Agglomerative Clustering of Growing Squares*

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

1 **TU Eindhoven**
[t.h.a.castermans|b.speckmann|k.a.b.verbeek]@tue.nl
2 **Utrecht University**
f.staals@uu.nl

1 Introduction

We study an agglomerative clustering problem motivated by interactive glyphs in geo-visualization. *GlamMap* [5] is a visual analytics tool for the eHumanities which allows the user to interactively explore 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 glyphs start to overlap, they are merged into larger glyphs to keep the map clear and uncluttered. To allow the user to filter and browse real world data sets at interactive speed we hence need an efficient agglomerative clustering algorithm for growing squares (glyphs).

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. Furthermore, we refer to P as a set of squares rather than a 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 $z_w = p_w + q_w$. Our goal is to compute the complete sequence of events where squares intersect and merge.

Results. We present a fully dynamic data structure that can maintain a set P of n disjoint growing squares. Our data structure can report the first time two squares in P intersect, and supports updates (inserting or deleting a square) in $O(\log^7 n)$ amortized time. Queries asking whether a query square $\square_q$ currently intersects a square $\square_p$ in P take $O(\log^3 n)$ time, and the space usage is $O(n(\log n \log\log n)^2)$. Using this data structure we can 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. This abstract focuses on the update and query times for our data structure. Omitted proofs and detailed bounds on space usage, as well as related discussions on the relation between canonical subsets in dominance queries, can be found in the full version [4].

Related Work. Funke, Krumpe, and Storandt [6] introduced so-called "ball tournaments". 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 lower priority ball is eliminated. The goal is to

* 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).

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

compute the elimination sequence efficiently. Funke and Storandt [7] 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 and Δ the ratio of the largest to the smallest radius. Ahn et al. [2] recently developed the first sub-quadratic algorithms to compute elimination orders for ball tournaments. Their results apply to balls and boxes in two dimensions or higher. Specifically, for squares in 2D they can compute an elimination order in $O(n \log^4 n)$ time. Their results critically depend on the fact that the elimination priorities are given and that they have to handle only deletions.

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.

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 2.1.** Let $p \in D(q)$ be a point dominating 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.

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 points in $D(q)$ 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. 1).

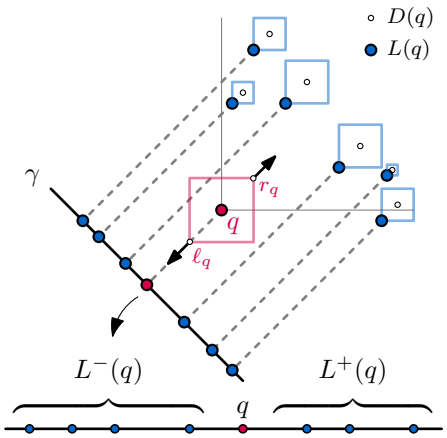

Figure 1 The projection of the square centers and relevant corners onto line γ.

▶ **Observation 2.2.** 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 $p \in D^-(q)$;
- $\ell_x = r_x$ and $\ell_y \leq r_y$ if and only if $p \in D^+(q)$.

Observation 2.2 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 2.3.** *Let t^* be the time that a square $\square_p$ of a point $p \in D(q)$ touches $\square_q$. We have*

(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 $p \in D^-(q)$, 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^** *if $p \in D^+(q)$.*

3 A Kinetic Data Structure for Growing Squares

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 $p \in D^-(q)$, and then use four copies of these data structures, one for each quadrant, to detect the first intersection among all pairs of 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 [8], 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 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_v^L 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 2.3 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

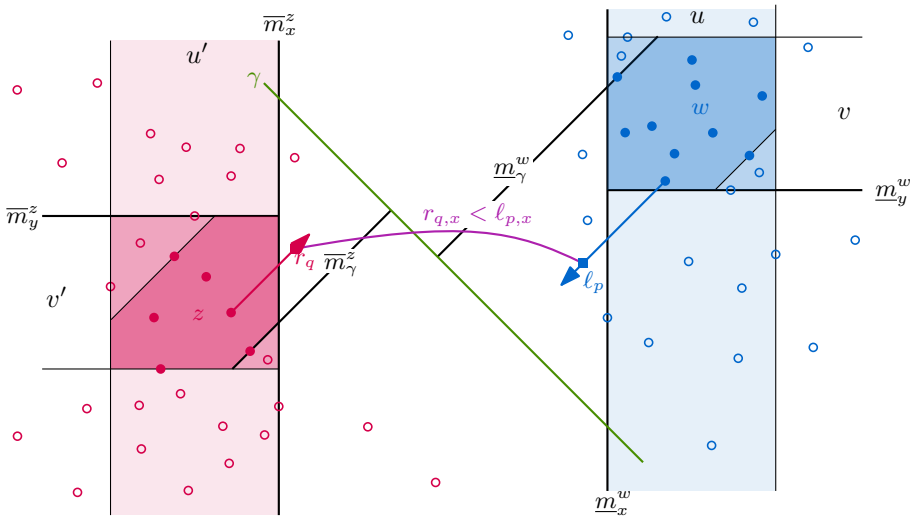

Figure 2 The points $\overline{m}^z$ and $\underline{m}^w$ are defined by a pair of nodes $z \in \mathcal{X}^R_{v'}$, with $v' \in T_{u'}$, and $w \in X^L_v$, with $v \in T_u$. If $w \in Q^L(\overline{m}^z)$ and $z \in Q(\underline{m}^w)$ then we add a linking certificate between the rightmost upper right-vertex r_q, $q \in P_z$, and the leftmost bottom left vertex ℓ_p, $p \in P_w$.

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^R_v, 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. 2.

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 3.1.** *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 3.2.** *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. There is a pair of nodes $b \in T^B$ and $r \in T^R$, such that (i) $p \in P_b$ and $b \in C(T^B, [\max P_r, \infty))$, and (ii) $q \in P_r$ and $r \in C(T^R, (-\infty, \min P_b])$, where $C(T^S, I)$ denotes the minimal set of nodes in T^S whose canonical subsets together represent exactly the elements of $S \cap I$.*

▶ **Lemma 3.3.** *Let $\square_p$ and $\square_q$, with $p \in D^+(q)$, be the first pair of squares to intersect, at some time t^*. There is a pair of nodes w, z that have a linking certificate that fails at time t^*.*

From Lemma 3.3 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 2.3, 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 3.1), and thus the total space used by our data structure is $O(n \log^5 n)$. In the full version [4], we show that the number of certificates that we maintain (and thus the space used by our data structure) is actually only $O(n(\log n \log\log n)^2)$.

3.2 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 3.4.** *Inserting or deleting a square in T^L takes $O(\log^3 n)$ 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^L_v (for some node v). Let $\mathcal{X}^L_i$ denote the set of nodes affected by update i ($\mathcal{X}^R_i$ of T^R is defined analogously). For each node $w \in \mathcal{X}^L_i$, 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 $O(\sum_i |\mathcal{X}^L_i| \log^3 n)$ time in total, for all nodes w (analogously for $\mathcal{X}^R_i$).

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^L_v 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 $u \in C(T^L, [\overline{m}^z_x, \infty))$, $v \in C(T_u, [\overline{m}^z_y, \infty))$, and $w \in C(X^L_v, [\overline{m}^z_\gamma, \infty))$. We can test each of these conditions in constant time:

▶ **Observation 3.5.** 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 3.6.** *Inserting a disjoint square into P, or deleting a square from P takes $O(\log^7 n)$ amortized time.*

3.3 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 3.7.** *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.*

To simulate the process of growing the squares in P, we now maintain eight copies of the data structure from Theorem 3.7: two data structures for each quadrant (one for D^+, the other for D^-). Using these data structures we obtain the following agglomerative glyph clustering solution.

▶ **Theorem 3.8.** *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.*

References

1 P. K. Agarwal, H. Kaplan, and M. Sharir. Kinetic and Dynamic Data Structures for Closest Pair and All Nearest Neighbors. *ACM Transactions on Algorithms*, 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 *Proceedings of the 28th International Symposium on Algorithms and Computation*, 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. *Computational Geometry: Theory and Applications*, 36(2):144–1158, 2007.

4 T. Castermans, B. Speckmann, F. Staals, and K. Verbeek. Agglomerative clustering of growing squares. *ArXiv e-prints*, 2017. `arXiv:1706.10195`.

5 T. Castermans, B. Speckmann, K. Verbeek, M. A. Westenberg, A. Betti, and H. van den Berg. GlamMap: Geovisualization for e-Humanities. In *Proceedings of the 1st Workshop on Visualization for the Digital Humanities*, 2016.

6 S. Funke, F. Krumpe, and S. Storandt. Crushing Disks Efficiently. In *Proceedings of the 27th International Workshop on Combinatorial Algorithms*, pages 43–54, 2016.

7 S. Funke and S. Storandt. Parametrized Runtimes for Ball Tournaments. In *Proceedings of the 33rd European Workshop on Computational Geometry*, pages 221–224, 2017.

8 J. Nievergelt and E. M. Reingold. Binary Search Trees of Bounded Balance. *SIAM Journal on Computing*, 2(1):33–43, 1973.