Optimal Decremental Connectivity in Planar Graphs*

Jakub Łącki[1] **and Piotr Sankowski**[2]

1 **University of Warsaw**
Warsaw, Poland
j.lacki@mimuw.edu.pl

2 **University of Warsaw**
Warsaw, Poland
sank@mimuw.edu.pl

Abstract

We show an algorithm for dynamic maintenance of connectivity information in an undirected planar graph subject to edge deletions. Our algorithm may answer connectivity queries of the form 'Are vertices u and v connected with a path?' in constant time. The queries can be intermixed with any sequence of edge deletions, and the algorithm handles all updates in $O(n)$ time. This results improves over previously known $O(n \log n)$ time algorithm.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases decremental connectivity, planar graphs, dynamic connectivity, algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.608

1 Introduction

The *dynamic graph connectivity* problem consists in maintaining connectivity information about an undirected graph, which is undergoing modifications. Typically, the modifications are additions or removals of edges or vertices. In this paper we focus on the problems in which each modification adds or removes a single edge. These problems have three variants: in the *incremental* version, edges can only be added to the graph, in the *decremental* one the edges may only be removed, whereas in the *fully dynamic* version both edge insertions and deletions are allowed. Graph updates are intermixed with a set of connectivity queries of the form 'Are vertices u and w in the same connected component?'

We consider the decremental connectivity problem for planar graphs, and show an algorithm that may answer connectivity queries in constant time and process any sequence of edge deletions in $O(n)$ time. The previously known best running time of $O(n \log n)$ was obtained by using the fully dynamic algorithm. We assume word-RAM model with standard operations.

* Jakub Łącki is a recipient of the Google Europe Fellowship in Graph Algorithms, and this research is supported in part by this Google Fellowship. Piotr Sankowski is partially supported by ERC grant PAAl no. 259515, NCN grant "Efficient planar graph algorithms" and the Foundation for Polish Science.

1.1 Prior work

It is easy to see that incremental graph connectivity can be solved using an algorithm for the union-find problem. It follows from the result of Tarjan [16] that a sequence of t edge insertions and t queries can be handled in $O(t\alpha(t))$ time, where $\alpha(t)$ is the extremely slowly growing inverse Ackermann function.

There has been a long line of research considering the fully dynamic connectivity in general graphs [6, 3, 8, 10, 19, 11, 21]. The best currently known algorithms have polylogarithmic update and query time. Thorup [19] has shown a randomized Monte Carlo algorithm with $O(\log n(\log\log n)^3)$ amortized update and $O(\log n/\log\log\log n)$ query time.[1] An algorithm by Wulff-Nilsen [21] handles updates in slightly worse $O(\log^2 n/\log\log n)$ amortized time, but it is deterministic and answers queries in $O(\log n/\log\log n)$ time. The best algorithm with worst-case update guarantee is a randomized algorithm by Kapron, King and Mountjoy [11], which processes updates in $O(\log^5 n)$ time and answers queries in $O(\log n/\log\log n)$ time. However, if we require a deterministic algorithm with worst-case running time guarantee, nothing better than a $O(\sqrt{n})$ time algorithm is known [6, 3, 2].

For the decremental variant, Thorup [18] has shown a randomized algorithm, which can process any sequence of edge deletions in $O(m\log(n^2/m) + n(\log n)^3(\log\log n)^2)$ time and answers queries in constant time. Here, m is the initial number of edges in the graph. If $m = \Theta(n^2)$, the update time is $O(m)$, whereas for $m = \Omega(n(\log n\log\log n)^2)$ it is $O(m\log n)$.

The picture is much simpler in case of planar graphs. Eppstein et. al [5] gave a fully dynamic algorithm which handles updates and queries in $O(\log n)$ amortized time, but requires that the graph embedding remains fixed. For the general case (i.e., when the embedding may change) Eppstein et. al [4] gave an algorithm with $O(\log^2 n)$ worst-case update time and $O(\log n)$ query time.

In planar graphs, the best known solution for the incremental connectivity problem is the union-find algorithm. However, for the special case when the final resulting planar graph is given upfront, and the edge insertions and queries are given later in a dynamic fashion Gustedt [7] has shown an $O(n)$ time algorithm. On the other hand, for the decremental problem nothing better than a direct application of the fully dynamic algorithm is known. This is different from both general graphs and trees, where the decremental connectivity problems have better solutions than what could be achieved by a simple application of their fully dynamic counterparts. In case of general graphs, the best total update time was $O(m\log n)$ [18] (except for very sparse graphs, including planar graphs), compared to $O(m\log n(\log\log n)^3)$ time for the fully dynamic variant. For trees, only $O(n)$ time is necessary to perform all updates in the decremental scenario [1], while in the fully dynamic case one can use dynamic trees and obtain $O(\log n)$ worst case update time.

There has also been some progress in obtaining lower bounds for dynamic connectivity problems. Tarjan and La Poutré [17, 15] have shown that incremental connectivity requires $\Omega(\alpha(n))$ time per operation on a pointer machine. Henzinger and Fredman [9] considered the fully dynamic problem and RAM model and obtained a lower bound of $\Omega(\log n/\log\log n)$, which also works for plane graphs. This was improved by Demaine and Pătraşcu [14] to a lower bound of $\Omega(\log n)$ in cell-probe model. The lower bound holds also for plane graphs.

[1] Throughout the paper we use n and m to denote, respectively, the number of vertices and the number of edges in the graph.

1.2 Our results

We show an algorithm for the decremental connectivity problem in planar graphs, which processes any sequence of edge deletions in $O(n)$ time and answers queries in constant time. This improves over the previous bound of $O(n \log n)$, which can be obtained by applying the fully dynamic algorithm by Eppstein [5], and matches the running time of decremental connectivity on trees [1].

In fact, we present a $O(n)$ time reduction from the decremental connectivity problem to a collection of incremental problems in graphs of total size $O(n)$. These incremental problems have a specific structure: the set of allowed union operations forms a planar graph and is given in advance. As shown by Gustedt [7], such a problem can be solved in linear time. Our result shows that in terms of total update time, the decremental connectivity problem in planar graphs is definitely not harder than the incremental one. It should be noted that the union-find algorithm can process any sequence of k query or update operations in $O(k\alpha(n))$ time, while in our algorithm we are only able to bound the time to process any sequence of edge deletions.

Moreover, since fully dynamic connectivity has a lower bound of $\Omega(\log n)$ (even in plane graphs) shown by Demaine and Pătraşcu [14], our results imply that in planar graphs decremental connectivity is strictly easier than the fully dynamic one. We suspect that the same holds for general graphs, and we conjecture that it is possible to break the $\Omega(\log n)$ bound for a single operation of a decremental connectivity algorithm, or the $\Omega(m \log n)$ bound for processing a sequence of m edge deletions.

Our algorithm, unlike the majority of algorithms for maintaining connectivity, does not maintain the spanning tree of the current graph. As a result, it does not have to search for a replacement edge when an edge from the spanning tree is deleted. Our approach is based on a novel and very simple approach for detecting bridges, which alone gives $O(n \log n)$ total update time. We use the fact that a deletion of edge uw in the graph causes some connected component to split if both sides of uw belong to the same face. This condition can in turn be verified by solving an incremental connectivity problem in the dual graph. When we detect a deletion that splits a connected component, we start two parallel DFS searches from u and w to identify the *smaller* of the two new components. Once the first search finishes, the other one is stopped. A simple argument shows that this algorithm runs in $O(n \log n)$ time.

We then show that the DFS searches can be speeded up using an r-division, that is a decomposition of a planar graph into subgraphs of size at most $r = \log^2 n$. This gives an algorithm running in $O(n \log \log n)$ time. For further illustration of this idea we show how to apply it twice in order to obtain an $O(n \log \log \log n)$ time algorithm. Then, we observe that the $O(n \log \log \log n)$ time algorithm reduces the problem of maintaining connectivity in the input graph to maintaining connectivity in a number of graphs of size at most $O(\log^2 \log n)$. The number of such graphs is so small that we can simply precompute the answers for all of them and use these precomputed answers to obtain the main result of the paper. The preprocessing of all graphs of bounded size is again an idea that, to the best of our knowledge, has never been previously used for designing dynamic graph algorithms.

1.3 Organization of the paper

In Section 2 we introduce notation and recall some of the concepts that we later use. The following sections describe our algorithm. We start with the description of the simple $O(n \log n)$ time algorithm in Section 3, and then in every section we show an improvement in the running time.

In Section 4 we show how to use r-division to get an $O(n \log \log n)$ algorithm. Section 5, shows how to improve the reduction, so that it can be used more than once, which results in an $O(n \log \log \log n)$ time algorithm. Finally, in Section 6 we show how to solve the decremental connectivity in optimal time for graphs of size $O(\log^2 \log n)$, after initial preprocessing. This, combined with the reduction applied twice, gives the main result of the paper.

2 Preliminaries

Let $G = (V, E)$ be an undirected, unweighted planar graph, and $n = |V|$. By $V(G)$, $E(G)$ and $F(G)$ we denote the sets of vertices, edges and faces of G. The Euler's formula states that $|V(G)| - |E(G)| + |F(G)| = |CC(G)| + 1$, where $CC(G)$ is the set of connected components of G. The *dual graph* G^* is constructed from G by embedding a single vertex in every face of G and connecting the vertices in adjacent faces of G. Note that if two faces f_1, f_2 share more than one edge, G^* has multiple edges between f_1 and f_2.

In the paper we deal with algorithms that maintain the connectivity information about a graph G subject to edge deletions. By the total running time we denote the total time of handling deletions of all edges from the graph.

The identifier of a connected component (henceforth denoted *cc-identifier*) is a value assigned to a vertex $v \in V$, which uniquely identifies the connected component of G, i.e., two vertices have the same cc-identifier if and only if they belong to the same connected component. The cc-identifiers change as the edges are deleted, and they may not be preserved after edge deletion. An algorithm maintains cc-identifiers *explicitly* if after every deletion it returns the list of changes to the cc-identifiers. We assume that cc-identifiers are integers that require $\log n + O(1)$ bits.[2] Note that an algorithm which maintains cc-identifiers explicitly can be simply turned into an algorithm with constant query time. In order to answer a query regarding two vertices, it suffices to compare the cc-identifiers of the two vertices. By definition, the vertices are in the same connected component if and only if their cc-identifiers are equal.

Let us now recall the notion of an r-division. A *region* R is an edge-induced subgraph of G. A *boundary vertex* of a region R is a vertex $v \in V(R)$ that is adjacent to an edge $e \notin E(R)$. We denote the set of boundary vertices of a region R by $\partial(R)$. An *r-division* $\mathcal{P}$ of G is a partition of G into $O(n/r)$ edge-disjoint regions (which might share vertices), such that each region contains at most r vertices and $O(\sqrt{r})$ boundary vertices. The set of boundary vertices of a division $\mathcal{P}$, denoted $\partial(\mathcal{P})$ is the union of the sets $\partial(R)$ over all regions R of $\mathcal{P}$. Note that $|\partial(\mathcal{P})| = O(n/\sqrt{r})$.

▶ **Lemma 1** ([13, 20]). *Let $G = (V, E)$ be an n-vertex biconnected triangulated planar graph and $1 \le r \le n$. An r-division of G can be constructed in $O(n)$ time.*

Let G be a planar graph. In the preprocessing phase of our algorithms, we build an r-division of G. This r-division will be updated in a natural way, as edges are deleted from G. Namely, when an edge is deleted from the graph, we update its r-division by deleting the corresponding edge. However, if we strictly follow the definition, what we obtain may no longer be an r-division.

For that reason, we loosen the definition of an r-division, so that it includes the divisions obtained by deleting edges. Consider an r-division $\mathcal{P}$ built for a graph G. Moreover, let G'

[2] Throughout this paper, $\log n$ denotes binary logarithm.

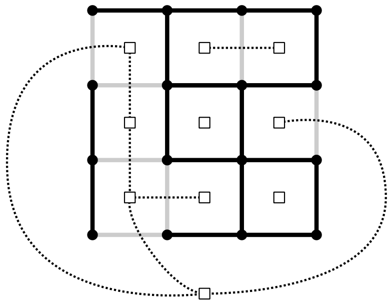

Figure 1 The graphs from the proof of Lemma 3. Edges of G are drawn with solid black lines, whereas the gray lines depict edges that have been deleted from G. The small squares are vertices of D_G, and the dotted lines are edges of D_G.

be a graph obtained from G by deleting edges, and let $\mathcal{P}'$ be the r-division $\mathcal{P}$ updated in the following way. Let R be a region of $\mathcal{P}$. Then, we define the graph R' in $\mathcal{P}$ obtained by removing edges from R to be a region of $\mathcal{P}'$, although it may no longer be an edge-induced subgraph of G', e.g., it may contain isolated vertices. Similarly, we define the set of boundary vertices of $\mathcal{P}'$ to be the set of boundary vertices of $\mathcal{P}$. Again, according to this definition, a boundary vertex v of $\mathcal{P}'$ may be incident to edges of a single region (because the edges incident to v that belonged to other regions have been deleted). In the following, we say that $\mathcal{P}'$ is an r-division of G'.

Since Lemma 1 requires the graph to be biconnected and triangulated, in order to obtain an r-division for a graph which does not have these properties, we first add edges to G to make it biconnected and triangulated, then compute the r-division of G, and then delete the added edges both from G and its division.

Without loss of generality, we can assume that each vertex $v \in V$ has degree at most 3. This can be assured by triangulating the dual graph in the very beginning. In particular, this assures that each vertex belongs to a constant number of regions in an r-division.

3 $O(n \log n)$ Time Algorithm

Let G be a planar graph subject to edge deletions. We call an edge deletion *critical* if and only if it increases the number of components of G, i.e., the deleted edge is a bridge in G. We first show a dynamic algorithm that for every edge deletion decides, whether it is critical. It is based on a simple relation between the graph G and its dual.

▶ **Lemma 2.** *Let G be a planar graph subject to edge deletions. There exists an algorithm that for each edge deletion decides whether it is critical. It runs in $O(n)$ total time.*

Proof. The intuition behind the proof is as follows. We maintain the number of faces in G. In order to do that, when an edge e is deleted, we simply merge faces on both sides of e (if they are different from each other). This can be implemented using union-find data structure on the vertices of the dual graph.

More formally, we build and maintain a graph D_G. Initially, this is a graph consisting of vertices of G^* (faces of G). When an edge is deleted from G, we add its dual edge to D_G (see Fig. 1). Clearly, the connected components of D_G are exactly the faces of G. Since edges are only added to D_G, we can easily maintain the number of connected components in D_G with a union-find data structure.

This allows us to detect critical deletions in G. After every edge deletion, we know the number of edges and vertices of G. Moreover, we know that the number of faces of G is equal to the number of connected components of D_G, which we also maintain. As a result, by Euler's formula, we get the number of connected components of G, so in particular we may check if the deletion caused the number of connected components to increase. The algorithm executes $O(n)$ find and union operations on the union-find data structure.

In addition to that, the sequence of union operations has a certain structure. Let G_1 be the initial version of the graph G (before any edge deletion). Observe that each union operation takes as arguments the endpoints of an edge of G_1^*. The variant of the union-find problem, in which the set of allowed union operations forms a planar graph given during initialization, was considered by Gustedt [7]. He showed that for this special case of the union-find problem there exists an algorithm that may execute any sequence of $O(n)$ operations in $O(n)$ time (for an n-vertex planar graph). Thus, we infer that our algorithm runs in $O(n)$ time. ◀

We can now use Lemma 2 to show a simple decremental connectivity algorithm that runs in $O(n \log n)$ total time.

▶ **Lemma 3.** *Let G be a planar graph subject to edge deletions. There exists a decremental connectivity algorithm that for every vertex of G maintains its cc-identifier explicitly. It runs in $O(n \log n)$ total time.*

Proof. We use Lemma 2 to detect critical deletions. When an edge uw is deleted, and the deletion is not critical, nothing has to be done. Otherwise, after a critical deletion, some connected component C breaks into two components C_u and C_w ($u \in C_u$, $w \in C_w$) and we start two parallel depth-first searches from u and w. We stop both searches once the first of them finishes. W.l.o.g. assume that it is the search started from u. Thus, we know that the size of C_u is at most half of the size of C.[3] We can now iterate through all vertices of C_u and change their cc-identifiers to a new unique number. All these steps require $O(|C_u|)$ time. The running time of the algorithm is proportional to the total number of changes of the cc-identifiers. Since every vertex changes its identifier only when the size of its connected component halves, we infer that the total running time is $O(n \log n)$. ◀

4 $O(n \log \log n)$ Time Algorithm

In order to speed up the $O(n \log n)$ algorithm, we need to speed up the linear depth-first searches that are run after a critical edge deletion. We build an r-division $\mathcal{P}$ of G for $r = \log^2 n$ and use a separate decremental connectivity algorithm to maintain the connectivity information inside each region. On top of that, we maintain a *skeleton graph* that represents connectivity information between the set of boundary vertices (and possibly some other vertices that we consider important). Loosely speaking, since the number of boundary vertices is $O(n/\log n)$ we can pay a cost of $O(\log n)$ for maintaining the cc-identifier for each of them.

▶ **Definition 4.** Consider an r-division $\mathcal{P}$ of a planar graph $G = (V, E)$ and a set V_s (called a *skeleton set*), such that $\partial(\mathcal{P}) \subseteq V_s \subseteq V$. The *skeleton graph* for $\mathcal{P}$ and V_s is a graph over the skeleton set V_s and some additional auxiliary vertices. Consider a region R of $\mathcal{P}$. Group

[3] Since the graph has constant degree, we may assure that both searches are synchronized in terms of the number of visited vertices.

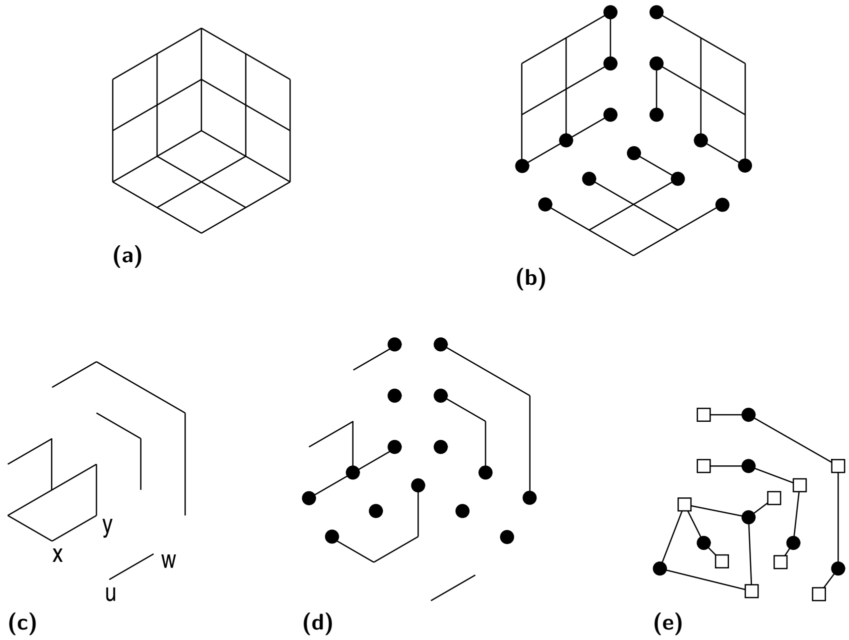

Figure 2 Panels 2a and 2b show a sample graph G and its r-division into three regions (boundary vertices are marked with small circles). In panel 2c there is graph G' obtained from G by a sequence of edge deletions. Panel 2d shows its r-division obtained from the r-division of G (again, boundary vertices are marked with small circles). Finally, panel 2e contains the skeleton graph of G'. Auxiliary vertices are marked with squares.

vertices of $V_s \cap V(R)$ into sets $V_1, \ldots, V_k$, such that two vertices belong to the same set if and only if there is a path in R that connects them. For each set V_i add a new auxiliary vertex w_i and add an edge $w_i x$ for every $x \in V_i$.

For illustration, see Fig. 2.

▶ **Lemma 5.** *The skeleton graph has $O(|V_s|)$ vertices and edges.*

Proof. For each region R, we add at most one vertex and edge per each vertex of $V_s \cap V(R)$. Since each vertex belongs to a constant number of regions, we get the desired bound. ◀

▶ **Lemma 6.** *If $u, w \in V_s$, then u and w are connected in the skeleton graph if and only if they are connected in G.*

Proof. Consider a region R of the r-division. From the construction it follows that two vertices of $V_s \cap V(R)$ are connected in G with a path inside R iff they are connected in the part of the skeleton graph built for this region.
($\implies$) Follows directly from the above observation.
($\impliedby$) Consider a path P in G between u and w. Break this path into subpaths at each element of V_s. Since $\partial(\mathcal{P}) \subseteq V_s \subseteq V$, each resulting subpath is fully contained in one region of the r-division. Clearly, from the property given at the beginning of the proof, for each subpath there exists a corresponding path in the skeleton graph. ◀

In our algorithm we will update the skeleton graph of G, as edges are deleted. Similarly to the $O(n \log n)$ algorithm, we need a way of detecting whether an edge deletion in G increases the number of connected components in the skeleton graph.

▶ **Lemma 7.** *Let G be a dynamic planar graph, subject to edge deletions. Assume that we maintain its skeleton graph G_s computed for an r-division $\mathcal{P}$ and a skeleton set V_s. An edge deletion in G causes an increase in the number of connected components in G_s if and only if the deletion is critical in G and there exists a region of $\mathcal{P}$, in which the deletion disconnects some two vertices of V_s.*

Before we proceed with the proof, let us note that all its conditions are necessary. In particular, a critical deletion in G may not disconnect some two vertices of a skeleton set in a region (e.g. edge uw in Fig. 2c, whose deletion does not affect the skeleton graph at all). It may also happen that the deletion is not critical in G, but inside some region it disconnects some two vertices of V_s (e.g. edge xy in Fig. 2c).

Proof. By Lemma 6, two vertices of V_s are connected in G iff they are connected in G_s.
($\Longrightarrow$) If two vertices of V_s become disconnected in G_s, they also become disconnected in G, so the edge deletion is critical. The deletion has to disconnect some two vertices in a region, because otherwise the graph G_s would not change at all.
($\Longleftarrow$) Assume that the deletion disconnected vertices $u, w \in V_s$ in a region R. Thus, the deleted edge was on some path from u to w. Since the edge deletion is critical in G, the deleted edge was a bridge in G. After the deletion there is no path from u to w in G and consequently also in G_s. ◀

Before we proceed with the algorithm, we show how to extend an algorithm maintaining cc-identifiers with two useful operations.

▶ **Lemma 8.** *Let $G = (V, E)$ be a planar graph and let $X \subseteq V$. Assume there exists a decremental connectivity algorithm that maintains cc-identifiers of a set $X \subseteq V$ explicitly and processes updates in $\Omega(n)$ total time. Then, we can extend the algorithm, so that:*

- *after every edge deletion, if the deletion disconnects some two vertices of X, it reports a pair of vertices that become disconnected,*
- *given a cc-identifier, it returns a vertex $v \in X$ with the same cc-identifier (or reports that such a vertex does not exist).*

The extended algorithm has the same asymptotic running time.

Proof. Since each cc-identifier can be encoded in $\log n + O(1)$ bits, there are $O(n)$ possible cc-identifiers. Thus, for each possible cc-identifier c, we maintain a list L_c of vertices of X, whose cc-identifier is c. Note that maintaining these lists takes time that is linear in the number of changes of cc-identifiers. Moreover, we need $O(n)$ time to initialize the lists L_c.

Observe that the lists allow us to find a vertex of X of given cc-identifier in constant time, so the second claim follows. To show the first claim, consider a case when after an edge deletion some (but not all) elements from a list L_c are removed. All this elements have to be added to a single list $L_{c'}$ and $L_{c'}$ must have been empty before the new elements were added. This means that the number of distinct cc-identifiers have increased, and some elements of X became disconnected. We can now take any $u \in L_c$ and $w \in L_{c'}$ and report that u and w became disconnected. ◀

We are ready to show the main building block of our $O(n \log \log n)$ algorithm.

▶ **Lemma 9.** *Let G be a planar graph. Assume there exists a decremental connectivity algorithm that runs in $f(n)$ time and maintains cc-identifiers explicitly. Then, there exists a decremental connectivity algorithm that runs in $O(n + n \cdot f(\log^2 n)/\log^2 n)$ time and answers queries in $O(1)$ time.*

Proof. We build an r-division $\mathcal{P}$ of G for $r = \log^2 n$. By Lemma 1, this takes $O(n)$ time. For each region R of the division, we run the assumed decremental algorithm to handle edge deletions. We use A_R to denote the algorithm run for region R. A_R maintains cc-identifiers of $V(R)$ explicitly. We call these cc-identifiers *local* cc-identifiers. We also extend each A_R according to Lemma 8, taking $X = \partial(\mathcal{P}) \cap V(R)$. Moreover, we use Lemma 2 to detect critical deletions in G.

We build the skeleton graph G_s for G, r-division $\mathcal{P}$ and a skeleton set $V_s = \partial(\mathcal{P})$. We maintain G_s, as edges are deleted, that is the deletions in G are reflected in G_s. This can be done using the algorithms A_R. By Lemma 8, A_R can report that some two vertices of V_s become disconnected inside R. This means that G_s needs to be updated. Observe that the part of G_s inside a region R can be implicitly represented as a partition of $V_s \cap V(R)$, where two vertices belong to the same element of the partition, if they are connected in R. Thus, if a deletion causes t local cc-identifiers to change, we may update G_s in $O(t)$ time. As a result, the time for updating G_s is linear in the number of local cc-identifiers that are changed.

For every vertex of G_s, we maintain its cc-identifier (called a *global* cc-identifier). Once G_s is updated after an edge deletion, we use Lemma 7 to check whether the number of connected components of G_s increased. According to the Lemma, it suffices to check whether the deletion is critical in G (this is reported by the algorithm of Lemma 2), and whether some two elements of the skeleton set became disconnected within some region (using Lemma 8).

When we detect that the number of connected components of the skeleton graph G_s has increased, similarly to the $O(n \log n)$ algorithm, we run two parallel DFS searches to identify the smaller of the two new connected components, and update the global cc-identifiers.

In order to answer a query regarding two vertices u and w, we perform two checks. First, if the vertices belong to the same region, we check whether there exists a path connecting them that does not contain any boundary vertices. This can be done by querying algorithm A_R for the appropriate region.

Then, we check whether there is a path from u to w that that contains some boundary vertex. For each of the two vertices, we find two arbitrary boundary vertices b_u and b_w that u and w are connected to (using Lemma 8). Then, we check whether b_u and b_w have the same global cc-identifier.

Let us now analyze the running time. The algorithm of Lemma 2 requires $O(n)$ time. The algorithms A_R take $O(n \cdot f(r)/r) = O(n \cdot f(\log^2 n)/\log^2 n)$ time. Lastly, we bound the running time of the DFS searches performed to update the global cc-identifiers. We use an argument similar to the one in the proof of Lemma 3. The skeleton graph has $O(n/\log n)$ vertices, and each global cc-identifier can change at most $O(\log(n/\log n)) = O(\log n)$ times. Hence, the DFS searches require $O((n/\log n)\log n) = O(n)$ time. The lemma follows. ◀

By applying Lemma 3 to Lemma 9, we obtain the following.

▶ **Lemma 10.** *There exists a decremental connectivity algorithm for planar graphs that runs in $O(n \log\log n)$ total time.*

Proof. The total update time of the algorithm of Lemma 3 is $f(n) = O(n \log n)$. Thus, the running time is $O(n+n\cdot f(\log^2 n)/\log^2 n) = O(n+n\log^2 n \log\log n/\log^2 n) = O(n \log\log n)$. ◀

5 $O(n \log \log \log n)$ Time Algorithm

In order to obtain a faster algorithm, we would like to use Lemma 9 multiple times, starting from the $O(n \log n)$ algorithm, and each time applying the Lemma to the algorithm obtained in the previous step. This, however, cannot be done directly. While the Lemma requires an algorithm that maintains all cc-identifiers explicitly, it does not produce an algorithm with this property. We deal with this problem in this section.

Observe that in the proof of Lemma 9 we only needed the assumed decremental algorithm to maintain the cc-identifiers of the vertices of the skeleton set. This fact can be exploited in the following way. We show that if we have an algorithm that maintains cc-identifiers of some vertices, we may construct another (possibly faster) algorithm with the same property.

▶ **Lemma 11.** *Assume there exists a decremental connectivity algorithm for planar graphs that, given a graph $G = (V, E)$ and a set $V_e \subseteq V$ (called an* explicit *set):*
- *maintains cc-identifiers of the vertices of V_e explicitly,*
- *processes updates in $f(n) + O(|V_e| \log n)$ time,*
- *may return the cc-identifier of any vertex in $g(n)$ time,*

where $f(n)$ and $g(n)$ are nondecreasing functions.

Then, there exists a decremental connectivity algorithm for planar graphs, which, given a graph $G = (V, E)$ and a set $V_e \subseteq V$:
- *maintains cc-identifiers of the vertices of V_e explicitly,*
- *processes updates in $O(n + |V_e| \log n + n \cdot f(\log^2 n)/ \log^2 n)$ time,*
- *may return the cc-identifier of any vertex in $g(\log^2 n) + O(1)$ time.*

Proof. We build an r-division $\mathcal{P}$ of G for $r = \log^2 n$. By Lemma 1, this takes $O(n)$ time. We also build a skeleton graph G_s, by taking a skeleton set $V_s := V_e \cup \partial(\mathcal{P})$. Hence, $|V_s| = |V_e| + n/ \log n$.

For each region R of $\mathcal{P}$, we run a copy A_R of the assumed decremental connectivity algorithm, extended according to Lemma 8. Observe that in the proof of Lemma 9, we only need A_R to explicitly maintain cc-identifiers of $V_s \cap V(R)$. Thus, the set of explicit vertices for algorithm A_R is $V_s \cap V(R)$. Hence, A_R maintains local cc-identifiers of these vertices.

We maintain the graph G_s and its global cc-identifiers in the same way as in the proof of Lemma 9. The only difference is that now the skeleton set V_s is bigger. Let us bound the running time. First, algorithm A_R uses $f(\log^2 n) + O(|V_s \cap V(R)| \log n)$ time. Summing it over all regions, we obtain

$$\begin{aligned}\sum_{R\in\mathcal{P}} f(\log^2 n) + O(|V_s \cap V(R)| \log n)& \\ &= O(n \cdot f(\log^2 n)/ \log^2 n + |V_s| \log n) \\ &= (n \cdot f(\log^2 n)/ \log^2 n + |V_e| \log n + n/ \log n \cdot \log n) \\ &= (n \cdot f(\log^2 n)/ \log^2 n + |V_e| \log n + n).\end{aligned}$$

Note that we use the fact that each vertex is contained in a constant number of regions. The the running time of depth-first searches used to update the global cc-identifiers is

$$O(|V_s| \log n) = O(n/ \log n \cdot \log n + |V_e| \log n) = O(n + |V_e| \log n).$$

Thus, the total update time is $O(n + |V_e| \log n + n \cdot f(\log^2 n)/ \log^2 n)$.

Since the cc-identifiers of vertices of G_s are maintained explicitly, in particular we explicitly maintain the cc-identifiers of vertices of V_e. It remains to describe the process of computing the global cc-identifier of an arbitrary vertex $v \in V$. Assume that v belongs to a region R (in case v is a boundary vertex, we may use an arbitrary region containing it). We first query A_R to obtain the local cc-identifier of v. We use Lemma 8 to check whether there exists a vertex b_v in $V_s \cap V(R)$ that has the same local cc-identifier as v. If this is the case, since b_v belongs to the skeleton set, we return its global cc-identifier (maintained explicitly). Otherwise, we return a new cc-identifier by encoding as an integer a pair consisting of the identifier of the region containing v (this requires $\log O(n/\log^2 n) = \log n + O(1) - 2\log\log n$ bits) and the local cc-identifier of v (which requires $\log\log^2 n + O(1) = 2\log\log n + O(1)$ bits). Overall, the resulting cc-identifier requires $\log n + O(1)$ bits. Thus, obtaining a cc-identifier of an arbitrary vertex requires $g(\log^2 n) + O(1)$ time. ◀

The main advantage of Lemma 11 over Lemma 9 is that we may apply Lemma 11 recursively to obtain better algorithms. We can view applying Lemma 11 as reducing connectivity in a graph of size n to connectivity in a collection of graphs of size $\log^2 n$. If we apply Lemma 11 to itself, we obtain the following.

▶ **Lemma 12.** *Assume there exists a decremental connectivity algorithm for planar graphs that, given a graph $G = (V, E)$ and a set $V_e \subseteq V$ (called an* explicit *set):*

- *maintains cc-identifiers of the vertices of V_e explicitly,*
- *processes updates in $f(n) + O(|V_e| \log n)$ time,*
- *may return the cc-identifier of any vertex in $g(n)$ time,*

where $f(n)$ and $g(n)$ are nondecreasing functions.

Then, there exists a decremental connectivity algorithm for planar graphs, which, given a graph $G = (V, E)$ and a set $V_e \subseteq V$:

- *maintains cc-identifiers of the vertices of V_e explicitly,*
- *processes updates in $O(n + |V_e| \log n + n \cdot f(\log^2 \log^2 n)/\log^2 \log^2 n)$ time,*
- *may return the cc-identifier of any vertex in $g(\log^2 \log^2 n) + O(1)$ time.*

Proof. We apply Lemma 11 to the assumed algorithm and obtain an algorithm with total update time $f_1(n) + O(|V_e| \log n)$, where $f_1(n) = O(n + n \cdot f(\log^2 n)/\log^2 n)$ and query time $g_1(n) = g(\log^2 n) + O(1)$. Then, we apply the Lemma again to the new algorithm and get a new algorithm, whose total update time is

$$\begin{aligned} &O(n + |V_e| \log n + n \cdot f_1(\log^2 n)/\log^2 n) = \\ &\quad = O(n + |V_e| \log n + n(\log^2 n + \log^2 n \cdot f(\log^2 \log^2 n)/\log^2 \log^2 n)/\log^2 n) \\ &\quad = O(n + |V_e| \log n + n \cdot f(\log^2 \log^2 n)/\log^2 \log^2 n). \end{aligned}$$

It answers queries in $g(\log^2 \log^2 n) + O(1)$ time. ◀

We may now apply Lemma 12 to the simple $O(n \log n)$ algorithm (see Lemma 3) to obtain the following.

▶ **Lemma 13.** *There exists a decremental connectivity algorithm, which processes any sequence of updates in $O(n \log\log\log n)$ time.*

Proof. The simple algorithm processes updates in $f(n) = O(n \log n)$ time. Thus, we have $f(\log^2 \log^2 n) = O((\log^2 \log^2 n) \log(\log^2 \log^2 n)) = O((\log^2 \log^2 n) \log\log\log n)$, so the total update time is $O(n \log\log\log n)$. Since $g(n) = O(1)$, the query time is constant. ◀

6 $O(n)$ Time Algorithm

In this section we finally show an algorithm that runs in $O(n)$ time. Observe that in Lemma 12, we run the assumed decremental algorithm on graphs of size $\log^2 \log^2 n$. However, the number of all such graphs is so small, that we may precompute all necessary connectivity information for all of them.

▶ **Lemma 14.** *Let w be the word size and $\log n \leq w$. After preprocessing in $o(n)$ time, we may repeatedly initialize and run algorithms for decremental maintenance of connected components in graphs of size $t = O(\log^2 \log n)$. These algorithms may be given a set of vertices V_e, and maintain the cc-identifiers of vertices of V_e explicitly. An algorithm for a graph of size t runs in $O(t + |V_e| \log t)$ time and may return the cc-identifier of every vertex in $O(1)$ time.*

Proof. We will call the set V_e the *explicit set*. The state of the algorithm is uniquely described by the current set of edges in the graph and the explicit set. There are $2^{t(t-1)/2}$ labeled undirected graphs on t vertices (including non-planar graphs) and $O(2^t)$ possible explicit sets. Thus, there are $O(2^{t^2})$ possible states, which, for $t = O(\log^2 \log n)$ gives $2^{O(\log^4 \log n)} = 2^{o(\log n)} = o(n)$. In particular, each state can be encoded as a binary string of length $O(\log^4 \log n)$ which fits in a single machine word.

For each state, we precompute cc-identifiers. Moreover, for each pair of state and an edge to be deleted, we compute the changes to the cc-identifiers of vertices in the explicit set. Observe that if the edge deletion is critical, we simply need to compute the set of vertices in the smaller out of the two connected components that are created and store the intersection of this set and V_e. These vertices should be assigned new, unique cc-identifiers.

We encode the graph by a binary word of length $O(\log^4 \log n)$, where each bit represents an edge between some pair of vertices. Thus, when an edge is deleted, we may compute the new state of the algorithm in constant time by switching off a single bit. For any planar graph and any sequence of deletions, the number of changes of cc-identifiers of vertices of V_e is $O(|V_e| \log n)$ (using the analysis similar to the one from the proof of Lemma 3). The query time is constant, since the cc-identifiers are maintained explicitly. For each of the $2^{O(\log^4 \log n)}$ states, we require $O(\log^4 \log n)$ preprocessing time. Thus, the preprocessing time is $o(n)$. ◀

We may now apply Lemma 12 to the algorithm of Lemma 14 to obtain the main result of this paper.

▶ **Theorem 15.** *There exists a decremental connectivity algorithm for planar graphs that supports updates in $O(n)$ total time and answers queries in constant time.*

7 Conclusion and Open Problems

We have shown a reduction from the decremental connectivity problem in planar graphs to incremental connectivity. As a result, we obtain an algorithm for decremental connectivity that processes all updates in optimal $O(n)$ time and answers queries in constant time. This shows that the total time complexity of the deceremental problem is not $\Omega(n \log n)$, which seemed to be a natural bound. In other words we have shown that a lower bound of $\Omega(n \log n)$, that would be an analogous to the lower bound in [14], cannot hold for decremental algorithms in planar graphs. We actually conjecture that even for general graphs with $O(n)$ edges there exists an $o(n \log n)$ time decremental algorithm.

An interesting question would be to study the worst-case time complexity of decremental connectivity in planar graphs, which has not been fully understood yet. And, contrary to the incremental problem, no nontrivial lower bounds are known.

References

1. Stephen Alstrup, Jens P. Secher, and Maz Spork. Optimal on-line decremental connectivity in trees. *Inf. Process. Lett.*, 64(4):161–164, 1997.
2. David Eppstein, Zvi Galil, and Giuseppe F Italiano. *Improved sparsification.* Information and Computer Science, University of California, Irvine, 1993.
3. David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44:669–696, 1997.
4. David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification: I. Planarity testing and minimum spanning trees. *J. Comput. Syst. Sci.*, 52(1):3–27, 1996.
5. David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert Endre Tarjan, Jeffery Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33–54, 1992.
6. Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985.
7. Jens Gustedt. Efficient union-find for planar graphs and other sparse graph classes. *Theoretical Computer Science*, 203(1):123 – 141, 1998.
8. Monika R. Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, July 1999.
9. Monika Rauch Henzinger and Michael L. Fredman. Lower bounds for fully dynamic connectivity problems in graphs. *Algorithmica*, 22(3):351–362, 1998.
10. Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.
11. Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 1131–1142. SIAM, 2013.
12. Sanjeev Khanna, editor. *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013.* SIAM, 2013.
13. Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 505–514. ACM, 2013.
14. Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35(4):932–963, 2006.
15. Johannes A. La Poutré. Lower bounds for the union-find and the sp;it-find problem on pointer machines. *J. Comput. Syst. Sci.*, 52(1):87–99, 1996.
16. Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.
17. Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. Comput. Syst. Sci.*, 18(2):110–127, 1979.
18. Mikkel Thorup. Decremental dynamic connectivity. *J. Algorithms*, 33(2):229–243, 1999.
19. Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 343–350. ACM, 2000.

20 Freek van Walderveen, Norbert Zeh, and Lars Arge. Multiway simple cycle separators and I/O-efficient algorithms for planar graphs. In Khanna [12], pages 901–918.
21 Christian Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In Khanna [12], pages 1757–1769.