[22] D. D. Sleator, and R. E. Tarjan, "A data structure for dynamic trees", *J. Comput. System Sci.* 24 (1983), 362–381.

[23] D. D. Sleator, and R. E. Tarjan, "Self-adjusting binary search trees", *J. Assoc. Comput. Mach.* 32 (1985), 652–686.

[24] P. M. Spira, and A. Pan, "On finding and updating spanning trees and shortest paths", *SIAM J. Comput.* 4 (1975), 375–380.

[25] R. Tamassia, "A dynamic data structure for planar graph embedding", *Proc. 15th Int. Colloquium on Automata, Languages and Programming*, *Lecture Notes in Computer Science*, vol. 317, Springer-Verlag, Berlin, 1988, 576–590.

[26] R. E. Tarjan, "Depth first search and linear graph algorithms", *SIAM J. Comput.* 1 (1972), 146–160.

[27] R. E. Tarjan, and J. van Leeuwen, "Worst case analysis of set union algorithms", *J. Assoc. Comput. Mach.* 31 (1984), 245–281.

[28] R. E. Tarjan, and U. Vishkin, "An efficient parallel biconnectivity algorithm", *SIAM J. Comput.* 14 (1985), 862–864.

[29] J. Westbrook, "Algorithms and data structures for dynamic graph problems", Ph. D. Dissertation, Tech. Rep. CS-TR-229-89, Department of Computer Science, Princeton University, October 1989.

[30] J. Westbrook, and R. E. Tarjan, "Maintaining bridge-connected and biconnected components on-line", Tech. Rep. CS-TR-228-89, Department of Computer Science, Princeton University, August 1989. To appear in *Algorithmica*.

[31] D. M. Yellin, "A dynamic transitive closure algorithm", Research Report, IBM Research Division, T. J. Watson Research Center, 1988.

[7] G. Di Battista, and R. Tamassia, "On-line graph algorithms with SPQR-trees", *Proc. 17th Int. Colloquium on Automata, Languages and Programming*, *Lecture Notes in Computer Science* 443, Springer-Verlag, Berlin, 1990, 598–611.

[8] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung, "Maintenance of a minimum spanning forest in a dynamic planar graph", *Proc. First Annual ACM-SIAM Symp. on Discrete Algorithms*, 1990, 1–11.

[9] S. Even, and H. Gazit, "Updating distances in dynamic graphs", *Methods of Operations Research* 49 (1985), 371–387.

[10] S. Even, and Y. Shiloach, "An on-line edge deletion problem", *J. Assoc. Comput. Mach.* 28 (1981), 1–4.

[11] G. N. Frederickson, "Data structures for on-line updating of minimum spanning trees", *SIAM J. Comput.* 14 (1985), 781–798.

[12] Z. Galil, and G. F. Italiano, "Fully dynamic algorithms for edge connectivity problems", *Proc. 23rd Annual ACM Symp. on Theory of Computing*, 1991.

[13] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.

[14] D. Harel, "On-line maintenance of the connected components of dynamic graphs", Unpublished manuscript, 1982.

[15] T. Ibaraki, and N. Katoh, "On-line computation of transitive closure for graphs", *Inform. Process. Lett.* 16 (1983), 95–97.

[16] G. F. Italiano, "Amortized efficiency of a path retrieval data structure", *Theoret. Comput. Sci.* 48 (1986), 273–281.

[17] G. F. Italiano, "Finding paths and deleting edges in directed acyclic graphs", *Inform. Process. Lett.* 28 (1988), 5–11.

[18] J. A. La Poutré, and J. van Leeuwen, "Maintenance of transitive closure and transitive reduction of graphs", *Proc. Workshop on Graph-Theoretic Concepts in Computer Science*, *Lecture Notes in Computer Science*, vol. 314, Springer-Verlag, Berlin, 1988, 106–120.

[19] F. P. Preparata, and R. Tamassia, "Fully dynamic techniques for point location and transitive closure in planar structures", *Proc. 29th Annual Symp. on Foundations of Computer Science*, 1988, 558–567.

[20] J. H. Reif, "A topological approach to dynamic graph connectivity", *Inform. Process. Lett.* 25 (1987), 65–70.

[21] H. Rohnert, "A dynamization of the all pairs least cost path problem", *Proc. 2nd Annual Symp. on Theoretical Aspects of Computer Science*, *Lecture Notes in Computer Science*, vol. 182, Springer-Verlag, Berlin, 1985, 279–286.

5 Concluding Remarks

We have presented a new data structure that is able to maintain the biconnected components of a planar graphs under insertions and deletions of edges. Each operation is supported in $O(n^{2/3})$ time in the worst case, where n is the total number of vertices in the graph.

Our data structure is also able to return an articulation point separating two given vertices x and y (if one exists) in the same $O(n^{2/3})$ time bound. Furthermore, insertions of new vertices and deletions of disconnected vertices can be supported in the same time bound in the worst case. However, the choice of the integer k depends now on the number n of current vertices, which may change due to insertions and deletions of disconnected vertices. We circumvent this by recomputing the topological partition of G every time n gets either twice larger or twice smaller than before. Since recomputing the topological partition as well as the full and the abridged representations of the new clusters can be done in linear time, this can be amortized against the number of vertices inserted and deleted after the last time we recomputed the topological partition. As a result, the times for our operations become amortized. Other standard techniques (see for instance [12]) allow us to make these times worst-case.

We remark that our algorithm does not need to recompute the possible change of planar embedding induced by the insertion of new edges. Indeed, we do not even know of any algorithm that checks in $o(n)$ time per operation whether a graph remains planar during sequences of both edge insertions and deletions.

Finally we observe that the $O(n^{2/3})$ bound given in Theorem 4.1 applies to a larger family of graphs, that is sparse graphs whose sparseness is preserved under contractions of edges. Unlike planarity, this property can be easily checked throughout any sequence of operations.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] G. Ausiello, G. F. Italiano, A. Marchetti-Spaccamela, U. Nanni, "Incremental algorithms for minimal length paths", *J. Algorithms*, to appear.

[3] G. Ausiello, A. Marchetti-Spaccamela, and U. Nanni, "Dynamic maintenance of paths and path expressions in graphs", *Proc. Int. Symp. on Symbolic and Algebraic Computation*, *Lecture Notes in Computer Science* 358, Springer-Verlag, Berlin, 1989, 1–12.

[4] A. L. Buchsbaum, P. C. Kanellakis, and J. S. Vitter, "A data structure for arc insertion and regular path finding", *Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms*, 1990, 22–31.

[5] F. Chin, and D. Houk, "Algorithms for updating minimum spanning trees", *J. Comput. System. Sci.* 16 (1978), 333–344.

[6] G. Di Battista, and R. Tamassia, "Incremental planarity testing", *Proc. 30th Annual Symp. on Foundations of Computer Science*, 1989, 436–441.

time required to determine whatever splits and merges are needed as well as to perform them along with the re-organization of the full and abridged representation of the clusters involved is $O(k^2)$.

If $\mathcal{C}_{x'} \neq \mathcal{C}_{y'}$, then we have to recompute the full and abridged representations of the two clusters. As said before, this can be done in $O(k)$ time. If $\mathcal{C}_{x'} = \mathcal{C}_{y'} = \mathcal{C}$, then we have the following two cases. If the deletion of (x, y) leaves the cluster $\mathcal{C}$ connected, then we have the compute full and abridged representation of cluster $\mathcal{C}$. This can be done in $O(k)$ time by lemma 2.1. Otherwise, the deletion of (x, y) disconnects $\mathcal{C}$ and $\mathcal{C}$ can no longer be a cluster. Denote by $\mathcal{C}'$ and $\mathcal{C}''$ the two connected components into which the deletion of (x, y) splits cluster $\mathcal{C}$. As shown before, this splitting may be performed in $O(k^2)$ time. If either $\mathcal{C}'$ or $\mathcal{C}''$ has size less than k, then we merge it with an adjacent cluster. If this adjacent cluster has now more than $3k - 2$ vertices, it must be split in two clusters. It follows that the total time required to determine whatever splits and merges are needed as well as to perform them along with the re-organization of the full and abridged representations of the clusters involved in a *Delete* operation is $O(k^2)$.

Theorem 3.1 *Our data structure supports each* Insert, Delete *and* SameBlock *operation in* $O(n^{2/3})$ *worst-case time on a connected planar graph, where* n *is the number of vertices in the graph.*

Proof: As shown before, each *Insert* and *Delete* operation can be performed in $O(k^2)$ time, while *SameBlock* requires $O(k + \frac{n}{k})$ time. Choosing $k = \left\lceil n^{1/3} \right\rceil$ gives the claimed bounds. □

4 Dealing with Unconnected Planar Graphs

We show how to deal with unconnected planar graphs within the same time bounds. Let G be an unconnected planar graph. If G consists of $q \geq 2$ connected components, we augment G by inserting $q - 1$ *dummy* edges which make it connected, such that there is no cycle containing two dummy edges. Notice that the size of the augmented graph is still $O(n)$. Furthermore, this transformation does not change the biconnected components of G, and two vertices x and y are in the same biconnected component in the augmented graph if and only if they are in the same biconnected component in the original graph. Furthermore, we assign cost 1 to each dashed edge of G, cost 2 to each solid edge and cost 3 to each dummy edge, and we maintain a topological partition of a minimum spanning tree of the augmented graph.

To insert an edge $e = (x, y)$, we first check whether there is already a dummy edge between x and y. If so, we decrease its cost from 3 to 2. Otherwise, we insert the edge (x, y) of cost 2 and update the minimum spanning tree in $O(\sqrt{n}\,)$ time using the algorithm by Frederickson [11]. If (x, y) enters the minimum spanning tree T and a dummy edge e' leaves T, we delete e'. This can be accomplished within the $O(n^{2/3})$ time bound.

To delete an edge $e = (x, y)$, we first check whether e is in the minimum spanning tree T. If e is not in T, we simply delete e. If e is in T, we check by using the algorithm by Frederickson [11] whether there is a replacement edge for e. If there is such a replacement, we simply delete e. Otherwise, removing e will disconnected the augmented graph and therefore we make e dummy by increasing its cost from 2 to 3. Once again, this does not change the $O(n^{2/3})$ time bound.

This gives us the following theorem:

Theorem 4.1 *Our data structure supports each* Insert, Delete *and* SameBlock *operation in* $O(n^{2/3})$ *time in the worst case on unconnected planar graphs, where* n *is the total number of vertices.*

graph induced by $\{x, y\}$. Replacing $\mathcal{C}_{x'}$ and $\mathcal{C}_{y'}$ with their full representation requires $O(k)$ time and replacing the remaining clusters with their abridged representations requires a total of $O(\frac{n}{k})$ time because of Lemma 2.6. Once $\mathcal{G}_{x,y}(G)$ has been computed in $O(k + \frac{n}{k})$ time, we check whether there is a square node that is an articulation point separating x and y in $\mathcal{G}_{x,y}(G)$. If at least one such articulation point is found, we return *false*. Otherwise we return *true*. This gives us a correct answer because of Lemma 2.5. By Lemma 2.4, it can be accomplished in a total of $O(k + \frac{n}{k})$ time by using an off-line algorithm for finding the biconnected components of $\mathcal{G}_{x,y}(G)$ [26].

Insert(x, y)

When the new edge (x, y) is inserted, then the degrees of x and y in the original graph increase by one. If either degree becomes four, then the transformation given earlier has to be applied. If the degree becomes larger than four then the transformation has been already applied but now must be updated. In both cases, this introduces at most a constant number of extra vertices and edges and can be easily handled. We now distinguish two cases depending on whether (x, y) is an internal edge or an inter-cluster edge. As usual we denote by $\mathcal{C}_{x'}$ and $\mathcal{C}_{y'}$ the clusters containing x' and y' respectively.

If $\mathcal{C}_{x'} = \mathcal{C}_{y'} = \mathcal{C}$, then (x, y) is an internal edge. As a result, the insertion of (x, y) changes the interior of $\mathcal{C}$, that is the full and the abridged representation of $\mathcal{C}$. The change in the full representation is simply the new edge (x, y). On the other hand, the new abridged representation of $\mathcal{C}$ can be computed in $O(k)$ time as shown in Lemma 2.1. However, because of the insertion of extra vertices to keep a degree no greater than three, it may happen that the size of $\mathcal{C}$ has now become greater than $3k - 2$. In this case, the cluster $\mathcal{C}$ is split into two clusters $\mathcal{C}'$ and $\mathcal{C}''$. Since $\mathcal{C}$ has size $O(k)$, this splitting may be performed in $O(k)$ time. Furthermore, the full and abridged representations of the new clusters $\mathcal{C}'$ and $\mathcal{C}''$ can be computed in $O(k)$. However, because of this splitting, the full and abridged representations of all the clusters previously adjacent to $\mathcal{C}$ might change. Since there are at most $O(k)$ edges incident to a cluster, at most $O(k)$ clusters can be adjacent to $\mathcal{C}$. Consequently, the full and abridged representations of the clusters that were previously adjacent to $\mathcal{C}$ can be computed in a total of $O(k^2)$ time.

If $\mathcal{C}_{x'} \neq \mathcal{C}_{y'}$, then we have to insert an inter-cluster edge (x, y). Besides the possible updates due to the transformation to keep the vertex degrees no greater than three, we have now to recompute the full and abridged representations of clusters $\mathcal{C}_{x'}$ and $\mathcal{C}_{y'}$. By Lemma 2.1, this can be done in $O(k)$ time. If the edge (x, y) caused any cluster size to exceed $3k - 2$, then we split the cluster as described above.

The total time required to perform an *Insert* operation is therefore $O(k^2)$.

Delete(x, y)

Let (x', y') be the edge of G' corresponding to edge (x, y) in G. Again we denote by $\mathcal{C}_{x'}$ and $\mathcal{C}_{y'}$ the clusters containing respectively x' and y'. When the edge (x, y) is being deleted, we may have to apply the reverse of the transformation used to keep the vertex degrees no greater than three. This can be easily taken care of, since it deletes only a constant number of vertices and edges and it involves only a constant number of clusters. If because of this either $\mathcal{C}_{x'}$ or $\mathcal{C}_{y'}$ has size less than k, then it is merged with an adjacent cluster. If this adjacent cluster has now more than $3k - 2$ vertices, it must be split in two clusters. Merging two clusters $\mathcal{C}'$ and $\mathcal{C}''$ into one requires updating the full and abridged representations of all the clusters previously adjacent to $\mathcal{C}'$ and $\mathcal{C}''$. As shown before for the case of cluster splits, this can be done in $O(k^2)$ time. Therefore the total

(T1) Biconnected subgraph replacement: Given a biconnected subgraph $\widetilde{G}$ of G, remove all the edges in $\widetilde{G}$ and introduce a round node ρ. Connect to ρ all the vertices in $\widetilde{G}$.

If there is an articulation point separating x and y in G, then (T1) preserves this articulation point. It is possible to re-obtain G from $\widehat{G}$ by means of the inverse transformation of (T1), which again preserve articulation points. This proves (i).

To prove (ii) we notice that $\mathcal{G}_{x,y}(G)$ can be obtained from $\widehat{G}$ by means of a sequence of the following two transformations.

(T2) Leaf elimination: Given a (round or square) node v ($v \neq x$ and $v \neq y$) of degree one, delete v together with its incident edge.

(T3) Round node contraction: Given a round node ρ of degree two that is not adjacent to x and to y, denote by u and v the two square nodes adjacent to ρ. Delete round node ρ and merge u and v into a new square node (i.e., all the edges previously incident to u and v are now incident to the new node).

Indeed transformations (T2) and (T3) correspond to the operations performed while computing the abridged representation of a cluster $\mathcal{C}$ starting from its full representation: (T2) corresponds to rule (R1), and (T3) to rule (R2). If there is a square node that is an articulation point separating x and y in $\widehat{G}$, then transformations (T2) and (T3) preserve at least one square node that is an articulation point separating x and y. Indeed during (T2) v cannot be an articulation point separating x and y, while in (T3) if u is an articulation point separating x and y, also v must be an articulation point separating x and y and therefore the new square node will be an articulation point separating x and y too. Similarly, the inverse transformations of (T2) and (T3) preserve articulation points separating x and y from $\mathcal{G}_{x,y}(G)$ to $\widehat{G}$. □

We now describe our data structure. During any sequence of operations, we maintain a topological partition of order k of the underlying planar graph. That is, we partition the planar graph into $O(\frac{n}{k})$ clusters such that each cluster contains a connected subgraph having between k and $3k-2$ vertices. Furthermore, we maintain for each cluster in the partition both its full and abridged representation. The following lemma is immediate.

Lemma 2.6 *Any cluster $\mathcal{C}$ can be replaced with its full representation in $O(k)$ time, and with its abridged representation in $O(d(\mathcal{C}))$ time.*

3 Implementing the Operations

We now show how to perform *SameBlock*(x,y), *Insert*(x,y), and *Delete*(x,y) operations. As explained earlier, each vertex of G corresponds to vertices of G' that can possibly be in different clusters. In order to execute any operation involving vertices x and y in G, we arbitrarily choose in G' vertices x' and y', corresponding to x and y respectively. This arbitrary choice is possible because all the vertices x' of G' corresponding to vertex x in G are merged into the same vertex during the contraction of dashed edges in the abridged graph.

SameBlock(x,y)

In order to test whether two vertices belong to the same biconnected component, we first locate the two cluster $\mathcal{C}_{x'}$ and $\mathcal{C}_{y'}$ containing respectively x' and y'. We then compute $\mathcal{G}_{x,y}(G)$, the abridged

Given any vertex x' in G', let us denote by $\mathcal{C}_{x'}$ the cluster containing x'. Notice that, because of the transformation from G to G', a vertex of G may have the corresponding vertices in G' in different clusters. We define a graph $\mathcal{G}_{x,y}(G)$, referred to as the *abridged graph induced by the pair* $\{x, y\}$, by arbitrarily choosing in G' vertices x' and y', corresponding to x and y respectively, and by replacing $\mathcal{C}_{x'}$ and $\mathcal{C}_{y'}$ with their full representations, and each other non-singleton cluster with its abridged representation. The rules for combining full and abridged representations as well as two full representations through solid and dashed edges are the same as described above. Notice that all the dashed edges of G' have been contracted in $\mathcal{G}_{x,y}(G)$. Therefore, a vertex of G corresponds to at most one square node in $\mathcal{G}_{x,y}(G)$. In particular, there is exactly one square node x and one square node y in $\mathcal{G}_{x,y}(G)$. We now characterize some properties of $\mathcal{G}_{x,y}(G)$.

Lemma 2.4 *Given any two vertices x and y, $\mathcal{G}_{x,y}(G)$, the abridged graph induced by the pair $\{x, y\}$, has at most $O(k + \frac{n}{k})$ vertices and edges and can be computed in $O(k + \frac{n}{k})$ time.*

Proof: The size of $\mathcal{G}_{x,y}(G)$ is at most $O(k + \frac{n}{k})$ because of both Lemma 2.3 and the fact that the full representation of each cluster has at most $O(k)$ vertices and edges.

The time required to compute the abridged graph induced by x and y can be computed as follows. Since the abridged representation of clusters is available in the adjacency list form, combining the abridged representation of two adjacent clusters $\mathcal{C}'$ and $\mathcal{C}''$ through either a solid or a dashed edge can be done in constant time. As a consequence, the total time spent while combining the abridged representations of adjacent clusters is

$$\sum_{\mathcal{C}} d(\mathcal{C}) = O\left(\frac{n}{k}\right).$$

Since combining the full representation of clusters $\mathcal{C}_{x'}$ and $\mathcal{C}_{y'}$ with the rest of the achieved abridged graph involves examining at most $O(k)$ solid and dashed edges, the total time required to compute $\mathcal{G}_{x,y}(G)$ becomes $O(k + \frac{n}{k})$. □

The abridged graph induced by the pair $\{x, y\}$ satisfies nice properties with respect to the biconnected components of G, as the following lemma shows.

Lemma 2.5 *There is an articulation point separating x and y in G if and only if there is at least one square node that is an articulation point separating x and y in $\mathcal{G}_{x,y}(G)$.*

Proof: Define $\widehat{G}$ to be the graph obtained from G by replacing each cluster with its full representation and by merging adjacent clusters through solid and dashed edges as explained earlier. Notice that there are two square nodes x and y in both $\widehat{G}$ and $\mathcal{G}_{x,y}(G)$. We first show that

(i) there is an articulation point separating x and y in G if and only if there is a square node that is an articulation point separating x and y in $\widehat{G}$.

We then complete the proof by showing that

(ii) there is square node that is an articulation point separating x and y in $\widehat{G}$ if and only if there is at least one square node that is an articulation point separating x and y in $\mathcal{G}_{x,y}(G)$.

$\widehat{G}$ can be obtained from G by first making all the vertices of G square nodes and then by repeatedly applying the following transformation.

Proof: First notice that since $\mathcal{C}'$ and $\mathcal{C}''$ are non-singleton clusters, ρ' and ρ'' are always defined. Furthermore, since each cluster is connected, cluster node $\mathcal{C}'$ cannot be an articulation point in $\mathcal{G}'(\mathcal{C}'')$. The same holds for cluster node $\mathcal{C}''$ in $\mathcal{G}'(\mathcal{C}')$. Consequently, ρ' and ρ'' are unique. Let e be an inter-cluster edge whose endpoints are in super-blocks ρ' and ρ'' and assume by contradiction that there is another inter-cluster edge $e' \neq e$ whose endpoints are in ρ_1 and ρ_2, with ρ_1 and ρ_2 being super-blocks in $\mathcal{C}'$ and $\mathcal{C}''$ respectively, and either $\rho_1 \neq \rho'$ or $\rho_2 \neq \rho''$. Since the clusters $\mathcal{C}'$ and $\mathcal{C}''$ are connected, then there is a simple cycle containing both e and e'. As a result, these two edges are in a same biconnected component of both $\mathcal{G}'(\mathcal{C}')$ and $\mathcal{G}'(\mathcal{C}'')$. Therefore, we must have $\rho_1 = \rho'$ and $\rho_2 = \rho''$. □

The abridged representations of all clusters can be combined together to define a new graph $\mathcal{G}(G)$, referred to as the *abridged graph* of G, as follows. For each cluster $\mathcal{C}$, $\mathcal{G}(G)$ contains the vertices and edges of the abridged representation which are internal to $\mathcal{C}$. We handle the inter-cluster edges as follows. Consider any two adjacent non-singleton clusters $\mathcal{C}'$ and $\mathcal{C}''$: we combine them as follows. If there is a cluster node $\mathcal{C}''$ in the abridged representation of $\mathcal{C}'$ (and consequently a cluster node $\mathcal{C}'$ in the abridged representation of $\mathcal{C}''$), as in Lemma 2.2 denote by ρ' (respectively ρ'') the super-block of $\mathcal{C}'$ (respectively $\mathcal{C}''$) containing cluster node $\mathcal{C}''$ (respectively $\mathcal{C}'$). Then we combine the two abridged representations by merging ρ' and ρ'' into one new round node ρ. The edges previously incident to ρ' and ρ'' are now incident to ρ. Furthermore, the two cluster nodes $\mathcal{C}'$ and $\mathcal{C}''$ are deleted. Lemma 2.2 justifies the correctness of this operation. We refer to this operation as combining non-singleton clusters $\mathcal{C}'$ and $\mathcal{C}''$ *through a solid edge* (see Figure 4a).

Finally, if there is a dashed edge e between $\mathcal{C}'$ and $\mathcal{C}''$, then there is a dashed vertex e in both the abridged representations of $\mathcal{C}'$ and $\mathcal{C}''$. In this case, let us denote by v' and v'' the two square nodes in the abridged representations of $\mathcal{C}'$ and $\mathcal{C}''$ at distance two from e. We delete the dashed vertices e as well as their adjacent round nodes in the abridged representations of $\mathcal{C}'$ and $\mathcal{C}''$. The two abridged representations are now combined by merging v' and v'' into one new square node v, which is adjacent to the previous nodes v' and v'' were adjacent to. We refer to this operation as combining non-singleton clusters $\mathcal{C}'$ and $\mathcal{C}''$ *through a dashed edge* (see Figure 4b).

[Figure 4]

By applying these operations to all the non-singleton clusters, we obtain the *abridged graph* $\mathcal{G}(G)$. Figure 5 shows the abridged graph of a given graph.

[Figure 5]

Lemma 2.3 *The abridged graph $\mathcal{G}(G)$ of a graph G has at most $O(\frac{n}{k})$ vertices and edges.*

Proof: As usual, let $d(\mathcal{C})$ denote the number of different clusters adjacent to cluster $\mathcal{C}$. By Lemma 2.1, the abridged representation of $\mathcal{C}$ contains $O(d(\mathcal{C}))$ vertices and edges. Since combining the abridged representations of adjacent clusters does not introduce new vertices and edges, the total number of vertices and edges in $\mathcal{G}(G)$ can be at most

$$\sum_{\mathcal{C}} O(d(\mathcal{C})) = O\left(\frac{n}{k}\right).$$

□

dashed vertex, for each dashed vertex v in the full representation of $\mathcal{C}$ there is exactly one square node at distance two from v. We compute a succinct representation $\mathcal{T}'$ of the block tree $\mathcal{T}$ by applying any sequence of the following two rules:

(R1) delete any non-terminal leaf (either square or round node);

(R2) if there is any degree-two round node ρ whose two adjacent square nodes, say u and v, are both non-terminal, then delete round node ρ and merge u and v into a new square node (i.e., all the edges previously incident to u and v are now incident to the new node).

The resulting tree $\mathcal{T}'$ is called the *abridged representation* of cluster $\mathcal{C}$. The round nodes of $\mathcal{T}'$ are called *super-blocks of* $\mathcal{C}$. Figure 3 shows a cluster together with its full and abridged representations.

[Figure 3]

Consider a cluster $\mathcal{C}$ which is adjacent to only one other cluster $\mathcal{C}'$ by means of solid edges. In this case the abridged representation of $\mathcal{C}$ degenerates into only one square node (corresponding to cluster node $\mathcal{C}'$), and $\mathcal{C}$ is said to be a *singleton* cluster.

For any cluster $\mathcal{C}$ let us denote by $d(\mathcal{C})$ the number of different clusters adjacent to $\mathcal{C}$. We refer to $d(\mathcal{C})$ also as the *external degree* of cluster $\mathcal{C}$. The following lemmas characterize the abridged representation of a cluster.

Lemma 2.1 *The abridged representation of a cluster $\mathcal{C}$ has $O(d(\mathcal{C}))$ vertices and edges and can be computed in $O(k)$ time starting from the full representation of $\mathcal{C}$.*

Proof: Since there are at most $O(d(\mathcal{C}))$ cluster nodes and dashed vertices in the abridged representation of $\mathcal{G}'(\mathcal{C})$, the number of terminal vertices in $\mathcal{T}$ is at most $O(d(\mathcal{C}))$.

We now show that the size of the abridged representation of cluster $\mathcal{C}$ (i.e., the size of the compressed tree $\mathcal{T}'$) is $O(d(\mathcal{C}))$. The rules used to obtain $\mathcal{T}'$ from $\mathcal{T}$ are (R1) deleting a non-terminal leaf, and (R2) contracting a degree two round node adjacent to non-terminal square nodes. Because of rule (R1) all the leaves of $\mathcal{T}'$ are terminal square nodes. Because of rule (R2) there cannot be degree two round nodes adjacent to two non-terminal square nodes in $\mathcal{T}'$. Consequently, the number of round nodes in $\mathcal{T}'$ is no greater than the number of terminal square nodes. This implies that $|\mathcal{T}'|$ is linear in the number of terminal nodes in $\mathcal{T}$, and therefore $|\mathcal{T}'| = O(d(\mathcal{C}))$.

We now prove that the time required to compute the abridged representation of a cluster $\mathcal{C}$ is $O(k)$. Indeed the abridged representation of cluster $\mathcal{C}$ is computed starting from $\mathcal{T}$, which is of size $O(k)$. Finding the terminal and non-terminal nodes in $\mathcal{T}$ can be done in $O(k)$ by simply visiting $\mathcal{T}$. By representing $\mathcal{T}$ by means of adjacency lists (such a representation can be constructed in $O(k)$ time), any sequence of the two rules used to obtain $\mathcal{T}'$ can be implemented in a total of $O(k)$ time. □

Lemma 2.2 *Let $\mathcal{C}'$ and $\mathcal{C}''$ be any two adjacent non-singleton clusters in the topological partition. Let ρ' be the super-block in the abridged representation of $\mathcal{C}'$ containing the cluster node $\mathcal{C}''$ (i.e., ρ' is the round node adjacent to cluster node $\mathcal{C}''$ in the abridged representation of $\mathcal{C}'$). Similarly, let ρ'' be the super-block in the abridged representation of $\mathcal{C}''$ containing the cluster node $\mathcal{C}'$. Then all the inter-cluster edges of the graph G' between $\mathcal{C}'$ and $\mathcal{C}''$ are incident to vertices belonging to the super-blocks ρ' and ρ''.*

edge (u_i, v_i), $0 \leq i \leq d-1$. Furthermore, we add new *dashed* edges $(u_i, u_{(i+1)})$, $0 \leq i \leq d-2$. We associate costs with edges in G' as follows: solid edges have cost 2 and dashed edges have cost 1. Using this transformation, each edge (x, y) in G corresponds to a unique solid edge (x_i, y_j) in G' and vice versa. However, this property does not hold for dashed edges. Another property of this transformation is that all the dashed edges are contained in any minimum spanning tree of G'. It is easy to show that given a graph G with m edges and n vertices, G' contains at most $O(m+n)$ edges and vertices and can be generated in $O(m+n)$ time. Furthermore if G is planar, $m = O(n)$ and G' is planar.

Let k be a positive integer to be chosen later. We partition the graph G' into vertex clusters having between k and $3k-2$ vertices, by using a topological partition of order k of a minimum spanning tree T of G', as defined in [11]. We recall that a topological partition of T of order k consists of $O(\frac{n}{k})$ vertex clusters of T such that each cluster contains a connected subgraph with between k and $3k-2$ vertices. As shown in [11], this can be accomplished in $O(n)$ time.

Since all the dashed edges will be in T and the intersection of any cluster with T is a tree, there can be be at most $O(\frac{n}{k})$ inter-cluster dashed edges. An edge of G' with both endpoints in a cluster $\mathcal{C}$ is said to be *internal* to $\mathcal{C}$, while an edge of G' with only one endpoint in $\mathcal{C}$ is said to be *incident* to $\mathcal{C}$. For each cluster $\mathcal{C}$, the *subgraph of G' induced by $\mathcal{C}$* is called $\mathcal{G}(\mathcal{C})$ and is defined as follows. $\mathcal{G}(\mathcal{C})$ has one vertex for each cluster in the topological partition adjacent to $\mathcal{C}$ (referred to as a *cluster node*) plus all the vertices in $\mathcal{C}$. The edge set of $\mathcal{G}(\mathcal{C})$ is composed of all the edges of G internal and incident to cluster $\mathcal{C}$. Since $\mathcal{C}$ contains at most $3k-2$ vertices, all of degree no more than three, then $\mathcal{G}(\mathcal{C})$ has at most $O(k)$ vertices and edges. For each cluster $\mathcal{C}$ in the topological partition of T, we maintain a *full representation* and an *abridged representation*, defined as follows.

The *full representation* of a cluster $\mathcal{C}$ is computed as follows. Starting from $\mathcal{G}(\mathcal{C})$, we contract the dashed edges internal to cluster $\mathcal{C}$. As for dashed edges incident to cluster $\mathcal{C}$, in $\mathcal{G}(\mathcal{C})$ they are of the form $(\hat{x}, \mathcal{C}')$, with $\hat{x}$ being a vertex internal to $\mathcal{C}$ corresponding to a vertex x in the original graph, and $\mathcal{C}'$ being a cluster different from $\mathcal{C}$. For each such an edge $e = (\hat{x}, \mathcal{C}')$ we introduce in its place a new special vertex e, referred to as a *dashed vertex*, and an edge $(\hat{x}, e)$. The resulting graph, which we call $\mathcal{G}'(\mathcal{C})$, contains at most $O(k)$ vertices and edges and can be computed in $O(k)$ time. Figure 2 shows a graph $\mathcal{G}'(\mathcal{C})$ obtained after these operations.

[Figure 2]

We then compute the biconnected components of $\mathcal{G}'(\mathcal{C})$, and subsequently construct the block tree $\mathcal{T}$ of $\mathcal{G}'(\mathcal{C})$ (as defined for instance in [13]). We recall here that a block tree of a graph is composed of square nodes (corresponding to vertices in the graph) and round nodes (corresponding to blocks in the graph): whenever a vertex v belongs to a given block, there is a tree edge between the corresponding square and round nodes in the block tree. As a consequence, each path in a block tree alternates between square and round nodes. The construction of the block tree is immediate once the biconnected components of $\mathcal{G}'(\mathcal{C})$ have been found. We define the block tree $\mathcal{T}$ to be the *full representation* of cluster $\mathcal{C}$. Since the size of $\mathcal{G}'(\mathcal{C})$ is $O(k)$, the full representation of $\mathcal{C}$ can be computed in a total of $O(k)$ time.

We define the *abridged representation of a cluster* $\mathcal{C}$ starting from its full representation (block tree) $\mathcal{T}$ as follows. We define the cluster nodes, the dashed vertices and the square nodes at distance two from dashed vertices to be *terminal nodes* in $\mathcal{T}$. All the remaining nodes of $\mathcal{T}$ are defined *non-terminal*. With this definition, round nodes are always non-terminal. Notice that by definition of

is the first sublinear-time algorithm known for this problem, and the case of general (non-planar) graphs remains open. The case of planar graphs, however, seems to be important because a fully dynamic algorithm for maintaining the biconnected components constitutes a first step towards the study of fully dynamic planarity testing.

The gap between the times required to solve the partially and the fully dynamic version of these problems is not surprising. Indeed fully dynamic graph problems are usually much more difficult to solve than the corresponding partially dynamic problems. For example, the partially dynamic maintenance of the connected components of an undirected graph during edge insertions only can be done in $O(\alpha(m,n))$ amortized time per operation using set-union data structures [27]. Similarly, the partially dynamic maintenance of a minimum spanning tree during edge insertions can be handled in $O(\log m)$ [1] time per operation [11] using the dynamic trees of Sleator and Tarjan [22, 23]. However, in both cases no better bound than $O(\sqrt{m})$ is known for the corresponding fully dynamic problems [11]. Moreover, despite intensive research on dynamic problems on graphs (such as dynamic maintenance of connectivity [7, 8, 10, 11, 14, 20, 22, 29, 30], 2- and 3-connectivity [7, 12, 29, 30], transitive closure [3, 4, 15, 16, 17, 18, 19, 31], planar graphs [6, 7, 19, 25], shortest paths [2, 9, 21, 24, 31] and minimum spanning trees [5, 8, 11, 24]), there are very few graph-theoretic problems for which a fully dynamic non-trivial algorithm is known.

As mentioned in [30] the fully dynamic maintenance of the connected components of a graph differs substantially from the fully dynamic maintenance of the biconnected components. Indeed in the first problem a single edge insertion can merge at most two components into one. On the other hand, for the second problem a single edge insertion can produce a cycle in the graph that might combine as many as $\Theta(n)$ biconnected components into one. As a result, by simply alternating the insertion and deletion of the same edge, we can generate a sequence of operations that at each step changes the number of components by $\Theta(n)$. This also explains why we do not maintain biconnected components' names: indeed in such a scenario a *Delete* operation would have to specify as many as $\Theta(n)$ different new names.

The remainder of this paper consists of four sections. Section 2 presents a data structure for connected planar graphs. Section 3 describes the implementation of the operations and analyzes their time complexity. Section 4 deals with unconnected planar graphs, while Section 5 lists some concluding remarks.

2 A Data Structure for Connected Planar Graphs

In this section we describe a data structure for maintaining the biconnected components of an undirected planar graph G under an intermixed sequence of *SameBlock*, *Insert*, and *Delete* operations. For sake of simplicity, we start by describing algorithms dealing with connected planar graphs. We will show later how to extend the same algorithms to unconnected planar graphs with the same time bounds.

We describe a data structure that operates on planar graphs with bounded degree. It is well known (see for example [13, page 132]) that each graph $G = (V, E)$ can be transformed into a graph $G' = (V', E')$, whose vertices have degree no greater than three. We use the following transformation. For each vertex u of G of degree $d \geq 4$, where $v_0, v_1, \ldots, v_{d-1}$ are the vertices adjacent to u, we replace u with new vertices $u_0, u_1, \ldots, u_{d-1}$ and replace edge (u, v_i) with a *solid*

[1] In this paper all the logarithms are assumed to be base two.

1 Introduction

An undirected graph $G = (V, E)$ is said to be *connected* if there is a path from each vertex to every other vertex, and *biconnected* if there are at least two vertex disjoint paths from each vertex to every other vertex. It is well known [1] that a graph G is not biconnected if and only if there is a vertex whose removal disconnects G: such a vertex is called an *articulation point*. Given a graph $G = (V, E)$, let $E_1, E_2, \ldots, E_h$ be the partition of E into equivalence classes such that two edges e' and e'' are in the same equivalence class if and only if either (i) $e' = e''$ or (ii) there is a simple cycle (i.e., a cycle with no repetition of edges and vertices) of G containing both e' and e''. For $1 \leq i \leq h$ let V_i be the set of vertices that are endpoints of the edges in E_i. The subgraphs $G_i = (V_i, E_i)$ are called the *biconnected components* (or *blocks*) of G. An articulation point appears in more than one biconnected component. The set of edges of every graph can be partitioned into biconnected components in a unique way and every edge belongs exactly to one biconnected component. Figure 1 shows a graph together with its biconnected components.

[Figure 1]

The problems of computing the biconnected components of undirected graphs arise naturally in many applications and have been extensively studied. Tarjan [26] give optimal linear-time sequential algorithms. Tarjan and Vishkin [28] give logarithmic-time parallel algorithms. Recently, Westbrook and Tarjan [30] considered the problem of maintaining the biconnected components of a graph undergoing a sequence of edge and vertex insertions. They presented algorithms that run in a total of $O(q\alpha(q, n))$ time, where q is the total number of operations, n is the number of vertices and α is a very slowly growing function (a functional inverse of Ackermann's function). In the rest of this paper, we will refer to a problem in which only insertions are defined as a *partially dynamic* problem. Problems in which both insertions and deletions are allowed are called *fully dynamic*.

In this paper we are interested in the fully dynamic maintenance of the biconnected components of an undirected planar graph. We would like to perform any sequence of three kinds of operations starting on an initially null graph G (i.e., a graph with n vertices and no edges).

> *SameBlock*(u, v): Return *true* if vertices u and v are in a same biconnected component. Return *false* otherwise.
>
> *Insert*(x, y): Insert a new edge between the two vertices x and y, provided that the insertion of the new edge leaves the graph planar.
>
> *Delete*(x, y): Delete the edge between the two vertices x and y.

As mentioned in [30], in such a fully dynamic setting there has been no known solution better than a repeated application of an off-line algorithm, which yields a single operation worst-case time complexity of $O(m)$, where m is the current number of edges in the graph. We show that for planar graphs this problem can be solved in $O(n^{2/3})$ time per operation. We obtain the same bounds even if we are initially given a non-null graph $G_0 = (V_0, E_0)$ and we allow $O(|V_0| + |E_0|)$ preprocessing time. In addition our data structure is able to return an articulation point separating two given vertices x and y (if one exists) in the same $O(n^{2/3})$ time bound. Furthermore, insertions of new vertices and deletions of disconnected vertices can be supported in the same time bound. This

Maintaining Biconnected Components of Dynamic Planar Graphs *

Zvi Galil
Columbia University, and
Tel-Aviv University

Giuseppe F. Italiano †
Columbia University, and
Università di Roma, Italy

Abstract

We present algorithms for maintaining the biconnected components of a planar graph undergoing repeated dynamic modifications, such as insertions and deletions of edges and vertices. We show how to test at any time whether two vertices belong to the same biconnected component, and how to insert and delete an edge in $O(n^{2/3})$ time in the worst case, where n is the number of vertices in the graph. The data structure supports also insertions of new vertices and deletions of disconnected vertices in the same time bounds. This is the first sublinear-time algorithm known for this problem.

Key Words: Analysis of algorithms, dynamic data structures, planar graphs, vertex connectivity.

*Work partially supported by NSF Grants CCR-8814977, CCR-9014605, by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (Project ALCOM), and by the Italian MURST Project "Algoritmi e Strutture di Calcolo". A preliminary version of this paper appears in [*Proc. 18th Int. Colloquium on Automata, Languages, and Programming*].

†Partially supported by an IBM Graduate Fellowship. Part of this work was done while visiting the IBM T. J. Watson Research Center and Universität des Saarlandes, Saarbrücken, Germany.