Dynamic Geometric Connectivity in the Plane with Constant Query Time

Timothy M. Chan
Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA

Zhengcheng Huang
Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA

Abstract

We present the first fully dynamic connectivity data structures for geometric intersection graphs achieving *constant* query time and *sublinear* amortized update time for many classes of geometric objects in 2D. Our data structures can answer connectivity queries between two objects, as well as "global" connectivity queries (e.g., deciding whether the entire graph is connected). Previously, the data structure by Afshani and Chan (ESA'06) achieved such bounds only in the special case of axis-aligned line segments or rectangles but did not work for arbitrary line segments or disks, whereas the data structures by Chan, Pătraşcu, and Roditty (FOCS'08) worked for more general classes of geometric objects but required $n^{\Omega(1)}$ query time and could not handle global connectivity queries.

Specifically, we obtain new data structures with $O(1)$ query time and amortized update time near $n^{4/5}$, $n^{7/8}$, and $n^{20/21}$ for axis-aligned line segments, disks, and arbitrary line segments respectively. Besides greatly reducing the query time, our data structures also improve the previous update times for axis-aligned line segments by Afshani and Chan (from near $n^{10/11}$ to $n^{4/5}$) and for disks by Chan, Pătraşcu, and Roditty (from near $n^{20/21}$ to $n^{7/8}$).

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Connectivity, dynamic data structures, geometric intersection graphs

Digital Object Identifier 10.4230/LIPIcs.SoCG.2024.36

Related Version *Full Version*: `https://arxiv.org/abs/2402.05357`

Funding *Timothy M. Chan*: Work supported by NSF Grant CCF-2224271.

1 Introduction

Dynamic graph connectivity – maintaining an undirected graph under edge insertions and deletions to answer connectivity queries – is a popular topic in data structure design and dynamic graph algorithms [15, 16, 24, 11, 17]. At STOC'02, Chan [5] initiated the study of dynamic connectivity in *geometric* settings:

> Maintain a set of n geometric objects, subject to insertions and deletions of objects, so that we can quickly determine whether two query objects are connected in the intersection graph.

The challenge here is that a single insertion/deletion of an object may change $\Omega(n)$ edges in the intersection graph in the worst case, and so we can't afford to maintain the intersection graph explicitly.

Previous work by Chan. Chan [5] obtained the first fully dynamic data structure with sublinear ($O(n^{0.94})$) amortized update time and sublinear ($\tilde{O}(m^{1/3})$)[1] query time for axis-aligned line segments or rectangles in 2D or axis-aligned boxes in any constant dimension. The exponent (0.94) arose from matrix multiplication. The heart of his solution was a data structure for an extension of dynamic graph connectivity with vertex (re)insertions and deletions, called *dynamic subgraph connectivity* – maintaining a (sparse) graph with m edges and a subset A of "active" vertices under (re)insertions and deletions in A, so that we can quickly determine whether two vertices are connected in the subgraph induced by A. As Chan observed, the geometric problem in the case of axis-aligned boxes can be reduced to dynamic subgraph connectivity (ignoring polylogarithmic factors), by using so-called "biclique covers" (which are related to range searching or intersection searching).

For more general classes of geometric objects such as arbitrary line segments, biclique covers have superlinear complexity, so Chan's reduction was not sufficient to yield sublinear update time, unfortunately.

Previous work by Chan, Pătrașcu, and Roditty. A subsequent paper by Chan, Pătraşcu, and Roditty [9] (FOCS'08) rectified the problem: They first obtained a better data structure for dynamic subgraph connectivity, avoiding fast matrix multiplication, and then incorporated range searching techniques into their data structure in a more efficient way, thereby obtaining data structures for dynamic geometric connectivity with sublinear amortized update time for virtually all families of objects with constant description complexity. For example, for axis-aligned line segments or rectangles in 2D and axis-aligned boxes in any constant dimension, the amortized update time is $\tilde{O}(n^{2/3})$ and query time is $\tilde{O}(n^{1/3})$; for arbitrary line segments in 2D, the amortized update time is $O^*(n^{9/10})$ and query time is $\tilde{O}(n^{1/5})$; for disks in 2D, the amortized update time is $O^*(n^{20/21})$ and query time is $O^*(n^{1/7})$.

Recently, Jin and Xu [18] proved an $\Omega(m^{2/3-\varepsilon})$ conditional lower bound on the amortized update time for the dynamic subgraph connectivity problem for any data structure with $O(m^{1-\varepsilon})$ query time,[2] assuming the "Combinatorial 4-Clique Hypothesis", if we restrict ourselves to "combinatorial" algorithms that do not use algebraic techniques for fast matrix multiplication. Chan [5] observed that dynamic subgraph connectivity with m edges can be reduced back to dynamic geometric connectivity for $O(m)$ axis-aligned line segments or boxes in 3D (roughly speaking, because in 3D, axis-aligned segments or boxes can "simulate" arbitrary graphs), and so Jin and Xu's proof implies conditional optimality of Chan, Pătraşcu, and Roditty's $\tilde{O}(n^{2/3})$ update bound for axis-aligned boxes in dimension 3 and above, at least with respect to combinatorial algorithms.

However, one major drawback in Chan, Pătraşcu, and Roditty's data structures is that the query times are super-polylogarithmic (unlike known static data structures). One can envision applications where queries occur much more frequently than updates. Although trade-offs with smaller query time and larger update time are possible with Chan, Pătraşcu, and Roditty's data structures, they do not achieve sublinear update time in the regime of constant or polylogarithmic query time.

In some ways, their data structures encode connectivity information only "implicitly". Thus, another drawback is that they cannot handle "global" connectivity queries, e.g., deciding whether the entire intersection graph is connected, or counting the number of connected

[1] Throughout the paper, the $\tilde{O}$ notation hides $\log^{O(1)} n$ factors, and the O^* notation hides n^{ε} factors for an arbitrarily small constant $\varepsilon > 0$.
[2] Throughout the paper, ε denotes an arbitrarily small positive constant.

components. Abboud and Vassilevska Williams [1] proved an $\Omega(m^{1-\varepsilon})$ conditional lower bound on the query/update time for global connectivity queries for the dynamic subgraph connectivity problem, assuming the Strong Exponential-Time Hypothesis. So, sublinear query and update time are conditionally not possible for global connectivity queries for axis-aligned segments and boxes in dimension 3 and above. Still, a fundamental question remains as to whether sublinear update time bound is achievable for global connectivity queries for various types of geometric objects in 2D (which is where the most natural geometric applications occur).

Previous work by Afshani and Chan. A different (earlier) data structure by Afshani and Chan [2] (ESA'06) addressed some of these drawbacks. Specifically, they obtained $O(1)$ query time and $\tilde{O}(n^{10/11})$ update time in the case of axis-aligned line segments or rectangles in 2D. Unfortunately, they were unable to extend their method to other types of objects in 2D, such as arbitrary line segments or disks.

Other related work. Better data structures are known for some easier special cases; for example, a recent SoCG'22 paper by Kaplan et al. [19] gave polylogarithmic results for unit disks, or disks with small maximum-to-minimum-radius ratio, or for arbitrary disks in the incremental (insertion-only) or decremental (deletion-only) case (see also Kaplan et al.'s SOSA'24 paper [20]). However, these results are not applicable to arbitrary disks in the fully dynamic setting.

New results. We obtain new data structures for dynamic geometric connectivity for different types of objects in 2D, as summarized in Table 1.[3]

Qualitatively, we obtain the first data structures with *constant* query time and sublinear amortized update time for arbitrary line segments in 2D as well as disks in 2D, and the first data structures that can handle global connectivity queries for such objects. The line segment case is especially general (since it can handle arbitrary polylines of bounded complexity). Our approach can in fact handle any family of semialgebraic curves with constant description complexity in 2D, though the exponent in the update bound depends on the degree of the curves.

Quantitatively, for the case of disks in 2D, our data structure not only greatly reduces the query time of Chan, Pătraşcu, and Roditty's method but does so without hurting the update time – in fact, the update bound is *improved* (albeit slightly), from near $n^{20/21}$ to $n^{7/8}$. For the special case of axis-aligned line segments in 2D, our data structure also improves the update time of Afshani and Chan's method, from near $n^{10/11}$ to $n^{4/5}$.

Techniques and new ideas. Our approach builds on Afshani and Chan's method [2], based on the key notion of *equivalence classes* of connected components. They proved combinatorial bounds and described efficient algorithms for computing and maintaining such classes. Although their combinatorial bounds actually hold for arbitrary geometric objects in 2D, their algorithms are specialized to *axis-aligned* objects in 2D. We describe a different way

[3] For the previous result on simplices in $\mathbb{R}^d$, Chan, Pătraşcu, and Roditty [9] originally stated query time $O^*(n^{1/(2d+1)})$ and update time $O^*(n^{1-1/d(2d+1)})$, but they mistakenly assumed bounds for simplex range searching extend to simplex intersection searching. However, simplex intersection searching in $\mathbb{R}^d$ does reduce to semialgebraic range searching in $\mathbb{R}^{O(d^2)}$, so their framework implies the sublinear bounds shown in the table.

Table 1 Previous and new results on dynamic geometric connectivity.

Type of objects	Query time	Update time (amort.)	Ref.
Axis-aligned segments/boxes in $\mathbb{R}^d$	$\tilde{O}(n^{1/3})$	$\tilde{O}(n^{2/3})$	[9]
Arbitrary line segments in $\mathbb{R}^2$	$O^*(n^{1/5})$	$O^*(n^{9/10})$	
Disks in $\mathbb{R}^2$	$O^*(n^{1/7})$	$O^*(n^{20/21})$	
Balls in $\mathbb{R}^d$	$O^*(n^{\frac{1}{2d+3}})$	$O^*(n^{1-\frac{1}{(d+1)(2d+3)}})$	
Simplices in $\mathbb{R}^d$	$O^*(n^{\frac{1}{\Theta(d^2)}})$	$O^*(n^{1-\frac{1}{\Theta(d^4)}})$	
Axis-aligned segments/rectangles in $\mathbb{R}^2$	$O(1)$	$\tilde{O}(n^{10/11})$	[2]
Axis-aligned segments in $\mathbb{R}^2$	$O(1)$	$\tilde{O}(n^{4/5})$	
Arbitrary line segments in $\mathbb{R}^2$	$O(1)$	$O^*(n^{20/21})$	new
Disks in $\mathbb{R}^2$	$O(1)$	$O^*(n^{7/8})$	

to compute and maintain equivalence classes of components, based on a *repeated splitting* idea which is simple in hindsight, but has somehow been overlooked by researchers for more than a decade. This idea allows us to reduce equivalence-class data structures to *colored* variants of range or intersection searching, which can be solved by known geometric data structuring techniques. This new idea allows us to obtain constant query time and sublinear amortized update time for all the types of 2D geometric objects considered here; see Sections 4–5.

Of independent interest is also a new variant of Afshani and Chan's combinatorial lemma, which helps in reducing the update time further; see Section 3.

In the full version of this paper [8], we obtain a still further improvement for the case of disks by incorporating another new idea – namely, the usage of *separators* (specifically, Smith and Wormald's geometric separator theorem [25]). This is interesting, as separators have not been widely used before in the context of dynamic geometric connectivity (they have been used in dynamic subgraph connectivity for planar graphs [13] but not for more general classes of geometric intersection graphs).

2 Recap of Afshani and Chan's Method

Before describing our new data structures, we first review Afshani and Chan's previous method for dynamic connectivity for axis-aligned segments/rectangles in 2D [2]. To illustrate the overall ideas, for simplicity we will focus on the *offline* setting of the problem, where the upcoming updates are known in advance (the online setting requires more work and slightly worse update time), and we will focus on just the case of axis-aligned segments.

As often seen in previous works [4, 5], Afshani and Chan used the standard technique of handling insertions lazily and rebuilding periodically. Each period phase consists of a preprocessing step followed by q updates, so that the preprocessing cost can be amortized. In the offline setting, we know which objects will be updated in each phase.

Let S be the objects that exist at the beginning of the phase and will not be deleted during the phase, and let Q be the sequence of objects $s_1, \ldots, s_q$ that will be updated during the phase. (Thus, S stays static during a phase, whereas Q is small.) Afshani and Chan defined two connected components of S to be *equivalent* if they intersect the exact same set of objects from Q. Clearly, for any $s \in Q$, if s intersects an arbitrary component from a class L, then s intersects all components from L. More importantly, using the fact that distinct connected components never intersect each other and may be viewed as a collection of disjoint "curves" or "strings" (not necessarily of constant complexity) when the objects are in $\mathbb{R}^2$, Afshani and Chan proved a polynomial upper bound on the number of equivalence classes at any moment. The general combinatorial lemma is stated as follows.

▶ **Lemma 1** (Afshani and Chan's combinatorial lemma [2]). *Consider a set Q of q disjoint regions with simple connected boundaries and a set C of disjoint curves in $\mathbb{R}^2$. Then C consists of at most $O(q^3)$ equivalence classes with respect to Q. (This bound is tight.)*

To readers familiar with the notion of "VC dimension" or "shatter dimension" of set systems [23], the above lemma is equivalent to the statement that the set system $(Q, \mathcal{R})$ with $\mathcal{R} = \{\{s \in Q : s \cap c \neq \emptyset\} : c \in C\}$ has shatter dimension at most 3. We stress that what makes the lemma interesting is that the curves in C do not need to have constant complexity. On the other hand, disjointness of the curves in C is crucial (otherwise, it is easy to find counterexamples with exponentially many equivalence classes).

In the dynamic connectivity problem, the inserted segments are not disjoint. However, the arrangement of Q can be broken down into $O(q^2)$ non-intersecting sub-segments. This implies an $O(q^6)$ bound on the number of equivalence classes.

Below, we sketch how equivalence classes can be used to obtain an offline dynamic connectivity data structure for axis-aligned segments.

Preprocessing. At the beginning of a phase, the first task is to find the set of equivalence classes among components of S with respect to Q. This step is nontrivial, but Afshani and Chan showed how to compute these classes, or more precisely, the equivalences classes with respect to the $O(q^2)$ non-intersecting sub-segments, in $\tilde{O}(n)$ time in the case of axis-aligned segments (we will say more about this later).

Throughout the phase, we maintain a *proxy graph* H, such that the connected components of the geometric objects roughly correspond to the connected components of H. The graph H is defined as follows:

- For each equivalence class L, there is a *class vertex* corresponding to L.
- For each inserted segment $s \in Q$, there is an *insertion vertex* corresponding to s.
- The edges of H indicate which pairs of objects intersect.

The graph H has $O(q^6)$ vertices and $O(q^7)$ edges, and can be stored in a dynamic graph connectivity structure supporting edge updates in $\tilde{O}(1)$ time [16, 17].

Offline updates. Updates of objects are done only to Q and not to S. Whenever an object s is inserted to Q, we create a vertex for s in H. For each class L, we decide whether s intersects all components of L by picking an arbitrary component from L, and then query an orthogonal intersection searching structure [12]. Each insertion/deletion in Q causes $\tilde{O}(q^6)$ edge updates in H.

The preprocessing step takes $\tilde{O}(n)$ time, while each update takes $\tilde{O}(q^6)$ time. Choosing $q = n^{1/7}$, we achieve $\tilde{O}(n^{6/7})$ amortized update time.

Query. Given two query objects u, v, there are now two cases to consider.

- If both $u, v \in S$, then let c_u, c_v be the components containing u, v. If $c_u = c_v$, then u and v are connected. If $c_u \neq c_v$ and either c_u or c_v belongs to an equivalence class that corresponds to an isolated vertex in H, then u and v are not connected.
- Otherwise, u and v are connected if and only if their corresponding vertices in H are connected (if $u \in S$, then its corresponding vertex is the class containing c_u).

Thus, queries between two objects can be handled in $O(1)$ time. We can also handle global connectivity queries: the overall number of connected components is equal to the number of components in H (which can be maintained by a dynamic graph connectivity structure [24]) plus the number of components in isolated vertices of H (which is straightforward to maintain).

3 A New Version of the Combinatorial Lemma

It turns out that the $O(q^6)$ combinatorial bound on the number of equivalence classes, when Q may be intersecting, can be improved to $O(q^3)$. There is no need to subdivide the arrangement of Q into $O(q^2)$ non-intersecting parts before applying Lemma 1. Rather, we can modify Afshani and Chan's proof directly. (The proof of the generalized lemma can be found in the full version [8].)

▶ **Lemma 2** (New combinatorial lemma). *Consider a set Q of q regions with simple connected boundaries and a set C of disjoint curves in $\mathbb{R}^2$, where the boundaries of any two regions from Q intersect at most $O(1)$ times. Then C consists of at most $O(q^3)$ equivalence classes with respect to Q. (This bound is tight.)*

Note that the lemma holds also when C is a set of disjoint connected regions (since regions may be "simulated" by strings [22]).

4 Solving the Key Subproblem: Computing Equivalence Classes

As illustrated by Afshani and Chan's method, dynamic connectivity – at least in the offline settings – reduces to the following key subproblem.

▶ **Subproblem 3** (Equivalence class computation). *Given a set S of n geometric objects and a set Q of q geometric objects in $\mathbb{R}^2$, compute the equivalence classes among connected components of S with respect to Q faster than $O(qn)$ time (ideally, in $\tilde{O}(n)$ time when q is not too large).*

For axis-aligned line segments, Afshani and Chan solved Subproblem 3 roughly as follows. We first form a grid by drawing the grid lines at the endpoints of the segments of Q. The grid lines then divide the segments of Q into $O(q^2)$ non-intersecting sub-segments. We will actually compute equivalence classes with respect to these $O(q^2)$ sub-segments instead of Q (this is sufficient for the application to dynamic connectivity). The sub-segments within each row (or column) are linearly ordered. Since any axis-aligned segment u lies in exactly one row or column, the set of sub-segments intersecting u can be represented as an integer interval, which can be computed in $\tilde{O}(1)$ time using binary search. For each connected component c of S, the class that c belongs to can be represented as the union of the intervals for all segments of c. Thus, the representations of all components can be found in $\tilde{O}(n)$ time. Crucially, two components have the same representation if and only if they belong to the same equivalence class. With some careful analysis, Afshani and Chan showed that the representations can be sorted in $\tilde{O}(n)$ time, allowing us to then compute all equivalence classes by a linear scan in $\tilde{O}(n)$ total time.

However, as one can see, the above ad hoc, grid-based approach does not work when the objects are not axis-aligned (it fails even for line segments with three possible slopes).

In this section, we propose a more general approach to solving Subproblem 3 that can handle most types of objects in 2D, including disks and arbitrary line segments. Our approach does not require subdividing Q into nonintersecting pieces first, so we can exploit our new improved combinatorial lemma to get better results even for axis-aligned segments. Furthermore, the *incremental* manner in which we compute the equivalence classes will help in handling online updates more efficiently in our dynamic connectivity data structures.

4.1 New Idea: Repeated Class Splitting

Our central idea is simple: we will compute the equivalence classes incrementally by inserting objects of Q one at a time. (This is fundamentally different from Afshani and Chan's approach, which scans through elements of S instead.) Initially, no object has been inserted, so all components of S belong to the same class. Whenever an object s is inserted, we examine each class L that existed before the insertion. If s intersects some but not all components in L, then L must be split into two "child classes" – one whose components intersect s, and one whose components don't. This way, we can maintain the equivalence classes by repeatedly splitting existing classes as objects are being inserted. (A similar idea was used by Chan [6] to solve a completely different problem.)

Class splitting data structure. To split a class into two child classes, we design a *class splitting* data structure $\mathcal{D}(L)$, such that for any object s that splits L into child classes L_1, L_2, we can efficiently produce data structures $\mathcal{D}(L_1)$ and $\mathcal{D}(L_2)$. An ostensible obstacle is that when forming $\mathcal{D}(L_1)$ and $\mathcal{D}(L_2)$, it seems that we can't avoid spending time at least $O(1)$ time per object in L. If so, computing the classes may take $\Omega(n)$ time per insertion.

To get around this obstacle, we use an amortization trick, namely, a reverse version of the standard "weighted union heuristic": instead of building both $\mathcal{D}(L_1)$ and $\mathcal{D}(L_2)$ from scratch, we do so only for the child class with smaller total size. Specifically, if L_2 has smaller total size than L_1, then we build $\mathcal{D}(L_2)$ from scratch, while $\mathcal{D}(L_1)$ is obtained by deleting everything associated with L_2 from $\mathcal{D}(L)$. Intuitively, we "displace" L_2 from L without necessarily examining objects from L_1. We claim that the sum of the number of displaced objects over all updates in the phase is $O(n \log n)$. One proof is via a potential function argument: defining $\Phi = \sum_{\text{class } L} |L| \log |L|$ (where $|L| := \sum_{\text{component } c \in L} |c|$), it is not difficult to see that a split with y displacements decreases Φ by at least $\Omega(y)$ (because of the inequality $(x + y)\log(x + y) - x \log x - y \log y \geq y \log(x/y + 1) \geq y$ when $x \geq y$), and the claim follows. (An alternative, simple proof is to argue that each object can be displaced at most $O(\log n)$ times, since each time this happens, the size of the class containing the object is reduced by at least a factor of 2.)

Reduction to component (non-)intersection reporting. The key part of the class splitting operation is building a data structure $\mathcal{D}(L)$ that can report the smaller child class L' in time linear in the total size of L', but sublinear in the total size of L. It suffices for the data structure to handle the following two types of queries:

1. Report all connected components in L that intersect s.
2. Report all connected components in L that *do not* intersect s.

By running the two query-answering algorithms concurrently and stopping when either of them terminates, we can report L' within the desired time bound. We need the data structure to support deletions of components, so that we can subsequently delete each component in L' from L.

The component intersection reporting problem we face can be viewed as a *colored* intersection reporting problem, where the objective is to preprocess a set of colored objects, so that one can efficiently report all colors intersected by a query object. Similarly, we need to solve the corresponding colored non-intersection reporting problem. (See [14] for background on colored range searching.)

In the next three subsections, we apply known geometric data structuring techniques (some of which are inspired by colored range searching), to solve this component (non-)intersection reporting problem for axis-aligned line segments, disks, and arbitrary line segments.

4.2 Axis-Aligned Line Segments

▶ **Lemma 4.** *Given a set of connected components formed by n axis-aligned segments in $\mathbb{R}^2$, there is a component (non-)intersection reporting data structure with $\tilde{O}(n)$ preprocessing time and $\tilde{O}(1+k)$ query time, where k is the number of components reported. Furthermore, we can support deletion of any component in $\tilde{O}(1)$ amortized time, and insertion of any component c in $\tilde{O}(|c|)$ amortized time.*

Proof. In the following, we assume that the inserted segment is vertical; the other case can be handled symmetrically. For each component c, we compute the vertical decomposition T_c of the horizontal segments of c. There are $O(n)$ rectangular cells over all components. We make the following observation.

▶ **Observation 5.** *Let s be a vertical segment and p be its lower endpoint. For any component c, let Δ be the cell in T_c that contains p, and let u be the horizontal segment in c that bounds Δ from above. Then s intersects a segment in c if and only if s intersects u.*

By Observation 5, components that intersect s can be reported as follows:
1. Among the cells of T_c for all components c, find all cells that contain p.
2. Report, among the upper-bounding segments of these cells, the ones that intersect s.

Components that do not intersect s can be reported similarly, except in the second step we instead report the segments that do not intersect s. We support the above reporting query with a two-level data structure $\mathcal{D}$.
- The primary structure is an orthogonal rectangle stabbing structure with $\tilde{O}(1)$ query time and $\tilde{O}(1)$ update time (e.g., [12], or by reduction to orthogonal range searching in $\mathbb{R}^4$).
- For each canonical subset B in the primary structure, we associate each cell $\Delta \in B$ with the segment that bounds Δ from above. We keep two auxiliary structures on the associated line segments. Given a vertical segment s, one auxiliary structure reports all segments intersecting s, while the other reports all segments not intersecting s. Both structures have $\tilde{O}(1+k)$ query time (where k is the output size) and $\tilde{O}(1)$ update time (as these subproblems also reduce to orthogonal range searching).

The preprocessing and query time bounds are clearly satisfied. Insertion/deletion of a component c requires inserting/deleting all cells of T_c and takes $\tilde{O}(|c|)$ time; we can charge the cost of deletion to preprocessing or insertion. ◀

▶ **Corollary 6.** *For n axis-aligned line segments in $\mathbb{R}^2$, we can solve Subproblem 3 in $\tilde{O}(n+q^4)$ time. Furthermore, each insertion to Q takes $\tilde{O}(q^3+n/q)$ amortized time. We can also support deletion of a component in $\tilde{O}(1)$ amortized time and insertion of a component c in $\tilde{O}(|c|)$ amortized time, if we are given c's class.*

Proof. We store each equivalence class in the data structure from the preceding lemma. For $i=1,\ldots,q$, let $L_{i,1},\ldots,L_{i,O(q^3)}$ be the existing classes before the i-th insertion. Let $n_{i,1},\ldots,n_{i,O(q^3)}$ be the sizes of the classes. Whenever a class $L_{i,j}$ is split, let $m_{i,j}$ be the total size of the displaced child class. As noted, the total number of displacements is $\sum_{i,j} m_{i,j} = \tilde{O}(n)$, when there are no insertions of components; in general, it is $\tilde{O}(n+\Sigma)$, where Σ is the sum of the sizes of the components inserted (because insertion of a component c increases the potential Φ by $\tilde{O}(|c|)$). Thus, the total running time for class splitting operations over all insertions to Q is at most

$$\tilde{O}\left(\sum_{i=1}^{q}\sum_{j=1}^{O(q^3)}(1+m_{i,j})\right) = \tilde{O}\left(q^4+n+\Sigma\right).$$

This amortizes to $\tilde{O}(q^3 + n/q)$ time per insertion to Q, with the Σ term charged to insertions of components. ◀

4.3 Disks

▶ **Lemma 7.** *Given a set of connected components formed by n disks in $\mathbb{R}^2$, there is a component (non-)intersection reporting data structure with $O^*(n)$ preprocessing time and $O^*(n^{4/5} + k)$ query time, where k is the number of components reported. Furthermore, we can support deletion of a component in $O^*(1)$ amortized time and insertion of a component c in $O^*(|c|)$ amortized time.*

The proof follows the same argument as for axis-aligned segments, except we use different data structures than orthogonal range searching for the primary and auxiliary structures. The complete proof can be found in the full version [8].

▶ **Corollary 8.** *For n disks in $\mathbb{R}^2$, we can solve Subproblem 3 in $O^*(n + q^{8/5}n^{4/5})$ time. Furthermore, each insertion to Q takes amortized $O^*(q^{3/5}n^{4/5} + n/q)$ time. We can also support deletion of a component in amortized $O^*(1)$ time and insertion of a component c in amortized $O^*(|c|)$ time, if we are given c's class.*

Proof. We define $n_{i,j}$ and $m_{i,j}$ for $i = 1, \ldots, q$ and $j = 1, \ldots, O(q^3)$, as well as Σ, in the same way as we did for axis-aligned segments. Using the same analysis, the total running time for class splitting operations over all insertions to Q is at most

$$O^*\left(\sum_{i=1}^{q}\sum_{j=1}^{O(q^3)}\left(n_{i,j}^{4/5} + m_{i,j}\right)\right),$$

which, by Hölder's inequality, is bounded by $O^*(q \cdot (q^{3/5}n^{4/5}) + n + \Sigma)$. This amortizes to $O^*(q^{3/5}n^{4/5} + n/q)$ time per insertion to Q. ◀

4.4 Arbitrary Line Segments

The component intersection reporting problem becomes tougher for the case of arbitrary line segments. Although we are unable to get near-linear preprocessing time, we can obtain the following preprocessing/query-time trade-off. The idea uses *cuttings* and is inspired by a known idea for a different colored range searching problem (namely, "range mode", i.e., finding the most frequent color in a range) [7].

▶ **Lemma 9.** *Given a set of connected components formed by n line segments in $\mathbb{R}^2$ and a parameter r, there is a component (non-)intersection reporting data structure with $O^*(nr^4)$ preprocessing time and $O^*(1 + n/r + k)$ query time, where k is the number of components reported. Furthermore, we can support deletion of a component in $O^*(r^4)$ amortized time and insertion of a component c in $O^*(|c|r^4)$ amortized time.*

Proof. A line segment in $\mathbb{R}^2$ can be represented by 4 real values (the coordinates of its two endpoints) and so can be viewed as a point in $\mathbb{R}^4$. For each input line segment s, the set of all line segments intersecting s then corresponds a semialgebraic set in $\mathbb{R}^4$ with constant description complexity. Let S be the collection of these n semialgebraic sets.

Now, compute a $(1/r)$-*cutting* Γ of S, consisting of $O^*(r^4)$ cells (due to known combinatorial bounds on vertical decompositions) [21], in $O^*(nr^4)$ time.

For each cell $\Delta \in \Gamma$, we store the *conflict list* of Δ, consisting of all semialgebraic sets $s \in S$ with boundaries crossing Δ; by definition of cuttings, each conflict list has size $O(n/r)$. For each component c with no semialgebraic set in the conflict list of Δ, we store c in a list $A(\Delta)$ if at least one semialgebraic set from c completely contains Δ, or store c in another list $B(\Delta)$ otherwise. To allow for efficient update, for each component c, we keep track of all semialgebraic sets from c that contains Δ.

For each semialgebraic set $s \in S$ and each cell $\Delta \in \Gamma$, we can decide in $O(1)$ time which list s should be stored in. Thus, the data structure has $O^*(nr^4)$ preprocessing time. For the same reason, deletion of a semialgebraic set can be handled in $O^*(r^4)$ time.

To handle a query for a line segment represented as a point $p \in \mathbb{R}^4$, we first locate the cell $\Delta \in \Gamma$ containing p. To report all components stabbed (resp. not stabbed) by p, we linearly search the conflict list, and then report the components stored in the list $A(\Delta)$ (resp. $B(\Delta)$). The reporting takes $O^*(1 + n/r + k)$ time, where k is the output size.

Insertions of components can be handled by the logarithmic method [3]. ◀

▶ **Corollary 10.** *For n line segments in $\mathbb{R}^2$ and a parameter r, we can solve Subproblem 3 in $O^*(r^4 n + qn/r + q^4)$ time. Furthermore, each insertion to Q takes amortized $O^*(n/r + q^3 + r^4 n/q)$ time. We can also support deletion of a component in $O^*(r^4)$ time and insertion of a component c in amortized $O^*(|c|r^4)$ time, if we are given c's class.*

Proof. We define $n_{i,j}$ and $m_{i,j}$ for $i = 1, \ldots, q$ and $j = 1, \ldots, O(q^3)$, as well as Σ, in the same way as we did for axis-aligned segments and disks. Using the same analysis, the total running time for class splitting over all insertions to Q is at most

$$O^*\left(\sum_{i=1}^{q} \sum_{j=1}^{O(q^3)} \left(1 + \frac{n_{i,j}}{r} + m_{i,j} r^4\right)\right) = O^*\left(\frac{qn}{r} + q^4 + r^4(n + \Sigma)\right),$$

This amortizes to $O^*(n/r + q^3 + r^4 n/q)$ per insertion to Q. ◀

5 Dynamic Connectivity

Now, we apply the equivalence-class data structures developed in Section 4 to design dynamic connectivity structures for axis-aligned line segments, disks, and arbitrary line segments in $\mathbb{R}^2$. We follow the basic approach from Section 2 but describe how to handle general online updates.

5.1 Axis-Aligned Line Segments

For axis-aligned segments, we maintain the following data structures during a phase:

- The equivalence-class data structure.
- A decremental data structure for explicitly maintaining the connected components of S, along with their sizes. This structure has $\tilde{O}(n)$ preprocessing time and $\tilde{O}(n)$ total update time over the entire phase. We achieve this by reducing to decremental graph connectivity under edge deletions [26] (using the biclique cover technique from Chan [5]).
- For each component c, an orthogonal intersection searching structure with $\tilde{O}(1)$ query and update time (e.g., [12]).

Throughout the phase, we maintain a proxy graph H in the same way as in Afshani and Chan's method, as described in Section 2. Because updates are online, we need to handle not only insertions and deletions in Q but also deletions in S.

Insertion/deletion in Q. Insertions in Q can be done as before, since the equivalence-class structure already supports insertions to Q. By Corollary 6, the amortized cost of equivalence class maintenance per insertion to Q is $\tilde{O}(q^3 + n/q)$. The number of edge changes in the proxy graph H is $O(q^4)$; we can afford to reconstruct the connectivity structure of H in $O(q^4)$ time.

For deletions in Q, we don't even need to update the equivalence-class data structure.

Deletion in S. Suppose that a segment $s \in S$ is deleted in the i-th update. Let c be the component that contains s. The deletion of s may divide c into smaller sub-components $c_1, \ldots, c_z$, listed in order of decreasing size. Let $m_i = |c_2| + \cdots + |c_z|$. Since $|c_2|, \ldots, |c_z| \le |c|/2$, the sum $\sum_i m_i$ over the entire phase is $\tilde{O}(n)$.

First, we update the decremental connectivity structure; this takes $\tilde{O}(n)$ time over the entire phase. We then report the disks of $c_2, \ldots, c_z$, delete them from the orthogonal intersection searching structure for c, and then rebuild the structure for each of $c_2, \ldots, c_z$. This takes $\tilde{O}(m_i)$ time. We make c_1 a singleton class, if it isn't already. We delete c_1 from the equivalence-class structure.[4] We create at most q singleton classes this way, which is negligible.

Next, we compute the classes among $c_2, \ldots, c_z$ with respect to Q by "replaying" the insertions in Q from scratch. This takes $\tilde{O}(q^4 + m_i)$ time. For each class L' found this way, we find the existing class L that is equivalent to L' (this takes $\tilde{O}(q^4)$ total time, by viewing each class as a q-bit vector and sorting $O(q^3)$ such vectors). If such a class L exists, then we insert the components in L' to the equivalence-class structure. This takes $\tilde{O}(m_i)$ amortized time. Thus, the total time for handling deletions is bounded by $\tilde{O}(q^5 + n)$, which amortizes to $\tilde{O}(q^4 + n/q)$. This is the overall amortized update time. Choosing $q = n^{1/5}$, we achieve update time $\tilde{O}(n^{4/5})$.

▶ **Theorem 11.** *For n axis-aligned line segments in $\mathbb{R}^2$, there is a dynamic connectivity data structure with $O(1)$ query time and $\tilde{O}(n^{4/5})$ amortized update time.*

5.2 Disks

Using similar ideas, we achieve the following result for disks (the complete proof can be found in the full version [8]).

▶ **Theorem 12.** *For n disks in $\mathbb{R}^2$, there is a dynamic connectivity data structure with $O(1)$ query time and $O^*(n^{8/9})$ amortized update time.*

5.3 Arbitrary Line Segments

For the case of arbitrary line segments the idea is again similar to the case of axis-aligned segments, but because the time bound for Subproblem 3 is superlinear and dependent on the parameter r even when q is small, the settings of parameters are more delicate. There is also a new issue: we don't have a decremental connectivity structure for arbitrary line segments with both near-linear preprocessing time and near-linear total update time (the best static algorithm requires $O(n^{4/3})$ time [10]).

[4] All we want is that if two components are in the same class, they intersect the same elements of Q; we don't need the converse. This explains why it is fine to handle some classes such as c_1 separately, and why we can ignore deletions in Q in the equivalence-class structure.

Fortunately, we observe that in the preprocessing step between every two update phases, no more than q new segments are added to the decremental connectivity structure. Thus, the decremental connectivity structure does not necessarily need $O^*(n)$ preprocessing time. Instead, it only needs to support *batch insertion* of at most q segments in near-linear time. Such a data structure has been given by Chan, Pătraşcu, and Roditty [9].

▶ **Lemma 13** ([9]). *Let S be a set of n line segments in $\mathbb{R}^2$. There is a decremental connectivity data structure with $\tilde{O}(n)$ total deletion time over any sequence of deletions, and supports batch insertion of any q line segments in $\tilde{O}(n + q\sqrt{n})$ time.*

Since we will choose $q \ll \sqrt{n}$, the $\tilde{O}(q\sqrt{n})$ term is negligible.

Update time. The update algorithm is again identical to the case of axis-aligned segments. By Corollary 10, the amortized cost of equivalence-class maintenance per insertion to Q is $\tilde{O}(n/r + q^3 + r^4 n/q)$. For deletions in S, define m_i for $i = 1, \dots, q$ in the same way as we did for disks. Then the total cost over all deletions is bounded by

$$O^*\left(\sum_{i=1}^{q}\left(\frac{qm_i}{r} + q^4 + r^4 m_j\right) + n\right) \;=\; O^*\left(\frac{qn}{r} + q^5 + r^4 n\right),$$

which amortizes to $O^*(n/r + q^4 + r^4 n/q)$ per deletion. This is the dominating term in the overall amortized update time. Choosing $r = n^{1/21}$ and $q = n^{5/21}$, the amortized update time minimizes to $O^*(n^{20/21})$.

▶ **Theorem 14.** *For n arbitrary line segments in $\mathbb{R}^2$, there is a dynamic connectivity data structure with $O(1)$ query time and $O^*(n^{20/21})$ amortized update time.*

We remark that the same approach also works more generally for fixed-degree algebraic curves in 2D, with a larger exponent of the update bound depending on the degree (as the dimension of the lifted space gets larger in the proof of Lemma 9).

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. 55th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 434–443, 2014. doi:10.1109/FOCS.2014.53.

2 Peyman Afshani and Timothy M. Chan. Dynamic connectivity for axis-parallel rectangles. *Algorithmica*, 53(4):474–487, 2009. Preliminary version in ESA'06. doi:10.1007/s00453-008-9234-7.

3 Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980. doi:10.1016/0196-6774(80)90015-2.

4 Timothy M. Chan. Semi-online maintenance of geometric optima and measures. *SIAM J. Comput.*, 32(3):700–716, 2003. doi:10.1137/S0097539702404389.

5 Timothy M. Chan. Dynamic subgraph connectivity with geometric applications. *SIAM J. Comput.*, 36(3):681–694, 2006. Preliminary version in STOC'02. doi:10.1137/S009753970343912X.

6 Timothy M. Chan. Near-optimal randomized algorithms for selection in totally monotone matrices. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1483–1495, 2021. doi:10.1137/1.9781611976465.89.

7 Timothy M. Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T. Wilkinson. Linear-space data structures for range mode query in arrays. *Theory Comput. Syst.*, 55(4):719–741, 2014. doi:10.1007/S00224-013-9455-2.

8 Timothy M. Chan and Zhengcheng Huang. Dynamic geometric connectivity in the plane with constant query time. *CoRR*, abs/2402.05357, 2024. doi:10.48550/arXiv.2402.05357.

9 Timothy M. Chan, Mihai Pătraşcu, and Liam Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM J. Comput.*, 40(2):333–349, 2011. Preliminary version in FOCS'08. `doi:10.1137/090751670`.

10 Timothy M. Chan and Da Wei Zheng. Hopcroft's problem, log-star shaving, 2D fractional cascading, and decision trees. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 190–210, 2022. `doi:10.1137/1.9781611977073.10`.

11 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *Proc. 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167, 2020. `doi:10.1109/FOCS46700.2020.00111`.

12 Herbert Edelsbrunner and Hermann A. Maurer. On the intersection of orthogonal objects. *Inf. Process. Lett.*, 13(4/5):177–181, 1981. `doi:10.1016/0020-0190(81)90053-3`.

13 Daniele Frigioni and Giuseppe F. Italiano. Dynamically switching vertices in planar graphs. *Algorithmica*, 28(1):76–103, 2000. `doi:10.1007/S004530010032`.

14 Prosenjit Gupta, Ravi Janardan, Saladi Rahul, and Michiel Smid. Computational geometry: Generalized (or colored) intersection searching. In *Handbook of Data Structures and Applications*, pages 1043–1058. Chapman and Hall/CRC, 2018. URL: `https://www.csa.iisc.ac.in/~saladi/Papers/ds2-handbook.pdf`.

15 Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999. `doi:10.1145/320211.320215`.

16 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. `doi:10.1145/502090.502095`.

17 Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Fully dynamic connectivity in $O(\log n(\log\log n)^2)$ amortized expected time. *TheoretiCS*, 2, 2023. `doi:10.46298/THEORETICS.23.6`.

18 Ce Jin and Yinzhan Xu. Tight dynamic problem lower bounds from generalized BMM and OMv. In *Proc. 54th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1515–1528, 2022. `doi:10.1145/3519935.3520036`.

19 Haim Kaplan, Alexander Kauer, Katharina Klost, Kristin Knorr, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Dynamic connectivity in disk graphs. In *Proc. 38th International Symposium on Computational Geometry (SoCG)*, pages 49:1–49:17, 2022. `doi:10.4230/LIPICS.SOCG.2022.49`.

20 Haim Kaplan, Katharina Klost, Kristin Knorr, Wolfgang Mulzer, and Liam Roditty. Insertion-only dynamic connectivity in general disk graphs. In *Proc. SIAM Symposium on Simplicity of Algorithms (SOSA)*, pages 299–305, 2024. `doi:10.1137/1.9781611977936.27`.

21 Vladlen Koltun. Almost tight upper bounds for vertical decompositions in four dimensions. *J. ACM*, 51(5):699–730, 2004. Preliminary verion in FOCS'01. `doi:10.1145/1017460.1017461`.

22 James R. Lee. Separators in region intersection graphs. In *Proc. 8th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 1:1–1:8, 2017. `doi:10.4230/LIPICS.ITCS.2017.1`.

23 Jirí Matoušek. *Lectures on Discrete Geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer, 2002.

24 Mihai Pătraşcu and Mikkel Thorup. Planning for fast connectivity updates. In *Proc. 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 263–271, 2007. `doi:10.1109/FOCS.2007.54`.

25 Warren D. Smith and Nicholas C. Wormald. Geometric separator theorems & applications. In *Proc. 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 232–243, 1998. `doi:10.1109/SFCS.1998.743449`.

26 Mikkel Thorup. Decremental dynamic connectivity. *J. Algorithms*, 33(2):229–243, 1999. Preliminary version in SODA'97. `doi:10.1006/JAGM.1999.1033`.