Agglomerative clustering of growing squares

Agglomerative Clustering of Growing Squares[⋆]

Thom Castermans[1], Bettina Speckmann[1], Frank Staals[2], and Kevin Verbeek[1]

[1] TU Eindhoven, the Netherlands
[t.h.a.castermans|b.speckmann|k.a.b.verbeek]@tue.nl

[2] Utrecht University, the Netherlands
f.staals@uu.nl

Abstract. We study an agglomerative clustering problem motivated by interactive glyphs in geo-visualization. Consider a set of disjoint square glyphs on an interactive map. When the user zooms out, the glyphs grow in size relative to the map, possibly with different speeds. When two glyphs intersect, we wish to replace them by a new glyph that captures the information of the intersecting glyphs.

We present a fully dynamic kinetic data structure that maintains a set of n disjoint growing squares. Our data structure uses $O(n(\log n \log\log n)^2)$ space, supports queries in worst case $O(\log^3 n)$ time, and updates in $O(\log^7 n)$ amortized time. This leads to an $O(n\alpha(n)\log^7 n)$ time algorithm (where α is the inverse Ackermann function) to solve the agglomerative clustering problem, which is a significant improvement over the straightforward $O(n^2 \log n)$ time algorithm.

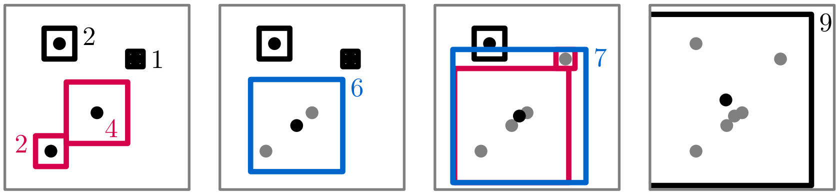
1 Introduction

We study an agglomerative clustering problem motivated by interactive glyphs in geo-visualization. Specifically, *GlamMap*[3] [6] is a visual analytics tool for the eHumanities which allows the user to interactively explore datasets which contain metadata of a book collection. Each book is depicted by a square, color-coded by publication year, and placed on a map according to the location of its publisher. Overlapping squares are recursively aggregated into a larger glyph until all glyphs are disjoint. As the user zooms out, the glyphs "grow" relative to the map to remain legible. As a result, glyphs start to overlap and need to be merged into larger glyphs to keep the map clear and uncluttered. To allow the user to filter and browse real world data sets[4] at interactive speed we hence need an efficient agglomerative clustering algorithm for growing squares (glyphs).

[⋆] The Netherlands Organisation for Scientific Research (NWO) is supporting T. Castermans (project number 314.99.117), B. Speckmann (project number 639.023.208), F. Staals (project number 612.001.651), and K. Verbeek (project number 639.021.541).

[3] http://glammap.net/glamdev/maps/1, best viewed in Chrome. GlamMap currently does not implement the algorithm described in this article.

[4] For example, the catalogue of WorldCat contains more than 321 million library records at hundreds of thousands of distinct locations.



$t = 1$, numbers = weights

$t = \frac{26}{21}$

$t = \frac{26}{21}$, final square

Fig. 1. Events for growing squares: intersecting squares in red, merged squares in blue.

Formal problem statement. Let P be a set of points in $\mathbb{R}^2$. Each point $p \in P$ has a positive weight p_w. Given a "time" parameter t, we interpret the points in P as squares. More specifically, let $\square_p(t)$ be the square centered at p with width tp_w. For ease of exposition we assume all point locations to be unique. With some abuse of notation we may refer to P as a set of squares rather than the set of center points of squares. Observe that initially, i.e. at $t = 0$, all squares in P are disjoint. As t increases, the squares in P grow, and hence they may start to intersect. When two squares $\square_p(t)$ and $\square_q(t)$ intersect at time t, we remove both p and q and replace them by a new point $z = \kappa p + (1 - \kappa)q$, with $\kappa = p_w/(p_w + q_w)$, of weight $z_w = p_w + q_w$ (see Fig. 1). Our goal is to compute the complete sequence of events where squares intersect and merge.

We present a fully dynamic data structure that uses $O(n(\log n \log\log n)^2)$ space, supports updates in $O(\log^7 n)$ amortized time, and queries in $O(\log^3 n)$ time, which allows us to compute the agglomerative clustering for n squares in $O(n\alpha(n)\log^7 n)$ time. Here, α is the extremely slowly growing inverse Ackermann function. To the best of our knowledge, this is the first fully dynamic clustering algorithm which beats the straightforward $O(n^2 \log n)$ time bound.

Related Work. Funke, Krumpe, and Storandt [7] introduced so-called "ball tournaments", a related, but simpler, problem, which is motivated by map labeling. Their input is a set of balls in $\mathbb{R}^d$ with an associated set of priorities. The balls grow linearly and whenever two balls touch, the ball with the lower priority is eliminated. The goal is to compute the elimination sequence efficiently. Bahrdt et al. [4] and Funke and Storandt [8] improved upon the initial results and presented bounds which depend on the ratio Δ of the largest to the smallest radius. Specifically, Funke and Storandt [8] show how to compute an elimination sequence for n balls in $O(n \log \Delta(\log n + \Delta^{d-1}))$ time in arbitrary dimensions and in $O(Cn \operatorname{polylog} n)$ time for $d = 2$, where C denotes the number of different radii. In our setting eliminations are not sufficient, since merged glyphs need to be re-inserted. Furthermore, as opposed to typical map labeling problems where labels come in fixed sizes, our glyphs can vary by a factor of 10.000 or more.

Ahn et al. [2] very recently and independently developed the first sub-quadratic algorithms to compute elimination orders for ball tournaments. Their results apply to balls and boxes in two or higher dimensions. Specifically, for squares in two dimensions they can compute an elimination order in $O(n \log^4 n)$ time. Their results critically depend on the fact that they know the elimination priorities at the start of their algorithm and that they have to handle only deletions. Hence they do not have to run an explicit simulation of the growth process and can achieve their results by the clever use of advanced data structures. In contrast, we are handling the fully dynamic setting with both insertions and deletions, and without a specified set of priorities.

Our clustering problem combines both dynamic and kinetic aspects: squares grow, which is a restricted form of movement, and squares are both inserted and deleted. There are comparatively few papers which tackle dynamic kinetic problems. Alexandron et al. [3] present a dynamic and kinetic data structure for maintaining the convex hull of points moving in $\mathbb{R}^2$. Their data structure processes (in expectation) $O(n^2\beta_{s+2}(n)\log n)$ events in $O(\log^2 n)$ time each. Here, $\beta_s(n) = \lambda_s(n)/n$, and $\lambda_s(n)$ is the maximum length of a Davenport-Schinzel sequence on n symbols of order s. Agarwal et al. [1] present dynamic and kinetic data structures for maintaining the closest pair and all nearest neighbors. The expected number of events processed is roughly $O(n^2\beta_{s+2}(n)\operatorname{polylog} n)$, each of which can be handled in $O(\operatorname{polylog} n)$ expected time. Some of our ideas and constructions are similar in flavor to the structures presented in their paper.

Results and organization. We present a fully dynamic data structure that can maintain a set P of disjoint growing squares. Our data structure reports an *intersection event* at every time t when a square $\square_q$ touches $\square_p$ of a point $p \in P$ that *dominates* q. Here we say that a point p dominates q if and only if $q_x \leq p_x$ and $q_y \leq p_y$. We combine four of these data structures, one for each quadrant, to ensure that all squares in P remain disjoint. When our structure detects an intersection event we have to delete two or more of the squares and subsequently insert a new, merged, square. At any time, our data structure supports querying if a new square is disjoint from the ones in P (see Section 3.2), and inserting a new disjoint square or removing an existing square (see Section 3.3).

The crucial observation is that we can maintain the points $D(q)$ dominating q in an order so that a prefix of $D(q)$ will have their squares intersect the top side of $\square_q$ first, and the remaining squares will intersect the right side of $\square_q$ first. We formalize this in Section 2. We then present our data structure—essentially a pair of range trees interlinked with "linking certificates"—in Section 3. While our data structure is conceptually simple, the details are somewhat intricate. Our initial analysis shows that our data structure maintains $O(\log^6 n)$ certificates per square, which yields an $O(\log^7 n)$ amortized update time. This allows us to simulate the process of growing the squares in P—and thus solve the agglomerative glyph clustering problem—in $O(n\alpha(n)\log^7 n)$ time using $O(n\log^6 n)$ space.

In Section 4 we analyze the relation between canonical subsets in dominance queries. We show that for two range trees T^R and T^B in $\mathbb{R}^d$, the number of pairs of nodes $r \in T^R$ and $b \in T^B$ for which r occurs in the canonical subset of a dominance query defined by b and vice versa is only $O(n(\log n \log\log n)^2)$, where n is the total size of T^R and T^B. This implies that the number of linking certificates that our data structure maintains, as well as the total space used, is actually only $O(n(\log n \log\log n)^2)$. Since the linking certificates actually provide an efficient representation of all dominance relations between two point sets (or within a point set), we believe that this result is of independent interest.

All proofs omitted from this article can be found in the full version [5].

2 Geometric Properties

Let ℓ_q denote the bottom left vertex of a square $\square_q$, and let r_q denote the top right vertex of $\square_q$. Furthermore, let $D(q)$ denote the subset of points of P dominating q, and let $L(q) = \{\ell_p \mid p \in D(q)\}$ denote the set of bottom left vertices of the squares of those points.

Observation 1. Let $p \in D(q)$ be a point dominating point q. The squares $\square_q(t)$ and $\square_p(t)$ intersect at time t if and only if $r_q(t)$ dominates $\ell_p(t)$ at time t.

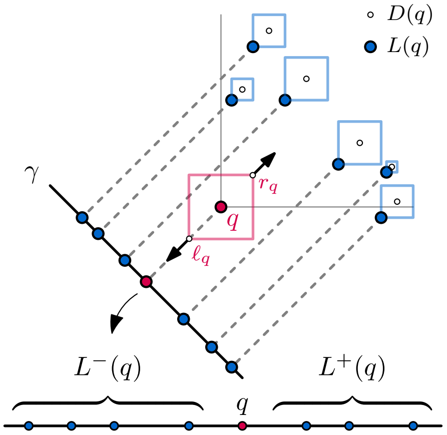


Fig. 2. The projection of the square centers and relevant corners onto line γ.

Consider a line γ with slope minus one, project all points in $Z(t) = \{r_q(t)\} \cup L(q)(t)$, for some time t, onto γ, and order them from left to right. Observe that, since all points in Z move along lines with slope one, this order does not depend on the time t. Moreover, for any point p, we have $r_p(0) = \ell_p(0) = p$, so we can easily compute this order by projecting the centers of the squares onto γ and sorting them. Let $D^-(q)$ denote the (ordered) subset of $D(q)$ that occur before q in the order along γ, and let $D^+(q)$ denote the ordered subset of $D(q)$ that occur at or after q in the order along γ. We define $L^-(q)$ and $L^+(q)$ analogously (see Fig. 2).

Observation 2. Let $p \in D(q)$ be a point dominating point q, and let t^* be the first time at which $r = r_q(t^*)$ dominates $\ell = \ell_p(t^*)$. We then have that

- $\ell_x < r_x$ and $\ell_y = r_y$ if and only if $p \in D^-(q)$, and
- $\ell_x = r_x$ and $\ell_y \leq r_y$ if and only if $p \in D^+(q)$.

Observation 2 implies that the points p in $D^-(q)$ will start to intersect $\square_q$ at some time t^* because the bottom left vertex ℓ_p of $\square_p$ will enter $\square_q$ through the top edge, whereas the bottom left vertex of the (squares of the) points in $D^+(q)$ will enter $\square_q$ through the right edge. We thus obtain the following result.

Lemma 3. *Let t^* be the time that a square $\square_p$ of a point $p \in D(q)$ touches $\square_q$. We then have that*

(i) $r_q(t^*)_y = \ell_p(t^*)_y$, and $\ell_p(t^*)$ is the point with minimum y-coordinate among the points in $L^-(q)(t^*)$ at time t^*, if and only if $p \in D^-(q)$, and

(ii) $r_q(t^*)_x = \ell_p(t^*)_x$, and $\ell_p(t^*)$ is the point with minimum x-coordinate among the points in $L^+(q)(t^*)$ at time t^*, otherwise (i.e. if and only if $p \in D^+(q)$).

3 A Kinetic Data Structure for Growing Squares

In this section we present a data structure that can detect the first intersection among a dynamic set of disjoint growing squares. In particular, we describe a data structure that can detect intersections between all pairs of squares $\square_p, \square_q$ in P such that $p \in D^+(q)$. We build an analogous data structure for when $p \in D^-(q)$. This covers all intersections between pairs of squares $\square_p, \square_q$, where $p \in D(q)$. We then use four copies of these data structures, one for each quadrant, to detect the first intersection among all pairs of squares.

We describe the data structure itself in Section 3.1, and we briefly describe how to query it in Section 3.2. We deal with updates, e.g. inserting a new square into P or deleting an existing square from P, in Section 3.3. In Section 3.4 we analyze the total number of events that we have to process, and the time required to do so, when we grow the squares.

3.1 The Data Structure

Our data structure consists of two three-layered trees T^L and T^R, and a set of certificates linking nodes from T^L and T^R. These trees essentially form two 3D range trees on the centers of the squares in P, taking the third coordinate p_γ of each point to be their rank in the order (from left to right) along the line γ. The third layer of T^L doubles as a kinetic tournament tracking the bottom left vertices of squares. Similarly, T^R tracks the top right vertices of the squares.

The Layered Trees. The tree T^L is a 3D-range tree storing the center points in P. Each layer is implemented by a BB[α] tree [11], and each node μ corresponds to a canonical subset P_μ of points stored in the leaves of the subtree rooted at μ. The points are ordered on x-coordinate first, then on y-coordinate, and finally on γ-coordinate. Let L_μ denote the set of bottom left vertices of squares corresponding to the set P_μ, for some node μ.

Consider the associated structure X_v^L of some secondary node v. We consider X_v^L as a kinetic tournament on the x-coordinates of the points L_v [1]. More specifically, every internal node $w \in X_v^L$ corresponds to a set of points P_w consecutive along the line γ. Since the γ-coordinates of a point p and its bottom left vertex ℓ_p are equal, this means w also corresponds to a set of consecutive bottom left vertices L_w. Node w stores the vertex ℓ_p in L_w with minimum x-coordinate, and will maintain certificates that guarantee this [1].

The tree T^R has the same structure as T^L: it is a three-layered range tree on the center points in P. The difference is that a ternary structure X_v^R, for some secondary node v, forms a kinetic tournament maintaining the maximum x-coordinate of the points in R_v, where R_v are the top right vertices of the squares (with center points) in P_v. Hence, every ternary node $z \in X_v^R$ stores the vertex r_q with maximum x-coordinate among R_v. Let $\mathcal{X}^L$ and $\mathcal{X}^R$ denote the set of all kinetic tournament nodes in T^L and T^R, respectively.

Linking the Trees. Next, we describe how to add *linking certificates* between the kinetic tournament nodes in the trees T^L and T^R that guarantee the squares

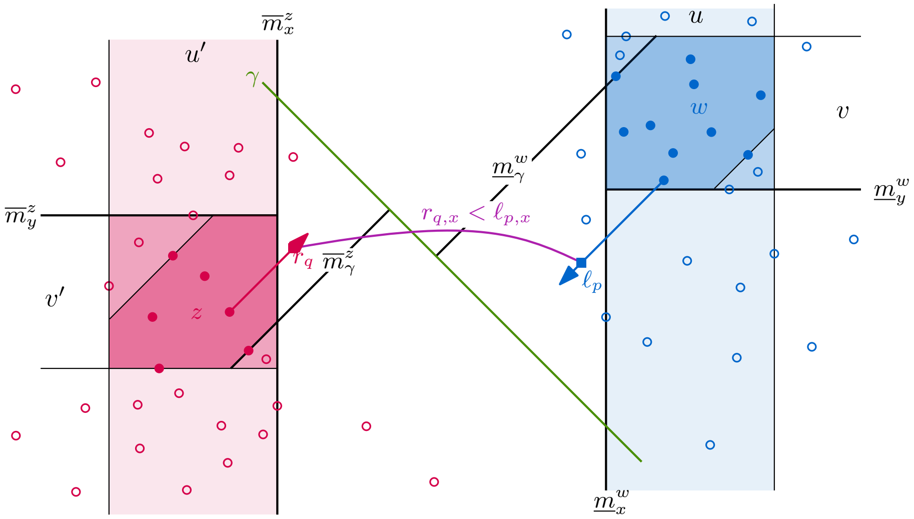


Fig. 3. The points $\overline{m}^z$ and $\underline{m}^w$ are defined by a pair of nodes $z \in \mathcal{X}^R_{v'}$, with $v' \in T_{u'}$, and $w \in X^L_v$, with $v \in T_u$. If $w \in Q^L(\overline{m}^z)$ and $z \in Q(\underline{m}^w)$ then we add a linking certificate between the rightmost upper right-vertex r_q, $q \in P_z$, and the leftmost bottom left vertex ℓ_p, $p \in P_w$.

are disjoint. More specifically, we describe the certificates, between nodes $w \in \mathcal{X}^L$ and $z \in \mathcal{X}^R$, that guarantee that the squares $\square_p$ and $\square_q$ are disjoint, for all pairs $q \in P$ and $p \in D^+(q)$.

Consider a point q. There are $O(\log^2 n)$ nodes in the secondary trees of T^L, whose canonical subsets together represent exactly $D(q)$. For each of these nodes v we can then find $O(\log n)$ nodes in X^L_v representing the points in $L^+(q)$. So, in total q is *interested in* a set $Q^L(q)$ of $O(\log^3 n)$ kinetic tournament nodes. It now follows from Lemma 3 that if we were to add certificates certifying that r_q is left of the point stored at the nodes in $Q^L(q)$ we can detect when $\square_q$ intersects with a square of a point in $D^+(q)$. However, as there may be many points q interested in a particular kinetic tournament node w, we cannot afford to maintain all of these certificates. The main idea is to represent all of these points q by a number of canonical subsets of nodes in T^R, and add certificates to only these nodes.

Consider a point p. Symmetric to the above construction, there are $O(\log^3 n)$ nodes in kinetic tournaments associated with T^R that together exactly represent the (top right corners of) the points q dominated by p and for which $p \in D^+(q)$. Let $Q^R(p)$ denote this set of kinetic tournament nodes.

Next, we extend the definitions of Q^L and Q^R to kinetic tournament nodes. To this end, we first associate each kinetic tournament node with a (query) point in $\mathbb{R}^3$. Consider a kinetic tournament node w in a tournament X^L_v, and let u be the node in the primary T^L for which $v \in T_u$. Let $\underline{m}^w = (\min_{a \in P_u} a_x, \min_{b \in P_v} b_y, \min_{c \in P_w} c_\gamma)$ be the point associated with w (note that we take the minimum over different sets P_u, P_v, and P_w for the different coordinates), and define $Q^R(w) = Q^R(\underline{m}^w)$. Symmetrically, for a node z in a tournament X_v^R, with $v \in T_u$ and $u \in T^R$, we define $\overline{m}^z = (\max_{a\in P_u} a_x, \max_{b\in P_v} b_y, \max_{c\in P_z} c_\gamma)$ and $Q^L(z) = Q^L(\overline{m}^z)$. See Fig. 3.

We now add a linking certificate between every pair of nodes $w \in \mathcal{X}^L$ and $z \in \mathcal{X}^R$ for which (i) w is a node in the canonical subset of z, that is $w \in Q^L(z)$, *and* (ii) vice versa, $z \in Q^R(w)$. Such a certificate will guarantee that the point r_q currently stored at z lies left of the point ℓ_p stored at w.

Lemma 4. *Every kinetic tournament node is involved in $O(\log^3 n)$ linking certificates, and thus every point p is associated with at most $O(\log^6 n)$ certificates.*

We now argue that we can still detect the first upcoming intersection.

Lemma 5. *Consider two sets of elements, say blue elements B and red elements R, stored in the leaves of two binary search trees T^B and T^R, respectively, and let $p \in B$ and $q \in R$, with $q < p$, be leaves in trees T^B and T^R, respectively. There is a pair of nodes $b \in T^B$ and $r \in T^R$, such that (i) $p \in P_b$ and $b \in C(T^B, [x', \infty))$, and (ii) $q \in P_r$ and $r \in C(T^R, (-\infty, x])$, where $x' = \max P_r$, $x = \min P_b$, and $C(T^S, I)$ denotes the minimal set of nodes in T^S whose canonical subsets together represent exactly the elements of $S \cap I$.*

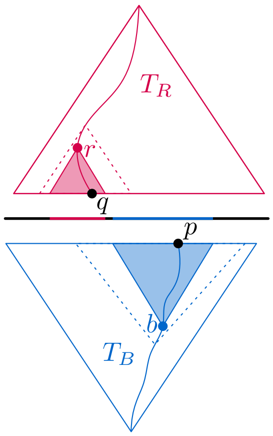


Fig. 4. The nodes b and r in the trees T^B and T^R.

Proof. Let b be the first node on the path from the root of T^B to p such that the canonical subset P_b of b is contained in the interval $[q, \infty)$, but the canonical subset of the parent of b is not. We define b to be the root of T^B if no such node exists. We define r to be the first node on the path from the root of T^R to q for which P_r is contained in $(-\infty, x]$ but the canonical subset of the parent is not. We again define r as the root of T^R if no such node exists (see Fig. 4). Clearly, we now have that r is one of the nodes whose canonical subsets form $R \cap (-\infty, x]$, and that $q \in P_r$ (as r lies on the search path to q). It is also easy to see that $p \in P_b$, as b lies on the search path to p. All that remains is to show that b is one of the canonical subsets that together form $B \cap [x', \infty)$. This follows from the fact that $q \leq x' < x \leq p$ —and thus P_b is indeed a subset of $[x', \infty)$— and the fact that the subset of the parent v of b contains an element smaller than q, and can thus not be a subset of $[x', \infty)$. $\square$

Lemma 6. *Let $\square_p$ and $\square_q$, with $p \in D^+(q)$, be the first pair of squares to intersect, at some time t^*, then there is a pair of nodes w, z that have a linking certificate that fails at time t^*.*

Proof. Consider the leaves representing p and q in T^L and T^R, respectively. By Lemma 5 we get that there is a pair of nodes $u \in T^L$ and $u' \in T^R$ that, among other properties, have $p \in P_u$ and $q \in P_{u'}$. Hence, we can apply Lemma 5 again on the associated trees of u and u', giving us nodes $v \in T_u$ and $v' \in T_{u'}$ which again have $p \in P_v$ and $q \in P_{v'}$. Applying Lemma 5 once more on X^L_v and $X^R_{v'}$ gives us nodes $w \in X^L_v$ and $z \in X^R_{v'}$ with $p \in P_w$ and $q \in P_z$. In addition, these three applications of Lemma 5 give us two points (x, y, γ) and (x', y', γ') where:

- P_u occurs as a canonical subset representing $P \cap ([x', \infty) \times \mathbb{R}^2)$,
- P_v occurs as a canonical subset representing $P_u \cap (\mathbb{R} \times [y', \infty) \times \mathbb{R})$, and
- P_w occurs as a canonical subset representing $P_v \cap (\mathbb{R}^2 \times [\gamma', \infty))$,

and such that

- $P_{u'}$ occurs as a canonical subset representing $P \cap ((-\infty, x] \times \mathbb{R}^2)$,
- $P_{v'}$ occurs as a canonical subset representing $P_{u'} \cap (\mathbb{R} \times (-\infty, y] \times \mathbb{R})$, and
- P_z occurs as a canonical subset representing $P_{v'} \cap (\mathbb{R}^2 \times (-\infty, \gamma])$.

Combining these first three facts, and observing that $\overline{m}^z = (x', y', \gamma')$ gives us that P_w occurs as a canonical subset representing $P \cap ([x', \infty) \times [y', \infty) \times [\gamma', \infty)) = D^+((x', y', \gamma'))$, and hence $w \in Q^L(\overline{m}^z) = Q^L(z)$. Analogously, combining the latter three facts and $\underline{m}^w = (x, y, \gamma)$ gives us $z \in Q^R(w)$. Therefore, w and z have a linking certificate. This linking certificate involves the leftmost bottom left vertex ℓ_a for some point $a \in P_w$ and the rightmost top right vertex r_b for some point $b \in P_z$. Since $p \in P_w$ and $q \in P_z$, we have that $r_q \leq r_b$ and $\ell_a \leq \ell_p$, and thus we detect their intersection at time t^*. □

From Lemma 6 it follows that we can now detect the first intersection between a pair of squares $\square_p, \square_q$, with $p \in D^+(q)$. We define an analogous data structure for when $p \in D^-(q)$. Following Lemma 3, the kinetic tournaments will maintain the vertices with minimum and maximum y-coordinate for this case. We then again link up the kinetic tournament nodes in the two trees appropriately.

Space Usage. Our trees T^L and T^R are range trees in $\mathbb{R}^3$, and thus use $O(n \log^2 n)$ space. However, it is easy to see that this is dominated by the space required to store the certificates. For all $O(n \log^2 n)$ kinetic tournament nodes we store at most $O(\log^3 n)$ certificates (Lemma 4), and thus the total space used by our data structure is $O(n \log^5 n)$. In Section 4 we will show that the number of certificates that we maintain is actually only $O(n(\log n \log\log n)^2)$. This means that our data structure also uses only $O(n(\log n \log\log n)^2)$ space.

3.2 Answering Queries

The basic query that our data structure supports is testing if a query square $\square_q$ currently intersects with a square $\square_p$ in P, with $p \in D^+(q)$. To this end, we simply select the $O(\log^3 n)$ kinetic tournament nodes whose canonical subsets together represent $D^+(q)$. For each node w we check if the x-coordinate of the lower-left vertex ℓ_p stored at that node (which has minimum x-coordinate among L_w) is smaller than the x-coordinate of r_q. If so, the squares intersect. The correctness of our query algorithm directly follows from Observation 2. The total time required for a query is $O(\log^3 n)$. Similarly, we can test if a given query point q is contained in a square $\square_p$, with $p \in D^+(q)$. Our complete data structure contains additional trees analogous to T^L that can be used to check if there is a square $\square_p \in P$ that intersects $\square_q$, with $p \in D^-(q)$ or p in one of the other quadrants defined by q.

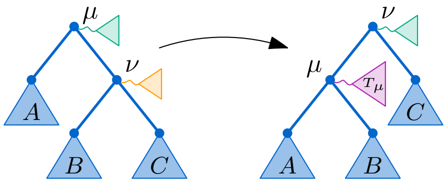


Fig. 5. After a left rotation around an edge (μ, ν), the associated data structure T_μ of node μ (pink) has to be rebuilt from scratch as its canonical subset has changed. For node ν we can reuse the old associated data of node μ. No other nodes are affected.

3.3 Inserting or Deleting a Square

At an insertion or deletion of a square $\square_p$ we proceed in three steps. (1) We update T^L and T^R, restoring range tree properties, and ensure that the ternary data structures are correct kinetic tournaments. (2) For each kinetic tournament node in $\mathcal{X}^L$ affected by the update, we query T^R to find a new set of linking certificates. We update $\mathcal{X}^R$ analogously. (3) We update the global event queue.

Lemma 7. *Inserting or deleting a square in T^L takes $O(\log^3 n)$ amortized time.*

Proof. We use the following standard procedure for updating the three-level BB[α] trees T^L in $O(\log^3 n)$ amortized time. An update (insertion or deletion) in a ternary data structure can easily be handled in $O(\log n)$ time. When we insert into or delete an element x in a BB[α] tree that has associated data structures, we add or remove the leaf that contains x, rebalance the tree by rotations, and finally add or remove x from the associated data structures. When we do a left rotation around an edge (μ, ν) we have to build a new associated data structure for node μ from scratch. See Fig. 5. Right rotations are handled analogously. It is well known that if building the associated data structure at node μ takes $O(|P_\mu| \log^c |P_\mu|)$ time, for some $c \geq 0$, then the costs of all rebalancing operations in a sequence of m insertions and deletions takes a total of $O(m \log^{c+1} n)$ time, where n is the maximum size of the tree at any time [10]. We can build a new kinetic tournament X_v^L for node v (using the associated data structures at its children) in linear time. Note that this cost excludes updating the global event queue. Building a new secondary tree T_v, including its associated kinetic tournaments, takes $O(|T_v| \log |T_v|)$ time. It then follows that the cost of our rebalancing operations is at most $O(m \log^2 n)$. This is dominated by the total number of nodes created and deleted, $O(m \log^3 n)$, during these operations. Hence, we can insert or delete a point (square) in T^L in $O(\log^3 n)$ amortized time. □

Clearly we can update T^R in $O(\log^3 n)$ amortized time as well. Next, we update the linking certificates. We say that a kinetic tournament node w in T^L is *affected by* an update if (i) the update added or removed a leaf node in the subtree rooted at w, (ii) node w was involved in a tree rotation, or (iii) w occurs in a newly built associated tree X_v^L (for some node v). Let $\mathcal{X}_i^L$ denote the set of nodes affected by update i ($\mathcal{X}_i^R$ of T^R is defined analogously). For each node $w \in \mathcal{X}_i^L$, we query

T^R to find the set of $O(\log^3 n)$ nodes whose canonical subsets represent $Q^R(w)$. For each node z in this set, we test if we have to add a linking certificate between w and z. As we show next, this takes constant time for each node z, and thus $O(\sum_i |\mathcal{X}_i^L| \log^3 n)$ time in total, for all nodes w (analogously for $\mathcal{X}_i^R$).

We have to add a link between a node $z \in Q^R(w)$ and w if and only if we also have $w \in Q^L(z)$. We test this as follows. Let v be the node whose associated tree X_v^L contains w, and let u be the node in T^L whose associated tree contains v. We have that $w \in Q^L(z)$ if and only if $u \in C(T^L, [\overline{m}_x^z, \infty))$, $v \in C(T_u, [\overline{m}_y^z, \infty))$, and $w \in C(X_v^L, [\overline{m}_\gamma^z, \infty))$. We can test each of these conditions in constant time:

Observation 8. Let q be a query point in $\mathbb{R}^1$, let w be a node in a binary search tree T, and let $x_p = \min P_p$ of the parent p of w in T, or $x_p = -\infty$ if no such node exists. We have that $w \in C(T, [q, \infty))$ if and only if $q \leq \min P_w$ and $q > x_p$.

Finally, we delete all certificates involving no longer existing nodes from our global event queue, and replace them by all newly created certificates. This takes $O(\log n)$ time per certificate. We charge the cost of deleting a certificate to when it gets created. Since every node w affected creates at most $O(\log^3 n)$ new certificates, all that remains is to bound the total number of affected nodes. Here we can use basically the same argument as when bounding the update time.

Lemma 9. *Inserting a disjoint square into P, or deleting a square from P takes $O(\log^7 n)$ amortized time.*

3.4 Running the Simulation

All that remains is to analyze the number of events processed, and the time to do so. Since each failure of a linking certificate produces an intersection, and thus an update the number of such events is at most the number of updates. To bound the number of events created by the tournament trees we use an argument similar to that of Agarwal et al. [1].

Theorem 10. *We can maintain a set P of n disjoint growing squares in a fully dynamic data structure such that we can detect the first time that a square $\square_q$ intersects with a square $\square_p$, with $p \in D^+(q)$. Our data structure uses $O(n(\log n \log\log n)^2)$ space, supports updates in $O(\log^7 n)$ amortized time, and queries in $O(\log^3 n)$ time. For a sequence of m operations, the structure processes a total of $O(m\alpha(n)\log^3 n)$ events in a total of $O(m\alpha(n)\log^7 n)$ time.*

Proof. We argued the bounds on the space usage, the query time, and the update time before. All that remains is to bound the number of events processed, and the time it takes to do so.

We start by the observation that each failure of a linking certificate produces an intersection, and thus a subsequent update. It thus follows that the number of such events is at most m.

To bound the number of events created by the tournament trees we extend the argument of Agarwal et al. [1]. For any kinetic tournament node w in T^L, the minimum x-coordinate corresponds to a lower envelope of line-segments in the t, x-space. This envelope has complexity $O(|P^*_w|\alpha(|P^*_w|)) = O(|P^*_w|\alpha(n))$, where P^*_w is the multiset of points that ever occur in P_w, i.e. that are stored in a leaf of the subtree rooted at w at some time t. Hence, the number of tournament events involving node w is also at most $O(|P^*_w|\alpha(n))$. It then follows that the total number of events is proportional to the size of these sets P^*_w, over all w in our tree. As in Lemma 7, every update directly contributes one point to $O(\log^3 n)$ nodes. The remaining contribution is due to rebalancing operations, and this cost is again bounded by $O(m \log^2 n)$. Thus, the total number of events processed is $O(m\alpha(n) \log^3 n)$.

At every event, we have to update the $O(\log^3 n)$ linking certificates of w. This can be done in $O(\log^4 n)$ time (including the time to update the global event queue). Thus, the total time for processing all kinetic tournament events in T^L is $O(m\alpha(n) \log^7 n)$. The analysis for the kinetic tournament nodes z in T^R is analogous. □

To simulate the process of growing the squares in P, we now maintain eight copies of the data structure from Theorem 10: two data structures for each quadrant (one for D^+, the other for D^-). We thus obtain the following result.

Theorem 11. *We can maintain a set P of n disjoint growing squares in a fully dynamic data structure such that we can detect the first time that two squares in P intersect. Our data structure uses $O(n(\log n \log\log n)^2)$ space, supports updates in $O(\log^7 n)$ amortized time, and queries in $O(\log^3 n)$ time. For a sequence of m operations, the structure processes $O(m\alpha(n) \log^3 n)$ events in a total of $O(m\alpha(n) \log^7 n)$ time.*

And thus we obtain the following agglomerative glyph clustering solution.

Theorem 12. *Given a set of n initial square glyphs P, we can compute an agglomerative clustering of the squares in P in $O(n\alpha(n) \log^7 n)$ time using $O(n(\log n \log\log n)^2)$ space.*

4 Efficient Representation of Dominance Relations

The linking certificates of our data structure actually comprise an efficient representation of all dominance relations between two point sets. This representation, and in particular the tighter analysis in this section, is of independent interest.

Let R and B be two point sets in $\mathbb{R}^d$ with $|R| = n$ and $|B| = m$, and let T^R and T^B be range trees built on R and B, respectively. We assume that each layer of T^R and T^B is a BB[α]-tree. By definition, every node u on the lowest layer of T^R or T^B has an associated d-dimensional range Q_u (the hyper-box, not the subset of points). For a node $u \in T^R$, we consider the subset of points in B that dominate all points in Q_u, which can be comprised of $O(\log^d m)$ canonical subsets of B, represented by nodes in T^B. Similarly, for a node $v \in T^B$, we consider the subset of points in R that are dominated by all points in Q_v, which can be represented by $O(\log^d n)$ nodes in T^R. We now link a node $u \in T^R$ and a node $v \in T^B$ if and only if v represents such a canonical subset for u and vice versa. By repeatedly applying Lemma 5 for each dimension, it can easily be shown that these links represent all dominance relations between R and B.

As a d-dimensional range tree consists of $O(n \log^{d-1} n)$ nodes, a trivial bound on the number of links is $O(m \log^{2d-1} n)$ (assuming $n \geq m$). Below we show that the number of links can be bounded by $O(n(\log n \log\log n)^{d-1})$.

Analyzing the Number of Links in 1D. Let R and B be point sets in $\mathbb{R}$ with $|R| = n$, $|B| = m$, and $n \geq m$. Now, every associated range of a node u in T^R or T^B is an interval I_u. We extend the interval to infinity in one direction; to the left for $u \in T^R$, and to the right for $u \in T^B$. For analysis purposes we construct another range tree T on $R \cup B$, where T is not a BB[α]-tree, but instead a perfectly balanced tree with height $\lceil \log(n+m) \rceil$. For convenience we slightly expand the associated intervals of T so that all points in $R \cup B$ are interior to the associated intervals. We associate a node u in T^R or T^B with a node v in T if the endpoint of I_u is contained in the associated interval I_v of v.

Observation 13. Nodes of T^R or T^B are associated with at most one node per level of T.

For two intervals $I_u = (-\infty, a]$ and $I_v = [b, \infty)$, corresponding to a node $u \in T^R$ and a node $v \in T^B$, let $[a, b]$ be the *spanning interval* of u and v. We now want to charge spanning intervals of links to nodes of T. We charge a spanning interval $I_{uv} = [a, b]$ to a node w of T if and only if $[a, b]$ is a subset of I_w, and $[a, b]$ is cut by the splitting coordinate of w. Clearly, every spanning interval can be charged to exactly one node of T. Now, for a node u of T, let $h_R(u)$ be the height of the highest node of T^R associated with u, and let $h_B(u)$ be the height of the highest node of T^B associated with u.

Lemma 14. *$O(h_R(u) \cdot h_B(u))$ spanning intervals are charged to a node u of T.*

Proof. Let x be the splitting coordinate of u and let $r \in T^R$ and $b \in T^B$ form a spanning interval that is charged to u. We claim that, using the notation introduced in Lemma 5, $r \in C(T^R, (-\infty, x])$ (and symmetrically, $b \in C(T^B, [x, \infty))$). Let $I_b = [x', \infty)$ be the associated interval of b, where $x' > x$. By definition, $r \in C(T^R, (-\infty, x'])$. If $r \notin C(T^R, (-\infty, x])$, then the right endpoint of I_r must lie between x and x'. But then the spanning interval of r and b would not be charged to u. As a result, we can only charge spanning intervals between $h_R(u)$ nodes of T^R and $h_B(u)$ nodes of T^B, of which there are $O(h_R(u) \cdot h_B(u))$. □

Using Lemma 14, we count the total number of charged spanning intervals and hence, links between T^R and T^B. We refer to this number as $N(T^R, T^B)$. This is simply $\sum_{u \in T} O(h_R(u) \cdot h_B(u)) \leq \sum_{u \in T} O(h_R(u)^2 + h_B(u)^2)$. We can split the sum and assume w.l.o.g. that $N(T^R, T^B) \leq 2 \sum_{u \in T} O(h_R(u)^2)$. Rewriting the sum based on heights in T^R and writing $n_T(h_R)$ for the number of nodes of T that have a node of height h_R associated with it gives

$$N(T^R, T^B) \leq \sum_{h_R = 0}^{height(T^R)} n_T(h_R) \cdot O(h_R^2).$$

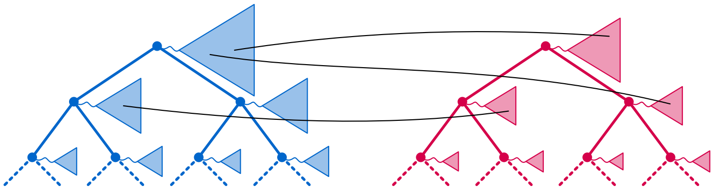


Fig. 6. Two layered trees with two layers, and the links between them (sketched in black). We are interested in bounding the number of such links.

To bound $n_T(h)$ we use Observation 13 and the fact that T^R is a $\textsc{bb}[\alpha]$ tree. Let $c = \frac{1}{1-\alpha}$, then we get that $\mathit{height}(T^R) \leq \log_c(n)$ from properties of $\textsc{bb}[\alpha]$ trees. Therefore, the number of nodes in T^R that have height h is at most $O(\frac{n}{c^h})$.

Lemma 15. $n_T(h) = O\left(\frac{(n+m)h}{c^h}\right)$.

Proof. As argued, there are at most $O(n/c^h)$ nodes in T^R of height h. Consider cutting the tree T at level $\log(n/c^h)$. This results in a top tree of size $O(n/c^h)$, and $O(n/c^h)$ bottom trees. Clearly, the top tree contributes at most its size to $n_T(h)$. All bottom trees have height at most $\lceil \log(n+m) \rceil - \log(n/c^h) = O(\log(c^h) + \log(1 + m/n)) = O(h + m/n)$. Every node in T^R of height h can, in the worst case, be associated with one distinct node per level in the bottom trees by Observation 13. Hence, the bottom trees contribute at most $O(n(h+m/n)/c^h) = O((nh+m)/c^h) = O((n+m)h/c^h)$ to $n_T(h)$. □

Using this bound on $n_T(h)$ in the sum we previously obtained gives

$$N(T^R, T^B) \leq \sum_{h_R=0}^{\mathit{height}(T^R)} O\left(\frac{(n+m)h_R^3}{c^{h_R}}\right) \leq O(n+m) \sum_{h=0}^{\infty} \frac{h^3}{c^h} = O(n+m).$$

Where indeed, $\sum_{h=0}^{\infty} \frac{h^3}{c^h} = O(1)$ because $c > 1$. Thus, we conclude:

Theorem 16. *The number of links between two* 1*-dimensional range trees* T^R *and* T^B *containing* n *and* m *points, respectively, is bounded by* $O(n+m)$*.*

Extending to Higher Dimensions. We now extend the bound to d dimensions. We first determine the links for the top-layer of the range trees. This results in links between associated range trees of $d-1$ dimensions (see Fig. 6). We then bound the number of links within the linked associated trees by induction on d.

Theorem 17. *The number of links between two* d*-dimensional range trees* T^R *(on* n *points) and* T^B *(on* $m \leq n$ *points), is bounded by* $O(n(\log n \log\log n)^{d-1})$*.*

Proof. We show by induction on d that the number of links is bounded by the minimum of $O(n(\log n \log\log n)^{d-1})$ and $O(m \log^{2d-1} n)$. The second bound is the trivial bound stated above. The base case $d = 1$ is provided by Theorem 16.

Consider the case $d > 1$. We first determine the links for the top-layer of T^R and T^B. Now consider the links between an associated tree T_u in T^R containing k points and other associated trees $T_0, \ldots, T_r$ that contain at most k points. Since T_u can be linked with only one associated tree per level, and because both range trees use BB[α] trees, the number of points $m_0, \ldots, m_r$ in $T_0, \ldots, T_r$ satisfy $m_i \leq k/c^i$ $(0 \leq i \leq r)$ where $c = \frac{1}{1-\alpha}$. By induction, the number of links between T_u and T_i is bounded by the minimum of $O(k(\log n \log\log n)^{d-2})$ and $O(m_i \log^{2d-3} n)$. Now let $i^* = \log_c(\log^{d-1} n) = O(\log\log n)$. Then, for $i \geq i^*$, we get that $O(m_i \log^{2d-3} n) = O(k \log^{d-2} n)$. Since the sizes of the associated trees decrease geometrically, the total number of links between T_u and T_i for $i \geq i^*$ is bounded by $O(k \log^{d-2} n)$. The links with the remaining trees can be bounded by $O(k \log^{d-2} n(\log\log n)^{d-1})$. Finally note that the top-layer of each range tree has $O(\log n)$ levels, and that each level contains n points in total. Thus, we obtain $O(n \log^{d-1} n(\log\log n)^{d-1})$ links in total. The remaining links for which the associated tree in T^B is larger than in T^R can be bounded analogously. □

It follows from Theorem 17 that the number of certificates maintained, and thus the space used, by our data structure from Section 3 is only $O(n(\log n \log\log n)^2)$.

References

1. P. K. Agarwal, H. Kaplan, and M. Sharir. Kinetic and Dynamic Data Structures for Closest Pair and All Nearest Neighbors. *ACM Trans. Alg.*, 5(1):4:1–4:37, 2008.
2. H.-K. Ahn, S. W. Bae, J. Choi, M. Korman, W. Mulzer, E. Oh, J.-W. Park, A. van Renssen, and A. Vigneron. Faster Algorithms for Growing Prioritized Disks and Rectangles. In *Proc. 28th Intern. Symp. Alg. Comp.*, pages 3:1–3:13, 2017.
3. G. Alexandron, H. Kaplan, and M. Sharir. Kinetic and dynamic data structures for convex hulls and upper envelopes. *Comp. Geom. Theory Appl.*, 36(2):144–1158, 2007.
4. D. Bahrdt, M. Becher, S. Funke, F. Krumpe, A. Nusser, M. Seybold, and S. Storandt. Growing Balls in $\mathbb{R}^d$. In *Proc. 19th Workshop Alg. Eng. Exp.*, pages 247–258, 2017.
5. T. Castermans, B. Speckmann, F. Staals, and K. Verbeek. Agglomerative clustering of growing squares. *ArXiv e-prints*, 2017.
6. T. Castermans, B. Speckmann, K. Verbeek, M. A. Westenberg, A. Betti, and H. van den Berg. GlamMap: Geovisualization for e-Humanities. In *Proc. 1st Workshop Vis. Dig. Hum.*, 2016.
7. S. Funke, F. Krumpe, and S. Storandt. Crushing Disks Efficiently. In *Proc. 27th Intern. Workshop Comb. Alg.*, pages 43–54, 2016.
8. S. Funke and S. Storandt. Parametrized Runtimes for Ball Tournaments. In *Proc. 33rd Europ. Workshop Comp. Geom.*, pages 221–224, 2017.
9. L. Guibas. Kinetic data structures. In D. P. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*, pages 23-1–23-18. CRC Press, 2004.
10. K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching.* Springer, 1984.
11. J. Nievergelt and E. M. Reingold. Binary Search Trees of Bounded Balance. *SIAM J. Comp.*, 2(1):33–43, 1973.