On-line and Dynamic Algorithms for Shortest Path Problems ⋆ (to appear in STACS'95)

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

[1] Computer Science Dept, Rice University, P.O. Box 1892, Houston, TX 77251, USA
[2] Computer Science Dept, University of Central Florida, Orlando FL 82816, USA
[3] Max-Planck 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. An important feature of our algorithms is that they can work in a dynamic environment, where the cost of any edge can be changed or the edge can be deleted. Our data structures can be updated after any such change in only polylogarithmic time, while a single-pair query is answered in sublinear time. We also describe the first parallel algorithms for solving the dynamic version of the shortest path problem.

1 Introduction

There has been a growing interest in dynamic graph problems in the recent years [1, 9, 17, 24, 28]. The goal is to design efficient data structures that not only allow one to give 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 (see e.g. [3, 31, 32]).

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 and w of G is the minimum length of a path between v and w. The path of minimum length is called a *shortest path* between v and w. Finding shortest path information in graphs is an important and intensively studied problem with many applications. Recent papers [4, 6, 11–13, 16, 21, 25, 27] 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 *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

⋆ This work was partially supported by the EC ESPRIT Basic Research Action No. 7141 (ALCOM II), by the EC Cooperative Action IC-1000 (project ALTEC) and by the NSF grants No. CDA-9211155 and No. CCR-9409191. Email: hristo@cs.rice.edu, pantziou@cs.ucf.edu, zaro@mpi-sb.mpg.de.

distance queries. In case of edge deletion or edge cost modification of G, update the data structure in an appropriately short time.

The dynamic version of the shortest paths problem has a lot of applications. A dynamic algorithm for the shortest path problem can be used to dynamically maintain a maximum st-flow in a planar network [19], a feasible flow between multiple sources and sinks, as well as a bipartite perfect matching in planar graphs [26]. Dynamic algorithms for shortest paths appear also to be fundamental procedures in incremental computations for data flow analysis and interactive systems design [29, 32].

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. Also the recent results of [23] for single-source shortest paths in planar digraphs (along with other techniques) lead to dynamic algorithms for shortest paths in planar digraphs with integral edge costs. However, the preprocessing as well as the update and query time bounds are superlinear. For example, the time for a query or an update operation is $O(n^{9/7} \log n)$. In the case of planar digraphs with real edge costs the best dynamic algorithms are due to [10]. 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 (NC) 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 introduced by Frederickson in [12, 13] which is proportional to the cardinality of a minimum number of faces covering all vertices of G (among all embeddings of G in the plane). Note that q can be as small as 1 and is always strictly smaller than n in the worst-case. Hence, our results are improvements over the best previous ones in all cases where $q = o(n)$, i.e. in all cases where G has nice topological properties. As an example, consider outerplanar digraphs ($q = 1$), or planar digraphs which satisfy the k-interval property ($k \ll n$ and $q = k$) as they are defined in [15]. Another class of graphs with important applications are the graphs describing global area networks. Typically such graphs can be represented as a tree plus a small number of non-tree edges and thus have a small value of q. In the case where G is outerplanar our preprocessing time and space are optimal (linear) and the distance query and the update time are logarithmic. (Similar results have been achieved independently in [2].) Our algorithms provide a simple direct solution compared with the algorithms in [13, 14] which are not dynamic and were based on manipulations with compact routing tables.

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, 27]. (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).

Our algorithms can detect a negative cycle, either if it exists in the initial digraph, or if it is created after an edge cost modification. 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. An efficient parallelization, as well as other extensions of our results are discussed in Section 5. Due to lack of space some proofs are omitted, but can be found in the full paper [7].

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 divides G into two disjoint 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_{op} and a set M of vertices of G_{op}, *compressing* G_{op} *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 the distance between the same vertices in G_{op} [13]. (In our algorithms the size of M will be $O(1)$.)

Definition 1. *Let G_{op} be an outerplanar digraph and S be an α-separator of G_{op} that divides G_{op} 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: divide G into two subgraphs by using p as a separation pair, 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* of G is a decomposition of G into certain outerplanar digraphs called *hammocks*. This decomposition is defined with respect 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, 27] 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 and $O(n \log n \log^* n)$ work on a CREW PRAM. Also, by [12, 21], 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 the minimum possible (within a constant factor) among all possible decompositions.

In the sequel, we can assume w.l.o.g. that G_{op} is a biconnected n-vertex outerplanar digraph. Note that if G_{op} is not biconnected we can add an appropriate number of additional edges of very large costs in order to convert it into a biconnected outerplanar digraph (see [13, 27]).

2.1 Constructing a separator decomposition

We describe an algorithm that generates a decomposition of G_{op} (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_{op}. 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

resulting component is $O(1)$. The following algorithm finds such a partitioning and constructs the associated *separator tree*, $ST(G_{op})$, used to support binary search in G_{op}. Let in the algorithm below G denote a subgraph of G_{op} (initially $G := G_{op}$).

ALGORITHM Sep_Tree$(G, ST(G))$

1. If $|V(G)| \leq 4$, then halt. Else let S denote the set of separation pairs in G_{op} found during all previous iterations. (Initially $S = \emptyset$.) Let n_{sep} denote the number of separation pairs of S attached to G.

1.1. If $n_{sep} \leq 3$, then let $p = \{p_1, p_2\}$ be a separation pair of G that divides G 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 into subgraphs G_1 and G_2 each containing no more than 2/3 of the number of separation pairs attached to G.

2. Add p to S and run this algorithm recursively on G_i for $i = 1, 2$. Create a separator tree $ST(G)$ rooted at a new node v associated with p and G, and whose children are the roots of $ST(G_1)$ and $ST(G_2)$.

Observe that the nodes of $ST(G_{op})$ are associated with subgraphs of G_{op} which we will call *descendant subgraphs* of G_{op}. With each descendant subgraph a distinct separation pair is associated. From the description of the algorithm, it follows that:

Lemma 1. *Any descendant subgraph G of G_{op} at level i in $ST(G_{op})$ 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. Using the dynamic tree data structure of [30] and working on the dual graph of G_{op} (which is a tree), we can show the following [7].

Lemma 2. *Algorithm Sep_Tree$(G_{op}, ST(G_{op}))$ can be implemented to run in $O(n)$ time and space. The depth of the resulting separator tree $ST(G_{op})$ is $O(\log n)$.*

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_{op} 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 $ST(G_{op})$. Each node of $ST(G_{op})$ is associated with a descendant subgraph G of G_{op} along with its separation pair as determined

by algorithm Sep_Tree and also contains a pointer to the sparse representative $SR(G)$ of G.

(II) The sparse representative $SR(G)$ for all graphs G of $ST(G_{op})$. According to Definition 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 plus the separation pair dividing G, where G_1 and G_2 are the children of G in $ST(G_{op})$. 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 $SR(G)$ in constant time; (b) for each leaf of $ST(G_{op})$ 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_{op} is sufficient to compute the distance between any 2 vertices of G_{op} 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 that constructs the above data structures in linear time.

ALGORITHM Pre_Outerplanar(G_{op})
1. Construct a separator tree $ST(G_{op})$ using algorithm Sep_Tree($G_{op}, ST(G_{op})$).
2. Compute the sparse representative $SR(G_{op})$ of G_{op} as follows.
   for each child G of G_{op} in $ST(G_{op})$ **do**
(a) **if** G is a leaf of $ST(G_{op})$ **then** $SR(G) = G$
   else find $SR(G)$ by running Step 2 recursively on G.
(b) Construct the sparse representative of G_{op} as described in Definition 1 by using the sparse representatives of the children of G_{op}.

Lemma 3. *Algorithm Pre_Outerplanar(G_{op}) runs in $O(n)$ time and uses $O(n)$ space.*

Proof. Step 1 needs $O(n)$ time and space by Lemma 2. Let $P(n)$ be the maximum 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 $ST(G_{op})$ since each sparse representative has $O(1)$ size. Therefore the space needed for the above data structures is $O(|ST(G_{op})|) = O(n)$. 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_{op}. The algorithm proceeds as follows. First search $ST(G_{op})$ to find a descendant subgraph G of G_{op} such that the separation pair $p = (p_1, p_2)$ associated with G separates v from z. Let $d(v, z)$ denote the distance between v and z. Then obviously

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

Hence, it suffices to compute the distances $d(v, p_1)$, $d(p_1, z)$, $d(v, p_2)$ and $d(p_2, z)$. In order to do this we will need the shortest path information encoded in the sparse representatives.

Now we will analyze how one can use the information the sparse representatives provide. Let $s = (s_1, s_2)$ be any separation pair attached to G. Let s divide some descendant subgraph H of G_{op} 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 an ancestor of G in $ST(G_{op})$, we call s an *ancestor* separation pair of G and if H is a parent of G, we call s a *parent* separation pair of 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_{op}, if s is an ancestor separation pair. (If s is not an ancestor separation pair the distances are the same.) Note that G can have more than one ancestor separation pair, but only one parent separation pair (if $G \neq G_{op}$).

Assume that $G \neq G_{op}$. Denote by $M(G)$ the set of the parent separation pairs of all descendant subgraphs of G_{op} that are ancestors of G in $ST(G_{op})$ (including G). Then $M(G)$ contains all ancestor separation pairs of G. Let $D(G)$ be the set of all distances $d(x_1, x_2)$ and $d(x_2, x_1)$ in G_{op}, where (x_1, x_2) is a separation pair in $M(G)$. Then $D(G)$ can be found by the following algorithm.

ALGORITHM Parent_Pairs(G)

1. Let G' be the parent of G in $ST(G_{op})$. If $G' = G_{op}$ then $D(G') := \emptyset$; otherwise compute recursively $D(G')$ by this algorithm.

2. Find $d(s_1', s_2')$ and $d(s_2', s_1')$ in G_{op} by using $SR(G')$ and the information in $D(G')$, where (s_1', s_2') is the separation pair associated with G'. Set $D(G) := D(G') \cup \{d(s_1', s_2'), d(s_2', s_1')\}$.

Note that by the above discussion $D(G)$ contains the distances in G_{op} between the vertices of all ancestor separation pairs attached to G. The time complexity of Algorithm Parent_Pairs is clearly $O(\log n)$. Thus Algorithm Parent_Pairs can be used to compute in $O(\log n)$ time the distances in G_{op} between the pairs of vertices of all ancestor separation pairs attached to G so that one can ignore the rest of G_{op} when computing distances in G.

Next we describe the query algorithm. Let v' be a vertex that belongs to the same descendant subgraph of G_{op} that is a leaf of $ST(G_{op})$ 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 $ST(G_{op})$ containing z. For any two pairs p' and p'' of vertices, let $D(p', p'')$ denote the set of all four distances in a vertex from p' to a vertex in 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_{op}, v, z)

1. Search $ST(G_{op})$ (starting from the root) to find pairs of vertices $p(v)$ and $p(z)$ as defined above.

2. Search $ST(G_{op})$ (starting from the root) to find a descendant subgraph G of G_{op} 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 ancestor separation pairs of G by Algorithm Parent_Pairs (if G has an ancestor separation pair).

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

4.1. Search $ST(G_{op})$ (starting from G) to find a descendant subgraph 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 $ST(G_{op})$, 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 $D(p(v), p(z))$.

Lemma 4. *Algorithm Dist_Query_Outerplanar(G_{op}, v, z) finds the distance between any two vertices v and z of an n-vertex outerplanar digraph G_{op} in $O(\log n)$ time.*

Proof. The correctness follows from the description of the algorithm. Searching the tree $ST(G_{op})$ in Steps 1 and 2 takes in total $O(\log n)$ time by Lemma 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 $ST(G_{op})$ and l_{max} is the maximum level of $ST(G_{op})$. Then from the description of the algorithm $Q(l) \leq Q(l+1) + O(1)$ 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 $ST(G_{op})$. Uncompressing an edge from a graph $SR(G)$ involves a traversal of a subtree of $ST(G_{op})$, 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. Hence:

Lemma 5. *The shortest path between any two vertices v and z of an n-vertex outerplanar digraph G_{op} can be found in $O(\log n + L)$ time, where L is the number of 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 cost, 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_{op} is based on the following idea: the edge will belong to at most $O(\log n)$ subgraphs of G_{op}, 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 $ST(G_{op})$) to the root of $ST(G_{op})$. Let $parent(G)$ denote the parent of a node G in $ST(G_{op})$, and $\hat{G}$ denote the sibling of a node G in a $ST(G_{op})$. 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($G_{op}, e, w(e)$)
1. Find a leaf G of $ST(G_{op})$ 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** $G \neq G_{op}$ **do**
   (a) Update $SR(parent(G))$ using the new versions of $SR(G)$ and $SR(\hat{G})$.
   (b) $G := parent(G)$.

Lemma 6. *Algorithm Update_Outerplanar updates after an edge cost modification the data structures created by the preprocessing algorithm in $O(\log n)$ time.*

3.4 Handling of negative cycles and summary of the results

The initial digraph G_{op} can be tested for existence of a negative cycle in $O(n)$ time by [21]. Assume now that G_{op} does not contain a negative cycle and that the cost $c(v, w)$ of an edge $\langle v, w \rangle$ in G_{op} 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_{op} 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 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 is the following: (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 into a planar digraph G_q, which is of size $O(q)$. (4) Run the preprocessing algorithm of [10] in G_q. Let us call the above algorithm *Pre_Planar*. From the results in [10, 13], the discussion in Section 2 and Theorem 1, we have that algorithm Pre_Planar takes $O(n+q\log q)$ time and space.

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, the data structures created by algorithm Pre_Planar can be updated 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, $d(v,z) = \min\{d_H(v,z), d_{ij}(v,z)\}$. Since querying in G_q's data structure takes $O(q)$ time by [10] and querying in each hammock takes $O(\log n)$ time, we have that distance query takes $O(q+\log n)$ time in total. (A shortest path query can be computed similarly.)

The case of negative edge costs is handled in a similar way with that of outerplanar digraphs. Therefore, we have:

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* $O(n)$.

5 Further Results

In this section we give other results following from our approach to the dynamic shortest path problem. We shall first discuss the parallel implementation of our algorithms on the CREW PRAM model of computation.

Using the dual graph of G_{op} (which is a tree) and standard techniques for computing functions in a tree (see e.g. [20], Chapter 3), we can implement the data structure from Theorem 1 in $O(\log n)$ time and $O(n \log n)$ work. The sequential distance query and the update algorithms for outerplanar digraphs are logarithmic, but the shortest path sequential query algorithm requires $O(L + \log n)$ time, where L is the number of edges of the path. We can find an optimal logarithmic-time parallel implementation of the shortest path query algorithm by the following observations. Algorithm Dist_Query_Outerplanar determines in $O(\log n)$ time a subtree of $ST(G_{op})$ consisting of the descendant subgraphs of G_{op} that contain 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.

In the case of planar digraphs we need a parallel algorithm to build the data structures in G_q (recall Section 4). We will make use of the following recent result of Cohen [4]: *In any q-vertex planar digraph J the shortest paths from s sources can be computed in $O(\log^2 q)$ time and $O(sq)$ work. A preprocessing phase is needed which takes $O(\log^3 q)$ time and $O(q^{1.5})$ work.* Note that J should be provided by a separator decomposition (i.e. a recursive decomposition of J using 2/3-separators of size $O(\sqrt{q})$), for the algorithm of [4] to be applied. Using the result of [18], such a decomposition for J is constructed in $O(\log^5 q)$ time using $O(q^{1+\varepsilon})$ work, for any arbitrarily small $(1/2) > \varepsilon > 0$. Furthermore, finding a hammock decomposition (Step 1 of algorithm Pre_Planar) takes $O(\log n \log^* n)$ time and $O(n \log n \log^* n)$ work by [27]. Combining these results with the ones discussed above for outerplanar digraphs and using the construction from Section 4 we have the following.

Theorem 3. *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 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 $O(\log n \log^* n + \log^5 q)$ and $O(n \log n \log^* n + q^{1.5})$ work, using $O(n + q^{1.5})$ space; (ii) distance query time $O(\log n + \log^2 q)$ and $O(\log n + q)$ work; (iii) shortest path query time $O(\log n + \log^2 q)$ and $O(\log n + q + L)$ work; and (iv) update time (after an edge cost modification or edge deletion) $O(\log n + \log^3 q)$ and $O(\log n + q^{1.5})$ work.*

Observe that we can cut the additive factors depending on q in both preprocessing and update bounds, at the expense of an additive factor of $O(q^{1.5})$ in the query work.

Another well-known version of the shortest path problem is to find a single-source shortest path tree rooted at a vertex v of a digraph G, i.e. find shortest paths between v and all other vertices in G. This problem can be solved by the same data structure and by using similar techniques with the ones described in Sections 3 and 4. We will first present the solution for the outerplanar case.

Let G_{op} be an outerplanar digraph. Let $U \subset V$ be a subset of $O(1)$ vertices of G_{op} with a weight $d_0(u)$ on any $u \in U$. For any vertex v of G_{op} the weighted distance $d(U, v)$ is defined by $d(U, v) = \min\{d_o(u) + d(u, v) | u \in U\}$. We assume that $d(U, v) = d_0(v)$ for every $v \in U$. The following algorithm computes $d(U, v)$, $\forall v \in G_{op}$.

ALGORITHM Single_Source_Query_Outerplanar(G_{op}, U)

1. Let S be the 2/3-separator associated with the root G_{op} of $ST(G_{op})$. Compute $d(u, s)$ for all vertices $u \in U$ and $s \in S$ by using algorithm Dist_Query_Outerplanar.

2. For any $s \in S$ define $d_0(s) = \min\{d(u, s) | u \in U\} = d(U, s)$.

3. Run recursively Single_Source_Query_Outerplanar$(G, (S \cup U) \cap G)$, on each child subgraph G of G_{op} which is not a leaf of $ST(G_{op})$. (If G is a leaf, then distances are computed easily since the associated subgraph is of $O(1)$ size.)

The correctness of the algorithm follows easily from its description. Let $D(n)$ be the running time of the algorithm. Then, $D(n) \leq \max\{D(n_1) + D(n_2) \mid n_1 + n_2 = n, \quad n_1, n_2 \leq 2n/3\} + O(|S| \cdot |U| \cdot \log n) = \max\{D(n_1) + D(n_2) \mid n_1 + n_2 = n, \quad n_1, n_2 \leq 2n/3\} + O(\log n)$, which gives $D(n) = O(n)$.

Let v be any vertex of G_{op}. By running Single_Source_Query_Outerplanar$(G_{op}, \{v\})$ with $d_0(v) = 0$, we can compute, in $O(n)$ time, all distances from v to every other vertex in G_{op}. Having the distances, it is not difficult to compute the single-source shortest path tree rooted at v in $O(n)$ time.

Using the above result and the methodology of Section 4, our data structures for planar digraphs can answer single-source shortest path tree queries, in $O(n + q\sqrt{\log \log q})$ time. It is not difficult to see that algorithm Single_Source_Query_Outerplanar can be implemented to run in $O(\log^2 n)$ time and $O(n)$ work on a CREW PRAM. Within the same resource bounds the shortest path tree can be also constructed.

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, 21] 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 and $O((n + m) \log n \log \log n)$ work on a CREW PRAM [21], 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 (as well as to the algorithms in [10]) 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 [5]. Furthermore, an embedding of G does not need to be provided by the input. (For the CREW PRAM implementation, such a separator should be provided

with the input [4].) Thus the statement of Theorem 2 as well as its extensions discussed in this section, hold for the class of graphs of genus $\gamma = o(n)$.

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. H. Bondlaender, "Dynamic Algorithms for Graphs with Treewidth 2", *Proc. 19th WG'93*, LNCS 790, pp.112-124, Springer-Verlag, 1994.
3. M. Carroll and B. Ryder, "Incremental Data Flow Analysis via Dominator and Attribute Grammars", *Proc. 15th Ann. ACM POPL*, 1988.
4. E. Cohen, "Efficient Parallel Shortest-paths in Digraphs with a Separator Decomposition", *Proc. 5th ACM SPAA*, 1993, pp.57-67.
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", *Proc. 18th ICALP'91*, LNCS 510, pp.327-339, Springer-Verlag.
7. H. Djidjev, G. Pantziou and C. Zaroliagis, "On-line and Dynamic Algorithms for Shortest Path Problems", Tech. Rep. MPI-I-94-114, Max-Planck-Institut für Informatik, April 1994.
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 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 IEEE Symp. on FOCS*, 1989, pp.448-453.
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'93*, LNCS 700, pp.28-39, Springer-Verlag.
15. G.N. Frederickson and R. Janardan, "Designing Networks with Compact Routing Tables", *Algorithmica*, 3(1988), pp.171-190.
16. M. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms", *JACM*, 34(1987), pp. 596-615.
17. Z. Galil and G. Italiano, "Fully Dynamic Algorithms for Edge-connectivity Problems", *Proc. 23rd ACM STOC*, 1991, pp.317-327.
18. 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.
19. R. Hassin, "Maximum flow in (s,t)-planar networks", *IPL*, 13(1981), p.107.
20. J. JáJá, "An Introduction to Parallel Algorithms", Addison-Wesley, 1992.
21. D. Kavvadias, G. Pantziou, P. Spirakis and C. Zaroliagis, "Hammock-on-Ears Decomposition: A Technique for the Efficient Parallel Solution of Shortest Paths and Other Problems", *Proc. 19th MFCS'94*, LNCS 841, pp.462-472, Springer-Verlag.

22. D. Kavvadias, G. Pantziou, P. Spirakis and C. Zaroliagis, "Efficient Sequential and Parallel Algorithms for the Negative Cycle Problem", *Proc. 5th ISAAC'94*, LNCS 834, pp.270-278, Springer-Verlag.
23. P. Klein, S. Rao, M. Rauch and S. Subramanian, "Faster shortest-path algorithms for planar graphs", *Proc. 26th ACM STOC*, 1994, pp.27-37.
24. J.A. La Poutré, "Alpha-Algorithms for Incremental Planarity Testing", *Proc. 26th ACM STOC*, 1994, pp.706-715.
25. A. Lingas, "Efficient Parallel Algorithms for Path Problems in Planar Directed Graphs", *Proc. SIGAL'90*, LNCS 450, pp.447-457, 1990, Springer-Verlag.
26. G. Miller and J. Naor, "Flows in planar graphs with multiple sources and sinks", *Proc. 30th IEEE Symp. on FOCS*, 1991, pp.112-117.
27. G. Pantziou, P. Spirakis and C. Zaroliagis, "Efficient Parallel Algorithms for Shortest Paths in Planar Digraphs", *BIT* 32 (1992), pp.215-236.
28. M. Rauch, "Fully Dynamic Biconnectivity in Graphs", *Proc. 33rd IEEE Symp. on FOCS*, 1992, pp.50-59.
29. G. Ramalingan and T. Reps, "On the Computational Complexity of Incremental Algorithms", Technical Report, University of Wisconsin-Madison, 1991.
30. D. Sleator and R. Tarjan, "A Data Structure for Dynamic Trees", *Journal Comput. System Sci.* 26 (1983), pp. 362–391.
31. M. Yannakakis, "Graph Theoretic Methods in Database Theory", *Proc. ACM conference on Principles of Database Systems*, 1990.
32. D. Yellin and R. Strom, "INC: A language for incremental computations", *ACM Trans. Prog. Lang. Systems*, 13 (2), pp.211-236, April 1991.