Lower Bounds for Dynamic Connectivity Problems in Graphs

Michael L. Fredman*
Monika H. Rauch**

TR 94-1420
April 1994

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

* Department of Computer Science, Rutgers University, New Brunswick, NJ 0893.
** Department of Computer Science, Cornell University, Ithaca, NY 14853.

Lower Bounds for Dynamic Connectivity Problems in Graphs

Michael L. Fredman*
Monika H. Rauch †

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 or deletion or per 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 dynamic connectivity problems. They show that the upper bound of $O(\log n)$ or $O(\log^2 n)$ for fully dynamic connectivity, 2-connectivity, and planarity testing in plane graphs are close to optimal.

1 Introduction

This paper investigates lower bounds for dynamic data structures. Given a *plane* (=planar embedded) graph G, the *fully dynamic k-edge or k-vertex connectivity problem in plane graphs* is to execute the following operations in arbitrary order:

Insert(u,v) : Add the edge (u,v) to G if this does not destroy the planarity of the embedding.

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

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

If G is not embedded, arbitrary insertions are allowed. This problem is called *fully dynamic k-edge or k-vertex connectivity problem in general graphs.*

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

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

*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

in arbitrary order. If G is not embedded, all edge insertions are allowed and the problem is called *fully dynamic planarity testing problem in general graphs.*

The current best algorithms in plane graphs take time $O(\log n)$ [1] per operation for connectivity [1], and $O(\log^2 n)$ for 2-edge-connectivity [8], 2-vertex-connectivity [9], and planarity testing [7]. In planar graphs connectivity and 2-edge connectivity can be maintained in time $O(\log n)$ per insertion and $O(\log^2 n)$ per deletion [3]. The dynamic 2-vertex, 3-edge, 3-vertex, and 4-edge connectivity and planarity testing problem can be solved in planar graphs in time $O(\sqrt{n})$ [3].

In general graphs the best upper bounds are $O(\sqrt{n})$ per operation for the dynamic connectivity and 2-edge-connectivity problem [2] and $\min(\sqrt{m}\log n, n)$ per operation for the 2-vertex connectivity problem [9].

In this paper we establish a lower bound on the time per operation for fully-dynamic planarity testing of embedded graphs with n vertices, in the cell probe model with wordsize b. In the cell probe model of computation [12], the time complexity of a sequential computation is defined to be the number of accessed memory words. Any other operations are free. This model is powerful enough to express all algorithms for random access machines. We show that each operation takes amortized time $\Omega(\log n/(\log\log n + \log b))$, where n is the number of vertices in the graph. We also describe a similar construction providing lower bounds of $\Omega(\log n/k(\log\log n + \log b))$ on the amortized time per operation for fully dynamic k-edge connectivity and fully dynamic k-vertex connectivity in plane graphs. This shows that the above upper bounds in plane graphs are close to optimal. The lower bounds also apply to (k-1)-vertex connected graphs (and thus also to (k-1)-edge connected graphs), i.e. the "hardness" of the k-edge connectivity problem does not depend on the "hardness" of the (k-1)-edge connectivity problem. No k-edge or k-vertex connected graph with $k > 3$ is plane, but we can show that the lower bounds hold in c-vertex connected plane graphs where $c = \min(3, k-1)$. An earlier version of this work has appeared in [9].

We review next some related work. Westbrook and Tarjan [11] gave a lower bound for maintaining the connected components of a graph under a sequence of edge insertions and backtracking edge deletions. They proved that in the separable pointer machine model for any n and any m there exists a sequence of m operations whose cost is $\Omega(m\log n/\log\log n)$.

Fredman and Saks [6] gave a lower bound for the modular prefix sum problem in the cell probe model of computation. The modular prefix sum problem is defined as follows: Given an array $A[1],\ldots,A[n]$ of integers mod k with initial value zero execute the following operations in arbitrary order:

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

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

Given an algorithm in the cell probe model the proof shows that there exists a sequence of m operations such that the algorithm takes time $\Omega(m\log n/(\log\log n + \log b))$. The lower bound holds for arbitrary values of k. We make in the following section use of the lower bound for the case $k = 2$. We call this the *parity prefix sum problem.*

2 The Lower Bounds

To show the lower bound for the dynamic connectivity and planarity testing problems we reduce the parity prefix sum problem to them. In Section 2.1 we describe this reduction for the planarity testing problem and in Section 2.2 we describe it for fully-dynamic k-edge and k-vertex connectivity.

2.1 A lower bound for fully dynamic planarity testing

We are given an instance of the parity prefix sum problem, i.e. an array A of integers mod 2 and we want to solve it using an algorithm for fully-dynamic planarity testing. To represent A we construct the following graph with $n+3$ vertices which are labeled with consecutive numbers starting with 0.

- Vertex i represents S_i for $1 \leq i \leq n$.
- Vertices 0, $n+1$, and $n+2$ form a triangle whose face is not incident to any of the other vertices.
- All vertices i with odd S_i are connected by a chain (called *odd chain*) in the following way. If S_l is odd and i is the largest index smaller than l such that S_i is odd, there is an edge between vertex i and vertex l. All vertices with even S_i are connected in the same way.
- 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 order of the vertices at l_{odd}.

For an example see Figure 1.

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 inside this cycle, but vertex $n+2$ is outside. 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.

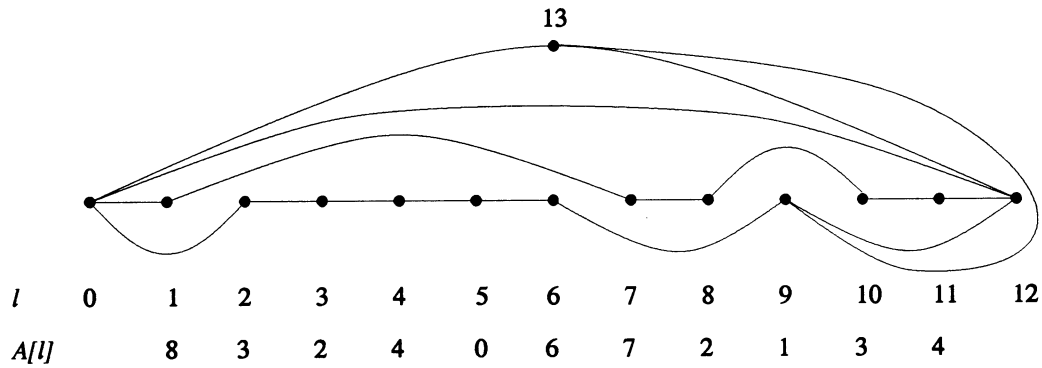

Figure 1: The plane graph for the array 8 3 2 4 0 6 7 2 1 3 4.

An *Add*(l) changes S_j for all $j \geq l$ from odd to even and vice versa. To update the graph 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}. 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.

To maintain the planarity of the embedding (and also the connectedness of the graph) during these steps we maintain pointers to l_{odd} and l_{even} and execute these steps in this 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.
5. Insert the edges (i_{odd}, j_{even}), $(l_{odd}, n+1)$ 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* [10] for finding predecessors in integer sets. Given the universe $\{0, 1, \ldots, n\}$ 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.

Each of these operations can be executed in time $O(\log\log n)$ in the cell probe model [10]. In the following we denote by $pred(x)$ (resp. $succ(x)$) the vertex 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 $succ(l+1) \leq n+1$. The priority queue can be updated with a constant number of insert and delete operations after each $Add(l)$ operation.

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

Lemma 2.1 *Let S be the value of S_l before and $Add(l)$ operation. If (i_{odd}, j_{odd}) and (i_{even}, j_{even}) are the edges that have to be deleted after the $Add(l)$ operation and (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. ∎

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

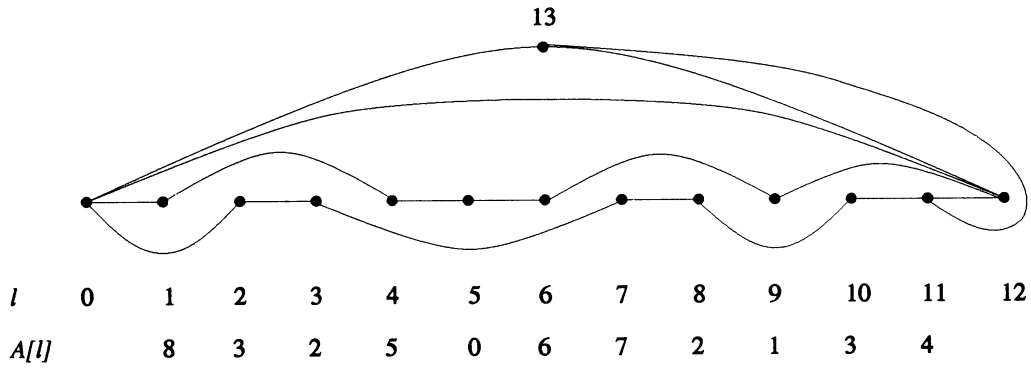

Figure 2: The plane graph of the array 8 3 2 5 0 6 7 2 1 3 4.

Fredman and Saks [6] show that for any algorithm for the parity prefix sum problem there exists a sequence of m operations such that the algorithm takes time $\Omega(m \log n/(\log\log n +$

$\log b)$) in the cell probe model. In the algorithm presented above this sequence uses $O(m)$ edge insertions, edge deletions, and planarity queries in the plane graph and $O(m)$ operations in the Van-Emde-Boas priority queue. Since each operation in the Van-Emde-Boas priority queue takes time $O(\log\log n)$ in the cell probe model, it follows that the sequence of $O(m)$ edge insertions, edge deletions, and planarity queries takes time $\Omega(m \log n/(\log\log n + \log b))$. This establishes the following bound.

Theorem 2.2 *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 where* b *is the number of bits in a word.*

2.2 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.

Theorem 2.3 *In the cell probe model any algorithm for maintaining k-edge or k-vertex connectivity 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.*

Proof: Let S_0 be 1. Given a parity prefix sum problem, we construct a graph consisting of $k(n+1)$ vertices, labeled qp with $0 \leq q \leq n$ and $1 \leq p \leq k$. For a fixed q the vertices qp are connected by a complete graph K_k. The vertices qp represent the sum S_q. If S_q is odd and q' is the largest index smaller than q such that $S_{q'}$ is odd, there is an edge $(q'p, qp)$ for all p. This creates k *odd* chains. The vertices representing even S_q 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. If $k > 1$, there is an edge $(f_{even}p, 0p)$ for $1 \leq p < k$. Note that the resulting graph is (k-1)-vertex connected. In the example of Figure 3 $f_{even} = 3$ and $f_{odd} = 1$.

Vertex $l1$ and vertex 01 are k-edge and k-vertex connected if and only if S_l is odd. Thus a $Sum(l)$ operation corresponds to 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 that is smaller than l and let j_{odd} be the smallest vertex on the odd chain $\geq l$. Let i_{even} and j_{even} be defined accordingly. To find i_{even} and j_{even} we maintain 1, n, and all indices i such that $S_{i-1} + S_i$ is odd in a Van-Emde-Boas priority queue.

To update the graph insert the edges $(i_{odd}q, j_{even}q)$ and $(i_{even}q, j_{odd}q)$ for $1 \leq q \leq k$. If $i_{odd} = 0$ or $i_{even} = 0$, then the first elements of the even and odd chains changes. We update the pointers to f_{even} and f_{odd}, insert $(f_{even}k, 0k)$, and delete $(f_{odd}k, 0k)$.

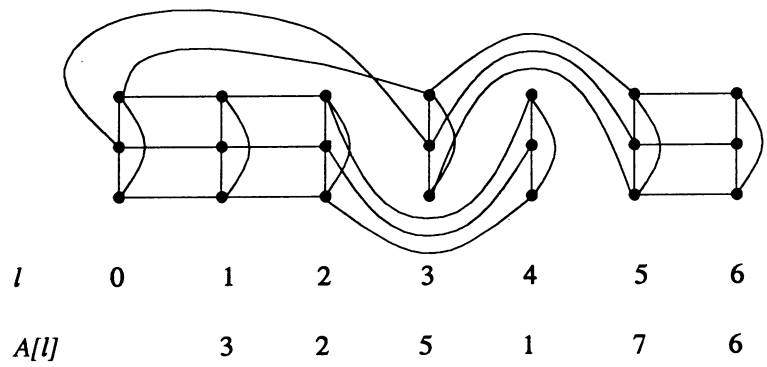

Figure 3: The graph constructed for the array 3 2 5 1 7 6 to show a lower bound for 3-vertex and 3-edge connectivity in a 2-vertex connected graph.

Afterwards delete the edges $(i_{odd}q, j_{odd}q)$ and $(i_{even}q, j_{even}q)$. Since the graph is (k-1)-vertex connected before and after the update and we first insert and afterwards deleted edges, the graph remains (k-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-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. ∎

Theorem 2.4 *In the cell probe model any algorithm for maintaining k-edge or k-vertex connectivity 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 $c = \min(k-1, 3)$.*

Proof: Let S_0 be 1. Given a parity prefix sum problem we construct a graph of $k(n+2)$ vertices, labeled qp with $0 \leq q \leq n+1$ and $1 \leq p \leq k$. There is an edge $(qp, q(p+1))$ for all q and $1 \leq p < k$. All vertices qp represent S_q for $0 \leq q \leq n$. If S_q is odd and q' is the largest index smaller than q such that $S_{q'}$ is odd, there is an edge $(q'p, qp)$ for all p. This creates k *odd* chains. All vertices with even S_q are connected in the same way, creating k *even* chains. There are no edges incident to vertices $(n+1)p$, except during a $Sum(l)$ operation.

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). We maintain pointers to these 4 vertices. There is an edge $(0(k+1-p), f_{even}p)$ for all $1 \leq p < k$. The embedding of the edges at qp is $(qp, q(p-1)), (qp, q''p), (qp, q(p+1))$, and $(qp, q'p)$, where $q'p$ is the ancestor and $q''p$ is the descendant of qp in the chain

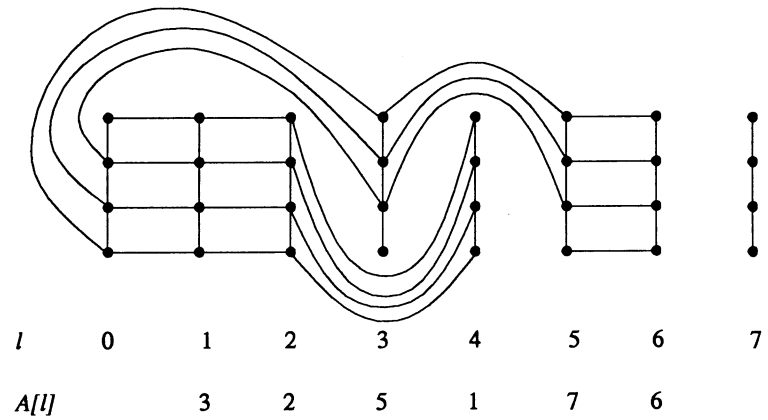

Figure 4: The plane 3-vertex connected graph constructed for the array 3 2 5 1 7 6 to show a lower bound for 4-edge and 4-vertex connectivity.

containing qp if $q > 0$ and $q' = f_{odd}$ and $q'' = f_{even}$ if $q = 0$. Note that this gives a plane graph. Figure 4 gives an example.

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

To answer a $Sum(l)$ query, we add for every p the edge $(l_{odd}p, (n+1)(k+1-p))$ and $(l_{even}p, (n+1)p)$. This guarantees that the graph stays c-vertex connected. Then we remove the edges $(lp, l'p)$ with $l' > l$ and insert $(lp, l1)$ for $p > 2$. Now $Sum(l)$ is 1 if and only if vertex $l1$ and vertex 01 are k-vertex (or k-edge) connected. Afterwards the original graph is restored. Thus each $Sum(l)$ query requires $O(k)$ edge insertions and deletions and one connectivity query in the graph.

An $Add(l)$ operation changes S_j for $j \geq l$ from even to odd and vice versa. Let $(i_{odd}p, j_{odd}p)$ for all p be the edges of the odd chains that have to be deleted, and let $(i_{even}p, j_{even}p)$ be the edges on the even chain. These edges can be found in time $O(\log\log n)$ using the Van-Emde-Boas priority queue as shown in lemma 2.1. To maintain a plane c-connected graph we update the graph with the following steps:

1. Insert the edge $(f_{even}k, 01)$.
2. If $i_{odd} > 0$, then execute the following steps in increasing order of p: Insert $(i_{even}p, j_{odd}p)$ and $(i_{odd}p, j_{even}p)$ (consistent with the embedding) and afterwards delete $(i_{odd}p, j_{odd}p)$ and $(i_{even}p, j_{even}p)$.
   If $i_{odd} = 0$, then execute the following steps in increasing order of p: Insert $(i_{even}p, 0p)$ and $(i_{odd}p, 0(k+1-p))$ (consistent with the embedding) and afterwards delete $(i_{odd}p, 0p)$ and $(i_{even}p, 0(k+1-p))$.
3. Update f_{even}, f_{odd}, l_{even}, and l_{odd}.
4. Delete $(f_{even}k, 01)$.

As in the previous proofs, m *Add* and *Sum* operations correspond to $O(km)$ operations in the dynamic data structure and also $O(m)$ operations in the Van-Emde-Boas priority queue which implies the lower bound. ∎

References

[1] 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.

[2] D. Eppstein, Z. Galil, G. F. Italiano, A. Nissenzweig, "Sparsification - A technique for speeding up dynamic graph algorithms" *Proc. 33nd Annual IEEE Symp. on Foundations of Computer Science*, 1992.

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

[4] G. N. Frederickson, "Data Structures for On-line Updating of Minimum Spanning Trees" *SIAM J. Comput.* 14 (1985), 781–798.

[5] G. N. Frederickson, "Ambivalent Data Structures for Dynamic 2-edge-connectivity and k smallest spanning trees" *Proc. 32nd Annual IEEE Symp. on Foundation of Comput. Sci.*, 1991, 632–641.

[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] 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.

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

[9] M. H. Rauch. "Improved Data Structures for Fully Dynamic Biconnectivity" to appear in *Proc. 26th Annual Symp. on Theory of Computing* 1994.

[10] 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.

[11] R. E. Tarjan, J. Westbrook, "Amortized Analysis of Algorithms for Set Union with Backtracking", *SIAM J. Comput.*, 18(1), 1989, 1–11. biconnected components on-line" *Algorithmica* 7 (1992), 433–464.

[12] A. Yao, "Should tables be sorted", *J. Assoc. Comput. Mach.*, 28(3), 1981, 615–628.