of Williams [25], the leftist heaps of Crane [5] as modified by Knuth [20], and the binomial heaps of Vuillemin [24].

The introduction of Fibonacci heaps [15] by Fredman and Tarjan was a breakthrough since they achieved $O(1)$ amortized time for all the operations above except for delete and delete-min, which require $O(\lg n)$ amortized time, where n is the number of items in the heap and lg the base-two logarithm. The drawback of Fibonacci heaps is that they are complicated compared to existing solutions and not as efficient in practice as other, theoretically less efficient solutions. Thus, Fibonacci heaps opened the way for further

progress on the problem of heaps, and many solutions based on the amortized approach have been presented since then, trying to match the time complexities of Fibonacci heaps while being at the same time simpler and more efficient in practice.

Self adjusting data structures provided a framework towards this direction. A self-adjusting data structure is a structure that does not maintain structural information (like size or height) within its nodes, but still can adjust itself to perform efficiently. Within this framework, Sleator and Tarjan introduced the skew heap [23], which was an amortized version of the leftist heap. They matched the complexity of Fibonacci heaps on all the operations except for decrease-key, which takes $O(\lg n)$ amortized time. The pairing heap, introduced by Fredman, Sedgewick, Sleator and Tarjan [14], was the amortized version of the binomial heap, and it achieved the same time complexity as Fibonacci heaps except again for the decrease-key, the time complexity of which remained unknown for many years. In 1999, Fredman [13] proved that the lower bound for the decrease-key operation on pairing heaps is $\Omega(\lg \lg n)$; thus the amortized performance of pairing heaps does not match the amortized performance of Fibonacci heaps. In 2005, Pettie [22] proved that the time complexity of the decrease-key operation is $2^{O(\sqrt{\lg \lg n})}$. Later, Elmasry [9] gave a variant of pairing heaps that needs only $O(\lg \lg n)$ amortized time for decrease-key.

Heaps having amortized performance matching the amortized time complexities of Fibonacci heaps have also been presented. In particular, Driscoll, Gabow, Shrairman and Tarjan [6] proposed rank-relaxed heaps, Kaplan and Tarjan [19] presented thin heaps, Chan [4] introduced quake heaps, Haeupler, Sen and Tarjan introduced rank-pairing heaps [16], and Elmasry introduced violation heaps [10]. Elmasry improved the number of comparisons of Fibonacci heaps by a constant factor [7] and also examined versions of pairing heaps, skew heaps, and skew-pairing heaps [8]. Some researchers, aiming to match the amortized bounds of Fibonacci heaps in a simpler way, followed different directions. Peterson [21] presented a structure based on AVL trees and Høyer [17] presented several structures, including ones based on red-black trees, AVL trees, and (a, b)-trees.

Let us now review the progress on this problem, based on the worst-case approach. The goal of worst-case efficient heaps is to eliminate the unpredictability of amortized ones, since this unpredictability is not desired in e.g. real time applications.

The targets for the worst case approach were given by Fibonacci heaps, i.e. the time bounds of Fibonacci heaps should ideally be matched in the worst case. The next improvement after binomial heaps came with the the implicit heaps of Carlsson, Munro and Poblete [3] supporting worst-case $O(1)$ time insertions and $O(\lg n)$ time deletions on a single heap stored in an array. Run-relaxed heaps [6] achieve the amortized bounds given by Fibonacci heaps in the worst case, with the exception of the meld operation, which is supported in $O(\lg n)$ time in the worst case. The same result was later also achieved by Kaplan and Tarjan [18] with fat heaps. Fat heaps without meld can be implemented on a pointer machine, but to support meld in $O(\lg n)$ time arrays are required. The meld operation was the next target for achieving constant time in the worst case framework, in order to match the time complexities of Fibonacci heaps. Brodal [1] achieved $O(1)$ worst case time for the meld operation on a pointer machine, but not for the decrease-key operation. It then became obvious that although the decrease-key and the meld operation can be achieved in $O(1)$ worst case time separately, it is very difficult to achieve constant time for both operations in the same data structure. Brodal [2] managed to solve this problem, but his solution is very complicated and requires the use of (extendable) arrays. For the pointer machine model of computation, the problem of matching the time bounds of Fibonacci heaps remained open until now, and progress within the worst case framework has been accomplished only in other directions. In particular, Elmasry, Jensen and Katajainen presented two-tier relaxed heaps [12] in which the number of key comparisons is reduced to $\lg n + 3 \lg \lg n + O(1)$ per delete operation. They also presented [11] a new idea (which we adapt in this paper) for handling decrease-key operations by introducing structural violations instead of heap order violations.

1.1 Our contribution

In this paper we present the first heap implementation that matches the time bounds of Fibonacci heaps in the worst case on a pointer machine, i.e. we achieve a linear space data structure supporting make-heap, insert, find-min, meld and decrease-key in worst-case $O(1)$ time, and delete and delete-min in worst-case $O(\lg n)$ time. This adds the final step after the previous step made by Brodal [2] and answers the long standing open problem of whether such a heap is possible.

Much of the previous work, including [1, 2, 3, 11, 12, 18], used redundant binary counting schemes to keep the structural violations logarithmically bounded during operations. For the heaps described in this paper we use the simpler approach of applying the pigeonhole principle. Our heaps are essentially heap ordered trees, where the structural violations are subtrees being cut off (and attached to the root), as in Fibonacci heaps. The crucial new idea is that when melding two heaps, the data structures maintained for the smaller tree are discarded by marking all these nodes as being ***passive***. We mark all (active) nodes in the smaller tree passive in $O(1)$ time using an indirectly accessed shared flag.

In Sections 2-4 we describe our data structure, ignoring the pointer-level representation, and analyze it in Section 5. The pointer-level representation is given in Section 6. In Section 7 we give some concluding remarks and discuss possible variations.

2. DATA STRUCTURE AND INVARIANTS

In this section we describe our data structure on an abstract level. The representation at the pointer level is given in Section 6.

A heap storing n items is represented by a single ordered tree with n nodes. Each node stores one item. The ***size*** of a tree is the number of nodes it contains. The ***degree*** of a node is the number of children of the node. We assume that all keys are distinct; if not, we break ties by item identifier. We let x.key denote the key of the item stored in node x. The items satisfy ***heap order***, i.e. if x is a child of y then x.key $>$ y.key. Heap order implies that the item with minimum key is stored in the root.

The basic idea of our construction is to ensure that all nodes have logarithmic degree, that a meld operation makes the root with the larger key a child of the root with the

smaller key, and that a decrease-key operation on a node detaches the subtree rooted at the node and reattaches it as a subtree of the root. To guide the necessary restructuring, we need the following concepts and invariants.

Each node is marked either ***active*** or ***passive***. An active node with a passive parent is called an ***active root***. The ***rank*** of an active node is the number of active children. Each active node is assigned a non-negative integer ***loss***. The ***total loss*** of a heap is the sum of the loss over all active nodes. A passive node is ***linkable*** if all its children are passive.

In order to keep the node degrees logarithmic during deletions, we maintain all nodes of a heap except for the root in a queue Q. A non-root node has ***position*** p if it is the p-th node on the queue Q.

Invariants

Let $R = 2\lg n + 6$. Note that R is not necessarily an integer. We later show that R is a bound on the rank of active nodes (Corollary 1). The value of R is only needed for the analysis — it is not maintained by the algorithms.

I1 (Structure) For all nodes the active children are to the left of the passive children. The root is passive and the linkable passive children of the root are the rightmost children. For an active node, the i-th rightmost active child has rank+loss at least $i-1$. An active root has loss zero.

I2 (Active roots) The total number of active roots is at most $R+1$.

I3 (Loss) The total loss is at most $R+1$.

I4 (Degrees) The maximum degree of the root is $R+3$. Let x be a non-root node, and let p denote its position in Q. If x is a passive node or an active node with positive loss its degree is at most $2\lg(2n-p)+9$; otherwise, x is an active node with loss zero and is allowed to have degree one higher, i.e. degree at most $2\lg(2n-p)+10$.

Note that I4 implies that all nodes have degree at most $2\lg n+12$. The above invariants imply a bound on the maximum rank an active node can have. The following lemma captures how the maximum rank depends on the value of R, for arbitrary values of R.

LEMMA 1. *If I1 is satisfied and the total loss is L, then the maximum rank is at most* $\lg n + \sqrt{2L} + 2$.

PROOF. Assume x is an active node of maximum rank $r \geq k+1+\lg n$, where k is the minimum integer such that $k(k+1)/2 \geq L$. We will prove the contradiction that the subtree rooted at x contains at least $n+1$ nodes. Let T_x be the subtree rooted at x. We prune from T_x all subtrees rooted at passive nodes. If y is a child of x, z is a child of y, and there is a node with positive loss in the subtree rooted at z, then we prune the subtree rooted at z and increase the loss of y by one (so that I1 remains satisfied). This removes the positive loss contributed by the subtree rooted at z, and only increases the loss of y by one, i.e. the total loss is still bounded by L. Now only the children of x can have a positive loss. We reduce the rank+loss of the i-th rightmost child of x to $i-1$, by lowering the loss and possibly pruning grandchildren. Finally, for all nodes $v \neq x$ we repeatedly prune grandchildren such that the i-th rightmost child of v has degree exactly $i-1$. The remaining subtrees T_v are binomial trees of size $2^{\text{degree}(v)}$. The minimum size of such a T_x is achieved by starting with a binomial tree of size 2^r, and repeating the following step L times: prune a grandchild of the root with maximum degree. Since the maximum grandchild degree of a binomial tree of size 2^r is $r-2$, and generally there are j grandchildren of degree $r-j-1$, there are $\sum_{j=1}^{k} j = k(k+1)/2$ grandchildren of degree $\geq r-k-1$. Since $k(k+1)/2 \geq L$, no grandchild of degree $\leq r-k-2$ is pruned, i.e. the $(r-k)$-th rightmost child w of x has degree $r-k-1$ and loss zero. By the assumption on r, the degree of w is $\geq \lg n$ and T_w has size $\geq n$. It follows that T_x has size at least $n+1$, which is a contradiction. This gives $r < k+1+\lg n \leq \sqrt{2L}+2+\lg n$, since $(k-1)k/2 < L$. □

By I3 we have $L \leq R+1$. Since $\lg n + \sqrt{2(R+1)}+2 \leq R$ for $R = 2\lg n + 6$, we have the following corollary:

COROLLARY 1. *All nodes have rank $\leq R$.*

The corollary implies a bound on the maximal rank before a heap operation. If we violate I2 or I3 temporarily during a heap operation, then the pigeonhole principle guarantees that we can apply the transformations described in Section 3. If the total loss is $> R+1$, then there exists either a node with loss at least two, or there exist two nodes with equal rank each with loss exactly one. Similarly if there are $> R+1$ active roots, then at least two active roots have the same rank. Finally, if I1-I3 are satisfied but the root violates I4, then the root has at least three passive linkable children, since the root has at most $R+1$ children or grandchildren that can be active roots, i.e. at most $R+1$ children of the root are active roots or passive non-linkable nodes.

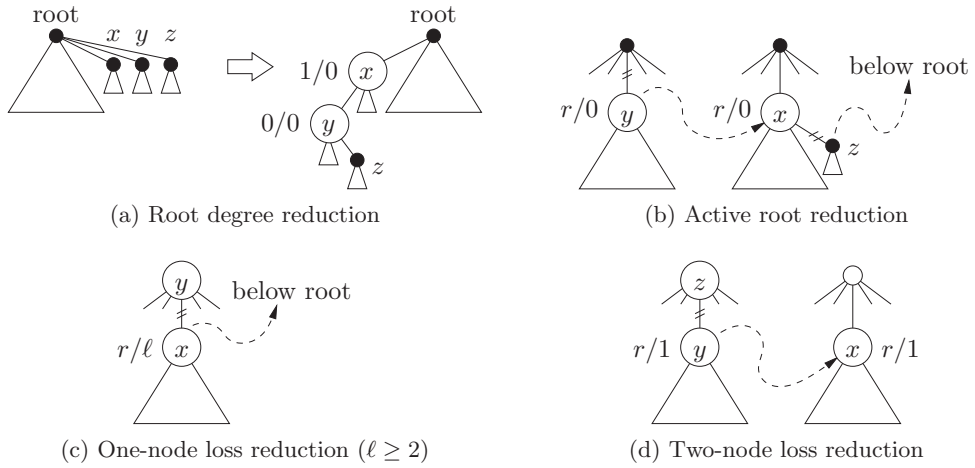
3. TRANSFORMATIONS

The basic transformation is to ***link*** a node x and its subtree below another node y, by removing x from the child list of its current parent and making x a child of y. If x is active it is made the leftmost child of y; if x is passive it is made the rightmost child of y.

The following transformations use link to reestablish the invariants I2-I4 when they get violated. The transformations are illustrated in Figure 1 and the main properties of the transformations are captured by Table 1.

Active root reduction Let x and y be active roots of equal rank r. Compare x.key and y.key. Assume w.l.o.g. x.key $<$ y.key. Link y to x and increase the rank of x by one. If the rightmost child z of x is passive make z a child of the root. In this transform the number of active roots is decreased by one and the degree of the root possibly increased by one.

Root degree reduction Let x, y, z be the three rightmost passive linkable children of the root. Using three comparisons, sort x, y, z by key. Assume w.l.o.g. x.key $<$ y.key $<$ z.key. Mark x and y as active. Link z to y, and link y to x. Make x the leftmost child of the root. Assign both x and y loss zero, and rank one and zero respectively. In this transform both x and y change from being passive to active with loss zero, both get one more child, and x becomes a new active root. The degree of the root decreases by two and the number of active roots increases by one.



(a) Root degree reduction

(b) Active root reduction

(c) One-node loss reduction ($\ell \geq 2$)

(d) Two-node loss reduction

Figure 1: Transformations to reduce the root degree, the number of active roots, and the total loss. Black/white nodes are passive/active nodes. For an active node r/ℓ shows rank/loss.

Loss reduction To reduce the total loss we have two different transformations. The ***one-node loss reduction*** applies when there exists an active node x with loss ≥ 2. Let y be the parent of x. In this case x is linked to the root and made an active root with loss zero, and the rank of y is decreased by one. If y is not an active root, the loss of y is increased by one. Since the loss of x decreases by at least two, the total loss is decreased by at least one. The second transformation, ***two-node loss reduction***, applies when two active nodes x and y with rank r both have a loss of exactly one. Compare x.key and y.key. Assume w.l.o.g. x.key $< y$.key. Let z be the parent of y. Link y to x, increase the rank of x, and set the loss of x and y to zero. The degree and rank of z is decreased by one. If z is not an active root, the loss of z is increased by one.

Certain combinations of active root reductions and root degree reductions have only beneficial effects (see Table 1). When doing such combinations, we do the reductions "to the extent possible": we do them in any order, stopping only when all reductions are done or when no undone reduction can be done. An active root reduction and a root degree reduction decrease the root degree by at least one. Two active root reductions and a root degree reduction decrease the number of active roots by one without increasing the root degree. Three active root reductions and two root degree reductions decrease both the number of active roots and the root degree by at least one.

It should be noted that the distinct key assumption together with the heap order invariant ensures that no cycles are created in the tree when an active root reduction or two-node loss reduction is performed.

4. IMPLEMENTATION OF THE HEAP OPERATIONS

The various heap operations are implemented as follows. To find the minimum in a heap, return the item in the root. To make an empty heap, return an empty tree. To insert an item into a heap, create a new one-node tree with a passive root containing the item, and meld this with the existing heap. To delete an arbitrary item, decrease its key to minus infinity and do a minimum deletion.

To decrease the key of the item in node x, in the tree with root z, begin by decreasing the key of the item. If x is the root we are done. Otherwise, if x.key $< z$.key, swap the items in x and z (actually we assume each node only stores a pointer to the item that is stored externally with a pointer to the node of the item). Let y be the parent of x. Make x a child of the root. If x was an active node but not an active root, then x becomes an active root with loss zero and the rank of y is decreased by one. If y is active but not an active root, then the loss of y is increased by one. Do a loss reduction if possible. Finally, do six active root reductions and four root degree reductions to the extent possible.

To delete the minimum in the tree with root z, first find the node x of minimum key among the children of the root. If x is active then make x passive and all active children of x become active roots. Make each of the other children of z a child of x. Make the passive linkable children of x the rightmost children of x. Remove x from Q. Destroy z. Repeat twice: move the front node y on Q to the back; link the two rightmost children of y to x, if they are passive. Do a loss reduction if possible. Do active root reductions and root degree reductions in any order until none of either is possible.

To meld two heaps with roots x and y, rename x and y if necessary so that the tree rooted at x has size at most the size of the tree rooted at y. Make all nodes in the tree rooted at x passive. (Do this implicitly, as described in Section 6, so that it takes $O(1)$ time.) Let u be the root of smaller key and v the other root. Make v a child of u. Set $Q = Q_x \& [v] \& Q_y$, where "&" denotes catenation and Q_x and Q_y are the queues of the heaps with root x and y respectively. Do an active root reduction and a root degree reduction to the extent possible.

This method of melding "forgets" the structure of the tree of smaller size, which eliminates the need to combine complicated data structures during melding. This is the main novelty in the presented data structure.

Table 1: Effect of the different transformations

	Root degree	Total loss	Active roots	Key comparisons
Active root reduction (A)	$\leq +1$	0	-1	$+1$
Root degree reduction (R)	-2	0	$+1$	$+3$
Loss reduction	$\leq +1$	≤ -1	$\leq +1$	$\leq +1$
– one-node	$+1$	≤ -1	$+1$	0
– two-node	0	-1	0	$+1$
(A) + (R)	≤ -1	0	0	$+4$
2×(A) + (R)	≤ 0	0	-1	$+5$
3×(A) + 2×(R)	≤ -1	0	-1	$+9$

Table 2: The changes caused by the different heap operations

	Root degree	Total loss	Active roots
decrease-key	$\leq 1+1+6-8$	$\leq 1-1+0+0$	$\leq 1+1-6+4$
meld	$\leq 1+0+1-2$	$\leq 0+0+0+0$	$\leq 0+0-1+1$
delete-min	$\leq (2\lg n+12+4)+1$	$\leq 0-1$	$\leq R+1$

5. CORRECTNESS

In the following we verify that each operation preserves the invariants I1-I4.

For I1 the interesting property to verify is that the i-th active child of an active node has rank+loss $\geq i-1$. All other properties in I1 are straightforward to verify. We start by observing that if an active child x is detached from its parent y satisfying I1, then I1 is also satisfied for y after the detachment. This follows since the i-th rightmost active child z after the detachment was either the i-th or $(i+1)$-st active child before the detachment, i.e. for z the new rank+loss is at least $i-1$. An active node only gets a new active child as a result of an active root reduction, two-node loss reduction, or root degree reduction. In the first two cases a new $(r+1)$-st rightmost active child is added with rank r (and loss zero), and in the later case I1 holds by construction for the two new active nodes.

For invariants I2 and I3, Table 2 captures the change to the degree of the root, the total loss, and the number of active roots when performing the operations decrease-key, meld, and delete-min, respectively. Each entry is a sum of four terms stating the change caused by the initial transformations performed by the operations, and by the loss reduction transformations, active root reductions, and root degree reductions. Each entry is an upper bound on the change, except for the cases where no reduction is possible (e.g. the loss increases, but is still $\leq R+1$). For meld the root degree is the change to the old root that becomes the new root, whereas total loss and number of active roots is compared to the heap that is not made passive. For delete-min the bounds are stated before the repeated active root and root degree transformations are applied.

Observe that for both decrease-key and meld all sums are zero, and that R increases during a meld, i.e. invariants I2 and I3 remain valid for each of these operations. Delete-min reduces the size of the heap by one, reducing $R = 2\lg n + 6$ by at most one, if $n \geq 4$ before the delete-min operation (if $n \leq 3$ before the delete-min operation, I2 and I3 are trivially true after). The reduction in loss by one ensures that I3 is valid after delete-min. Since active root reductions are performed until they are not possible, I2 trivially holds — provided that the repeated application of active root reductions and root degree reductions terminate. Termination immediately follows from the facts that both reductions reduce the measure

$$2 \cdot \text{root degree} + 3 \cdot \#\text{active roots}$$

by one, and that the initial value of this measure is $O(\log n)$. This immediately also implies the claimed time bounds for our heap.

To prove the validity of I4, we first observe that for the degree of the root we can use the same argument as above using Table 2.

During the transformations a non-root node can only increase its degree in three cases. During an active root reduction there is no passive right-child to detach. In this case all children of the node are active, and the degree bound follows from Corollary 1. During a two-node loss reduction the degree of the node x increases by one, but this is okay by I4 since the loss of the node decreases from one to zero. During a root degree transformation two passive nodes become active, both getting loss zero and degree increased by one. Again this is okay by I4. During a meld one root becomes a non-root, but since $R+3 \leq 2\lg(2n-p)+9$ for all possible p, again I4 holds. The interesting case is when we perform a delete-min operation. I4 holds for the root trivially, since we repeatedly perform root degree reductions until none are possible. For non-root nodes we observe that their invariant is strengthened since R decreases. The role of Q is to deal with this case. By removing the two first nodes in the queue, all nodes get their position in Q decreased by two or three (depending if they were in front of the deleted node in the queue). By observing that $2n-p$ does not increase in this case, it follows that the invariant for non-root nodes is not strengthened and I4 remains valid. For the two nodes moved to the end of the queue the term $2\lg(2n-p)$ decreases by two, implying that their degree constraint is strengthened by two. Since we detach two passive nodes from these nodes I4 remains valid for these nodes also. During meld all elements in the smaller heap become passive, and their degree constraint goes from the "+10" to the "+9" case. But since they remain in their position in the queue, and the result-

ing queue is at least twice the size, we have that $2n - p$ increases by a factor two, and I4 remains valid for the elements in the smaller heap. For the elements in the larger of the two heaps, they keep their active status. Both n and p increase for these elements by the size of the smaller queue. Therefore $2n - p$ is non-decreasing and I4 remains valid.

A note on the loss: In the previous description the loss of a node is assumed to be a non-negative integer. Even though the loss plays an essential role in invariant I1, only values 0, 1, and $'' \geq 2''$ are relevant for the algorithm. As soon as the loss is ≥ 2, then the loss can only decrease when it is set to zero by a one-node loss transformation. Since the algorithm only tests if the loss is zero, one or ≥ 2, it is sufficient for the algorithm to keep track of these three states. It follows that we can store the loss using only two bits.

6. REPRESENTATION DETAILS

To represent a heap we have the following types of records, also illustrated in Figures 2 and 3. Each node is represented by a ***node record***. To mark if a node is active or passive we will not store the flag directly in the node but indirectly in an ***active record***. In particular all active nodes of a tree point to the same active record. This allows all nodes of a tree to be made passive in $O(1)$ time by only changing a single flag. The entry point to a heap is a pointer to a ***heap record***, that has a pointer to the root of the tree and additional information required for accessing the relevant nodes for performing the operations described in Section 4.

We call a rank ***active-root transformable***, if there are at least two active roots of that rank. We call a rank ***loss transformable***, if the total loss of the nodes of that rank is at least two. For each rank we maintain a node, and all such nodes belong to the ***rank-list***. The rightmost node corresponds to rank zero, and the node that corresponds to rank k is the left sibling of the one that corresponds to $k-1$. (see Figure 3). We maintain all active nodes that potentially could participate in one of the transformations from Section 3 (i.e. active roots and active nodes with positive loss) in a list called the ***fix-list***. Each node with rank k on the fix-list points to node k of the rank-list. The fix-list is divided left-to-right into four parts (1-4), where Parts 1-2 contain active roots and Parts 3-4 contain nodes with positive loss. Part 1 contains the active roots of active-root transformable ranks. All the active roots of the same rank are adjacent, and one of the nodes has a pointer from the corresponding node of the rank-list. Part 2 contains the remaining active roots. Each node has a pointer from the corresponding node of the rank-list. Part 3 contains active nodes with loss one and a rank that is not loss-transformable. Each node of this part has a pointer from the corresponding node of the rank-list. Finally, Part 4 contains all the active nodes of loss transformable rank. As in Part 1, all nodes of equal rank are adjacent and one of the nodes is pointed to by the corresponding node of the rank-list. Observe that for some ranks there may exist only one node in Part 1 (because if the loss of a node is at least two, its rank is loss-transformable).

From the above description it follows that we can always perform an active root reduction as long as Part 1 of the fix-list is nonempty, and we can always perform a loss reduction transformation as long as Part 4 is nonempty. We maintain a pointer (in the heap record) indicating the boundary between Parts 2 and 3. The above construction and pointers (see Figure 3 for more details) allow us to take the following

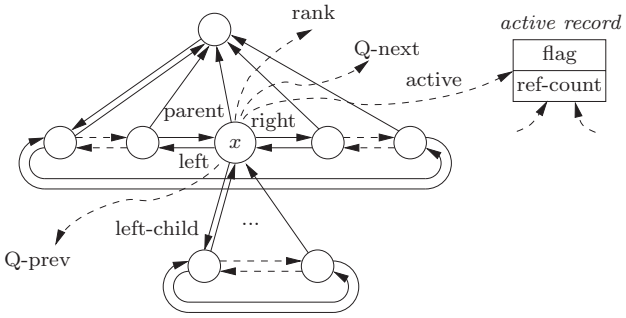


Figure 2: The fields of a node record x and an active record. The fields of x not shown are the item and the loss.

actions: In order to perform a loss reduction transformation, we go to the right-end of the fix-list and perform a one-node loss reduction (if we access a node of multiple loss) or a two-node loss reduction (if the two rightmost nodes of the fix-list have loss one). Otherwise Part 4 is empty. After a loss reduction transformation on a rank k, we may have to transfer one node of rank k into Part 3, if its loss is one and it is the last node in Part 3 with this rank. Whenever the loss of a node that has a loss-transformable rank increases, we insert it into (the appropriate group of) Part 4. If its rank is not loss transformable, we insert it into Part 3, unless there is another node of the same rank there, in which case we move both nodes into Part 4 at the right end of the fix-list.

In order to perform an active root reduction, we go to the left end of the fix-list and link the two leftmost active roots, if they have the same rank (otherwise, Part 1 is empty). If after the reduction, the two leftmost nodes in the fix-list are not active roots of equal degree, we transfer the leftmost node into Part 2. When an active node of rank k becomes an active root and there is no other active root of the same rank, we insert the new active root into Part 2. Otherwise, we insert it adjacent to the active roots of that rank, unless only one active root has this rank (i.e. it is located in Part 2), in which case we transfer the existing active root of rank k with the new one into Part 1 (at the left end of the fix-list). When the rank of a node that belongs to the fix-list changes, we can easily perform the necessary updates to the fix-list so that all parts of the list are consistent with the above description. The details are straightforward and thus omitted.

The details of the fields of the individual records are as follows (see also Figures 2 and 3).

Node record

item A pointer to the item (including its associated key).

left, right, parent, left-child Pointers to the node records for the the left and right sibling of the node (the left and right pointers form a cyclic linked list), the parent node, and the leftmost child. The later two are NULL if the nodes do not exist.

active Pointer to an active record, indicating if the node is active or passive.

Q-prev, Q-next Pointers to the node records for the previous and the next node on the queue Q, which is maintained as a cyclic linked list.

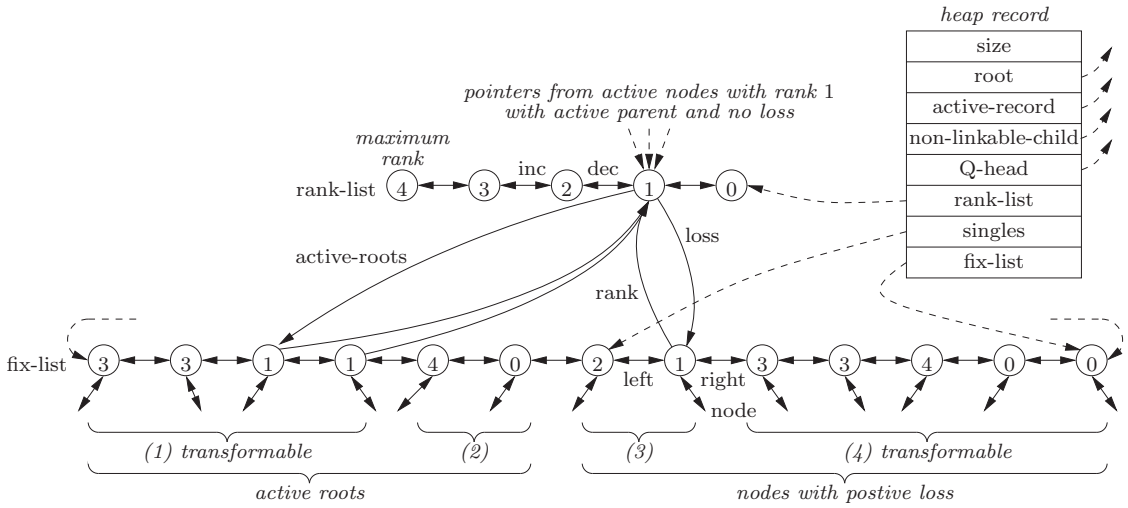


Figure 3: The heap-record, rank-list, and fix-list. The ref-count field in the records of the rank-list is not shown. Only pointers for nodes with rank equal to one are shown. The numbers in the nodes in the rank-list and fix-list are the ranks of the active nodes pointing to these nodes; these numbers are not stored.

loss Non-negative integer equal to the loss of the node. Undefined if the node is passive.

rank If the node is passive, the value of the pointer is not defined (it points to some old node that has been released for garbage collection). If the node is an active root or an active node with positive loss (i.e. it is on the fix-list), rank points to the corresponding record in the fix-list. Otherwise rank points to the record in the rank-list corresponding to the rank of the node. (The cases can be distinguished using the active field of the node and the parent together with the loss field).

Active record

flag A Boolean value. Nodes pointing to this record are active if and only if the flag is true.

ref-count The number of nodes pointing to the record. If flag is false and ref-count = 0, then the node is released for garbage collection.

Heap record

size An integer equal to the number of items in the heap.

root A pointer to the node record of the root (NULL if and only if the heap is empty).

active-record A pointer to the active record shared by all active nodes in the heap (one distinct active record for each heap).

non-linkable-child A pointer to the node record of the leftmost passive non-linkable child of the root. If all passive children of the root are linkable the pointer is to the rightmost active child of the root. Otherwise it is NULL.

Q-head A pointer to the node record for the node that is the head of the queue Q.

rank-list A pointer to the rightmost record in the rank-list.

fix-list A pointer to the rightmost node in the fix-list.

singles A pointer to the leftmost node in the fix-list with a positive loss, if such a node exists. Otherwise it is NULL.

Rank-list record (representing rank r)

inc, dec Pointers to the records on the rank-list for rank $r+1$ and $r-1$, if they exist. Otherwise they are NULL.

loss A pointer to a record in the fix-list for an active node with rank r and positive loss. NULL if no such node exists.

active-roots A pointer to a record in the fix-list for an active root with rank r. NULL if no such node exists.

ref-count The number of node records and fix-list records pointing to this record. If the leftmost record on the rank-list gets a ref-count = 0, then the record is deleted from the rank-list and is released for garbage collection.

Fix-list record

node A pointer to the node record for the node.

left, right Pointers to the left and right siblings on the fix-list, that is maintained as a cyclic linked list.

rank A pointer to the record in the rank-list corresponding to the rank of this node.

A detail concerning garbage collection: When performing a meld operation on two heaps, all nodes in one heap are first made passive by clearing the flag in the active record given by the heap record. The heap record, the rank-list, and the fix-list for this heap are released for incremental garbage collection.

We now bound the space required by our structure. For each item we have one node record and possibly one fix-list record. The number of active records is bounded by the number of nodes, since in the worst case each active record has a ref-count = 1. Finally for each heap we have one heap-record and a number of rank-list records bounded by one plus the maximum rank of a node, i.e. logarithmic in the size of the heap. It follows that the total space for a heap is linear in the number of stored items.

7. CONCLUSION

We have described the first pointer-based heap implementation achieving the performance of Fibonacci heaps in the worst-case. What we presented is just one possible implementation. Based on the active/passive node idea we have considered many alternative implementations. One option is to allow a third explicit marking "active root" (instead of an active root being a function of the active/passive marks), such that a child of an active node can also be an active root. This eliminates the distinction between passive linkable and non-linkable nodes. Another option is to allow the root to be active also. This simplifies the decrease-key operation, since then the node getting a smaller key can also be made the root, without ever swapping items in the nodes. A third option is to adopt redundant counters instead of the pigeonhole principle to bound the number of active roots and holes created by cutting of subtrees. Further alternatives are to consider ternary linking instead of binary linking, and using a different "inactivation" criterion during meld, e.g. size plus number of active nodes. All these variations can be combined, each solution implying different bounds on the maximum degrees, constants in the time bounds, and complexity in the reduction transformations. In the presented solution we aimed at reducing the complexity in the description, whereas the constants in the solution were of secondary interest.

8. REFERENCES

[1] Gerth Stølting Brodal. Fast meldable priority queues. In *Proc. 4th International Workshop Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 282–290. Springer, 1995.

[2] Gerth Stølting Brodal. Worst-case efficient priority queues. In *Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 52–58. SIAM, 1996.

[3] Svante Carlsson, J. Ian Munro, and Patricio V. Poblete. An implicit binomial queue with constant insertion time. In *Proc. 1st Scandinavian Workshop on Algorithm Theory*, volume 318 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1988.

[4] Timothy M. Chan. Quake heaps: a simple alternative to Fibonacci heaps. Manuscript, 2009.

[5] Clark Allan Crane. *Linear lists and priority queues as balanced binary trees*. PhD thesis, Stanford University, Stanford, CA, USA, 1972.

[6] James R. Driscoll, Harold N. Gabow, Ruth Shrairman, and Robert E. Tarjan. Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation. *Communications of the ACM*, 31(11):1343–1354, 1988.

[7] Amr Elmasry. Layered heaps. In *Proc. 9th Scandinavian Workshop on Algorithm Theory*, volume 3111 of *Lecture Notes in Computer Science*, pages 212–222. Springer, 2004.

[8] Amr Elmasry. Parameterized self-adjusting heaps. *J. Algorithms*, 52(2):103–119, 2004.

[9] Amr Elmasry. Pairing heaps with $o(\log \log n)$ decrease cost. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 471–476. SIAM, 2009.

[10] Amr Elmasry. The violation heap: A relaxed Fibonacci-like heap. In *Proc. 16th Annual International Conference on Computing and Combinatorics*, volume 6196 of *Lecture Notes in Computer Science*, pages 479–488. Springer, 2010.

[11] Amr Elmasry, Claus Jensen, and Jyrki Katajainen. On the power of structural violations in priority queues. In *Proc. 13th Computing: The Australasian Theory Symposium.*, volume 65 of *CRPIT*, pages 45–53. Australian Computer Society, 2007.

[12] Amr Elmasry, Claus Jensen, and Jyrki Katajainen. Two-tier relaxed heaps. *Acta Informatica*, 45(3):193–210, 2008.

[13] Michael L. Fredman. On the efficiency of pairing heaps and related data structures. *Journal of the ACM*, 46(4):473–501, 1999.

[14] Michael L. Fredman, Robert Sedgewick, Daniel Dominic Sleator, and Robert Endre Tarjan. The pairing heap: A new form of self-adjusting heap. *Algorithmica*, 1(1):111–129, 1986.

[15] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[16] Bernhard Haeupler, Siddhartha Sen, and Robert Endre Tarjan. Rank-pairing heaps. In *Proc. 17th Annual European Symposium on Algorithsm*, volume 5757 of *Lecture Notes in Computer Science*, pages 659–670. Springer, 2009. SIAM Journal of Computing, to appear.

[17] Peter Høyer. A general technique for implementation of efficient priority queues. In *Proc. Third Israel Symposium on the Theory of Computing and Systems*, pages 57–66, 1995.

[18] Haim Kaplan and Robert E. Tarjan. New heap data structures. Technical report, Department of Computer Science, Princeton University, 1999.

[19] Haim Kaplan and Robert Endre Tarjan. Thin heaps, thick heaps. *ACM Transactions on Algorithms*, 4(1), 2008.

[20] Donald E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms, 2nd Edition*. Addison-Wesley, 1973.

[21] Gary L. Peterson. A balanced tree scheme for meldable heaps with updates. Technical Report GIT-ICS-87-23, School of Information and Computer Science, Georgia Institute of Technology, 1987.

[22] Seth Pettie. Towards a final analysis of pairing heaps. In *Proc. 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 174–183. IEEE Computer Society, 2005.

[23] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting heaps. *SIAM Journal of Computing*, 15(1):52–69, 1986.

[24] Jean Vuillemin. A data structure for manipulating priority queues. *Communications of the ACM*, 21(4):309–315, 1978.

[25] John William Joseph Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, 1964.

The Cell Probe Complexity of Dynamic Range Counting

Kasper Green Larsen*
MADALGO†, Department of Computer Science
Aarhus University
Aarhus, Denmark
larsen@cs.au.dk

ABSTRACT

In this paper we develop a new technique for proving lower bounds on the update time and query time of dynamic data structures in the cell probe model. With this technique, we prove the highest lower bound to date for any explicit problem, namely a lower bound of $t_q = \Omega((\lg n/\lg(wt_u))^2)$. Here n is the number of update operations, w the cell size, t_q the query time and t_u the update time. In the most natural setting of cell size $w = \Theta(\lg n)$, this gives a lower bound of $t_q = \Omega((\lg n/\lg\lg n)^2)$ for any polylogarithmic update time. This bound is almost a quadratic improvement over the highest previous lower bound of $\Omega(\lg n)$, due to Pătraşcu and Demaine [SICOMP'06].

We prove our lower bound for the fundamental problem of weighted orthogonal range counting. In this problem, we are to support insertions of two-dimensional points, each assigned a $\Theta(\lg n)$-bit integer weight. A query to this problem is specified by a point $q = (x, y)$, and the goal is to report the sum of the weights assigned to the points dominated by q, where a point (x', y') is dominated by q if $x' \leq x$ and $y' \leq y$. In addition to being the highest cell probe lower bound to date, our lower bound is also tight for data structures with update time $t_u = \Omega(\lg^{2+\varepsilon} n)$, where $\varepsilon > 0$ is an arbitrarily small constant.

Categories and Subject Descriptors

E.1 [**Data**]: Data Structures

Keywords

Data Structures, Cell Probe, Lower Bounds, Range Searching, Computational Geometry

*Kasper Green Larsen is a recipient of the Google Europe Fellowship in Search and Information Retrieval, and this research is also supported in part by this Google Fellowship.
†Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

STOC'12, May 19–22, 2012, New York, New York, USA.

1. INTRODUCTION

Proving lower bounds on the operational time of data structures has been an active line of research for decades. During these years, numerous models of computation have been proposed, including the cell probe model of Yao [12]. The cell probe model is one of the least restrictive lower bound models, thus lower bounds proved in the cell probe model apply to essentially every imaginable data structure, including those developed in the popular upper bound model, the word RAM. Unfortunately this generality comes at a cost: The highest lower bound that has been proved for any explicit data structure problem is $\Omega(\lg n)$, both for static and even dynamic data structures[1].

In this paper, we break this barrier by introducing a new technique for proving dynamic cell probe lower bounds. Using our technique, we obtain a query time lower bound of $\Omega((\lg n/\lg\lg n)^2)$ for any polylogarithmic update time. We prove the bound for the fundamental problem of dynamic *weighted orthogonal range counting* in two-dimensional space. In dynamic weighted orthogonal range counting (in 2-d), the goal is to maintain a set of (2-d) points under insertions, where each point is assigned an integer weight. In addition to supporting insertions, a data structure must support answering queries. A query is specified by a query point $q = (x, y)$, and the data structure must return the sum of the weights assigned to the points *dominated* by q. Here we say that a point (x', y') is dominated by q if $x' \leq x$ and $y' \leq y$.

1.1 The Cell Probe Model

A dynamic data structure in the cell probe model consists of a set of memory cells, each storing w bits. Each cell of the data structure is identified by an integer address, which is assumed to fit in w bits, i.e. each address is amongst $[2^w] = \{0, \ldots, 2^w - 1\}$. We will make the additional standard assumption that a cell also has enough bits to address any update operation performed on it, i.e. we assume $w = \Omega(\lg n)$ when analysing a data structures performance on a sequence of n updates.

When presented with an update operation, a data structure reads and updates a number of the stored cells to reflect the changes. We refer to the reading or writing of a cell as probing the cell, hence the name cell probe model. The update time of a data structure is the number of cells probed when processing an update operation.

To answer a query, a data structure similarly probes a

[1]This is true under the most natural assumption of cell size $\Theta(\lg n)$.

number of cells from the data structure, and from the contents of the probed cells, the data structure must return the correct answer to the query. We similarly define the query time of a data structure as the number of cells probed when answering a query.

Previous Results.

In the following, we give a brief overview of the most important techniques that have been introduced for proving cell probe lower bounds for dynamic data structures. We also review the previous cell probe lower bounds obtained for orthogonal range counting and related problems. In Section 2 we then give a more thorough review of the previous techniques most relevant to our work, followed by a description of the key ideas in our new technique.

In their seminal paper [2], Fredman and Saks introduced the celebrated chronogram technique. They applied their technique to the *partial sums* problem and obtained a lower bound stating that $t_q = \Omega(\lg n/\lg(wt_u))$, where t_q is the query time and t_u the update time. In the partial sums problem, we are to maintain an array of n $O(w)$-bit integers under updates of the entries. A query to the problem consists of two indices i and j, and the goal is to compute the sum of the integers in the subarray from index i to j. The lower bound of Fredman and Saks holds even when the data structure is allowed amortization and randomization.

The bounds of Fredman and Saks remained the highest achieved until the breakthrough results of Pătraşcu and Demaine [9]. In their paper, they extended upon the ideas of Fredman and Saks to give a tight lower bound for the partial sums problem. Their results state that $t_q \lg(t_u/t_q) = \Omega(\lg n)$ and $t_u \lg(t_q/t_u) = \Omega(\lg n)$ when the integers have $\Omega(w)$ bits, which in particular implies $\max\{t_q, t_u\} = \Omega(\lg n)$. We note that they also obtain tight lower bounds in the regime of smaller integers. Again, the bounds hold even when allowed amortization and randomization. For the most natural cell size of $w = \Theta(\lg n)$, this remains until today the highest achieved lower bound.

The two above techniques both lead to smooth tradeoff curves between update time and query time. While this behaviour is correct for the partial sums problem, there are many examples where this is certainly not the case. Pătraşcu and Thorup [10] recently presented a new extension of the chronogram technique, which can prove strong threshold lower bounds. In particular they showed that any data structure for maintaining the connectivity of a graph under edge insertions and deletions, must either have amortized update time $\Omega(\lg n)$ or the query time explodes to $n^{1-o(1)}$.

In the search for super-logarithmic lower bounds, Pătraşcu introduced a dynamic set-disjointness problem named the multiphase problem [8]. Based on a widely believed conjecture about the hardness of 3-SUM, Pătraşcu first reduced 3-SUM to the multiphase problem and then gave a series of reductions to different dynamic data structure problems, implying polynomial lower bounds under the 3-SUM conjecture.

Finally, we mention that Pătraşcu [6] presented a technique capable of proving a query time lower bound of $t_q = \Omega((\lg n/\lg(wt_u))^2)$ for dynamic weighted orthogonal range counting, but only when the weights are $\lg^{2+\Omega(1)} n$-bit integers. Thus the magnitude of the lower bound compared to the number of bits, δ, needed to describe an update operation or a query, remains below $\Omega(\delta)$. This bound holds when t_u is the worst case update time and t_q the expected average[2] query time of a data structure.

The particular problem of orthogonal range counting has received much attention from a lower bound perspective. In the static case, Pătraşcu [6] first proved a lower bound of $t = \Omega(\lg n/\lg(Sw/n))$ where t is the expected average query time and S the space of the data structure in number of cells. This lower bound holds for regular counting (without weights), and even when just the parity of the number of points in the range is to be returned. In [7] he reproved this bound using an elegant reduction from the communication game known as lop-sided set disjointness. Subsequently Jørgensen and Larsen [3] proved a matching bound for the strongly related problems of range selection and range median. Finally, as mentioned earlier, Pătraşcu [6] proved a $t_q = \Omega((\lg n/\lg(wt_u))^2)$ lower bound for dynamic weighted orthogonal range counting when the weights are $\lg^{2+\Omega(1)} n$-bit integers. In the concluding remarks of that paper, he posed it as an interesting open problem to prove the same lower bound for regular counting.

Our Results.

In this paper we introduce a new technique for proving dynamic cell probe lower bounds. Using this technique, we obtain a lower bound of $t_q = \Omega((\lg n/\lg(wt_u))^2)$, where t_u is the worst case update time and t_q is the expected average query time of the data structure. Our lower bound holds for any cell size $w = \Omega(\lg n)$, and is the highest achieved to date in the most natural setting of cell size $w = \Theta(\lg n)$. For polylogarithmic t_u and logarithmic cell size, this bound is $t_q = \Omega((\lg n/\lg\lg n)^2)$, i.e. almost a quadratic improvement over the highest previous lower bound of Pătraşcu and Demaine.

We prove our lower bound for dynamic weighted orthogonal range counting in two-dimensional space, where the weights are $\Theta(\lg n)$-bit integers. This gives a partial answer to the open problem posed by Pătraşcu by reducing the requirement of the magnitude of weights from $\lg^{2+\Omega(1)} n$ to just logarithmic. Finally, our lower bound is also tight for update time $t_u = \lg^{2+\Omega(1)} n$, hence deepening our understanding of one of the most fundamental range searching problems.

Overview.

In Section 2 we discuss the two previous techniques most related to ours, i.e. that of Fredman and Saks [2] and of Pătraşcu [6]. Following this discussion we give a description of the key ideas behind our new technique. Having introduced our technique, we proceed to the lower bound proof in Section 3 and finally we conclude in Section 5 with a discussion of the limitations of our technique and the intriguing open problems these limitations pose.

2. TECHNIQUES

In this section, we first review the two previous techniques most important to our work, and then present our new technique.

[2]i.e. for any data structure with a possibly randomized query algorithm, there exists a sequence of updates U, such that the expected cost of answering a uniform random query after the updates U is t_q.

Fredman and Saks [2].

This technique is known as the chronogram technique. The basic idea is to consider batches, or *epochs*, of updates to a data structure problem. More formally, one defines an epoch i for each $i = 1, \ldots, \lg_\beta n$, where $\beta > 1$ is a parameter. The i'th epoch consists of performing β^i carefully chosen updates to a data structure. The epochs occur in time from largest to smallest epoch, and at the end of epoch 1, every cell of the constructed data structure is associated to the epoch in which it was last updated. The goal is to argue that to answer a query after epoch 1, the query algorithm has to probe one cell associated to each epoch. Since a cell is only associated to one epoch, this gives a total query time lower bound of $\Omega(\lg_\beta n)$.

Arguing that the query algorithm must probe one cell associated to each epoch is typically done by setting β somewhat larger than the worst case update time t_u. Since cells associated to an epoch j cannot contain useful information about an epoch $i < j$ (the updates of epoch j where performed before knowing what the updates of epoch i was), one can ignore cells associated to previous epochs when analysing the probes to an epoch i. Similarly, since all epochs following epoch i (future updates) writes a total of $O(\beta^{i-1} t_u) = o(\beta^i)$ cells, these cells do not contain enough information about the β^i updates of epoch i to be of any use. Thus if the answer to a query depends on an update of epoch i, then the query algorithm must probe a cell associated to epoch i to answer the query.

We note that Fredman and Saks also defined the notion of epochs over a sequence of intermixed updates and queries. Here the epochs are defined relative to each query, and from this approach they obtain their amortized bounds.

Pătraşcu [6].

This technique uses the same setup as the chronogram technique, i.e. one considers epochs $i = 1, \ldots, \lg_\beta n$ of updates, followed by one query. The idea is to use a static $\Omega(\lg_\beta n)$ lower bound proof to argue that the query algorithm must probe $\Omega(\lg_\beta n)$ cells from each epoch, and not just one. Summing over all epochs, this gives a lower bound of $\Omega(\lg^2_\beta n)$. In the following, we give a coarse overview of the general framework for doing so.

One first proves a lower bound on the amount of communication in the following (static) communication game (for every epoch i): Bob receives all epochs of updates to the dynamic data structure problem and Alice receives a set of queries and all updates of the epochs preceding epoch i. The goal for them is to compute the answer to Alice's queries after all the epochs of updates.

When such a lower bound has been established, one considers each epoch i in turn and uses the dynamic data structure to obtain an efficient protocol for the above communication game between Alice and Bob. The key idea is to let Alice simulate the query algorithm of the dynamic data structure on each of her queries, and whenever a cell associated to epoch i is requested, she asks Bob for the contents. Bob replies and she continues the simulation. Clearly the amount of communication is proportional to the number of probes to cells associated to epoch i, and thus a lower bound follows from the communication game lower bound. The main difficulty in implementing this protocol is that Alice must somehow recover the contents of the cells not associated to epoch i without asking Bob for it. This is accomplished by first letting Bob send all cells associated to epochs $j < i$ to Alice. For sufficiently large β, this does not break the communication lower bound. Now Alice can execute the updates of the epochs preceding epoch i (epochs $j > i$) herself, and she knows the cells associated to epochs $j < i$, thus all she needs to realize the protocol is a way to recognize whether a cell requested belongs to epoch i or not. If not, she has the contents herself, and if it does, she asks Bob for the contents. This last step is solved by changing to a non-deterministic communication game and letting an all-powerful prover send a Bloom filter to Alice, specifying the cells that Alice will probe from epoch i. This last step is expensive, costing at least one bit for each of the t_q cells probed by each of Alice's queries. This is the reason why the lower bound of Pătraşcu requires large weights assigned to the input points (by having larger weights, the lower bound on the communication game increases since knowing the answer to a query reveals more information).

Our Technique.

Our new technique elegantly circumvents the limitations of Pătraşcu's technique by exploiting recent ideas by Panigrahy et al. [5] for proving static lower bounds. The basic setup is the same, i.e. we consider epochs $i = 1, \ldots, \lg_\beta n$, where the i'th epoch consists of β^i updates. As with the two previous techniques, we associate a cell to the epoch in which it was last updates. Lower bounds now follow by showing that any data structure must probe $\Omega(\lg_\beta n)$ cells associated to each epoch i when answering a query at the end of epoch 1. Summing over all $\lg_\beta n$ epochs, this gives us a lower bound of $\Omega(\lg^2_\beta n)$.

To show that $\Omega(\lg_\beta n)$ probes to cells associated to an epoch i are required, we assume for contradiction that a data structure probing $o(\lg_\beta n)$ cells associated to epoch i exists. Using this data structure, we then consider a game between an encoder and a decoder. The encoder receives as input the updates of all epochs, and must from this send a message to the decoder. The decoder then sees this message and all updates preceding epoch i and must from this uniquely recover the updates of epoch i. If the message is smaller than the entropy of the updates of epoch i (conditioned on preceding epochs), this gives an information theoretic contradiction. The trick is to find a way for the encoder to exploit the small number of probed cells to send a short message.

As mentioned, we use the ideas in [5] to exploit the small number of probes. In [5] it was observed that if S is a set of cells, and if the query algorithm of a data structure probes $o(\lg_\beta n)$ cells from S on average over all queries (for large enough β), then there is a subset of cells $S' \subseteq S$ which *resolves* a large number of queries. Here we say that a subset of cells $S' \subseteq S$ resolves a query, if the query algorithm probes no cells in $S \setminus S'$ when answering that query. What this observation gives us compared to the approach of Pătraşcu, is that we can find a large set of queries that are all resolved by the same small subset of cells associated to an epoch i. Thus when amortizing over all the queries resolved by the set S', we no longer have to use t_q bits per query just to specify the cells it probes.

With this observation in mind, the encoder proceeds as follows: First he executes all the updates of all epochs on the claimed data structure. He then sends all cells associated to epochs $j < i$. For large enough β, this message is

smaller than the entropy of the β^i updates of epoch i. Letting S_i denote the cells associated to epoch i, the decoder then finds a subset of cells $S'_i \subseteq S_i$, such that a large number of queries are resolved by S'_i. He then sends a description of those cells and proceeds by finding a subset Q of the queries resolved by S'_i, such that knowing the answer to all queries in Q reduces the entropy of the updates of epoch i by more than the number of bits needed to describe S'_i, Q and the cells associated to epochs $j < i$. He then sends a description of Q followed by an encoding of the updates of epoch i, conditioned on the answers to queries in Q. Since the entropy of the updates of epoch i is reduced by more bits than was already send, this gives our contradiction (if the decoder can recover the updates from the above messages).

To recover the updates of epoch i, the decoder first executes the updates preceding epoch i. His goal is to simulate the query algorithm for every query in Q to recover all the answers. He achieves this in the following way: For each cell c requested when answering a query $q \in Q$, he examines the cells associated to epochs $j < i$ (those cells were send by the encoder), and if c is contained in one of those he immediately recovers the contents. If not, he proceeds by examining the set S'_i. If c is included in this set, he has again recovered the contents and can continue the simulation. Finally, if c is not in S'_i, then c must be associated to an epoch preceding epoch i (since queries in Q probe no cells in $S_i \setminus S'_i$), thus the decoder recovers the contents of c from the updates that he executed initially. In this manner, the decoder can recover the answer to every query in Q, and from the last part of the message he recovers the updates of epoch i.

The main technical challenge in using our technique lies in arguing that if $o(\lg_\beta n)$ cells are probed amongst the cells associated to epoch i, then the claimed cell set S'_i and query set Q exists. In the next section, we demonstrate the technique on a concrete problem by proving our lower bound for dynamic weighted orthogonal range counting.

3. DYNAMIC LOWER BOUND

In this section we prove our main result, which we have formulated in the following theorem:

THEOREM 1. *Any data structure for dynamic weighted orthogonal range counting in the cell probe model, must satisfy $t_q = \Omega((\lg n/\lg(wt_u))^2)$. Here t_q is the expected average query time and t_u the worst case update time. This lower bound holds when the weights of the inserted points are $\Theta(\lg n)$-bit integers.*

We prove Theorem 1 by devising a hard distribution over updates, followed by one uniform random query. We then lower bound the expected cost (over the distribution) of answering the query for any *deterministic* data structure with worst case update time t_u. By Yao's principle [11], this translates into a lower bound on the expected average query time of a possibly randomized data structure. In our proof we assume the weights are $4\lg n$-bit integers and note that our lower bound applies to any $\varepsilon \lg n$-bit weights, where $\varepsilon > 0$ is an arbitrarily small constant, simply because a data structure for $\varepsilon \lg n$-bit integer weights can be used to solve the problem for any $O(\lg n)$-bit integer weights with a constant factor overhead by dividing the bits of the weights into $\lceil \delta/(\varepsilon \lg n) \rceil = O(1)$ chunks and maintaining a data structure for each chunk. We begin our proof by presenting our hard distribution over updates and queries.

Hard Distribution.

Our updates arrive in batches, or *epochs*, of exponentially decreasing size. For $i = 1, \ldots, \lg_\beta n$ we define epoch i as a sequence of β^i updates, for a parameter $\beta > 1$ to be fixed later. The epochs occur in time from biggest to smallest epoch, and at the end of epoch 1 we execute a uniform random query in $[n] \times [n]$.

What remains is to specify which updates are performed in each epoch i. The updates of epoch i are chosen to mimic the hard input distribution for static orthogonal range counting on a set of β^i points. We first define the following point set known as the Fibonacci lattice:

DEFINITION 1 ([4]). *The Fibonacci lattice F_m is the set of m two-dimensional points defined by $F_m = \{(i, if_{k-1} \bmod m) \mid i = 0, \ldots, m-1\}$, where $m = f_k$ is the k'th Fibonacci number.*

The β^i updates of epoch i now consists of inserting each point of the Fibonacci lattice F_{β^i}, but scaled to fit the input region $[n] \times [n]$, i.e. the j'th update of epoch i inserts the point with coordinates $(n/\beta^i \cdot j, n/\beta^i \cdot (jf_{k_i-1} \bmod \beta^i))$, for $j = 0, \ldots, \beta^i$. The weight of each inserted point is a uniform random integer amongst $[\Delta]$, where Δ is the largest prime number smaller than $2^{4\lg n} = n^4$. This concludes the description of our hard distribution.

The Fibonacci lattice has the desirable property that it is very uniform. This plays an important role in our lower bound proof, and we have formulated this property in the following lemma:

LEMMA 1 ([1]). *For the Fibonacci lattice F_{β^i}, where the coordinates of each point have been multiplied by n/β^i, and for $\alpha > 0$, any axis-aligned rectangle in $[0, n - n/\beta^i] \times [0, n - n/\beta^i]$ with area $\alpha n^2/\beta^i$ contains between $\lfloor \alpha/a_1 \rfloor$ and $\lceil \alpha/a_2 \rceil$ points, where $a_1 \approx 1.9$ and $a_2 \approx 0.45$.*

Note that we assume each β^i to be a Fibonacci number (denoted f_{k_i}), and that each β^i divides n. These assumptions can easily be removed by fiddling with the constants, but this would only clutter the exposition.

For the remainder of the paper, we let $\mathbf{U}_i$ denote the random variable giving the sequence of updates in epoch i, and we let $\mathbf{U} = \mathbf{U}_{\lg_\beta n} \cdots \mathbf{U}_1$ denote the random variable giving all updates of all $\lg_\beta n$ epochs. Finally, we let $\mathbf{q}$ be the random variable giving the query.

A Chronogram Approach.

Having defined our hard distribution over updates and queries, we now give a high-level proof of our lower bound. Assume a deterministic data structure solution exists with worst case update time t_u. From this data structure and a sequence of updates $U = U_{\lg_\beta n}, \ldots, U_1$, where each U_j is a possible outcome of $\mathbf{U}_j$, we define $S(U)$ to be the set of cells stored in the data structure after executing the updates U. Now associate each cell in $S(U)$ to the last epoch in which its contents were updated, and let $S_i(U)$ denote the subset of $S(U)$ associated to epoch i for $i = 1, \ldots, \lg_\beta n$. Also let $t_i(U, q)$ denote the number of cells in $S_i(U)$ probed by the query algorithm of the data structure when answering the query $q \in [n] \times [n]$ after the sequence of updates U. Finally, let $t_i(U)$ denote the average cost of answering a query $q \in [n] \times [n]$ after the sequence of updates U, i.e. let $t_i(U) = \sum_{q \in [n] \times [n]} t_i(U, q)/n^2$. Then the following holds:

LEMMA 2. *If $\beta = (wt_u)^9$, then $\mathbb{E}[t_i(\boldsymbol{U}, \boldsymbol{q})] = \Omega(\lg_\beta n)$ for all $i \geq \frac{15}{16}\lg_\beta n$.*

Before giving the proof of Lemma 2, we show that it implies Theorem 1: Let β be as in Lemma 2. Since the cell sets $S_{\lg_\beta n}(\mathbf{U}), \ldots, S_1(\mathbf{U})$ are disjoint, we get that the number of cells probed when answering the query $\mathbf{q}$ is $\sum_i t_i(\mathbf{U}, \mathbf{q})$. It now follows immediately from linearity of expectation that the expected number of cells probed when answering $\mathbf{q}$ is $\Omega(\lg_\beta n \cdot \lg_\beta n) = \Omega((\lg n/\lg(wt_u))^2)$, which completes the proof of Theorem 1.

The hard part thus lies in proving Lemma 2, i.e. in showing that the random query must probe a lot of cells associated to each of the epochs $i = \frac{15}{16}\lg_\beta n, \ldots, \lg_\beta n$. This will be the focus of the next section.

3.1 Bounding the Probes to Epoch i

As also pointed out in Section 2, we prove Lemma 2 using an encoding argument. Assume for contradiction that there exists a data structure solution such that under our hard distribution, with $\beta = (wt_u)^9$, there exists an epoch $i^* \geq \frac{15}{16}\lg_\beta n$, such that the claimed data structure satisfies $\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})] = o(\lg_\beta n)$.

First observe that $\mathbf{U}_{i^*}$ is independent of $\mathbf{U}_{\lg_\beta n} \cdots \mathbf{U}_{i^*+1}$, i.e. $H(\mathbf{U}_{i^*} \mid \mathbf{U}_{\lg_\beta n} \cdots \mathbf{U}_{i^*+1}) = H(\mathbf{U}_{i^*})$, where $H(\cdot)$ denotes binary entropy. Furthermore, we have $H(\mathbf{U}_{i^*}) = \beta^{i^*}\lg\Delta$, since the updates of epoch i^* consists of inserting β^{i^*} fixed points, each with a uniform random weight amongst the integers $[\Delta]$. Our goal is to show that, conditioned on $\mathbf{U}_{\lg_\beta n} \cdots \mathbf{U}_{i^*+1}$, we can use the claimed data structure solution to encode $\mathbf{U}_{i^*}$ in less than $H(\mathbf{U}_{i^*})$ bits in expectation, i.e. a contradiction. We view this encoding step as a game between an encoder and a decoder. The encoder receives as input the sequence of updates $\mathbf{U} = \mathbf{U}_{\lg_\beta n}, \ldots, \mathbf{U}_1$. The encoder now examines these updates and from them sends a message to the decoder (an encoding). The decoder now sees this message, as well as $\mathbf{U}_{\lg_\beta n}, \ldots, \mathbf{U}_{i^*+1}$ (we conditioned on these variables), and must from this uniquely recover $\mathbf{U}_{i^*}$. If we can design a procedure for constructing and decoding the encoder's message, such that the expected size of the message is less than $H(\mathbf{U}_{i^*}) = \beta^{i^*}\lg\Delta$ bits, then we have reached our contradiction.

Before presenting our encoding and decoding procedures, we show exactly what breaks down if the claimed data structure probes too few cells from epoch i^*. For this, we first introduce some terminology. For a query point $q = (x, y) \in [n] \times [n]$, we define for every epoch $i = 1, \ldots, \lg_\beta n$ the *incidence vector* $\chi_i(q)$, as a vector in $\{0,1\}^{\beta^i}$. The j'th coordinate of $\chi_i(q)$ is 1 if the j'th point inserted in epoch i is dominated by q, and 0 otherwise. More formally, for a query $q = (x, y)$, the j'th coordinate $\chi_i(q)_j$ is given by:

$$\chi_i(q)_j = \begin{cases} 1 & \text{if } jn/\beta^i \leq x \wedge (jf_{k_i-1} \bmod \beta^i)n/\beta^i \leq y \\ 0 & \text{otherwise} \end{cases}$$

Similarly, we define for a fixed sequence of updates U_i, where U_i is a possible outcome of $\mathbf{U}_i$, the β^i-dimensional vector u_i for which the j'th coordinate equals the weight assigned to the j'th inserted point in U_i. We note that U_i and u_i uniquely specify each other, since all possible outcomes of $\mathbf{U}_i$ inserts the same fixed points, only the weights vary.

Finally observe that the answer to a query q after a fixed sequence of updates $U_{\lg_\beta n}, \ldots, U_1$ is $\sum_{i=1}^{\lg_\beta n}\langle\chi_i(q), u_i\rangle$, where $\langle\cdot,\cdot\rangle$ denotes the standard inner product. With these definitions, we now present the main result forcing a data structure to probe many cells from each epoch:

LEMMA 3. *Let $i \geq \frac{15}{16}\lg_\beta n$ be an epoch, and let $U = U_{\lg_\beta n}, \ldots, U_1$ be a fixed sequence of updates, where each U_j is a possible outcome of $\mathbf{U}_j$. If $t_i(U) = o(\lg_\beta n)$, then there exists a subset of cells $C_i \subseteq S_i(U)$ and a set of query points $Q \subseteq [n] \times [n]$ such that:*

1. *$|C_i| = O(\beta^{i-1}w)$.*
2. *$|Q| = \Omega(\beta^{i-3/4})$.*
3. *The set of incidence vectors $\chi_i(Q) = \{\chi_i(q) \mid q \in Q\}$ is a linearly independent set of vectors in $[\Delta]^{\beta^i}$.*
4. *The query algorithm of the data structure solution probes no cells in $S_i(U) \setminus C_i$ when answering a query $q \in Q$ after the sequence of updates U.*

The contradiction that this lemma intuitively gives us, is that the queries in Q reveal more information about U_i than the bits in C_i can describe. We note that Lemma 3 essentially is a generalization of the results proved in the static range counting papers [6, 3], simply phrased in terms of cell subsets answering many queries instead of communication complexity. Since the proof contains only few new ideas, we have deferred it to Section 4 and instead move on to show how we use this lemma in our encoding and decoding procedures.

Encoding.

Let $U = U_{\lg_\beta n}, \ldots, U_1$ be a fixed sequence of updates given as input to the encoder, where each U_j is a possible outcome of $\mathbf{U}_j$. Also let $i^* \geq \frac{15}{16}\lg_\beta n$ be the epoch for which $\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})] = o(\lg_\beta n)$. We construct the message of the encoder by the following procedure:

1. First the encoder executes the sequence of updates U on the claimed data structure, and from this obtains the sets $S_{\lg_\beta n}(U), \ldots, S_1(U)$. He then simulates the query algorithm on the data structure for every query $q \in [n] \times [n]$. From this, the encoder computes $t_{i^*}(U)$ (just the average number of cells in $S_{i^*}(U)$ that are probed).
2. If $t_{i^*}(U) > 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})]$, then the encoder writes a 1-bit, followed by $\lceil\beta^{i^*}\lg\Delta\rceil = H(\mathbf{U}_{i^*}) + O(1)$ bits, simply specifying each weight assigned to a point in U_{i^*} (this can be done in the claimed amount of bits by interpreting the weights as one big integer in $[\Delta^{\beta^{i^*}}]$). This is the complete message send to the decoder when $t_{i^*}(U) > 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})]$.
3. If $t_{i^*}(U) \leq 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})]$, then the encoder first writes a 0-bit. Now since $t_{i^*}(U) \leq 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})] = o(\lg_\beta n)$, we get from Lemma 3 that there must exist a set of cells $C_{i^*} \subseteq S_{i^*}(U)$ and a set of queries $Q \subseteq [n] \times [n]$ satisfying 1-4 in Lemma 3. The encoder finds such sets C_{i^*} and Q simply by trying all possible sets in some arbitrary but fixed order (given two candidate sets C'_{i^*} and Q' it is straight forward to verify whether they satisfy properties 1-4 of Lemma 3). The encoder now writes down these two sets, including addresses

and contents of the cells in C_{i^*}, for a total of at most $O(w) + 2|C_{i^*}|w + \lg\binom{n^2}{|Q|}$ bits (the $O(w)$ bits specifies $|C_{i^*}|$ and $|Q|$).

4. The encoder now constructs a set X, such that $X = \chi_{i^*}(Q)$ initially. Then he iterates through all vectors in $\{0,1\}^{\beta^{i^*}}$, in some arbitrary but fixed order, and for each such vector x, checks whether x is in span(X). If not, the encoder adds x to X. This process continues until $\dim(\text{span}(X)) = \beta^{i^*}$, at which point the encoder computes and writes down $(\langle x, u_{i^*}\rangle \mod \Delta)$ for each x that was added to X. Since $\dim(\text{span}(\chi_{i^*}(Q))) = |Q|$ (by point 3 in Lemma 3), this adds a total of $\lceil(\beta^{i^*} - |Q|)\lg\Delta\rceil$ bits to the message (by interpreting the values as one big integer in $[\Delta^{\beta^{i^*}-|Q|}]$).

5. Finally, the encoder writes down all of the cell sets $S_{i^*-1}(U), \ldots, S_1(U)$, including addresses and contents, plus all of the vectors $u_{i^*-1}, \ldots, u_1$. This takes at most $\sum_{j=1}^{i^*-1}(2|S_j(U)|w + \beta^j \lg\Delta + O(w))$ bits. When this is done, the encoder sends the constructed message to the decoder.

Before analysing the expected size of the encoding, we present the decoding procedure.

Decoding.

In the following, we describe our decoding procedure. The decoder receives as input the updates $U_{\lg_\beta n}, \ldots, U_{i^*+1}$ and the message from the encoder. The decoder now recovers U_{i^*} by the following procedure:

1. The decoder examines the first bit of the message. If this bit is 1, then the decoder immediately recovers U_{i^*} from the encoding (step 2 in the encoding procedure). If not, the decoder instead executes the updates $U_{\lg_\beta n} \cdots U_{i^*+1}$ on the claimed data structure solution and obtains the cells sets $S^{i^*+1}_{\lg_\beta n}(U), \ldots, S^{i^*+1}_{i^*+1}(U)$ where $S^{i^*+1}_j(U)$ contains the cells that were last updated during epoch j when executing updates $U_{\lg_\beta n}, \ldots, U_{i^*+1}$ (and not the entire sequence of updates $U_{\lg_\beta n}, \ldots, U_1$).

2. The decoder now recovers $Q, C_{i^*}, S_{i^*-1}(U), \ldots, S_1(U)$ and $u_{i^*-1}, \ldots, u_1$ from the encoding. For each query $q \in Q$, the decoder then computes the answer to q as if all updates $U_{\lg_\beta n}, \ldots, U_1$ had been performed. The decoder accomplishes this by simulating the query algorithm on q, and for each cell requested, the decoder recovers the contents of that cell as it would have been if all updates $U_{\lg_\beta n}, \ldots, U_1$ had been performed. This is done in the following way: When the query algorithm requests a cell c, the decoder first determines whether c is in one of the sets $S_{i^*-1}(U), \ldots, S_1(U)$. If so, the correct contents of c (the contents after the updates $U = U_{\lg_\beta n}, \ldots, U_1$) is directly recovered. If c is not amongst these cells, the decoder checks whether c is in C_{i^*}. If so, we have again recovered the contents. Finally, if c is not in C_{i^*}, then from point 4 of Lemma 3, we get that c is not in $S_{i^*}(U)$. Since c is not in any of $S_{i^*}(U), \ldots, S_1(U)$, this means that the contents of c has not changed during the updates $U_{i^*}, \ldots, U_1$, and thus the decoder finally recovers the contents of c from $S^{i^*+1}_{\lg_\beta n}(U), \ldots, S^{i^*+1}_{i^*+1}(U)$. The decoder can therefore recover the answer to each query q in Q if it had been executed after the sequence of updates $U_{\lg_\beta n}, \ldots, U_1$, i.e. for all $q \in Q$, he knows $\sum_{i=1}^{\lg_\beta n}\langle\chi_i(q), u_i\rangle$.

3. The next decoding step consists of computing for each query q in Q, the value $\langle\chi_{i^*}(q), u_{i^*}\rangle$. For each $q \in Q$, the decoder already knows the value $\sum_{i=1}^{\lg_\beta n}\langle\chi_i(q), u_i\rangle$ from the above. From the encoding of $u_{i^*-1}, \ldots, u_1$, the decoder can compute the value $\sum_{i=1}^{i^*-1}\langle\chi_i(q), u_i\rangle$ and finally from $U_{\lg_\beta n}, \ldots, U_{i^*+1}$ the decoder computes $\sum_{i=i^*+1}^{\lg_\beta n}\langle\chi_i(q), u_i\rangle$. The decoder can now recover the value $\langle\chi_{i^*}(q), u_{i^*}\rangle$ simply by observing that $\langle\chi_{i^*}(q), u_{i^*}\rangle = \sum_{i=1}^{\lg_\beta n}\langle\chi_i(q), u_i\rangle - \sum_{i\neq i^*}\langle\chi_i(q), u_i\rangle$.

4. Now from the query set Q, the decoder construct the set of vectors $X = \chi_{i^*}(Q)$, and then iterates through all vectors in $\{0,1\}^{\beta^{i^*}}$, in the same fixed order as the encoder. For each such vector x, the decoder again verifies whether x is in span(X), and if not, adds x to X and recovers $\langle x, u_{i^*}\rangle \mod \Delta$ from the encoding. The decoder now constructs the $\beta^{i^*} \times \beta^{i^*}$ matrix A, having the vectors in X as rows. Similarly, he construct the vector $\mathbf{z}$ having one coordinate for each row of A. The coordinate of $\mathbf{z}$ corresponding to a row vector x, has the value $\langle x, u_{i^*}\rangle \mod \Delta$. Note that this value is already known to the decoder, regardless of whether x is also in $\chi_{i^*}(Q)$ (simply taking modulo Δ on the value $\langle\chi_{i^*}(x), u_{i^*}\rangle$ computed above for each vector in $\chi_{i^*}(Q)$), or was added later. Since A has full rank, and since the set $[\Delta]$ endowed with integer addition and multiplication modulo Δ is a finite field, it follows that the system of equations $A \otimes \mathbf{y} = \mathbf{z}$ has a unique solution $\mathbf{y} \in [\Delta]^{\beta^{i^*}}$ (here $\otimes$ denotes matrix-vector multiplication modulo Δ). But $u_{i^*} \in [\Delta]^{\beta^{i^*}}$ and $A \otimes u_{i^*} = \mathbf{z}$, thus the decoder now solves the linear system of equations $A \otimes \mathbf{y} = \mathbf{z}$ and uniquely recovers u_{i^*}, and therefore also U_{i^*}. This completes the decoding procedure.

Analysis.

We now analyse the expected size of the encoding of $\mathbf{U}_{i^*}$. We first analyse the size of the encoding when $t_{i^*}(\mathbf{U}) \leq 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})]$. In this case, the encoder sends a message of

$$2|C_{i^*}|w + \lg\binom{n^2}{|Q|} + (\beta^{i^*} - |Q|)\lg\Delta$$

$$+O(w\lg_\beta n) + \sum_{j=1}^{i^*-1}(2|S_j(\mathbf{U})|w + \beta^j\lg\Delta)$$

bits. Since $\beta^{i^*}\lg\Delta = H(\mathbf{U}_{i^*})$ and $|C_{i^*}|w = o(|Q|)$, the above is upper bounded by

$$H(\mathbf{U}_{i^*}) - |Q|\lg(\Delta/n^2) + o(|Q|) + \sum_{j=1}^{i^*-1}(2|S_j(\mathbf{U})|w + \beta^j\lg\Delta).$$

Since $\beta \geq 2$, we also have $\sum_{j=1}^{i^*-1}\beta^j\lg\Delta < 2\beta^{i^*-1}\lg\Delta = o(|Q|\lg\Delta)$. Similarly, we have $|S_j(\mathbf{U})| \leq \beta^j t_u$, which gives

us $\sum_{j=1}^{i^*-1} 2|S_j(\mathbf{U})|w < 4\beta^{i^*-1}wt_u = o(|Q|)$. From standard results on prime numbers, we have that the largest prime number smaller than n^4 is at least n^3 for infinitely many n, i.e. we can assume $\lg(\Delta/n^2) = \Omega(\lg \Delta)$. Therefore, the above is again upper bounded by

$$H(\mathbf{U}_{i^*}) - \Omega(|Q| \lg \Delta) = H(\mathbf{U}_{i^*}) - \Omega(\beta^{i^*-3/4} \lg \Delta).$$

This part thus contributes at most

$$\Pr[t_{i^*}(\mathbf{U}) \leq 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})]] \cdot (H(\mathbf{U}_{i^*}) - \Omega(\beta^{i^*-3/4} \lg \Delta))$$

bits to the expected size of the encoding. The case where $t_{i^*}(\mathbf{U}) > 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})]$ similarly contributes $\Pr[t_{i^*}(\mathbf{U}) > 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})]] \cdot (H(\mathbf{U}_{i^*}) + O(1))$ bits to the expected size of the encoding. Now since $\mathbf{q}$ is uniform, we have $\mathbb{E}[t_{i^*}(\mathbf{U})] = \mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})]$, we therefore get from Markov's inequality that $\Pr[t_{i^*}(\mathbf{U}) > 2\mathbb{E}[t_{i^*}(\mathbf{U}, \mathbf{q})]] < \frac{1}{2}$. Therefore the expected size of our encoding is upper bounded by $O(1) + \frac{1}{2}H(\mathbf{U}_{i^*}) + \frac{1}{2}(H(\mathbf{U}_{i^*}) - \Omega(\beta^{i^*-3/4} \lg \Delta)) < H(\mathbf{U}_{i^*})$ finally leading to our contradiction and completing the proof of Lemma 2.

4. THE STATIC SETUP

Finally, in this section we prove Lemma 3, the last piece in our lower bound proof. As already mentioned, we prove the lemma by extending on previous ideas for proving lower bounds on static range counting. We note that we have chosen a more geometric (and we believe more intuitive) approach to the proof than the previous papers.

For the remainder of the section, we let $U = U_{\lg_\beta n}, \ldots, U_1$ be a fixed sequence of updates, where each U_j is a possible outcome of $\mathbf{U}_j$, and $i \geq \frac{15}{16} \lg_\beta n$ an epoch. Furthermore, we assume that the claimed data structure satisfies $t_i(U) = o(\lg_\beta n)$, and our task is to show that the claimed cell set C_i and query set Q exists.

Our first step is to find a geometric property of a set of queries Q, such that $\chi_i(Q)$ is a linearly independent set of vectors. One property that ensures this, is that the queries in Q are sufficiently *well spread*. To make this more formal, we introduce the following terminology:

A *grid* G with *width* $\mu \geq 1$ and *height* $\gamma \geq 1$, is the collection of *grid cells* $[j\mu, (j+1)\mu) \times [h\gamma, (h+1)\gamma)$ such that $0 \leq j < n/\mu$ and $0 \leq h < n/\gamma$. We say that a query point $q = (x, y) \in [n] \times [n]$ *hits* a grid cell $[j\mu, (j+1)\mu) \times [h\gamma, (h+1)\gamma)$ of G, if the point (x, y) lies within that grid cell, i.e. if $j\mu \leq x < (j+1)\mu$ and $h\gamma \leq y < (h+1)\gamma$. Finally, we define the *hitting number* of a set of queries Q' on a grid G, as the number of distinct grid cells in G that is hit by a query in Q'.

With this terminology, we have the following lemma:

LEMMA 4. *Let Q' be a set of queries and G a grid with width μ and height $n^2/\beta^i\mu$ for some parameter $n/\beta^i \leq \mu \leq n$. Let h denote the hitting number of Q' on G. Then there is a subset of queries $Q \subseteq Q'$, such that $|Q| = \Omega(h - 6n/\mu - 6\mu\beta^i/n)$ and $\chi_i(Q)$ is a linearly independent set of vectors in $[\Delta]^{\beta^i}$.*

We defer the proof of Lemma 4 to Section 4.1, and instead continue our proof of Lemma 3.

In light of Lemma 4, we set out to find a set of cells $C_i \subseteq S_i(U)$ and a grid G, such that the set of queries Q_{C_i} that probe no cells in $S_i(U) \setminus C_i$, hit a large number of grid cells in G. For this, first define the grids $G_2, \ldots, G_{2i-2}$ where G_j has width $n/\beta^{i-j/2}$ and height $n/\beta^{j/2}$. The existence of C_i is guaranteed by the following lemma:

LEMMA 5. *Let $i \geq \frac{15}{16} \lg_\beta n$ be an epoch and $U_{\lg_\beta n}, \ldots, U_1$ a fixed sequence of updates, where each U_j is a possible outcome of $\mathbf{U}_j$. Assume furthermore that the claimed data structure satisfies $t_i(U) = o(\lg_\beta n)$. Then there exists a set of cells $C_i \subseteq S_i(U)$ and an index $j \in \{2, \ldots, 2i-2\}$, such that $|C_i| = O(\beta^{i-1}w)$ and Q_{C_i} has hitting number $\Omega(\beta^{i-3/4})$ on the grid G_j.*

To not remove focus from our proof of Lemma 3 we have moved the proof of this lemma to Section 4.2. We thus move on to show that Lemma 4 and Lemma 5 implies Lemma 3. By assumption we have $t_i(U) = o(\lg_\beta n)$. Combining this with Lemma 5, we get that there exists a set of cells $C_i \subseteq S_i(U)$ and an index $j \in \{2, \ldots, 2i-2\}$, such that $|C_i| = O(\beta^{i-1}w)$ and the set of queries Q_{C_i} has hitting number $\Omega(\beta^{i-3/4})$ on the grid G_j. Furthermore, we have that grid G_j is a grid of the form required by Lemma 4, with $\mu = n/\beta^{i-j/2}$. Thus by Lemma 4 there is a subset $Q \subseteq Q_{C_i}$ such that $|Q| = \Omega(\beta^{i-3/4} - 12\beta^{i-1}) = \Omega(\beta^{i-3/4})$ and $\chi_i(Q)$ is a linearly independent set of vectors in $[\Delta]^{\beta^i}$. This completes the proof of Lemma 3.

4.1 Proof of Lemma 4

We prove the lemma by giving an explicit construction of the set Q.

First initialize Q to contain one query point from Q' from each cell of G that is hit by Q'. We will now repeatedly eliminate queries from Q until the remaining set is linearly independent. We do this by *crossing out* rows and columns of G. By crossing out a row (column) of G, we mean deleting all queries in Q that hits a cell in that row (column). Our procedure for crossing out rows and columns is as follows:

First cross out the bottom two rows and leftmost two columns. Amongst the remaining columns, cross out either the even or odd columns, whichever of the two contains the fewest remaining points in Q. Repeat this once again for the columns, with even and odd redefined over the remaining columns. Finally, do the same for the rows. We claim that the remaining set of queries are linearly independent. To see this, order the remaining queries in increasing order of column index (leftmost column has lowest index), and secondarily in increasing order of row index (bottom row has lowest index). Let $q_1, \ldots, q_{|Q|}$ denote the resulting sequence of queries. For this sequence, it holds that for every query q_j, there exists a coordinate $\chi_i(q_j)_h$, such that $\chi_i(q_j)_h = 1$, and at the same time $\chi_i(q_k)_h = 0$ for all $k < j$. Clearly this implies linear independence. To prove that the remaining vectors have this property, we must show that for each query q_j, there is some point in the scaled Fibonacci lattice F_{β^i} that is dominated by q_j, but not by any of $q_1, \ldots, q_{j-1}$: Associate each remaining query q_j to the two-by-two crossed out grid cells to the bottom-left of the grid cell hit by q_j. These four grid cells have area $4n^2/\beta^i$ and are contained within the rectangle $[0, n - n/\beta^i] \times [0, n - n/\beta^i]$, thus from Lemma 1 it follows that at least one point of the scaled Fibonacci lattice F_{β^i} is contained therein, and thus dominated by q_j. But all q_k, where $k < j$, either hit a grid cell in a column with index at least three less than that hit by q_j (we crossed out the two columns preceding that hit by q_j), or they hit a grid cell in the same column as q_j but with

a row index that is at least three lower than that hit by q_j (we crossed out the two rows preceding that hit by q_j). In either case, such a query cannot dominate the point inside the cells associated to q_j.

What remains is to bound the size of Q. Initially, we have $|Q| = h$. The bottom two rows have a total area of $2n^3/\beta^i\mu$, thus by Lemma 1 they contain at most $6n/\mu$ points. The leftmost two columns have area $2n\mu$ and thus contain at most $6\mu\beta^i/n$ points. After crossing out these rows and column we are therefore left with $|Q| \geq h - 6n/\mu - 6\mu\beta^i/n$. Finally, when crossing out even or odd rows we always choose the one eliminating fewest points, thus the remaining steps at most reduce the size of Q by a factor 16. This completes the proof of Lemma 4.

4.2 Proof of Lemma 5

We prove the lemma using another encoding argument. However, this time we do not encode the update sequence, but instead we define a distribution over query sets, such that if Lemma 5 is not true, then we can encode such a query set in too few bits.

Let $U = U_{\lg_\beta n}, \ldots, U_1$ be a fixed sequence of updates, where each U_j is a possible outcome of $\mathbf{U}_j$. Furthermore, assume for contradiction that the claimed data structure satisfies both $t_i(U) = o(\lg_\beta n)$ and for all cell sets $C \subseteq S_i(U)$ of size $|C| = O(\beta^{i-1}w)$ and every index $j \in \{2, \ldots, 2i-2\}$, it holds that the hitting number of Q_C on grid G_j is $o(\beta^{i-3/4})$. Here Q_C denotes the set of all queries q in $[n] \times [n]$ such that the query algorithm of the claimed data structure probes no cells in $S_i(U) \setminus C$ when answering q after the sequence of updates U. Under these assumptions we will construct an impossible encoder. As mentioned, we will encode a set of queries:

Hard Distribution.

Let $\mathbf{Q}$ denote a random set of queries, constructed by drawing one uniform random query (with integer coordinates) from each of the β^{i-1} vertical slabs of the form

$$[hn/\beta^{i-1}, (h+1)n/\beta^{i-1}) \times [0, n),$$

where $h \in [\beta^{i-1}]$. Our goal is to encode $\mathbf{Q}$ in less than $H(\mathbf{Q}) = \beta^{i-1}\lg(n^2/\beta^{i-1})$ bits in expectation. Before giving our encoding and decoding procedures, we prove some simple properties of $\mathbf{Q}$:

Define a query q in a query set Q' to be *well-separated* if for all other queries $q' \in Q'$, where $q \neq q'$, q and q' do not lie within an axis-aligned rectangle of area $n^2/\beta^{i-1/2}$. Finally, define a query set Q' to be *well-separated* if at least $\frac{1}{2}|Q'|$ queries in Q' are well-separated. We then have:

LEMMA 6. *The query set $\boldsymbol{Q}$ is well-separated with probability at least* 3/4.

PROOF. Let $\mathbf{q}_h$ denote the random query in $\mathbf{Q}$ lying in the h'th vertical slab. The probability that $\mathbf{q}_h$ lies within a distance of at most $n/\beta^{i-3/4}$ from the x-border of the h'th slab is precisely $(2n/\beta^{i-3/4})/(n/\beta^{i-1}) = 2/\beta^{1/4}$. If this is not the case, then for another query $\mathbf{q}_k$ in $\mathbf{Q}$, we know that the x-coordinates of $\mathbf{q}_h$ and $\mathbf{q}_k$ differ by at least $(|k-h|-1)n/\beta^{i-1} + n/\beta^{i-3/4}$. This implies that $\mathbf{q}_h$ and $\mathbf{q}_k$ can only be within an axis-aligned rectangle of area $n^2/\beta^{i-1/2}$ if their y-coordinates differ by at most $n/((|k-h|-1)\beta^{1/2} + \beta^{1/4})$. This happens with probability at most $2/((|k-h|-1)\beta^{1/2} + \beta^{1/4})$. The probability that a query $\mathbf{q}_h$ in $\mathbf{Q}$ is not well-separated is therefore bounded by

$$\frac{2}{\beta^{1/4}} + (1 - \frac{2}{\beta^{1/4}}) \sum_{k \neq j} \frac{2}{(|k-h|-1)\beta^{1/2} + \beta^{1/4}}$$

$$\leq \frac{10}{\beta^{1/4}} + \sum_{k \neq j} \frac{2}{|k-h|\beta^{1/2}} = O\left(\frac{1}{\beta^{1/4}} + \frac{\lg n}{\beta^{1/2}}\right).$$

Since $\beta = (wt_u)^9 = \omega(\lg^2 n)$ this probability is $o(1)$, and the result now follows from linearity of expectation and Markov's inequality. □

Now let $S_i(Q, U) \subseteq S_i(U)$ denote the subset of cells in $S_i(U)$ probed by the query algorithm of the claimed data structure when answering all queries in a set of queries Q after the sequence of updates U (i.e. the union of the cells probed for each query in Q). Since a uniform random query from $\mathbf{Q}$ is uniform in $[n] \times [n]$, we get by linearity of expectation that $\mathbb{E}[|S_i(\mathbf{Q}, U)|] = \beta^{i-1}t_i(U)$. From this, Lemma 6, Markov's inequality and a union bound, we conclude

LEMMA 7. *The query set $\boldsymbol{Q}$ is both well-separated and $|S_i(\boldsymbol{Q}, U)| \leq 4\beta^{i-1}t_i(U)$ with probability at least* 1/2.

With this established, we are now ready to give our impossible encoding of $\mathbf{Q}$.

Encoding.

In the following we describe the encoding procedure. The encoder receives as input a set of queries Q, where Q is a possible outcome of $\mathbf{Q}$. He then executes the below procedure to encode Q:

1. The encoder first executes the fixed sequence of updates U on the claimed data structure, and from this obtains the sets $S_{\lg_\beta n}(U), \ldots, S_1(U)$. He then runs the query algorithm for every query $q \in Q$ and collects the set $S_i(Q, U)$.

2. If Q is not well-separated or if $|S_i(Q, U)| > 4\beta^{i-1}t_i(U)$, then the encoder sends a 1-bit followed by a straight forward encoding of Q using $H(\mathbf{Q}) + O(1)$ bits in total. This is the complete encoding procedure when either Q is not well-separated or $|S_i(Q, U)| > 4\beta^{i-1}t_i(U)$.

3. If Q is both well-separated and $|S_i(Q, U)| \leq 4\beta^{i-1}t_i(U)$, then the encoder first writes a 0-bit and then executes the remaining four steps.

4. The encoder examines Q and finds the at most $\frac{1}{2}|Q|$ queries that are not well-separated. Denote this set Q'. The encoder now writes down Q' by first specifying $|Q'|$, then which vertical slabs contain the queries in Q' and finally what the coordinates of each query in Q' is within its slab. This takes $O(w) + \lg\binom{|Q|}{|Q'|} + |Q'|\lg(n^2/\beta^{i-1}) = O(w) + O(\beta^{i-1}) + |Q'|\lg(n^2/\beta^{i-1})$ bits.

5. The encoder now writes down the cell set $S_i(Q, U)$, including **only** the addresses and **not** the contents. This takes $o(H(\mathbf{Q}))$ bits since

$$\begin{aligned} \lg\binom{|S_i(U)|}{|S_i(Q,U)|} &= O(\beta^{i-1}t_i(U)\lg(\beta t_u)) \\ &= o(\beta^{i-1}\lg(n^2/\beta^{i-1})), \end{aligned}$$

where in the first line we used that $|S_i(U)| \leq \beta^i t_u$ and $|S_i(Q,U)| \leq 4\beta^{i-1}t_i(U)$. The second line follows from the fact that $t_i(U) = o(\lg_\beta n) = o(\lg(n^2/\beta^{i-1})/\lg(\beta t_u))$ since $\beta = \omega(t_u)$.

6. Next we encode the x-coordinates of the well-separated queries in Q. Since we have already encoded which vertical slabs contain well-separated queries (we really encoded the slabs containing queries that are not well-separated, but this is equivalent), we do this by specifying only the offset within each slab. This takes $(|Q|-|Q'|)\lg(n/\beta^{i-1})+O(1)$ bits. Following that, the encoder considers the last grid G_{2i-2}, and for each well-separated query q, he writes down the y-offset of q within the grid cell of G_{2i-2} hit by q. Since the grid cells of G_{2i-2} have height n/β^{i-1}, this takes $(|Q|-|Q'|)\lg(n/\beta^{i-1})+O(1)$ bits. Combined with the encoding of the x-coordinates, this step adds a total of $(|Q|-|Q'|)\lg(n^2/\beta^{2i-2})+O(1)$ bits to the size of the encoding.

7. In the last step of the encoding procedure, the encoder simulates the query algorithm for every query in $[n]\times[n]$ and from this obtains the set $Q_{S_i(Q,U)}$, i.e. the set of all those queries that probe no cells in $S_i(U)\setminus S_i(Q,U)$. Observe that $Q \subseteq Q_{S_i(Q,U)}$. The encoder now considers each of the grids G_j, for $j = 2,\ldots,2i-2$, and determines both the set of grid cells $G_j^{Q_{S_i(Q,U)}} \subseteq G_j$ hit by a query in $Q_{S_i(Q,U)}$, and the set of grid cells $G_j^Q \subseteq G_j^{Q_{S_i(Q,U)}} \subseteq G_j$ hit by a well-separated query in Q. The last step of the encoding consists of specifying G_j^Q. This is done by encoding which subset of $G_j^{Q_{S_i(Q,U)}}$ corresponds to G_j^Q. This takes $\lg\binom{|G_j^{Q_{S_i(Q,U)}}|}{|G_j^Q|}$ bits for each $j = 2,\ldots,2i-2$.

   Since $|S_i(Q,U)| = o(\beta^{i-1}\lg_\beta n) = o(\beta^{i-1}w)$ we get from our contradictory assumption that the hitting number of $Q_{S_i(Q,U)}$ on each grid G_j is $o(\beta^{i-3/4})$, thus $|G_j^{Q_{S_i(Q,U)}}| = o(\beta^{i-3/4})$. Therefore the above amount of bits is at most

$$\begin{aligned}(|Q|-|Q'|)\lg(\beta^{i-3/4}e/(|Q|-|Q'|))(2i-3) &\leq\\ (|Q|-|Q'|)\lg(\beta^{1/4})2i + O(\beta^{i-1}i) &\leq\\ (|Q|-|Q'|)\tfrac{1}{4}\lg(\beta)2\lg_\beta n + O(\beta^{i-1}\lg_\beta n) &\leq\\ (|Q|-|Q'|)\tfrac{1}{2}\lg n + o(H(\mathbf{Q}))\end{aligned}$$

   This completes the encoding procedure, and the encoder finishes by sending the constructed message to the decoder.

Before analysing the size of the encoding, we show that the decoder can recover Q from the encoding.

Decoding.

In this paragraph we describe the decoding procedure. The decoder only knows the fixed sequence $U = U_{\lg_\beta n},\ldots,U_1$ and the message received from the encoder. The goal is to recover Q, which is done by the following steps:

1. The decoder examines the first bit of the message. If this is a 1-bit, the decoder immediately recovers Q from the remaining part of the encoding.

2. If the first bit is 0, the decoder proceeds with this step and all of the below steps. The decoder executes the updates U on the claimed data structure and obtains the sets $S_{\lg_\beta n}(U),\ldots,S_1(U)$. From step 4 of the encoding procedure, the decoder also recovers Q'.

3. From step 5 of the encoding procedure, the decoder now recovers the addresses of the cells in $S_i(Q,U)$. Since the decoder has the data structure, he already knows the contents. Following this, the decoder now simulates every query in $[n]\times[n]$, and from this and $S_i(Q,U)$ recovers the set $Q_{S_i(Q,U)}$.

4. From step 6 of the encoding procedure, the decoder now recovers the x-coordinates of every well-separated query in Q (the offsets are enough since the decoder knows which vertical slabs contain queries in Q', and thus also those that contain well-separated queries). Following that, the decoder also recovers the y-offset of each well-separated query $q \in Q$ within the grid cell of G_{2i-2} hit by q (note that the decoder does not know what grid cell it is, he only knows the offset).

5. From the set $Q_{S_i(Q,U)}$ the decoder now recovers the set $G_j^{Q_{S_i(Q,U)}}$ for each $j = 2,\ldots,2i-2$. This information is immediate from the set $Q_{S_i(Q,U)}$. From $G_j^{Q_{S_i(Q,U)}}$ and step 7 of the encoding procedure, the decoder now recovers G_j^Q for each j. In grid G_2, we know that Q has only one query in every column, thus the decoder can determine uniquely from G_2^Q which grid cell of G_2 is hit by each well-separated query in Q. Now observe that the axis-aligned rectangle enclosing all $\beta^{1/2}$ grid cells in G_{j+1} that intersects a fixed grid cell in G_j has area $n^2/\beta^{i-1/2}$. Since we are considering well-separated queries, i.e. queries where no two lie within an axis-aligned rectangle of area $n^2/\beta^{i-1/2}$, this means that G_{j+1}^Q contains at most one grid cell in such a group of $\beta^{1/2}$ grid cells. Thus if q is a well-separated query in Q, we can determine uniquely which grid cell of G_{j+1} that is hit by q, directly from G_{j+1}^Q and the grid cell in G_j hit by q. But we already know this information for grid G_2, thus we can recover this information for grid $G_3, G_4, \ldots, G_{2i-2}$. Thus we know for each well-separated query in Q which grid cell of G_{2i-2} it hits. From the encoding of the x-coordinates and the y-offsets, the decoder have thus recovered Q.

Analysis.

Finally we analyse the size of the encoding. First consider the case where $\mathbf{Q}$ is both well-separated and $|S_i(\mathbf{Q},U)| \leq 4\beta^{i-1}t_i(U)$. In this setting, the size of the message is bounded by

$$|\mathbf{Q}'|\lg(n^2/\beta^{i-1}) + (|\mathbf{Q}|-|\mathbf{Q}'|)(\lg(n^2/\beta^{2i-2}) + \tfrac{1}{2}\lg n) + o(H(\mathbf{Q}))$$

bits. This equals

$$|\mathbf{Q}|\lg(n^{2+1/2}/\beta^{2i-2}) + |\mathbf{Q}'|\lg(\beta^{i-1}/n^{1/2}) + o(H(\mathbf{Q}))$$

bits. Since we are considering an epoch $i \geq \frac{15}{16}\lg_\beta n$, we have $\lg(n^{2+1/2}/\beta^{2i-2}) \leq \lg(n^{5/8}\beta^2)$, thus the above amount of bits is upper bounded by

$$|\mathbf{Q}|\lg(n^{5/8}\beta^2) + |\mathbf{Q}'|\lg(n^{1/2}) + o(H(\mathbf{Q})).$$

Since $|\mathbf{Q}'| \leq \frac{1}{2}|\mathbf{Q}|$, this is again bounded by

$$|\mathbf{Q}| \lg(n^{7/8}\beta^2) + o(H(\mathbf{Q}))$$

bits. But $H(\mathbf{Q}) = |\mathbf{Q}| \lg(n^2/\beta^i) \geq |\mathbf{Q}| \lg n$, i.e. our encoding uses less than $\frac{15}{16}H(\mathbf{Q})$ bits.

Finally, let E denote the event that $\mathbf{Q}$ is well-separated and at the same time $|S_i(\mathbf{Q}, U)| \leq 4\beta^{i-1}t_i(U)$, then the expected number of bits used by the entire encoding is bounded by

$$O(1) + \Pr[E](1 - \Omega(1))H(\mathbf{Q}) + (1 - \Pr[E])H(\mathbf{Q})$$

The contradiction is now reached by invoking Lemma 7 to conclude that $\Pr[E] \geq 1/2$.

5. CONCLUDING REMARKS

In this paper we presented a new technique for proving dynamic cell probe lower bounds. With this technique we proved the highest dynamic cell probe lower bound to date under the most natural setting of cell size $w = \Theta(\lg n)$, namely a lower bound of $t_q = \Omega((\lg n/\lg(wt_u))^2)$.

While our results have taken the field of cell probe lower bounds one step further, there is still a long way to go. Amongst the results that seems within grasp, we find it a very intriguing open problem to prove an $\omega(\lg n)$ lower bound for a problem where the queries have a one bit output. Our technique crucially relies on the output having more bits than it takes to describe a query, since otherwise the encoder cannot afford to tell the decoder which queries to simulate. Since many interesting data structure problems have a one bit output size, finding a technique for handling this case would allow us to attack many more fundamental data structure problems.

Applying our technique to other problems is also an important task, however such problems must again have a logarithmic number of bits in the output of queries.

6. ACKNOWLEDGMENT

The author wishes to thank Peter Bro Miltersen for much useful discussion on both the results and writing of this paper.

7. REFERENCES

[1] A. Fiat and A. Shamir. How to find a battleship. *Networks*, 19:361–371, 1989.

[2] M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proc. 21st ACM Symposium on Theory of Computation*, pages 345–354, 1989.

[3] A. G. Jørgensen and K. G. Larsen. Range selection and median: Tight cell probe lower bounds and adaptive data structures. In *Proc. 22nd ACM/SIAM Symposium on Discrete Algorithms*, pages 805–813, 2011.

[4] J. Matoušek. *Geometric Discrepancy*. Springer, 1999.

[5] R. Panigrahy, K. Talwar, and U. Wieder. Lower bounds on near neighbor search via metric expansion. In *Proc. 51st IEEE Symposium on Foundations of Computer Science*, pages 805–814, 2010.

[6] M. Pătraşcu. Lower bounds for 2-dimensional range counting. In *Proc. 39th ACM Symposium on Theory of Computation*, pages 40–46, 2007.

[7] M. Pătraşcu. Unifying the landscape of cell-probe lower bounds. In *Proc. 49th IEEE Symposium on Foundations of Computer Science*, pages 434–443, 2008.

[8] M. Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proc. 42nd ACM Symposium on Theory of Computation*, pages 603–610, 2010.

[9] M. Pătraşcu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35:932–963, April 2006.

[10] M. Pătraşcu and M. Thorup. Don't rush into a union: Take time to find your roots. In *Proc. 43rd ACM Symposium on Theory of Computation*, 2011. To appear. See also arXiv:1102.1783.

[11] A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. 18th IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1977.

[12] A. C. C. Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981.