A Data Structure Supporting Exclusion Persistence Range Search

Stuart A. MacGillivray* Bradford G. Nickerson†

Abstract

We present a new type of query on spatiotemporal data, termed "exclusion persistence". When bulk updates are made to stored spatial data, all previous versions remain accessible. A query returns the most recently observed data intersecting the query region, while permitting the exclusion of any subset of previous versions in areas where data overlaps. We propose several solutions to this new problem defined on a set of N points. For an axis-aligned rectangular exclusion persistence query, we give a 2-dimensional linear-space data structure that, after m updates, answers the query in $O(\sqrt{\frac{mN}{B}}+m^2+\frac{K}{B})$ I/Os, where B is the number of points fitting in one disk block, and K is the number of points in range.

1 Introduction

In most large scale earth observation systems, data is gathered in short duration surveys of significantly sized regions. To maintain up-to-date representations of the covered area, successive surveys are performed. Surveys normally cover different areas, resulting in overlap as shown in Figure 1. These types of surveys usually observe a massive number of data points for each update. Queries are expected to return data intersecting the query region from the most recent survey where overlap occurs. The survey areas can be represented as overlapping polygonal regions, where the newest data in overlapping regions replaces older data.

When searching for up-to-date information, various search options are possible. If the data is obtained from multiple sources, we may wish to exclude all information obtained from an unreliable source, or restrict search only to those data sources we trust. A search that excludes all data in a specific time period is useful if uncorrectable errors are known to occur in data observed during that period. Lastly, we may simply wish to examine the region as it was in some period before the present time.

Given point data that maps to d-dimensional space, we consider search on N data points added over the course of m updates. Normally, $N \gg m$ and we can assume that $m < \log N$. This is a reasonable assumption under all known earth observation applications; for instance, LIDAR surveys can contain tens of millions of points per update [5].

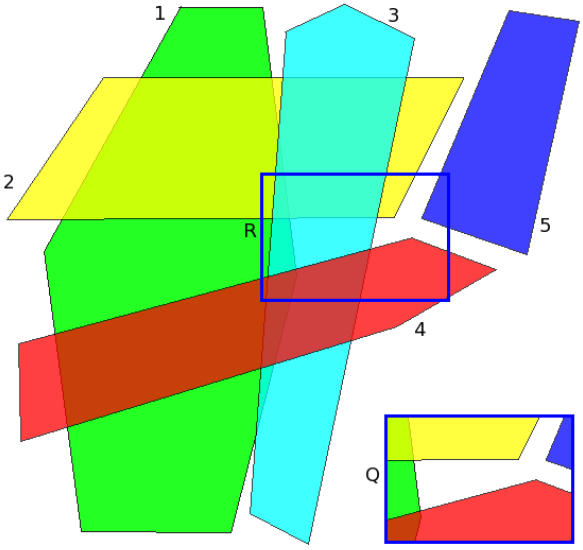

Figure 1: Overlapping updates and a query rectangle R, showing $Q = (R, 5, \{3\})$ where the query time $t_q = t_5$, the excluded points index set $T_e = \{3\}$, and $d = 2$. Areas of overlap have multiple distinct accessible versions.

2 Exclusion Persistence

These problems do not map cleanly to any existing model of persistence as described by Tarjan et al [4]. To resolve the issue, we define the idea of "exclusion persistence". If we assume that new data in a spatial region replaces older data in the same region, we can apply exclusion persistence to support the types of searches described above.

A data structure supporting exclusion persistence maintains updates independent of one another, and searches performed on the structure can omit any subset of past updates from consideration. Formally, we have m sets of d-dimensional data points $S_1 ... S_m$, contained in bounding regions $B_1 ... B_m$ and added to the structure at times $t_1 ... t_m$, respectively.

*Faculty of Computer Science, University of New Brunswick, t172q@unb.ca

†Faculty of Computer Science, University of New Brunswick, bgn@unb.ca

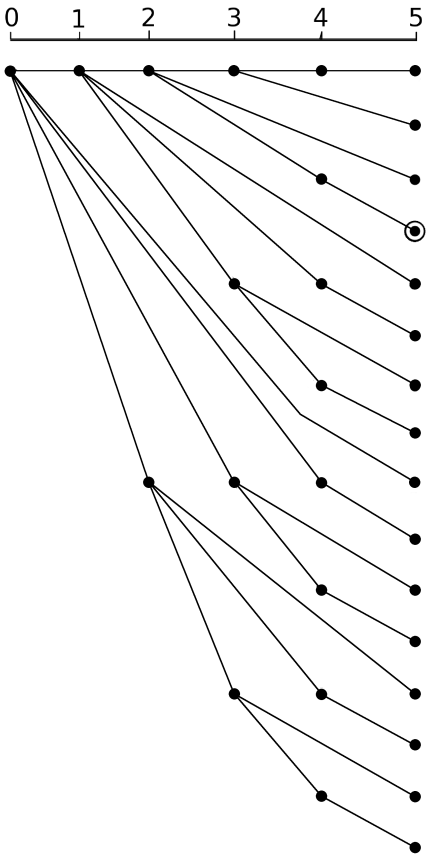

Figure 2: The 2^m possible versions in exclusion persistence with $m = 5$ updates, with the circled version matching the example query from Figure 1.

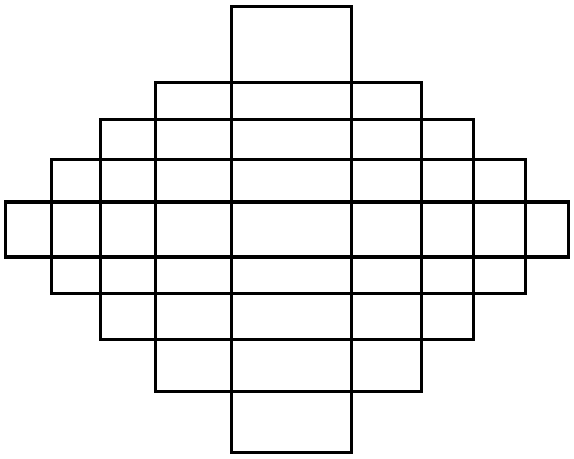

Figure 3: Subregions for rectangular regions grow quadratically in the worst case, as shown with $m = 5$.

We make the simplifying assumption that data is observed at a single time epoch, representing the time period the update covers. Given a convex query region R to search, and a set T_e of time epoch indices whose matching data sets are excluded from consideration, we define an exclusion persistence range search as follows.

An exclusion persistence range query is defined as $Q = (R, q, T_e)$, which asks to find all points intersecting R whose time epoch $t_i \leq t_q$, and whose time epoch index $i \notin T_e$. The result of such a query is the union of data contained in queried regions C_i over all non-excluded updates i, i.e. $\bigcup_{\substack{i=1 \\ i \notin T_e}}^{q} C_i \cap S_i$, where $C_i = (B_i \cap R) \setminus (B_i \cap (\bigcup_{\substack{j=i+1 \\ j \notin T_e}}^{q} C_j)$.

This search will return all data in the valid sets added on or before t_q intersected by R, returning only the newest data (whose time epoch $t_i \leq t_q$, and whose time epoch index $i \notin T_e$) in areas where bounding regions overlap. As any set may be excluded for a given query, however, older data sets are still accessible and must be maintained.

Informally, the problem can be summarized as follows. We have m updates, each of which is a spatial region B_i containing a set of data points S_i added to our structure at a time t_i. We assume that new data takes precedence over old data, meaning that in areas where multiple regions overlap, we only return data from the newest set. A structure not supporting any form of persistence could thus simply delete all data falling within a new region B_i before adding S_i to the structure. While partial persistence allows us to ignore all data newer than the query time, an exclusion persistence search has the ability to ignore any of the updates at the searcher's discretion.

While storing B points per disk block in the I/O model [7], is there a linear space data structure using $O((\frac{N}{B})^{\frac{d-1}{d}} + \frac{K}{B})$ I/Os to answer an exclusion persistence range search returning K points? If not, what tradeoffs are possible? Henceforth, we restrict the query region R to an axis-aligned rectangle.

3 Naïve Solutions

Two naïve solutions are possible. The first is to simply store each update completely independently, search all appropriate data sets independently, and remove excluded data after the fact. In such a scenario, efficiency can be obtained by storing each update in an optimal linear-space structure such as the Priority R-Tree [3]. While this means that only linear storage space is required overall, searches will be highly redundant. As we store no information about where overlap occurs, we return data from old updates even if they are completely covered by newer ones. In the worst case, each update region completely covers the previous, leaving us with a worst case of $O(m(\frac{N}{B})^{\frac{d-1}{d}} + \frac{mK}{B})$ search I/Os.

The second naïve approach is to use full persistence to store every possible combination of sets as a different version, and simply search the appropriate one. This gives the desired search I/Os, but to cover all possible combinations the storage requirement increases by a factor of 2^{m-1}. This is evident by virtue of the fact that m updates can be combined to produce 2^m possible versions, as shown in Figure 2, and that any given data point will be part of half those combinations. For N points, this requires $O(2^{m-1}\frac{N}{B})$ disk blocks of storage, as each data point is stored in 2^{m-1} versions.

4 Convex Regions Approach

Neither of the naïve solutions is acceptable; naïve linear storage has highly redundant searches, and full persistence is wasteful of space no matter what the value of m is. An improved solution is to store the updates or sets of points independently, and split data into subsets based on spatial overlap of the updates. We do this to avoid the problem of naïve linear storage, splitting old updates according to how newer ones cover them. We first consider the restricted case when all updates have a bounding region consisting of a convex polygon; our results are dependent on an upper bound f on the number of faces per polygon.

Let a 2-d structure solving this problem be defined as follows. In memory, we store an index of the m regions covering the m data sets. We also store, for each polygonal subregion created from the intersection of these m regions, a stack of pointers to data structures on disk. Each structure is guaranteed to contain data covering the subregion, meaning that only the topmost non-excluded structure of the stack must be searched. The structures on disk are any linear-space data structure supporting range search in $O((\frac{N}{B})^{\frac{1}{2}} + \frac{K}{B})$ I/Os (e.g. a Priority R-Tree [3]). A range search determines (in main memory) which sub-regions are intersected, follows the pointer from the stack to the most recent non-excluded data structure in each intersected sub-region, and performs an I/O-efficient range search independently on each structure.

As the sub-regions are distinct, there is no redundancy in this range search. The overall search cost is reasonable for small m; an exclusion range search requires $O((\frac{mN}{B})^{\frac{1}{2}} + m^2 + \frac{K}{B})$ I/Os in the worst case, as shown in Theorem 2. The m^2 term will not dominate unless $m > \sqrt[3]{\frac{N}{B}}$. Our initial proof in Theorem 2 requires that the regions be axis-aligned rectangles, but Theorem 6 extends the result to f-sided convex polygonal regions.

Lemma 1 *Given $x_i \in \mathbb{R}_{>0}$ such that $\sum_{i=1}^{m} x_i = N$, then $\sum_{i=1}^{m} \sqrt{x_i}$ has a maximum value of $\sqrt{mN}$ when $x_i = \frac{N}{m} \forall i$.*

Proof. Define a function $C = \sum_{i=1}^{m} \sqrt{x_i}$. As $x_i \in \mathbb{R}_{>0}$ and $\sum_{i=1}^{m} x_i = N$, this can be written as $C = \sum_{i=1}^{m-1} \sqrt{x_i} + \sqrt{N - \sum_{i=1}^{m-1} x_i}$. For all x_i where $1 \leq i \leq m-1$, we take the partial derivative $\frac{\partial C}{\partial x_i} = \frac{1}{2\sqrt{x_i}} - \frac{1}{2\sqrt{N - \sum_{i=1}^{m-1} x_i}}$. We find the critical values of C by setting each partial first derivative to zero, resulting in simultaneous equations $x_i = (N - \sum_{i=1}^{m-1} x_i) \forall i$. A unique solution exists where $x_i = \frac{N}{m} \forall i$. The second derivatives are $\frac{\partial^2 C}{\partial x_i \partial x_j} = -\frac{1}{4\sqrt{N - \sum_{i=1}^{m-1} x_i}^3} \forall i \neq j$, and $\frac{\partial^2 C}{\partial x_i^2} = \frac{-1}{4\sqrt{x_i}^3} - \frac{1}{4\sqrt{N - \sum_{i=1}^{m-1} x_i}^3}$. As these second derivatives are negative for all values of x_i, this critical value is a global maximum. In short, C is maximized when $x_i = \frac{N}{m} \forall i$, and has a value of $\sum_{i=1}^{m} \sqrt{\frac{N}{m}} = m\sqrt{\frac{N}{m}} = \sqrt{mN}$. □

Theorem 2 *Assuming we have N 2-dimensional data points from m updates, where each update is covered by an axis-aligned rectangle, there exists a data structure that can perform a rectangular exclusion persistence range search in $O((\frac{mN}{B})^{\frac{1}{2}} + m^2 + \frac{K}{B})$ I/Os.*

Proof. With m overlapping rectangles, the maximum number of intersections between their subregions and a horizontal or vertical line is $2m-1$; the number of subregions intersecting each side of the query rectangle R is, therefore, linear in m. Figure 3 shows that in the worst case the number of subregions is quadratic in m, meaning that at most $O(m^2)$ subregions can be covered by R. For each of these $O(m^2)$ subregions, we store a stack of pointers to linear space optimal I/O-efficient structures on disk storing data covering that subregion. At most one of these structures is searched for each subregion intersecting R. The geometric index structure describing each of the $O(m^2)$ subregions is assumed to be in main memory, so the I/O cost to find the intersected subregions is zero.

Our range search is an aggregate of range searches over the intersected subregions, with each subregion i containing x_i points. The subregions intersecting the sides require $O(\sqrt{\frac{x_i}{B}} + 1 + \lfloor \frac{K_i}{B} \rfloor)$ I/Os to return K_i points. Subregions entirely contained by the search require $O(1 + \lfloor \frac{K_i}{B} \rfloor)$ I/Os. The worst case occurs when all N points are distributed among the n_1 subregions intersected by the sides of R and none are in the n_2 subregions contained by R (see the illustration in Figure 4). This gives a total cost of $O(\sum_{i=1}^{n_1} \sqrt{\frac{x_i}{B}} + n_1 + n_2 + \frac{K}{B})$ I/Os to return K points, and $\sum_{i=1}^{n_1} x_i = N$. Lemma 1 shows

that $\sum_{i=1}^{n_1} \sqrt{\frac{x_i}{B}}$ has a maximum value of $\sqrt{\frac{n_1 N}{B}}$, as in the worst case, $\sum_{i=1}^{n_1} x_i = N$. As n_1 is $O(m)$ and n_2 is $O(m^2)$, the total search cost is $O((\frac{mN}{B})^{\frac{1}{2}} + m^2 + \frac{K}{B})$ I/Os. □

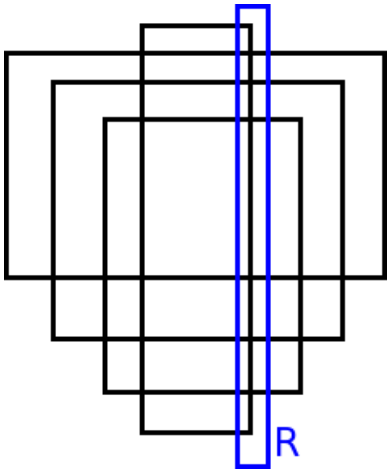

Figure 4: A worst case query on $m = 4$ subregions intersecting $O(m)$ subregions with the potential to be empty. Here, $n_1 = 12$ and $n_2 = 0$.

While the rectangular case is useful, we can generalize. We still require that the query region R be an axis-aligned rectangle, but the regions defining the data sets can be any convex polygon of at most f sides.

Theorem 3 *The intersection of m overlapping convex polygons with at most f sides will produce at most fm^2 subregions.*

Proof. By the definition of convexity, a line intersecting a convex polygon can intersect at most two of its sides. When adding a new f-sided polygon to a set of $i-1$ polygons, each side of the ith polygon can intersect at most two sides from each previous polygon. These intersections partition each new side into $2(i-1)+1 = 2i-1$ line segments. Each of these line segments can belong to at most two subregions, and the addition of the ith polygon can create at most one new subregion for each segment. The addition of the ith polygon to the set therefore creates at most $f(2i-1) = 2if - f$ subregions. A set of m polygons therefore has an upper bound of $\sum_{i=1}^{m} 2if - f = 2f(\frac{m(m+1)}{2}) - mf = fm^2 + mf - mf = fm^2$ subregions. □

Lemma 4 *A straight line passing through a set of m convex polygons can intersect at most $2m-1$ subregions.*

Proof. A straight line passing through a convex polygon begins intersecting that polygon at one point, and stops intersecting it at another. A straight line passing through a set of m polygons therefore has at most $2m$ intersections where it begins or stops passing through a polygon. The straight line intersects a new subregion if and only if one of those $2m$ intersections occurs, and the last such intersection denotes where the line stops intersecting any of the m polygons; as such, it can intersect at most $2m-1$ subregions. □

Lemma 5 *The sides of a rectangular query region R intersect $O(m)$ subregions of a set of m convex polygons.*

Proof. Lemma 4 shows that a straight line can intersect at most $2m-1$ subregions from a set of m convex polygons. Each side of R is a straight line, and as such the sides of R can only intersect at most $8m-4$ subregions. □

Theorem 6 *Assuming we have N 2-dimensional data points from m updates, where each update is covered by a convex region with at most f sides, there exists a data structure that can perform a rectangular exclusion persistence range search in $O((\frac{mN}{B})^{\frac{1}{2}} + fm^2 + \frac{K}{B})$ I/Os.*

Proof. Lemma 5 shows that $O(m)$ subregions will intersect the sides of the query region R, and Theorem 3 shows that $O(fm^2)$ subregions can be contained by a query. The proof of Theorem 2 therefore applies to the general case, giving the desired I/O bound. □

5 Algorithms

Our convex regions solution to exclusion persistence range search consists of a spatial partitioning of 2-dimensional space into subregions, with each subregion having a stack of pointers to spatial data structures on disk. Each pointer is given a time stamp denoting what update added its data. Space not covered by any update is treated as a subregion with an empty stack. Figure 5 illustrates the structure, demonstrating a simple insertion.

Algorithm 2 shows an implementation of exclusion persistence range search. The repeat-until loop finds the top non-excluded time epoch on the stack for each subregion. Lines 9-14 of Algorithm 2 are invoked at most once per subregion. We apply the entire query region R to each intersected subregion at line 13 for simplicity, as each structure only contains data within its subregion. The subregion shape could force a range search on highly clustered data, which is still linear space optimal. While our analysis of worst case range search requires axis-aligned rectangular query regions, Algorithm 2 can also be used with general polygonal queries. Algorithm 2 provides the correct exclusion persistence range search result by specifying T_e such that $i \in T_e \forall t_i > t_q$.

Algorithm 1: Insert(P,S,R,t,j)

Input : A convex regions data structure P, a set S of new data points to be inserted, a convex shape R that contains S, a time epoch t associated with S, the index j for time epoch t

Output: An updated convex regions data structure P that includes S

```
1  begin
2      Use polygonal differences to find all subregions P_i of P that intersect R;
3      for each subregion P_i ∈ P contained by R do
4          Search S for the points S_i that fall within P_i;
5          Keeping S_i in memory, delete the points in S_i from S;
6          Bulk-load S_i into a new linear-space data structure D_i stored on disk;
7          Push a pointer to D_i stamped with time (t, j) onto the stack for P_i;
8      for each subregion P_i ∈ P intersecting the edges of R do
9          Update P with new subregions P_k formed by the intersections of P_i and R;
10         for each pointer p in the stack for P_i, from the bottom to the top do
11             Follow the pointer p to its data structure D_i, storing the time stamp (t_i, ℓ) in memory;
12             for each new subregion P_k do
13                 Search D_i for the set of points S_k that fall within P_k; Bulk-load S_k into a new linear-space data structure D_k stored on disk;
14                 Push a pointer to D_k stamped with time (t_i, ℓ) onto the stack for P_k;
15             Delete D_i;
16         for each new subregion P_k do
17             Search S for the points S_k that fall within P_k;
18             Keeping S_k in memory, delete the points in S_k from S;
19             Bulk-load S_k into a new linear-space data structure D_k stored on disk;
20             Push a pointer to D_k stamped with time (t, j) onto the stack for P_k;
```

Our data structure has an update cost dependent on the data structures used for the subregions. Our analysis requires a linear-space spatial data structure supporting range search in $O(\sqrt{\frac{N}{B}} + \frac{K}{B})$ I/Os, such as the bkd-tree [6] or Priority R-Tree [3]. From Theorem 2.4 of [3] we know that such a structure can be bulk-loaded in $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os. This leads to the following theorem:

Algorithm 2: Search(P,R,T_e)

Input : A convex regions data structure P, a query region R, a set T_e of time epoch indices to be excluded

Output: A set K of points found by the exclusion persistence query

```
1  begin
2      Use polygonal differences to find all subregions P_i of P that intersect R;
3      for each subregion P_i ∈ P intersecting R do
4          repeat
5              Pop the top pointer p_k from the stack for P_i;
6              Let (t_k, j) = the time stamp for p_k;
7          until j ∉ T_e or the stack for P_i is empty;
8          if j ∉ T_e then
9              Follow p_k to its data structure D_k;
10             if P_i is contained by R then
11                 Add all points in D_k to K;
12             else
13                 S_i = a range search on D_k over R;
14                 Add the points in S_i to K;
15         Push all popped pointers p_k back onto the stack for P_i with their respective time stamps (t_k, j);
```

Theorem 7 *Using Algorithm 1, the total insertion cost of m updates into a convex regions data structure requires $O(\frac{mN}{B}\log_{M/B}\frac{N}{B})$ I/Os, where N is the total number of points after all updates.*

Proof. In the worst case, an update S that is inserted into a structure that previously contained $N-|S|$ points intersects subregions containing $O(N)$ of those points. Each of the intersected subregions must be split, requiring new data structures to be created. Loading the affected structures will require $O(N)$ I/Os. Structures must also be created for the $|S|$ points from the new update, for a total of j structures. Each new structure D_i, containing x_i points, can be bulk-loaded in $O(\frac{x_i}{B}\log_{M/B}\frac{x_i}{B})$ I/Os. As $\sum_{i=1}^{j} x_i \leq N$, creating j structures containing a total of $O(N)$ points requires $O(\frac{N}{B}\log_{M/B}\frac{N}{B})$ I/Os, which dominates the cost. □

Figure 5 illustrates the result of the insertion process described in Algorithm 1. A new update is added to the set, intersecting one previously existing subregion

and the uncovered space. This results in data from the first update being repartitioned, and part of that data covered by a portion of the new update. Some of the resulting subregions are non-convex, but the previously proven search bounds apply regardless.

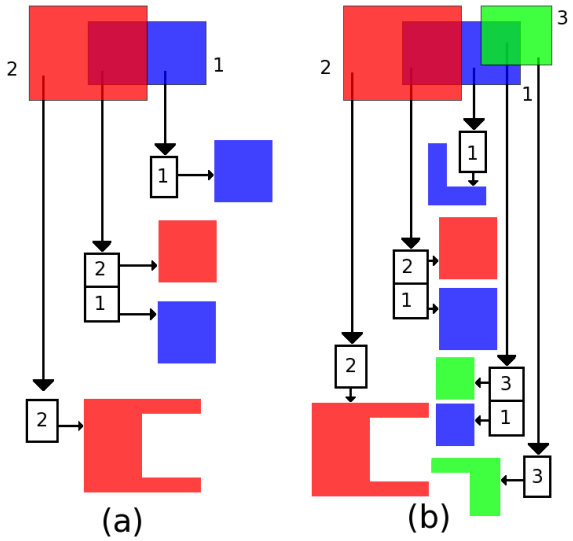

Figure 5: Subregions arising from $m = 2$ and $m = 3$ intersecting regions, with the data stored in each version of each subregion.

6 Conclusion

We have presented the exclusion persistence problem in spatiotemporal queries, along with a 2-dimensional solution. For a set of N data points collected over a series of m updates, where each update is bounded by a convex region of at most f sides, our linear space solution requires $O((\frac{mN}{B})^{\frac{1}{2}} + fm^2 + \frac{K}{B})$ I/Os in the worst case to perform an exclusion persistence range search, with K points reported in range. While relaxing the space requirement on the subregion structures could improve search cost as shown by Afshani et al [1][2], this would lead to the overall storage requirement becoming non-linear. Experimental validation of the data structure remains to be done.

Several interesting open problems remain. Is a linear space data structure supporting exclusion persistence range search in $O((\frac{N}{B})^{\frac{1}{2}} + \frac{K}{B})$ I/Os possible? What worst case search complexity (in the I/O model) is possible for general convex query regions R in place of rectangles? Is there a non-trivial linear space data structure storing d-dimensional points that can efficiently answer exclusion persistence search queries? What I/O search complexity arises when data updates are described by non-convex boundaries?

References

[1] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting in three and higher dimensions. In *FOCS*, pages 149–158. IEEE Computer Society, 2009.

[2] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In J. Snoeyink, M. de Berg, J. S. B. Mitchell, G. Rote, and M. Teillaud, editors, *Symposium on Computational Geometry*, pages 240–246. ACM, 2010.

[3] L. Arge, M. de Berg, H. Haverkort, and K. Yi. The priority R-tree: A practically efficient and worst-case optimal R-tree. *ACM Trans. Algorithms*, 4(1):1–30, 2008.

[4] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989.

[5] D. Latypov. Estimating relative lidar accuracy information from overlapping flight lines. *ISPRS Journal of Photogrammetry and Remote Sensing*, 56(4):236 – 245, 2002.

[6] O. Procopiuc, P. Agarwal, L. Arge, and J. Vitter. Bkd-tree: A dynamic scalable kd-tree. In T. Hadzilacos, Y. Manolopoulos, J. Roddick, and Y. Theodoridis, editors, *Advances in Spatial and Temporal Databases*, volume 2750 of *Lecture Notes in Computer Science*, pages 46–65. Springer Berlin / Heidelberg, 2003.

[7] J. S. Vitter. External memory algorithms and data structures. *ACM Comput. Surv.*, 33(2):209–271, 2001.