Universiteit Utrecht

Master thesis COSC

Includes 28 illustrations

Dynamic Stabbing Queries with Sub-logarithmic Local Replacement for Overlapping Regions in $\mathbb{R}^2$

Author
I.D. van der Hoog
ICA-4141741

Supervisors
Dr. Maarten Löffler
Prof. Dr. Marc Van Kreveld

August 23, 2017

Abstract

We present an approximation data structure to maintain a set of *fat* regions in $\mathbb{R}^2$ subject to fast insertions and deletions of the regions, stabbing queries, local replacement. Local replacement is a new concept where we replace a region with a region that is "similar" to the original region. We elaborate on earlier result obtained by Löffler, Strash and Simons which shows that it is possible to have a linear size data structure that supports insertions, deletions and stabbing queries in logarithmic time and local replacement in sub-logarithmic time, if the regions are disjoint. We also discuss another earlier result from Löffler and Khramtcova where they present a data structure that supports these operations for overlapping intervals in $\mathbb{R}^1$, where the time bounds for our desired operations depend on the maximum overlap (*ply*) of the intervals. We prove that this approach cannot be extended to $\mathbb{R}^2$ and continue with introducing a data structure that supports approximate queries. Our data structure can support ϵ-approximate stabbing queries, for $\epsilon = \frac{1}{2^m+1}$ in $\mathcal{O}(m + \log(n))$ time and local replacement in $\mathcal{O}(2^m \log(\log(n)))$ time. Lastly we present a theorem that says that no reduction proof from the problem of stabbing queries with sub-logarithmic local replacement in $\mathbb{R}^2$ to binary search can exist. Our approximation bounds show that a logarithmic lower bound for stabbing queries with local replacement is likely, but our results show that proving this lower bound through a reduction to binary search or heap operations is infeasible.

1 Introduction

An important and well-studied problem in Computational Geometry is the problem where one is given a set of n regions in $\mathbb{R}^d$, and needs to find the regions in that set that all contain a given query point. Queries of this form are called *stabbing queries.* There are many variants of stabbing queries. There is the decision variant, which only reports whether or not the query point intersects at least one region and the counting variant where we only how many regions our query point intersects. We will focus on the reporting variant, where we have to return the specific ID's of the regions that contain our query point. In a static environment, it is common to make a subdivision of the plane based on the regions. Given a query point q, we then try to quickly find the cell in the subdivision that contains the point q. Well-known subdivision methods are R-trees, quadtrees and (after applying a duality transformation) KD-trees [3] [2]. 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 both stabbing queries and updates such as removing or translating regions. These dynamic stabbing queries appear as a sub-problem of many geometric problems: a natural application of this problem would be the tracking of moving regions. GPS signals are not continuously updated but rather sent out in (often large) time intervals. Suppose you have n entities each with their own GPS signal. Each time unit after an update, you become more uncertain about their current position. The unknown position of the entity could be described as an uncertainty region in $\mathbb{R}^d$ which keeps growing until the next update. Stabbing queries could be used to answer questions like: which entities could be in contact right now? Similarly, dynamic stabbing queries can be used to track moving entities with an action radius (e.g. ambulance service or the police). A stabbing query could be used to answer a question such as: which entities are in range to aid person X at location Y? A related problem that also makes use of stabbing queries is dealing with data imprecision [2] (see also references therein). One way to model an imprecise point is to keep track of a region of possible locations of the point. These regions could be used to solve what is known as the identity query: Given a query point, is there a point in the data structure that is equal to the query point? When the points in the data structure are imprecise, the answer to this question may have three possible values: "certainly", "possibly", or "certainly not." The answer can be determined with a stabbing query on the uncertainty regions. The last application we mention is data-analysis (or popularly, *big data*). Classes of related data often get represented as a multi-dimensional region, and multi-class classification then becomes equivalent to a stabbing query.

In this paper we focus on closed and bounded regions in $\mathbb{R}^d$ with $d \in \{1, 2, 3\}$. Regions in $\mathbb{R}^1$ will be defined as compact intervals and when we extend the data structure for regions in $\mathbb{R}^2$, we will mostly define regions as either closed disks or closed squares. In certain applications, for example the earlier mentioned applications involving moving data, a special kind of update is frequently performed, called local replacement by Nekrich [5]. Intuitively, a local replacement replaces a region by another region similar to it: the new region has roughly the same size and location as the old region. In our previous example the ever-increasing uncertainty radius of GPS and the moving action radii could both be modeled using local replacement. A local replacement does not "change too much" in the set of regions and because of this, it is suggested [1] that it should be possible to perform such replacement strictly faster than the traditional logarithmic time for deleting and inserting a region. Löffler *et al.* in [1] present a data structure that supports stabbing queries in logarithmic time and local replacement in sub-logarithmic time for disjoint *fat* regions[1] in $\mathbb{R}^1$ and $\mathbb{R}^2$. In [2] they extend this approach by allowing regions in $\mathbb{R}^1$ to overlap. In this paper we aim to improve the work done in both papers. We extend the data structures of both papers to try to support logarithmic stabbing queries and sub-logarithmic local replacement for overlapping closed disks and squares in $\mathbb{R}^2$. We prove that a specific query used for local replacement in [1] and [2] (which we will call the **level query**) can never be done in sub-logarithmic time if regions are allowed to overlap in $\mathbb{R}^d$ with $d > 1$. Later sections then relax the constraints on the traditional stabbing queries. We replace exact stabbing queries with what we will call λ-approximate stabbing queries. We will provide a data structure and a query algorithm that can support these λ-approximate stabbing queries in logarithmic time, with sub-logarithmic local replacement

[1] The formal definitions for a fat region and quadtrees are provided in the Preliminaries (Section 2).

for constant λ.

Data structures and intersecting regions. The most trivial solution to the dynamic stabbing query problem is the solution where one simply stores the set of regions and iterates over all the regions and checks for each region if they contain the query point. If the regions have a constant complexity and if we have n regions, this trivial approach has $\mathcal{O}(n)$ storage and stabbing queries take $\mathcal{O}(n)$ time. For faster solutions, we need a data structure. The classical form of this problem is in $\mathbb{R}^1$ where regions are closed and bounded intervals. Many data structures exist for querying intervals such as the well-known R-trees, priority search trees, interval skip lists and interval trees [3]. In higher dimensions one could use R-trees, quadtrees or KD-trees to solve this problem. Löffler *et al.* present a data structure in [2] for storing and querying a set of disjoint *fat* regions in $\mathbb{R}^1$ and $\mathbb{R}^2$ with the use of quadtrees. A natural extension of the data structure would be to allow regions to not be disjoint. However, if we would allow arbitrary intersections between regions, we would make it much harder for all the traditional data structures to realize an efficient solution: recall that data structures such as quadtrees and R-trees subdivide the plane into cells based on the given set of regions. A stabbing query then looks at the cell that contains the query point q. If there is no bound on the number of overlapping regions, then there is no bound on the number of cells that might contain the query point q. All the regions contained in these $\mathcal{O}(n)$ cells may or may not contain the point q and this can create problematic edge cases where the query spends $\mathcal{O}(n)$ time steps regardless of output size. We therefore want some bound on how much regions can intersect and overlap. The easiest way to model this would be to demand that each region can only intersect one other region. In practice however, this constraint is nearly as restrictive as demanding that regions must be disjoint:

Assume that we have a set of regions $\mathcal{B}$ that we randomly want to place in a bounding box $\mathcal{K}$. Let one region $B_1 \in \mathcal{B}$ have $\frac{3}{4}$'th the size of the bounding box and let the other $n-1$ regions be small and randomly place B_1. If we demand that all the regions must be disjoint, then we can place $n-1$ regions in only $\frac{1}{4}$'th of the bounding box. Now demand that all regions can only overlap one other region, then we can pick one B_2 to be more 'freely placed' and intersect B_1 but the remaining $n-2$ regions can still only be placed in $\frac{1}{4}$'th of the bounding box. This shows that allowing a limited number of intersections between regions does not make you 'gain' much. This paper instead follows the approach presented in [1] and poses restrictions on how many regions can overlap in a single point (later defined as **ply**).

The layout of this thesis. Section 2 of this paper contains the preliminaries where we state our problem definition and provide the definition of the auxiliary data structures. In Section 3 we describe an adjusted version of the stabbing query data structure in [1] and [2] for storing and querying disjoint regions in $\mathbb{R}^2$. In Section 4 we try to extend the data structure in [1] to work for regions in $\mathbb{R}^2$ with limited ply. In this section we elaborate on the problems that arise with such an extension and we prove a problematic lower bound on the runtime of an essential part of local replacement, the **level query**. In Section 5 we will introduce λ-approximate stabbing queries as a relaxation of the exact stabbing query requirements and will introduce a data structure that uses these λ-approximate stabbing queries to get an ϵ-approximation with $\epsilon = \frac{1}{2^m+1}$ that supports stabbing queries in $\mathcal{O}(m + \log(n))$ time and local replacement in $\mathcal{O}(2^m \log(\log(n)))$ time. Lastly Section 6 consists out of possible future work, we sketch conjectured properties of a reduction proof for our problem and we prove that given these properties, a reduction proof from our stabbing query problem to *lowest number* can never exist.

2 Preliminaries.

2.1 Problem definition.

The goal of this research is to construct a data structure that can store a set of either closed squares or disks with limited ply, subject to logarithmic stabbing queries and sub-logarithmic local updates. We will use this subsection to elaborate on the formal definition of this problem. In this problem we are given a set

$\mathcal{B}$ of closed, bounded and *fat* regions in $\mathbb{R}^d$ with $|\mathcal{B}| = n$. We measure the size of any region $B \in \mathcal{B}$ with an adaption of the L_1 or L_∞ metric: $|B| = \max\{|l| \mid l \subset B, l \textit{ vertical or horizontal}\}$[2]. Note that with this definition of size, a horizontal line segment in $\mathbb{R}^2$ has size zero. This might seem contradictory, but to store regions we sub-divide the plane into axis-aligned cells until we find a cell that is covered by the region. That is why for each region only the purely horizontal and vertical size matters. We assume that all regions are contained within an axis-aligned bounding square $\mathcal{K}$ with $|\mathcal{K}| = K$. Early sections will demand that all the regions are disjoint. We will later try to expand the results of earlier sections when we relax this constraint by allowing $\mathcal{B}$ to be a set of regions with limited **ply**:

Definition 1. *The **ply** of a set $\mathcal{B}$ of regions is the maximum over all $q \in \mathbb{R}^d$ of the number of $B \in \mathcal{B}$ that contain q.*

In $\mathbb{R}^1$, regions are defined as closed and bounded intervals. In $\mathbb{R}^2$ there are more possible definitions for regions. In [2] Löffler *et Al* introduced the concept of a **fat** region which we will elaborate on later but we instead restrict us to having regions as closed disks and squares. Knowing what regions and ply are, the goal of this paper can then be summarized by the following conjecture:

Conjecture 1. *Given a set $\mathcal{B}$ of n regions in $\mathbb{R}^d$ with ply limited by some k and with $d \in \{1, 2\}$, we can construct a data structure that takes $\mathcal{O}(n)$ space that supports **stabbing queries** in $\mathcal{O}(\log(n))$, insertion and deletion in $\mathcal{O}(\log(n))$ time and **local replacement** in order $\mathcal{O}(\log(\log(n)))$ time.*

All that remains is to formalize the concepts of **stabbing queries** and **local replacement**:

Definition 2. *Given a point $q \in \mathbb{R}^d$, the **stabbing query** of q finds all $B \in \mathcal{B}$ that contain q.*

Definition 3. *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 B with $|B| \leq \rho \min\{|B_1|, |B_2|\}$ such that $B_1, B_2 \subset B$.*

Definition 4. *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.*

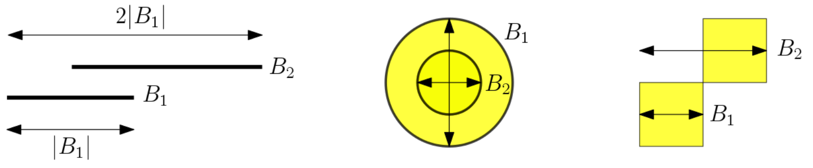

Fig. 1: Three examples of a region B_1 and a 2-similar region B_2.

2.2 Quadtrees.

We always work within an axis-aligned bounding box $\mathcal{K}$ with finite size. A **quadtree** T on $\mathcal{K}$ is a hierarchical partition of $\mathcal{K}$ into smaller axis aligned **cells**. Each node of the tree corresponds to a cell C which represents an axis-aligned area in $\mathbb{R}^d$. Each node either is a leaf of T or has 2^d equal sized children who partition its cell C. As an example we can see a quadtree on $\mathbb{R}^1$ as a binary tree of intervals. Similarly a quadtree on $\mathbb{R}^2$ can be seen as an actual quad-tree[3] and can be embedded in $\mathbb{R}^3$ by letting each node represent a square in $\mathbb{R}^2$ and by letting the **ancestor** relation induce **depth** on the third axis. The data structures that are presented in this paper will be making use of quadtrees over $\mathbb{R}^1$, $\mathbb{R}^2$ and $\mathbb{R}^3$. Each of the regions in $\mathcal{B}$ will be stored in exactly one cell of the quadtree. Each $B \in \mathcal{B}$ gets assigned a *center point* m. For intervals, axis-aligned squares and circles the definition of this center point is clear. For other regions, the formal

[2] Observe that this metric is also significantly different from the taxicab metric, often associated with the L_1 metric

[3] Because each node has four children.

definition of center point is presented in Section 3.1.1. A cell C will store a region B only if C is the largest cell that is covered by B **and** the cell C contains its center point m. For each cell C we define its **neighbors** as the adjacent cells in $\mathbb{R}^d$ of equal dimension. **Family neighbors** are neighbors with the same parent cell. In a quadtree over $\mathbb{R}^2$, a cell has family neighbors in both the x and y direction.

Definition 5. *Given a point $q \in \mathbb{R}^d$ contained in a cell C, we say that a region $B \in \mathcal{B}$ **reaches** q if B is not stored in C but does contain q.*

This definition only introduces new wording to express that a region B contains a point q. The intuition behind this definition is that the region B must stretch out over the edges of its storing cell to "reach" the point q.

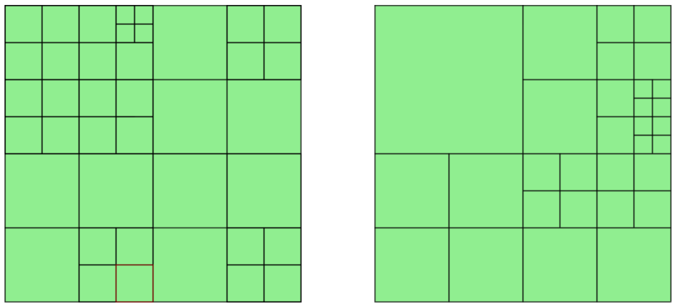
Fig. 2: A fully balanced (left) and smooth quadtree (right) in $\mathbb{R}^2$.

Construction and compression Quadtrees can be balanced, smooth or neither. There are several interpretations of these terms present in the field so we provide the definition of balanced and smooth that will be used in this paper: a tree is **balanced** if all subtrees are balanced and each pair of family neighboring subtrees differs in height by at most one. Recall that each cell in the quadtree represents a region in $\mathbb{R}^d$. We call a quadtree **smooth** if for each leaf cell, its sides are intersected by at most two neighbors. An equivalent way to define smooth is to say that each neighboring leaf differs at most in 1 in height. Figure 2 shows the difference between a balanced and a smooth quadtree in $\mathbb{R}^2$. Note that a balanced quadtree does not have to be smooth and vice versa, this can be seen in the difference between the upper two nodes in the figure. In this paper we use quadtrees to store regions, cells that store a region have to be covered by that region and so we add levels of depth to our quadtree (refine our quadtree) until we can store all regions. If $\mathcal{K}$ is sufficiently large and a region $B \in \mathcal{B}$ is very small, we need to add many levels of depth and a balanced quadtree could then become massive in size, so it is not clear whether our quadtree can be balanced. Even if the quadtree is not balanced it could easily have non linear size: Let $\mathcal{K}$ be the bounding box with size K and let there be a region $B \in \mathcal{B}$ with $|B| = K2^{-2^n}$. Then we would need an exponential amount of levels before we would have a cell size that could store that region B. To counter this problem, we use α-compression with a large constant α. Traditionally, an α-compressed cell C is a cell with only one child cell in the quadtree C' such that $|C'| \leq \alpha^{-1}|C|$. C then compresses the long simple path to C' into one cell. One downside of compression is that it is harder to walk over a compressed tree. A balanced tree can always have level links to walk from one cell to another but when the tree is compressed that is not the case anymore. We will combat this problem of walking over the tree later but to do that we already extend the definition of compressed node to include a second condition: if a cell C is a leaf that has a neighbor that differs more than $\log(\alpha)$ in depth from C, C is also compressed.

Definition 6. *We call a node C **compressed** if C either is a cell with only one child cell C' with $|C'| \leq \alpha^{-1}|C|$* or *C is a leaf with a neighboring cell with a depth difference at least $\log(\alpha)$.*

Observe that our α-compressed quadtree does not adhere to the rules normally associated with a quadtree: each node does not have 2^d children anymore and the leaves of the tree do not have to cover the bounding box

$\mathcal{K}$. Our quadtree has more differences to a conventional quadtree: we demand that between every non-empty leaf there are at most a constant amount of empty leaves. To combat these two leaf problems we introduce a **block node** $\overline{C}$. All block nodes must be disjoint and their union must cover the bounding box $\mathcal{K}$. Each leaf cell will become a block node and each compressed cell will get at most two block nodes, so that its child cell C' together with the two block nodes cover the compressed cell. Figure 3 shows an example of 8-compression where the node C_1 is compressed because it has a descendant C_1' and the node C_2 is compressed because it has a neighbor with a depth difference of at least $\log(8) = 3$. Theorem 1 in [6] states that you can use compression to construct a non-balanced or smooth quadtree to store n regions in $\mathbb{R}^d$ using linear space.

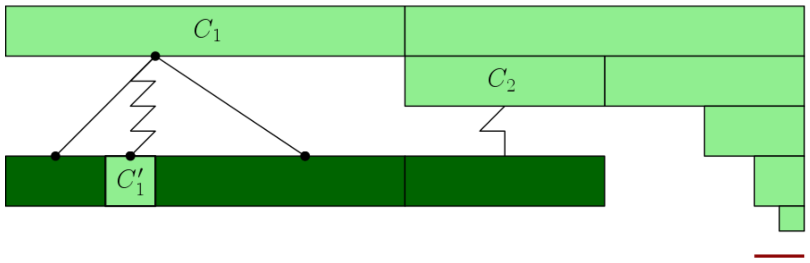

Fig. 3: Two 8-compressed with their block nodes (dark green)

M. Löffler *et Al.* in [1], [2] and we in this paper achieve ρ-replacement in sub-logarithmic time by walking through the quadtree with a constant walk towards the cell that should store the new region. If the quadtree is unbalanced, there are scenarios where the cells to actually walk over are not initialized and are instead missing or even compressed and so walking using level pointers becomes problematic. There are two options to combat this problem: the first option used in [2] is to maintain a **smooth** quadtree. The smoothness allows for a constant amount of level links between cells in the quadtree and thus allows traversal. The problem with this approach is that it is not proven yet that one can dynamically maintain a smooth quadtree to store $\mathcal{B}$ in linear time. Bern *et al.* in [6] only show that the minimal required smooth quadtree takes linear space. When you delete or insert a cell in the quadtree it could be that the next minimal quadtree differs in more than a linear amount of cells. Löffler *et Al.* in [2] look at this problem and are close to an approach where they maintain a slightly larger than minimal dynamic quadtree that is still linear in space but the proof of correctness is not finished. Instead we use a technique used in [1] that we will commonly use in the rest of this paper: **marked-ancestor trees**.

2.3 Marked-ancestor trees

Suppose we are given a simple path where some nodes in the path can be marked and we want to support the following query for any node C: "Which is the first marked node which comes after node C in the path?" and we also want to support updates where nodes 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 [7] with the use of marked-ancestor trees. Given a directed graph T (in our case our quadtree), the authors maintain what they call an ART-universe. They define a **heavy node** as a node with more than one child node and they partition T into connected smaller trees called micro-trees with $\mathcal{O}(\log(n))$ heavy nodes per micro-tree. They construct the concatenation of micro-trees in such a way that the path from any leaf cell C in T to the root, only traverses at most $\mathcal{O}(\frac{\log(n)}{\log(\log(n))})$ micro-trees. This partition of T allows for what they call the *firstmarked* query:

Definition 7. *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)** gives the first marked cell in π starting from C.*

The firstmarked query can be solved in $\mathcal{O}(\log(\log(n)))$ time within a micro-tree. 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?". The marked ancestor problem can easily be solved with the firstmarked query. Every node is part of a connected path to the root, so we make one marked-ancestor tree that includes all those paths. The firstmarked cell on an upwards path in the tree is then always the first marked ancestor of a cell C. When we construct our marked ancestor trees we assume that they already include these upwards paths and we will refer to these upwards firstmarked queries as **marked-ancestor queries**.

Stepping over the graph using marked-ancestor queries. Let each cell in $\mathbb{R}^d$ have N_d neighboring cells ($N_1 = 2$ and $N_2 = 8$). We create N_d marked ancestor trees Y_i over our quadtree with $i \in [N_d]$. Each compressed cell will mark its N_d neighboring cells in one Y_i tree, depending on which neighbor it is. The proofs in [1], [2] and this paper rely on the conjecture that you can make any constant virtual walk in constant time. In [2] the authors conjectured that this could be achieved with the use of dynamic balanced compressed quadtrees. It has not yet been proven that such quadtrees can exist and thus we adopted the reasoning in [1] where they try to make these virtual walks with the use of marked-ancestor queries. We will partly show you the current proof, and the edge case that makes the proof invalid:

Conjecture 2. *Given a pointer to a cell C, and a (possibly non-existent) target cell C' such that C and C' are ρ-similar for some constant ρ. If $\rho^k << \log(\alpha)$, we can always walk from C to C' using pointers in $\mathcal{O}(\alpha + \log(\log(n)))$ time.*

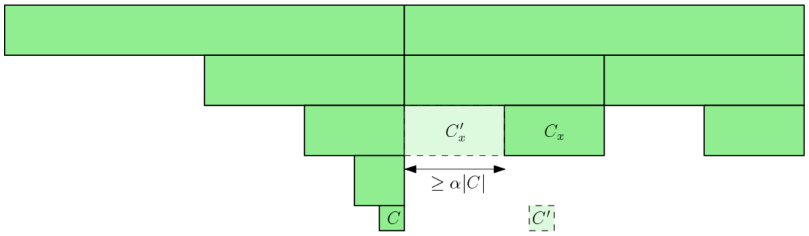

Fig. 4: Rho replacing C with C'

Incomplete proof. Assume a cell C and our possibly non-existent target cell C'. Then either the theoretical C' is contained an already buffer node or it is not. If C' is not contained in a buffer node, we claim that we can simply walk to C' in $\mathcal{O}(\alpha)$ steps with a contradiction argument:

Case 1: no child of a buffered cell. : Figure 4 illustrates the proof. Let C_x be the largest initialized leaf that contains C' and assume that the difference in height between C_x and C is larger than $\log(\alpha)$. If C_x neighbors an ancestor of C, then per definition C_x should be compressed so we know that there is at least one (possibly not yet initialized) cell C'_x neighboring C_x that is not an ancestor of C. But now we have a cell C'_x that lies in between C and C' and that has a size at least $\alpha|C|$ so C and C' are too far apart. □

Case 2: C' is contain in a compressed cell. If C' is contained in an already compressed cell C_a we **assume** that if we can find C_a, we can add C' in $\mathcal{O}(\alpha)$ time. The unique candidate for C_a must be the cell marking the lowest-marked ancestor of C in Y_i and a marked-ancestor query finds this ancestor in $\mathcal{O}(\log(\log(n)))$ time. This is because any higher marked ancestor and any cell adjacent to C_a would only be able to create cells more than $\log(\alpha)$ steps away from C. □

An edge case for the proof. The argument relies on the assumption that once we find the compressed ancestor of C', C_a that we can add C' in $\mathcal{O}(\alpha)$ time (we highlighted the assume in the proof). That assumption is in its current state false due to one edge case which we illustrate in Figure 5. Let C_a have one child cell, C'_a (dark green). In our example C' is the $\mathcal{O}(n)$'th descendant of C'_a. If the path from C'_a to C' cannot be compressed (because there are other regions in the subtree of C'_a), then we would have to traverse $\mathcal{O}(n)$ pointers before we reach the desired target C'. We however strongly believe that this problem can be solved. One solution would be to have a balanced compressed quadtree and earlier work leaves us to believe that these trees can exist in a dynamic setting.

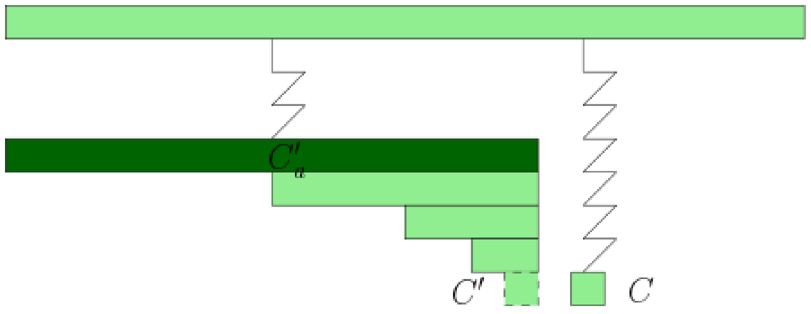

Fig. 5: The edge case with C_a (green) and the walk from C to C'.

2.4 Edge-oracle trees.

In the introduction we stated that traditionally (using R-trees and quadtrees) we solve stabbing queries by making a planar subdivision of cells. A stabbing query for a point q then involves locating the cell C that contains q. In a balanced quadtree of size $\mathcal{O}(n)$, finding this cell C can be done in $\mathcal{O}(\log(n))$ by simply traversing the quadtree. An unbalanced quadtree could have a depth of $\mathcal{O}(n)$ which would make traversal from the root too slow. Instead we introduce an axillary search structure presented in [2], an **edge oracle tree**. Let T be an abstract tree of size $\mathcal{O}(n)$ with a constant maximum degree d. Suppose that the nodes in the tree are given unique labels, and suppose that each edge $e \in T$ has an oracle which for any node label q can answer the following question: "If we remove e and T is split into two components, which component contains q?". The edge-oracle tree is a search structure built over the edges of T which allows us to navigate from any node $u \in T$ to any other node $v \in T$ in $\mathcal{O}(\log(|T|))$ time. We can construct an edge-oracle tree for T by recursively locating an edge which divides T into two components of approximately equal size.

3 Data structures for ply bounded by 1.

We start off by demanding that all regions in $\mathcal{B}$ are disjoint. All regions in $\mathcal{B}$ in this paper will always be closed and bounded. In this we section will introduce data structures for disjoint regions in $\mathbb{R}^d$ with $d \in \{1, 2, 3\}$. The first data structures for storing closed intervals and fat regions in $\mathbb{R}^1$ and $\mathbb{R}^2$ are presented by Löffler *et Al* in [1] and [2]. Later data structures in this work are adjustments of these data structures.

3.1 The first data structure for intervals on $\mathbb{R}^1$

Given a set $\mathcal{B}$ of compact intervals within a bounding box $\mathcal{K}$ on $\mathbb{R}^1$, we intend to store $\mathcal{B}$ in a data structure subject to **stabbing queries** with a runtime logarithmic in n and **local replacement** with a runtime sub-logarithmic in n. To solve this problem we construct a quadtree over $\mathcal{K}$ such that each region $B \in \mathcal{B}$ has its center point m stored in a cell in T. We note that a cell C can only store at most one region B and that if C stores a region B then C must be a leaf.[4] This is because per definition the region B must cover C and because regions must be disjoint. We also claim that the following lemma holds:

Lemma 1. *Each region $B \in \mathcal{B}$ that intersects a leaf cell C either is stored in C or in the nearest non-empty leaf to the left or right of C.*

[4] see Preliminaries on how to construct the unbalanced quadtree

Proof. Assume for the sake of contradiction that B is stored in a non-empty leaf C_2 that is not the closest non-empty leaf to the left or the right of C. Then there must be a non-empty leaf C_1 inbetween C and C_2 that is covered by a region B_1. If B then has its center point in C_2 and contains a point in C then B has to also cover C_1 and thus intersect B_1, violating disjointness. $\square$

Because of this lemma we store in each block (leaf) a pointer to the closest non-empty leaf to its left and right. It is clear that due to our compression, we only have to update a constant amount of pointers when inserting or deleting an interval from $\mathcal{B}$.

Insertion and deletion are performed in $\mathcal{O}(\log(n))$ time by first querying the auxilirary edge-oracle tree to find the appropriate cells in the quadtree and then updating nearest-leaf pointers. So to prove Conjecture 1 for $d = 1$ and $k = 1$ we only need to show that stabbing queries can be performed in $\mathcal{O}(\log(n))$ time and that local updates can be done in $\mathcal{O}(\log(\log(n)))$ time. Given a query point $q \in \mathbb{R}^1$ we can again access the leaf that contains q in $\mathcal{O}(\log(n))$ time. If that leaf stores a region, we report that region. If the leaf does not store a region, it must be a block node and it thus has two pointers to the nearest non-empty leaves. Lemma 1 guarantees that these two leaf cells contain the unique two intervals that could intersect C and thus contain q.
Local updates can be preformed in $\mathcal{O}(\log(\log(n)))$ time: if we want to replace a region B_1 with a region B_2 that is ρ-similar to B_1 for a constant ρ then we can find the cell that should store B_2 with a constant walk from the cell C_1 that stores B_1: because B_1 and B_2 are ρ-similar we know that $|B_2|$ is at least $\frac{1}{\rho}|B_1|$ and at most $\rho|B_1|$ so the cell that should contain B_2 is at most $\mathcal{O}(\log(\rho))$ steps in **depth** from C_1. Similarly B_2 can be at most $\rho|B_1|$ to the left or right of B_1 making any walk from the cell that stores B_1 to the cell that stores B_2 $\mathcal{O}(\rho)$ steps long. Conjecture 2 in the Preliminaries then shows that the walk and additional compression then can be done in $\mathcal{O}(\log(\log(n)))$ time. Updating the nearest-leaf pointers of all the leaves can be done in constant time since compression guarantees that we only need to update a constant amount of leaf cells and all the pointers are available.

3.1.1 Extending this data structure to $\mathbb{R}^2$

Assume we want to extend this data structure to $\mathbb{R}^2$. The first question we then ask ourselves is: what does the set $\mathcal{B}$ look like? This paper will use three different cases of $\mathcal{B}$: The first and easiest case is when regions are axis-aligned **Squares**. The second and harder case is when regions are **circles** and the last and hardest case is when regions are arbitrary **β-fat regions**. The definitions of squares and circles and their center points m are trivial. A region B is β-fat if there exists a pair of co-centric circles I, O with $I \subset B \subset O$ and $|O| \leq \beta|I|$. The center point m of B is then the shared center of I and O. For each region we choose I to be an approximate largest circle contained in B.

If we want to extend the previous idea with pointers to nearest non-empty leaves for $\mathbb{R}^1$ to $\mathbb{R}^2$ then we need a similar property as lemma 1 defines: If a leaf cell C would be intersected by a region B then that region should either be stored in C or in one of the nearest non-empty leaves for a finite set of directions Φ. If that would be true then we could construct a quadtree over the bounding box in $\mathbb{R}^2$ and again add nearest non-empty leaf pointers for each direction in Φ. But for squares, circles and fat regions such a statement can never be true:

Proof. Assume that we have a finite set of directions Φ out of the center of a cell C. Then each $\phi \in \Phi$ defines an infinite cone from the center of C. Now let ϕ_{min} be the smallest angle in Φ. Because ϕ_{min} is a finite number there must be an edge of C that is intersected in an infinite amount of points by the cone of ϕ_{min}. We denote the outer-most intersection points on that edge as the points a and b and the intersection's line segment as $\overline{ab}$. We now construct a counterexample for this hypothetical lemma shown in Figure 6. Construct a region B_1 with size $\frac{1}{1000}|ab|$. Let C_1 be the leaf cell that stores B_1 and place C_1 as far to the left and as close to ab as possible without B_1 intersecting ab and whilst still being contained in ϕ_{min}. Then $d(C, C_1) < \frac{2}{1000}|ab|$. Now let B_2 be a fat region with size $\frac{1}{50}|ab|$ stored in a cell C_2 that is contained in ϕ_{min},

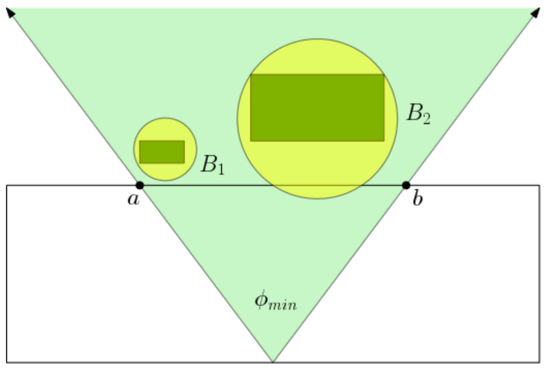

Fig. 6: The counter-example for the proof of lemma 1 in $\mathbb{R}^2$

as far to the right as possible and with $\frac{2}{1000}|ab| \leq d(C, C_2) < \frac{2}{100}|ab|$. Then B_2 is clearly disjoint from B_1 but it intersects C even though B_1 is stored in a closer non-empty leaf. As long as Φ is finite, $|ab|$ will have a size larger than zero so this trick always applies. □

3.2 A second data structure for intervals on $\mathbb{R}^1$

The problem with the previous data structure was that we could get the nearest non-empty leaves but that we did not have any guarantee about the size of the region stored in the leaf. This allows when we extend the data structure to $\mathbb{R}^2$ for edge cases to exist where the nearest non-empty neighbor of a cell C is not the cell in that direction that stores the region that intersects C. To tackle this problem we introduce **marked-ancestor trees** on top of our quadtree cells. In the next few sections we will provide specific implementations of these marked-ancestor trees for intervals, squares and circles and the last section will give a generic data structure for β-fat regions.

3.2.1 The one dimensional case.

We extend the trivial data structure with the quadtree and the pointers by building three *marked-ancestor trees* (LMAT, CMAT, RMAT) on top of the quadtree. We mark a $B \in \mathcal{B}$ in a cell C in the tree $CMAT$ when the following condition is met: C is the largest cell that contains the center point of B and C is covered by B.

Definition 8. *We say a cell C* ***stores*** *a region B if B marks C in $CMAT$.*

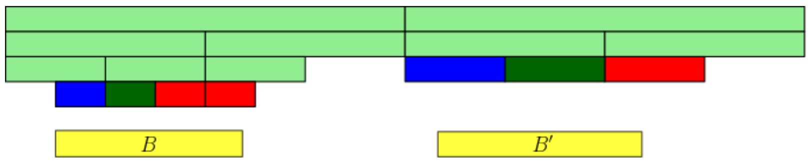

Fig. 7: Two regions and the cells they mark in $CMAT$ (green), $RMAT$ (blue) and $LMAT$ (red)

A cell C can be **marked** in any of the other ancestor trees by a region B if C is intersected by B and the cell that marks B in $CMAT$ has size $|C|$. If the center point of B lies to the right of C it is marked in $RMAT$, if it lies to the left C is marked in $LMAT$. An example is shown in Figure 7. If we want to show that we still satisfy Conjecture 1, we again need to show that local updates, insertions, deletions and stabbing queries can be done in the correct time.

Insertions, deletions and local replacements now consist out of two components. When we do one of these operations we first ensure that the quadtree gets updated. Since the quadtree itself hasn't changed these operations still take $\mathcal{O}(\log(n))$ and $\mathcal{O}(\log(\log(n))$ time. After updating the quadtree we must update the marked-ancestor trees. Since any interval intersects at most four quadtree cells of similar size so this results in a constant amount of operations that each take $\mathcal{O}(\log(\log(n)))$ time.

Stabbing queries. What remains is to explain how stabbing queries work and to show that these can be done in $\mathcal{O}(\log(n))$ time. Given a query point $q \in \mathbb{R}^2$ we find the quadtree leaf C that contains q in $\mathcal{O}(\log(n))$ time using our auxiliary edge oracle tree. We then find the lowest marked ancestors of C in $CMAT$, $LMAT$ and $RMAT$ in $\mathcal{O}(\log(\log(n)))$ time using marked ancestor queries and check if the regions that mark these ancestors contain q. Our claim is that these intervals are the only intervals that can contain q:

Lemma 2. *Given a point $q \in \mathbb{R}^1$ contained in a cell C, if C has a lowest marked ancestor C_1 marked in $CMAT$ by a region B_1 then B_1 contains q.*

Proof. The proof is trivial: per definition, B_1 covers C_1 and because C_1 is an ancestor of C, B_1 must also cover C and thus contain q. ☐

Since no regions may overlap, this is the unique region that can contain q and mark an ancestor of C in $CMAT$. The second theorem shows that if C has a lowest marked ancestor marked in $LMAT$ or $RMAT$ by a region B_1, then B_1 is the only region marking an ancestor of C in $LMAT$ or $RMAT$ that can contain q:

Lemma 3. *Given a point $q \in \mathbb{R}$ contained in a cell C, if C has a lowest-marked ancestor C_1 marked in $RMAT$ or $LMAT$ by an interval B_1, then that is the only region marking an ancestor of C in $RMAT$ or $LMAT$ that can contain q.*

Proof. We give the proof for $RMAT$, the proof for $LMAT$ is symmetrical: assume that the lowest marked ancestor C_1 of C is marked by a region B_1 that does not reach q and that there is a higher marked ancestor C_2 marked by a region B_2 that does contain q. We note that for any C, C_1 and C_2, the rightmost point of C_2 is as least as far to the right as that of C_1 as that of C. If B_2 then reaches q from over the rightmost point of C_2, B_2 would have to intersect the rightmost point of C_1 first, intersecting with B_1 on that point and thus violating the constraint that no intervals can overlap. ☐

Given the cell C, our algorithm first checks if a marked ancestor of C exists in $CMAT$ in $\mathcal{O}(\log\log((n)))$ time with a marked ancestor query. If such a marked ancestor exists we report the region marking that ancestor. Else our algorithm finds the lowest marked ancestor in $RMAT$ and $LMAT$ and verifies if the regions marking those ancestors contain q. If not, Lemma 2 and 3 show that no other ancestors of C marked by a region that contains q can exist.

3.3 Circles and axis-aligned squares in $\mathbb{R}^2$

Let $\mathcal{B}$ be a set of either axis-aligned squares or a set of circles in $\mathbb{R}^2$. We show how we can extend the data structure so that it works for these regions too. The one-dimensional data structure uses marked-ancestor trees with the guarantee that a cell C is marked only by regions that have a size at least as large as $|C|$. We built three marked-ancestor trees resembling different *directions*. There are more directions in $\mathbb{R}^2$ than only left and right and that is why we extend this data structure by creating nine marked-ancestor trees instead of three. Each cell C in our quadtree now defines an axis-aligned square or rectangle in $\mathbb{R}^2$. We create nine marked-ancestor trees. One tree is $CCMAT$ and we use this tree to store regions in the same way as we stored regions in $CMAT$. The other eight marked-ancestor trees mark cells that get intersected by a region that is stored in a cell of equal dimensions, we pick a tree based on which geometry of the storing cell C is closest. This geometry will be called **key geometry** and the formal definition will be

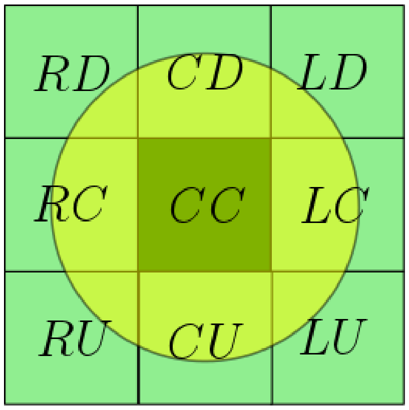

Fig. 8: The cells that get marked by a region B and the prefixes of the names of those trees.

provided in section 4.[5] The trees are called **(CENTER,CENTER)-MAT, (CENTER,UP)-MAT, (CENTER,DOWN)-MAT, (LEFT,CENTER)-MAT,(RIGHT,CENTER)-MAT, (RIGHT,UP)-MAT, (RIGHT,DOWN)-MAT, (LEFT,UP)-MAT, (LEFT,DOWN)-MAT**. Figure 8 shows the canonical case and the names and orientation for the nine marked-ancestor trees. We will use this data structure as the data structure to satisfy Conjecture 1 for $k = 1$, $d = 2$ with $\mathcal{B}$ as a set of axis-aligned squares and a set of circles.

Condition 1. *A region $B \in \mathcal{B}$ marks a cell C_1 in $CCMAT$ if C_1 is the largest cell in the quadtree such that B covers C_1 and C_1 contains the center point of B. We refer to C_1 as the* ***cell that stores*** *B.*

Condition 2. *A region B in$\mathcal{B}$ marks a cell C in one of the eight other marked-ancestor trees if B intersects C and if the cell C_1 that stores B has equal dimensions to C. Which of the eight trees is determined by which of C_1's* ***key geometry*** *is closest to C.*

Insert, delete and local replacements. Theorem 1 in [6] assures us that even in $\mathbb{R}^2$ we can construct a quadtree with an amount of cells linear in n. This means that insertions and deletions in our quadtree can still be done in $\mathcal{O}(\log(n))$ time with use of an auxiliary edge oracle tree[6]. Updating the marked-ancestor trees still requires $\mathcal{O}(\log(\log(n)))$ time. We now show that the local replacement itself in the quadtree can still be done in constant time.

Definition 9. *We define the* ***size*** *of a cell C in our quadtree as $|C| = \max\{x, y\}$*

Lemma 4. *If a circle or axis-aligned square B marks a cell C with width x and height y in $CCMAT$ then the region size $|B|$ is upper-bound by $6|C| = 6\max\{x, y\}$ and lower-bound by $|C| = \max\{x, y\}$.*

Proof. The lower bound is trivial since per definition the region B must cover C. To prove the upper bound we start with a region B that barely covers C and we will increase $|B|$ until it can not be stored in C any more. Note that if B covers all **family neighbors**[7] of C then B has to be stored in the parent of C. Also note that if B is stored in C then the center point of B must also lie in C. The width and the height of the parent of C is upper bound by 2 times the maximum edge length of C. This implies that the distance between any point in C and any point in a family neighbor of C is at most $\sqrt{4|C|^2 + 4|C|^2} = \sqrt{8|C|^2} \leq 3|C|$ Now because B is a square or a circle, if $|B| \geq 6|C|$ then B either does not have its center point in C or B also covers the parent of C. □

[5] for now the figure plus intuition will suffice.

[6] see preliminaries

[7] defined in preliminaries

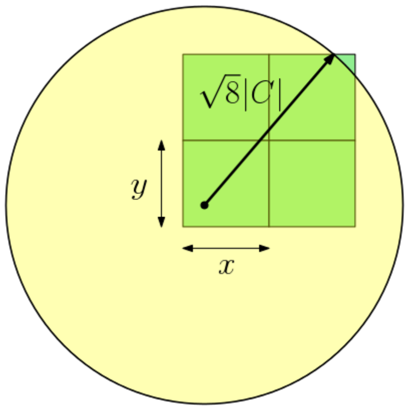

Fig. 9: A cell C of width x and height y that stores a near-maximal sized region

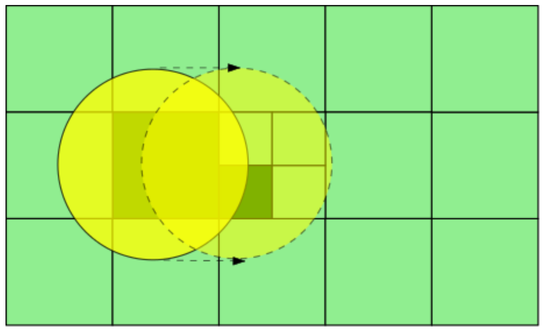

Fig. 10: Locally replacing the yellow region means that it suddenly cannot be stored at the same level of depth anymore.

A local replacement is then again done with two operations: translating and scaling. Let us replace a region B_1 stored in a cell C_1 with a region B_2 that is ρ-similar to B_1. If B_2 differs in size from B_1 then we might need to store B_2 in a cell with a larger or smaller maximum edge size since a cell can only store regions with a size between 1 and 3 times the maximum edge size of that cell. Similarly when translating we could place the region unfavourably on cell edges so that they can not be stored by a cell with equal maximum edge size. Figure 10 shows an example where the yellow region was stored in the large dark green region, if we were to translate the region a bit to the right, it would need to be stored in a cell with a smaller maximum edge size. Finding cells with increased or decreased maximum edge size means taking steps in the ancestry level (**depth**). Noting that B_2 is ρ-similar to B_1 means that $|B_2|$ is at most $\rho|B_1|$ and at least $\frac{1}{\rho}|B_1|$. Going one step up or down in depth increases or decreases the maximum edge size by a factor 2. So if all the cells were initialized we only have to make $\mathcal{O}(\log(\rho))$ steps in depth. If B_2 has its center point at a different location than B_1, it could be stored in a cell that is not an ancestor of C. Assume that we have found an ancestor or descendant of C, C' with a correct maximum edge size. Then due to ρ-similarity and Lemma 4, B_2 is at most $6\rho|C_1|$ away from B_1. Combining both distances means that the new storing cell C_2 is at most $\mathcal{O}(\rho^k)$ steps away from C_1 and Conjecture 2 states that such a walk can be done in $\mathcal{O}(\log(\log(n)))$ time making the total time for local replacement $\mathcal{O}(\log(\log(n)))$.

Stabbing queries. For any point $q \in \mathbb{R}^2$ we can find the leaf C that contains q in $\mathcal{O}(\log(n))$ time using our auxiliary edge oracle tree. We first check if C itself or an ancestor of C is marked in $CCMAT$ using a marked-ancestor query. If this ancestor exists then we report the region marking that ancestor since that region per definition covers C and thus contains q. Else we check each of the eight other marked ancestor for the lowest marked ancestor of C. We verify whether the circle or axis-aligned square marking that ancestor contains q and if not we are done. To prove that this method of querying is correct we again have to prove that the regions marking the lowest marked ancestors of C are the only regions that can contain q:

Lemma 5. *Given a point $q \in \mathbb{R}^2$ contained in a cell C, if C has a lowest marked ancestor C_1 marked in one of the four trees $RUMAT$, $LUMAT$, $LDMAT$ or $RDMAT$ by a circle or a square B_1, then B_1 is the only region marking an ancestor of C in that marked-ancestor tree that can contain q.*

Proof. We only provide the proof for $RUMAT$, the proofs for the other cases are symmetrical. Assume that we have found the lowest marked ancestor C_1 in $RUMAT$ and that the region B_1 marking C_1 does not contain q. Could there be another circle B_2 marking an ancestor of C_1 in $RUMAT$, that contains q but does not intersect B_1? Note that for any ancestor C' of C, the top right corner of C' is at least as high and at least as far to the right as the top right corner of C. $q \in C$ implies that q lies to the bottom left of this top right corner and so for any B_2, the top right corner of C_1 is closer to the center point of B_2 in both x and y direction. This means that any axis-aligned square or circle B_2 that can reach q has to intersect B_1 in that corner and that is not allowed. □

Lemma 6. *Given a point $q \in \mathbb{R}^2$ contained in a cell C. If C has an ancestor C_1 marked in one of the four trees $CUMAT$, $CDMAT$, $LCMAT$ or $RCMAT$ by an circle or axis-aligned square B_1, then B_1 is the only region marking an ancestor of C in that marked-ancestor tree that can contain q.*

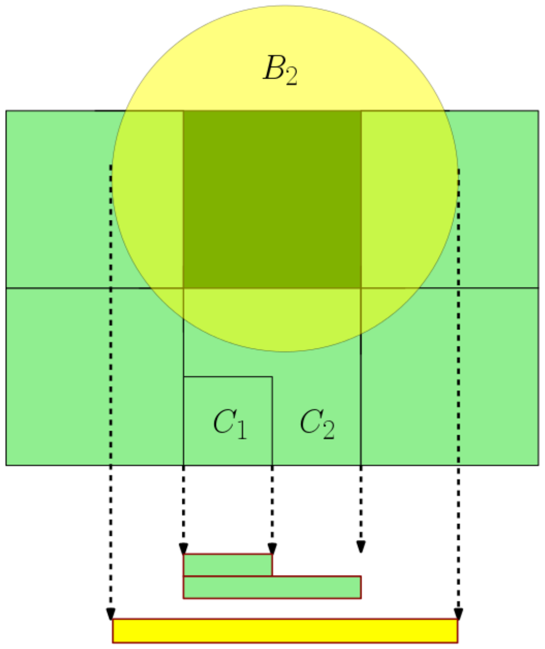

Fig. 11: A cell C_1 and its ancestor C_2 that is marked in $CUMAT$ by a region B_2, and their projections

Proof. We only provide the proof for $CUMAT$, the other proofs are symmetrical. Assume that we have found the lowest marked ancestor C_1 in $CUMAT$ and that the region B_1 marking C_1 does not contain q. Assume that there is a higher marked ancestor C_2 marked by a region B_2 that does reach q. We note that the upper edge of any ancestor C' of C must lie at least as high as the upper edge of C, so the center points of B_1 and B_2 must both lie above the upper edge of C and C_1 respectively. There are however fewer bounds on the x coordinate of the center points of the regions that mark the ancestors since C_1 and C_2 can expand freely to the left or to the right. Note however that if B_1 marks an ancestor C_1 of C in $CUMAT$, B_1 covers a cell above C_1 with equal dimensions to C_1. Since C_1 is at least as wide as C, the projection of B_1 onto the x axis must cover the projection of C and similarly the projection of B_2 must cover the projection of C_1 (see Figure 11 for an illustration of this concept). Combining these two observations means that for any region B_2 marking an ancestor of C_1, we can pick a point p on B_1 with a smaller distance to the center of B_2 in both x as y than the distance between q and the center of B_2. So any B_2 that reaches q would have to intersect B_1 in p and that violates the disjointness of the regions. □

This proves that stabbing queries can be done in $\mathcal{O}(\log(n))$ time by finding the lowest marked ancestor in each tree, thus proving Conjecture 1 for squares and circles in $\mathbb{R}^2$.

3.3.1 Arbitrary convex β-fat regions in $\mathbb{R}^2$

We now focus our attention on disjoint fat regions in the plane. Intuitively, a fat region should not have any long skinny pieces. Formally, we defined β-fat regions as regions B for which there is a pair of co-centric balls I and O, such that $I \subset B$, $B \subset O$ and $\frac{|O|}{|I|} \leq \beta$. In [2] Löffler *et al.* define a data structure that supports logarithmic stabbing queries and sub-logarithmic local replacement for arbitrary β-fat regions. We will restrict ourselves to convex β-fat regions for two reasons: first of all, the authors in [2] use smooth quadtrees. This allows for them to make a distinction between what they call *true* cells and *balancing* cells. Intuitively, the true cells are the cells that need to exist to store all the center points of the regions in $\mathcal{B}$. The balancing cells are then built upon this 'skeleton' to preserve the smoothness of the tree. The authors specifically use these true cells to store regions. We do not use these regions, so we use a different condition to store regions which is easier to define for convex regions. The second reason is that we later want to scale the regions, when we look at sets of regions with limited ply and demanding that the regions are convex makes this scaling easier.

Recall that for each region B, we defined the center point of B as the center of the approximately largest inner circle I_B. From now on, we fix for any region its inner circle and denote it as I_B. A region B is stored in a cell C if C is the largest cell that contains the center point of B and C is covered by I_B. This new storing condition immediately proves Lemma 4.1 and 4.2 from [2]:

Lemma 7. *Let B be a convex β-fat region in $\mathcal{B}$ stored in a cell C. Then C has a size at most $\frac{|B|}{6\beta}$, and B is covered by at most $\mathcal{O}(\beta)$ cells of size $|C|$.*

Proof. Because B is stored in C only based on its inner circle, Lemma 4 proves that the inner circle I_B is less than 6 times the size of C. The size of B is upper bound by the size of its outer circle, which is at most β times the size of the inner circle. This implies that the size of B is at most 6β times the size of C. If B is at most 6β times the size of C, B can be covered by a constant amount of cells of size $|C|$. □

If regions are axis-aligned squares or circles, we know that each leaf cell is intersected by at most eight regions stored in cells that are at least as large as the leaf. If regions are β-fat regions however, a large number of regions can intersect a single cell even if they are stored in a cell at least as large as the leaf! The number of regions that can intersect a single leaf, is however not infinite!

Lemma 8. *The number of β-fat regions intersecting it stored in a cell with at least size $|C|$ is at most $\mathcal{O}(\beta)$.*

Proof. The argument is a different argument from the argument in [2]. Observe that there can be at most eight regions stored in a cell with size $|C|$ adjacent to C. What remains is to count how many regions B with I_B at least $|C|$ can intersect C whilst not being stored in a cell adjacent to C. Observe that all these regions cannot reach C with their inner region. Let B be such a region that intersects C in a point q and let D be a circle with its center at the center of C and with radius $|C|$. Because B is convex, we can make a triangle Δ_1 from q with a diameter of I_B as its base such that Δ_1 is contained in B. The left part of Figure 12 shows this scenario. Now observe that the height of triangle Δ_1 is at most $\beta|I_B|$ because B can only reach that far and that the height of the second triangle Δ_2 is at least $\frac{1}{2}|C|$. The base of Δ_1 has size

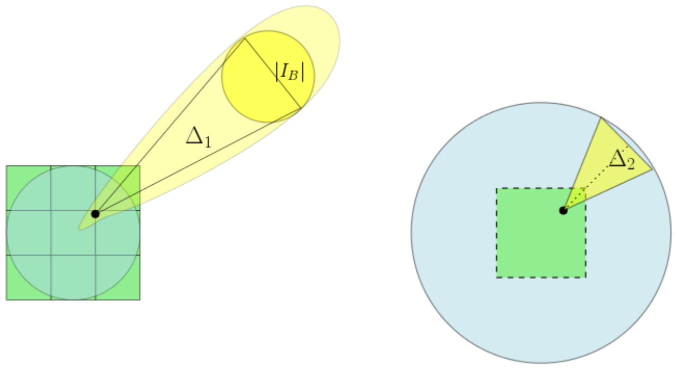

Fig. 12: The proof's construction (left) and the critical zoom (right).

$|I_B|$ so the base of Δ_2 is at least $\frac{\frac{1}{2}|C|}{\beta|I_B|}|I_B| = \frac{1}{2\beta}|C|$. B covers at least the circumference of D in between the base of Δ_2 and that subset of the circumference thus has a size larger than $\frac{1}{2\beta}|C|$. The result is that after a constant amount of regions, we cannot add another region that does not intersect another region at the circumference of D. □

Recall that our nearest-pointer approach looked at a finite set of *directions* Φ and that for each direction it looked at the closest region. Section 3.1.1 showed that in $\mathbb{R}^2$ this approach could not work without the help of marked-ancestor trees. The directions in $\mathbb{R}^2$ for circles and squares could roughly be described as: left/right, up/down and combinations of those. If regions are arbitrary convex β-fat regions, we again need a finite set Φ of directions to look at and Lemma 8 suggests that the number of directions should be $\mathcal{O}(\beta)$. Each direction $\phi \in \Phi$ then gets its own marked-ancestor tree ϕ-MAT. A region B marks a cell C in ϕ-MAT if B intersects C and if the center point m of B lies in the direction ϕ **and** if $|I_B| \geq 2\sqrt{2}|C|$. It is however not possible to make an actual implementation without knowing the exact number of marked-ancestor trees that we need, so the following lemma proves that that number is 4β.

Lemma 9. *For a fixed β, if $|\Phi| = 4\beta$, then for any cell C and for any $\phi \in \Phi$ we have that if C is marked by a region B in ϕ-MAT, then any half-line l that starts in C and that has a direction in ϕ must intersect B.*

Proof. The proof is illustrated in Figure 13. Since we know that B marks C in ϕ-mat, we know that there is a line $\bar{l}$ from the center point of C to m with a direction in ϕ and $|\bar{l}| \leq \frac{\beta}{2}|I_B|$. We know that we can rotate $\bar{l}$ by at most $\frac{2\pi}{|\Phi|}$ in either the clockwise or counter-clockwise direction. Basic geometry then tells us that a rotated $\bar{l}$ reaches at most $\frac{\beta}{2}|I_B|\sin(\frac{2\pi}{|\Phi|})$ in either direction perpendicular to $\bar{l}$. Moreover we can translate each rotated line by the diameter of C, $\frac{1}{\sqrt{2}}|C|$ in each direction perpendicular to $\bar{l}$. Each line l out of C with a direction in ϕ can be created by rotating and translating $\bar{l}$. We now compute how small $|\Phi|$ must be so that no l can reach further perpendicular away from m than $\frac{1}{2}|I_B|$:

$$\frac{\beta}{2}|I_B|\sin(\frac{2\pi}{|\Phi|}) + \frac{1}{\sqrt{2}}|C| \leq (\frac{\beta}{2}\sin(\frac{2\pi}{|\Phi|}) + \frac{1}{4})|I_B| \leq \frac{1}{2}|I_B| \Rightarrow$$
$$\frac{\beta}{2}\sin(\frac{2\pi}{|\Phi|}) + \frac{1}{4} \leq \frac{1}{2} \Rightarrow \beta|\sin(X)| \leq \frac{1}{2}.$$
$$\textit{Now we note that } X \textit{ lies between zero and one since } |\Phi| \textit{ is at least } 8 \Rightarrow$$
$$X - \frac{X^3}{6} < \frac{1}{2\beta} \Rightarrow X = \frac{1}{2\beta} \Rightarrow |\Phi| = 4\beta$$

□

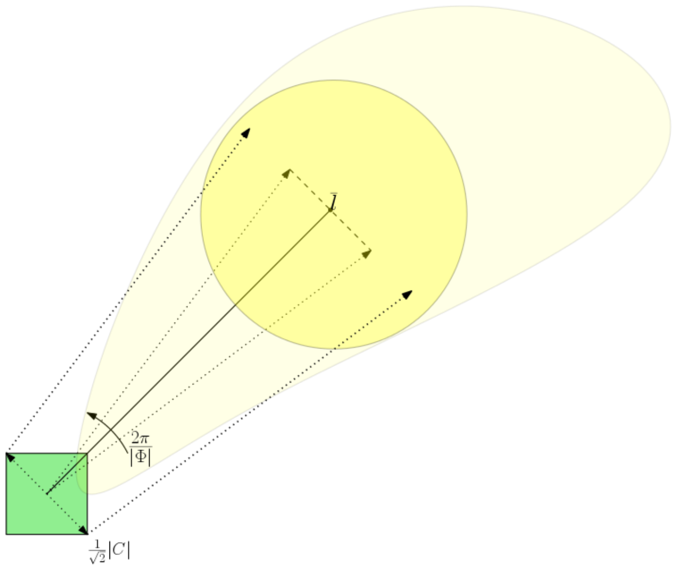

Fig. 13: The line $\bar{l}$ and all its transformations.

Lemma 9 then allows us to easily prove the required lemma to make stabbing queries work:

Lemma 10. *Given a point $q \in \mathbb{R}^2$ contained in a cell C. If C has an ancestor C_1 marked in a marked-ancestor tree X_ϕ for a $\phi \in \Phi$ by a β-fat convex region B_1, then B_1 is the only β-fat convex region marking an ancestor of C in X_ϕ that can contain q.*

Proof. Assume for the sake of contradiction that there is a region $B_2 \in \mathcal{B}$ that marks an ancestor of C_1 and that reaches the query point q. Then because B_2 is larger than B_1, and because its center point m must lie in the cone given by ϕ, m must lie behind B_1 or else the inner region of B_2 must intersect B_1. Observe that the line segment qm has a direction in ϕ and because B_2 is convex, qm is contained in B_2. Since qm reaches beyond B_1, Lemma 9 tells us that qm must intersect B_1 in a point p and B_2 must thus also intersect B_1 in p, violating disjointness. ☐

Stabbing queries can now be performed by manually checking all adjacent cells of the query cell C, and by then querying each of the 4β marked-ancestor trees X_ϕ for their lowest marked ancestor of C. Local replacement is not changed when regions are arbitrary β-fat regions.

3.3.2 Spheres and axis-aligned cubes in $\mathbb{R}^3$

Let $\mathcal{B}$ be either a set of axis-aligned cubes or a set of spheres in $\mathbb{R}^3$. We show how we can extend the data structure to work for these regions too. The extension is made by once again adding more *directions*. Cells in the quadtree now represent axis-aligned cubes in $\mathbb{R}^3$. Just as in the previous cases, a region B marks a cell C in $CCCMAT$ if C is the largest cell that is covered by B and if C contains the center point of B. We refer the cell C that is marked in $CCCMAT$ by a region B as the cell that stores B. For all other cells intersected by B, we choose in which tree we mark that cell depending on what **key geometry** of the storing cell is closest to that cell. We have 6 trees for when the strictly closest geometry to a cell is a face, twelve trees for when the strictly closest geometry is an edge and 8 trees for when the strictly closest geometry is a vertex. That, together with the marked-ancestor tree for regions that entirely cover a cuboid makes 27 marked-ancestor trees that mark cells when they get intersected by a region $B \in \mathcal{B}$. For succinctness we only describe four unique marked-ancestor trees that marks a cell C: $CCCMAT$, $UMAT$ for when the region is stored in a cell whose upper face is closest to C, $RUMAT$ for when the region is stored in a cell whose upper right edge is closest to C and $RUFMAT$ for when the region is stored a cell whose Right Upper Frontal vertex is closest to C.

Insert, delete and local replacements. Theorem 1 in [6] shows that even in $\mathbb{R}^3$ we can construct a quadtree with a number of cells linear in n. This means that insertions and deletions in our quadtree can still be done in $\mathcal{O}(\log(n))$ time. Updating the marked-ancestor trees still requires $\mathcal{O}(\log(\log(n)))$ time. To show that local replacement can be done in $\mathcal{O}(\log(\log(n)))$ time requires us to show that the new cell that has to store the replacement is at most a constant walk away.

Lemma 11. *If a sphere or axis-aligned cube B is stored in a cell C with width x and height y, then the diameter $|B|$ is upper bound by $8\max\{x, y, z\}$ and lower bound by $\max\{x, y, z\}$.*

Proof. Section 2.2 noted that quadtrees in $\mathbb{R}^d$ consist out of cells C with 2^d equally sized children that partition C. In $\mathbb{R}^3$ this means that each cell C has eight children. The lower bound of this lemma is trivial since per definition B must cover C. To prove the upper bound we once again start with a region B that barely covers C and we will increase $|B|$ until it can't be stored in C any more. Observe that the parent of C has a size double in every direction. This means that any point in C is at most $\sqrt{4x^2 + 4y^2 + 4z^2} \leq \sqrt{12\max\{x, y, z\}^2} < 4\max\{x, y, z\}$. This means that if we enlarge $|B|$ by more than a factor 8, we either also cover the parent of C or can't have the center point of B in C anymore. ☐

Similarly to the previous proof, having a size range bound by a constant means that the new cell is at most a constant walk away. Conjecture 2 assures us that such a walk can be made in $\mathcal{O}(\log(\log(n)))$ time.

Stabbing queries. For any point $q \in \mathbb{R}^3$ we can find the leaf C that contains q in $\mathcal{O}(\log(n))$ time using our auxiliary edge-oracle tree. We first check if C has an ancestor marked in $CCCMAT$. If this ancestor exists then we report the sphere or axis-aligned cube intersecting that ancestor since that region covers an ancestor of C and thus contains q. Else we check each of the other trees for the lowest marked ancestor. We check if the region marking that ancestor contains q and if not we're done. To prove that this method of querying is correct we again have to prove that the regions marking the lowest marked ancestors of C are the only regions marking an ancestor of C that can contain q:

Lemma 12. *Given a point $q \in \mathbb{R}^3$ contained in a cell C, if C has a lowest marked ancestor C_1 marked in $RUFMAT$ by a sphere or an axis-aligned cube B_1 then B_1 is the only region marking an ancestor of C in $RUFMAT$ that can contain q.*

Proof. Assume that we have found the lowest marked ancestor C_1 in $RUFMAT$ and that the region B_1 marking C_1 does not contain q. Could there be another region B_2 marking an ancestor of C_1 in $RUFMAT$ that does reach q but does not intersect B_1? The answer is a clear no, since the rightmost-upper front corner of any ancestor C_2 would have to lie further to the front, right and upside than the corner of C_1 meaning that the center point of B_2 would always be closer to the RUF corner of C_1 than to q and any B_2 would thus have to intersect B_1 before reaching q. ☐

Lemma 13. *Given a point $q \in \mathbb{R}^3$ contained in a cell C, if C has a lowest marked ancestor C_1 marked in $RUMAT$ by a sphere or an axis-aligned cube B_1 then B_1 is the only region marking an ancestor of C in $RUMAT$ that can contain q.*

Proof. Assume that we have found the lowest marked ancestor of C, C_1 in $RUMAT$ and that the region B_1 marking C_1 does not contain q. We note that the upper right edge of any ancestor C' of C must lie at least as high and at least as far to the right as the upper right edge of C. This gives a reasonable bound on the region where the center point of B_2 can lie but there are fewer bounds on the z coordinate of the center point of B_2 since C_1 and then C_2 can expand freely in both depth directions. This means that B_1 for instance can lie 'in front' of C and B_2 'behind'. Note that because B_1 marks an ancestor of C and because B_1 is stored in a cell of equal dimensions as that ancestor, that we can project both B_1 and B_2 onto the z

axis and that they then have to overlap. This means that we can find a point p on B_1 that is both closer in x, y and z coordinates to the center of B_2 than that any point in C is. It follows that any axis-aligned cube or sphere B_2 that reaches q would have to intersect B_1 in p first. ☐

Lemma 14. *Given a point $q \in \mathbb{R}^3$ contained in a cell C, if C has an ancestor C_1 marked in $CUMAT$, $CDMAT$, $LCMAT$ or $RCMAT$ by a sphere or axis-aligned cube B_1, then B_1 is the only region marking an ancestor of C that can contain q.*

Proof. The argument is similar to the argument above. We know that the upper plane of any ancestor is at least as high as the upper plane of C_1. Observing that B_1 and B_2 have to overlap when projected onto the floor plane means that we again can find a point p on B_1 that is closer to the center of B_2 than the point q is. ☐

4 Data structures for ply bounded by 2.

Previous sections demanded that the intervals and circles were disjoint. A natural extension of the data structure would be to get rid of this demand. The easiest way to extend the problem would be to demand that each region can only intersect one other region. In practice however, this constraint is as restricting as the previous one since it is very hard for large regions to only intersect one smaller one. Allowing infinite intersections creates a risk of having to search in each level of depth in the quadtree which could take $\mathcal{O}(n)$ time so we instead impose restrictions on the **ply** of our problem.

Definition 10. *The **ply** k of a set $\mathcal{B}$ is the maximum number of $B \in \mathcal{B}$ that an arbitrary point $q \in \mathbb{R}^d$ intersects.*

4.1 Ply 2 in the one dimensional case.

We presented two data structures for the one dimensional variant of our data structure problem. A simple one which kept leaf pointers to the closest non-empty leaf and a more complicated one with three marked-ancestor trees. In section 3.1.1 we showed that the simple data structure could not be extended to $\mathbb{R}^2$. The main problem with this data structure was that we can never construct a finite set of directions Φ with the guarantee that we only have to look at the closest non-empty leaf in each direction in Φ. We showed specifically that for any such finite set Φ we could construct a counter example where the region that intersected C was not stored in the closest non-empty leaf for a direction $\phi \in \Phi$.

The problem found in section 3.1.1 is related to the problem that arises when we try to extend the nearest-pointer data structure for limited ply. Observe that any cell C can either be covered or intersected from the left and the right. Moreover, a ply constraint of $k = 2$ means that any cell C can only be intersected by at most 2 intervals coming the left and by at most 2 intervals coming from the right. We would like to construct a lemma that tells us that given a leaf cell C, we can find the unique two intervals in the left and the unique two intervals in the right direction that intersect C by following a constant amount of pointers. If the ply is restricted to 1 we know that all intervals are stored in leaf cells. If the ply is two it can be that a leaf cell stores two intervals but it can also be that a node in the tree stores a region B_1 and that several descendants of that node store a region that intersects B_1. So having pointers to only leaf nodes does not suffice. The naive approach would be for any region B stored in a leaf to have a pointer to all the higher stored regions that intersect the region B but it is clear that there are scenarios in which updating such pointers when removing a region takes $\mathcal{O}(n)$ time.

Instead we would need an equivalent to lemma 1 where we only need to check a finite amount of **closest** non-empty leaf cells. But again we prove that such a hypothetical lemma cannot exist:

Proof. Observe that the quadtree storing the cells can be embedded in $\mathbb{R}^2$. **Closest** would then have to be closest with respect to both the distance in left and right as the distance in the ancestry relation. We want

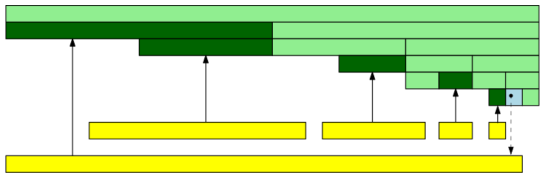
Fig. 14: An illustration with a query cell (blue) and 4 nearest regions.

a general proof so assume that we have an arbitrary metric d that tells us which cells are closest based on their x and y value. We can construct the metric d from any ordering by noting that the ordering must hold for every cell and must thus include the triangle inequality. Now assume for the sake of contradiction that we only have to look at a finite amount k' of **closest** cells. Let all those cells be leaves and disjoint and let one cell C contain a region B that almost reaches our query point q. We can now construct a region B' from B that is not stored in the k' closest cells but does reach q: simply let B reach q and keep increasing its size. This will mean that B has to be stored in an increasingly higher cell and thus increasingly further away. We can increase B without violating any ply until the cell storing B is not in the k' closest cells anymore. Figure 14 shows an example where we took the euclidean metric on the embedding and $k' = 4$ nearest regions. □

4.1.1 Extending the second data structure

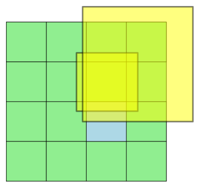
Fig. 15

The idea for this extension comes from the work of M. Löffler *et Al.* in [1] where they present a data structure for bounded ply in the one-dimensional case. The idea is that they increase the number of marked-ancestor trees ($RMAT$, $LMAT$) with a factor of the ply. Regions that fully cover other regions are stored in 'higher' trees. I will however not be using this method for two reasons: first and foremost, this method is hard to extend to $\mathbb{R}^2$ where regions can reach over other regions in an important direction and thus "cover" that region for all intents and purposes without fully covering them in $\mathbb{R}^2$. Figure 15 gives an example. Let the blue square be our query cell and let the two regions above it mark the region cell in $CUMAT$. Then for all intents and purposes in $CUMAT$, the larger region covers the smaller region since it covers the whole width of the query cell and reaches further down. The second reason is that I believe that this definition does not capture the essence of the problem: we want to be able to prove that we can again find the unique region that contains q by querying for the lowest marked ancestor in *each tree*. The adaptions of the authors in [1] has an edge case shown in figure 16. This figure depicts two scenarios where in each scenario we have a query cell C (blue) with an unspecified query point q and two regions that mark the cell in $RMAT$ and contain q. With the definition in [1], in the left case the region B marks C and is covered by the region B_a that marks an ancestor of C. To find q we would only have to search for the lowest marked ancestor of our query cell C in both $RMAT$ trees. In the right case however, B_a does not cover B but it does reach q. If we would only store B_a in a higher tree if it covers another region, then B and B_a would have to be in the same tree and we would thus have to search that tree twice.That is why the algorithm presented in [1] searches each tree twice which drives up time and proof complexity.

We therefore adjust the definition of the data structure for the one dimensional case. We note that in both cases, the region B_a and B both intersect the rightmost point of C. It turns out that whether or not a region a region intersects an endpoint of a cell is **key** for deciding in which level a region should mark a cell. We therefore define for both $LMAT$ and $RMAT$ **key geometry**.

Definition 11. *For each marked-ancestor tree with multiple levels X_i we define **key geometry**. For $LMAT_i$ the **key geometry** of a cell C is its leftmost point of C. For $RMAT_i$ the **key geometry** of a cell C is its rightmost point.*

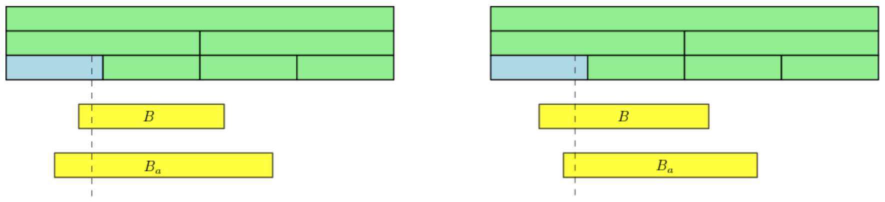

Fig. 16: A query cell (blue) and two regions marking it in $RMAT$.

This solidifies the definition of our marked-ancestor trees that do not have this edge case and earlier sections already extended this definition to the two dimensional case. This section will now prove that these new marked-ancestor trees can perform stabbing queries and local replacement on a set $\mathcal{B}$ with a ply of 2. The lemmas and theorems presented in this section follow a similar order and spirit as the lemmas in [1].

4.1.2 The data structure

The quadtree and CMAT remain the same. We however keep two versions of the other marked-ancestor trees, $RMAT_i$ and $LMAT_i$ with $i \in \{1, 2\}$. If a cell C is intersected by a region $B \in \mathcal{B}$ that marks a cell in $CMAT$ with equal dimensions, we mark that cell in one of the marked-ancestor trees under the following condition:

Condition 3. *If a cell C is marked by a region B in a marked-ancestor tree X_i and C has a descendant C' marked in X_1 by some region B' and B intersects the* ***key geometry*** *of C' then C is marked in X_2. Else in X_1.*

When inserting, deleting or locally replacing intervals we need to know whether the intervals marks a cell on level 1 or level 2. We make use of a special query that tells us in which level a cell should be marked and name this query appropriately a **level query**. We claim that these level queries can be done in $\mathcal{O}(\log(\log(n)))$ time.

Definition 12. *A* ***level query*** *checks for a given region B, cell C and marked-ancestor tree X_i in which level i the region B marks C.*

Lemma 15. *Let C and C_a both be marked in $RMAT_i$ in the same level $i \in 1, 2$ by a region B and B_a respectively. 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 for $LMAT_i$.*

Proof. We prove this by contradiction for $RUMAT_i$: assume that the leftmost point of B_a lies to the left of B. Then clearly any **key geometry** (rightmost point) of any cell C' that is intersected by B, must be intersected by B_a. If $i = 1$ then because the key geometry of C is intersected by B_a, B_a should have been stored be stored in $RMAT_2$ and not $RMAT_1$. If $i = 2$ then per definition, B intersects the key geometry of a descendant C_d of C marked in $RMAT_1$ by a region B_d and thus intersects the region B_d in the key geometry of C_d. Since the leftmost point of B_a lies to the left of B, B_a intersects the key geometry of C_d and thus B_d violating a ply of 2. ☐

Marked-ancestor queries. Section 2.3 in the preliminaries discussed the *firstMarked* query in marked-ancestor trees. Given a marked-ancestor tree over a tree of size n and a path through that tree the *firstMarked* query takes a cell C in the path and returns the next marked cell in the path from C. Our **level query** is a *firstMarked* query over a path with two properties: for each cell C, the subtree induced by C must be a continuous path in the chain and for each marked-ancestor tree X_1 the path returns the cells sorted on the coordinates of their **key geometry**. The first demand is not mentioned in [1] and [2] but is vital, since else the *firstMarked* query could return a cell that does not lie in C whilst there actually are other marked descendants of C. It is clear that that could result in problematic cases for our **level query**. For $LMAT_1$ this path is given by the pre-order traversal of the tree. Pre-order traversal guarantees that each subtree is a connected chain in the path and pre-order traversal exactly gives an ordering on the leftmost point of each cell in the tree. For $RMAT_1$ this path is given by the post-order traversal of the tree. The **level query** $Level(C, B, X_i)$ for the problem in $\mathbb{R}^1$ then returns the cell C_1 that is returned by the *firstMarked* query over one of these paths together with whether or not B intersects the key geometry of C_1.

Lemma 16. *A cell C is marked in X_2 by an interval B if and only if $C_1 = Level(C, B, X_1)$ exists <u>and</u> B intersects the key geometry of C_1.*

Proof. The proof in the right direction is trivial: if there is a marked descendant of C, C_1 such that the key geometry of C_1 is intersected by B then per definition C must be marked in X_2. The other way around is less trivial. Assume a cell C is marked in X_2, then per definition there is at least one descendant of C denoted C_2 marked in X_1 whose key geometry is intersected by B. Denote C_1 as the cell returned by the level query. If C_1 is the only descendant of C marked in X_1 then $C_1 = C_2$ so assume there is more than one marked descendant of C. We note that if C_2 was not returned by the level query, then C_2 must lie further in the path from C. This means that the **key geometry** of C_2 must thus lie at least as far to the left or right as the key geometry of C_1 for $RMAT$ or $LMAT$ respectively. If the key geometry of C_2 lies at least as far away from the center point of B than the key geometry of C_1 then B also intersects the key geometry of C_1. ◻

4.1.3 Local replacement.

The local replacement itself, exchanging a region B with a region B', can still be done in $\mathcal{O}(1)$ time. The only thing that changes is that we now have to adjust several marked-ancestor trees by deleting the region B and inserting the region B':

Deleting. Let the original region B mark cells in $CMAT$, $LMAT_2$ or $RMAT_2$. Then nothing changes in comparison with the original case and thus we still update the markings in $\mathcal{O}(\log(\log(n)))$ time. If B is marked in $RMAT_1$ or $LMAT_1$ however, it could be that unmarking these cells changes whether or not another cell should be marked in $RMAT_2$ or $LMAT_2$. Note that for these cells C_l and C_r marked in $LMAT_1$ or $RMAT_1$, the only cells that they could affect are the lowest marked ancestors of C_l and C_r in $LMAT_2$ and $RMAT_2$. This is because the key geometry of these cells can be seen as a query point $q \in R$ contained in C_l or C_r and our stabbing query proof shows that the only cells we then have to look at are the first marked cells in $LMAT_2$ and $RMAT_2$. Checking both with a marked-ancestor query and possibly moving them to a lower tree then takes $\mathcal{O}(\log\log(n)))$ time. ◻

Inserting. If the new region B' marks a cell C in $CMAT$. Then nothing changes in comparison with the original case. If B' marks C in an $LMAT$ or $RMAT$ we need to decide if we mark C in X_1 or X_2. We first perform a **level query** on X_1 from C. If we find a marked cell C_1, then lemma 16 tells us that that is the unique candidate that could demand that B' marks C in X_2. If B' instead marks C cell in X_1 it could be that because of B', another region marking an ancestor of C in X_1 has to mark that cell in X_2 instead. Just as with deletion, we find the lowest marked ancestor of C in X_1 in $\mathcal{O}(\log(\log(n)))$ time and we know that this is the only cell and region that could be affected by B'. ◻

4.1.4 Stabbing queries.

Given a point $q \in \mathbb{R}$ we first return the lowest two marked ancestors in $CMAT$ (if they exist). We then return the lowest marked ancestor in $LMAT_1$ and $RMAT_1$ and if they exist the lowest marked ancestor in $LMAT_2$ or $RMAT_2$. The proof of correctness is a proof per case. If there are two cells C_1 and C_2 and if they are both marked in $CMAT$ then per definition their marking regions B_1 and B_2 cover C and thus both contain q. Because the ply is at most 2, we can stop here. Else we start searching in the remaining ancestor trees. This approach differs from the one in [1] since we terminate when we find the first lowest marked ancestor.

Theorem 1. *Given a point $q \in \mathbb{R}$ contained in a cell C. If C has a lowest marked ancestor C_1 marked in X_1 by a region B_1, then B_1 is the only region that can contain q. If C has a first and second lowest marked ancestor C_1, C_1' marked in X_2. Then their marking regions B_1 and B_1' are the only marked regions that can contain q and B_1' contains q only if B_1 contains q.*

Proof. We first look at X_1. Assume that we have found the lowest marked ancestor of the cell C in the tree X_1, the cell C_1 marked by a region B_1. Then B_1 either reaches the query point q or does not. If B_1 does not then Lemma 15 demands that any ancestor of C_1 marked in X_1 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 then any region marking a higher ancestor of C in X_1 that reaches q must also intersect the **key geometry** of C_1 and should have marked the ancestor in X_2.

Now let $i = 2$. If B_1 does not reach the point q Lemma 15 again tells us that no other marked ancestor of C_1 can be marked by a region that does. Similarly when B_1' does not reach the point q no region B_2 marking an ancestor of C_1' ever can. When they both contain the point q no other region can reach q without violating a ply of at most 2. ◻

For any query point $q \in \mathbb{R}^2$ we find the leaf containing q in $\mathcal{O}(\log(n))$ time using our search structure. The highest 2 marked ancestors of C in each marked-ancestor tree can be found in $\mathcal{O}(\log(\log(n)))$ time each making stabbing queries run in $\mathcal{O}(\log(n))$ time.

4.2 Axis-aligned squares in $\mathbb{R}^2$

We can extend this approach to storing regions with limited ply to $\mathbb{R}^2$ when we are storing axis-aligned squares or circles. We will encounter some problems with this approach when using axis-aligned squares and these problems are magnified when we use circles. That is why this section will demand that all regions in $\mathcal{B}$ are axis-aligned squares. We mimic the approach for the one-dimensional problem by creating two versions of our ancestor trees. For simplicity and succinctness we only describe the process for $CCMAT$, $RUMAT$ and $CUMAT$. We make two versions of each marked-ancestor tree apart from $CCMAT$ and denote the two levels as $RUMAT_i$ and $CUMAT_i$ with $i \in \{1, 2\}$. For $RUMAT_i$ we define the **key geometry** to be the top right corner of any cell C. For $CUMAT_i$ we define the key geometry as the top edge of any cell C and all the other marked-ancestor trees have their key geometry defined symmetrically. We again mark cells in X_2 when the cell has a descendant marked in X_1 and if the region marking the cell intersects the key geometry of that descendant.

The claim is that this data structure supports local replacement in $\mathcal{O}(\log(\log(n)))$ and stabbing queries in $\mathcal{O}(\log(n))$. To support the local replacement operator we again need a definition for the **level queries**. In $\mathbb{R}^1$ the execution of the level query can be seen as a scanning dot that moves in a lexicographical order on first geometry coordinate and then cell size. In $\mathbb{R}^2$ this scanning dot intuitively becomes a scanning line. For $CUMAT$ this is a horizontal scanning line which moves down and for $RUMAT$ this is a diagonal scanning line that moves from the top right corner to the bottom left. If we want to implement such a scanning line using the *firstMarked* query, we would need a path through the quadtree that follows this scanning line.

For $CUMAT_i$ such a path is easily defined, we traverse the tree by sorting each cell on their highest y coordinate and by traversing in post-order over those cells. It is evident that in that case each subtree induced by a cell C is a connected subchain in the path. All that remains is a proof that a *firstMarked* query over this path also returns the current answer:

Lemma 17. *A cell C is marked by an axis-aligned square B in $CUMAT_2$ if and only if the result of the* firstMarked *query C_1 exists* *and* *the region B intersects the key geometry of C_1.*

Proof. The proof one way is trivial. The proof the other way can be done for squares: Assume the query returns C_1 and that the current state of the scan line is the line l, then the upper edge of C_1 must lie on l. Now let B intersect the top edge of a marked descendant of C, C_2. Then observe that the top edge of C_2 lies on or below the line l, or else the query would have returned C_2. Because the region B covers the width of C and B reaches over the line l, B covers the line l and thus intersects the top edge of C_1. ☐

4.2.1 A lower bound for the level query in $RUMAT$.

There are two problems with the level query for $RUMAT$. The first one is that the level query might not be implementable. Our level query uses the *firstMarked* query over a path. In the one-dimensional problem we chose $v = C$ in a post-order traversal and we showed that for each cell C, the subtree induced by C is a connected chain in that path. Observe that you cannot make a diagonal path through $\mathbb{R}^2$ that makes each subtree of the quadtree a connected chain.

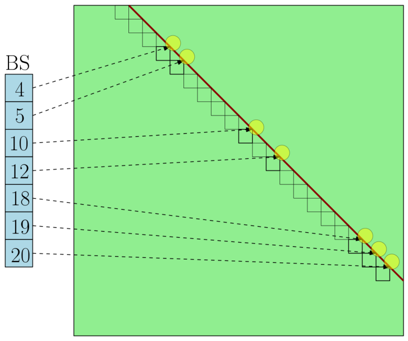

Fig. 17: A set BS and its transformation on a scan line (red).

The second problem is that even *if* we could make such a query, we could use it to make a reduction to binary search. Let BS be a set of n unique ordered numbers and let us query for a number $i^* \in BS$. Let the numbers in BS range from 0 to $k' > n$. For an arbitrary cell C and a diagonal scan line l in C we can find a level of depth such that l intersects at least k' descendant cells of C of equal size. For each $i \in BS$ we mark the i'th cell on that line with a region. Figure 17 shows an example of this transformation where we have a set of seven numbers ranging between 1 and $k' = 20$. A top-right query for a region B that intersects exactly one top right corner in l is now equal to performing binary search on a number in BS meaning that both updates and the search itself can never be done faster than $\mathcal{O}(\log(n))$.

The problem of the level query is even harder than this reduction depicts: this reduction shows that you cannot create an if-and-only-if-situation if all the marked descendants lie on one line. But even if all marked descendants do not lie on a line, we cannot solve this problem just with a diagonal scan line: Let a marked descendant of C, C_1 lie on a scan line l and let C_1 lie near the top of the query cell C. Let another marked descendant C_2 lie below l near the bottom of C. We can easily create a region that does not reach the top right corner of C_1 but does reach the top right corner of C_2. This would suggest that the problem at hand is even harder than binary search.

A sketch for a workaround for the special edge case One problem of the **level query** is that if we try to solve it with a scan line that there could be several marked cells intersecting that scan line. Our algorithm then has to randomly choose an order and that order could be wrong. This process is precisely what we reduced to binary search. The only way that binary search can be done faster than $\mathcal{O}(\log(n))$ time is when you know which indexes of the array you have to access. Observe that given a fixed scan line l, we can compute what 'part' of l a region B intersects in $\mathcal{O}(1)$ time. Ideally we would return the closest marked descendants below l that are closest to a line l' perpendicular to l through the center of B. For a fixed B this path might be implementable in a similar fashion as the level query in section 4.1.2: Marked descendants are kept in a linked list and so we only have to find the marked descendant in the chain that is closest to C and then traverse the linked list a constant amount of times to find the unique marked cell that B has to cover. Now note that if we would want this approach to *always* work we would need to store a path and a linked list for each possible B (and thus each possible l'). Inserting or deleting a marked descendant of C would then require us to update a more-than-linear amount of linked lists making this approach infeasible.

Observe that given a cell C and a marked-ancestor tree X_1, we can find the line l in $\mathcal{O}(\log(\log(n)))$ time using our level query with random order in l. Given the line l we can compute in $\mathcal{O}(1)$ time what part of l our region intersects. Each marked cell in the quadtree gives rise to only one scan line per marked-ancestor tree (it could be that several marked cells have their top corner or edge on the same line). If we store all at most n lines together with pointers to all the cells on each line, we know that each local update updates at most two lines. Given the line l, the only remaining problem is then whether the range of l that B intersects is non-empty. One dimensional non-emptyness queries can be done in sub-logarithmic time (link). This solution might 'bypass' the binary search query, by only looking at a part of all possible marked descendants, namely the ones on $\bar{l}$. This approach is worth investigating more but the second edge case and the problems with implementing the path remain.

4.2.2 Stabbing queries with axis-aligned squares.

We write X_i as an instantiation of an arbitrary marked-ancestor tree at a level i, apart from $CCMAT$. We try to show that stabbing queries can be done in $\mathcal{O}(\log(n))$ time with the help of the following theorem:

Lemma 18. *Given a point $q \in \mathbb{R}^2$ contained in a cell C. If C has a lowest marked ancestor C_1 marked in X_i by an axis-aligned square B_1 that does not reach q, then there is no axis-aligned square in $\mathcal{B}$ marking an ancestor of C in X_i that can contain q.*

Proof for $CUMAT$. Let $i = 1$ and let B_1 not reach q. Let there be a higher marked ancestor of C, C_2 marked by a square B_2 that does reach q. We now note that because of the ancestor relation the upper edges of C_2, C_1 and C projected down all overlap. Moreover, the upper edge of C_2 is at least as high of that of C_1 and C. This means that if B_2 wants to reach q, B_2 would have to intersect the upper edge of C_1 first and B_2 would thus mark C_2 in $CUMAT_2$ instead.
Let $i = 2$ and let B_1 not reach q. Let there be an ancestor C_2 of C_1 marked by a square B_2 that does reach q. Because $i = 2$, there must be a descendant of C_1 that is marked in $CUMAT_1$ and whose top edge is intersected by B_1. Since B_2 fully covers C_2 in width, it must also cover the width of any descendant of C_1 so B_2 cannot reach lower than B_1 without also intersecting the top edge of the marked descendant violating the ply of at most 2. □

$RUMAT$ The argument for $RUMAT_1$ is very similar. Let $i = 1$ and let B_1 not reach q. Let there be an ancestor C_2 of C_1 marked by a square B_2 that does reach q. e note that because C_2 is an ancestor of C_1 and C_1 of C, that the top right corner of C_2 is at least as high and at least as far to the right as the top right corner of C_1 and also of C. This means, that the top right corner of C_1 is closer to the center of B_2 than the point q can ever be, meaning that B_2 would have to cover the top right corner of C_1 and B_2 should thus be stored in $RUMAT_2$ instead.

$RUMAT_2$ is harder. Let $i = 2$ and let B_1 not reach q. Let B_1 cross the key geometry of a marked descendant of C_1, denoted C_d. For any B_2 marking an ancestor C_2 of C_1 we would want a similar argument as above where we would cross the top corner of C_d and thus violate a ply of 2. But note that although we definitely have to cross the top corner of C_1, this doesn't mean that we cover the region that B_1 intersects. That region can be hidden anywhere in the lower corner of B_1 and since we don't have to cover the entire lower corner when we reach q, we could 'miss'. Figure 18 shows an example of this. This, combined with

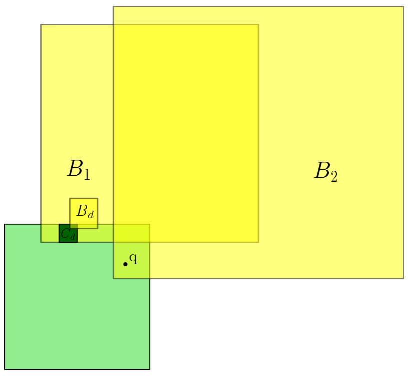

Fig. 18: A region B_1, covering the key geometry of a marked descendant C_d and a region B_2 reaching the query point.

the fact that fast local replacement also wasn't possible for $RUMAT$ would suggest that we can't have sub-logarithmic local replacements AND logarithmic stabbing queries for ANY definition of $RUMAT$.

Extra notes on this edge case: How to tackle this edge case? Observe that if we find two squares marked in ancestors of X_2 that intersect, we're done. Since any other square that marks an ancestor would reach the intersection before it would reach q and would thus violate a ply of at most 2 on that intersection. A first idea would be to find the top x marked ancestors and that somehow they would have to intersect then. But it could be that all ancestors of a cell C only extend in one relevant direction (say to the right) and that there are n disjoint squares to the right of C that all mark an ancestor of C in X_2. A second idea would be to have squares store pointers to all (or some) squares that intersect that square and to use that to quicken the search but there clearly are worst case scenario's where a local update would have to adjust an order n pointers.

Observe that the edge cases for $RUMAT_2$ come in two types: disjoint squares above C and disjoint squares to the right of C. For this to work, each of the upper squares would have to extend left beyond the line $x = q_x$ and each of the right squares would have to extend below the line $y = q_y$. Then there could be an ancestor that only intersects all squares and reaches q. So we either need to find intersecting ancestors or an ancestor that is entirely contained in the upper quadrant from q.

Observe that if we can find the square that reaches q then we can also find the square that intersects with the square marking C_1. That means that we almost have an iff relation between finding intersections between moving squares and stabbing queries. I believe I saw a paper where they also related these two operators. Maybe we can derive a lower bound from there.

5 Approximating queries for arbitrary fat convex regions.

The previous data structure had two main problems when we introduced a ply of two. The first problem was with local replacement. With squares, $RUMAT$ and its similar structures could not detect sub-logarithmically on which level a region should be stored as the problem of **level queries** was reduced to binary search. With circles this reduction applies to level queries for all marked ancestor trees. The second problem was that with squares we were not able to perform stabbing queries in $RUMAT$ in logarithmic time, again the problems found are present for circles in all marked ancestor trees apart from $CMAT$. The lower bound proof on level queries calls for either an adjustment of the data structure or an adjustment of the proposed queries. This section looks at the latter where we 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. Stabbing queries return all cells whose inner region contains q, whilst ply is still defined on the outer region B. The area between the outer region B and the inner region of B could be seen as a "buffer" are that safeguards the actual inner region. The classical form of approximation in computer science is an ϵ-approximation. However, the traditional definition of an ϵ approximation is awkward in the necessary arithmetic for the correctness proofs. That is why we instead define our approximation differently as λ-approximate stabbing queries and later even substitute $\lambda = 2^{-m}$. The goal of this section is to provide a data structure that for a constant m (or constant λ) supports these approximate stabbing queries in logarithmic time and local replacement in sub-logarithmic time. The time bounds of our operations will depend on the approximation constant m and we will briefly show how to rewrite these bounds as a traditional ϵ approximation algorithm.

Definition 13. *For any convex region B, we define the* ***inner region*** *with respect to λ as a map I_λ which takes a region and produces its lambda approximate inner region. Given a region B, $I_\lambda(B)$ is the scaled down version of B with $|B| = (1+\lambda)|I_\lambda(B)|$ with the center of $I_\lambda(B)$ on the center of B. If λ is clear from the context we will denote the inner region $I_\lambda(B)$ as $I(B)$.*

Definition 14. *A λ-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 the* ***inner region*** *of B with $|B| = (1+\lambda)|I(B)|$ and might return arbitrary other regions in which q is contained.*

An example of a region B and its inner region is shown in Figure 19. If $\lambda = 2^m$, this definition allows for an ϵ approximation of stabbing queries with $\epsilon = \frac{1}{2^m+1}$.[8] From now on we will always assume that we have a fixed λ and we will denote the inner region as $I(B)$.

Adjustments in the data structure We keep an identical data structure as before but adjust the data structure's storage constraints and the queries.

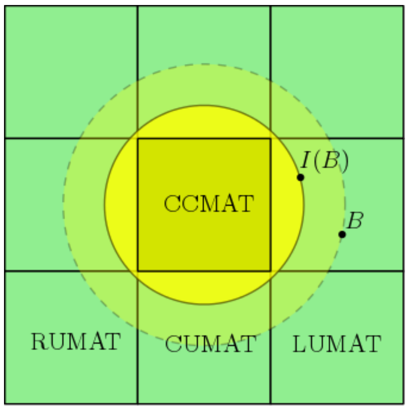

Fig. 19: A region B with its inner region $I(B)$ and the cells marked by the region.

Condition 4. *For a fixed λ we store a region $B \in \mathcal{B}$ in a cell C if C is the largest cell that is covered by the inner region $I(B)$ <u>and</u> if C contains the center point of $I(B)$.*

We mark a cell in a marked-ancestor tree if the cell is intersected by the inner region of a region B and if that region is stored in a cell of similar dimensions. A cell C is marked in a higher level of a marked-ancestor tree when its marking region B intersects the key geometry of a marked descendant of C. The new definition of region brings the following question: should we let the storage level be determined by the inner or the outer region? Since the reduction proof was based on the outer region of B this subsection looks first at the option when the *inner region* has to intersect **key geometry**.

[8] Note that we can define an ϵ approximation as $|B|(1+\epsilon) = I(B)$ and then re-write.

Condition 5. *For any marked ancestor tree X_i apart from $CMAT$, a cell C is marked by a region B in X_2 if there is a descendant of C, C_1 marked in X_1 and $I(B)$ intersects the **key geometry** of C_1.*

5.1 λ approximations for axis-aligned squares.

The smaller λ is, the better the approximation of B is. To show how our algorithm scales with the size of λ, we take $\lambda = \frac{1}{2^m}$ for any $m \in \mathbb{N}$. In this subsection we demand that $\mathcal{B}$ is a set of axis-aligned squares. Our approach also works when $\mathcal{B}$ is a set of circles, but some constants change. Our first claim is that this data structure can do local replacement in $\mathcal{O}(4^m \log(\log(n)))$ time and λ-approximate stabbing queries in $\mathcal{O}(m + \log(n))$ time. Regular updates and deletions can (with use of a fast enough local replacement strategy) still be done in $\mathcal{O}(\log(n))$ time.

5.1.1 Stabbing queries for all marked ancestor trees.

Given condition 5 we look at how stabbing queries would work.

Lemma 19. *Given a point $q \in \mathbb{R}^2$ contained in a cell C. If C has a lowest marked ancestor C_1 marked in X_i by an axis-aligned square B_1 whose inner region does not reach q, then there is no axis-aligned square in $\mathcal{B}$ marking an ancestor of C in X_i that has an inner region that contains q.*

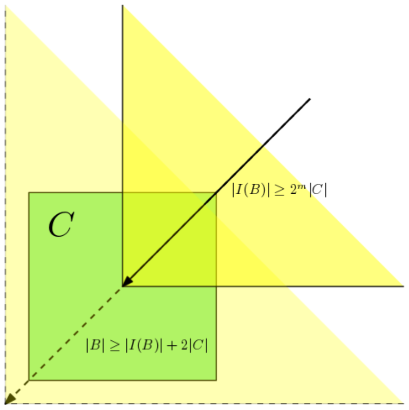

Fig. 20: An illustration of the proof of the second part of the Lemma.

Proof. Assume that we are searching in a tree X_i and that we have found a lowest marked ancestor C_1 marked by a B_1 that does not reach our query point. If $i = 1$, if any higher marked descendant C_2 is marked by a region B_2 where $I(B_2)$ reaches q, then $I(B_2)$ must surely intersect the **key geometry** of C_1 and must thus mark C_2 in X_2 instead.
If $i = 2$ we adjust our searching algorithm: Given C_1 and B_1 we manually check m levels of ancestry in the quadtree above C_1 for regions that could reach q. Any region marking a cell stored more than m levels above C_1 would be marked by a region B_2 where $I(B_2)$ has at least 2^{m+1} times the diameter of $I(B_1)$. This means that any region B_2 marking an ancestor more than m levels above C_1, that reaches q has an expansion that covers all of C_1 (see figure 20). Because B_2 covers C_1, B_2 must intersect any marked descendants of C_1 that $I(B_1)$ intersects there as well, violating a ply of at most 2. □

5.1.2 The problem with approximating level queries

Local replacement itself has not changed because the new storing condition 5 does not change the way we do local replacement. Therefore the only thing that has to be proven is that we can detect fast enough whether or not a cell should be stored in level 1 or level 2.

Remember that the **top right query** in section 4.2 assumed to search through the marked descendants of a square C with a diagonal scan line l. There were two main problems with this query: the necessary path could not be correctly defined and it had problematic edge cases. One edge case was the case where there was a set BS of marked descendants on the scan line l with $|BS| = \mathcal{O}(n)$. In this edge case we could make a reduction to binary search (section 4.2.1). This is because it is impossible to create a path through the marked descendant tree that always reaches an arbitrary fixed marked cell on l first. If we want to be faster than binary search, we need some sort of finger search approach that allows us to look at key elements in the set to skip steps in the search and that is achieved by the following algorithm.

The bisecting algorithm for all marked ancestor trees. This section will introduce what we call the **bisecting algorithm** as a universal **level query** for any marked ancestor tree X_i. Let X_i be an arbitrary marked ancestor tree and let C be a cell marked by a region B in X_i. For the sake of argument we rotate $\mathbb{R}^2$ such that the region B is stored in a cell directly above C. We will try to prove the following lemma:

False Lemma 1. *If $\lambda = \frac{1}{2^m}$ then we can perform an accurate **level query** in $\mathcal{O}(4^{m-2}\log(\log(n)))$ time.*

The algorithm makes use of the *firstMarked* query provided by marked-ancestor trees but to perform a *firstMarked* query we first need a path π through the quadtree. Earlier observations showed that π needed an extra condition, namely that each subtree induced by an arbitrary cell is a connected chain in π. The new path π is easily defined recursively: A cell has four sub-cells each with their own maximal and minimal height. The path orders the sub-cells of each cell on height and then does a post-order traversal.

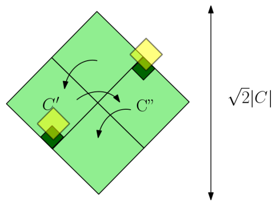

Fig. 21: The path π and marked cells

Lemma 20. *For each cell C, the* firstMarked *query over π from C returns a marked descendant of C that is at most $\frac{1}{\sqrt{2}}|C|$ lower than the highest marked descendant of C.*

Proof. Observe that worst case the algorithm enters a cell C' and returns the first marked descendant of C' on π, whilst the highest marked descendant of C was a descendant of an adjacent cell C" with equal minimal and maximal height to C'. Figure 21 shows an illustration where the dark green cells are the marked descendants. The height of a child cell is at most half of the height of C and the difference in height is thus at most $\frac{\sqrt{2}|C|}{2}$ □

If $\lambda = 2^{-m}$ the bisecting algorithm starts with bisecting the query cell C m times creating 2^m vertical strips with a maximal width of $\frac{\sqrt{2}|C|}{2^m}$ each. Observe that if we go m descendants down from C in our quadtree, each of those descendants either lies in a strip or is bisected by a strip. We assign each of those 2^m descendants to the strip that covers the descendant and bisected descendants get assigned to the strip to the right (see Figure 22). These 2^m cells will be called **defining cells** and all other descendants of C will have at least one corresponding defining cell:

Definition 15. *Given an m, and a cell C in our quadtree. We call the **defining cells** of a descendant C_1 of C the set $Def(C_1) = \{C' \mid |C'| = \frac{1}{2^m}|C|\}$ where one of the following conditions holds for each C':*

- *The **key geometry** of C_1 intersects the key geometry of C'.*
- *$C_1 \subset C'$.*

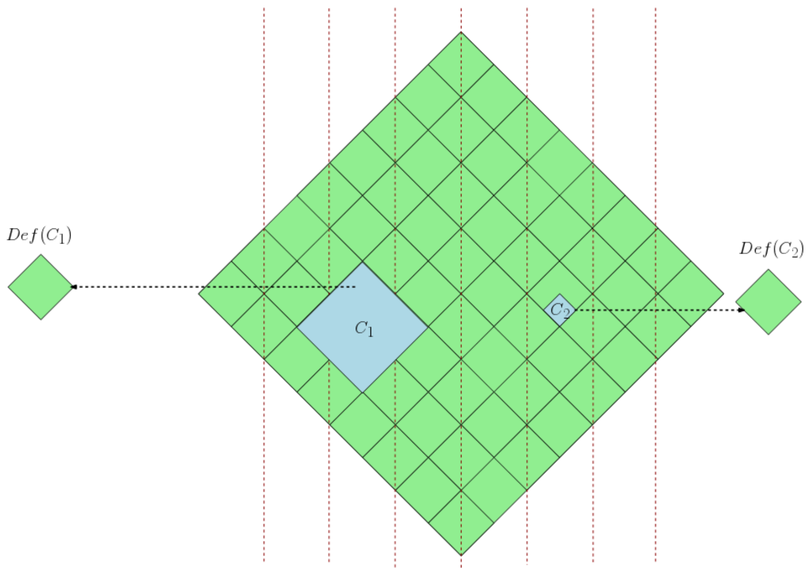

Fig. 22: an example of a 3-bisection

Figure 22 gives an example for $m = 3$. A cell C is bisected 3 times by the red lines. 3 descendants down from C in our quadtree, we see $4^3 = 64$ cells each 2^{-3} the size of C. The figure also shows two descendants of C, a larger one (left) and a smaller one (right) each with their set of defining cells.

Observe that the set of defining cells of a cell always forms a connected chain in π. The bisecting algorithm as a level query takes a cell C, a marked ancestor tree X_1 and with that a path π. The algorithm starts with an empty result set $\overline{V}$. The algorithm performs consecutive *firstMarked* queries over π. Each time we get a marked descendant C_1, we add C_1 to $\overline{V}$ and we find at least one defining cell of C_1. We continue the next *firstMarked* query from the end of the chain given by the defining cells of C_1. Since with each result we skip at least $\frac{1}{4^m}$ of the area of C, we return a set $\overline{V}$ of at most 4^m marked cells.

False Lemma 2. *A region B marks a cell C in $RUMAT_2$ if and only if B intersects the* ***key geometry*** *of at least one $C_1 \in \overline{V}$.*

Proof. The proof of the first direction. We first prove that if a region B marks C in X_2 then B must intersect the **key geometry** of a $C_1 \in \overline{V}$. Fix m and assume that B intersects the key geometry of a cell C_2 that is not in $\overline{V}$.

Lemma 21. *For any marked descendant C_2 of C, for any defining cell $C_d \in Def(C_2)$, there is a $C_1 \in \overline{V}$ such that $C_d \in Def(C_1)$.*

Proof. We know that there are 4^m defining cells in C each forming a connected subchain in π. The bisecting algorithm performs a *firstMarked* query on each of the subchains unless there is a cell found whose defining set covers the subchain, the subchain is then skipped. This means that for each defining cell there is at least one $C_1 \in \overline{V}$ that has that defining cell in its defining set. □

Now observe that the size of the region that marks C is $|B| \geq (1 + \frac{1}{2^m})|C|$, and so B reaches at least $\frac{1}{2^m}|C|$ further than $I(B)$ in the x and y direction. This in turn means that if $I(B)$ reaches in a defining cell of size $\frac{1}{2^m}|C|$, B entirely covers that cell. The Lemma above tells us that there is at least one $C_1 \in \overline{V}$ whose defining set $I(B)$ intersects, so the expansion B must intersect the key geometry of C_1 □

The other way around used to be trivial but is not anymore: Assume that a region B intersects the key geometry of one of the cells in $\overline{V}$, should $I(B)$ then cover the key geometry of a marked descendant of

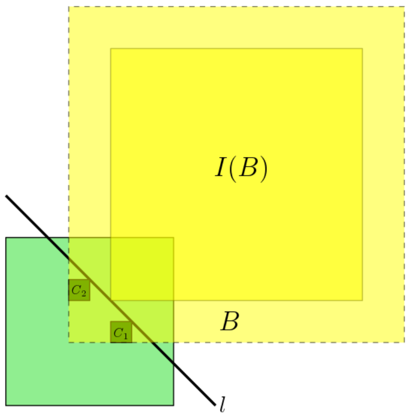

Fig. 23: A counter example for the second part of the proof.

C? Figure 23 gives a clear counter example where the outer region reaches a cell C_1 but the inner region does not. So the algorithm can give 'false positives' for the level query and false Lemma 2 is not correct. Moreover, there could be $\mathcal{O}(n)$ marked descendants on a line between two neighboring elements of $\overline{V}$ where $I(B)$ only intersects one (Imagine our scan line l in between the two marked descendants in figure 23). This in turn again allows a reduction to binary search.

The algorithm fails for all convex inner region approximations So squares cannot be approximated with a condition like condition 5. A another question would then be, can we choose another kind of inner region $I(B)$ that can perform these level queries with condition 5? Note that such an inner region needs to be convex or else it does not allow scaling to approximate B. Now note that if the region is convex, that there is always a point p with a tangent line though that only intersects the polygon in p. We use p to make a binary search reduction: Given a cell C that stores a region B with such a point p. Then one of the eight neighboring cells C' of C contains p. We make a tangent line l through p in C' that only intersects p and we can move B such that B intersects an infinite amount of points on l uniquely whilst still storing B in C. This in turn allows us to do the earlier reduction to binary search in section 4.2.1.

5.1.3 Adjusting condition 5

The conclusion of the bisecting algorithm is that it does not give false negatives but can give us false positives. We proved that this is the case for any convex fat inner region so the only thing we can do is adjust condition 5.

Condition 6. *For any marked ancestor tree X_i apart from $CCMAT$, for any cell C and any region B marking C in X_i we mark C in level 1 or 2 based on the following conditions:*

1. *If there is a descendant C_1 of C marked in X_1 and $I(B)$ intersects **key geometry** of C_1 then C is marked in X_2.*
2. *If B intersects no key geometry of any descendant of C then C is marked in X_1.*
3. *Any other regions might be marked in X_1 or X_2.*

Now (with use of the earlier analysis) the false lemma 2 is clearly true so we have a (relatively fast) way to distinguish whether a region should mark a cell in level 1 or level 2. We do need to show that stabbing queries still work.

Stabbing queries.

Lemma 22. *If $\lambda = \frac{1}{2^m}$ then with condition 6 we can perform a λ-approximate* ***stabbing query*** *in $\mathcal{O}(m + \log(n))$ time.*

Proof. We provide the proof for all X_i. Let regions mark cells according to condition 6. Let q be the query point and let C be the leaf cell that contains q found with our edge oracle tree. Let C_1 be the lowest marked ancestor of C marked by a region B_1 that does not reach q. If $i = 1$ then clearly any marked ancestor marked by a region B whose inner region reaches to q would also have to intersect the **key geometry** of C_1 and would thus have to be stored in X_2.

If $i = 2$ we make use of the old adaption but manually search one level higher: given C_1 and B_1 we manually check $m + 2$ ancestors of C_1 in the quadtree for regions that could reach q. Assume that there exists a region B_2 marking a cell more than $m + 2$ higher than C_1 and let $I(B_2)$ reach q. If $I(B_2)$ covers the region B_1 we are done since we would then clearly violate a ply of at most 2. If $I(B_2)$ does not cover B_1 then either B_1 reaches further down or further to the right or left[9]. Given that B_1 marks C_1 and with use of lemma 4 we know that the lowest coordinate of B_1 is at most $(1 + \lambda)3|C_1| < 4|C_1|$ below q, similar bounds can be found for the rightmost and leftmost point of B_1. The extension B_2 of $I(B_2)$ reaches least $\frac{|C_2|}{2^{-m}} \geq \frac{2^{m+2}|C_1|}{2^{-m}} = 4|C_1|$ further in both x and y and thus covers B_1 violating a ply of at most 2. $\square$

The proven false Lemma 2 and Lemma 22 then form the proof for our first approximation theorem:

Theorem 2. *Let $\lambda = \frac{1}{2^m}$ for any $m \in \mathbb{N}$ and let $\mathcal{B}$ be a set of squares. One can devise a data structure that can do local replacement in $\mathcal{O}(4^m \log(\log(n)))$ time and λ-approximate stabbing queries in $\mathcal{O}(m + \log(n))$ time.*

5.1.4 Better bisection management

Algorithm 1 A quicker version of the bisecting algorithm.

```
1:  procedure LevelQ(Cell $C$, MAT $X_i$, int $m$)
2:      array $\leftarrow newArray[2^m]$
3:      $\overline{V} \leftarrow \{\}$
4:      $C_1 \leftarrow C$
5:      while containsZero(array) do
6:          $C_1 \leftarrow firstMarked(C_1, Quadtree, \pi)$
7:          if $C_1 \notin C$ then return $\overline{V}$
8:          $\overline{V} + C_1$
9:          array$[index(C_1)] \leftarrow 1$
10:         for direction $\in \{left, right\}$ do
11:             $C_2 \leftarrow firstMarked(C_1, \text{array}, \text{direction})$
12:             $X \leftarrow mid(C_1, C_2)$
13:             if array[X] = nil then
14:                 array[X] $\leftarrow (X, Y)$
15:             else
16:                 if beyond($C_1$, array[X]) then
17:                     $fill(\text{array}, index(C_1), index(C_2))$
        return $\overline{V}$
```

A factor 4^m extra for local replacement is large and observe that we have not used our vertical strips yet. The bisecting algorithm relied on checking <u>all</u> the defining cells. Intuitively this is an overkill: assume that

[9] remember that we rotated $\mathbb{R}^2$

you would have a line of defining cells that spanned the entire width of the cell. The line has $\mathcal{O}(2^m)$ cells and clearly any region that tries to reach beyond the line would have an expansion that intersects the **key geometry** of cells in the line. We make use of this property to get an algorithm with a 2^m multiplicative factor. We create an array of zeroes of size 2^m, with each index corresponding to a vertical strip and the defining cells of that strip. Add a marked ancestor tree on top of the array. Each time we do the *firstMarked* query in the bisecting algorithm and find a defining cell, that cell belongs to a strip and thus an index in our array. There are two cases:

Case 1: The first case is that the newfound index is part of a consecutive chain of ones. A chain of i ones represents i neighboring strips. We find the first cell in π from our found cell that is (partially) not contained in the area formed by the strips and continue the query from there. We call this a **skip**.

Case 2: In the second case we flag the index of C_1 with a one. We use the *firstMarked* query to find the nearest non-empty index to the left and to the right of our current index in $\mathcal{O}(\log(\log(2^m))) = \mathcal{O}(\log(m)$ time. For each of those two neighbors, we check to see if the median cell between them is empty, if it is empty we fill that cell with a tuple (X, Y) where X is the index number of the median and Y is the lowest height of the two cells minus their difference in X. If the median is non-empty, we check whether we have crossed the point stored in the median. If so, we mark all non-empty cells in between with a 1, call this filling. We terminate the bisecting algorithm when the array entirely consists out of ones.

Lemma 23. *The bisecting algorithm takes $\mathcal{O}(2^m)$ steps.*

Proof. We prove this lemma by proving that both case 1 and case 2 happen at most $\mathcal{O}(2^m)$ times. Assume that we have a chain of consecutive ones in the array of size i. Then if we skip the cell C_2 two things can happen: we either find a new cell with an index in our chain or we end up in a new part of the array. For the latter case we use induction on the remaining argument. The first case can only happen 2^m times before we reach the bottom of C since each time we skip and return, we must be at least one defining cell lower.
The amount of steps case 2 requires is based on the time it takes to fill a hole of zeroes in the array of size i. Assume we have two non-empty neighboring indexes with i empty indexes in between. If we insert a new index in between the boundary two things can happen: the index could lie on the boundary or the index could lie in between. The first case can only happen $2i$ times before we must have crossed the point (X, Y) and then we fill the hole. In the second case, we split the hole into two holes of size i_1 and i_2 with $i_1 + i_2 < i$. Since the algorithm starts with a hole if size 2^m the algorithm takes $\mathcal{O}(2^m)$ steps.
Each step performs a firstMarked query of $\mathcal{O}(\log(\log(n)))$ time and a firstMarked query of $\mathcal{O}(\log(\log(2^m)))$ time so the **level query** takes $\mathcal{O}(2^m(\log(\log(n)) + \log(m)))$ time. ☐

Lemma 24. *A region B* has to *mark a cell C in X_2 if and only if B intersects the* ***key geometry*** *of at least one $C_1 \in \overline{V}$.*

Proof. Due to the new level condition, the proof is again trivial in one direction. So we only prove that if a region $B \in \mathcal{B}$ must mark a cell C in X_2 then B must intersect the **key geometry** of a $C_1 \in \overline{V}$. Fix m and assume that $I(B)$ intersects the key geometry of a cell C_2 that is not in $\overline{V}$. Note that either the defining cells of C_2 overlap with the defining cells of a $C_1 \in \overline{V}$ or that the defining cells of C_2 were not discovered by the algorithm. For the first case we now know that if C *must* be marked in X_2, that the expansion B must cover the defining cells and thus intersect the key geometry of C_1. In the second case we pick a point q on the key geometry of C_2 and we split the argument in two cases: either there is a $C_1 \in \overline{V}$ with a defining cell above q or there is not. If there is, the argument is simple: the expansion B will cover the defining cell that contains q, and because the region lies above C, the expansion must also intersect the **key geometry** of C_1. If there is no C_1 directly above the point q, then the index of the defining cell containing q must have been skipped. This means that the index is in between the index of two defining cells C_3, C_4 with each a representative in $\overline{V}$. Figure 24 shows such a scenario for both $RUMAT$ and $CUMAT$. We know that q is at least $d_x(C_3, C_4)$ lower than the lowest of the two since else the algorithm would have found C_2. This

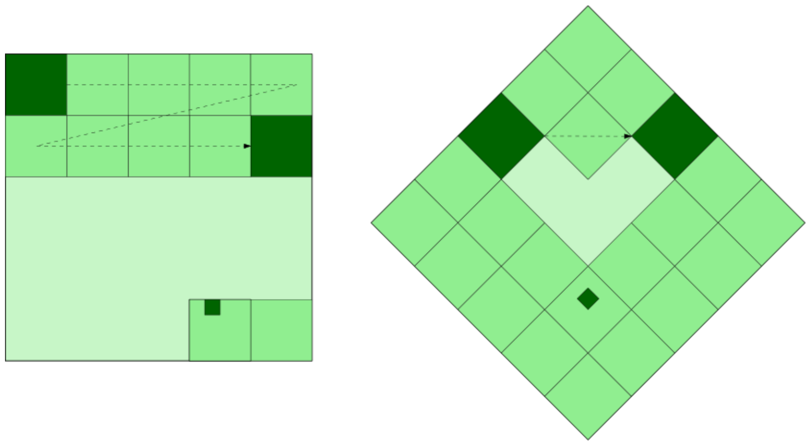

Fig. 24: Both in $RUMAT$ (right) and $CUMAT$ (left), the bisecting algorithm has assured that there are no marked descendants of the cell C with a defining set that lies in the opaque region.

means that any convex fat region has an expansion that covers both C_3 and C_4 and thus intersects the key geometry of the marked descendants that marked C_3 and C_4 in our array. □

5.1.5 Other region types and the final approximation theorem.

The proof was explained with regions as axis-aligned squares, but the proofs can easily be extended to work for arbitrary convex β-fat regions in $\mathbb{R}^2$. For a constant β, a cell C and a marked ancestor tree X_ϕ, the key geometry of C is simply the segment of the border of C that can be intersected by a line from C with a direction in ϕ. The correctness proof for stabbing queries and local replacement relies on one major observation: If you have a cell C and a region B whose inner region intersects C, if B marks an ancestor of C that is more than 2^m higher in tree depth than C, then the outer region B must cover C. This observation clearly holds for any convex β-fat region so we can summarize our result with the following theorem:

Theorem 3. *Let $\lambda = \frac{1}{2^m}$ for any $m \in \mathbb{N}$ and let $\mathcal{B}$ be a set of closed convex β-fat regions in $\mathbb{R}^2$. A data structure exists that can store $\mathcal{B}$ and that supports local replacement in $\mathcal{O}((2^m(\log(\log(n)) + \log(m)))$ time and λ-approximate stabbing queries in $\mathcal{O}(m + \log(n))$ time.*

6 Future work: Devising an impossibility theorem.

Previous sections showed that our current version of level queries can never be fast enough for exact queries with a reduction to binary search. That does not however imply that no data structure exists that has sub-logarithmic local replacement and logarithmic stabbing queries. We know that data structures exist that can handle stabbing queries in $\mathcal{O}(\log(n))$ time for limited ply and it is trivial to have sub-logarithmic local replacements when queries are allowed to take linear time. Yet our research in this paper seems to suggest that the combination of the two cannot be done fast enough in $\mathbb{R}^2$. This section will describe future research and a possible approach to proving a lower bound theorem on the combination of the two operators. The goal of this section is to provide two separate proofs for the following theorem:

Theorem 4 (The Ivorian Impossibility Theorem)**.** *There can be no reduction from our stabbing query problem with local replacement to binary search or lowest number.*

6.1 A dual formulation for the stabbing query problem.

There is a second (equivalent) way to look at the stabbing query problem and proofs. Later sections will use this approach to generalize our problem to aid our conjecture. Let there be a fixed query point q in a leaf

C. We then define an abstract measure Γ from all the possible regions $\mathcal{B}^*$ to the real line $\mathbb{R}^+$.

Definition 16. *Given a point $q \in \mathbb{R}^d$, contained in a quad leaf C and a set of regions S^C that mark ancestors of C in a marked-ancestor tree X_i, for each marked-ancestor tree we define the **Oracle distance measure** as a non-trivial projection $\Gamma :: \mathcal{B}^* \to \mathbb{R}^+$ such that: if one or more regions in S^C reaches q then one of the regions is the region $B \in S^C$ for which $\Gamma(B) = \min S^C$.*

As a concrete example we look at intervals in $\mathbb{R}^1$ that intersect a cell C from the right ($RMAT$). The Oracle distance Γ is in this case given by the left-most coordinate of each region: Given that all $B \in S^C$ mark an ancestor of C we surely know that if any region reaches q then the region with the left-most coordinate reaches q . Moreover we know that if we delete that region, the next 'candidate' for reaching q is the region that now has the lowest left-most coordinate so this definition has the required property for Γ. For intervals that intersect C from the left we can make a similar definition by taking the right-most coordinate of each region and by multiplying it with -1.

6.2 Intermezzo: Conjectured properties of an arbitrary reduction Γ

Assume that we would like to create a generalized Oracle distance measure Γ for all regions in $\mathcal{B}^*$ in $\mathbb{R}^2$. To create such a projection from regions to numbers, we would first need some more properties of that projection. Samewise, if we would want a reduction from lowest number to regions, we would need a projection that would map regions to numbers, and back! Moreover, it is clear that any reduction to lowest number, would have to satisfy the condition for an Oracle distance measure to be able to relate stabbing queries to finding the lowest number. In this section, we name all such projections, a projection Γ and we examine the properties of such a projection Γ.

Injectiveness (sort of)

Lemma 25. *Γ is injective once one region does not cover the other.*

Proof sketch. Recall that both our reduction and our Oracle distance measure, demand that a stabbing query returns a region corresponding to the lowest number. Assume that several regions A get projected to the same number and let for each region $B \in A$, $\Gamma(B) = \min\{\Gamma(S^C)\}$. Let one region $B \in A$ reach q and let all the other not reach q. This construction can be made since per definition no region in A can cover another. Then there exist several regions $B' \in S^C$ for which $\Gamma(B) \neq \Gamma(A)$. Now let there be a $B' \notin A$ that reaches q just like B (this can be done without violating ply k). If we now delete B, $\Gamma(B')$ should be minimal but $\Gamma(B')$ is not minimal because there is still at least one other region in A left. □

So Γ is injective on B^* to at least some degree. The key lies in once one region does not cover one another. Γ is probably not injective over the entire range of B^* but rather over some equivalence relation. If we take our earlier example of Γ over $RMAT$, we see that regions in S^C get compactified to their left-most point. This unknown equivalence relation raises two yet unanswered questions:

1. How do we define local updates on Γ? If numbers correspond with regions (as they clearly do in $\mathbb{R}^1$), then updating numbers in $\mathcal{B}$ should correspond to updating regions. Assume now that we have some equivalence relation on our regions then changing numbers actually means moving from one *class* of regions to the other. But if elements of this class can have varying sizes how can an update then remain local?

2. How do we define such an equivalence relation and what does it mean?

Section 6.6 will try to answer the second question.

Continuousness The function Γ is injective to some degree, functions can often be made surjective so Γ probably has an inverse. This would make sense, since a reduction induces an equivalence between objects in problem 1 and objects in problem 2. Intuitively this Γ and its inverse should be continuous: if we translate a region towards another region and scale it to its size we eventually should become that region in both problems. The question is how do we prove a property like continuousness? A map is always continuous with respect to something. The only measure we have for defining when regions are becoming similar is ρ-similarity so let us sketch a proof:

False Lemma 3. *For every $\epsilon > 0$ there should be a $\rho > 1$ such that if B_1 and B_3 are ρ similar then $|\Gamma(B_1) - \Gamma(B_3)| < \epsilon$.*

False proof. Let there be a B_1 and a B_2 such that $|\Gamma(B_1) - \Gamma(B_2)| = \epsilon$ and let B_2 touch the query point q and B_1 not. B_1 and B_2 are always ρ-similar for some $\rho \in \mathbb{R}$. We would like that any square B_3 that resides within the encapsulating region around B_1 and B_2 is less than ρ similar to both B_1 and B_2. Noting that only B_2 reaches q then implies: $\Gamma(B_1) > \Gamma(B_2)$, $\Gamma(B_3) > \Gamma(B_2)$ and thus $\Gamma(B_1) - \Gamma(B_3) < \epsilon$. $\square$

The assumption that any B_3 within the encapsulating square is more ρ-similar is however clearly false: if B_3 has a very small diameter it is not ρ-similar to anything. This is probably where the earlier eluded equivalence relation comes in that would map regions of similar orientation to the same equivalence class.

The second problem I have with such a proof is that it dodges a definition. Continuity can be defined with respect to a metric or an open in a topology. We directly used ρ-similarity as if it was a metric without verifying if it is.

ρ-similarity as a metric. Let S^C be the set of all possible regions marking an ancestor of a cell C. We denote for two regions $B_1, B_2 \in S^C$ the distance $d(B_1, B_2)$ as the minimal ρ for which B_1 and B_2 are ρ-similar. This distance is clearly always greater than one and we define it to be zero when $B_1 = B_2$. The last remaining axiom we need to satisfy is the triangle inequality:

Lemma 26. *ρ-similarity does not satisfy the triangle inequality.*

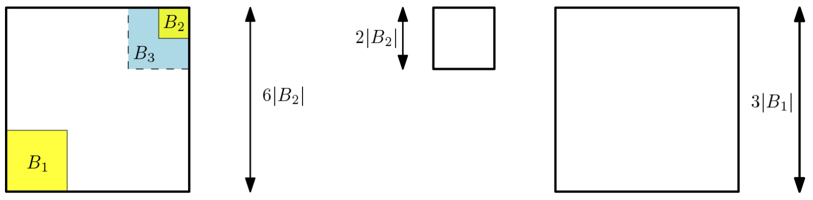

Fig. 25: A counter example showing that ρ-similarity is not a metric.

Proof. We prove it with the counterexample shown in Figure 25. Here we have two regions $B_1 = [0, 2]^2$ and $B_2 = [5, 6]^2$ which are 6-similar. But there exists a region $B_3 = [4, 6]^2$ that is 2-similar to B_2 and 3-similar to B_1 making $d(B_1, B_2) > d(B_1, B_3) + d(B_2, B_3)$. $\square$

It seems that the earlier ϵ, δ proof was bound to fail since ρ-similarity is not a metric. In section 6.5 we examine this continuity further but first we examine how we can use this Γ.

6.3 Using Γ to solve our problem.

We might not have Γ's properties fully defined but we do know how to formulate an exact Γ for the one dimensional case and even $CUMAT$ in the two-dimensional case when we have axis-aligned squares. We can use these exact Γ to solve our problem. For each marked-ancestor tree we define a Γ and the following: for each $B \in S^C$ we define a tuple (d, y) with d the depth in the ancestry level (further from the root is higher) and $y = \Gamma(B)$. Solving the stabbing query for ply two still is equivalent with finding the $B \in S^C$ for which y is minimal. Note that in our one-dimensional query problem, that for our set of tuples we have that when they are ordered on d, then we can never have three consecutive y values: if we would have three consecutive y values, we would have three regions where the ones that mark a higher ancestor reach further to the left in $RMAT$ and we would thus violate $ply = 2$ in their intersection (Lemma 15 in section 4.1.2). For ease of notation, we call finding the minimal number under this restriction: **restricted minimum search**.

Efficiently solving restricted minimum search.

Lemma 27. *We can solve any instance of restricted minimum search with queries in $\mathcal{O}(\log(n))$ time and local replacement in $\mathcal{O}(\log(\log(n)))$ time.*

Proof. If we plot our tuples in the (d, y)-plane, the restriction would mean that that all points can be dominated by at most one other point: because if you dominate a point then that means that you create a consecutive y value whilst you have a consecutive d value. A query point q in a cell C is covered by a region B if and only if the point q from C dominates another point. These severe restrictions make the problem solvable: We create a marked-ancestor tree X_i for two levels $i \in \{1, 2\}$. Every point that has an empty top right corner gets marked in X_1 and every point that has a non-empty top right corner gets marked in X_2. For any query point q we now look at the leftmost-marked point in both trees. For X_1, any further point would have that one in its top right and would thus have a non-empty top right. For X_2 any further point would dominate two points violating ply constraints. It is a more abstract version of the earlier arguments. $\square$

We can also prove the correctness of the $CUMAT$ query with an **Oracle measure**. Let for a point q in a leaf C, $\Gamma :: S^C \to \mathbb{R}^+$ be defined as $\Gamma(B)$ as the lowest y coordinate of B. Then we again have that the lowest $\Gamma(B)$ is the region that we look for and that no consecutive descendants can have an increasing y value, so we solve the problem via the **restricted minimum search**.

6.4 Defining an oracle-distance measure Γ on $RUMAT$ and proving that no reduction can exist.

If we want to extend the same approach used in [1] and [2], we probably need a definition for Γ for $RUMAT$: If a square B marking an ancestor of a query cell C in $RUMAT$ would want to reach the query point q then somehow its bottom left tip would have to intersect q. If we want a unique number and a unique minimum we need a few things: some sort of anchor point in the lower tip from which we measure, and a mapping from the anchor point's x and y distance into one number. Finding this Γ is very hard! For instance: we can not just take the deepest point with respect to the line $y = -x$ going through the top right corner of C. Because there could be a region that does not intersect our query cell C but does reach very far below the line $y = -x$ through the top corner of C. We can also not look at the distance of the lowest point of the region B with respect to that line because a big square could cut miles below that line and still cover q. Note that the first option could be done in $CUMAT$ and that is why we were able to solve the problem.

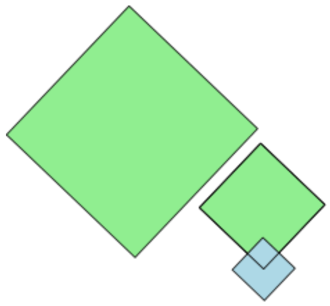
Fig. 26: Drawing (1) and (2).

But suppose that Γ somehow does exist and creates this unique measure. The claim is that this measure then transforms the problem into a **restricted minimum problem**:

Lemma 28. *Any Oracle distance measure Γ over $RUMAT$, transforms stabbing queries into a restricted minimum search.*

Proof. Assume that you have three consecutive tuples $(1, y_1)$, $(2, y_2)$ and $(3, y_3)$ with increasing y. And assume that we would want to place them and our query point q into the plane. Figure 26 illustrates the proof. We start with drawing (1) at an arbitrary location. We can now not draw (2) disjoint from (1): if we do, then because (2) is a lower-marked ancestor of the query cell, the query cell would have to be contained in a descendant of the blue square shown in Figure 26: but if y_1 is then still lower than y_2, then we would break the continuity of Γ because translating (1) towards (2) would bring us in the measure further from q. In the same manner, the measure also imposes restrictions on the orientation of the query cell that contains q. If (2) lies to the right of (1) then surely q cannot lie fully to the right of (2) without breaking the same restrictions. Similarly, (3) has to intersect both (1) and (2) and given that all regions are squares pointed down, (3) then intersects (1) and (2) in one point. ☐

6.4.1 Proving that no reduction to lowest number can exist

Armed with Lemma 27 and Lemma 28 we can give our first proof of our theorem: that for the stabbing query problem, no reduction to lowest number can exist.

Proof. Assume that we would have a reduction Γ that induces an equivalence between a set of numbers $\mathcal{N}$ and our set of regions $\mathcal{B}$. Moreover, solving the stabbing query would be equivalent to finding the lowest number of a subset of $\mathcal{N}$. We claim that the existence of Γ is the solution to our problem! Observe that any reduction Γ from stabbing queries in $RUMAT$ to finding the lowest number is per definition an oracle distance measure. The first argument is that Lemma 28 states that any oracle distance measure transforms the stabbing query problem into a **restricted** minimum search and Lemma 27 states that any restricted minimum search can support stabbing queries in $\mathcal{O}(\log(n))$ time and local updates in sub-logarithmic time. So the existence of the reduction Γ would contradict the result of the reduction. ☐

6.5 Introducing duality to prove again that Γ cannot exist.

Stabbing queries on rectangular regions in $\mathbb{R}^d$ have a well known dual problem in orthogonal range reporting queries in $\mathbb{R}^{2d}$ [4]. Orthogonal range reporting queries can be solved using a kd-tree which traditionally uses $\mathcal{O}(n)$ storage and with $\mathcal{O}(\sqrt{n})$ query and $\Theta(\log(n))$ update time for ranges in $\mathbb{R}^2$. This section will look at the duality problems for stabbing queries of axis-aligned squares and circles, at algorithms that can solve those duality problems and at a transformation between data structures.

The duality of stabbing queries. The duality transformation between stabbing queries on regions and range reporting is a well known transformation. The classical transformation is a transformation from rectangles in $\mathbb{R}^d$ to orthogonal half-range reporting in $\mathbb{R}^{2d}$. Instead of reporting all the regions that intersect a point q, this transformation changes the problem to reporting all the points within an orthogonal half-space originating from q. The dual problem of our stabbing problem is a highly restricted version of the orthogonal range reporting problem.

6.5.1 Duality in $\mathbb{R}^1$

M. Löffler *et Al.* noted in [1] that storing a set of intervals subject to stabbing queries is dual to storing a point set in $\mathbb{R}^2$ subject to quarter-plane range queries. This subsection will try to paraphrase this previous explanation with similar definitions. This duality is derived as follows: Each interval $[a, b]$ is mapped to the point $(a, b) \in \mathbb{R}^2$. Any point $x \in \mathbb{R}$ is seen as a singleton interval $[x, x]$ and thus mapped to (x, x). Note that since for any interval $[a, b]$ $b \geq a$, all points lie above the line $y = x$. Querying if a point q is contained in an interval, is equal to querying if any point is contained in a halfplane originating from (q, q). This dual problem is a highly restricted version of the orthogonal range reporting problem in $\mathcal{R}^2$ which is usually done with kd-trees using $\mathcal{O}(n)$ storage and with $\mathcal{O}(n \log(n))$ query and $\Theta(\log(n))$ update time. Our dual problem has more restrictions than normal orthogonal range reporting since the number of reported points is bounded by the original *ply* and because the query halfplanes all have to originate from the line $y = x$. Because the problem is more restricted, our data structure solves this problem with a faster local update time.

So what does our data structure look like on the dual problem? To help define our data structure on the dual problem, we need a few lemmas:

Lemma 29. *A point q is in an interval $[a, b]$ if the point (a, b) is contained in the plane defined by the lines $(x = q, y = q)$ above $y = x$.*

Proof. If $q \in [a, b]$ then $b \geq q$ so the point (a, b) must lie above $y = q$ and similar $a \leq q$ implies that the point (a, b) lies to the left of $x = q$. ◻

Knowing this, we can define when two intervals overlap.

Lemma 30. *Two intervals $[a, b]$, $[c, d]$ overlap if (a, b) is contained in the halfplane defined by $\text{half}(c, c) \cup \text{half}(d, d) \cup \text{triangle}((c, c), (c, d), (d, d))$ and then always also $(c, d) \in \text{half}(a, a) \cup \text{half}(b, b) \cup \text{triangle}((a, a), (a, b), (b, b))$*

Proof. Note that two intervals overlap if there exist at least 1 point that is contained in both. So to check if they overlap, we simply have to check for all points in $[a, b]$ if they are contained in $[c, d]$. Per Lemma 30 we know that a point $q \in [a, b]$ lies in $[c, d]$ if the point (c, d) lies in the halfplane defined by q. So we create the union of all halfplanes defined by all points $q \in [a, b]$ and check whether the point (c, d) lies in that union. ◻

Lastly we need to define when an interval is covered by another.

Lemma 31. *An interval $[a, b]$ is covered by an interval $[c, d]$ if (c, d) is contained in the halfplane defined by all points to the top left of (a, b)*

Proof. This proof is trivial since we must have that $c \leq a$ and so c left from $x = a$ and $d \geq b$ so d above $y = b$. ◻

Intermezzo: The shape of the embedding of the quadtree. Recall that our data structure is a tree of intervals. An interval B is stored in a cell C if C is the largest cell that is covered by B and C contains the center point of B. Recall that we could embed cells in the quadtree over $\mathbb{R}^1$ in $\mathbb{R}^2$, by drawing the intervals of the cell as blocks and by letting the ancestry relation induce depth in y. In the dual plane, the intervals defined by the cells are only points which makes for a poor embedding. Rather we represent each cell C by the region in $\mathbb{R}^2$ of all intervals B that can be stored in C.
The starting point of such an area is defined by the smallest interval that can be stored in C and that clearly is the interval (and thus the point) defined by C itself. We denote the point that represents the smallest interval that can be stored in C the **base point** of C. Figure 27 shows an example of this construction, with the largest cell being the bounding box of $[0, 4]$. If we are building our one-dimensional quadtree, we keep halving each cell into two intervals. This means that we are splitting each base point into two base points, one with the same x coordinate and one with the same y coordinate. It is clear that because of this, the children of each **base point** must be symmetric in a line parallel to $y = -x$ that goes through that **base point**.

Lemma 31 now tells us, that each cell in the quadtree is covered by all points that lie to the top left of the basepoint of the cell. So to finish the embedding of the quadtree, we extend each basepoint to the left and up. If we hit another basepoint, that must be an ancestor basepoint: since basepoints resemble intervals in our tree and this interval covers our interval (lemma 31). If we hit a symmetry line we can also stop since

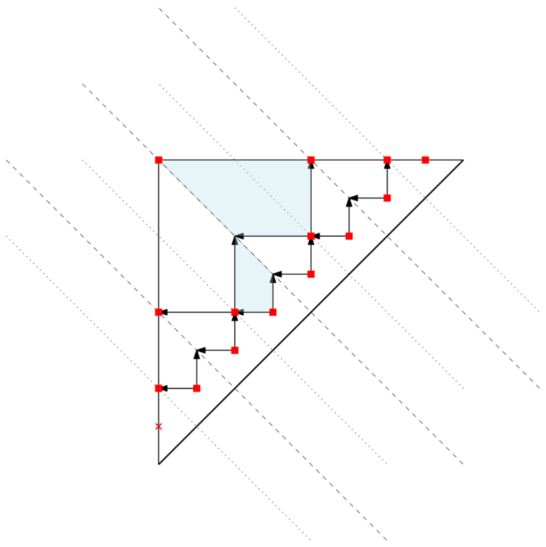
Fig. 27: Constructing our quadtree in the dual plane.

any interval beyond the symmetry line has its center point not in this interval but in the next. Figure 27 shows the construction path of an embedding. Here the bounding box $\mathcal{K}$ is $[0, 8]$, the two blue highlighted areas are the cells $[4, 6]$ and $[3, 4]$.

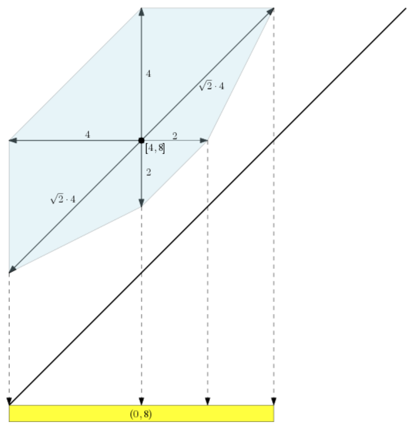

Fig. 28: ρ-similarity illustrated.

ρ-similarity. Recall that Section 6.2 suggested that our Oracle distance measure Γ was continuous and that we raised the question: "with respect to what?". We showed that ρ-similarity was not a metric, so the only thing that remains is that Γ is continuous with respect to a topology basis. It is hard to prove that ρ-similarity is a topology basis from the notation alone but the duality perspective allows for a fast and intuitive proof: Given a point (x, y) in the dual plane corresponding to a region $[x, y]$. We can create ρ-similar regions through translating $[a, b]$, scaling or both. If we have a region $[a, b]$ and a fixed ρ, we can translate $[a, b]$ to the left or the right by at most ρ times the diameter. Translating without scaling means moving in the $y = x$ direction, so we can transform the point (a, b) to at most $(a', b') = (a \pm \rho(b-a), b \pm \rho(b-a)) \rightarrow d((a, b), (a', b')) = \sqrt{2}\rho(b-a)$. So all points with a distance of at most $\sqrt{2}\rho(b-a)$ in the $y = x$ and $y = -x$ direction are ρ-similar to (a, b). When scaling $[a, b]$ we can increase its size by decreasing a (moving to the left) or increasing b (moving up). If we are only adjusting a or b we can move both at most $\rho(b-a)$ up and left. When we scale to decrease the size of the region, the new region is the smaller region so ρ-similarity is measured using the size of the new region. This means that moving one unit, increases the minimal ρ-similarity by two instead of one so we can only make $\frac{\rho}{2}(b-a)$ steps down and to the right. Combining these constrains shows that all ρ-similar regions to a point must be contained in the closed 6-gon around the point. Figure 28 shows the region $[4, 8]$ and its 2-similar points.

With this illustration, it becomes trivial to show that ρ-similarity can be used to define a topology basis. We define the topology basis $\mathcal{T}_{\mathcal{B}^*}$ as the set of all open 6-gons around all points. Or on other words: for each point p, for each $\rho > 1$ we define an open as all the points which are less than ρ-similar to p. From this definition it is clear that any point in our bounding box $\mathcal{K}$ is contained in an open in $\mathcal{T}_{\mathcal{B}^*}$. All that remains is proving the second axiom: Let V_1 and V_2 be two opens in $\mathcal{T}_{\mathcal{B}^*}$. If their intersection is non-empty, it must form an open polygon. Then clearly for any point in the open polygon, we can construct a 6-gon around that polygon such that that 6-gon is contained in the intersection!

6.6 Proving again that Γ cannot exist.

According to Section 6.2 any reduction Γ would have to be a continuous function (in both directions) modulo an equivalence relation. We supplied arguments supporting that the continuity had to be with respect to ρ-similarity of regions and we showed that ρ-similarity is not a metric but is a topology basis. Claiming that Γ is continuous with respect to the topologies means that the reduction maps opens in our first problem space (ρ-similar regions) to opens in the second problem space (open intervals in $\mathbb{R}^1$).

6-gons can of course be mapped to open intervals in a continuous manner (the one-point removal trick already proves this). Recall now that Section 6.2 eluded to an equivalence relation over our continuous reduction. Observe with $RMAT$ and $LMAT$ (our problem in $\mathbb{R}^1$, that their restricted minimum search variant Γ only looked at the leftmost or the rightmost coordinate. In our dual space, that means projecting all regions (points) onto the x or y axis. Projecting an open 6-gon such as the one in Figure 28 onto the x or y axis creates exactly an open interval. So our restricted minimum search transformation is indeed a continuous projection. Assume that we have a bijective (continuous modulo an equivalence relation) reduction Γ from $RUMAT$ and stabbing queries to lowest number. Then again the only possible continous projection, is the one where you project the k-gon to an open interval. This means that you reduce the information of each region to a single point. In that scenario, ρ-similar regions in $\mathbb{R}^2$ have a 1-1 correspondence to open intervals, and with the inverse of the restricted minimum search transformation a 1-1 equivalence relation with ρ-similar regions in $\mathbb{R}^1$. Which would make the problem spaces equally hard!

7 Results and discussion.

We have shown that given Conjecture 2 we can maintain a set of arbitrary convex β-fat regions in $\mathbb{R}^2$ with constant ply in a data structure that supports 2^{-m}-approximate stabbing queries in $\mathcal{O}(m + \log(n))$ time and local replacement in $\mathcal{O}(2^m \log(\log(n)))$ time. This approximation data structure was based on existing data structures from [1] and [2] that can store disjoint regions in $\mathbb{R}^1$ and $\mathbb{R}^2$. We redefined these data structures and we proved that the approach in [1] for storing overlapping regions can never be extended to $\mathbb{R}^2$.

The reduction from the level query to binary search sparks a suspicion about the realizability of sub-logarithmic local replacement in $\mathbb{R}^2$. Also, the time bounds of the approximate data structure seem to suggest that sub-logarithmic local replacement might not be feasible for regions in $\mathbb{R}^2$ because stabbing queries scale logarithmically with the approximation factor, but local replacement scales linearly. It would be interesting to look at a lower bound proof for the combination of stabbing queries with local replacement that shows that both operations must run in $\Omega(\log(n))$ time. We suspect that such a lower bound might exist for this problem, but Theorem 4 from Section 6 shows that the traditional way of proving a lower bound with a reduction to either binary search or heap operations is infeasible.

Earlier we stated that the stabbing query problem with local replacement could easily be solved in $\mathcal{O}(\log(n))$ time if local replacement also can take $\mathcal{O}(\log(n))$ time with the use of more traditional techniques such as KD-trees and R-trees. For constant ply, our approximation algorithm is clearly faster than these more traditional methods. If the ply is a non-constant number k however, we perform each of our operations k times. When k nears n, the traditional methods are faster in both operations. An open question is whether or not we can transition our data structure to one of the more classical data structures swiftly if we notice that the ply is getting too high. Building these data scratch usually takes $\mathcal{O}(n \log(n))$ time, to store each region in a new data structure we clearly need at least $\Omega(n)$ time. It would be interesting to see if we can use the information that is stored in our quadtree to construct these classical data structures in less than $\mathcal{O}(n \log(n))$ time. It would also be interesting to see if we could make a data structure that also supports local replacement sub-logarithmically with high ply, perhaps with slower sub-linear stabbing queries.

Lastly we note that Conjecture 2 is still an open problem. Given a pointer to a cell C and a possibly non-existent ρ-similar cell C' in a compressed quadtree, can we walk from C to C' in sub-logarithmic time? The authors in [1] and we in this paper tried to solve this problem with the use of marked-ancestor queries. This approach works for all but one edge case which we discuss in our preliminaries. Can we find a way around this edge case and still solve this problem with marked-ancestor queries? Another approach would be to have a dynamic smooth compressed quadtree that stores our regions. The authors of [2] and we have also considered how to dynamically maintain such a quadtree. There might be an approach where we maintain a slightly larger than minimal compressed quadtree to store our point set or set of regions. These redundant cells then make sure that during an update, not too many new cells must be initiated. Another way to construct this dynamic smooth compressed quadtree would be to first provide a quadtree structure that can perform constant insertions and amortized constant deletions. We can then try to construct a fully dynamic smooth compressed quadtree using de-amortization techniques.

References

[1] E. Khramtcova, M. Löffler, *Dynamic stabbing queries with sub-logarithmic local updates for overlapping intervals.*

[2] M. Löffler, J. Simons, D. Strash, *Dynamic Planar Point Location with Sub-Logarithmic Local Updates*

[3] E.N. Hanson, T. Johnson *The Interval Skip List: A Data Structure for Finding All Intervals That Overlap a Point*, "Algorithms and Data Structures: 2nd Workshop", August 14-16, 1991, 153-164.

[4] J. Erickson, P.K. Agarwal, *Geometric Range Searching and Its Relatives*, Contemporary Mathematics 223, 1999, 1-56.

[5] Y Nekrich. *Data structures with local update operations*, Scandinavian Workshop on Algorithm Theory, 2008, 138–147.

[6] M. Bern, D. Eppstein, and J. Gilbert. *Provably good mesh generation.* Journal of Computer and System Sciences. Volume 48 issue 3, 1994, 384-409.

[7] S. Alstrup, T. Husfeldt, and T. Rauhe. *Marked ancestor problems.* Technical Report DIKU-TR-98/9, Department of Computer Science, University of Copenhagen, 1998.