Dynamic Connectivity: Connecting to Networks and Geometry

Timothy M. Chan
University of Waterloo
tmchan@uwaterloo.ca

Mihai Pătraşcu
MIT
mip@mit.edu

Liam Roditty
Weizmann Institute
liam.roditty@weizmann.ac.il

March 5, 2009

Abstract

Dynamic connectivity is a well-studied problem, but so far the most compelling progress has been confined to the edge-update model: maintain an understanding of connectivity in an undirected graph, subject to edge insertions and deletions. In this paper, we study two more challenging, yet equally fundamental problems:

Subgraph connectivity asks to maintain an understanding of connectivity under vertex updates: updates can turn vertices on and off, and queries refer to the subgraph induced by *on* vertices. (For instance, this is closer to applications in networks of routers, where node faults may occur.)

We describe a data structure supporting vertex updates in $\widetilde{O}(m^{2/3})$ amortized time, where m denotes the number of edges in the graph. This greatly improves over the previous result [Chan, STOC'02], which required fast matrix multiplication and had an update time of $O(m^{0.94})$. The new data structure is also simpler.

Geometric connectivity asks to maintain a dynamic set of n geometric objects, and query connectivity in their intersection graph. (For instance, the intersection graph of balls describes connectivity in a network of sensors with bounded transmission radius.)

Previously, nontrivial fully dynamic results were known only for special cases like axis-parallel line segments and rectangles. We provide similarly improved update times, $\widetilde{O}(n^{2/3})$, for these special cases. Moreover, we show how to obtain sublinear update bounds for virtually *all* families of geometric objects which allow sublinear-time range queries. In particular, we obtain the *first* sublinear update time for arbitrary 2D line segments: $O^*(n^{9/10})$; for d-dimensional simplices: $O^*(n^{1-\frac{1}{d(2d+1)}})$; and for d-dimensional balls: $O^*(n^{1-\frac{1}{(d+1)(2d+3)}})$.

1 Introduction

1.1 Dynamic Graphs

Dynamic graphs inspire a natural, challenging, and well-studied class of algorithmic problems. A rich body of the STOC/FOCS literature has considered problems ranging from the basic question of understanding connectivity in a dynamic graph [13, 17, 34, 6, 31], to maintaining the minimum spanning tree [20], the min-cut [36], shortest paths [9, 35], reachability in directed graphs [10, 25, 26, 32, 33], etc.

But what exactly makes a graph "dynamic"? Computer networks have long provided the common motivation. The dynamic nature of such networks is captured by two basic types of updates to the graph:

- edge updates: adding or removing an edge. These correspond to setting up a new cable connection, accidental cable cuts, etc.

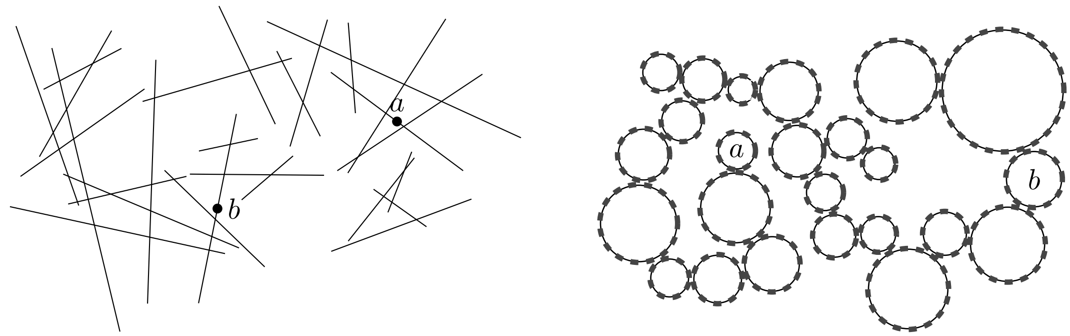
- vertex updates: turning a vertex on and off. Vertices (routers) can temporarily become "off" after events such as a misconfiguration, a software crash and reboot, etc. Problems involving only vertex updates have been called *dynamic subgraph* problems, since queries refer to the subgraph induced by vertices which are on.

Loosely speaking, dynamic graph problems fall into two categories. For "hard" problems, such as shortest paths and directed reachability, the best known running times are at least linear in the number of vertices. These high running times obscure the difference between vertex and edge updates, and identical bounds are often stated [9, 32, 33] for both operations. For the remainder of the problems, sublinear running times are known for edge updates, but sublinear bounds for vertex updates seems much harder to get. For instance, even iterating through all edges incident to a vertex may take linear time in the worst case. That vertex updates are slow is unfortunate. Referring to the computer-network metaphor, vertex updates are cheap "soft" events (misconfiguration or reboot), which occur more frequently than the costly physical events (cable cut) that cause an edge update.

Subgraph connectivity. As mentioned, most previous sublinear dynamic graph algorithms address edge updates but not the equally fundamental vertex updates. One notable exception, however, was a result of Chan [6] from STOC'02 on the basic connectivity problem for general sparse (undirected) graphs. This algorithm can support vertex updates in time[1] $O(m^{0.94})$ and decide whether two query vertices are connected in time $\widetilde{O}(m^{1/3})$.

Though an encouraging start, the nature of this result makes it appear more like a half breakthrough. For one, the update time is only slightly sublinear. Worse yet, Chan's algorithm requires fast matrix multiplication (FMM). The $O(m^{0.94})$ update time follows from the theoretical FMM algorithm of Coppersmith and Winograd [8]. If Strassen's algorithm is used instead, the update time becomes $O(m^{0.984})$. Even if optimistically FMM could be done in quadratic time, the update time would only improve to $O(m^{0.89})$. FMM has been used before in various dynamic graph algorithms (e.g., [10, 26]), and the paper [6] noted specific connections to some matrix-multiplication-related problems (see Section 2). All this naturally led one to suspect, as conjectured in the paper, that FMM might be essential to our problem. Thus, the result we are about to describe may come as a bit of a surprise…

[1]We use m and n to denote the number of edges and vertices of the graph respectively; $\widetilde{O}(\cdot)$ ignores polylogarithmic factors and $O^*(\cdot)$ hides n^ε factors for an arbitrarily small constant $\varepsilon > 0$. Update bounds in this paper are, by default, amortized.



(a) **(b)**

Figure 1: (a) Is b reachable from a staying on the roads? (b) Do the gears transmit rotation from a to b?

Our result. In this paper, we present a new algorithm for dynamic connectivity, achieving an improved vertex-update time of $\widetilde{O}(m^{2/3})$, with an identical query time of $\widetilde{O}(m^{1/3})$. First of all, this is a significant *quantitative* improvement (to anyone who regards an $m^{0.27}$ factor as substantial), and it represents the first convincingly sublinear running time. More importantly, it is a significant *qualitative* improvement, as our bound does not require FMM. Our algorithm involves a number of ideas, some of which can be traced back to earlier algorithms, but we use known edge-updatable connectivity structures to maintain a more cleverly designed intermediate graph. The end product is not straightforward at all, but still turns out to be *simpler* than the previous method [6] and has a compact, two-page description (we regard this as another plus, not a drawback).

1.2 Dynamic Geometry

We next turn to another important class of dynamic connectivity problems—those arising from geometry.

Geometric connectivity. Consider the following question, illustrated in Figure 1(a). Maintain a set of line segments in the plane, under insertions and deletions, to answer queries of the form: "given two points a and b, is there a path between a and b along the segments?"

This simple-sounding problem turns out to be a challenge. On one hand, understanding any local geometry does not seem to help, because the connecting path can be long and windy. On the other hand, the graph-theoretic understanding is based on the intersection graph, which is too expensive to maintain. A newly inserted (or deleted) segment can intersect a large number of objects in the set, changing the intersection graph dramatically.

Abstracting away, we can consider a broad class of problems of the form: maintain a set of n geometric objects, and answer connectivity queries in their intersection graph. Such graphs arise, for instance, in VLSI applications in the case of orthogonal segments, or gear transmission systems, in the case of touching disks; see Figure 1(b). A more compelling application can be found in sensor networks: if r is the radius within which two sensors can communicate, the communication network is the intersection graph of balls of radius $r/2$ centered at the sensors. While our focus is on theoretical understanding rather than the practicality of specific applications, these examples still indicate the natural appeal of geometric connectivity problems.

All these problems have a trivial $\widetilde{O}(n)$ solution, by maintaining the intersection graph through edge updates. A systematic approach to beating the linear time bound was proposed in Chan's paper as well [6], by drawing a connection to subgraph connectivity. Assume that a particular object type allows data structures for intersection range searching with space $S(n)$ and query time $T(n)$. It was shown that geometric connectivity can essentially be solved by maintaining a graph of size $m = O(S(n) + nT(n))$ and running $O(S(n)/n + T(n))$ vertex updates for every object insertion or deletion. Using the previous subgraph connectivity result [6], an update in the geometric connectivity problem took time $\widetilde{O}([S(n)/n+T(n)]\cdot[S(n)+nT(n)]^{0.94})$. Using our improved result, the bound becomes $\widetilde{O}([S(n)/n + T(n)] \cdot [S(n) + nT(n)]^{2/3})$.

The prime implication in the previous paper is that connectivity of axis-parallel boxes in any constant dimension (in particular, orthogonal line segments in the plane) reduces to subgraph connectivity, with a polylogarithmic cost. Indeed, for such boxes range trees yield $S(n) = n \cdot \lg^{O(d)} n$ and $T(n) = \lg^{O(d)} n$. Unfortunately, while nontrivial range searching results are known for many types of objects, very efficient range searching is hard to come by. Consider our main motivating examples:

- for arbitrary (non-orthogonal) line segments in $\mathbb{R}^2$, one can achieve $T(n) = O^*(\sqrt{n})$ and $S(n) = O^*(n)$, or $T(n) = O^*(n^{1/3})$ and $S(n) = O^*(n^{4/3})$ [28].
- for disks in $\mathbb{R}^2$, one can achieve $T(n) = O^*(n^{2/3})$ and $S(n) = O^*(n)$, or $T(n) = O^*(n^{1/2})$ and $S(n) = O^*(n^{3/2})$ [3].

Even with our improved vertex-update time, the $[S(n)/n + T(n)] \cdot [S(n) + nT(n)]^{2/3}$ bound is too weak to beat the trivial linear update time. For arbitrary line segments in $\mathbb{R}^2$, one would need to improve the vertex-update time to $m^{1/2-\varepsilon}$, which appears unlikely without FMM (see Section 2). The line segment case was in fact mentioned as a major open problem, implicitly in [6] and explicitly in [1]. The situation gets worse for objects of higher complexity or in higher dimensions.

Our results. In this paper, we are finally able to break the above barrier for dynamic geometric connectivity. At a high level, we show that range searching with *any* sublinear query time is enough to obtain sublinear update time in geometric connectivity. In particular, we get the *first* nontrivial update times for arbitrary line segments in the plane, disks of arbitrary radii, and simplices and balls in any fixed dimension. While the previous reduction [6] involves merely a straightforward usage of "biclique covers", our result here requires much more work. For starters, we need to devise a "degree-sensitive" version of our improved subgraph connectivity algorithm (which is of interest in itself); we then use this and known connectivity structures to maintain not one but two carefully designed intermediate graphs.

Essentially, if $T(n) = \widetilde{O}(n^{1-b})$ and $S(n) = \widetilde{O}(n)$, we can support dynamic geometric connectivity with update time $\widetilde{O}\big(n^{1-b^2/(2+b)}\big)$ and query time $\widetilde{O}\big(n^{b/(2+b)}\big)$. For non-orthogonal line segments in $\mathbb{R}^2$, this gives an update time of $O^*(n^{9/10})$ and a query time of $O^*(n^{1/5})$. For disks in $\mathbb{R}^2$, the update time is $O^*(n^{20/21})$, with a query time of $O^*(n^{1/7})$.

Known range searching techniques [2] from computational geometry almost always provide sublinear query time. For instance, Matoušek [28] showed that $b \approx 1/2$ is attainable for line segments, triangles, and any constant-size polygons in $\mathbb{R}^2$; more generally, $b \approx 1/d$ for simplices or constant-size polyhedra in $\mathbb{R}^d$. Further results by Agarwal and Matoušek [3] yield $b \approx 1/(d + 1)$ for balls in $\mathbb{R}^d$. Most generally, $b > 0$ is possible for *any* class of objects defined by semialgebraic sets of constant description complexity.

More results. Our general sublinear results undoubtedly invite further research into finding better bounds for specific classes of objects. In general, the complexity of range queries provides a natural barrier for the update time, since upon inserting an object we at least need to determine if it intersects any object already in the set. Essentially, our result has a quadratic loss compared to range queries: if $T(n) = n^{1-b}$, the update time is $n^{1-\Theta(b^2)}$.

In Section 5, We make a positive step towards closing this quadratic gap: we show that if the updates are given *offline* (i.e. are known in advance), the amortized update time can be made $n^{1-\Theta(b)}$. We need FMM this time, but the usage of FMM here is more intricate (and interesting) than typical. For one, it is crucial to use fast *rectangular* matrix multiplication. Along the way, we even find ourselves rederiving Yuster and Zwick's sparse matrix multiplication result [38] in a more general form. The juggling of parameters is also more unusual, as one can suspect from looking at our actual update bound, which is $\widetilde{O}(n^{\frac{1+\alpha-b\alpha}{1+\alpha-b\alpha/2}})$, where $\alpha = 0.294$ is an exponent associated with rectangular FMM.

2 Related Work

Before proceeding to our new algorithms, we mention more related work, for the sake of completeness.

Graphs. Most previous work on dynamic subgraph connectivity concerns special cases only. Frigioni and Italiano [14] considered vertex updates in planar graphs, and described a polylogarithmic solution.

If vertices have constant degree, vertex updates are equivalent to edge updates. For edge updates, Henzinger and King [17] were first to obtain polylogarithmic update times (randomized). This was improved by Holm et al. [20] to a deterministic solution with $O(\lg^2 m)$ time per update, and by Thorup [34] to a randomized solution with $O(\lg m \cdot (\lg\lg m)^3)$ update time. The randomized bound almost matches the $\Omega(\lg m)$ lower bound from [30]. All these data structures maintain a spanning forest as a certificate for connectivity. This idea fails for vertex updates in the general case, since the certificate can change substantially after just one update.

In many practical settings, these planar-graph and constant-degree special cases are unfortunately inadequate. In particular, large networks of routers are often designed as overlay graphs over a (small-degree) geographic graph. Long fiber-optic links bypass intermediate nodes, in order to minimize the latency cost of passing through the electric domain repeatedly.

For more difficult dynamic graph problems, the goal is typically changed from getting polylogarithmic bounds to finding better exponents in polynomial bounds; for example, see all the papers on directed reachability [10, 25, 32, 33]. Evidence suggests that dynamic subgraph connectivity fits this category. It was observed [6] that finding triangles (3-cycles) or quadrilaterals (4-cycles) in directed graphs can be reduced to $O(m)$ vertex updates. Thus, an update bound better than $\sqrt{m}$ appears unlikely without FMM, since the best running time for finding triangles without FMM is $O(m^{3/2})$, dating back to STOC'77 [24]. Even with FMM, known results are only slightly better: finding triangles and quadrilaterals takes time $O(m^{1.41})$ [5] and $O(m^{1.48})$ [37] respectively. Thus, current knowledge prevents an update bound better than $m^{0.48}$.

Geometry. It was shown [6] that subgraph connectivity can be reduced to dynamic connectivity of axis-parallel line segments in 3 dimensions. Thus, as soon as one gets enough combinatorial richness in the host geometric space, subgraph connectivity becomes the *only* possible way to solve geometric connectivity.

When the geometry is less combinatorially rich, it is possible to find *ad hoc* algorithms that do not rely on subgraph connectivity. Special cases that have been investigated include the following:

- for orthogonal segments or axis-parallel rectangles in the plane, Afshani and Chan [1] proposed a data structure with update time $\widetilde{O}(n^{10/11})$ and constant query time. This is incomparable to our result of update time $\widetilde{O}(n^{2/3})$ and query time $\widetilde{O}(n^{1/3})$.

- for unit axis-parallel hypercubes, the problem reduces to maintaining the minimum spanning tree under the ℓ_∞ metric. Eppstein [11] describes a general technique for dynamic geometric MST, ultimately appealing to range searching, and obtains polylogarithmic time per operation.

- for unit balls, the problem reduces to dynamic Euclidean MST, which in turn reduces to range searching by Eppstein's technique [11]. In two dimensions, Chan's dynamic nearest-neighbor data structure [7] implies an $O(\lg^{10} n)$ update time for this problem.

Dynamic geometric connectivity is a natural continuation of static geometric connectivity problems, which have been studied since the early 1980s. As in our case, the main challenge is to avoid working explicitly with the intersection graph, which could be of quadratic size. Known results include $O(n \lg n)$-time algorithms [22, 23] for computing the connected components of axis-aligned rectangles in the plane, and $\widetilde{O}(n^{4/3})$-time algorithms [16, 27] for arbitrary line segments in the plane. More generally, Chan [6] (and later Eppstein [12]) noted the connection of static geometric connectivity to range searching, which implied subquadratic algorithms for objects with constant description complexity. The connection carries over to the incremental (insertion-only) and decremental (deletion-only) cases [6], e.g., yielding $\widetilde{O}(n^{1/3})$ update time for arbitrary line segments, reproving and extending some older results [4].

Another related problem is maintaining connectivity in the kinetic setting, where objects move continuously according to known flight plans. See [18, 19] for the case of axis-parallel boxes, and [15] for unit disks.

3 Dynamic Subgraph Connectivity with $\widetilde{O}(m^{2/3})$ Update Time

In this section, we present our new method for the dynamic subgraph connectivity problem: maintaining a subset S of vertices in a graph G, under vertex insertions and deletions in S, so that we can decide whether any two query vertices are connected in the subgraph induced by S. We will call the vertices in S the *active* vertices. For now, we assume that the graph G itself is static.

The complete description of the new method is given in the proof of the following theorem. It is "short and sweet", especially if the reader compares with Chan's paper [6]. The previous method requires several stages of development, addressing the offline and semi-online special cases, along with the use of FMM—we completely bypass these intermediate stages, and FMM, here. Embedded below, one can find a number of different ideas (some also used in [6]): rebuilding periodically after a certain number of updates, distinguishing "high-degree" features from "low-degree" features (e.g., see [5, 37]), amortizing by splitting smaller subsets from larger ones, etc. The key lies in the definition of a new, yet deceptively simple, intermediate graph G^*, which is maintained by known polylogarithmic data structures for dynamic connectivity under edge updates [17, 20, 34]. Except for these known connectivity structures, the description is entirely self-contained.

Theorem 1. *We can design a data structure for dynamic subgraph connectivity for a graph $G = (V, E)$ with m edges, having amortized vertex update time $\widetilde{O}(m^{2/3})$, query time $\widetilde{O}(m^{1/3})$, and preprocessing time $\widetilde{O}(m^{4/3})$.*

Proof. We divide the update sequence into phases, each consisting of $q := m/\Delta$ updates. The active vertices are partitioned into two sets P and Q, where P undergoes only deletions and Q undergoes both insertions and deletions. Each vertex insertion is done to Q. At the end of each phase, we move the elements of Q to P and reset Q to the empty set. This way, $|Q|$ is kept at most q at all times.

Call a connected component in (the subgraph induced by) P *high* if the sum of the degrees of its vertices exceeds Δ, and *low* otherwise. Clearly, there are at most $O(m/\Delta)$ high components.

The data structure.

- We store the components of P in a data structure for decremental (deletion-only) connectivity that supports edge deletions in polylogarithmic amortized time.
- We maintain a bipartite multigraph Γ between V and the components γ in P: for each $uv \in E$ where v lies in component γ, we create a copy of an edge $u\gamma \in \Gamma$.
- For each vertex pair u,v, we maintain the value $C[u, v]$ defined as the number of low components in P that are adjacent to both u and v in Γ. (Actually, only $O(m\Delta)$ entries of $C[\cdot, \cdot]$ are nonzero and need to be stored.)
- We define a graph G^* whose vertices are the vertices of Q and components of P:

  (a) For each $u, v \in Q$, if $C[u, v] > 0$, then create an edge $uv \in G^*$.

  (b) For each vertex $u \in Q$ and high component γ in P, if $u\gamma \in \Gamma$, then create an edge $u\gamma \in G^*$.

  (c) For each $u, v \in Q$, if $uv \in E$, then create an edge $uv \in G^*$.

  We maintain G^* in another data structure for dynamic connectivity supporting polylogarithmic-time edge updates.

Justification. We claim that two vertices of Q are connected in the subgraph induced by the active vertices in G iff they are connected in G^*. The "if" direction is obvious. For the "only if" direction, suppose two vertices $u, v \in Q$ are "directly" connected in G by being adjacent to a common component γ in P. If γ is high, then edges of type (b) ensure that u and v are connected in G^*. If instead γ is low, then edges of type (a) ensure that u and v are connected in G^*. By concatenation, the argument extends to show that any two vertices $u, v \in Q$ connected by a path in G are connected in G^*.

Queries. Given two vertices v_1 and v_2, if both are in Q, we can simply test whether they are connected in G^*.

If instead v_j ($j \in \{1, 2\}$) is in a high component γ_j, then we can replace v_j with any vertex of Q adjacent to γ_j in G^*. If no such vertex exists, then because of type-(b) edges, γ_j is an isolated component and we can simply test whether v_1 and v_2 are both in the same component of P.

If on the other hand v_j is in a low component γ_j, then we can exhaustively search for a vertex in Q adjacent to γ_j in Γ, in $\widetilde{O}(\Delta)$ time, and replace v_j with such a vertex. Again if no such vertex exists, then γ_j is an isolated component and the test is easy. The query cost is $\widetilde{O}(\Delta)$.

Preprocessing per phase. At the beginning of each phase, we can compute the multigraph Γ in $\widetilde{O}(m)$ time. We can compute the matrix $C[\cdot, \cdot]$ in $\widetilde{O}(m\Delta)$ time, by examining each edge $v\gamma \in \Gamma$ and each of the $O(\Delta)$ vertices u adjacent to a low component γ and testing whether $\gamma u \in \Gamma$. The graph G^* can then be initialized. The cost per phase is $\widetilde{O}(m\Delta)$. We can cover this cost by charging every update operation with amortized cost $\widetilde{O}(m\Delta/q) = \widetilde{O}(\Delta^2)$.

Update of a vertex u in Q. We need to update $O(q)$ edges of types (a) and (c), and $O(m/\Delta)$ edges of type (b) in G^*. The cost is $\widetilde{O}(q + m/\Delta) = \widetilde{O}(m/\Delta)$.

Deletion of a vertex from a low component γ in P. The component γ is split into a number of subcomponents. Since the total degree in γ is $O(\Delta)$, we can update the multigraph Γ in $\widetilde{O}(\Delta)$ time. Furthermore, we can update the matrix $C[\cdot,\cdot]$ in $\widetilde{O}(\Delta^2)$ time, by examining each vertex pair u, v adjacent to γ and decrementing $C[u, v]$ if u and v lie in different subcomponents. Consequently, we need to update $O(\Delta^2)$ edges of type (a). The cost is $\widetilde{O}(\Delta^2)$.

Deletion of a vertex from a high component γ in P. The component γ is split into a number of subcomponents $\gamma_1, \ldots, \gamma_\ell$ with, say, γ_1 being the largest. We can update the multigraph Γ in time $\widetilde{O}(\deg(\gamma_2) + \cdots + \deg(\gamma_\ell))$ by splitting the smaller subcomponents from the largest subcomponent. Consequently, we need to update $O(\deg(\gamma_2) + \cdots + \deg(\gamma_\ell))$ edges of type (b) in G^*. Since P undergoes deletions only, a vertex can belong to the smaller subcomponents in at most $O(\lg n)$ splits over the entire phase, and so the total cost per phase is $\widetilde{O}(m)$, which is absorbed in the preprocessing cost of the phase.

For each low subcomponent γ_j, we update the matrix $C[\cdot,\cdot]$ in $\widetilde{O}(\deg(\gamma_j)\Delta)$ time, by examining each edge $\gamma_j v \in \Gamma$ and each of the $O(\Delta)$ vertices u adjacent to γ_j and testing whether $\gamma_j u \in \Gamma$. Consequently, we need to update $O(\deg(\gamma_j)\Delta)$ edges of type (a) in G^*. Since a vertex can change from being in a high component to a low component at most once over the entire phase, the total cost per phase is $\widetilde{O}(m\Delta)$, which is absorbed by the preprocessing cost.

Finale. The overall amortized cost per update operation is $\widetilde{O}(\Delta^2 + m/\Delta)$. Set $\Delta = m^{1/3}$. $\square$

Note that edge insertions and deletions in G can be accomodated easily (e.g., see Lemma 2 of the next section).

4 Dynamic Geometric Connectivity with Sublinear Update Time

In this section, we investigate geometric connectivity problems: maintaining a set S of n objects, under insertions and deletions of objects, so that we can decide whether two query objects are connected in the intersection graph of S. (In particular, we can decide whether two query points are connected in the union of S by finding two objects containing the two points, via range searching, and testing connectedness for these two objects.)

By the biclique-cover technique from [6], the result from the previous section immediately implies a dynamic connectivity method for axis-parallel boxes with $\widetilde{O}(n^{2/3})$ update time and $\widetilde{O}(n^{1/3})$ query time in any fixed dimension.

Unfortunately, this technique is not strong enough to lead to sublinear results for other objects, as we have explained in the introduction. This is because (i) the size of the maintained graph, $m = O(S(n) + nT(n))$, may be too large and (ii) the number of vertex updates triggered by an object update, $O(S(n)/n + T(n))$, may be too large.

We can overcome the first obstacle by using a different strategy that rebuilds the graph more often to keep it sparse; this is not obvious and will be described precisely later during the proof of Theorem 5. The second obstacle is even more critical: here, the key is to observe that although each geometric update requires multiple vertex updates, *many* of these vertex updates involves vertices of *low* degrees.

4.1 A degree-sensitive version of subgraph connectivity

The first ingredient we need is a dynamic subgraph connectivity method that works faster when the degree of the updated vertex is small. Fortunately, we can prove the following lemma, which extends Theorem 1 (if we set $\Delta = n^{1/3}$). The method follows that of Theorem 1, but with an extra twist: not only do we classify components of P as high or low, but we also classify vertices of Q as high or low.

Lemma 2. *Let $1 \leq \Delta \leq n$. We can design a data structure for dynamic subgraph connectivity for a graph $G = (V, E)$ with m edges, having amortized vertex update time*

$$\widetilde{O}(\Delta^2 + \min\{m/\Delta, \deg(u)\})$$

for a vertex u, query time $\widetilde{O}(\Delta)$, preprocessing time $\widetilde{O}(m\Delta)$, and amortized edge update time $\widetilde{O}(\Delta^2)$.

Proof. The data structure is the same as in the proof of Theorem 1, except for one difference: the definition of the graph G^*.

Call a vertex *high* if its degree exceeds m/Δ, and *low* otherwise. Clearly, there are at most $O(\Delta)$ high vertices.

- We define a graph G^* whose vertices are the vertices of Q and components of P:
  - (a$'$) For each high vertex $u \in Q$ and each vertex $v \in Q$, if $C[u, v] > 0$, then create an edge $uv \in G^*$.
  - (b) For each vertex $u \in Q$ and high component γ in P, if $u\gamma \in \Gamma$, then create an edge $u\gamma \in G^*$.
  - (b$'$) For each low vertex $u \in Q$ and each component γ in P, if $u\gamma \in \Gamma$, then create an edge $u\gamma \in G^*$.
  - (c) For each $u, v \in Q$, if $uv \in E$, then create an edge $uv \in G^*$.

  We maintain G^* in a data structure for dynamic connectivity with polylogarithmic-time edge updates.

Justification. We claim that two vertices of Q are connected in the subgraph induced by the active vertices in G iff they are connected in G^*. The "if" direction is obvious. For the "only if" direction, suppose two vertices $u, v \in Q$ are "directly" connected in G by being adjacent to a common component γ in P. If γ is high, then edges of type (b) ensure that u and v are connected in G^*. If u and v are both low, then edges of type (b$'$) ensure that u and v are connected in G^*. In the remaining case, at least one of the two vertices, say, u is high, and γ is low; here, edges of type (a$'$) ensure that u and v are again connected in G^*. The claim follows by concatenation.

Queries. Given two vertices v_1 and v_2, if both are in Q, we can simply test whether they are connected in G^*. If instead v_j ($j \in \{1, 2\}$) is in a component γ_j, then we can replace v_j with any vertex of Q adjacent to γ_j in G^*. If no such vertex exists, then because of type-(b$'$) edges, γ_j can only be adjacent to high vertices of Q. We can exhaustively search for a high vertex in Q adjacent to γ_j in Γ, in $\widetilde{O}(\Delta)$ time, and replace v_j with such a vertex. If no such vertex exists, then γ_j is an isolated component and we can simply test whether v_1 and v_2 are both in γ_j. The cost is $\widetilde{O}(\Delta)$.

Preprocessing per phase. At the beginning of each phase, the cost to preprocess the data structure is $\widetilde{O}(m\Delta)$ as before. We can charge every update operation with an amortized cost of $\widetilde{O}(m\Delta/q) = \widetilde{O}(\Delta^2)$.

Update of a high vertex u in Q. We need to update $O(q)$ edges of types (a′) and (c), and $O(m/\Delta)$ edges of type (b) in G^*. The cost is $\widetilde{O}(q + m/\Delta) = \widetilde{O}(m/\Delta)$.

Update of a low vertex u in Q. We need to update $O(\Delta)$ edges of type (a′), and $O(\deg(u))$ edges of types (b), (b′), and (c) in G^*. The cost is $\widetilde{O}(\deg(u) + \Delta)$.

Deletion of a vertex from a low/high component γ in P. Proceeding exactly as in the proof of Theorem 1, we can update the data structure with amortized cost $\widetilde{O}(\Delta^2)$.

Edge updates. We can simulate the insertion of an edge uv by inserting a new low vertex z adjacent to only u and v to Q. Since the degree is 2, the cost is $\widetilde{O}(1)$. We can later simulate the deletion of this edge by deleting the vertex z from Q. $\square$

4.2 Range searching tools from geometry

Next, we need known range searching techniques. These techniques give linear-space data structures ($S(n) = \widetilde{O}(n)$) that can retrieve all objects intersecting a query object in sublinear time ($T(n) = \widetilde{O}(n^{1-b})$) for many types of geometric objects. We assume that our class of geometric objects satisfies the following property for some constant $b > 0$—this property neatly summarizes all we need to know from geometry.

Property 3. *Given a set P of n objects, we can form a collection $\mathcal{C}$ of* canonical subsets *of total size $\widetilde{O}(n)$, in $\widetilde{O}(n)$ time, such that the subset of all objects of P intersecting any object z can be expressed as the union of disjoint subsets in a subcollection $\mathcal{C}_z$ of $\widetilde{O}(n^{1-b})$ canonical subsets, in $\widetilde{O}(n^{1-b})$ time. Furthermore, for every $1 \leq \Delta \leq n$,*

(i) *the number of subsets in $\mathcal{C}_z$ of size exceeding n/Δ is $\widetilde{O}(\Delta^{1-b})$.*

(ii) *the total size of all subsets in $\mathcal{C}_z$ of size at most n/Δ is $\widetilde{O}(n/\Delta^b)$.*

The property is typically proved by applying a suitable “partition theorem” in a recursive manner, thereby forming a so-called “partition tree”; for example, see the work by Matoušek [28] or the survey by Agarwal and Erickson [2]. Each canonical subset corresponds to a node of the partition tree (more precisely, the subset of all objects stored at the leaves underneath the node). Matoušek’s results imply that $b = 1/d - \varepsilon$ is attainable for simplices or constant-size polyhedra in $\mathbb{R}^d$. (To go from simplex range searching to intersection searching, one uses multi-level partition trees; e.g., see [29].) Further results by Agarwal and Matoušek [3] yield $b = 1/(d+1) - \varepsilon$ for balls in $\mathbb{R}^d$ and nontrivial values of b for other families of curved objects (semialgebraic sets of constant degree). The special case of axis-parallel boxes corresponds to $b = 1$.

The specific bounds in (i) and (ii) may not be too well known, but they follow from the hierarchical way in which canonical subsets are constructed. For example, (ii) follows since the subsets in $\mathcal{C}_z$ of size at most n/Δ are contained in $\widetilde{O}(\Delta^{1-b})$ subsets of size $\widetilde{O}(n/\Delta)$. In fact, (multi-level) partition trees guarantee a stronger inequality, $\sum_{C \in \mathcal{C}_z} |C|^{1-b} = \widetilde{O}(n^{1-b})$, from which both (i) and (ii) can be obtained after a moment’s thought.

As an illustration, we can use the above property to develop a data structure for a special case of dynamic geometric connectivity where insertions are done in “blocks” but arbitrary deletions are to be supported. Although the insertion time is at least linear, the result is good if the block size s is sufficiently large. This subroutine will make up a part of the final solution.

Lemma 4. *We can maintain the connected components among a set S of objects in a data structure that supports insertion of a block of s objects in $\widetilde{O}(n + sn^{1-b})$ amortized time ($s < n$), and deletion of a single object in $\widetilde{O}(1)$ amortized time.*

Proof. We maintain a multigraph H in a data structure for dynamic connectivity with polylogarithmic edge update time (which explicitly maintains the connected components), where the vertices are the objects of S. This multigraph will obey the invariant that two objects are geometrically connected iff they are connected in S. We do not insist that H has linear size.

Insertion of a block B to S. We first form a collection $\mathcal{C}$ of canonical subsets for $S \cup B$ by Property 3. For each $z \in B$ and each $C \in \mathcal{C}_z$, we *assign* z to C. For each canonical subset $C \in \mathcal{C}$, if C is assigned at least one object of B, then we create new edges in H linking all objects of C and all objects assigned to C in a path. (If this path overlaps with previous paths, we create multiple copies of edges.) The number of edges inserted is thus $\widetilde{O}(n + |B|n^{1-b})$.

Justification. The invariant is satisfied since all objects in a canonical subset C intersect all objects assigned to C, and are thus all connected if there is at least one object assigned to C.

Deletion of an object z from S. For each canonical subset C containing or assigned the object z, we need to delete at most 2 edges and insert 1 edge to maintain the path. As soon as the path contains no object assigned to C, we delete all the edges in the path. Since the length of the path can only decrease over the entire update sequence, the total number of such edge updates is proportional to the initial length of the path. We can charge the cost to edge insertions. $\square$

4.3 Putting it together

We are finally ready to present our sublinear result for dynamic geometric connectivity. We again need the idea of rebuilding periodically, and splitting smaller sets from larger ones. In addition to the graph H (of superlinear size) from Lemma 4, which undergoes insertions only in blocks, the key lies in the definition of another subtly crafted intermediate graph G (of linear size), maintained this time by the subgraph connectivity structure of Lemma 2. The definition of this graph involves multiple types of vertices and edges. The details of the analysis and the setting of parameters get more interesting.

Theorem 5. *Assume $0 < b \leq 1/2$. We can maintain a collection of objects in amortized update time $\widetilde{O}(n^{1-b^2/(2+b)})$ and answer connectivity queries in time $\widetilde{O}(n^{b/(2+b)})$.*

Proof. We divide the update sequence into phases, each consisting of $y := n^b$ updates. The current objects are partitioned into two sets X and Y, where X undergoes only deletions and Y undergoes both insertions and deletions. Each insertion is done to Y. At the end of each phase, we move the elements of Y to X and reset Y to the empty set. This way, $|Y|$ is kept at most y at all times.

At the beginning of each phase, we form a collection $\mathcal{C}$ of canonical subsets for X by Property 3.

The data structure.

- We maintain the components of X in the data structure from Lemma 4.
- We maintain the following graph G for dynamic subgraph connectivity, where the vertices are objects of $X \cup Y$, components of X, and the canonical subsets of the current phase:

(a) Create an edge in G between each component of X and each of its objects.

(b) Create an edge in G between each canonical subset and each of its objects in X.

(c) Create an edge in G between each object $z \in Y$ and each canonical subset $C \in \mathcal{C}_z$. Here, we *assign* z to C.

(d) Create an edge in G between every two intersecting objects in Y.

(e) We make a canonical subset active in G iff it is assigned at least one object in Y. Vertices that are objects or components are always active.

Note that there are $\widetilde{O}(n)$ edges of types (a) and (b), $\widetilde{O}(yn^{1-b})$ edges of type (c), and $O(y^2)$ edges of type (d). For $y = n^b$, the size of G is thus $\widetilde{O}(n)$.

Justification. We claim that two objects are geometrically connected in $X \cup Y$ iff they are connected in the subgraph induced by the active vertices in the graph G. The "only if" direction is obvious. For the "if" direction, we note that all objects in an active canonical subset C intersect all objects assigned to C and are thus all connected.

Queries. We answer a query by querying in the graph G. The cost is $\widetilde{O}(\Delta)$.

Preprocessing per phase. Before a new phase begins, we need to update the components in X as we move all elements of Y to X (a block insertion). By Lemma 4, the cost is $\widetilde{O}(n + yn^{1-b}) = \widetilde{O}(n)$. We can now reinitialize the graph G containing $\widetilde{O}(n)$ edges of types (a) and (b) in $\widetilde{O}(n\Delta)$ time by Lemma 2. We can charge every update operation an amortized cost of $\widetilde{O}(n\Delta/y) = \widetilde{O}(n^{1-b}\Delta)$.

Update of an object z in Y. We need to update $\widetilde{O}(n^{1-b})$ edges of type (c) and $O(y)$ edges of type (d) in G. The cost according to Lemma 2 is $\widetilde{O}(n^{1-b}\Delta^2)$.

Furthermore, because of (e), we may have to update the status of as many as $\widetilde{O}(n^{1-b})$ vertices. The number of such vertices of degree exceeding n/Δ is $\widetilde{O}(\Delta^{1-b})$ by Property 3(i), and the total degree among such vertices of degree at most n/Δ is $\widetilde{O}(n/\Delta^b)$ by Property 3(ii). Thus, according to Lemma 2, the cost of these vertex updates is $\widetilde{O}(n^{1-b}\Delta^2 + \Delta^{1-b} \cdot n/\Delta + n/\Delta^b) = \widetilde{O}(n^{1-b}\Delta^2 + n/\Delta^b)$.

Deletion of an object z in X. We first update the components of X. By Lemma 4, the amortized cost is $\widetilde{O}(1)$. We can now update the edges of type (a) in G. The total number of such edge updates per phase is $O(n \lg n)$, by always splitting smaller components from larger ones. The amortized number of edge updates is thus $\widetilde{O}(n/y)$. The amortized cost is $\widetilde{O}((n/y)\Delta^2) = \widetilde{O}(n^{1-b}\Delta^2)$.

Finale. The overall amortized cost per update operation is $\widetilde{O}(n^{1-b}\Delta^2 + n/\Delta^b)$. Set $\Delta = n^{b/(2+b)}$. $\square$

Note that we can still prove the theorem for $b > 1/2$, by handling the $O(y^2)$ intersections among Y (the type (d) edges) in a less naive way. However, we are not aware of any specific applications with $b \in (1/2, 1)$.

5 Offline Dynamic Geometric Connectivity

For the special case of offline updates, we can improve the result of Section 4 for small values of b by a different method using rectangular matrix multiplication.

Let $M[n_1, n_2, n_3]$ represent the cost of multiplying a Boolean $n_1 \times n_2$ matrix A with a Boolean $n_2 \times n_3$ matrix B. Let $M[n_1, n_2, n_3 \,|\, m_1, m_2]$ represent the same cost under the knowledge that the number of 1's in A is m_1 and the number of 1's in B is m_2. We can reinterpret this task in graph terms: Suppose we are given a tripartite graph with vertex classes V_1, V_2, V_3 of sizes n_1, n_2, n_3 respectively where there are m_1 edges between V_1 and V_2 and m_2 edges between V_2 and V_3. Then $M[n_1, n_2, n_3 \,|\, m_1, m_2]$ represent the cost of deciding, for each $u \in V_1$ and $v \in V_3$, whether u and v are adjacent to a common vertex in V_2.

5.1 An offline degree-sensitive version of subgraph connectivity

We begin with an offline variant of Lemma 2:

Lemma 6. *Let $1 \le \Delta \le q \le m$. We can design a data structure for offline dynamic subgraph connectivity for a graph $G = (V, E)$ with m edges and n vertices, under the assumption that $O(\Delta)$ vertices are classified as* high *and at most m_H edges are incident to high vertices. Updates of a low vertex u take amortized time*

$$\widetilde{O}(M[\Delta, n, q \,|\, m_H, m]/q \;+\; \deg(u)),$$

updates of high vertices take amortized time $\widetilde{O}(q)$, queries take time $\widetilde{O}(\Delta)$, and preprocessing takes time $O(M[\Delta, n, q \,|\, m_H, m])$.

Proof. We divide the update sequence into phases, each consisting of q low-vertex updates. The active vertices are partitioned into two sets P and Q, with $Q \subseteq Q_0$, where P and Q_0 are static and Q undergoes both insertions and deletions. Each vertex insertion/deletion is done to Q. At the end of each phase, we reset Q_0 to hold all $O(\Delta)$ high vertices plus the low vertices involved in the updates of the next phase, reset P to hold all active vertices not in Q_0, and reset Q to hold all active vertices in Q_0. Clearly, $|Q| \le |Q_0| = O(q)$.

The data structure is the same as the one in the proof of Lemma 2, with one key difference: we only maintain the value $C[u, v]$ when u is a high vertex in Q_0 and v is a (high or low) vertex in Q_0. Moreover, we do not need to distinguish between high and low components, i.e., all components are considered low.

During preprocessing of each phase, we can now compute $C[\cdot, \cdot]$ by matrix multiplication in time $O(M[\Delta, n, q \,|\, m_H, m])$, since there are $O(\Delta)$ choices for the high vertex u and $O(q)$ choices for the vertex $v \in Q_0$. The amortized cost per low-vertex update for this step is $O(M[\Delta, n, q \,|\, m_H, m]/q)$.

Updating a high vertex u in Q now requires updating $O(q)$ edges of types (a′) and (c) (there are no edges of type (b) now). The cost is $\widetilde{O}(q)$.

Updating a low vertex u in Q requires updating $O(\Delta)$ edges of type (a′), and $O(\deg(u))$ edges of types (b′) and (c) in G^*. The cost is $\widetilde{O}(\deg(u))$.

Deletions in P do not occur now. $\square$

5.2 Sparse and dense rectangular matrix multiplication

Sparse matrix multiplication can be reduced to multiplying smaller dense matrices, by using a "high-low" trick [5]. Fact 7(i) below can be viewed as a variant of [6, Lemma 3.1] and a result of Yuster and Zwick [38]—incidentally, this fact is sufficiently powerful to yield a simple(r) proof of Yuster and Zwick's sparse matrix multiplication result, when combined with known bounds on dense rectangular matrix multiplication. Fact 7(ii) below states one known bound on dense rectangular matrix multiplication which we will use.

Fact 7.

(i) *For* $1 \le t \le m_1$*, we have* $M[n_1, n_2, n_3 \,|\, m_1, m_2] = O(M[n_1, m_1/t, n_3] + m_2 t)$.

(ii) *Let* $\alpha = 0.294$*. If* $n_1 \le \min\{n_2, n_3\}^\alpha$*, then* $M[n_1, n_2, n_3] = \widetilde{O}(n_2 n_3)$.

Proof. For (i), consider the tripartite graph setting with vertex classes V_1, V_2, V_3. Call a vertex in V_2 *high* if it is incident to at least t vertices in V_1, and *low* otherwise. There are at most $O(m_1/t)$ high vertices. For each $u \in V_1$ and $v \in V_3$, we can determine whether u and v are adjacent to a common high vertex, in $O(M[n_1, m_1/t, n_3])$ total time. On the other hand, we can enumerate all $(u, v) \in V_1 \times V_3$ such that u and v are adjacent to a common low vertex, in $O(m_2 t)$ time, by examining each edge wv with $(w, v) \in V_2 \times V_3$ and each of the at most t neighbors $u \in V_1$ of w.

For (ii), Huang and Pan [21] have shown that $M[n^\alpha, n, n] = M[n, n^\alpha, n] = \widetilde{O}(n^2)$. Thus,

$$M[n_1, n_2, n_3] \;=\; O(\left\lceil n_2/n_1^{1/\alpha}\right\rceil \cdot \left\lceil n_3/n_1^{1/\alpha}\right\rceil \cdot M[n_1, n_1^{1/\alpha}, n_1^{1/\alpha}]) \;=\; O(\left\lceil n_2/n_1^{1/\alpha}\right\rceil \cdot \left\lceil n_3/n_1^{1/\alpha}\right\rceil \cdot n_1^{2/\alpha}).$$

□

5.3 Putting it together

We now present our offline result for dynamic geometric connectivity using Lemma 6. Although we also use Property 3, the design of the key graph G is quite different from the one in the proof of Theorem 5. For instance, the size of the graph is larger (and no longer $\widetilde{O}(n)$), but the number of edges incident to high vertices remains linear; furthermore, each object update triggers only a constant number of vertex updates in the graph. All the details come together in the analysis to lead to some intriguing choices of parameters.

Theorem 8. *Assume* $0 < b \le 1$*. Let* $\alpha = 0.294$*. We can maintain a collection of objects in amortized time* $\widetilde{O}(n^{\frac{1+\alpha-b\alpha}{1+\alpha-b\alpha/2}})$ *for offline updates and answer connectivity queries in time* $\widetilde{O}(n^{\frac{\alpha}{1+\alpha-b\alpha/2}})$.

Proof. We divide the update sequence into phases, each consisting of q updates, where q is a parameter satisfying $\Delta \le q \le n/\Delta^{1-b}$. The current objects are partitioned into two sets X and Y, with $Y \subseteq Y_0$ where X and Y_0 are static and Y undergoes both insertions and deletions. Each insertion/deletion is done to Y. At the end of each phase, we reset Y_0 to hold all objects involved the objects of the next phase, X to hold all current objects not in Y_0, and Y to hold all current objects in Y_0. Clearly, $|Y| \le |Y_0| = O(q)$.

At the beginning of each phase, we form a collection $\mathcal{C}$ of canonical subsets for $X \cup Y_0$ by Property 3.

The data structure.

- We maintain the components of X in the data structure from Lemma 4.
- We maintain the following graph G for offline dynamic subgraph connectivity, where the vertices are objects of $X \cup Y_0$, components of X, and canonical subsets of size exceeding n/Δ:
  (a) Create an edge in G between each component of X and each of its objects.
  (b) Create an edge in G between each canonical subset C of size exceeding n/Δ and each of its objects in $X \cup Y$.
  (c) Create an edge in G between each object $z \in Y_0$ and each canonical subset $C \in \mathcal{C}_z$ of size exceeding n/Δ. Here, we *assign* z to C.

(d) Create an edge in G between each object $z \in Y_0$ and each object in the union of the canonical subsets in $\mathcal{C}_z$ of size at most n/Δ.

(e) We make a canonical subset active in G iff it is assigned at least one object in Y. We make the vertices in $X \cup Y$ active, and all components active. The *high* vertices are precisely the canonical subsets of size exceeding n/Δ; there are $\widetilde{O}(\Delta)$ such vertices.

Note that there are $\widetilde{O}(n)$ edges of types (a) and (b), $\widetilde{O}(q\Delta^{1-b})$ edges of type (c) by Property 3(i), and $\widetilde{O}(qn/\Delta^b)$ edges of type (d) by Property 3(ii). So the graph has size $m = \widetilde{O}(n + qn/\Delta^b) = \widetilde{O}(qn/\Delta^b)$, and the number of edges incident to high vertices is $m_H = \widetilde{O}(n + q\Delta^{1-b}) = \widetilde{O}(n)$.

Preprocessing per phase. Before a new phase begins, we need to update the components in X as we delete $O(q)$ vertices from and insert $O(q)$ vertices to X. By Lemma 4, the cost is $\widetilde{O}(n + qn^{1-b})$. We can then determine the edges of type (a) in G in $\widetilde{O}(n)$ time. We can now initialize G in $O(M[\Delta, n, q \,|\, n, m])$ time by Lemma 8. We can charge every update operation with an amortized cost of $\widetilde{O}(M[\Delta, n, q \,|\, n, m]/q + n/q + n^{1-b})$.

Update of an object z in Y. We need to make a single vertex update z in G, which has degree $\widetilde{O}(n/\Delta^b)$ by Property 3(ii). Furthermore, we may have to change the status of as many as $\widetilde{O}(\Delta^{1-b})$ high vertices by Property 3(i). According to Lemma 8, the cost of these vertex updates is $\widetilde{O}(M[\Delta, n, q \,|\, n, m]/q + n/\Delta^b + \Delta^{1-b}q)$.

Finale. By Fact 7, assuming that $\Delta \le q^\alpha$ and $q \le n/t$, we have $M[\Delta, n, q \,|\, n, m] = O(M[\Delta, n/t, q] + mt) = \widetilde{O}(nq/t + nqt/\Delta^b)$. Choosing $t = \Delta^{b/2}$ gives $\widetilde{O}(nq/\Delta^{b/2})$.

The overall amortized cost per update operation is thus $\widetilde{O}(n/\Delta^{b/2} + \Delta^{1-b}q + n/q + n^{1-b})$. Set $\Delta = q^\alpha$ and $q = n^{\frac{1}{1+\alpha-b\alpha/2}}$ and the result follows. (Note that indeed $\Delta \le q \le n/\Delta^{1-b}$ and $q \le n/t$ for these choices of parameters.) $\square$

Compared to Theorem 5, the dependence on b of the exponent in the update bound is only $1 - \Theta(b)$ rather than $1 - \Theta(b^2)$. The bound is better, for example, for $b \le 1/4$.

6 Open Problems

Our work opens up many interesting directions for further research. For subgraph connectivity, an obvious question is whether the $\widetilde{O}(m^{2/3})$ vertex-update bound can be improved (without or with FMM); as we have mentioned, improvements beyond $\sqrt{m}$ without FMM are not possible without a breakthrough on the triangle-finding problem. An intriguing question is whether for dense graphs we can achieve update time sublinear in n, i.e., $O(n^{1-\varepsilon})$ (or possibly even sublinear in the degree).

For geometric connectivity, it would be desirable to determine the best update bounds for specific shapes such as line segments and disks in two dimensions. Also, *directed* settings of geometric connectivity arise in applications and are worth studying; for example, when sensors' transmission ranges are balls of different radii or wedges, a sensor may lie in another sensor's range without the reverse being true.

For both subgraph and geometric connectivity, we can reduce the query time at the expense of increasing the update time, but we do not know whether constant or polylogarithmic query time is possible with sublinear update time in general (see [1] for a result on the 2-dimensional orthogonal special case). Currently, we do not know how to obtain our update bounds with linear space (e.g., Theorem 1 requires $\widetilde{O}(m^{4/3})$ space), nor do we know how to get good worst-case update bounds (since the known polylogarithmic results for connectivity under edge updates are all amortized). Also, the queries we have considered are about connectivity between two vertices/objects. Can nontrivial results be obtained for richer queries such as counting the number of connected components (see [1] on the 2-dimensional orthogonal case), or perhaps shortest paths or minimum cut?

References

[1] Peyman Afshani and Timothy M. Chan. Dynamic connectivity for axis-parallel rectangles. In *Proc. 14th European Symposium on Algorithms (ESA)*, pages 16–27, 2006.

[2] Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, pages 1–56. American Mathematical Society, 1999.

[3] Pankaj K. Agarwal and Jirí Matoušek. On range searching with semialgebraic sets. *Discrete & Computational Geometry*, 11:393–418, 1994. See also MFCS'92.

[4] Pankaj K. Agarwal and Marc J. van Kreveld. Connected component and simple polygon intersection searching. *Algorithmica*, 15(6):626–660, 1996. See also WADS'93.

[5] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. See also ESA'94.

[6] Timothy M. Chan. Dynamic subgraph connectivity with geometric applications. In *Proc. 34th ACM Symposium on Theory of Computing (STOC)*, pages 7–13, 2002.

[7] Timothy M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. In *Proc. 17th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 1196–1202, 2006.

[8] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990. See also STOC'87.

[9] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004. See also STOC'03.

[10] Camil Demetrescu and Giuseppe F. Italiano. Trade-offs for fully dynamic transitive closure on DAGs: breaking through the $O(n^2)$ barrier. *Journal of the ACM*, 52(2):147–156, 2005. See also FOCS'00.

[11] David Eppstein. Dynamic Euclidean minimum spanning trees and extrema of binary functions. *Discrete & Computational Geometry*, 13:111–122, 1995.

[12] David Eppstein. Testing bipartiteness of geometric graphs. In *Proc. 15th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 853–861, 2004.

[13] Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4):781–798, 1985. See also STOC'83.

[14] Daniele Frigioni and Giuseppe F. Italiano. Dynamically switching vertices in planar graphs. *Algorithmica*, 28(1):76–103, 2000. See also ESA'97.

[15] Leonidas J. Guibas, John Hershberger, Subhash Suri, and Li Zhang. Kinetic connectivity for unit disks. *Discrete & Computational Geometry*, 25(4):591–610, 2001. See also SoCG'00.

[16] Leonidas J. Guibas, M. H. Overmars, and Micha Sharir. Intersecting line segments, ray shooting, and other applications of geometric partitioning techniques. In *Proc. 1st Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 64–73, 1988.

[17] Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, 1999. See also STOC'95.

[18] John Hershberger and Subhash Suri. Kinetic connectivity of rectangles. In *Proc. 15th ACM Symposium on Computational Geometry (SoCG)*, pages 237–246, 1999.

[19] John Hershberger and Subhash Suri. Simplified kinetic connectivity for rectangles and hypercubes. In *Proc. 12th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 158–167, 2001.

[20] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001. See also STOC'98.

[21] Xiaohan Huang and Victor Y. Pan. Fast rectangular matrix multiplication and applications. *Journal of Complexity*, 14(2):257–299, 1998.

[22] Hiroshi Imai. Finding connected components of an intersection graph of squares in the Euclidean plane. *Information Processing Letters*, 15(3):125–128, 1982.

[23] Hiroshi Imai and Takao Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of Algorithms*, 4(4):310–323, 1983.

[24] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978. See also STOC'77.

[25] Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 81–91, 1999.

[26] Valerie King and Garry Sagert. A fully dynamic algorithm for maintaining the transitive closure. *Journal of Computer and System Sciences*, 65(1):150–167, 2002. See also STOC'99.

[27] Mario Alberto López and Ramakrishna Thurimella. On computing connected components of line segments. *IEEE Transactions on Computers*, 44(4):597–601, 1995.

[28] Jirí Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8:315–334, 1992. See also SoCG'91.

[29] Jirí Matoušek. Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*, 10(2):157–182, 1993.

[30] Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006. See also SODA'04 and STOC'04.

[31] Mihai Pătraşcu and Mikkel Thorup. Planning for fast connectivity updates. In *Proc. 48th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 263–271, 2007.

[32] Liam Roditty and Uri Zwick. A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In *Proc. 36th ACM Symposium on Theory of Computing (STOC)*, pages 184–191, 2004.

[33] Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *Proc. 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 509–517, 2004.

[34] Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32nd ACM Symposium on Theory of Computing (STOC)*, pages 343–350, 2000.

[35] Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proc. 37th ACM Symposium on Theory of Computing (STOC)*, pages 112–119, 2005.

[36] Mikkel Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007. See also STOC'01.

[37] Raphael Yuster and Uri Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In *Proc. 15th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 254–260, 2004.

[38] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Transactions on Algorithms*, 1(1):2–13, 2005. See also ESA'04.