On-line and Dynamic Algorithms for Shortest Path Problems *

Hristo N. Djidjev
Department of Computer Science, Rice University,
P.O. Box 1892, Houston, TX 77251, USA
Phone: +1 (713) 527-8750 ext. 3246, Fax: +1 (713) 285-5930
Email: hristo@cs.rice.edu

Grammati E. Pantziou
Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
and
Department of Mathematics and Computer Science,
Dartmouth College, Hanover NH 03755, USA
Phone: +1 (603) 646-1613, Fax: +1 (603) 298-8312
Email: pantziou@cs.dartmouth.edu

Christos D. Zaroliagis
Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
and
Max-Plank Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany
Phone: +49 (681) 302-5356, Fax: +49 (681) 302-5401
Email: zaro@mpi-sb.mpg.de

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. 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. We also describe the first parallel algorithms for solving the dynamic version of the shortest path problem. Our results can be extended to hold for digraphs of genus $o(n)$.

Keywords: Shortest path, dynamic algorithm, planar digraph, outerplanar digraph.

*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.

1 Introduction

1.1 The problem and its motivation

There has been a growing interest in dynamic graph problems in the recent years [1, 9, 16, 19, 23, 26]. 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 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, 28, 29, 30], 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 and 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 problem with many applications. Recent papers [3, 6, 11, 12, 13, 15, 20, 22, 24, 25] 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 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. Consider for example, the *vehicle routing* problem: 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 close (because of repair works or accidents), certain roads may change behavior at rush hours, or some other ones may change direction. The on-line and dynamic shortest path problem is one of the fundamental problems that one has to solve in order to develop a software system that will give fast and efficient solutions to the vehicle routing problem.

1.2 Previous work

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.

1.3 Our results

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.

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, 21] 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 shortest path tree query time, and $O(\log n)$ update time. For comparison of our results with previous ones, see Table 1.

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, 25].

(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 digraph, or if it is created after an edge cost modification.

(iii) Using the ideas of [12, 20], 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 construct 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 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, 25]. (In our algorithms the size of M will be $O(1)$.)

Definition 2.1 *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, 25] 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, 20], 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. Hammock decompositions allows us to *reduce* the solution of a given problem Π on a planar digraph to a solution of Π on an outerplanar digraph.

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, 25]).

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($G, ST(G)$)
BEGIN

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 $ST(G_2)$.

END.

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, the following fact follows.

Lemma 2.1 *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. 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($G_{op}, ST(G_{op})$) can be implemented to run in $O(n)$ time and $O(n)$ space. The depth of the resulting separator tree $ST(G_{op})$ 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 in Step 1.2 we keep for each component K into which S divides G 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_{op} (excluding the outer face), which is a tree. By using the data structure of [27] for dynamic trees we can find one separation pair in $O(\log n)$ time. Then the maximum time $T(n)$ needed to find all separation pairs satisfies the recurrence

$$T(n) \leq \max\{T(n_1) + T(n_2) \mid n_1 + n_2 = n, \quad n_1, n_2 \leq 2n/3\} + O(\log n), \quad n > 1$$

which has a solution $T(n) = O(n)$. Since $ST(G_{op})$ 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_{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 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 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})

BEGIN

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 2.1 by using the sparse representatives of the children of G_{op}.

END.

Lemma 3.1 *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 2.2. Let $P(n)$ be the maximum time required by Step 2. Then $P(n)$ satisfies the recurrence

$$P(n) \leq \max\{P(n_1) + P(n_2) \mid n_1 + n_2 = n, \quad n_1, n_2 \leq 2n/3\} + O(1), \quad n > 1$$

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

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

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

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 $D(G) := D(G') \cup \{d(s'_1, s'_2), d(s'_2, s'_1)\}$.

END.

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)

BEGIN

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 $p(v)$ a nd $p(z)$ 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 $D(p(v), p(z))$.

END.

Lemma 3.2 *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 $ST(G_{op})$ 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 $ST(G_{op})$ and l_{max} is the maximum level of $ST(G_{op})$. Then from the description of the algorithm

$$Q(l) \leq Q(l+1) + O(1) \quad for\ l < l_{max},$$

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 $ST(G_{op})$. 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. 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_{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($G_{op}, e, w(e)$)
BEGIN
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) $G := parent(G)$.

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 $ST(G_{op})$ is $O(\log n)$, Step 1 clearly can be implemented in logarithmic time by doing a binary search on $ST(G_{op})$. 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_{op} is $O(1)$. ∎

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 [20]. 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 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) $O(\log n)$.*

Proof: Follows by Lemmata 3.1, 3.2, 3.3 and 3.4. ∎

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.

ALGORITHM Pre_Planar(G)
BEGIN
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.
END.

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.*

The query algorithm for finding the shortest path or distance between any two vertices v and z of G is the following. (Note that if both v and z belong to the same hammock H, then their shortest path does not necessarily have to stay in H.)

ALGORITHM Query_Planar(G, v, z)
BEGIN
(* Let H, H' be hammocks with attachment vertices $a_i, 1 \le i \le 4$ and $a'_i, 1 \le i \le 4$, respectively, such that $v \in H$ and $z \in H'$. *)
if $H \equiv H'$ (* i.e. both v, z belong to H *) **then**

1. Run Dist_Query_Outerplanar(H, v, z) and let $d_H(v, z)$ be its output.
2. $d_{ij}(v, z) = \min_{i,j}\{d(v, a_i) + d(a_i, a_j) + d(a_j, z)\}$.
3. $d(v, z) = \min\{d_H(v, z), d_{ij}(v, z)\}$.

else (* $H \neq H'$ *)

$d(v, z) = \min_{i,j}\{d(v, a_i) + d(a_i, a'_j) + d(a'_j, z)\}$.

END.

Lemma 4.3 *Algorithm Query_Planar computes the shortest path (resp., distance) between any two vertices in a planar digraph 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 analyze 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 $O(n)$.*

Proof: Follows by Lemmata 4.1, 4.2 and 4.3. The $O(n)$ space in the case of computing distances only, comes from Theorem 1 and the fact that the algorithm of [10] needs $O(q)$ space when applied to G_q for this problem. ∎

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 [20]. The procedure for accepting or not an edge cost modification is similar to the one described for outerplanar digraphs.

5 Related Results

In this section we give other results following from our approach to the dynamic shortest path problem. We first present an efficient parallel implementation of our algorithms on the CREW PRAM model of computation. We start with the case of outerplanar digraphs. We will show how the preprocessing algorithm from Section 3 can be implemented in parallel. Step 1 can be implemented in $O(\log n)$ time and $O(n \log n)$ work as follows. Let G_{op} be an n-vertex outerplanar digraph. Triangulate each face of G_{op} and construct the dual graph T of the resulting triangulation G_T, excluding the outer face of G_{op}. Since G_{op} is outerplanar, then T is a tree. Assign each vertex x of G_{op} to a unique triangle of G_T incident on x and determine for each triangle t the number of vertices of G_{op} assigned to t. Call this number the *weight of* t and also the *weight of the vertex of* T that corresponds to t. Then compute for each node v of T the number $w(v)$ equal to the sum of the weights of all descendants of v (including v itself). This can be easily done in $O(\log n)$ time and $O(n)$ work (see e.g. [18], Chapter 3). Using the numbers $w(v)$, find in constant time and $O(n)$ work an edge e of T whose removal divides T into two subtrees T_1 and T_2 each of total weight on its vertices at most 2/3 of the total weight of T. Then, the pair of the endpoints of the edge in G_T corresponding to e will be a 2/3-separator of G_{op}. Moreover, updating the numbers $w(\cdot)$ for T_1 and T_2 requires $O(1)$ time and $O(n)$ work. Thus Step 1 requires $O(\log n)$ 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 $T_p(n) = O(\log n)$. Hence, we have the following.

Theorem 3 *Given an n-vertex outerplanar digraph G with real-valued edge costs but no negative cycles, the data structure from Theorem 1 can be constructed 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 [3]. *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 [3] to be applied. Using the result of [17], 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 [25]. Combining these results with Theorem 3 and using the construction from Section 4 we derive a parallel algorithm for the class of n-vertex planar digraphs with the following 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. Note that our results compare favorably with those of [3] in all cases where $q = o(n)$.

Another well-known version of the shortest path problem is the following: Given a digraph G with real-valued edge costs but no negative cycles, find a single-source shortest path tree rooted at a vertex v of 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)

Begin

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.)

End.

The correctness of the algorithm follows from its description. Let $D(n)$ be the running time of the algorithm. Then, $D(n) \leq 2D(n/2) + O(|S| \cdot |U| \cdot \log n) = 2D(n/2) + O(\log n)$, which gives $D(n) = O(n)$.

Let v be any vertex of G_{op}. The single-source shortest path tree rooted at v can be computed as follows: (i) Run Single_Source_Query_Outerplanar($G_{op}, \{v\}$) with $d_0(v) = 0$. (ii) Each vertex $y \neq v$ checks its neighbors and selects as its parent that vertex x which satisfies $d(v,y) = d(v,x) + c(x,y)$, where $c(x,y)$ is the cost of edge $\langle x, y\rangle$. Hence, a single-source shortest path tree in G_{op} can be constructed in $O(n)$ time.

Algorithm Single_Source_Query_Outerplanar can be implemented to run in $O(\log^2 n)$ time and $O(n)$ work on a CREW PRAM. The recurrence for the work done by the algorithm satisfies the same recurrence as $D(n)$. The parallel time satisfies the recurrence $T_p(n) = T_p(n/2) + O(\log n)$, whose solution is $T_p(n) = O(\log^2 n)$. Moreover, step (ii) above runs in $O(\log n)$ time and $O(n)$ work, since we have to resolve conflicts in the case where there are more than one candidate parents which satisfy the distance condition.

Using the above result and the methodology of Section 4, we have the following.

Theorem 4 *Let G be an n-vertex planar digraph with real-valued edge costs but no negative cycles. There exists an 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 time and space $O(n + q\log q)$; (ii) single-source shortest path query in $O(n + q\sqrt{\log\log q})$ time; and (iii) update time (after an edge cost modification or edge deletion) $O(\log n + \log^3 q)$.*

For comparison, the best previous results for this problem are those in [10] (recall Section 1). On a CREW PRAM, a single-source query is answered in $O(\log^2 n)$ time and $O(n)$ work, thus matching the bound given in [3].

Notice that our update plus query work bound of $O(n + q^{1.5})$ compares favorably with the preprocessing plus query work of $O(n^{1.5})$ needed by the algorithm of [3] in order to answer a query after an edge cost modification. Also 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. Although this latter result still compares favorably with the query $O(n^{1.5})$-work bound of [3], it is not as good as the result we present here with respect to the amortized complexity of answering a large number of queries after one update – since in a dynamic system the queries are expected to be much more frequent than the updates.

Note that our results for outerplanar digraphs are important for the following reasons: (a) Our data structure can be updated after an edge cost modification or edge deletion in $O(\log n)$ time, while the algorithms in [13, 14] are not dynamic. In addition our algorithms provide simple direct solution, while the previous algorithms were based on manipulations with compact routing tables. (b) The CREW PRAM implementation of our results compares favorably with the results in [25] and moreover, the results here are dynamic.

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, 20] 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 [20], 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 [4, 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 by the input [3].) Thus the statement of Theorem 2 as well as its extensions discussed in this section, hold for the class of graphs of genus $\gamma = o(n)$.

Acknowledgement. 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] J. JáJá, "An Introduction to Parallel Algorithms", Addison-Wesley, 1992.

[19] 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.

[20] D. Kavvadias, G. Pantziou, P. Spirakis and C. Zaroliagis, "Hammock-on-Ears Decomposition: A Technique for Parallel and On-line Path Problems", Technical Report, CTI-TR-93.05.22, Patras, 1993; a preliminary version will appear at *Proc. 19th MFCS*, 1994.

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

[22] 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.

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

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

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

[26] M. Rauch, "Fully Dynamic Biconnectivity in Graphs", *Proc. 33rd IEEE Symp. on FOCS*, 1992, pp.50-59.

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

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

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

[30] 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.

	Djidjev, Pantziou & Zaroliagis [6]	Frederickson [14]	Feuerstein & Spaccamela [10]	This paper
Dynamic	No	No	Yes	Yes
Preprocessing Time & Space	$O(n \log n + q^2)$	$O(n + q^2)$	$O(n \log n)$	$O(n + q \log q)$
Single-Pair Dist. Query	$O(\log n)$	$O(L + \log n)$	$O(n)$	$O(q + \log n)$
Single-Pair SP Query	$O(L + \log n)$	$O(L + \log n)$	$O(n)$	$O(L + q + \log n)$
Single-Source SP Tree Query	$O(n)$	$O(n)$	$O(n\sqrt{\log \log n})$	$O(n + q\sqrt{\log \log q})$
Update Time	$O(n \log n + q^2)$	$O(n + q^2)$	$O(\log^3 n)$	$O(\log n + \log^3 q)$

TABLE 1: Comparison of results for planar digraphs. L is the number of the edges of a shortest path (SP). To see how our results compare to known ones [6, 10, 14] for outerplanar digraphs or digraphs with $q = O(1)$, just remove all terms depending on q in the above table.