Approximate stabbing queries with sub-logarithmic local replacement

Ivor Hoog v.d.[1] **and Maarten Löffler**[1]

1 **Dept. of Inform. and Computing Sciences, Utrecht University, the Netherlands**
[i.d.vanderhoog|m.loffler]@uu.nl

Abstract

In this work we present the key ingredients to construct a linear-size data structure that stores a set of spheres or axis-parallel hypercubes in $\mathbb{R}^d$ and supports what we define as 2^m-approximate stabbing queries in logarithmic time and local replacement in sub-logarithmic time, if the regions not overlap "too much". This work uses known techniques such as quadtrees and marked-ancestor trees and introduces a new concept: *key facets* in a d-dimensional quadtree. We show the intuition behind how this new concept can help us perform fast (approximate) stabbing queries with sub-logarithmic local replacement if the dimension d and the approximation variable m are constant. For a detailed description of the query algorithm and for proofs of correctness we refer to our full version.

1 Introduction

An important and well-studied problem in Computational Geometry is the problem where one is given a set $\mathcal{B}$ of n regions in $\mathbb{R}^d$ and needs to find the regions in that set that contain a given query point q. Queries of this form are called *stabbing queries* and in this work we focus on the reporting variant where we have to return the regions that contain q. In a static environment, it is common to make a subdivision of the space based on the regions. Given q, we then quickly find the cell in the subdivision that contains q. Well-known subdivision methods are R-trees, quadtrees and (after applying a duality transformation) k-d trees [4]. Most current research on this topic focuses on the dynamic version of the problem where one wants to maintain a set of regions subject to stabbing queries and adding, removing or translating regions. These dynamic stabbing queries appear as a sub-problem in many day-to-day applications such as GPS tracking, handling data imprecision and data analysis. In certain applications a special kind of update called *local replacement* (Definition **??**) [5] is frequently performed. Intuitively, a local replacement replaces a region by another region similar to it. For example: the ever-increasing uncertainty radius between GPS updates or the moving action radius of an ambulance could both be modeled using local replacement. We assume that the unit itself has a *finger* to its location in the data structure and we want to use the finger to update the data structure and any auxiliary data structures.

In this paper regions in $\mathbb{R}^1$ will be defined as compact intervals. Regions in $\mathbb{R}^d$ will be spheres and axis-parallel hypercubes. A local replacement does not "change too much" in the set of regions and because of this, it is conjectured [5] that it should be possible to perform such a replacement strictly faster than the traditional logarithmic time required for deleting and inserting a region. Löffler *et al.* in [4] present a data structure that supports stabbing queries in logarithmic time and local replacement in sub-logarithmic time for disjoint regions in $\mathbb{R}^1$ and $\mathbb{R}^2$. In [3] Khramtcova and Löffler extend this approach by allowing regions in $\mathbb{R}^1$ to overlap. In this work, we improve the data structure in [3, 4] to work for overlapping regions in $\mathbb{R}^d$ if regions are spheres or axis-aligned hypercubes and if queries are not exact, but what we call 2^m-approximate for constant m. The question of whether exact logarithmic

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

stabbing queries can be performed in logarithmic time with sub-logarithmic replacement remains open.

1.1 Problem definition

In this paper we are given a set $\mathcal{B}$ of n spheres or axis-aligned hypercubes in $\mathbb{R}^d$. We measure the size of any region $B \in \mathcal{B}$ as its diameter. We assume that all regions are contained within an axis-aligned bounding hypercube $\mathcal{K}$. In [4] Löffler *et al.* demanded that the regions in $\mathcal{B}$ are disjoint. In [3] Khramtcova and Löffler relaxed this constraint by introducing limited ply: the **ply** of a set $\mathcal{B}$ of regions in $\mathbb{R}^d$ is the maximum over all points $q \in \mathbb{R}^d$ of the number of $B \in \mathcal{B}$ that contain q. With this restriction from [3] on the set of regions $\mathcal{B}$ we want to perform stabbing queries subject to sub-logarithmic local replacement. Local replacement is defined as replacing a region B_1 with a region B_2 such that the two regions are ρ-similar for constant ρ:

▶ **Definition 1.** Given two regions $B_1, B_2 \in \mathcal{B}$ and a $\rho \geq 1$, we call B_1 and B_2 **ρ-similar** if there exists a region $A \subset \mathbb{R}^d$ with $|A| \leq \rho \min\{|B_1|, |B_2|\}$ such that $B_1, B_2 \subset A$.

▶ **Definition 2.** We call replacing a $B_1 \in \mathcal{B}$ with B_2 a **local replacement** if B_1 and B_2 are ρ-similar for a constant ρ.

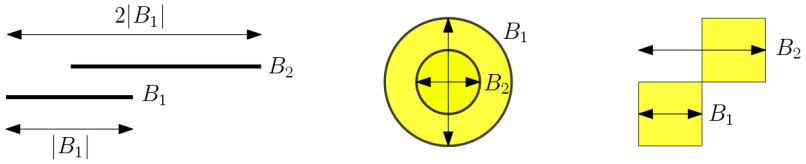

Figure 1 Three examples of a region B_1 and a 2-similar region B_2.

To perform this local replacement we use what we later define as a **level query**. In the full version of this paper we show that level queries cannot be solved in sub-logarithmic time in $\mathbb{R}^d$, so we decided to relax the requirements for stabbing queries and replace exact stabbing queries with approximate queries. Intuitively, we approximate each region B with a smaller inner region. Approximate stabbing queries return all regions B whose inner region $I(B)$ contains q and possibly regions B whose outer region $B \backslash I(B)$ contains q, whilst ply is still defined on the outer region B. The area between the outer and inner region could be seen as a "buffer" that safeguards the inner region. We define our 2^m-approximate stabbing queries using this concept of inner and outer region, the time bounds of our operations depend on the approximation constant m.

▶ **Definition 3.** For any convex region B, we define the **inner region** with respect to m as a map I_m which takes a region and produces its 2^m-approximate inner region. Given a region B, $I_m(B)$ is the scaled down version of B with $|B| = (1 + 2^{-m})|I_m(B)|$ with the center of $I_m(B)$ on the center of B.

▶ **Definition 4.** A 2^m-approximate query on a set of regions $\mathcal{B}$ is a query that given a point $q \in \mathbb{R}^d$ returns all $B \in \mathcal{B}$ for which q is contained in $I_m(B)$ and might return other regions which contain q but does not return regions which not contain q.

Our main result is the following theorem:

▶ **Theorem 5.** *Given a set $\mathcal{B}$ of either spheres or axis-aligned hypercubes in $\mathbb{R}^d$ with ply k and fixed m we can construct a data structure that*

- *takes* $\mathcal{O}(dn)$ *space,*
- *supports regular insertion and deletion of elements and a* 2^m*-approximation of the stabbing queries in* $\mathcal{O}(3^d k \frac{\log(n)}{\log(\log(n))} + \log(n) + m)$ *time,*
- *supports local replacement in* $\mathcal{O}(2^{(d-1)m} k \frac{\log(n)}{\log(\log(n))})$ *time.*

2 Preliminaries

Quadtrees. We always work within an axis-aligned bounding hypercube $\mathcal{K}$ with finite size. A **quadtree** T on $\mathcal{K}$ is a hierarchical partition of $\mathcal{K}$ into smaller axis-aligned **cells**. Given $\mathcal{B}$ we build a dynamic compressed quadtree T that stores $\mathcal{B}$ with the following storing condition: A cell C stores a region B if C is the largest (possible) cell in T such that B contains C *and* C contains the center point of B. In this extended abstract we assume for any set $\mathcal{B}$ we can compute quadtree T storing $\mathcal{B}$ with the following three properties (see [2] for details):

- The tree takes $\mathcal{O}(dn)$ space.
- If for any two cells $C_1, C_2 \in T$ their corresponding hypercubes are ρ-similar for constant ρ, we can walk from C_1 to C_2 in constant time using pointers.
- For each point q we can locate the smallest cell in T that contains q in logarithmic time.

Marked-ancestor trees. Suppose we are given a tree T with nodes (cells in our case) and a fixed simple directed path π over T (this path does not have to follow pre-existing edges in the tree, it is just an arbitrary simple path through the cells in T) where some cell in the path can be marked. In this model we want to support the following query for any cell C: Which is the first marked cell which comes after cell C in the path?. We also want to support updates where cells can be marked or unmarked and inserted into or deleted from the path. This is known as the *marked successor problem*. This problem is solved in [1] with the use of marked-ancestor trees. These trees support the marking and unmarking of cells in the path in $\mathcal{O}(\frac{\log(n)}{\log(\log(n))})$ time. Each marked-ancestor tree with a path π in T allows for what they call the *firstmarked* query:

▶ **Definition 6.** Given a connected path π in T, we can construct a marked-ancestor tree over T such that for each cell $C \in \pi$, **firstmarked(**C**,** π**,** T**)** gives the first marked cell in π starting from C.

The firstmarked query can be solved in $\mathcal{O}(\frac{\log(n)}{\log(\log(n))})$ time [1]. This paper will make extensive use of the firstmarked query. The marked successor problem is a more generic version of the marked-ancestor problem: "Given a cell C in a tree T, which is the first marked node that is an ancestor of C in T?". Observe that the marked-ancestor problem can be solved with a firstmarked query, we call this a **marked-ancestor query**.

3 Intuition and key facets

Let $\mathcal{B}$ be a set of closed and bounded intervals in $\mathbb{R}^1$. The goal of this section is twofold: we introduce a new concept called **key facets** and we use this new concept to introduce a data structure that supports exact stabbing queries in $\mathcal{B}$ in logarithmic time and local updates in sub-logarithmic time. The results in the remainder of this paper are already known: this abstract is an adaption of the data structure and methods used in [3] so that it works with key facets. However, this work contains the basic data structure that we use for 2^m-approximate stabbing queries in $\mathbb{R}^d$ in the full version [2].

Assume that we have a quadtree T that stores $\mathcal{B}$ and that for any query point q we can find the quadtree cell $C \in T$ which contains q in logarithmic time. Then any region B that

contains q either covers C, or intersects C from the left or the right. Let R be the set of all regions that have their center point to the right of C. Let $B_1 \in R$ be the region with the left-most left endpoint of the regions in R. If any region in R contains q then B_1 must also contain q. We can symmetrically construct the set L. Because the ply is limited by k, we only have to find the k left-most/right-most regions in R and L respectively to find all regions that can contain q. As in [3] we find these regions in $\mathcal{O}(k\frac{\log(n)}{\log(\log(n))})$ time using two marked-ancestor trees T_L and T_R and $2k$ marked-ancestor queries.

In [3] the authors obtained the marked-ancestor trees T_L and T_R by looking at which cells contained the left-most and right-most endpoints of the regions B and by marking those cells in T_R and T_L respectively. If we want to extend these results to $\mathbb{R}^d$ we must ask what an equivalent structure would look like in higher dimensions: given q we need to identify a constant number of *directions* and somehow find the k closest regions to q per direction. To do this we introduce **key facets**.

If T is a quadtree in $\mathbb{R}^d$ then its cells are hypercubes in $\mathbb{R}^d$. Each of these cells C have $3^d - 1$ d'-dimensional facets with $d' < d$. Given T we intuitively define the **key facet set** Ξ as the abstract notion of all these facets. For example: in $\mathbb{R}^2$ the set Ξ contains the abstract notion of four vertices and four edges.

▶ **Definition 7.** Let $\mathcal{K}$ be a d-dimensional bounding box and $\mathcal{P}(\mathcal{K})$ be the infinite set of all potential quadtree cells in $\mathcal{K}$. We then define the **key facet set** Ξ as a set of maps (**key facets**) $\chi :: \mathcal{P}(\mathcal{K}) \to \mathbb{R}^d$ where χ projects each cell C to the same d'-dimensional facet of that cell. By **the key facets of a cell** C we mean the set $\Xi(C) := \{\chi(C) \mid \chi \in \Xi\}$.

Let $\chi \in \Xi$ be any key facet and let $C \in T$ be an arbitrary quadtree cell. Let p be the center of C. If $\chi(C)$ is a point then the half-line l through p and $\chi(C)$ is unique. Otherwise let l be a half-line through p and $\chi(C)$ perpendicular to $\chi(C)$. Observe that for any cell $C \in T$, l has the same direction and observe that we can therefore rotate $\mathbb{R}^d$ such that l aligns with the x-axis. This rotation allows us to create a partial order on the cells in T for each $\chi \in \Xi$ by ordering the cells on their lowest x-coordinate after the rotation. Figure 2 shows this rotation where the cells in the partial order are $A <_\chi B =_\chi C <_\chi D <_\chi E$.

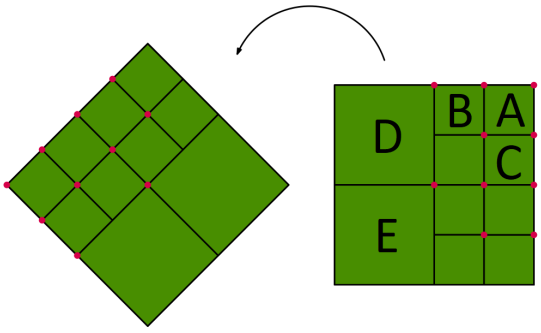

Figure 2 A quadtree in $\mathbb{R}^2$, We can choose χ as the general concept of the "top right vertex". This Figure then contains a rotated quadtree T, and the projection $\chi(T)$ as red dots.

▶ **Definition 8.** Let T be an arbitrary quadtree in $\mathbb{R}^d$ and $\chi \in \Xi$ be a key facet. We then define a **facet path** π_χ as any simple path through T with two properties: the linear order of this path is an extension of the partial ordering given by χ and if $C_1 =_\chi C_2$ but $C_2 \subset C_1$ then C_1 is in the path before C_2.

▶ Observation 1. In $\mathbb{R}^d$ there exists a facet path for each key facet. In $\mathbb{R}^1$ the facet path is unique and it is the pre-order traversal of the quadtree in both directions.

4 The solution in $\mathbb{R}^1$

Marked-ancestor trees together with key facets provide the tools we need to perform approximate stabbing queries in $\mathbb{R}^d$ with sub-logarithmic local replacement. However the algorithm and construction in $\mathbb{R}^d$ is beyond the scope of this paper. We instead introduce the data structure for $\mathbb{R}^d$ and show that with this data structure we can perform exact stabbing queries in $\mathbb{R}^1$ with the same time bounds as in [3] with sub-logarithmic local replacement.

Let the ply of $\mathcal{B}$ be bounded by k. In our data structure we maintain a quadtree T that stores $\mathcal{B}$ as specified in the Preliminaries. Together with T we maintain $k|\Xi| + 1$ marked-ancestor trees over T as follows: for each $\chi \in \Xi$ we maintain k marked-ancestor trees or *levels* denoted as a family of trees $\{T^i_\chi\}_{i\leq k}$. In this family of trees each tree T^i_χ will get the same facet path π_χ. Apart from these $k|\Xi|$ trees we maintain another marked-ancestor tree denoted by T^*. For each region $B \in \mathcal{B}$, we mark all the cells C that B intersects in one of the marked-ancestor trees. If B is stored in C, it marks C in T^*. Else it marks C based on the following condition:

▶ Condition 1. For any family of marked-ancestor trees $\{T^i_\chi\}$ apart from T^*, a cell C is marked by a region B in T^i_χ if:

1. $\chi(C)$ is the highest-dimensional key facet of C that B intersects *and*
2. i is the largest i such that there is a descendant C_d of C which is marked in T^{i-1}_χ and B intersects $\chi(C_d)$.

Stabbing queries in $\mathbb{R}^1$: With this marking condition we can use our marked-ancestor trees to solve any exact stabbing query in logarithmic time. Given a point q, we find the smallest cell C in T that contains q in logarithmic time. We then query each marked-ancestor tree T^i_χ with the *firstmarked* query from C with the path π_χ, this takes $\mathcal{O}(\log(n)+k\frac{\log(n)}{\log(\log(n))})$ time. In the remainder of this section we prove that the returned regions are the only regions that can contain $q \in \mathbb{R}^1$.

▶ **Lemma 9.** *Let C and C_a both be marked in T^i_χ in the same level i by a region B and B_a respectively. Let χ be the map to rightmost point of each cell. If C_a is an ancestor of C then the leftmost point of the region B_a must lie to the right of the leftmost point of the region B. A symmetric property holds if χ is the rightmost point.*

Proof. We prove this by contradiction: assume that the leftmost point of B_a lies to the left of B. Then clearly all C' where B intersects $\chi(C')$ are also intersected by B_a and thus also $\chi(C)$. If $i < k$ then because $\chi(C)$ is intersected by B_a, B_a should have been stored in T^{i+1}_χ and not T^i_χ. If $i = k$ then per definition, B intersects the key facet $\chi(C_d)$ of a descendant C_d of C. C_d is therefore per definition marked in T^i_χ by a region B_d. By our earlier observation, B_a must also intersect B_d in $\chi(C_d)$. We continue this argument all the way down to T^1_χ and see that we violate a ply of k. ◀

▶ **Lemma 10.** *Given a point $q \in \mathbb{R}^1$ contained in a cell C, if C has a lowest-marked ancestor C_1 marked in T^i_χ for any i, χ by an interval B_1 then B_1 is the only region marking an ancestor of C in T^i_χ that can contain q.*

Proof. Let T^i_χ be an arbitrary marked-ancestor tree in $\mathbb{R}^1$. Then χ is either the left-most or right-most vertex of a cell. Assume that we have found the lowest marked ancestor of the cell C in the marked-ancestor tree T^i_χ, the cell C_1 marked by a region B_1. Then B_1 either contains the query point q or does not. If B_1 does not contain q then Lemma 9 demands that any ancestor of C_1 marked in T^i_χ is marked by a region that reaches less far than B_1

does and so any other region marking an ancestor of C_1 cannot reach q. If B_1 does contain q then any region marking a higher ancestor of C in T_χ^i that reaches q must also intersect $\chi(C_1)$ and should therefore have marked the ancestor in T_χ^{i+1} or violate the ply of k. ◀

The result of this lemma is that in $\mathbb{R}^1$ with constant ply we can solve stabbing queries in logarithmic time with a marked-ancestor query in each marked-ancestor tree (the lowest-marked ancestor in T^* always contains q).

The level query in $\mathbb{R}^1$: Assume we want to replace a region B_1 with a region B_2 such that B_1 and B_2 are ρ-similar and that we have a *finger* to the cell C_1 that stores B_1. Because B_1 and B_2 are ρ-similar we can use at most $\mathcal{O}(\rho)$ pointers to reach the cell C_2 that should store B_2 (See preliminaries). What remains is to update the marked-ancestor trees in sub-logarithmic time. For that we define the **level query**.

▶ **Definition 11.** A **level query** checks for a given region B, cell C that B intersects and a family of marked-ancestor trees $\{T_\chi^i\}$ in which level i the region B marks C (if any).

In $\mathbb{R}^1$ the level query can be solved for each $\{T_\chi^i\}$ by just incrementally performing at most k firstmarked queries with the unique facet path π_χ. One can show that the result of that query gives the unique region that could "force" B to mark C in a higher level.

The solution in $\mathbb{R}^d$. We show in the full version [2] that in $\mathbb{R}^d$ the abstract level query has a lower bound of logarithmic time. This version also contains a more elaborate description of the data structure required to store and approximately query regions in $\mathbb{R}^d$. Specifically we introduce an extension of the marking Condition 1 and show how to perform 2^m-approximate stabbing queries and local replacements with the query times as specified in Theorem 5.

In this version and the full version we have demonstrated how the concept of key facets can give information about the closest regions that lie in a certain *direction* from a query point q. Given the cell that contains the query point, we can even provide this information in sub-logarithmic time. A future research direction could be to see if we can use key facets and marked-ancestor queries for sub-logarithmic visibility queries.

Acknowledgements. The authors would like to thank Elena Khramtcova for inspiring discussion of the problem. I.H. and M.L. were partially supported by the Netherlands Organisation for Scientific Research (NWO) through project no 614.001.504.

References

1 Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 534–543. IEEE, 1998.

2 Ivor Hoog v.d. and Maarten Löffler. Approximate stabbing queries with sub-logarithmic local replacement (full version). *arXiv to appear.*

3 Elena Khramtcova and Maarten Löffler. Dynamic stabbing queries with sub-logarithmic local updates for overlapping intervals: Proc. 12th international computer science symposium in russia. *Computer Science–Theory and Applications*, 10304, 2017.

4 Maarten Löffler, Joseph A. Simons, and Darren Strash. Dynamic planar point location with sub-logarithmic local updates. In *13th Int. Symp. Algorithms and Data Structures (WADS)*, pages 499–511. Springer Berlin Heidelberg, 2013. full version: arXiv preprint arXiv:1204.4714.

5 Yakov Nekrich. Data structures with local update operations. *Algorithm Theory–SWAT 2008*, pages 138–147, 2008.