Lower Bounds for Fully Dynamic Connectivity Problems in Graphs

Michael L. Fredman*
Monika Rauch Henzinger †

Abstract

We prove lower bounds on the complexity of maintaining fully dynamic k-edge or k-vertex connectivity in plane graphs and in $(k-1)$-vertex connected graphs. We show an amortized lower bound of $\Omega(\log n / k(\log\log n + \log b))$ per edge insertion, deletion, or query operation in the cell probe model, where b is the word size of the machine and n is the number of vertices in G. We also show an amortized lower bound of $\Omega(\log n / (\log\log n + \log b))$ per operation for fully dynamic planarity testing in embedded graphs. These are the first lower bounds for fully dynamic connectivity problems.

1 Introduction

This paper investigates lower bounds for fully dynamic data structures. We call a graph G *plane* if G is planar and we are given a fixed *embedding* of G. An *embedding* of a graph G is uniquely determined by fixing at each vertex the order of its incident edges. Two nodes x and y are k-edge (k-vertex) connected if there are k edge-disjoint (vertex-disjoint) paths connecting them. For further terminology we refer to [5].

Given a graph G, the *fully dynamic k-edge (k-vertex) connectivity problem* is to execute the following operations in arbitrary order:

$Insert(u, v)$: Add the edge (u, v) to G.

$Delete(u, v)$: Remove the edge (u, v) from G.

$Query(u, v)$: Return yes, if u and v are k-edge (k-vertex) connected, and no otherwise.

In the *fully dynamic planarity testing problem* we execute a sequence of edge insertions and deletions interleaved with queries of the form

*Department of Computer Science, Rutgers University, New Brunswick, NJ 08903.
†Department of Computer Science, Cornell University, Ithaca, NY 14853, USA. Email: mhr@cs.cornell.edu. Maiden name: Monika H. Rauch.

$Query(u, v)$: Return yes, if inserting the edge (u, v) does not destroy the planarity of the graph, and no otherwise.

in arbitrary order.

If G is plane, an insertion is given as parameter the new edge (u, v) and also the location of this edge in the order of edges at vertex u and at vertex v. We require that the resulting embedding remains planar. We call fully dynamic problems with these restrictions *fully dynamic problems in plane graphs.*

We give in Figure 1 below the current best upper bounds in plane graphs, planar graphs, and general graphs. For general graphs we present both, the best deterministic and the best randomized bounds.

	plane	planar	general, rand.	general, det.
connectivity	$O(\log n)$ [2]	$O(\log^2 n)$ [4]	$O(\log^3 n)$ [7]	$O(\sqrt{n})$ [3]
2-edge-connectivity	$O(\log^2 n)$ [9]	$O(\log^2 n)$ [4]	$O(\log^4 n)$ [7]	$O(\sqrt{n})$ [3]
2-vertex-connectivity	$O(\log^2 n)$ [11]	$O(\sqrt{n})$ [4]		$O(\sqrt{n}\log^{3/2} n)$ [8]
3-edge-connectivity		$O(\sqrt{n})$ [4]		$O(n^{2/3})$ [3]
3-vertex-connectivity		$O(\sqrt{n})$ [4]		$O(n)$ [3]
4-edge-connectivity		$O(\sqrt{n})$ [4]		$O(n\alpha(n))$ [3]
planarity testing	$O(\log^2 n)$ [10]			$O(\sqrt{n})$ [4]

Figure 1: The best upper bounds for fully dynamic problems.

We establish lower bounds on the time per operation for the above problems in the cell probe model. We show the lower bounds in the cell probe model of computation with wordsize b [16]. The time complexity of a sequential computation is defined to be the number of accessed memory words in a RAM of fixed word size b. All other operations are free. This model is at least as powerful as a random access machine of the same word size.

Given a graph G with n nodes we show the following bounds for arbitrary wordsize b:

1. A lower bound of $\Omega(\log n/(\log\log n + \log b))$ on the time per operation for the fully dynamic planarity testing problem in embedded and in general graphs.

2. A lower bound of $\Omega(\log n/(k(\log\log n + \log b)))$ on the time per operation for the fully dynamic k-edge or k-vertex connectivity problem in (general) (k-1)-vertex connected graphs. The lower bound holds even if every query tests the same two nodes s and t.

3. A lower bound of $\Omega(\log n/(k(\log\log n + \log b)))$ on the time per operation for the fully dynamic k-edge or k-vertex connectivity problem in plane c-vertex connected graphs, where $c = \min(k - 1, 2)$. The lower bound holds even if every query tests the same two nodes s and t.

These lower bounds show that the upper bounds for connectivity and 2-edge connectivity in plane graphs and the randomized bounds in general graphs are close to optimal. The fact that the lower bounds also apply to (k-1)-vertex connected graphs (and thus also to (k-1)-edge connected graphs) implies that the "hardness" of the k-edge (k-vertex) connectivity problem does not depend on the "hardness" of the (k-1)-edge ((k-1)-vertex) connectivity problem.

We review next some related work. Westbrook and Tarjan [14] gave a lower bound for a (stronger) variant of the dynamic connectivity, 2-edge-connectivity, and 2-vertex-connectivity problems. Their model of a dynamic algorithm requires that a query returns the component to which a vertex belongs. In this stronger model they proved that for the separable pointer machine model for any n and any m there exists a sequence of m edge insertions and backtracking edge deletions whose cost is $\Omega(m \log n / \log\log n)$.

If only insertions and queries, but no deletions are allowed, then tight lower bounds of $\Omega(\alpha(m,n))$ per operation are known for connectivity, 2-edge connectivity, 2-vertex connectivity, 3-edge connectivity, 3-vertex connectivity, and planarity testing [15, 13] by reducing these problems to the union-find problem.

An earlier version of this work has appeared in [11].

In the next section we give the general ideas of the lower bound proofs. In Section 3 and Section 4 we present the proofs for fully dynamic planarity testing and k-edge and k-vertex connectivity if $b < n^{1/\log\log n}/\log n$. In Section 5 we extend the proof for general b and remove the dependency on b in a more specific model of computation.

2 The General Idea

The lower bound proofs given in this paper have the following structure. We reduce the *parity prefix sum problem* to the dynamic problem. The parity prefix sum problem is defined as follows: Given an array $A[1], \ldots, A[n]$ of integers mod 2 with initial value zero execute an arbitary sequence of the following operations:

$Add(l)$: Increase $A[l]$ by 1.

$Sum(l)$: Return S_l mod 2, where $S_l = \sum_{i=1}^{l} A[i]$.

An index i is called *even* if $Sum(i)$ returns 0, and *odd* otherwise.

Given an instance of the parity prefix sum problem we construct a fully dynamic problem and a current graph G with at least $n+1$ nodes as follows. We label the nodes by consecutive numbers starting at 0 and partition the set of nodes labeled by i with $1 \leq i \leq n$ into an *even* and an *odd* set as follows: Vertex i is called *even* if $Sum(i)$ returns 0, and *odd* otherwise. The even nodes are connected by a chain of edges, called the *even chain*, the odd nodes are connected by another chain of edges, called the *odd chain*. Each chain might be connected to additional nodes. A $Sum(l)$ operation corresponds to determining the parity of node l which in turn corresponds to inserting and deleting $O(k)$ asking a query in the fully dynamic problem with node l as one of the parameters. An $Add(l)$ operation corresponds to inserting and deleting a constant number of suitable edges. To determine the suitable edges we use a constant number of operations in an additional data structure, namely in a Van-Emde-Boas priority queue [12].

Fredman and Saks [6] give a lower bound of $\Omega(m \log n/(\log\log n + \log b))$ for the parity prefix sum problem for any sequence of m operations in the cell probe model with word size b. The above implemetation of the parity prefix sum problem requires a constant number of operations in the fully dynamic data structure and in the Van-Emde-Boas priority queue. The operations in the latter take time $O(\log\log n)$. Let $T(n)$ be the time for an operation in the fully dynamic data structure. Thus the above implemetation of the parity prefix sum problem takes time $O(T(n) + \log\log n)$ per operation. This implies that $T(n) = \Omega(\log n/(\log\log n + \log b))$ if

$$\log\log n < \frac{\log n}{\log\log n + \log b}.$$

The latter condition is equivalent to $b < n^{1/\log\log n}/\log n$. Thus this shows a lower bound of $\Omega(\log n/(\log\log n + \log b))$ for the fully dynamic problem if the word size $b < n^{1/\log\log n}/\log n$. In Section 5 we describe how to remove the condition on b.

3 A Lower Bound for Fully Dynamic Planarity Testing

To reduce the parity prefix sum problem to the fully dynamic planarity testing problem we construct the following graph G with $n+3$ nodes.

- Vertices 0, $n+1$, and $n+2$ form a triangle.
- Even vertices are connected by a even chain, odd vertices are connected by an odd chain.
- The first vertex f_{even} of the even chain and the first vertex f_{odd} of the odd chain are connected by an edge to vertex 0. The clockwise order of the edges at vertex 0 is as follows: $(0, n+2)$, $(0, n+1)$, $(0, f_{even})$, $(0, f_{odd})$.
- The last vertex l_{even} of the even chain and the last vertex l_{odd} of the odd chain are connected to vertex $n+1$ by an edge. The clockwise order at vertex $n+1$ is $(n+1, l_{odd})$, $(n+1, l_{even})$, $(n+1, 0)$, and $(n+1, n+2)$.
- There is an edge between vertex l_{odd} and vertex $n+2$ such that the order of the edges at $n+2$ is $(n+2, l_{odd})$, $(n+2, n+1)$, $(n+2, 0)$. Let i be the predecessor of l_{odd}. Then $(l_{odd}, n+1)$, $(l_{odd}, n+2)$, and (l_{odd}, i) is the clockwise order of the vertices at l_{odd}.

For an example see Figure 2.

To reduce the parity prefix sum problem to the dynamic planarity testing problem we maintain a dynamic planarity testing data structure for G, the variables f_{even}, f_{odd}, l_{even}, and l_{odd}, and Van-Emde-Boas priority queue (described below).

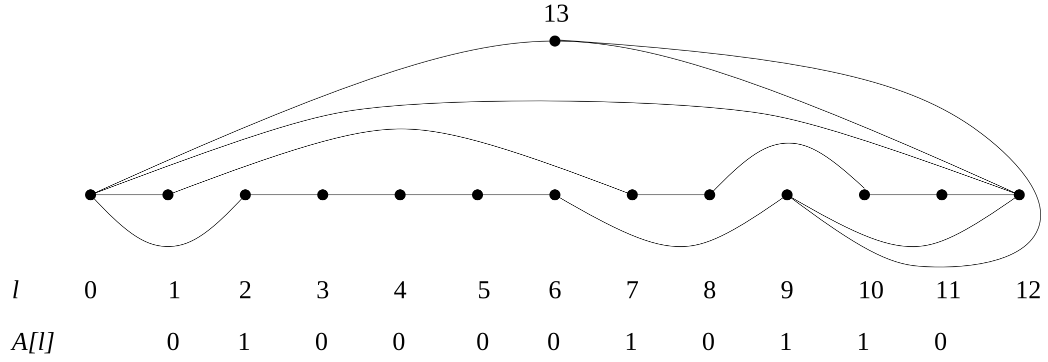

Figure 2: The plane graph for the array 0 1 0 0 0 0 1 0 1 1 0 ($n = 11$).

3.1 A *Sum* Query

The odd chain, along with the edges $(f_{odd}, 0)$, $(0, n+1)$, and $(n+1, l_{odd})$ forms a cycle of edges. All vertices on the even chain are on one side of this cycle, but the edge $(l_{odd}, n+2)$ forces the vertex $n+2$ to be on the other side. Thus adding an edge between a vertex on the even chain and vertex $n+2$ destroys the planarity of the graph, while adding an edge between an vertex on the odd chain and vertex $n+2$ preserves planarity. Hence, an edge between vertex l and vertex $n+2$ can be added to the graph if and only if S_l is odd. This implies that a $Sum(l)$ query can be answered by testing whether an edge between vertex $n+2$ and vertex l preserves the planarity of the graph.

3.2 An *Add* Operation

An $Add(l)$ changes S_j for all $j \geq l$ from odd to even and vice versa. To update the embedding appropriately we have to cut the edge (i_{odd}, j_{odd}) of the odd chain with $i_{odd} < l$ and $j_{odd} \geq l$ and the edge (i_{even}, j_{even}) of the even chain with $i_{even} < l$ and $j_{even} \geq l$. We describe below how to find i_{odd}, j_{odd}, i_{even}, and j_{even} using the Van-Emde-Boas priority queue. Then we insert an edge connecting the first part of the odd chain with the second part of the even chain and vice versa. Thus, only a constant number of edge insertions and deletions are necessary.

If we execute an $Add(4)$ operation in the example of Figure 2, then $(3,4)$ and $(1,7)$ are the edges to be deleted. In the example of Figure 2 the edges $(3,7)$ and $(1,4)$ have to be added. See the result in Figure 3.

To maintain the planarity of the embedding (and also the connectedness of the graph) we execute these steps in the following order.

1. Delete the edges (i_{odd}, j_{odd}) and $(l_{odd}, n+2)$.

2. Insert the edge (i_{even}, j_{odd}) (such that it is consistent with the embedding).

3. Delete the edges $(l_{odd}, n+1)$ and (i_{even}, j_{even}).

4. Replace l_{even} by l_{odd} and vice versa.

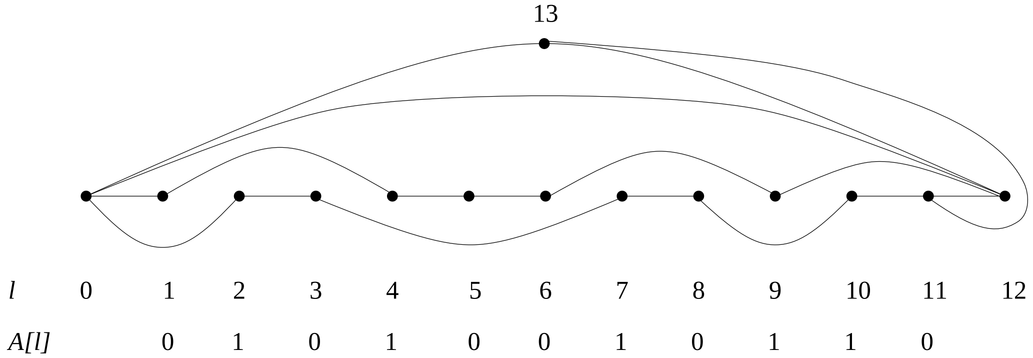

Figure 3: The plane graph of the array 0 1 0 1 0 0 1 0 1 1 0 ($n = 11$).

5. Insert the edges (i_{odd}, j_{even}), $(l_{even}, n+1)$ and $(l_{odd}, n+2)$ (in the right order of the embedding at the vertices l_{odd}, $n+1$, and $n+2$).

To find the vertices $i_{odd}, j_{odd}, i_{even}$, and j_{even} we use a *Van-Emde-Boas priority queue* for finding predecessors in integer sets. Given the universe $\{0, 1, \ldots, n, n+1\}$ and a subset S of the universe this data structure allows us to execute the following operation in arbitrary order:

$Insert(x)$: Insert x into S, if $x \notin S$.

$Delete(x)$: Delete x from S, if $x \in S$.

$Pred(x)$: Find the largest $y \in S$ such that $y \leq x$, or indicate that no such element exists.

$Succ(x)$: Find the smallest $y \in S$ such that $y \geq x$, or indicate that no such element exists.

In the following we denote by $pred(x)$ (resp. $succ(x)$) the value returned by the call $pred(x)$ (resp. $succ(x)$). Note that $x \in S$ implies that $succ(x) = pred(x) = x$.

We store 1, $n+1$, and all indices i such that $S_{i-1} + S_i$ is odd in a Van-Emde-Boas priority queue. The value 1 is inserted to guarantee that $pred(l) - 1 \geq 0$ for all l. The value $n+1$ is inserted to guarantee that $succ(l+1) \leq n+1$ for all $l \leq n$.

The following lemma describes how to find the edges that have to be cut.

Lemma 3.1 *Let S be the value of S_l before an $Add(l)$ operation. If (l',l) with $l' < l$ is the edge on the (odd or even) chain containing l, then*

- $i_{even} = pred(l) - 1$ *and* $j_{even} = succ(l+1)$ *and* $i_{odd} = l'$ *and* $j_{odd} = l$ *if S is odd, and*
- $i_{even} = l'$ *and* $j_{even} = l$ *and* $i_{odd} = pred(l) - 1$ *and* $j_{odd} = succ(l+1)$ *if S is even.*

Proof: If S is odd, the edge (l', l) is the edge of the odd chain that has to be deleted. This shows that $i_{odd} = l'$ and $j_{odd} = l$. To show that $i_{even} = pred(l) - 1$ note that $pred(l)$ is the largest vertex $\leq l$ such that $S_{pred(l)}$ is odd and $S_{pred(l)-1}$ is even. Thus,

$pred(l) - 1$ is the largest vertex smaller than l that lies on the even chain, and hence $i_{even} = pred(l) - 1$. To show that $j_{even} = succ(l+1)$ note that $succ(l+1)$ is the smallest vertex $\geq l+1$ such that $S_{succ(l+1)}$ is even and $S_{succ(l+1)-1}$ is odd. Thus, $succ(l)$ is the smallest vertex larger than l that lies on the even chain, and hence $j_{even} = succ(l+1)$.

The proof is symmetric if S is even. ∎

Since we can decide if S_l is even or odd using the above algorithm, the previous lemma shows how to compute i_{even}, i_{odd}, j_{even}, and j_{odd} using the priority queue. The priority queue can be updated with a constant number of insert and delete operations after each $Add(l)$ operation.

Thus we showed that a *Sum* operation can be answered with one planarity test and an *Add* operation causes a constant number of operations in the dynamic planarity testing data structure and in the Van-Emde-Boas priority queue. As shown in the previous section this implies the following theorem.

Theorem 3.2 *Any fully-dynamic planarity testing algorithm for an embedded graph requires* $\Omega(\log n/(\log\log n + \log b))$ *amortized time per operation in the cell probe model with wordsize* b*, where* $b < n^{1/\log\log n}/\log n$*.*

4 The Lower Bounds for Fully Dynamic Connectivity Problems

We first show the lower bounds for fully dynamic connectivity problems in $k-1$-connected graphs and then in plane graphs.

4.1 The Lower Bound in General Graphs

In this section we prove the following theorem.

Theorem 4.1 *In the cell probe model any algorithm for testing if two fixed nodes* s *and* t *are* k*-edge or* k*-vertex connected in a (k-1)-vertex connected graph under a sequence of insertions and deletions of edges requires* $\Omega(\log n/(k \log\log n + \log b))$ *amortized time per operation with* $b < n^{1/\log\log n}/\log n$*.*

Proof: Let S_0 be 1. For $k > 0$ we still define $S_k = \sum_{j=1}^{k} A[j]$. Given a parity prefix sum problem, we construct a graph consisting of a vertex labeled t and $k(n+1)$ vertices, labeled (l, p), with $0 \leq l \leq n$ and $0 \leq p \leq k-1$. The vertex $(0,0)$ is also labeled s. The graph contains the following edges:

- For a fixed l the vertices (l, p) are connected by a complete graph K_k. The vertices (l, p) represent the sum S_l.
- If S_l is odd and l' is the largest index smaller than l such that $S_{l'}$ is odd, there is an edge $((l', p), (l, p))$ for $0 \leq p \leq k-1$. This creates k *odd* chains. The vertices representing even S_l are connected in the same way and create k *even* chains.

- Let f_{odd} be the lowest index i larger than 0 such that S_i is odd and let f_{even} be the lowest index i such that S_i is even. There is an edge $((f_{odd}, p), (0, p))$ for $0 \leq p \leq k-1$. If $k > 1$, there is an edge $((f_{even}, p), (0, p))$ for $1 \leq p \leq k-1$. (Note that we do not insert an edge for $p = 0$.)
- For each $0 \leq p \leq k-1$ there is an edge $(t, (n, p))$.

Note that the resulting graph is (k-1)-vertex connected. In the example of Figure 4 $f_{even} = 3$ and $f_{odd} = 1$.

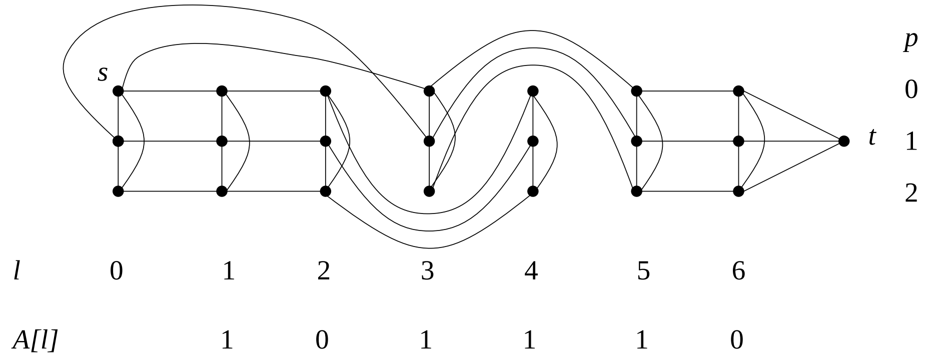

Figure 4: The graph constructed for the array 1 0 1 1 1 0 to show a lower bound for 3-vertex and 3-edge connectivity in a 2-vertex connected graph ($n = 6$). Here $f_{even} = 3$, $f_{odd} = 1$, $l_{even} = 6$, and $l_{odd} = 4$.

To answer a $Sum(l)$ operation we first add the edges $(t, (l, p))$ for all $0 \leq p \leq k-1$ and then delete all edges of the form $(t, (n, p))$. In the resulting graph vertex s and vertex t are k-edge and k-vertex connected if and only if S_l is odd. Thus we ask a $Query(s, t)$ and then restore the graph. This shows that a $Sum(l)$ operation corresponds to k edge insertions and deletions plus one k-edge or k-vertex connectivity query in the graph.

Each $Add(l)$ operation corresponds to the following at most $k + 1$ insertions and deletions of edges in the graph. Let i_{odd} be the largest vertex on the odd chain with $i_{odd} < l$ and let j_{odd} be the smallest vertex on the odd chain with $j_{odd} \geq l$. Let i_{even} and j_{even} be defined accordingly. We find i_{even} and j_{even} using Van-Emde-Boas priority queue as in the previous section. To update the graph we execute the following steps:

- We insert the edges $((i_{odd}, p), (j_{even}, p))$ and $((i_{even}, p), (j_{odd}, p))$ for $0 \leq p \leq k-1$.
- Afterwards we delete the edges $((i_{odd}, p), (j_{odd}, p))$ and $((i_{even}, p), (j_{even}, p))$ for $0 \leq p \leq k-1$.
- If f_{even} or f_{odd} changes, we update the edges to $(0, p)$ for all p appropriately. If l_{even} or l_{odd} changes, we update the edges incident to t appropriately.

Since the graph is $(k\text{-}1)$-vertex connected before and after the update and we first insert and afterwards deleted edges, the graph remains $(k\text{-}1)$-vertex connected during all insertions and deletions.

Thus, this reduces the prefix sum problem to a fully dynamic k-edge or k-vertex connectivity problem in a $(k\text{-}1)$-vertex connected graph. A sequence of m Add and Sum operations corresponds to $O(km)$ edge insertions, deletions, and connectivity queries and $O(m)$ operations in the Van-Emde-Boas priority queue. Thus, the lower bound for the prefix sum problem gives an amortized lower of $\Omega(\log n/(k(\log\log n + \log b))$ per operation. ∎

4.2 The Lower Bound in Plane Graphs

In this section we prove the following theorem.

Theorem 4.2 *In the cell probe model any algorithm for testing if two fixed nodes s and t are k-edge or k-vertex connected in a plane c-vertex connected graph under a sequence of insertions and deletions of edges requires $\Omega(\log n/k(\log\log n + \log b))$ amortized time per operation, where $b < n^{1/\log\log n}/\log n$ and $c = \min(k-1, 2)$.*

Proof: Let S_0 be 1. Given a parity prefix sum problem we construct a graph consisting of one vertex t and $k(n+1)$ vertices, labeled (l,p) with $0 \le l \le n$ and $0 \le p \le k-1$. The vertex $(0,0)$ is also labeled s. The graph contains the following edges.

- There is an edge $((l,p),(l,p+1))$ for all l and $0 \le p \le k-2$. The vertices (l,p) represent S_l for $0 \le l \le n$. If S_l is odd and l' is the largest index smaller than l such that $S_{l'}$ is odd, there is an edge $((l',p),(l,p))$ for $0 \le p \le k-1$. This creates k *odd* chains. All vertices with even S_l are connected in the same way, creating k *even* chains.
- Let f_{odd} (f_{even}) be the lowest index i larger than 0 such that S_i is odd (even) and let l_{odd} (l_{even}) be the highest index i smaller than $n+1$ such that S_i is odd (even). For all $0 \le p \le k-1$ there is an edge $((0,p),(f_{odd},p))$. Additionally, the graph contains an edge $((0,0),(f_{even},0))$ and $((0,k-1),(f_{even},k-1))$.
- The node t is connected to each node (l_{odd},p) for $0 \le p \le k-1$ by an edge. The order $(t,(l_{odd},0)),\cdots,(t,(l_{odd},k-1))$ corresponds to the counterclockwise embedding at t of these edges.
- There is an edge $((l_{even},0),(l_{even},k-1))$.
- The embedding of the edges at (l,p) is $((l,p),(l,p-1)),((l,p),(l'',p),((l,p),(l,p+1))$, and $((l,p),(l',p))$, where $l' < l < l''$ (non-existing edges omitted).

Note that this gives a plane graph. Figure 5 gives an example.

For the case that $k > 1$ we depict below each step of an operation. In these figures, the affected embedding is drawn with the even chains placed above the odd chains

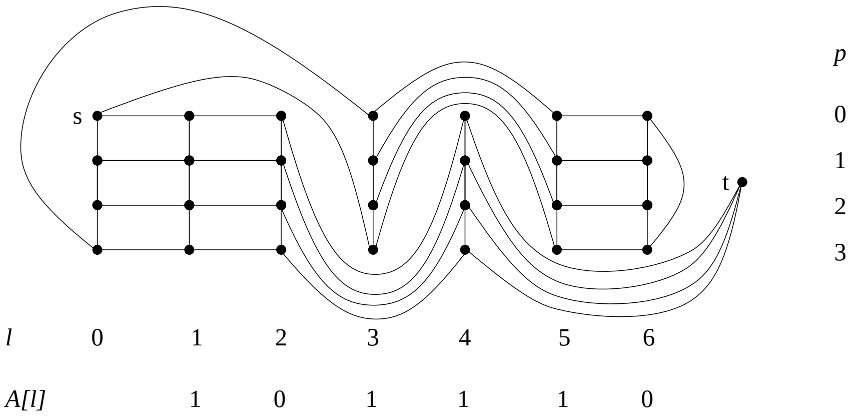

Figure 5: The plane 2-vertex connected graph constructed for the array 1 0 1 1 1 0 to show a lower bound for 4-edge and 4-vertex connectivity ($n = 6$). Here $f_{even} = 3$, $f_{odd} = 1$, $l_{even} = 6$, and $l_{odd} = 4$.

(see Figure 6). The grey areas depict parts of the graph that are not affected by the operation and thus not drawn in detail.

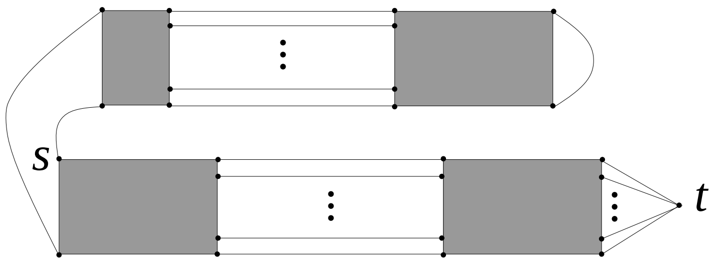

Figure 6: A planar graph and its embedding as constructed in this proof for the case $k > 1$.

As in the previous proofs, we maintain 1, n, and all indices i such that $S_i + S_{i-1}$ is odd in a Van-Emde-Boas priority queue.

Sum Queries

To answer a $Sum(l)$ query, we execute the following steps. We depict each step for the case that S_l is even. Newly inserted edges are drawn bold.

1. We insert the edges $(t, (l, 0))$ and $(t, (l, k-1))$ (see Figure 7 for even S_l).
2. We delete the edges $((l, p), (l', p))$ for all $0 \leq p \leq k - 3$, where $l' > l$ and (l', p) is the neighbor of (l, p) on the (even or odd) chain (see Figure 8 for even S_l).
3. We insert the edges $(t, (l, p))$ for all $1 \leq p \leq k - 2$ (see Figure 9 for even S_l).
4. We delete the edges $(t, (n, p))$ for all $0 \leq p \leq k - 1$ (see Figure 10 for even S_l).

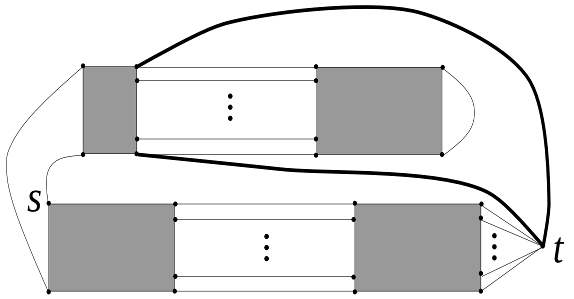

Figure 7: After Step 1.

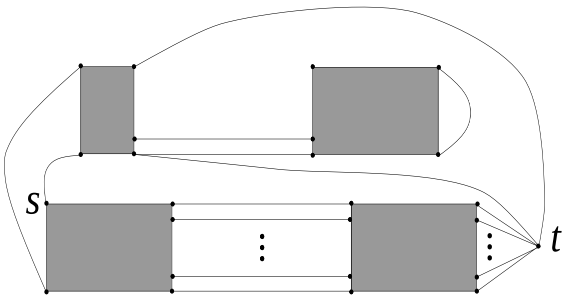

Figure 8: After Step 2.

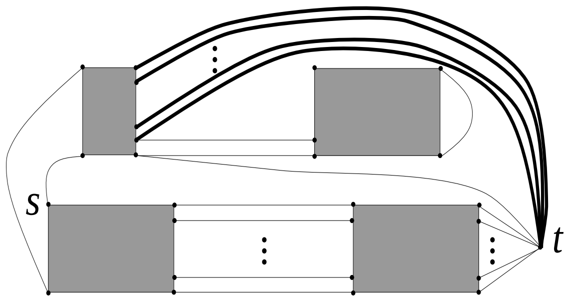

Figure 9: After Step 3.

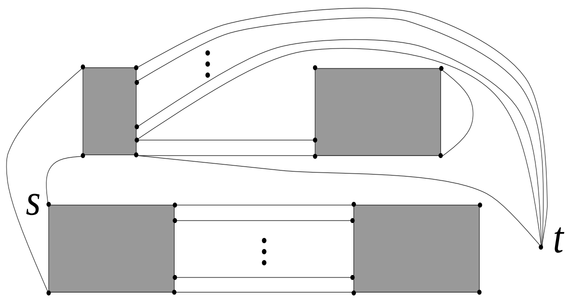

Figure 10: After Step 4.

5. In resulting graph s and t are k-edge (k-vertex) connected iff S_l is odd. Thus we test if s and t are k-edge (k-vertex) connected. Afterwards, we restored the graph.

Note that the above steps maintain the planarity of the embedding and the c-vertex connectivity of the graph. This shows that each $Sum(l)$ query requires $O(k)$ edge insertions and deletions and one k-edge (k-vertex) connectivity query in a plane, c-vertex connected graph.

Add Operations

For an $Add(l)$ operation we execute the following steps:

1. We add the edges $((i_{even}, k-1), (j_{even}, 0))$, $((i_{odd}, k-1), (j_{odd}, 0))$, and $(t, (l_{even}, k-1))$ (see Figure 11).
2. We delete the edges $((i_{even}, p), (j_{even}, p))$ for $1 \leq p \leq k-1$ and the edges $((i_{odd}, p), (j_{odd}, p))$ for $0 \leq p \leq k-2$ (see Figure 12).
3. We insert the edges $((i_{even}, p), (j_{odd}, p))$ for $1 \leq p \leq k-1$ (see Figure 13).

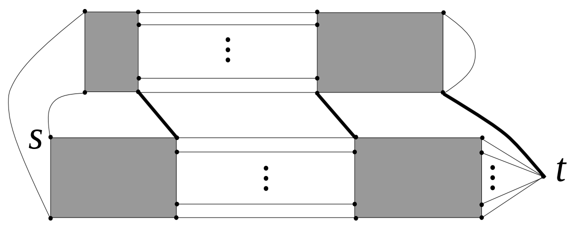

Figure 11: After Step 1.

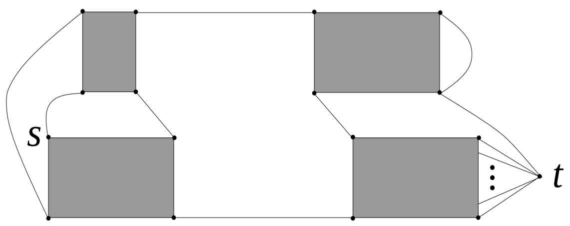

Figure 12: After Step 2.

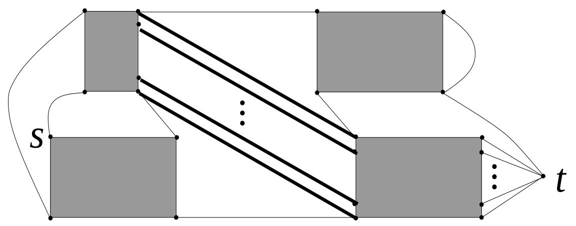

Figure 13: After Step 3.

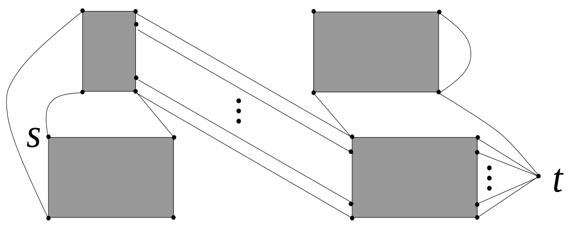

Figure 14: After Step 4.

4. We delete the edge $((i_{even}, 0), (j_{even}, 0))$ and the edge $((i_{odd}, k-1), (j_{odd}, k-1))$ see Figure 14).
5. We insert the edges $((i_{odd}, p), (j_{even}, p))$ for $1 \leq p \leq k-1$ (see Figure 15).

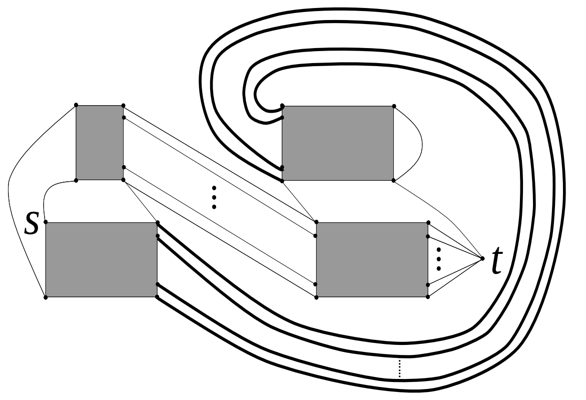

Figure 15: After Step 5.

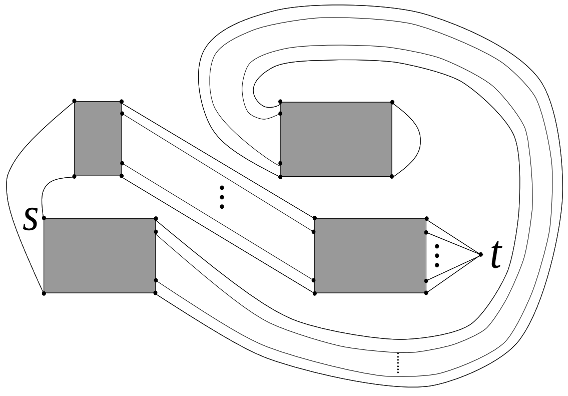

Figure 16: After Step 6.

6. We delete the edges $((i_{even}, k{-}1), (j_{even}, 0))$, $((i_{odd}, k{-}1), (j_{odd}, 0))$, and $(t, (l_{even}, k-1))$ (see Figure 16).
7. Finally we swap l_{even} and l_{odd}, delete $((l_{odd}, 0), (l_{odd}, k{-}1))$, insert $(t, (l_{odd}, 0)), \cdots, (t, (l_{odd}, k-1))$, delete $(t, (l_{even}, 0)), \cdots, (t, (l_{even}, k-1))$, and insert $((l_{even}, 0)(l_{even}, k-1))$ (see Figure 17).

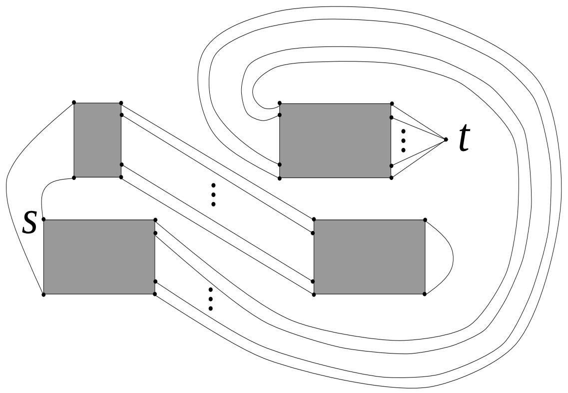

Figure 17: After Step 7.

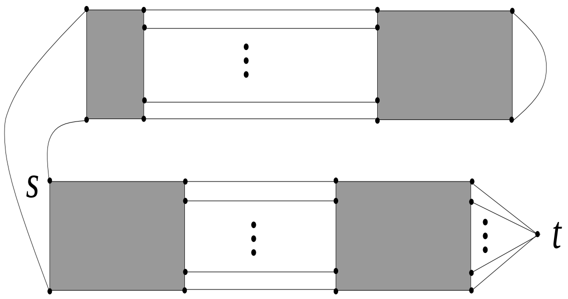

Figure 18: Final graph.

The embedding in the final graph (see Figure 17) is identical to the originally defined embedding of the graph. (see Figure 18).

The above steps maintain the planarity of the embedding and the c-vertex connectivity of the graph. This shows that each $Add(l)$ operation can be executed with $O(k)$ edge insertions and deletions in a plane, c-vertex connected graph and completes the proof of the theorem. ∎

5 Extensions

The condition that $b < n^{1/\log\log n}/\log n$ can be removed from all previous proofs by eliminating the use of the Van-Emde-Boas priority queue as follows. We define the *helpful parity prefix sum problem* to be the following modified parity prefix sum problem: Given an array $A[0], \ldots, A[n+1]$ of integers mod 2 such that initially all $A[i]$ are 0, except for $A[0]$ and $A[n+1]$ which are 1, execute the following operations in arbitrary order:

$Add(l, i, j)$: If $0 \leq i < l < j \leq n+1$, $A[i] > 0$, $A[j] > 0$, and $A[k] = 0$ for all $i < k < j$, then incremented $A[l]$ by 1. Otherwise, do nothing.

$Sum(l)$: Return S_l mod 2, where $S_l = \sum_{i=1}^{l} A[i]$.

It follows immediately from the reductions presented in the previous sections that a lower bound of $\Omega(\log n/\log\log n)$ to the helpful parity prefix sum problem removes the condition that $b < n^{1/\log\log n}/\log n$ from the previous theorems.

To show the lower bound for the helpful parity prefix sum problem we observe that the proof pertaining to the parity prefix sum problem applies to the helpful parity prefix sum problem, since it does not put any constraints on the information input of an update.

The dependence of the lower bounds on the word length can be removed for the price of assuming a more specific model of computation, namely a RAM with arithmetic instructions on integers of unbounded word size. Using the lower bounds of [1] for the parity prefix sum problem and the helpful parity prefix sum problem and the same reductions as in the previous sections we obtain a lower bound of $\Omega(\log n/\log\log n)$ if the data structure uses space that is polynomial in n and a lower bound of $\Omega(\log n/\log\log m)$ if the space is unrestricted and m operations are executed.

Acknowledgements

We thank an anonymous referee for pointing out the above extensions.

References

[1] A. M. Ben-Amram, "On the Power of Random Access Machines", Ph.D. Thesis, Tel-Aviv University, School of Mathematical Sciences, 1994.

[2] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, M. Yung, "Maintenance of a Minimum Spanning Forest in a Dynamic Planar Graph" *J.Algorithms*, 13 (1992), 33–54.

[3] D. Eppstein, Z. Galil, G. F. Italiano, A. Nissenzweig, "Sparsification - A Technique for Speeding Up Dynamic Graph Algorithms", *Proc. 33nd Annual Symp. on Foundations of Computer Science*, 1992.

[4] D. Eppstein, Z. Galil, G. F. Italiano, T. H. Spencer, "Separator Based Sparsification for Dynamic Planar Graph Algorithms", *Proc. 23nd Annual Symp. on Theory of Computing*, 1993.

[5] S. Even, "Graph Algorithms", Computer Science Press, 1979.

[6] M. L. Fredman and M. E. Saks, "The Cell Probe Complexity of Dynamic Data Structures", *Proc. 19th Annual Symp. on Theory of Computing*, 1989, 345–354.

[7] M. Rauch Henzinger and Valerie King, "Randomized Dynamic Algorithms with Polylogarithmic Time per Operation", to appear in *Proc. 27th Annual Symp. on Theory of Computing*, 1995.

[8] M. Rauch Henzinger and Han La Poutré, "Certificates and Fast Algorithms for Biconnectivity in Fully-Dynamic Graphs", submitted.

[9] J. Hershberger, M. Rauch, and S. Suri, "Fully Dynamic 2–Edge–Connectivity in Planar Graphs", *Proc. 3rd Scandinavian Workshop on Algorithm Theory*, LNCS 621, Springer-Verlag, 1992, 233–244.

[10] G. Italiano, H. La Poutré, and M. Rauch, "Fully Dynamic Planarity Testing in Embedded Graphs", in: T. Lengauer (Ed.), *Algorithms - ESA '93*, LNCS 726, Springer Verlag, 1993, pages 212-223.

[11] M. H. Rauch. "Improved Data Structures for Fully Dynamic Biconnectivity", *Proc. 26th Annual Symp. on Theory of Computing,* 1994, 686–695.

[12] P. van Emde Boas, "Preserving order in a Forest in Less Than Logarithmic Time and Linear Space", *Inform. Process. Lett.*, 6(3), 1977, 80–82.

[13] J. Westbrook, "Fast Incremental Planarity Testing", *Proc. 19th Int. Colloq. on Automata, Languages, and Programming (ICALP)*, 1992, 342–353.

[14] J. Westbrook and R E. Tarjan, "Amortized Analysis of Algorithms for Set Union with Backtracking", *SIAM J. Comput.*, 18(1), 1989, 1–11.

[15] J. Westbrook and R E. Tarjan, "Maintaining Bridge-Connected and Biconnected Components On-Line" *Algorithmica* 7 (1992), 433–464.

[16] A. Yao, "Should Tables Be Sorted", *J. Assoc. Comput. Mach.*, 28(3), 1981, 615–628.