On the Scalability of Computing Triplet and Quartet Distances

Morten Kragelund Holt

Jens Johansen

Gerth Stølting Brodal

MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

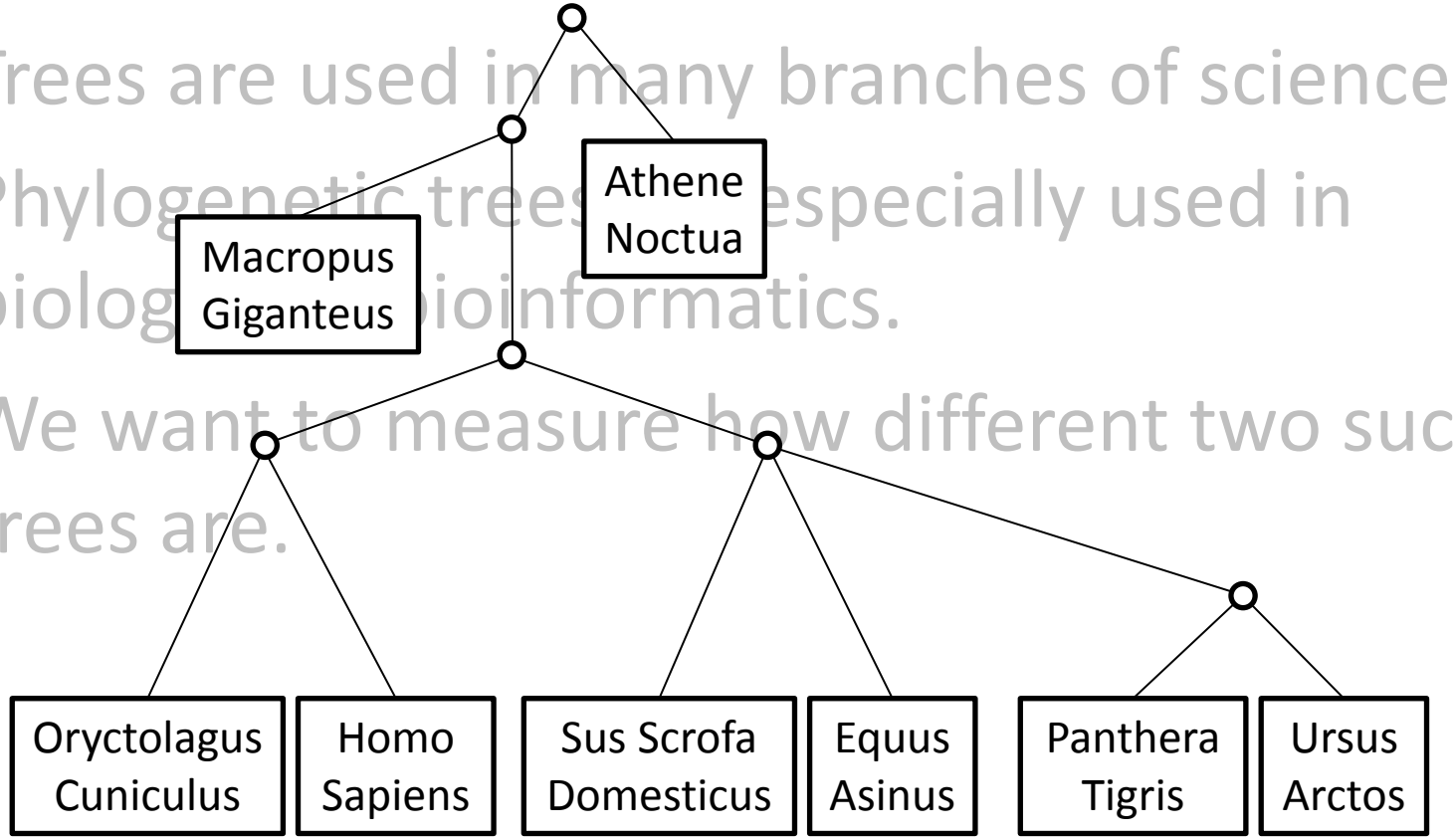
Aarhus University

Introduction

- Trees are used in many branches of science.
- Phylogenetic trees are especially used in biology and bioinformatics.
- We want to measure how different two such trees are.

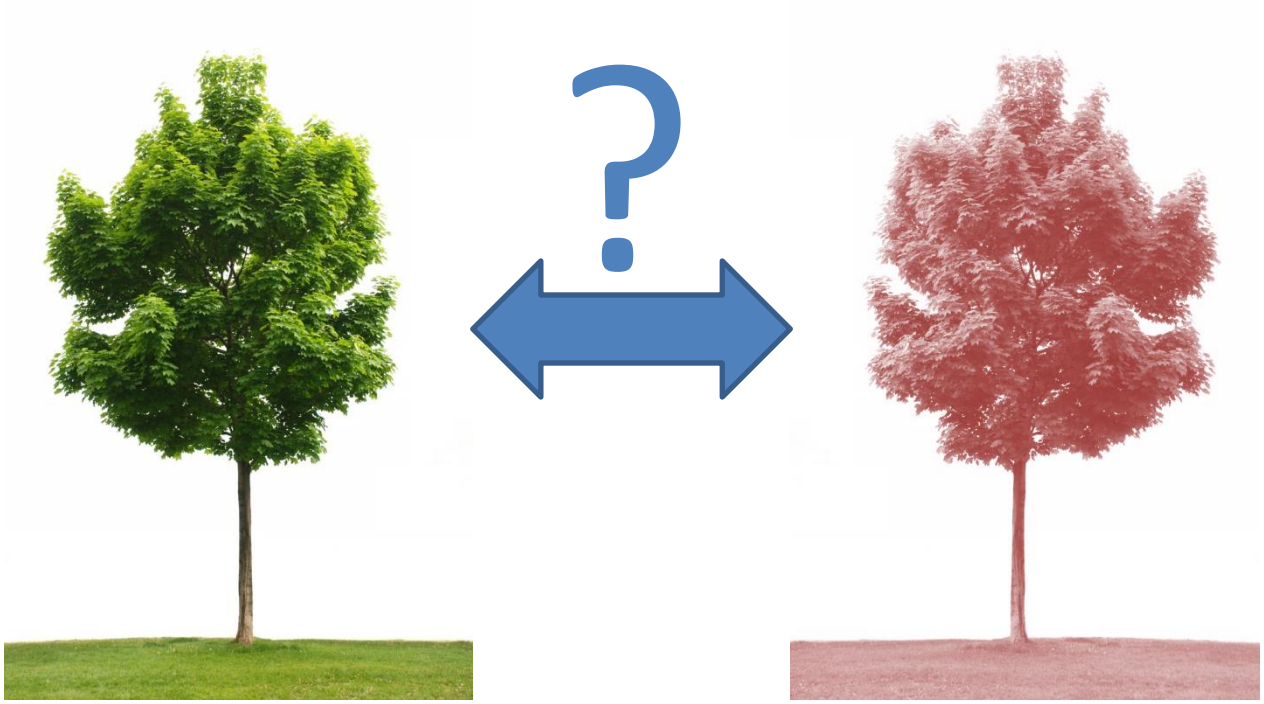
Introduction

- Trees are used in many branches of science.
- Phylogenetic trees especially used in biolog bioinformatics.
- We want to measure how different two such trees are.



Distances

- Natural in some cases.
- Between trees?



Triplets and Quartets

Triplets

- Used in **rooted** trees.
- Sub-trees consisting of **three** leaves.
- $\binom{n}{3}$ in a tree with n leaves.
- With 2,000 leaves, 1,331,334,000 triplets.
- Naïve algorithm runs in at least $\Omega(n^3)$.
- Number of *disagreeing* triplets.

Quartets

- Used in **unrooted** trees.
- Sub-trees consisting of **four** leaves.
- $\binom{n}{4}$ in a tree with n leaves.
- With 2,000 leaves, 664,668,499,500 quartets.
- Naïve algorithm runs in at least $\Omega(n^4)$.
- Number of *disagreeing* quartets.

Goal

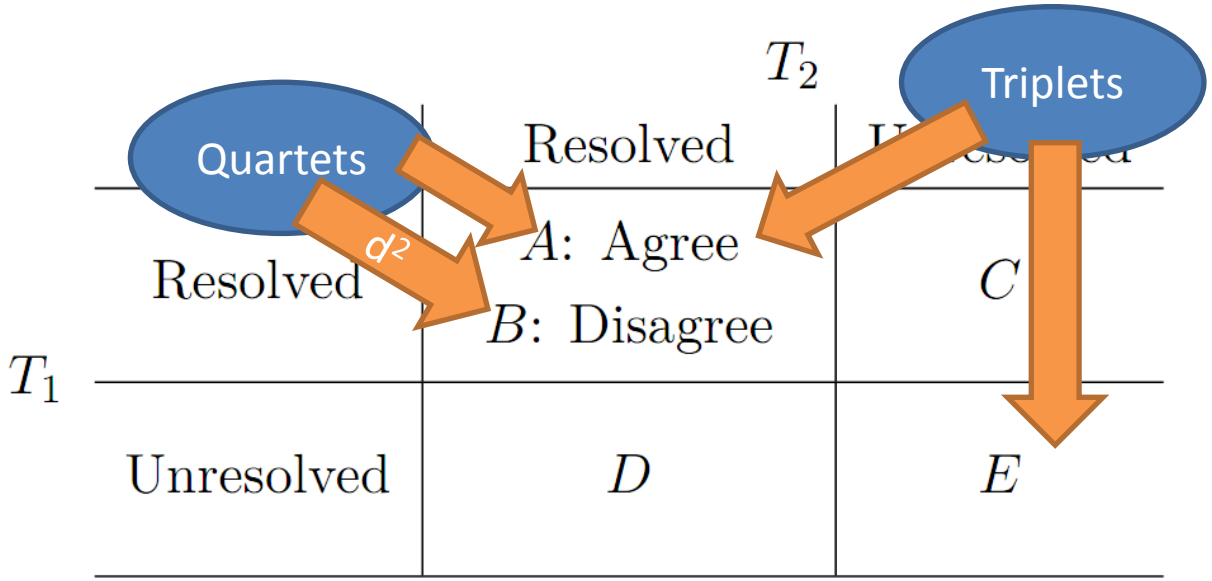
- Comparison of two trees ($\mathbf{T_1}$ and $\mathbf{T_2}$) with the same set of leaf-labels.
  - Numerical value of the **difference** of the two trees.
  - Number of different triplets (quartets) in the two input trees.
- A tree has a distance of 0 to itself.

Brodal *et al.* [SODA13]

		T_2	
		Resolved	Unresolved
T_1	Resolved	A: Agree B: Disagree	C
	Unresolved	D	E

- For binary trees C, D and E are all zero ☺

Brodal *et al.* [SODA13]



- For binary trees C, D and E are all zero ☺

Brodal *et al.* [SODA13]

	Binary	Arbitrary degree
Triplets	$O(n \lg n)$ Up to $4d+2$ counters in each HDT node	
Quartets	$O(n \lg n)$ $2d^2 + 79d + 22$ counters	$O(\max(d_1, d_2)\, n \lg n)$ $2d^2 + 79d + 22$ counters

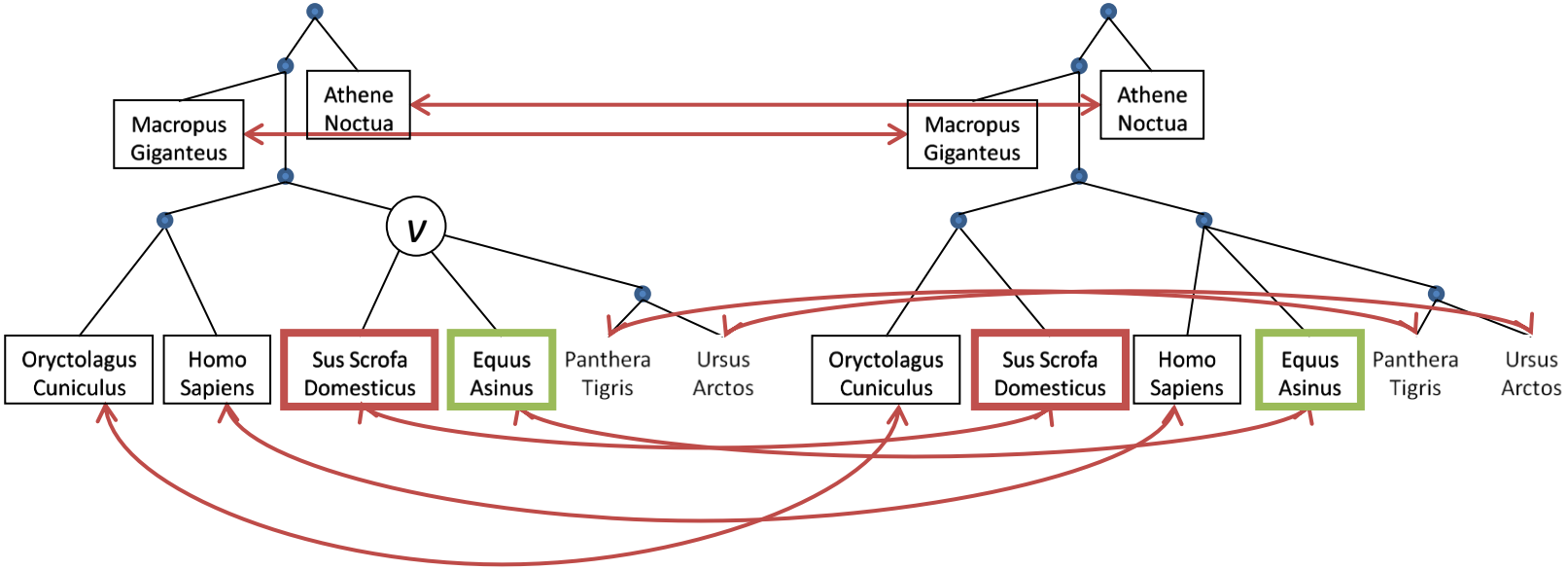
- A lot of counters ☹. Is this even feasible?
- Why the *d* factor on arbitrary degree quartets?
  - d^2 counters

Overview

- Basic idea
  - Each triplet (quartet) is anchored somewhere in $\mathbf{T_1}$.
  - Run through $\mathbf{T_1}$, and for each triplet (quartet), check if they are anchored the same way in $\mathbf{T_2}$.
- The algorithm consists of **four** parts
  1. Coloring
  2. Counting
  3. Hierarchical Decomposition Tree (HDT)
  4. Extraction and contraction

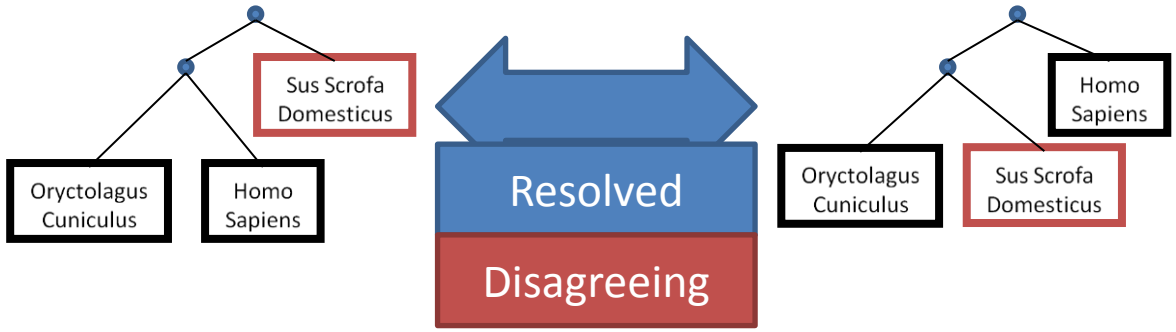
1. Coloring

- Consists of two steps
  1. Leaf-linking O(n)
  2. Recursive coloring O(n lg n)



2. Counting

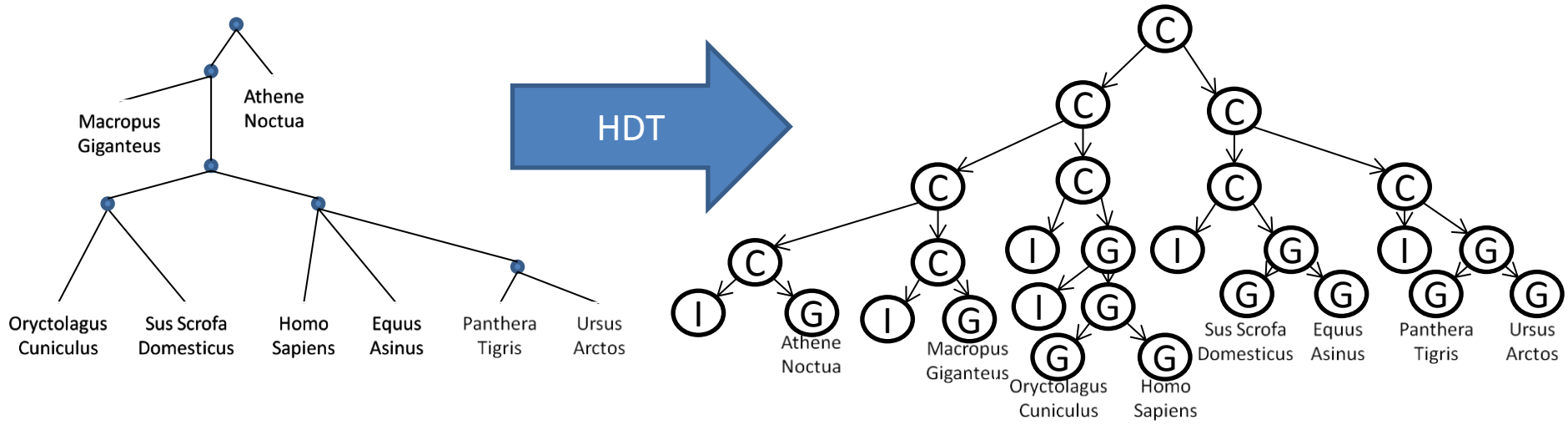
- Using the coloring of $\mathbf{T_1}$ and $\mathbf{T_2}$ we count the number of similar triplets (quartets).



- No reason to look at all triplets (would be much too slow)
  - Instead, look at inner nodes.
- In each inner node, we can keep track of the number of different triplets (quartets), rooted at the given node.
- Using counting and coloring, the triplet distance can be calculated in $O(n^2)$.

3. Hierarchical Decomposition Tree (HDT)

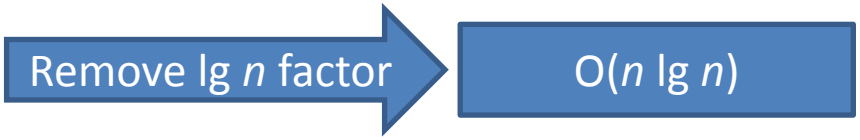
- Problem: $\mathbf{T_2}$ is unbalanced.
- Solution: Hierarchical Decomposition Trees.



Built in linear time | Locally balanced | Triplet distance in $O(n \lg^2 n)$

4. Extraction and Contraction

- Ensuring that the HDT is small, we can cut off that lg *n* factor.
- If the HDT is too large, remove the irrelevant parts.



Optimizations

1. [SODA13] hints at constructing HDTs early.
   Problem: HDTs take up a lot of memory.
   Solution: Postpone HDT construction.
   Result: **25-50%** reduction in memory usage.
   4-10% reduction in runtime.
2. Utilizing the standard C++ vector data structure.
   Problem: Relatively slow (for our needs).
   Solution: A purpose-built linked list implementation.
   Result: **6-9%** reduction in runtime on binary trees.
3. Allocating memory whenever needed.
   Problem: (Relatively) slow to allocate memory.
   Solution: Allocation in large blocks.
   Result: **18-25%** improvement in the runtime.
   10-20% increase in memory usage on large input.

25% improvement in runtime

45% reduction in memory usage

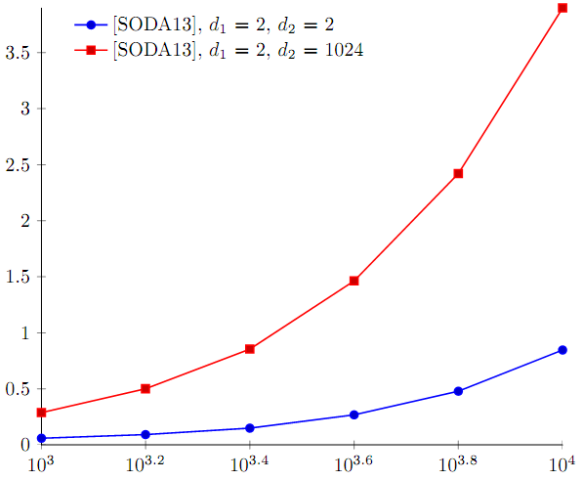
Limitations

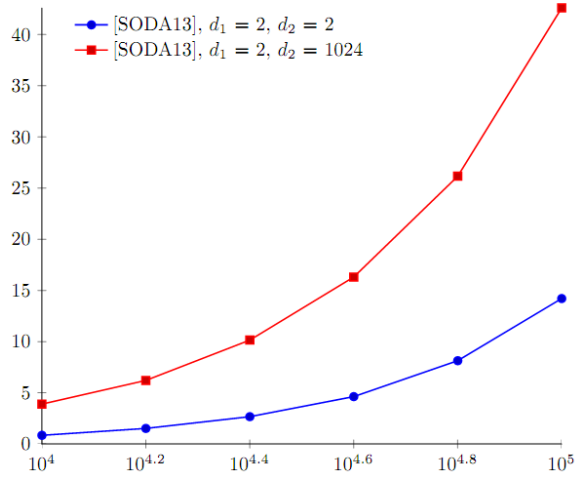
Two primary limitations in our implementation:

- **Integer representation**
  - $\binom{n}{3}$ and $\binom{n}{4}$ are in the order of n^3 and n^4.
  - With signed 64-bit integers, quartet distance of only 55,000 leaves.
  - *Solution:* Signed 128-bit integers for n^4 counters.
    - Quartet distance of up to 2,000,000 leaves.
- **Recursion depth**
  - OS imposed limitation in recursion stack depth.
  - Input, consisting of a very long chain, will fail.
  - Windows: Height ~4,000.
  - Linux: Height ~48,000.
  - *Solution:* Purpose built stack implementation*.

*Not done in the implementation

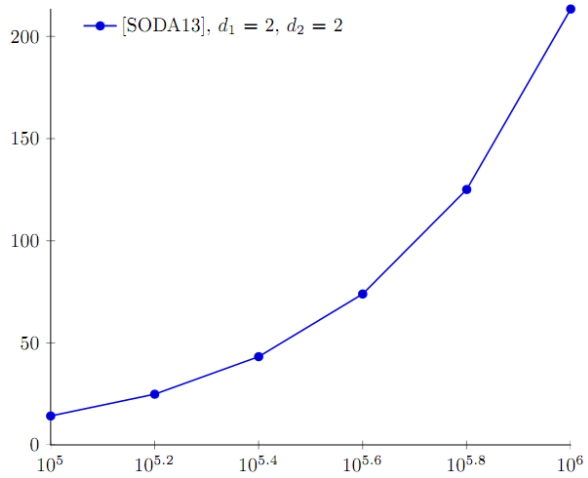
Results: [SODA13]





It works, and it is fast!

☺

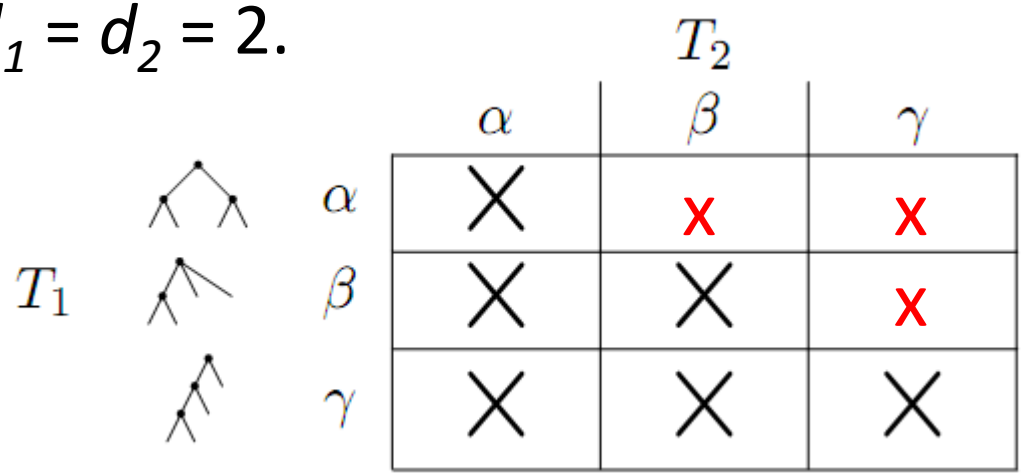


Leaves	Time (s)
1,000	.29
10,000	3.90
100,000	42.60
1,000,000	N/A

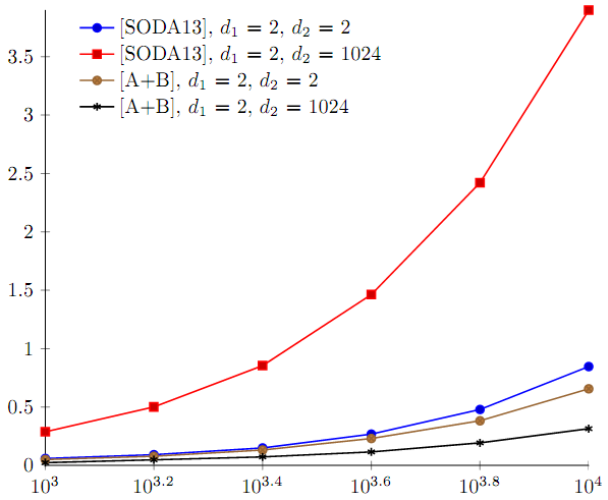
Improvements

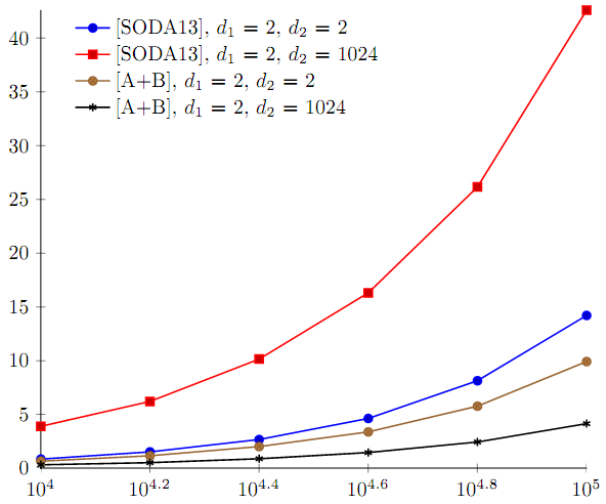
- Why min (d_1, d_2)?
  - d-counters given by first input tree
  - [SODA13]: Calculates 6 out of 9 cases.
  - [SODA13]: $d_1 = 2$, $d_2 = 1024$ is much slower than $d_1 = d_2 = 2$.

Add $5d^2 + 18d + 7$ counters
Total $7d^2 + 97d + 29$ counters
Remove need for swapping
$O(\min(d_1, d_2)\, n \lg n)$

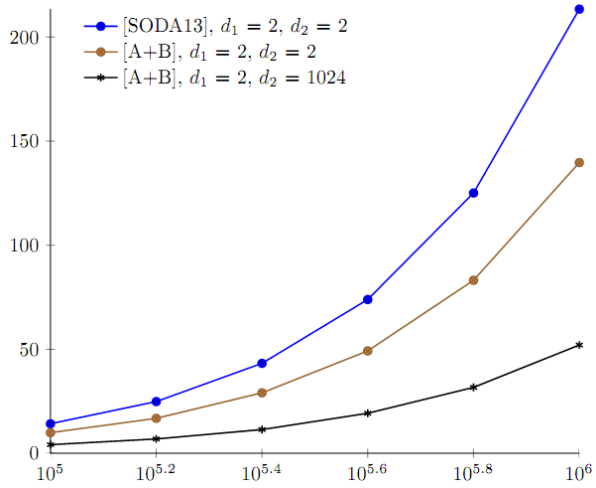


Results: Improved





Faster in alle cases ☺

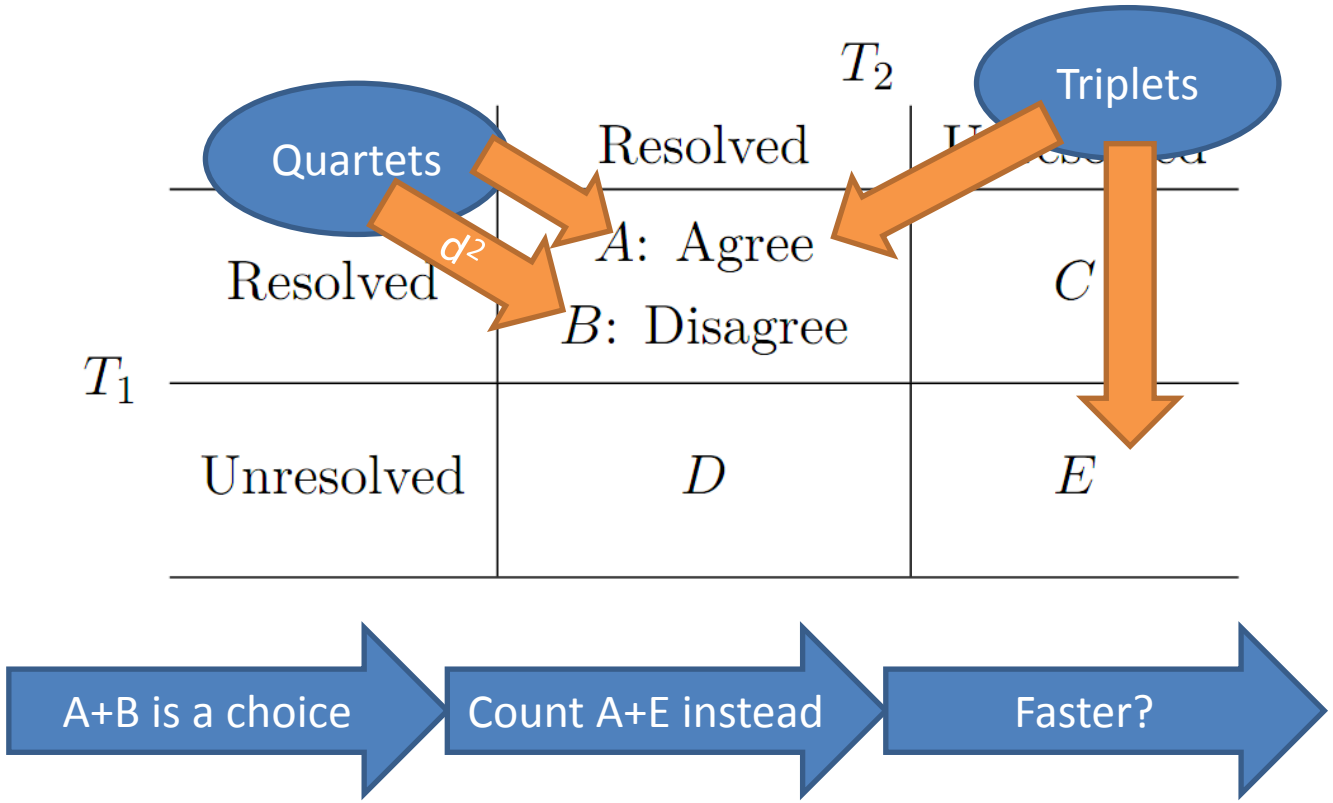


Leaves	Time (s)
1,000	.02
10,000	.31
100,000	4.14
1,000,000	52.05

MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

Holt, Johansen, Brodal
On the Scalability of Computing Triplet and Quartet Distances

More improvements



More improvements

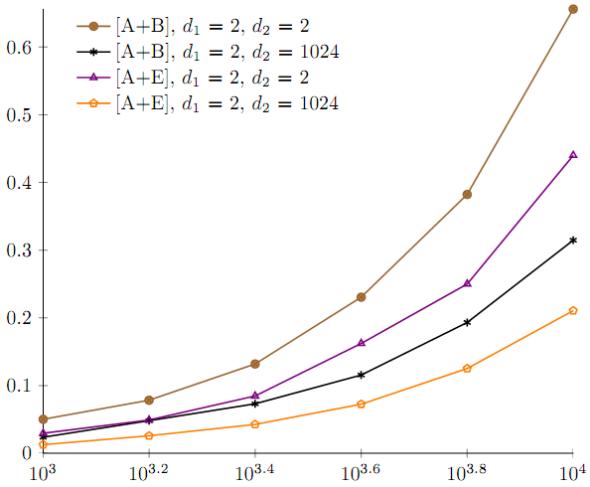
To count B

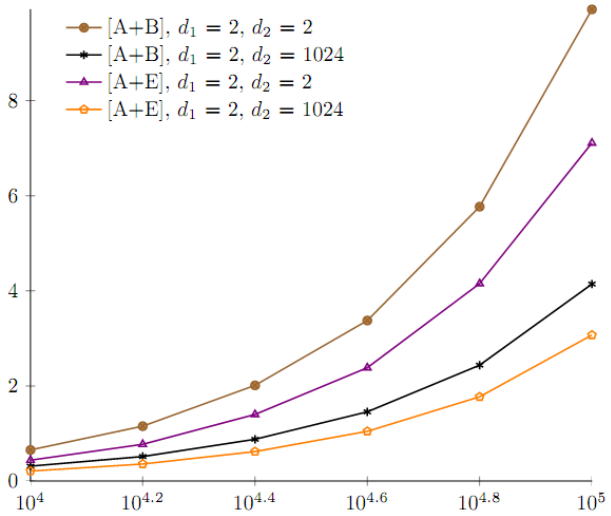
- 14 cases
- 92 sums
- $5d^2 + 48d + 8$ counters
- $O(\min(d_1, d_2)\, n \lg n)$

To Count E

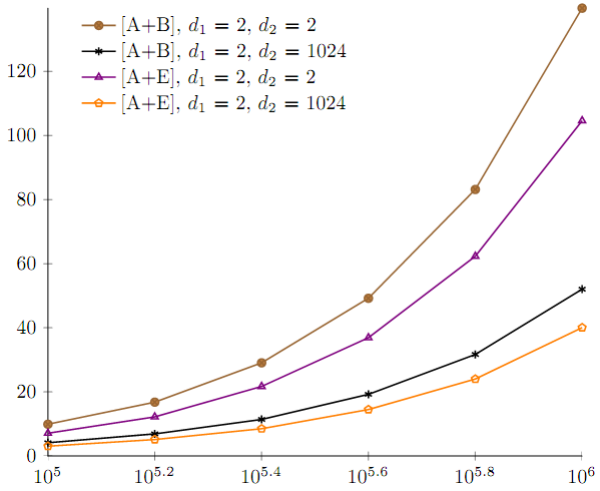
- 5 cases
- 21 sums
- $1d^2 + 12d + 12$ counters
- $O(\min(d_1, d_2)\, n \lg n)$

Results: More improvements





Fastest in the field ☺



Leaves	Time (s)
1,000	.01
10,000	.21
100,000	3.07
1,000,000	40.06

MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

Holt, Johansen, Brodal
On the Scalability of Computing Triplet and Quartet Distances

Overview

	Binary	**Arbitrary degree** $d_1 = d_2 = 256$
Triplets	[SODA13]: O(n lg n) [SODA13]: ~34 seconds	[SODA13]: O(n lg n) [SODA13]: ~7 seconds
Quartets	[SODA13]: O(n lg n) [SODA13]: ~125 seconds [ALENEX14] v1: ~83 seconds [ALENEX14] v2: ~62 seconds	[SODA13]: O(max(d_1, d_2) n lg n) [ALENEX14]: O(min(d_1, d_2) n lg n) [SODA13]: ~139 seconds [ALENEX14] v1: ~112 seconds [ALENEX14] v2: ~45 seconds

Balanced tree, 630.000 leaves

Conclusion

- [SODA13] is both practical and implementable.
- We have
  - Performed a thorough study of the alternative choices not studied in [SODA13].
  - Theoretically, and practically, found good choices for the parameters.
  - Shown that [SODA13], and derivatives, successfully scales up to trees with millions of nodes.
- Open problem
  - Current algorithm makes heavy use of random accesses, and doesn't scale to external memory.
  - Current algorithm is single-threaded.