On-line and Dynamic Shortest Paths through Graph Decompositions (Preliminary Version) [*]

Hristo N. Djidjev [1] Grammati E. Pantziou [2,3] Christos D. Zaroliagis [3,4]

March 24, 1994

(1) Department of Computer Science, Rice University, P.O. Box 1892, Houston, TX 77251, USA
(2) Department of Mathematics and Computer Science, Dartmouth College, Hanover NH 03755, USA
(3) Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
(4) Max-Plank Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany

Abstract

We describe algorithms for finding shortest paths and distances in a planar digraph which exploit the particular topology of the input graph. We give both sequential and parallel algorithms that work on a dynamic environment, where the cost of any edge can be changed or the edge can be deleted. For outerplanar digraphs, for instance, the data structures can be updated after any such change in only $O(\log n)$ time, where n is the number of vertices of the digraph. The parallel algorithms presented here are the first known ones for solving this problem. Our results can be extended to hold for digraphs of genus $o(n)$.

1 Introduction

There has been a growing interest in dynamic graph problems in the recent years [1, 9, 16, 18, 22]. The goal is to design efficient data structures that not only allow giving fast answers to a series of queries, but that can also be easily updated after a modification of the input data. Such an approach has immediate applications to a variety of problems which are of both theoretical and practical value. Dynamic algorithms for graph problems have applications in simulation of traffic networks, high level languages for incremental computations, incremental data flow analysis, interactive network design, maintenance of maximum flow in a network [2, 26, 27, 28], just to name a few.

Let G be an n-vertex digraph with real valued edge costs but no negative cycles. The *length* of a path p in G is the sum of the costs of all edges of p and the *distance* between two vertices v, w of G is the minimum length of a path between v and w. The path of minimum length between v, w is called a *shortest path* between v and w. Finding shortest path information in graphs is an important and intensively studied graph problem with many applications. Recent papers [3, 6, 11, 12, 13, 15, 19, 21, 23, 24] investigate the problem for different classes of input graphs and models of computation. All of the above-mentioned results, however, relate to the static version of the problem, i.e. the graph and the costs on its edges do not change over time. In contrast, we consider here a

[*]This work is partially supported by the EC ESPRIT Basic Research Action No. 7141 (ALCOM II) and by the EC Cooperative Action IC-1000 (project ALTEC). The work of the second author is also partially supported by the NSF postdoctoral fellowship No. CDA-9211155. Email: hristo@cs.rice.edu, pantziou@cs.dartmouth.edu, zaro@mpi-sb.mpg.de.

dynamic environment, where edges can be deleted and their costs can be modified. More precisely, we investigate the following *on-line and dynamic shortest path problem:* given G (as above), build a data structure that will enable fast on-line shortest path or distance queries. In case of edge deletion or edge cost modification of G, update the data structure in appropriately short time.

The dynamic version of the shortest paths problem has clearly a lot of applications. For example, it is one of the fundamental problems that one has to solve in order to get a solution to the so called *vehicle routing* problem. This problem is the following. Assume that you are in a vehicle located somewhere in the traffic network of a city, and you want to know at any time the shortest route to the nearest hospital, drugstore, hotel, etc, or to find the shortest route or distance to a specific place. Note that the underlying traffic network may change dynamically: some roads may be closed (because of works or accidents), certain roads may change behaviour at rush hours, or some other ones may change direction. This problem gives rise to the development of a software system loaded on a fast computer, where a number of operators receive on-line queries from the drivers, get the appropriate answers and transmit them back to the drivers.

There are a few previously known algorithms for the dynamic shortest path problem. For general digraphs, the best previous algorithms in the case of updating the data structure after edge insertions/deletions were due to [8] and require $O(n^2)$ update time after an edge insertion and $O(n^2 \log n)$ update time after an edge deletion. Some improvements of these algorithms have been achieved in [1] with respect to the amortized cost of a sequence of edge insertions, if the edge costs are integers. For the case of planar digraphs the best dynamic algorithms are due to [10] for the case of edge cost updates. The preprocessing time and space is $O(n \log n)$ ($O(n)$ space can be achieved, if the computation is restricted to finding distances only.) A single-pair query can be answered in $O(n)$ time, while a single-source query takes $O(n\sqrt{\log\log n})$ time. An update operation to this data structure, after an edge cost modification or deletion, can be performed in $O(\log^3 n)$ time. In parallel computation we are not aware of any previous results related to dynamic structures for maintaining shortest path information in the case of edge cost updates. On the other hand, efficient data structures for answering very fast on-line shortest path or distance queries for the sequential and the parallel models of computation have been proposed in [6, 14], but they do not support dynamization.

In this paper, we give efficient algorithms for solving the on-line and dynamic shortest path problem in planar digraphs which are parameterized in terms of a topological measure q of the input digraph. Our main result is the following (Section 4): *Given an n-vertex planar digraph G with real-valued edge costs but no negative cycles, there exists an algorithm for the on-line and dynamic shortest path problem on G that supports edge cost modification and edge deletion with the following performance characteristics: (i) preprocessing time and space $O(n + q \log q)$; (ii) single-pair distance query time $O(q + \log n)$; (iii) single-pair shortest path query time $O(L + q + \log n)$ (where L is the number of edges of the path); (iv) single-source shortest path tree query time $O(n + q\sqrt{\log\log q})$; (v) update time (after an edge cost modification or edge deletion) $O(\log n + \log^3 q)$. In the case where the computation is restricted to finding distances only the space can be reduced to $O(n)$.*

Here q is a topological measure of the input planar digraph G and is proportional to the cardinality of a minimum number of faces covering all vertices of G (among all embeddings of G in the plane). Our results are improvements over the best previous ones, in all cases where $q = o(n)$. In the case where G is outerplanar ($q = 1$) our preprocessing time and space are optimal (linear) and the distance query and the update time are logarithmic. Also, our algorithms seem to be very efficient for the class of all appropriately *sparse* graphs. As it has been established in [7, 20] random $G_{n,p}$ graphs with threshold function $1/n$ are with probability one planar and have *expected value* for q equal to $O(1)$. Then, our algorithms achieve the following expected performance for the above class of graphs: $O(n)$ preprocessing time and space, $O(\log n)$ (resp. $O(L + \log n)$) distance (resp. shortest path) query time, $O(n)$ single-source query time, and $O(\log n)$ update time. For comparison, see the best previous results of [10] stated above. Our solution is based on the following ideas:

(a) The input planar digraph is decomposed into a number, $O(q)$, of outerplanar subgraphs (called *hammocks*) satisfying certain separator conditions [13, 24].

(b) A decomposition strategy based on graph separators is employed for the efficient solution of the problem for the case of *outerplanar* digraphs (Section 2).

(c) A data structure is constructed during the decomposition of the outerplanar digraph and is updated after each edge cost modification or edge deletion (Section 3). This data structure contains information about the shortest paths between properly chosen $\Theta(n)$ pairs of vertices. It also has the property that the shortest path between any pair of vertices is a composition of $O(\log n)$ of the predefined paths and that any edge of the graph belongs to $O(\log n)$ of those paths (n is the size of the outerplanar digraph).

We mention also the following extensions and generalizations to our results discussed in the paper.

(i) We have constructed parallel versions of our algorithms for the CREW PRAM model of parallel computation (Section 5). There have been no previous parallel algorithms for the dynamic and on-line version of the shortest path problem

(ii) Our algorithms can detect a negative cycle, either if it exists in the initial graph, or if it is created after an edge cost modification.

(iii) Using the ideas of [12, 19], our results can be extended to hold for any digraph whose genus is $o(n)$. In such a case an embedding of the graph does not need to be provided by the input (Section 5).

(iv) Although our algorithms do not directly support edge insertion, they are so fast that even if the preprocessing algorithm is run from scratch after any edge insertion, they still provide better performance compared with [8]. Moreover, our algorithms can support a special kind of edge insertion, called edge *re-insertion.* That is, we can insert any edge that has previously been deleted within the resource bounds of the update operation.

The paper is organized as follows. Section 2 contains preliminaries. In Section 3 we consider sequential algorithms for outerplanar digraphs and in Section 4 we obtain our basic results for planar digraphs. In Section 5 we describe a parallel implementation and some generalizations of our results.

2 Preliminaries

Let $G = (V(G), E(G))$ be a connected planar n-vertex digraph with real edge costs but no negative cycles. A *separation pair* is a pair (x, y) of vertices whose removal separates G into two disjoint connected subgraphs G_1 and G_2. We add the vertices x, y and the edges $\langle x, y\rangle$ and $\langle y, x\rangle$ to both G_1 and G_2. Let $0 < \alpha < 1$ be a constant. An *α-separator* S of G is a pair of sets $(V(S), D(S))$, where $D(S)$ is a set of separation pairs and $V(S)$ is the set of the vertices of $D(S)$, such that the removal of $V(S)$ leaves no connected component of more than αn vertices. We will call the separation vertices (pairs) of S that belong to any such resulting component H and separate it from the rest of the graph separation vertices (pairs) *attached to H*. It is well known that if G is outerplanar then there exists a 2/3-separator of G which is a single separation pair. Also, given an n-vertex outerplanar digraph G_o and a set M of vertices of G_o, *compressing G_o with respect to M* means constructing a new outerplanar digraph of $O(|M|)$ size that contains M and such that the distance between any pair of vertices of M in the resulting graph is the same as in G_o [13, 24]. (In our algorithms the size of M will be $O(1)$.)

Definition 2.1 *Let G_o be an outerplanar digraph and S be an α-separator of G_o that divides G_o into connected components one of which is G. Let $p = (p_1, p_2)$ be a separation pair of G. Construct a graph $SR(G)$ as follows: remove the vertices of p from G, compress each resulting subgraph K with respect* *to* $(V(S) \cup \{p_1, p_2\}) \cap V(K)$ *and join the resulting graphs at vertices* p_1, p_2. *We call* $SR(G)$ *the* sparse representative *of* G.

A *hammock decomposition* is a decomposition of G into certain outerplanar digraphs called *hammocks*. This decomposition is defined relative to a given set of faces that cover all vertices of G. Let q be the minimum number of such faces (among all embeddings of G). It has been proved in [13, 24] that a planar digraph G can be decomposed into $O(q)$ hammocks either in $O(n)$ sequential time, or in $O(\log n \log^* n)$ parallel time using $O(n)$ CREW PRAM processors. Also, by [12, 19], we have that an embedding of G does not need to be provided by the input in order to compute a hammock decomposition of $O(q)$ hammocks. Hammocks satisfy the following properties: (i) each hammock has at most *four* vertices in common with any other hammock (and therefore with the rest of the graph), called *attachment vertices*; (ii) the hammock decomposition spans all the edges of G, i.e. each edge belongs to exactly one hammock; and (iii) the number of hammocks produced is order of the minimum possible among all possible decompositions. This decomposition allows us to *reduce* the solution of a given problem Π on a planar digraph into a solution of Π on an outerplanar digraph.

In the sequel, we can assume w.l.o.g. that G_o is a biconnected n-vertex outerplanar digraph. Note that if G_o is not biconnected we can add an appropriate number of additional edges of very large cost in order to convert it into a biconnected outerplanar digraph (see [13, 24]).

2.1 Constructing a separator decomposition

We describe an algorithm that generates a decomposition of G_o (by finding successive separators in a recursive way) that will be used in the construction of a suitable data structure for maintaining shortest path information in G_o. Our goal will be that, at each level of recursion, (i) the sizes of the connected components resulting after the deletion of the previously found separator vertices are appropriately small, and (ii) the number of separation vertices attached to each such component is $O(1)$. The following algorithm finds such a partitioning and constructs the associated *separator tree*, $ST(G_o)$, used to support binary search in G_o.

```
ALGORITHM Sep_Tree($G_o, ST(G_o)$)
BEGIN
    1. If $|V(G_o)| \leq 4$, then halt. Else let $S$ denote the set of separation pairs found during all previous
iterations. (Initially $S = \emptyset$.) Let $n_{sep}$ denote the number of separation pairs of $S$ attached to $G_o$.
    1.1. If $n_{sep} \leq 3$, then let $p = \{p_1, p_2\}$ be a separation pair of $G_o$ that divides $G_o$ into two subgraphs
$G_1$ and $G_2$ with no more than $2n/3$ vertices each.
    1.2. Otherwise ($n_{sep} > 3$), let $p = \{p_1, p_2\}$ be a separation pair that separates $G_o$ into subgraphs
$G_1$ and $G_2$ each containing no more than 2/3 of the number of separation pairs attached to $G_o$.
    2. Add $p$ to $S$ and run this algorithm recursively on $G_i$ for $i = 1, 2$. Create a separator tree
$ST(G_o)$ rooted at a new node $v$ associated with $p$ and $G_o$, whose children are the roots of $ST(G_1)$
and $ST(G_2)$.
END.
```

Observe that the nodes of $ST(G_o)$ are associated with subgraphs of G_o which we will call *descendant subgraphs.* With each descendant subgraph a distinct separation pair is associated. From the description of the algorithm, the following fact follows.

Lemma 2.1 *Any descendant subgraph* G *of* G_o *at level* i *in* $ST(G_o)$ *has no more than 4 separation pairs attached to it and the number of its vertices is no more than* $(2/3)^i n$.

Algorithm Sep_Tree can be easily implemented to run in $O(n \log n)$ time and $O(n)$ space. We show by the following lemma that there exists a more efficient implementation in $O(n)$ time and space.

Lemma 2.2 *Algorithm Sep_Tree($G_o, ST(G_o)$) can be implemented to run in $O(n)$ time and $O(n)$ space. The depth of the resulting separator tree $ST(G_o)$ is $O(\log n)$.*

Proof: Each recursive step of Algorithm Sep_Tree takes $O(1)$ time plus time necessary to find the separation pair p. Thus the total time needed by all steps of the algorithm is $O(n)$ plus the time required to find all separation pairs p. Furthermore, notice that finding all separation pairs from Step 1.2 can be implemented in $O(n)$ time, if we keep for each component K into which S divides G_o a list of the separation pairs attached to K. We can trivially update this list in $O(1)$ time when a new separation pair is attached to K, since we don't allow the number of the separation pairs in any list to exceed 4. Therefore we need to show that the time required to find all separation pairs p from Step 1.1 is linear. We construct the dual graph of G_o (excluding the outer face), which is a tree. By using the data structure of [25] for dynamic trees we can find any separation pair in $O(\log n)$ time. Then the time $T(n)$ needed to find all separation pairs satisfies the recurrence $T(n) \leq \max\{T(n_1) + T(n_2) \mid n_1 + n_2 = n, \;\; n_1, n_2 \leq 2n/3\} + O(\log n), \; n > 1$, which has a solution $T(n) = O(n)$. Since $ST(G_o)$ is a balanced tree, its depth is obviously logarithmic. ∎

3 Dynamic algorithms for outerplanar digraphs

In this section we will give algorithms for solving the on-line and dynamic shortest path problem for the special case of outerplanar digraphs. We will use these algorithms in Section 4 for solving shortest path problems for general planar digraphs. Throughout this section we denote by G_o an n-vertex biconnected outerplanar digraph.

3.1 The data structures and the preprocessing algorithm

The data structures used by our algorithms are the following:

(I) The separator tree $ST(G_o)$. Each node of $ST(G_o)$ is associated with a descendant subgraph G of G_o along with its separation pair as determined by algorithm Sep_Tree and contains a pointer to the sparse representative $SR(G)$ of G.

(II) The sparse representative $SR(G)$ for all graphs G of $ST(G_o)$. According to Definition 2.1, $SR(G)$ consists of the union of the compressed versions of G_1 and G_2 with respect to the separation pairs attached to G and the separation pair dividing G, where G_1 and G_2 are the children of G in $ST(G_o)$. Therefore the size of $SR(G)$ is $O(1)$. Note also that: (a) since the size of $SR(G)$ is $O(1)$, we can compute the distances between the vertices of $SR(G)$ (and therefore between the separator vertices attached to G) in constant time; (b) for each leaf of $ST(G_o)$ we have that $SR(G) \equiv G$, since in this case G is of $O(1)$ size.

In the following sections we will use the properties of the separator decomposition to show that the shortest path information encoded in the sparse representatives of the descendant subgraphs of G_o is sufficient to compute the distance between any 2 vertices of G_o in $O(\log n)$ time and that all sparse representatives can be updated after any edge cost modification also in $O(\log n)$ time. We next give an algorithm which constructs the above data structures in linear time.

```
ALGORITHM Pre_Outerplanar($G_o$)
BEGIN
1. Construct a separator tree $ST(G_o)$ using algorithm Sep_Tree($G_o, ST(G_o)$).
2. Compute the sparse representative $SR(G_o)$ of $G_o$ as follows.
   for each child $G$ of $G_o$ in $ST(G_o)$ do
(a) if $G$ is a leaf of $ST(G_o)$ then $SR(G) = G$
```

```
    else find $SR(G)$ by running Step 2 recursively on $G$.
(b) Construct the sparse representative of $G_o$ as described in Definition 2.1 by using the sparse
    representatives of the children of $G_o$.
END.
```

Lemma 3.1 *Algorithm Pre_Outerplanar(G_o) runs in $O(n)$ time and uses $O(n)$ space.*

Proof: Step 1 needs $O(n)$ sequential time and space by Lemma 2.2. Let $P(n)$ be the time required by Step 2. Then $P(n)$ satisfies the recurrence $P(n) \leq \max\{P(n_1) + P(n_2) \mid n_1 + n_2 = n, \;\; n_1, n_2 \leq 2n/3\} + O(1), \;\; n > 1$, which has a solution $P(n) = O(n)$. The space required is proportional to the size of $ST(G_o)$ since each sparse representative has $O(1)$ size. Therefore the space needed by the above data structures is $O(|ST(G_o)|) = O(n)$. The bounds follow. ∎

3.2 The single-pair query algorithm

We will first briefly describe the idea of the query algorithm for finding the distance between any two vertices v and z of G_o. The algorithm proceeds as follows. First search $ST(G_o)$ to find a descendant subgraph G of G_o such that the separation pair $p = (p_1, p_2)$ in G separates v from z. Let $d(v, z)$ denote the distance between v and z. Then, obviously,

$$d(v, z) = \min\{d(v, p_1) + d(p_1, z), d(v, p_2) + d(p_2, z)\}. \tag{1}$$

Hence, it suffices to compute the distances $d(v, p_1)$, $d(p_1, z)$, $d(v, p_2)$ and $d(p_2, z)$.

Now we will address the question of what kind of shortest path information the sparse representatives provide. Let $s = (s_1, s_2)$ be any separation pair attached to G. Let s divides some descendant graph H of G_o into subgraphs H_1 and H_2, where H_1 has no other common vertices with G except for s_1 and s_2. If H is a parent of G, we call s a *parent* separation pair for G. The distance from s_1 to s_2 in $SR(G)$ is, by the preprocessing algorithm, equal to the distance between s_1 and s_2 in G. However, the distance from s_1 to s_2 in G might be different from the distance between these vertices in G_o, if s is a parent separation pair. Note that G can have no more than one parent separation pair.

Let $D(s_1, s_2)$ denote the set of the (two) distances from s_1 to s_2 and from s_2 to s_1 in G_o. To compute $D(s_1, s_2)$ we apply the following two-step algorithm: (i) If H has no parent separation pair, then compute $D(s_1, s_2)$ directly from $SR(H)$; otherwise compute recursively $D(s'_1, s'_2)$, where (s'_1, s'_2) is the parent separation pair of H. (ii) Use $D(s'_1, s'_2)$ and $SR(G)$ to compute (in $O(1)$ time) $D(s_1, s_2)$. Obviously this procedure requires $O(\log n)$ time.

Next we describe the query algorithm. Let v' be a vertex that belongs to the same descendant subgraph of G_o that is a leaf of $ST(G_o)$ and that contains v. Let $p(v)$ be the pair of vertices v, v'. Similarly define a pair of vertices $p(z)$ that contains z and a vertex z' which belongs to the leaf of $ST(G_o)$ containing z. For any two pairs p' and p'' of vertices, let $D(p', p'')$ denote the set of all four distances from a vertex from p' to a vertex from p''. Then (1) shows that $D(p(v), p(z))$ can be found in constant time, given $D(p(v), p)$ and $D(p, p(z))$. The following recursive algorithm is based on the above fact.

```
ALGORITHM Dist_Query_Outerplanar($G_o, v, z$)
BEGIN
    1. Search $ST(G_o)$ (starting from the root) to find pairs of vertices $p(v)$ and $p(z)$ as defined above.
    2. Search $ST(G_o)$ (starting from the root) to find a descendant graph $G$ of $G_o$ such that the
separation pair $p$ associated with $G$ separates $p(v)$ and $p(z)$ in $G$.
```

3. Find the distances between the vertices of the parent separation pair of G as described above (if G has such a separation pair).

4. Find $D(p(v), p)$ as follows:

4.1. Search $ST(G_o)$ (starting from G) to find a descendant graph G' of G such that the separation pair p' associated with G' separates $p(v)$ and p in G'.

4.2. If G' is a leaf of $ST(G_o)$, then determine $D(p(v), p')$ directly in constant time.

4.3. If G' is not a leaf then find $D(p(v), p')$ by executing Step 4 recursively with $p := p'$, $G := G'$, and then find $D(p(v), p)$ by using (1). Note that $D(p', p)$ can be taken from $SR(G')$.

5. Find $D(p, p(z))$ as in Step 4.

6. Use $D(p(v), p)$, $D(p, p(z))$, and (1) to determine $D(p(v), p(z))$.

END.

Lemma 3.2 *Algorithm Dist_Query_Outerplanar(G_o, v, z) finds the distance between any two vertices v and z of an n-vertex outerplanar digraph G_o in $O(\log n)$ time.*

Proof: The correctness follows from the description of the algorithm. Searching $ST(G_o)$ in Steps 1 and 2 takes in total $O(\log n)$ time by Lemma 2.2. Step 3 takes $O(\log n)$ time by the above analysis. Let $Q(l)$ be the maximum time necessary to compute $D(p(v), p)$, where l is the level of G in $ST(G_o)$ and l_{max} is the maximum level of $ST(G_o)$. Then from the description of the algorithm $Q(l) \leq Q(l+1) + O(1) \quad for\ l < l_{max}$, which gives $Q(l) = O(l) = O(\log n)$. Similarly, the time necessary for Step 5 is $O(\log n)$. Thus the total time needed by the algorithm is $O(\log n)$. ∎

Algorithm Dist_Query_Outerplanar can be modified in order to answer path queries. The additional work (compared with the case of distances) involves uncompressing the shortest paths corresponding to edges of the sparse representatives of the graphs from $ST(G_o)$. Uncompressing an edge from a graph $SR(G)$ involves a traversal of a subtree of $ST(G_o)$, where at each step an edge is replaced by two new edges each possibly corresponding to a compressed path. Obviously this subtree will have no more than L leaves, where L is the number of the edges of the output path. Then the traversal time can not exceed the number of the vertices of a binary tree with L leaves in which each internal node has exactly 2 children. Any such tree has $2L - 1$ vertices. Thus the following claim follows.

Lemma 3.3 *The shortest path between any two vertices v and z of an n-vertex outerplanar digraph G_o can be found in $O(\log n + L)$ time, where L is the number of the edges of the path.*

3.3 The update algorithm

In the sequel, we will show how we can update our data structures for answering on-line shortest path and distance queries in outerplanar digraphs, in the case where an edge cost is modified. (Note that updating after an edge deletion is equivalent to the updating of the cost of the particular edge with a very large weight, such that this edge will not be used by any shortest path.) The algorithm for updating the cost of an edge e in an n-vertex outerplanar digraph G_o is based on the following idea: the edge will belong to at most $O(\log n)$ subgraphs of G_o, as they are determined by the Sep_Tree algorithm. Therefore, it suffices to update (in a bottom-up fashion) the sparse representatives of those subgraphs that are on the path from the subgraph G containing e (where G is a leaf of $ST(G_o)$) to the root of $ST(G_o)$. Let $parent(G)$ denote the parent of a node G in $ST(G_o)$, and $\hat{G}$ denote the sibling of a node G in a $ST(G_o)$. Note that $G \cup \hat{G} = parent(G)$ and $SR(G) \cup SR(\hat{G}) \supset SR(parent(G))$. The algorithm for the update operation is the following.

ALGORITHM Update_Outerplanar$(G_o, e, w(e))$
BEGIN

```
1. Find a leaf $G$ of $ST(G_o)$ for which $e \in E(G)$.
2. Update the cost of $e$ in $G$ with the new cost $w(e)$.
3. If $e$ belongs also to $\hat{G}$ then update the cost of $e$ in $\hat{G}$.
4. While $G \neq G_o$ do
        (a) Update $SR(parent(G))$ using the new versions of $SR(G)$ and $SR(\hat{G})$.
        (b) $G := parent(G)$.
END.
```

Lemma 3.4 *Algorithm Update_Outerplanar updates after an edge cost modification the data structures created by the preprocessing algorithm in* $O(\log n)$ *time.*

Proof: Since by Lemma 2.2 the depth of $ST(G_o)$ is $O(\log n)$, Step 1 obviously can be implemented in logarithmic time by doing a binary search on $ST(G_o)$. Steps 2 and 3 require $O(1)$ time. Finally, the number of iterations in Step 4 is $O(\log n)$ and each iteration takes constant time because the size of $SR(G)$ for any descendant subgraph G of G_o is $O(1)$. ∎

3.4 Handling of negative cycles and summary of the results

The initial digraph G_o can be tested for existence of a negative cycle in $O(n)$ time by [19]. Assume now that G_o does not contain a negative cycle and that the cost $c(v, w)$ of an edge $\langle v, w\rangle$ in G_o has to be changed to $c'(v, w)$. We must check if this change does not create a negative cycle. We modify our algorithms in the following way. Before running the Update_Outerplanar algorithm, run the algorithm Dist_Query_Outerplanar to find the distance $d(w, v)$. If $d(w, v) + c'(v, w) < 0$, then halt and announce non-acceptance of this edge cost modification. Otherwise, continue with the original update algorithms. Clearly, the above procedures for testing the initial digraph and testing the acceptance of the edge cost modification do not affect the resource bounds of our preprocessing or of our update algorithm, respectively. Our results, in the case of outerplanar digraphs, can be summarized in the following theorem.

Theorem 1 *Given an n-vertex outerplanar digraph G with real-valued edge costs but no negative cycles, there exists an algorithm for the on-line and dynamic shortest path problem on G that supports edge cost modification and edge deletion with the following performance characteristics: (i) preprocessing time and space $O(n)$; (ii) single-pair distance query time $O(\log n)$; (iii) single-pair shortest path query time $O(L + \log n)$ (where L is the number of edges of the path); (iv) update time (after an edge cost modification or edge deletion) $O(\log n)$.*

4 Dynamic algorithms for planar digraphs

The algorithms for maintaining all pairs shortest paths information in a planar n-vertex digraph G are based on the hammock decomposition idea and on the algorithms of the previous section. Let q be the minimum cardinality of a hammock decomposition of G. The preprocessing algorithm for G (say *Pre_Planar*) consists of the following steps. (1) Find a hammock decomposition of G into $O(q)$ hammocks. (2) Run the algorithm Pre_Outerplanar(H) in each hammock H. (3) Compress each hammock H with respect to its attachment vertices. This results in a planar digraph G_q which is of size $O(q)$. (4) Run the preprocessing algorithm of [10] in G_q.

Lemma 4.1 *Algorithm Pre_Planar runs in* $O(n + q \log q)$ *time and uses* $O(n + q \log q)$ *space.*

Proof: Step (1) can be implemented in $O(n)$ time by [13]. The resource bounds of Step (2) come from Theorem 1. Step (3) takes $O(1)$ time per hammock H (since by Step (2) we have already computed $SR(H)$), or $O(q)$ time in total. Since G_q is of size $O(q)$, Step (4) takes $O(q \log q)$ time and space by [10]. The bounds follow. ∎

The update algorithm is straightforward. Let e be the edge whose cost has been modified. There are two data structures that should be updated. The first one concerns the hammock H where e belongs to. This can be done by the algorithm Update_Outerplanar in $O(\log n)$ time. Note that this algorithm provides G_q with a new updated sparse representative of H, from which the compressed version of H (with respect to its attachment vertices) can be constructed in $O(1)$ time. The second data structure is that of the digraph G_q and can be updated in $O(\log^3 q)$ time by [10]. Therefore, we have the following lemma.

Lemma 4.2 *The data structures created by algorithm Pre_Planar can be updated in the case of an edge cost modification in $O(\log n + \log^3 q)$ time.*

A single-pair query between any two vertices v and z can be answered as follows (using the above data structures). If v and z do not belong to the same hammock, then their distance $d(v, z) = \min_{i,j}\{d(v, a_i) + d(a_i, a'_j) + d(a'_j, z)\}$ where a_i and a'_j respectively are the attachment vertices of the hammocks in which v and z belong to. If both v and z belong to the same hammock H, then note that the shortest path between them does not necessarily have to stay in H. Hence, first compute (using algorithm Dist_Query_Outerplanar) their distance $d_H(v, z)$ inside H. After that compute $d_{ij}(v, z) = \min_{i,j}\{d(v, a_i)+d(a_i, a_j)+d(a_j, z)\}$. Clearly, $d(v, z) = \min\{d_H(v, z), d_{ij}(v, z)\}$. (A shortest path query can be computed similarly.) The following lemma holds.

Lemma 4.3 *The shortest path (resp. distance) between any two vertices of G can be computed in $O(L + q + \log n)$ (resp. $O(q + \log n)$) time, where L is the number of the edges of the path.*

Proof: Let us analyse the time complexity of the above algorithm. We need $O(q)$ time for queries in G_q [10] (for computing a distance or a compressed shortest path) and $O(\log |H|)$ or $O(L_H + \log |H|)$ time respectively for distance and path queries in each hammock H (Theorem 1), where $|H|$ is the size of H and L_H is the portion (in number of edges) of the shortest path contained in H. This results in a total of $O(q + \log n)$ or $O(L + q + \log n)$ over all hammocks, where $L = \sum_H L_H$. ∎

Therefore, the results for planar digraphs can be summarized in the following theorem.

Theorem 2 *Let G be an n-vertex planar digraph with real-valued edge costs but no negative cycles and let q be the minimum cardinality of a hammock decomposition of G. There exists an algorithm for the on-line and dynamic shortest path problem on G that supports edge cost modification and edge deletion with the following performance characteristics: (i) preprocessing time and space $O(n + q \log q)$; (ii) single-pair distance query time $O(q + \log n)$; (iii) single-pair shortest path query time $O(L + q + \log n)$ (where L is the number of edges of the path); (iv) update time (after an edge cost modification or edge deletion) $O(\log n + \log^3 q)$. In the case where the computation is restricted to finding distances only, the space can be reduced to $O(n)$.*

The case of negative edge costs is handled in a similar way with that of outerplanar digraphs. The initial digraph can be tested for a negative cycle in $O(n + q^{1.5} \log q)$ time [19]. The procedure for accepting or not an edge cost modification is similar to the one described for outerplanar digraphs.

5 Parallelization and further results

In this section we describe briefly an efficient parallel implementation of our algorithms (Subsection 5.1) and some further results of our approach for solving the dynamic shortest path problem (Subsection 5.2).

5.1 Parallel algorithms for dynamic shortest paths

We use the CREW PRAM model of computation. We will start with the case of outerplanar graphs, showing how the algorithms from Section 3 can be implemented in parallel. Consider first the preprocessing algorithm. Step 1 can easily be implemented in $O(\log n)$ time and $O(n \log n)$ work. The total work required by Step 2 is described by the recurrence for $P(n)$ in the proof of Lemma 3.1. The parallel time of Step 2 satisfies the recurrence $T_p(n) = T_p(n/2) + O(1)$, whose solution is $T_p(n) = O(\log n)$. For the single-pair query notice the following. Using algorithm Dist_Query_Outerplanar we can determine in $O(\log n)$ time which is the subtree of $ST(G_o)$ that should be traversed in order to output the path. This subtree has at most L leaves and size $O(L)$. Thus we can output the path in $O(\log n)$ time and $O(L)$ work. Therefore, we have the following.

Theorem 3 *Given an n-vertex outerplanar digraph G with real-valued edge costs but no negative cycles, there exists a CREW PRAM algorithm for the on-line and dynamic shortest path problem on G that supports edge cost modification and edge deletion with the following performance characteristics: (i) preprocessing time $O(\log n)$ with $O(n \log n)$ work and $O(n)$ space; (ii) single-pair distance query time $O(\log n)$ and $O(\log n)$ work; (iii) single-pair shortest path query time $O(\log n)$ and $O(L + \log n)$ work (where L is the number of edges of the path); (iv) update time (after an edge cost modification or edge deletion) $O(\log n)$ and $O(\log n)$ work.*

In the case of planar digraphs we will first give a parallel CREW PRAM implementation of the approach used in [10] and then we will use it for implementing, querying and updating the data structures in G_q (recall Section 4). The algorithm of [10] is based on a topological division of a planar digraph G into regions [11]. We first need some definitions. A region R contains two types of vertices: interior vertices and boundary vertices. An interior vertex is contained in exactly one region while a boundary vertex is shared among at most three regions. Let $V(R)$ be the set of all vertices of R and $B(R)$ be the set of its boundary vertices. A *suitable r-division* of G [11] is a division of G into t regions R_i, $i = 1, 2, ..., t$, such that $r/2 < |V(R_i)| \le r$, $|B(R_i)| = O(\sqrt{r})$, $t \le \lceil n/|V(R_i)| \rceil = \lceil (c/r)n \rceil$, $1 \le c < 2$, and the total number of boundary vertices is $b = t \cdot |B(R_i)| = \Theta(n/\sqrt{r})$. We call a set S of vertices an *$(a, f(n))$-separator* of an n-vertex graph J, if S is an a-separator (recall the definition from Section 2) and $|V(S)| = O(f(n))$, where $f(n) = o(n)$ and $0 < a < 1$ is a constant. It is well known that any planar graph has a $(2/3, \sqrt{n})$-separator. An n^μ-separator decomposition, $0 < \mu < 1$, is a recursive decomposition of J using (a, n^μ)-separators, where subgraphs of size k have an (a, k^μ)-separator. We will also make use of the following recent result by Cohen [3]. If an n-vertex digraph J is provided with an n^μ-separator decomposition then shortest paths from s sources can be computed in $O(\log^2 n)$ time and $O(sn + sn^{2\mu})$ work. A preprocessing phase is needed which takes $O(\log^3 n)$ time and $O(n + n^{3\mu})$ work. In the case of planar digraphs (where an $n^{1/2}$-separator decomposition can be found in $O(\log^5 n)$ time and $O(n^{1+\varepsilon})$ work, for any arbitrarily small $(1/2) > \varepsilon > 0$, using the algorithm of [17]), we have that shortest paths from s sources are computed in $O(\log^5 n)$ time and $O(sn + n^{1.5})$ work. Now we are ready to discuss the CREW PRAM implementation of the algorithm in [10].

The preprocessing algorithm consists of the following steps: (1) Find a suitable r-division of G. (2) Compute inside each region R_i shortest paths between all vertices in $B(R_i)$. The main procedure used in the implementation of Step (1) is the algorithm of [17] for finding an $(2/3, \sqrt{n})$-separator in G. Hence, this step is implemented to run in $O(\log^5 n)$ time and $O(n^{1+\varepsilon})$ work. For Step (2) we run the algorithm of [3] which takes, for each region $O(\log^5 r)$ time and $O(r^{1.5})$ work, and in total $O(\log^5 r)$ time and $O(n\sqrt{r})$ work.

The single-pair query algorithm for computing the shortest path between a vertex v (belonging to a region R_v) and a vertex z (belonging to a region R_z) works as follows: (1) If v is an interior vertex in R_v, then compute shortest paths from v to every vertex in R_v. (2) Compute shortest paths from every vertex in $B(R_v)$ to all boundary vertices in G. (3) Compute shortest paths from all vertices in $B(R_z)$ to z. (4) The length $d(v, z)$ of the shortest path is given as the minimum among $d(v, b(R_v)) + d(b(R_v), b(R_z)) + d(b(R_z), z)$ over all $b(R_v)$ and $b(R_z)$, where $b(R_v)$ (resp., $b(R_z)$) is a boundary vertex of R_v (resp., R_z). (If $R_v \equiv R_z$ then compare this length with the one computed in step (1) and choose the minimum.) We have the following lemma.

Lemma 5.1 *A single-pair shortest path can be computed in $O(\log^3 b)$ time and $O(b^{1.5+\delta})$ work, for any $0 < \delta < 1/2$ arbitrarily small and $r = polylog(n)$.*

Proof: Steps (1) and (3) take $O(\log^5 r)$ time and $O(r^{1.5})$ work by [3]. Step (4) needs $O(\log r)$ time and $O(r)$ work. The most difficult part is the efficient parallelization of Step (2). We do it by constructing a new digraph G' as follows. Replace each region R in G with the complete digraph on the vertices of $B(R)$. Each edge of the complete digraph has a cost equal to the length of the shortest path between its endpoints in R. Thus G' has b vertices and it is not planar. Let S be an $(2/3, \sqrt{n})$-separator of G. Let S' be a set of vertices of G' constructed by the following rules: (a) if a vertex $u \in S$ is a boundary vertex of some region, then add u to S'; (b) if a vertex $u \in S$ is an interior vertex that belongs to region R, then add the set $B(R)$ to S'.

Claim: If $r = \text{polylog}(n)$, then S' as defined above, is an $(\beta, b^{1/2+x})$-separator of G', $0 < \beta < 1$ constant, and $0 < x < 1/6$ arbitrarily small.

Proof: By the construction of S', we have that $|S'| = O(\sqrt{nr})$. Also note that $O(\sqrt{nr}) = o((n/\sqrt{r})^{1/2+x})$, if $r = \text{polylog}(n)$ and $0 < x < 1/6$ can be arbitrarily small (for sufficiently large n). Therefore, it remains to prove that: (i) removal of S' disconnects G' and (ii) the largest of the components, say G'_1, contains at most βb vertices, $0 < \beta < 1$ a constant. Part (i) is obvious. For part (ii) assume for the moment that S consists only of boundary vertices. Then G_1 has $\leq (2/3)t$ regions. This is because a region in G is not split into two and if we apply the algorithm for producing a suitable r-division on G_1 (which has $\leq (2/3)n$ vertices), we will get $\leq \lceil (c/r)(2n/3) \rceil = (2/3)t$ regions. Hence, G_1 has $\leq (2/3)t|B(R)|$ boundary vertices which implies that G'_1 has $\leq (2/3)b$ vertices. Assume now that S does not consist only of boundary vertices. Consider a removal of S' from G. Then G_1 will have $\leq (2/3)n - O(\sqrt{nr}) \leq \theta n$ vertices, for some constant $0 < \theta < 1$. Hence G_1 will have at most $\lceil (c/r)\theta n \rceil = \theta t$ regions and G'_1 at most θb vertices. Therefore, in either case G'_1 has at most βb vertices, for some constant $0 < \beta < 1$. ∎

The above claim implies that a separator decomposition for G can be easily converted into a separator decomposition for G'. Hence, the algorithm of [3] applies, which means that Step (2) is implemented in $O(\log^3 b)$ time and $O(b\sqrt{r} + (b^{1/2+x})^3 + \sqrt{r}(b^{1/2+x})^2)$ work. The bounds follow. ∎

The update algorithm works as follows: It computes all pairs shortest paths among boundary vertices for the at most three regions where the edge (whose cost has been modified) belongs to. Clearly, this requires $O(\log^3 r)$ time and $O(r^{1.5})$ work. By choosing $r = \log^2 n$ we have the following.

Lemma 5.2 *Given an n-vertex planar digraph G with real-valued edge costs but no negative cycles, there exists a CREW PRAM algorithm for the on-line and dynamic shortest path problem on G that supports edge cost modification and edge deletion with the following performance characteristics: (i)* *preprocessing in $O(\log^5 n)$ time and $O(n^{1+\varepsilon})$ work, where $(1/2) > \varepsilon > 0$ can be arbitrarily small, and $O(n \log n)$ space; (ii) single-pair shortest path query in $O(\log^3 n)$ time and $O((n/\log n)^{1.5+\varepsilon})$ work; (iii) update in $O((\log\log n)^3)$ time and $O(\log^3 n)$ work.*

Now, using the approach described in Section 4, we can use the above described algorithms for making queries as well as implementing and updating the data structures of G_q. Thus combining Theorem 3 with the above lemma we get the following theorem.

Theorem 4 *Given an n-vertex planar digraph G with real-valued edge costs but no negative cycles, there exists a CREW PRAM algorithm for the on-line and dynamic shortest path problem that supports edge cost modification and edge deletion with the following performance characteristics: (i) preprocessing time $O(\log n \log^* n + \log^5 q)$ and $O(n \log n \log^* n + q^{1+\varepsilon})$ work, where $(1/2) > \varepsilon > 0$ can be arbitrarily small, and $O(n + q \log q)$ space; (ii) single-pair distance query time $O(\log n + \log^3 q)$ and $O(\log n + (q/\log q)^{1.5+\varepsilon})$ work; (iii) single-pair shortest path query time $O(\log n + \log^3 q)$ and $O(L + \log n + (q/\log q)^{1.5+\varepsilon})$ work (L is the number of edges of the path); (iv) update time (after an edge cost modification or edge deletion) $O(\log n + (\log\log q)^3)$ and $O(\log n + \log^3 q)$ work.*

5.2 Extensions of our results

Using our data structures we can also provide a solution to another well known version of the shortest path problem. Given a digraph G with real-valued edge costs but no negative cycles, a single-source shortest path query for a vertex v of G asks for the shortest paths between v and all other vertices of G. We can solve the problem by using a similar technique to the one described in the previous sections. The following result holds.

Theorem 5 *Let G be an n-vertex planar digraph with real-valued edge costs but no negative cycles. There exists a sequential and a CREW PRAM algorithm for the on-line and dynamic single-source shortest path tree problem on G that supports edge cost modification and edge deletion with the following performance characteristics: (i) preprocessing sequential time $O(n + q \log q)$, or parallel time $O(\log n \log^* n + \log^5 q)$ with $O(n \log n \log^* n + q^{1+\varepsilon})$ work, where $(1/2) > \varepsilon > 0$ can be arbitrarily small; (ii) preprocessing space $O(n + q \log q)$; (iii) single-source shortest path query in $O(n + q\sqrt{\log\log q})$ sequential time, or in $O(\log^2 n + \log^3 q)$ parallel time and $O(n + (q/\log q)^{1.5+\varepsilon})$ work; (iv) update time (after an edge cost modification or edge deletion) $O(\log n + (\log\log q)^3)$ and $O(\log n + \log^3 q)$ work.*

The hammock decomposition technique can be extended to n-vertex digraphs G of genus $\gamma = o(n)$. We make use of the fact [12] that the minimum number q of hammocks is at most a constant factor times $\gamma + q'$, where q' is the minimum number of faces of any embedding of G on a surface of genus γ that cover all vertices of G. Note that the methods of [12, 19] do not require such an embedding to be provided by the input in order to produce the hammock decomposition in $O(q)$ hammocks. The decomposition can be found in $O(n + m)$ sequential time [12], or in $O(\log n \log\log n)$ parallel time using $O(n + m)$ CREW PRAM processors [19], where m is the number of the edges of G. The only other property of planar graphs that is relevant to our shortest path algorithms is the existence of a 2/3-separator of size $O(\sqrt{n})$ for any planar n-vertex graph. For any n-vertex graph of genus $\gamma > 0$, a 2/3-separator of size $O(\sqrt{\gamma n})$ exists and such a separator can be found in linear time [4, 5]. Furthermore, an embedding of G does not need to be provided by the input. Thus the statements of Theorems 2, 4 and 5 hold for the class of graphs of genus $\gamma = o(n)$.

Acknowledgements. We are grateful to Esteban Feuerstein and Anil Maheshwari for many helpful discussions.

References

[1] G. Ausiello, G.F. Italiano, A.M. Spaccamela, U. Nanni, “Incremental algorithms for minimal length paths”, *J. of Algorithms*, 12 (1991), pp.615-638.

[2] M. Carroll and B. Ryder, “Incremental Data Flow Analysis via Dominator and Attribute Grammars”, *Proc. 15th Ann. ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages*, 1988.

[3] E. Cohen, “Efficient Parallel Shortest-paths in Digraphs with a Separator Decomposition”, *Proc. 5th ACM Symp. on Parallel Algorithms and Architectures*, 1993, pp.57-67.

[4] H. Djidjev, “A Separator Theorem for Graphs of Fixed Genus”, SERDICA, Vol.11, 1985, pp.319-329.

[5] H. Djidjev, “A Linear Algorithm for Partitioning Graphs of Fixed Genus”, SERDICA, Vol.11, 1985, pp.369-387.

[6] H. Djidjev, G. Pantziou and C. Zaroliagis, “Computing Shortest Paths and Distances in Planar Graphs”, in *Proc. 18th ICALP*, 1991, LNCS, Vol. 510, pp. 327-339, Springer-Verlag.

[7] P. Erdős and J. Spencer, “Probabilistic Methods in Combinatorics”, Academic Press, 1974.

[8] S. Even and H. Gazit, “Updating distances in dynamic graphs”, *Methods of Operations Research*, Vol.49, 1985, pp.371-387.

[9] D. Eppstein, Z. Galil, G. Italiano and A. Nissenzweig, “Sparsification - A Technique for Speeding Up Dynamic Graph Algorithms”, *Proc. 33rd Symp. on FOCS*, 1992, pp.60-69.

[10] E. Feuerstein and A.M. Spaccamela, “Dynamic Algorithms for Shortest Paths in Planar Graphs”, *Theor. Computer Science*, 116 (1993), pp.359-371.

[11] G.N. Frederickson, “Fast algorithms for shortest paths in planar graphs, with applications”, *SIAM J. on Computing*, 16 (1987), pp.1004-1022.

[12] G.N. Frederickson, “Using Cellular Graph Embeddings in Solving All Pairs Shortest Path Problems”, *Proc. 30th Annual IEEE Symp. on FOCS*, 1989, pp.448-453; also CSD–TR-897, Purdue University, August 1989.

[13] G.N. Frederickson, “Planar Graph Decomposition and All Pairs Shortest Paths”, *J. ACM*, Vol.38, No.1, January 1991, pp.162-204.

[14] G.N. Frederickson, “Searching among Intervals and Compact Routing Tables”, *Proc. 20th ICALP*, 1993, LNCS 700, pp.28-39, Springer-Verlag.

[15] M. Fredman and R. Tarjan, “Fibonacci heaps and their uses in improved network optimization algorithms”, *JACM*, 34(1987), pp. 596-615.

[16] Z. Galil and G. Italiano, “Fully Dynamic Algorithms for Edge-connectivity Problems”, *Proc. 23rd ACM STOC*, 1991, pp.317-327.

[17] H. Gazit and G. Miller, “A Parallel Algorithm for finding a Separator in Planar Graphs”, *Proc. 28th IEEE Symp. on FOCS*, 1987, pp.238-248.

[18] A. Kanevsky, G. Di Battista, R. Tamassia and J. Chen, “On-line Maintenance of the Four-Connected Components of a Graph”, *Proc. 32nd IEEE Symp. on FOCS*, 1991, pp.793-801.

[19] D. Kavvadias, G. Pantziou, P. Spirakis and C. Zaroliagis, “Hammock-on-Ears Decomposition: A Technique for Parallel and On-line Path Problems”, Computer Technology Institute Technical Report, CTI-TR-93.05.22, Patras, May 1993.

[20] D. Kavvadias, G. Pantziou, P. Spirakis and C. Zaroliagis, “On the expected number of hammocks in a graph”, preprint, October 1993.

[21] P. Klein and S. Subramanian, “A linear-processor polylog-time algorithm for shortest paths in planar graphs”, *Proc. 34th IEEE Symp. on FOCS*, 1993, pp.259-270.

[22] J.A. La Poutré, “Alpha-Algorithms for Incremental Planarity Testing”, to appear in *Proc. 26th ACM STOC*, 1994.

[23] A. Lingas, “Efficient Parallel Algorithms for Path Problems in Planar Directed Graphs”, *Proc. SIGAL'90*, LNCS 450, pp.447-457, 1990, Springer-Verlag.

[24] G. Pantziou, P. Spirakis and C. Zaroliagis, “Efficient Parallel Algorithms for Shortest Paths in Planar Digraphs”, *BIT* 32 (1992), pp.215-236.

[25] D. Sleator and R. Tarjan, “A Data Structure for Dynamic Trees”, *Journal Comput. System Sci.* 26 (1983), pp. 362–391.

[26] J. Westbrook, “Algorithms and Data Structures for Dynamic Graph Problems”, PhD Dissertation, CS-TR-229-89, Dept of Computer Science, Princeton University, 1989.

[27] M. Yannakakis, "Graph Theoretic Methods in Database Theory", *Proc. ACM conference on Principles of Database Systems*, 1990.

[28] D. Yellin and R. Strom, "INC: a language for incremental computations", *Proc. ACM SIGPLAN conf. on Programming Language Design and Implementation*, 1988, pp.115-124.