Dynamic Connectivity for Axis-Parallel Rectangles

Peyman Afshani and Timothy M. Chan*

School of Computer Science
University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada
{pafshani, tmchan}@uwaterloo.ca

Abstract. In this paper we give a fully dynamic data structure to maintain the connectivity of the intersection graph of n axis-parallel rectangles. The amortized update time (insertion and deletion of rectangles) is $O(n^{10/11} \operatorname{polylog} n)$ and the query time (deciding whether two given rectangles are connected) is $O(1)$. It slightly improves the update time ($O(n^{0.94})$) of the previous method while drastically reducing the query time (near $O(n^{1/3})$). Our method does not use fast matrix multiplication results and supports a wider range of queries.

1 Introduction

Dynamic connectivity for undirected graphs is one of the most basic problems in data structure design and has been extensively studied [7, 9, 10, 13–15]. Currently the best method, due to Thorup [15], supports insertions and deletions of edges in $O(\log n \log^3 \log n)$ randomized amortized time and can determine whether two vertices are connected in $O(\log n / \log\log\log n)$ time, where n denotes the number of vertices in the graph.

In this paper, we investigate geometric versions of the problem. Perhaps the simplest, and certainly one of the most naturally appealing, version concerns intersection graphs of orthogonal (horizontal and vertical) line segments. Such graphs arise in applications from VLSI design, geographic information systems, and other areas. In the dynamic setting, we want to answer connectivity queries between any two segments, while supporting insertions and deletions of segments.

Surprisingly, this simple-sounding dynamic geometric problem turns out to be quite difficult, more so than the original graph problem. For one thing, we cannot afford to maintain the intersection graph explicitly, because the insertion or deletion of a single object can bring forth as many as $\Omega(n)$ edge updates in the graph every time.

In STOC 2002, the second author [2] discovered the only nontrivial fully dynamic result for the problem known to date: a data structure that has $O(n^{0.939})$ amortized update time and $\tilde{O}(n^{1/3})$ query time. The $\tilde{O}$ notation hides polylogarithmic factors throughout this paper. This data structure more generally

* This work has been supported by an NSERC grant.

works for connectivity queries for axis-parallel rectangles or boxes in any fixed dimension.

The approach in the previous paper was to use so-called "bi-clique covers" to compactify the intersection graph. This process reduces the geometric problem to a new dynamic graph problem: how to maintain an undirected graph under not only edge updates but also vertex updates—namely, turning a vertex "on" or "off"—so that connectivity queries can be answered in the subgraph induced by the "on" vertices. The paper calls this the *dynamic subgraph connectivity* problem. This graph problem was then solved by a combination of techniques, including the use of Coppersmith and Winograd's fast matrix multiplication algorithm [5].

It was observed [2] that connectivity for axis-parallel line segments or boxes in any fixed dimension $d \geq 3$ is equivalent (up to polylogarithmic factors) to dynamic subgraph connectivity, and that dynamic subgraph connectivity is related to matrix multiplication. Thus, the approach taken is the "right" one in higher dimensions. However, the possibility of a different approach that exploits specifically the geometry of the two-dimensional case was left open.

The main result of this paper is a new fully dynamic data structure for connectivity queries among n axis-parallel line segments, or more generally, axis-aligned rectangles in two dimensions. The amortized update time is $\tilde{O}(n^{10/11}) = O(n^{0.910})$, and the query time is $O(1)$.

Although this update time is still fairly large and the improvement may not seem dramatic, we believe that the result is important for several reasons. First, the new method does not use overly complicated matrix multiplication algorithms and is entirely based on "elementary" techniques, and is thus actually implementable. (In the previous method, if Coopersmith and Winograd's algorithm is replaced by Strassen's, the update time would increase to $O(n^{0.984})$.) Second, the new method supports queries in addition to connectivity between two objects; for example, we can decide whether the entire intersection graph is connected, or count the number of connected components. The previous method inherently cannot deal with such queries of a global nature. Third, our significantly lower query time is attractive, especially in applications where queries are more frequent than updates. Finally, the geometric techniques we use are interesting and original; in particular, we introduce a simple but crucial combinatorial lemma about disjoint curves in the plane. This lemma appears new, to the best of our knowledge. (If not, its algorithmic significance has certainly been overlooked; for instance, it leads to at least one new intersection-searching result that cannot be obtained by previous techniques in the area.) We start with this lemma in the next section and then proceed to describe the main algorithm.

2 A Simple Combinatorial Lemma

In this section we prove a combinatorial lemma which later will be used in the analysis of the final algorithm. Consider a set R of n disjoint regions with simply connected boundaries and a set C of disjoint simple curves in the plane. We say two curves are *equivalent* if they cross the exact same set of regions from R. Generally, there could be up to 2^n curves such that no two are equivalent (one for each subset of R). However, we aim to show that if the curves are disjoint, then we cannot have too many such curves. In fact, there can be at most $\Theta(n^3)$ curves such that no two are equivalent.

We mention that a slightly weaker bound can be obtained by using *VC-dimension* techniques [12]. It is possible to show (by a K_5-avoidance argument) that the set system defined by the curves (where the ground set is R and each curve defines the set of regions it intersects) has VC-dimension four. This implies that the number of curves is $O(n^4)$. We omit the details, as the approach we now give produces a better bound:

Lemma 1. *Assume C is a set of pairwise non-crossing curves with common endpoints p and q, $p \neq q$. If no two curves are equivalent in C, then $|C| = O(n)$.*

Proof. Let $c_1, c_2, \ldots, c_m$ be the given curves, ordered clockwise around p. For every i, consider two adjacent curves c_i and c_{i+1}. These two curves are not equivalent, so they do not pass through the same set of regions. This implies that there is at least one region r which intersects exactly one of them. We charge c_i to r. Obviously, we have charged $m-1$ units in total. In the clockwise ordering of the curves, consider the first curve c_i and the last curve c_j which pass through r. All the curves from c_i to c_j intersect r and all the curves from c_j to c_i do not intersect r. Thus the total charge of r is at most two since only c_{i-1} and c_j can be charged to r. This proves $|V| = O(n)$. □

Lemma 2. (The Main Lemma) *If C is a set of disjoint curves containing no pairwise equivalent curves, then $|C| = O(n^3)$.*

Proof. Consider one curve $c \in C$. Begin from one endpoint of c and start erasing (or shrinking) the curve from that endpoint. This process can be viewed as moving the endpoint along the curve. We shrink the curve until the endpoint of the curve lies on a boundary point p of a region r such that r and c only intersect at p. We repeat the same process for the other endpoint of c and call the resulting curve c'. In other words, the curve c' is a minimal sub-curve of c which intersects the exact same regions as c; thus this operation preserves the equivalence relation. (See Figure 1.)

We do this operation on all the curves in C. Let C' be the set of shrunk curves. Every curve $c \in C'$ has the property that it starts from (and ends at) the boundary of a region r and never passes through that region again. For two regions r_i and r_j, let C_{ij} be the set of curves that have one endpoint in r_i and another endpoint in r_j. If we consider only the curves in C_{ij}, then we can contract the regions r_i and r_j to two points and apply Lemma 1. This implies that C_{ij} contains $O(n)$ curves. (For the special case $i = j$, we have $|C_{ii}| \leq 1$, since at most one curve is entirely contained in r_i.) There are $O(n^2)$ different sets of C_{ij} and thus the total number of curves in C' and C is $O(n^3)$. □

The above lemma is good enough to lead to new results for dynamic rectangle connectivity, but the following refinement will lead to a slightly better result:

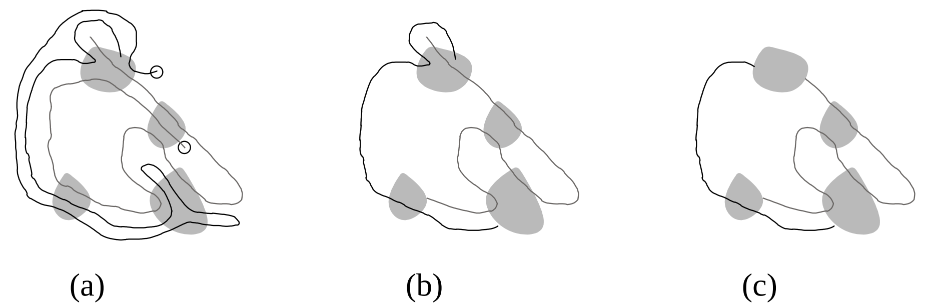

Fig. 1. (a) Erasing from the endpoints marked with circle. (b) Erasing from the other endpoints. (c) The final curves.

Lemma 3. *If C is a set of disjoint curves containing no pairwise equivalent curves and each curve intersects at most k regions, then $|C| = O(nk^2)$.*

Proof. We adapt a random sampling idea by Clarkson and Shor [4] that was originally used for the "($\leq k$)-set" problem.

Take a random sample $Q \subseteq R$ where each region is included with probability $1/k$. We define a planar (multi)graph G_Q where vertices are the contracted regions of Q, as follows. Assume the curves have been shrinked as in the earlier proof. Fix two regions r_i and r_j. Let $c_1, \ldots, c_m$ be the curves in C_{ij} in clockwise order around r_i. For $m = 1$, if r_i and r_j are in Q and all of the $\leq k$ regions intersecting c_1 are not in Q, then add c_1 to G_Q as an edge between r_i and r_j. Observe that the probability that c_1 is added is at least $1/k^2(1-1/k)^k = \Omega(1/k^2)$. For $m > 1$, take each consecutive pair (c_t, c_{t+1}) and let $r(c_t, c_{t+1})$ be a region intersected by c_{t+1} but not c_t, or a region intersected by c_t but not c_{t+1}. Color the pair *red* in the former case, and *blue* otherwise. Without loss of generality, assume that at least half of all pairs are red. For a red pair (c_t, c_{t+1}), if r_i, r_j, and $r(c_t, c_{t+1})$ are in Q and all of the $\leq k$ regions intersecting c_t are not in Q, then add the curve c_t to the graph G_Q as an edge between r_i and r_j. Observe that the probability that c_t is added is at least $1/k^3(1-1/k)^k = \Omega(1/k^3)$. Thus, $E[|E(G_Q)|] = \Omega(|C|/k^3)$.

On the other hand, $E[|V(G_Q)|] = O(n/k)$. By Euler's formula, every planar graph with all face lengths at least 3 (in particular, every simple planar graph) has a linear number of edges. Our graph G_Q is planar but not simple. However, between any 2 parallel edges, there is at least one vertex in G_Q: namely, if the red pairs (c_t, c_{t+1}) and (c_u, c_{u+1}), $t < u$, define 2 edges between r_i and r_j in G_Q, then $r(c_t, c_{t+1})$ would lie entirely between c_t and c_u. Because of this property, we may assume that G_Q has no faces of length 2 (by adding extra edges to isolated vertices if necessary). Thus, $E[|E(G_Q)|] = O(n/k)$.

We can conclude that $|C|/k^3 = O(n/k)$. □

We apply Lemmas 2 and 3 to the case of connected components formed by a set of axis-parallel rectangles, or more generally, polygons. Assume we have a set of polygons which form m connected components and a set R of n disjoint regions as before. We say two connected components are equivalent iff the set of regions they cross is identical.

Corollary 1. *Consider a set S of simple polygons forming a set C of connected components. If C contains no pairwise equivalent components, then $|C| = O(n^3)$. Furthermore, if X is the total number of crossings of the polygons' boundaries with the regions, then $|C| = O(n + n^{1/3}X^{2/3})$.*

Proof. If a component c_i completely contains a region r_j, then we delete both c_i and r_j. Since no other component can intersect r_j, this operation preserves the equivalence relation. By repeating this operation, we remove a total of $O(n)$ components and at the end no region is completely contained in a connected component. Thus, a region r_j intersects c_i iff r_j intersects the boundary of a polygon in c_i. So, it suffices to consider a set S of line segments rather than polygons.

Pick one connected component c_i and look at the arrangement created by the line segments of c_i. Pick one cell except the outer cell of the arrangement and cut one bounding segment of this cell at some arbitrary point (as in Figure 2). This operation connects this cell to its neighboring cell. We repeat this for all the other remaining cells until only one cell, the outer cell, is left in the arrangement. Define the curve c'_i as the Eulerian tour of this arrangement.

The resulting curves are disjoint and they cross the same set of regions as their corresponding connected component. Using Lemma 2 we conclude that $|C| = O(n^3)$.

To get a bound sensitive to X, observe that there are at most X/k components intersecting more than k regions. Using Lemma 3 we conclude that $|C| = O(nk^2 + X/k)$. We can set $k = (X/n)^{1/3}$. □

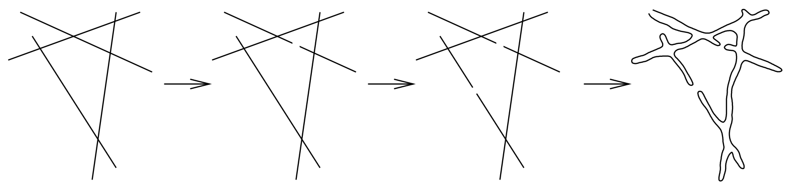

Fig. 2. Changing a connected component into a closed curve.

Remark. The bounds in Lemmas 2 and 3 and Corollary 1 are all tight, as we can see from the following example of a set R of $\Theta(n)$ regions and a set C of $\Theta(nk^2)$ curves with $X = \Theta(nk^3)$, for any given $k \leq n$: Let R contain the $2k$ vertical segments $\{i\} \times [0, n+1]$ for $i = -k, \ldots, -1$ and $i = 1, \ldots, k$, as well as n short vertical segments $\{0\} \times [t - \varepsilon, t + \varepsilon]$ for $t = 1, \ldots, n$. Let C contain nk^2 horizontal segments $[i, j] \times \{t\}$ for all $i = -k, \ldots, -1$, $j = 1, \ldots, k$, and $t = 1, \ldots, n$. Small perturbations can ensure that the segments in C are disjoint. No two segments in C are equivalent.

3 Dynamic Connectivity for Rectangles

We begin with a few preliminaries which will act as building blocks for the final fully dynamic algorithm. Let S be a set of n axis-parallel rectangles in the plane. Fix a parameter q and build a $q \times q$ grid by drawing vertical (and similarly horizontal) lines at every $4n/q$-th corner point of S. This construction ensures that each vertical or horizontal slab contains $O(n/q)$ corners. We call this grid a q-grid. Let the set of regions R be the set of $\Theta(q^2)$ non-crossing (vertical and horizontal) line segments which form this grid and let C be the set of connected components of S. We define equivalence as before and give subroutines that help in computing and maintaining the corresponding equivalence classes.

3.1 Equivalence-Class Management and Decremental Connectivity

Let c be a connected component of the rectangles. We begin by defining a canonical *representation* for the set of regions (grid segments) intersected by c, with the intention that two components are equivalent iff their representations are identical. (A naive bit-vector representation of size $\Theta(q^2)$ would be too long for our purposes.)

First, if a rectangle in c entirely contains a region r, then c is the only component of its class. In this case we store r as the representation for c. If this is not the case, then we proceed to find a representation for the set of regions intersected by b, the union of all line segments bounding the rectangles in c. Consider i-th row of the grid (a horizontal slab) and let $r_{i,0}, \cdots, r_{i,q}$ be the regions (vertical grid segments) contained in this row, ordered from left to right. We represent the regions intersected by b in this row by a list of *intervals*, $(r_{i,j_1}, r_{i,j_1'}), \cdots, (r_{i,j_k}, r_{i,j_k'})$, where $j_1 \leq j_1' \leq \cdots \leq j_k \leq j_k'$. Here, an interval $(r_{i,j}, r_{i,j'})$ indicates that b intersects regions $r_{i,j+1}$ to $r_{i,j'-1}$ but not the regions $r_{i,j}$ and $r_{i,j'}$. We build a similar representation for the columns of the grid (vertical slabs) and define the representation of c to be the concatenation of these lists for all the rows and columns of the grid. Note that the size of this representation is $O(\min\{|c|, q^2\})$, where $|c|$ denotes the number of rectangles in c.

Lemma 4. *Given a q-grid and a set of n axis-parallel rectangles, we can find the connected components and the set of equivalence classes in $\tilde{O}(n)$ time.*

Proof. The connected components can be found in $O(n \log n)$ time by a sweep-line algorithm [11]. To build the classes, we need to compute the representation for each connected component c. Within each row, the key subproblem is to compute the union of the x-intervals of the horizontal segments contained in the row. By sorting and scanning, the union of any m given (one-dimensional) intervals can be constructed in $O(m \log m)$ time. Within each column, we have a similar subproblem. The total time to compute the representation for c is therefore $O(|c| \log |c|)$.

If we interpret the representation of the components as long strings, we can compare the representation of two components c_1 and c_2 in $O(\min\{|c_1|, |c_2|\})$ time using the lexicographical ordering of strings. Since the total size of these strings is $O(n)$, we can sort the strings lexicographically in $\tilde{O}(n)$ time, for example, by mergesort. This will put all the elements of the same class in consecutive order. Finally, a linear scan can be used to separate the components into classes. $\square$

Lemma 5. *Given a connected component c and a set of classes L sorted lexicographically by their representations, in time $O(|c| \log n)$ we can find a class $\ell \in L$ corresponding to c or conclude that no such class exists.*

Proof. We can use binary search on the sorted list of representations to find the proper component. Just note that each comparison takes $O(|c|)$ time. $\square$

We need one more subroutine for our final algorithm: a *decremental* data structure that maintains connectivity of a set of rectangles under deletions. In [2] it was noted that by using a compact representation (bi-clique covers) for the intersection graph of the rectangles, we can obtain a decremental algorithm with $\tilde{O}(1)$ amortized update time. This is achieved by applying a known decremental connectivity algorithm for graphs, e.g., [14], which can explicitly maintain the connected components and in particular answer queries in constant time. Thus we have the following lemma.

Lemma 6. *Given a set of n axis-parallel rectangles, we can maintain the connected components under any sequence of deletions in $\tilde{O}(n)$ total time and answer queries in constant time.*

3.2 The Fully Dynamic Method

Our overall strategy is similar to the overall strategy of the previous method [2] (which in turn is based on an idea from [3]): we will be "lazy" about insertions but periodically rebuild the data structure to limit the "damage" caused by these insertions. Let r be a parameter which will be determined later. After every r updates we rebuild the whole data structure so that we can assume at any given time there have been less than r updates.

The preprocessing and data structure: We build a q-grid and compute the connected components and the corresponding set of classes according to Lemma 4. The amortized cost of the preprocessing over r updates is $\tilde{O}(n/r)$.

Let M be the maximum number of equivalence classes. According to Corollary 1, $M = O(|R|^3) = O(q^6)$. Noting that the number of crossings of the rectangles' boundaries with the q-grid is $X = O(nq)$, we also get an alternative bound $M = O(|R| + |R|^{1/3}X^{2/3}) = O(q^2 + q^{4/3}n^{2/3})$.

The data structure maintains the connectivity of a graph H that contains three types of vertices:

1. Rectangle vertex, i.e., a vertex corresponding to a rectangle inserted after the latest rebuild.
2. Class vertex, i.e., a vertex corresponding to an equivalent class for some subset of the connected components of the rectangles not represented by the rectangle vertices.
3. Component vertex, i.e., a vertex corresponding to a connected component not represented by the class vertices.

For each class vertex, we store a list of its connected components. The components (in both class and component vertices) are maintained in a decremental connectivity data structure by Lemma 6. For each component, we also keep a data structure for orthogonal intersection search, e.g., [6].

With a slight abuse of notation we use vertices of H to refer to their corresponding geometric objects as well. The graph H is defined as the intersection graph of the three types of geometric objects listed above and is stored in a connectivity data structure that supports polylogarithmic edge updates, e.g., [10]. We will ensure the following invariant: whenever there is an edge between a rectangle vertex s_i and a class vertex ℓ_j, the rectangle s_i intersects *all* the connected components in ℓ_j.

Initially, the graph has isolated class vertices and no rectangle or component vertices. An exception is the class corresponding to components intersecting no regions; we break down this class into $O(q^2)$ class vertices, one vertex for all the components lying inside a cell.

We will ensure that each update deletes at most $O(n/q)$ vertices of H and that the number of component vertices at the end is bounded by $O(rn/q)$. We already know that the number of class vertices at the end is at most M. Thus the total number of vertices of H created during the r updates is $O(M + rn/q)$. Now, we describe the update and query algorithms.

Insertion of a rectangle s: Consider the two horizontal slabs and two vertical slabs of the grid containing the corners of s. These *special* slabs contain $O(n/q)$ corners and thus there are $O(n/q)$ components with a corner in these slabs. We go through each such component c_i and remove them from their corresponding class ℓ_j and add one component vertex c_i to the graph H. Since c_i was previously a part of ℓ_j, we add an edge from c_i to each vertex of H adjacent to ℓ_j. We delete all the empty class vertices.

Now we claim that the invariant still holds, i.e., if a connected component c_1 of a class vertex ℓ intersects s, then any other member c_2 of ℓ intersects s as well. We consider two cases. The first case is when s entirely contains c_1. Note that if c_1 does not intersects any region then its cell must be contained by S since we have removed all the components from the special slabs of s. So we can assume c_1 intersects at least one region. Pick any region r_1 intersected by c_1. Since c_1 does not have any corners in the special slabs (otherwise it would be removed), s entirely contains r_1. Since c_1 and c_2 are equivalent, c_2 must also intersect r_1, and thus s. The second case is when c_1 intersects a bounding segment s' of s; say s' is horizontal. Since c_1 does not have any corners in the special slabs, we can find a grid cell (rectangle) such that c_1 cuts through the cell vertically while s' cuts through it horizontally. Since c_1 and c_2 are equivalent and c_2 does not have any corners in the special slabs either, c_2 must cut through the same cell vertically and must intersect s', and thus s.

Next we add a rectangle vertex s to H, and add an edge to each vertex it intersects. We can decide whether a component intersects s in polylogarithmic time by querying an orthogonal intersection search structure [6]. According to the invariant, for a class vertex, we only need to examine one of its components. Hence insertion can be performed in $\tilde{O}(|V(H)|) = \tilde{O}(M + rn/q)$ time. Notice that each insertion deletes $O(n/q)$ vertices and adds $O(n/q)$ new component vertices to H, which is acceptable.

Deletion of a rectangle s: For deletion, we use a "weighted split" strategy (like in [2]). We consider two cases.

- s is a rectangle vertex in H: In this case we simply remove the vertex s and at most $O(M + rn/q)$ incident edges.
- s is a rectangle inside a connected component c (either located inside a class vertex or a component vertex): In this case we remove s from the component by the decremental connectivity data structure. The amortized cost of this operation is $\tilde{O}(n/r)$ over r updates. This operation splits c into smaller connected components, $c_1, \ldots, c_z$ sorted in decreasing order according to size. Let $m = |c_2| + \cdots + |c_z|$. We can rebuild the classes corresponding to all the rectangles in $c_2, \ldots, c_z$ in $\tilde{O}(m)$ time according to Lemma 4 and insert them into the class vertices of H within the same time bound according to Lemma 5. Components that have a corner in the same slab as one of the $O(r)$ corners of the rectangle vertices, however, are moved to new component vertices; the earlier argument implies that this preserves the invariant. Note that the number of such component vertices at the end is $O(rn/q)$, which is acceptable. We can obtain intersection-search structures for $c_1, \ldots, c_z$ from the structure for c by performing $O(m)$ update operations in $\tilde{O}(m)$ additional time.
  For each new class/component vertex created in this phase, we add an edge to each of the $O(r)$ rectangle vertices it intersects. Since the total number of vertices created during the r updates is $O(M + rn/q)$, the total cost of this step is $\tilde{O}(rM + r^2 n/q)$, which yields an amortized cost of $\tilde{O}(M + rn/q)$.
  The size of each c_i, $2 \le i \le z$ is at most half the size of c. Thus the sum of all the m values encountered during the entire update sequence is $O(n \log n)$. This implies that the amortized time for processing the smaller components is $\tilde{O}(n/r)$. To take care of c_1 we simply remove it from its corresponding class vertex and add a single component vertex to H and add edges to all rectangle vertices it intersects, in $\tilde{O}(r)$ additional time.

After each update we regenerate the connected components of the graph H in $\tilde{O}(M + rn/q)$ time. We conclude that the overall amortized update time is

$\tilde{O}(n/r + q^2 + q^{4/3}n^{2/3} + rn/q)$, which is asymptotically minimized by picking $q = r^2$ and $r = n^{1/11}$.

Connectivity query between rectangles u and v: Given pointers to u and v, we want to determine whether u and v are connected. We first find the components containing u and v from the decremental structure in constant time (by Lemma 6). If u and v are inside the same component (either in a component vertex or class vertex), we return "yes". If they are in different components but inside the same class vertex ℓ, we return "yes" iff ℓ is not an isolated vertex in H. Otherwise, we return "yes" iff the vertex containing u and the vertex containing v are connected in the graph H. Since we know all the connected components of H, the query time is $O(1)$.

Thus we have proved the following theorem.

Theorem 1. *Given n axis-parallel rectangles, there exists a deterministic fully dynamic data structure which performs updates in $\tilde{O}(n^{10/11}) = O(n^{0.910})$ amortized time and answers connectivity queries in constant time.*

Remarks. Note that we can determine the connectivity between two points in the plane in polylogarithmic time, simply by performing orthogonal range search queries to find two rectangles containing the two points.

Our method enables us to answer many other types of queries by storing additional information. For example, the following queries cannot be handled by the previous method [2].

- We can determine the number of connected components in the given set of rectangles in $O(1)$ time: we just record the number of components that are in isolated class vertices of H, as well as the number of connected components in the graph H itself, after each update.
- We can determine whether the entire set of rectangles is connected in $O(1)$ time, just by checking whether the number of connected components is 1.
- We can list all rectangles in the same connected component as a given rectangle s in time proportional to the output size: if s is in an isolated class vertex, we just report all rectangles in the component of s from the decremental structure; otherwise, we report all rectangles that appears in the vertices of the connected component of the graph H containing s.

4 More Results

4.1 An Offline Method

It is possible to improve the update time of the algorithm if the list of insertions is known in advance. In this case queries and deletions may be online. Let I be the set of rectangles of the r future insertions. Instead of using a q-grid, we let the set of regions R be the set of all $O(r^2)$ non-crossing line segments in the arrangement of the bounding segments in I. We build equivalence classes with respect to this set of regions. We use the bound $M = O(r^6)$ on the number of equivalence classes.

The rest of the method is mostly the same, except that the invariant is more easily satisfied: when inserting a rectangle s, we know that each of its bounding segments s' is covered exactly by regions of R, so obviously if one connected component $c_1 \in \ell$ intersects s', then all other members of ℓ intersect s' as well. For this reason we can skip the creation of $O(rn/q)$ component vertices. Thus the number of vertices of H is reduced to $O(r^6)$. This implies that the amortized cost of an update is $\tilde{O}(r^6 + n/r)$, which is asympotically minimized for $r = n^{1/7}$ and yields an update time of $\tilde{O}(n^{6/7}) = O(n^{0.858})$.

4.2 Intersection Searching for Disjoint Rectilinear Polygons

We can also prove the following result by our techniques:

Theorem 2. *Given a set of disjoint simple rectilinear polygons in the plane of total complexity n, we can build a data structure in $\tilde{O}(n)$ time and space, so that we can count the number of polygons intersecting a query rectangle in $\tilde{O}(n^{6/7})$ time.*

Proof. We build a q-grid and the $M = O(q^6)$ equivalence classes for the polygons as in the previous section. For each polygon, we keep an intersection-search data structure. For each equivalence class ℓ, we record the number of its components. As before, we break down the class of components intersecting no regions into $O(q^2)$ smaller classes, one for each cell of the grid.

Given a query rectangle s, we go through the $O(n/q)$ polygons with a corner in one of the special slabs containing the corners of s and increase the counter iff it intersects s. This takes $\tilde{O}(n/q)$ time. We also mark these components to prevent double counting in the next step.

By an earlier argument, we know that if an unmarked component of a class ℓ intersects s, then any other unmarked member of ℓ intersects s as well. So, we go through each class ℓ, and if an arbitrary unmarked component in ℓ intersects s, we increase the counter by the number of unmarked components in ℓ. This takes $\tilde{O}(q^6)$ additional time. The total query time $\tilde{O}(q^6 + n/q)$ is asymptotically minimized for $q = n^{1/7}$. $\square$

Although the above intersection-searching result is somewhat specialized, we think it is of theoretical interest in view of previous work. For example, by standard results on orthogonal range/intersection searching [1], we can count the number of intersections of the polygons with an orthogonal line segment. However, for our problem, segments from the same polygon should be counted only once. On the other hand, by simple known techniques for *colored* intersection searching [8], we can report (the labels of) the polygons intersecting a rectangle in time $O(k \operatorname{polylog} n)$, where k is the number of output polygons. These techniques do not require disjointness of the given rectilinear polygons; however, they do not yield results that are independent of k for the counting problem.

5 Conclusion

We have presented a data structure with sublinear update time and constant query time which has the additional advantage of being able to answer global connectivity queries. Our work leaves a few open questions. The obvious one is reducing the update time, but probably the most challenging open problem is achieving a meaningful lower bound as it seems unlikely that this problem can be solved in polylogarithmic time.

Another direction is to solve the dynamic connectivity problem for other classes of geometric graphs, for instance the intersection graph of arbitrary line segments. Neither the previous technique [2] nor the new technique can be directly applied to this class of graphs (because the bi-clique cover of these graphs does not have linear size, and the management of the equivalent classes also becomes more difficult).

References

1. P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry*, pages 1–56. AMS Press, 1999.
2. T. M. Chan. Dynamic subgraph connectivity with geometric applications. In *Proc. 34th ACM Sympos. on Theory of Comput.*, pages 7–13, 2002.
3. T. M. Chan. Semi-online maintenance of geometric optima and measures. *SIAM J. Comput.*, 32:700–716, 2003.
4. K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.
5. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.*, 9:251–280, 1990.
6. H. Edelsbrunner and H. A. Maurer. On the intersection of orthogonal objects. *Inform. Process. Lett.*, 13:177–181, 1981.
7. M. Fredman and M. Henzinger. Lower bounds for fully dynamic connectivity problems in graphs. *Algorithmica*, 22:351–362, 1998.
8. P. Gupta, R. Janardan, and M. Smid. Computational geometry: generalized intersection searching. In *Handbook of Data Structures and Applications*, pages 64–1–64–17. Chapman & Hall/CRC, Boca Raton, FL, 2005.
9. M. R. Henzinger and V. King. Randomized dynamic graph algorithms with polylogarithmic time per operation. *J. of the ACM*, 46:76–103, 2000.
10. J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. of the ACM*, 48:723–760, 2001.
11. H. Imai and T. Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *J. Algorithms*, 4(4):310–323, 1983.
12. J. Matoušek. *Lectures on Discrete Geometry.* Springer-Verlag, 2002.
13. M. Pătraşcu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35(4):932–963, 2006.
14. M. Thorup. Decremental dynamic connectivity. *J. Algorithms*, 33(2):229–243, 1999.
15. M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32nd ACM Sympos. on Theory of Comput.*, pages 343–350, 2000.