Centerpoint Query Authentication

Magnus Haxen
Aarhus University

Morten Raeburn
Aarhus University

Peyman Afshani
Aarhus University

Panagiotis Karras
Aarhus University

ABSTRACT

The rise of online map services drives data owners to outsource spatial data to potentially untrusted database providers. Query results are provided along with verification objects that allow confirming their authenticity. Such authentication schemes have been proposed for several spatial and geometric queries, as well as for median queries in one dimension. However, to date, no authentication mechanism exists for centerpoint queries, which return a point lying in the middle of other points in multidimensional space. In this paper, we propose an authentication scheme for centerpoint queries, grounded on the algorithm for centerpoint queries on a finite planar set of points and authenticated aggregation R-trees and accompanying authenticated aggregation queries. We also provide methods for finding the centerpoint of a subset of the complete data set, and implement a range-based method. Our solution has a worst-case time-complexity of $O(n \log n)$ and space-complexity of $O(n)$. Our experimental study confirms these claims.

CCS CONCEPTS

• **Information systems → Data management systems**; • **Security and privacy → Information accountability and usage control**.

KEYWORDS

centerpoint query, query authentication

ACM Reference Format:
Magnus Haxen, Morten Raeburn, Peyman Afshani, and Panagiotis Karras. 2021. Centerpoint Query Authentication. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3459637.3482072

1 INTRODUCTION

With the widespread adaptation of internet-connected devices, many data owners (DOs) cannot afford an infrastructure for providing query results to end-users. Instead, they outsource their data to specialized third-party database providers. Yet outsourcing incurs the risk of tampering with or corruption of the supplied data. *Query authentication* mechanisms allow a database provider to append a verification object ($\mathcal{VO}$) to the data returned to the client, which the client can use, along with a signed digest of the entire data provided by the data owner, to verify the correctness and completeness of the result [8]; this solution has been applied, among others, to range [7], aggregation [5], and top-k queries [1].

Yet there are spatial queries for which no authentication scheme has been proposed. Consider, for example, a ride-sharing company that uses an outsourced database to provide drivers with suggestions of strategic locations to wait for new customers. Such locations may well be in the middle between currently busy clusters, to be returned by a *centerpoint query* [3], a generalization of a median in many dimensions. Assume the database provider launches a rival service aiming to gain a market advantage; in this situation, they may have an incentive to skew results sent to the rival's drivers. In these circumstances, an authentication scheme would ensure such competitiveness-motivated sabotage is not taking place.

To date, no algorithm has been proposed for authenticating *centerpoint queries*. We propose an $O(n \log n)$-time and $O(n)$-space scheme for this purpose, building upon the know-how for other spatial queries [1, 5, 7]. We also devise methods to query and authenticate centerpoints of subsets of the DOs static data.

2 RELATED WORKS

Here, we review some basic data structures for authentication.

2.1 Merkle Hash Trees

A Merkle Hash Tree (MHT) [6] is a binary tree used for authentication. Given a dataset, the leaves of its corresponding MHT keep the hashes of its elements by a collision-resistatn hash function; each internal tree node keeps the hash of the concatenation of its two children. The tree root has a hash value affected by every element in the dataset, hence any change to the dataset results in a different root value. This property allows for a verifying the contents of a database against the signature root-hash of the corresponding MHT; the verifier is given a verification object (VO) that contains the signature of the MHT-root and the siblings of the path leading from the root to the requested data, hence can compute the root by the hash function and verify it against the signature.

2.2 Merkel R-Trees

An R-Tree [2] is a tree data structure for spatial data using a hierarchy of minimum bounding rectangles (MBRs). Each node in an R-Tree has an associated MBR that contains all its descendants, while the MBR of a leaf node encloses indexed objects. While individual indexed objects may not overlap, the MBRs of internal nodes may do so. To authenticate spatial queries, we may combine properties of R-Trees and MHTs, to construct Merkel R-trees

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482072

(MR-trees) [8], which have the same purpose as MHTs but are tailored to *spatial data*. In an MR-tree, leaves correspond to data items and internal nodes have the form (p, MBR, h), where p is a set of pointers to its children, MBR is the minimum bound rectangle that covers those children, and h is the hash of the concatenation of all objects in the node. The size of each of these is at most *fanout*, the maximum number of children each internal node may have.

An extension of the MR-tree, the *Authenticated Aggregation R-tree* (AAR-tree) [5], is suitable for authenticating aggregation queries in multidimensional space. In an AAR-tree, each leaf holds an aggregation value, set to 1 when authenticating a COUNT query. Internal nodes hold aggregation values equal to $\sum a_i$, where a_i is the aggregation value of their ith child. In effect, internal node aggregation values indicate the number of descendent leaves. Each node, except for the root, has an associated label that reveals its position. The aggregate values of the *AAR*-tree allow the server to answer and authenticate an aggregation query [5], by traversing the *AAR*-tree to find the minimum set of nodes, $\mathcal{MCS}$, in the *AAR*-tree whose MBRs cover the query range. The aggregation query result is the sum of these nodes' aggregation values, $\sum_{i \in \mathcal{MCS}} \alpha_i$; thus, authenticating an aggregate query is reduced to authenticating $\mathcal{MCS}$. To do so, the server provides the minimum set of nodes $\mathcal{SIB}$, whose MBRs are disjoint with the query range, i.e., the siblings along the path to the root node, just like in an MHT. The $\mathcal{VO}$ contains $\mathcal{MCS}$ and $\mathcal{SIB}$, allowing the client to recompute the hash-value of the root and verify it against the provided signature. Given a correct $\mathcal{MCS}$, the client can calculate the requested aggregate by summing over the aggregate values in $\mathcal{MCS}$. Since aggregate values are used to eventually calculate the root-hash, the server cannot risk tampering with those values. The $\mathcal{VO}$ also contains the siblings of each node in $\mathcal{MCS}$, so the client can verify that sibling MBRs are disjoint from the specified query range [5].

2.3 Finding a Centerpoint

The *center* of a set of n points $\mathcal{P}$ in $\mathbf{R}^d$, $d \geq 1$, is the maximal subset of R^d where *any* intersecting hyperplane divides the points into two half-spaces that contain at *least* $\lceil n/d+1 \rceil$ points each; that closed convex set is also called a k-hull with $k = n/d+1$. A centerpoint is any member of the *center*. For $d = 1$, the centerpoint is the median. To compute a centerpoint of a point set $\mathcal{P}$ in linear time, we prune points in $\mathcal{P}$ so that the center of the resulting point set $\mathcal{R}$ is a subset of the center of $\mathcal{P}$, until we arrive at a set of at most 10 points, whereupon we find a center-point by any brute-force approach [3]. This algorithm relies on two pruning methods, the simplest being to find a triplet of points in $\mathcal{P}$ that satisfies the following lemma:

Lemma 2.1. *Let $\mathcal{Q}$ be any four points in $\mathcal{P}$ such that the (closed) convex hull of $\mathcal{Q}$ contains the $(\lceil |P|/3 \rceil - 1)$-hull of $\mathcal{P}$. Then the center of $((\mathcal{P} - \mathcal{Q}) \cup q)$ is a subset of the center of $\mathcal{P}$, where q is a* Radon *point of $\mathcal{Q}$, i.e., a point in the intersection of the convex hulls of two sets into which $\mathcal{Q}$ can be partitioned by Radon's theorem.*

This lemma removes a quadruple while adding its Radon point. If a point $r \in \mathcal{Q}$ is in the open convex hull $HULL(\mathcal{Q})$, then r is the Radon point of $\mathcal{Q}$; if no points in $\mathcal{Q}$) are contained in $HULL(\mathcal{Q})$, a new Radon point r is added to $\mathcal{P}$ at the intersection of two lines, each intersecting a distinct pair of points in $\mathcal{Q}$; if all four points in $\mathcal{Q}$ are on a line, the Radon point r is either the 2nd or 3rd point on that line. To find a quadruple that satisfies this lemma, the centerpoint algorithm defines four open half-spaces L, U, D and R, each containing less than $\lceil |P|/3 \rceil - 1$ points of $\mathcal{P}$ and their intersections containing approximately $\lceil |P|/3 \rceil - \lceil |P|/4 \rceil = \lceil |P|/12 \rceil$ points; we prune the points in these intersections. To find L, we focus on the point p that has the minimum abscissa (x-coordinate), and draw $|\mathcal{P}| - 1$ lines, one from p to each other point $p' \in \mathcal{P}$; the line having the $(\lceil |P|/3 \rceil - 1$-th largest slope defines half-space L, which contains less than $\lceil |P|/3 \rceil - 1$ points. U and D are found with respect to L, based on a generalized version of the Ham Sandwich cut algorithm, via lines that divide the points in L into a ratio of 1:3, and all other points into a ratio of 3:5; R is found similarly, with U playing the role of L [3]. In the following, we use $\mathcal{N}_p$ to denote $\lceil |P|/3 \rceil$ [3].

3 AUTHENTICATING CENTERPOINTS

To authenticate a centerpoint query, we need to prove to the client that half-spaces and pruned points are chosen correctly without involving the data owner. In one dimension there is a total order, which allows us to authenticate a median by authenticating a range count query, i.e., proving that the number of points before the median is equal to that after [5]. Unfortunately, this method cannot be extended to two dimensions, where there is no total order. We resort to a more elaborate solution using MR-trees and constructing a $\mathcal{VO}$ out of multiple $\mathcal{VO}$s, each corresponding to a pruning step. Algorithms 1 and 2 present the authentication and verification phase of our solution, respectively, which we describe below.

Algorithm 1 AuthCenterPointQuery(AAR, range) → VO

```
AAR ← Perform desired subset queries based on range
pruneVOs ← []
while (AAR.leaves > 10):
    AAR, pruneVO ← AuthPrune(AAR)
    pruneVOs.push(pruneVO)
finalVO ← authPoints(AAR)
centerVO ← {pruneVOs, finalVO}
return centerVO
```

Algorithm 2 Verify(δ, VO, size, fanout) → bool

```
for pruneVO in VO.pruneVOs:
    for authedHPlane in pruneVO.authedHPlanes:
        if !verifyHPlane(authedHPlane, δ):
            return false
    for prune in pruneVO.prunes:
        valid, δ ← VerifyPrune(prune, δ)
        if !valid
            return false
    for point in finalVO:
        if !verifyPoint(point, δ):
            return false
    return true
```

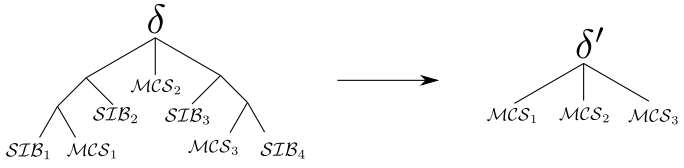


Figure 1: A range query returns 3 nodes in $\mathcal{MCS}$ and 4 nodes in $\mathcal{SIB}$, used to recompute the signed root digest δ and a new root digest δ' using the verified $\mathcal{MCS}$s.

3.1 Prerequisites

We use an *AAR* created and signed by the data owner, as well as an additional signature on information on the total number of points. We also allow for authenticated center-point queries on a subset of the original points: the client performs a range query that returns the minimum covering set of the specified range, $\mathcal{MCS}$, and all siblings, disjoint to the range, along the path to the root, $\mathcal{SIB}$. The client then recomputes the digest using $\mathcal{MCS}$ as Figure 1 shows.

The client trusts the $\mathcal{MCS}$ contents, as *AAR*-tree nodes contain their MBRs, which cover the requested range. Client and server compute a new sub-tree that contains the descendants of nodes in $\mathcal{MCS}$. MBRs, aggregate values, and hash values of parent nodes are computed from their children, so client and server can agree on a new *AAR*-root for the range query. As the client already trusts the $\mathcal{MCS}$ contents verified against δ, it subsequently verifies against δ'.

3.2 Authenticating Half-spaces

The server must prove that each of the open half-spaces L, U, D, and R contains at most $\mathcal{N}_p - 2$ points, hence the other side of the half-space has at least $|P| - \mathcal{N}_p + 2$ points; we focus on those points using a depth-first traversal of the *AAR*-tree. In Figure 2, points P_{1-8} are separated into two half-spaces. A client performing a COUNT query cannot verify that R_2 only contains points on the right side, as the dark gray area lies on the other side; to prove that there are more than $|P| - \mathcal{N}_p + 2$ points on the right side, the server's *AAR*-tree traversal proceeds to R_5 and R_6, which form the minimum set of nodes that cover all the points in the open half-space, $\mathcal{MCS}$, and returns the authenticated COUNT result, along with a $\mathcal{VO}$ containing $\mathcal{MCS}$ and $\mathcal{SIB}$, the minimum set of nodes that cover all the points disjoint to the half-space. In each pruning step, the client uses these sets to verify the digest and checks that the nodes in $\mathcal{MCS}$ are fully contained in the open half-space and the count is at least $|P| - \mathcal{N}_p + 2$, for all open half-spaces, and recalculate the new root digest of the *AAR*-tree after removing the pruned points, which is to be used in the next pruning step. The data owner does not need to sign the new root digests, as the client verifies them to be valid based on the provided $\mathcal{VO}$s; the owner only needs to sign the initial root digest δ and the total number of points $|P|$.

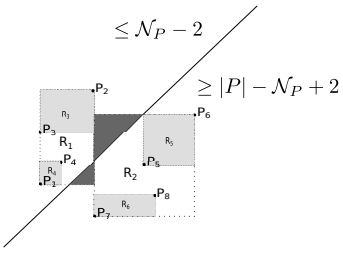


Figure 2: MBRs indexing half-spaces.

3.3 Authenticating Points

Having verified the half-spaces, the client can verify that the quadruplets to be pruned were chosen correctly. For a quadruplet to be correctly chosen each of the points has to be contained in an intersection, as indicted with the black dots in Figure 3. The server constructs a $\mathcal{VO}$ for a COUNT query on multiple rectangular ranges that cover the points in those intersections. Such a $\mathcal{VO}$ has only leaves in its $\mathcal{MCS}$. The client checks that all nodes in $\mathcal{MCS}$ have aggregation value 1 (assuming no points share the same position) and reconstructs the digest of the *AAR*-tree. The server does not need to provide the radon points that substitute quadruples, as the client can compute them. Still, server and client must choose the exact same quadruplets; we ensure such agreement using label order. Algorithms 3 and 4 summarize these steps.

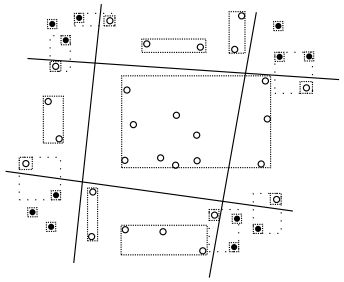


Figure 3: $\mathcal{VO}$ for authenticating quadruplets; black points lie in an intersection, each covered by a single MBR.

Algorithm 3 AuthPrune(AAR) → AAR, VO

```
HPlanes ← Compute half-spaces L, U, D, and R from AAR
authedHPlanes ← []
for HPlane in HPlanes:
    authHPlanes.push(AuthHPlane(HPlane, AAR))
prune ← Authenticate corners with MCS consisting of only leaves
LUs, LDs, RUs, RDs ← Distribute corner points based on label
while (LUs, LDs, RUs, RDs > 0):
    LU, LD, RU, RD ← Select and remove a quadruple from LUs,LDs,RUs,RDs
    radon ← Calculate radon from LU,LD,RU,RD
    AAR.remove(LU, LD, RU, RD)
    AAR.add(radon, index(LU))
pruneVO ← {authedHPlanes, prune}
return AAR, pruneVO
```

Algorithm 4 VerifyPrune(prune, δ, fanout) → bool, δ

```
valid ← Verify 'prune' as a COUNT query
if !valid
    return false, null
LUs, LDs, RUs, RDs ← Distribute corner points based on label
allCorners ← Checks whether LUs,LDs,RUs,RDs are fully contained each corner
if !allCorners:
    return false, null
AAR ← constructTree(prune, fanout)
if AAR.digest≠ δ:
    return false, null
while (LUs, LDs, RUs, RDs > 0):
    LU, LD, RU, RD ← Select and remove a quadruple from LUs,LDs,RUs,RDs
    radon ← Calculate radon from LU,LD,RU,RD
    AAR.remove(LU, LD, RU, RD)
    AAR.add(radon, index(LU))
return true, AAR.digest
```

3.4 Final Stage

The final stage of the center-point algorithm ends up with a set P' of at most 10 points such that CENTER(P') ⊆ CENTER(P); CENTER(P') is found by any brute-force approach. It suffices to authenticate

and send P' to the client. If the client is interested in finding a point in P within CENTER(P'), they can query the database using any authenticated spatial query method. The final $\mathcal{VO}$ consists of multiple $\mathcal{VO}$s, as illustrated in the following:

$$\mathcal{VO}_{center} = \{\mathcal{VO}^*_{prune}, \mathcal{VO}_{final}\}$$
$$\mathcal{VO}_{prune} = \{\mathcal{VO}_L, \mathcal{VO}_U, VO_D, \mathcal{VO}_R, \mathcal{VO}_q\}$$
$$\mathcal{VO}_H = \{H, \mathcal{MCS}_H, \mathcal{SIB}_h\}, H \in \{L, U, D, R\}$$
$$\mathcal{VO}_q = \{\mathcal{MCS}_{LU}, \mathcal{MCS}_{LD}, \mathcal{MCS}_{RU}, \mathcal{MCS}_{RD}, \mathcal{SIB}\}$$
$$\mathcal{VO}_{final} = \{\mathcal{MCS}, \mathcal{SIB}\}$$

where $(\cdot)^*$ represents zero or many. $\mathcal{VO}_{center}$ contains a collection of $\mathcal{VO}_{prune}$, each for a single pruning step, and $\mathcal{VO}_{final}$, which is a COUNT $\mathcal{VO}$ that verifies the points of the fully pruned set P', where $|P'| \leq 10$. Each $\mathcal{VO}_{prune}$ contains a $\mathcal{VO}$ for each half-space and one that proves the existence of pruned quadruples. Each $\mathcal{VO}_H$, where H is any of the four half-spaces, contains the half-space that has to be verified, so that the client can check that MBRs of nodes in $\mathcal{MCS}_H$ are contained in H, and those in $\mathcal{SIB}_H$ are disjoint from H. $\mathcal{VO}_q$ authenticates pruned quadruplets, with one $\mathcal{MCS}$ for each intersection, allowing the client to confirm Radon points.

3.5 Analysis

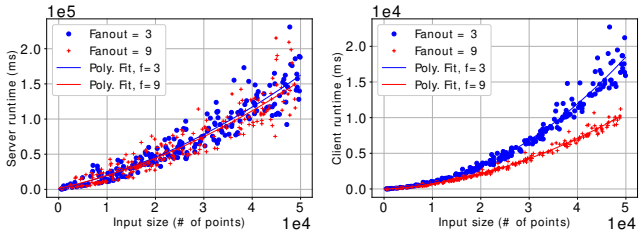
Our scheme requires counting points in each of the 4 open half-spaces during each pruning step, done via COUNT queries [5] based on an *AAR*-tree on all the points in the database. The worst-case scenario is that points are distributed in a way that necessitates traversing the tree to the leaf level for each point. Thus, the running time of our authentication scheme depends on the distribution of points and the state of the *AAR*-tree. In the worst case, the server goes through all internal nodes in the *AAR*-tree, taking $O(n \log n)$ time. The size of the point-set after each pruning decreases by at least $|P|/4$ points [3], hence the time it takes to run our authentication scheme with input size n is amortized to $O(n \log n)$. Likewise, in the worst-case the client is given a $\mathcal{VO}$ containing n leaf nodes and performs as many operations as there are internal nodes in the *AAR*-tree, taking $O(n \log n)$ time. The size of $\mathcal{VO}$s transferred between the server and client include, in the worst case, all leaf nodes, hence the total space required for each pruning step is $O(n)$. Besides, the server stores the entire tree in $O(n \log n)$ space; the client needs $O(n)$ storage to recompute new digests. Thus, space complexity is $O(n \log n)$ on the server and $O(n)$ on the client.

4 EXPERIMENTAL STUDY

We implemented our solution[1] in Go, while the external library[2] used to find centerpoints is in Python. We achieved interoperability between Go and Python using a web API. As the endpoints are REST (representational state transfer) compliant, the system is modular. We implemented an *AAR*-tree [5] to bulk-load the data and run the authentication. To facilitate the introduction of Radon points, we used `floats` to store coordinates, and included a global ϵ value that determines precision and thereby granularity. Our implementation uses web-based inter-process communication (IPC) only at the server, so as to communicate with the external library to find half-spaces. The client-server communication is done by passing $\mathcal{VO}$s as messages to the function executed by the client. We generate point-sets of a random sizes in a space of range [-50, 50] in both dimensions, distributed uniformly at random over the entire space.

4.1 Results

We measured runtime vs. input size with fanout of the *AAR*-tree set to 3 and 9. Figures 4a and 4b show our results.



(a) Server runtime vs. input size. (b) Client runtime vs. input size.

Figure 4: Runtimes on server and client.

As expected, the server runtime scales by $O(n \log n)$ wrt size; fanout has no big effect. Runtime varies a lot on larger inputs, as the query runs on a different point-set in each test. The time-complexity on the client-side is also $O(n \log n)$. The results suggest that the runtime slightly worsens with the lower fanout of 3; this is due to the recomputation of digests benefiting from larger fanout.

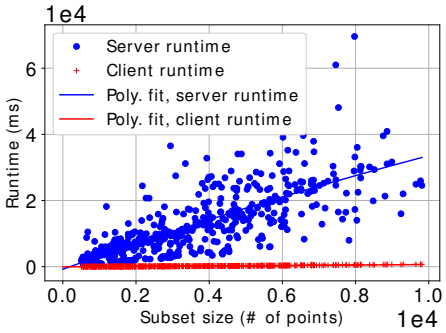


Figure 5: Memory and real-world data.

Figure 5 shows performance on real-world data, with fanout 3; the data contains roads in the USA[3], and the experiments find a centerpoint of a subset of this data having a random size $n \in [500, 10000]$. These results show that our solution is applicable on real-world data. Runtime on the client scales significantly better than that on the server, as we expect. The server runtime varies as a result of choosing a random subset of the data on each test.

5 CONCLUSION

We devised a method to authenticate centerpoint queries on any subset of a finite planar sets of points. Our mechanism produces a minimal set P' (with a size of at most 10), whose center is a subset of that of the original set P. By authenticating P', the client can compute the centerpoint based on P' using any brute-force approach. Our experiments verified that this mechanism works on real-world data as well as uniformly random point-sets. In the future, we will consider authentication over multidimensional synopses [4] and natural spatial joins [9].

[1] Available at https://github.com/1234JohnDoe/Auth-Centerpoint
[2] https://github.com/JianiLi/centerpoint
[3] https://drive.google.com/file/d/1pd5eGPWJUzzv8B26lYTz2rOkGgOuRM-I/view

REFERENCES

[1] Qian Chen, Haibo Hu, and Jianliang Xu. 2013. Authenticating top-k queries in location-based services with confidentiality. *Proc. VLDB Endow.* 7, 1 (2013), 49–60.

[2] Antonin Guttman. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*. 47–57.

[3] S. Jadhav and A. Mukhopadhyay. 1994. Computing a centerpoint of a finite planar set of points in linear time. *Discrete & Computational Geometry* 12, 3 (1994), 291–312.

[4] Panagiotis Karras and Nikos Mamoulis. 2008. Hierarchical synopses with optimal error guarantees. *ACM Trans. Database Syst.* 33, 3 (2008), 18:1–18:53.

[5] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. 2010. Authenticated Index Structures for Aggregation Queries. *ACM Trans. Inf. Syst. Secur.* 13, 4 (2010), 32:1–32:35.

[6] Ralph C. Merkle. 1987. A Digital Signature Based on a Conventional Encryption Function. In *CRYPTO*. 369–378.

[7] Dimitrios Papadopoulos, Stavros Papadopoulos, and Nikos Triandopoulos. 2014. Taking Authenticated Range Queries to Arbitrary Dimensions. In *CCS*. 819–830.

[8] Yin Yang, Stavros Papadopoulos, Dimitris Papadias, and George Kollios. 2008. Authenticated indexing for outsourced spatial databases. *The VLDB Journal* 18, 3 (2008), 631–648.

[9] Man Lung Yiu, Nikos Mamoulis, and Panagiotis Karras. 2008. Common Influence Join: A Natural Join Operation for Spatial Pointsets. In *ICDE*. 100–109.