Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 171 (2007) 3–15
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Algorithmic Techniques for Maintaining Shortest Routes in Dynamic Networks

Camil Demetrescu[1,2]

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Roma, Italy

Giuseppe F. Italiano[1,3]

Dipartimento di Informatica, Sistemi e Produzione
Università di Roma "Tor Vergata"
Roma, Italy

Abstract

In this paper, we survey algorithms for shortest paths in dynamic networks. Although research on this problem spans over more than three decades, in the last couple of years many novel algorithmic techniques have been proposed. In this survey, we will make a special effort to abstract some combinatorial and algebraic properties, and some common data-structural tools that are at the base of those techniques. This will help us try to present some of the newest results in a unifying framework so that they can be better understood and deployed also by non-specialists.

Keywords: Dynamic networks, dynamic graph problems, dynamic shortest paths.

1 Introduction

A fundamental problem in communication networks is finding suitable routes for transmitting data packets. To minimize transmission time and cost, many routing schemes deliver packets along shortest routes. To speed up the task of finding optimal paths, routers are typically based on precomputed lookup tables. In many scenarios, however, the costs associated with network links and the structure of the

[1] Work supported in part by the Sixth Framework Programme of the EU under contract number 507613 (Network of Excellence "EuroNGI: Designing and Engineering of the Next Generation Internet"), and number 001907 ("DELIS : Dynamically Evolving, Large Scale Information Systems"), and by the Italian Ministry of University and Research (Project "ALGO-NEXT: Algorithms for the Next Generation Internet and Web: Methodologies, Design and Experiments").

[2] Email: demetres@dis.uniroma1.it

[3] Email: italiano@disp.uniroma2.it

doi:10.1016/j.entcs.2006.11.006

newtork itself may change dynamically over time, forcing continuous recalculations of the routing tables. A typical example is routing in *ad hoc* wireless networks, which are collections of wireless mobile hosts forming a temporary network without the aid of any established infrastructure or centralized administration. In such an environment, it may be necessary for one mobile host to find intermediate hosts in forwarding a packet to its destination, due to the limited range of each mobile hosts wireless transmissions. Transmission protocols need to adapt quickly to routing changes when host movement is frequent, while requiring little or no overhead during periods in which hosts move less frequently [18].

In this paper we survey a number of recent algorithmic techniques for maintaining shortest routes in dynamic graphs. For the sake of generality, we focus on the all-pairs version of the problem, where one is interested in maintaining information about shortest paths between each pair of nodes. Many of the techniques described in this paper can be specialized for the case where only shortest paths between a subset of nodes in the network have to be maintained.

A dynamic graph algorithm maintains a given property $\mathcal{P}$ on a graph subject to dynamic changes, such as edge insertions, edge deletions and edge weight updates. A dynamic graph algorithm should process queries on property $\mathcal{P}$ quickly, and perform update operations faster than recomputing from scratch, as carried out by the fastest static algorithm. We say that an algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: we say that it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only. In this paper, we survey fully dynamic algorithms for maintaining information about shortest paths.

In the *fully dynamic All Pairs Shortest Path (APSP) problem* we wish to maintain a directed graph $G = (V, E)$ with real-valued edge weights under an intermixed sequence of the following operations:

Update(x, y, w):	update the weight of edge (x, y) to the real value w; this includes as a special case both edge insertion (if the weight is set from $+\infty$ to $w < +\infty$) and edge deletion (if the weight is set to $w = +\infty$);
Distance(x, y):	output the shortest distance from x to y.
Path(x, y):	report a shortest path from x to y, if any.

Throughout the paper, we denote by m and by n the number of edges and vertices in G, respectively.

Although research on dynamic shortest paths spans over more than three decades, in the last couple of years we have witnessed a surprising resurge of interests in those two problems. The goal of this paper is to survey the newest algorithmic techniques that have been recently proposed in the literature. In particular, we will make a special effort to abstract some combinatorial and algebraic properties, and some common data-structural tools that are at the base of those techniques. This will help us try to present all the newest results in a unifying framework so that they

can be better understood and deployed also by non-specialists.

1.1 History of the Problem

The first papers on dynamic shortest paths date back to 1967 [20,22,25]. In 1985 Even and Gazit [7] and Rohnert [26] presented algorithms for maintaining shortest paths on directed graphs with arbitrary real weights. Their algorithms required $O(n^2)$ per edge insertion; however, the worst-case bounds for edge deletions were comparable to recomputing APSP from scratch. Also Ramalingam and Reps [23,24] considered dynamic shortest path algorithms with arbitrary real weights, but in a different model. Namely, the running time of their algorithm is analyzed in terms of the output change rather than the input size (*output bounded complexity*). Frigioni *et al.* [10,11] designed fast algorithms for graphs with bounded genus, bounded degree graphs, and bounded treewidth graphs in the same model. Again, in the worst case the running times of output-bounded dynamic algorithms are comparable to recomputing APSP from scratch.

Up to few years ago, there seemed to be few dynamic shortest path algorithms which were provably faster than recomputing APSP from scratch, and they only worked on special cases and with small integer weights. In particular, Ausiello *et al.* [1] proposed an incremental shortest path algorithm for directed graphs having positive integer weights less than C: the amortized running time of their algorithm is $O(Cn \log n)$ per edge insertion. Henzinger *et al.* [16] designed a fully dynamic algorithm for APSP on planar graphs with integer weights, with a running time of $O(n^{9/7} \log(nC))$ per operation. Fakcharoemphol and Rao in [9] designed a fully dynamic algorithm for single-source shortest paths in planar directed graphs that supports both queries and edge weight updates in $O(n^{4/5} \log^{13/5} n)$ amortized time per operation.

1.2 Novel Techniques for Dynamic Path Problems

Quite recently, many new algorithms for dynamic shortest path problems have been proposed. In particular, King [19] presented a fully dynamic algorithm for maintaining all pairs shortest paths in directed graphs with positive integer weights less than C: the running time of her algorithm is $O(n^{2.5}\sqrt{C \log n})$ per update. This algorithm is based on clever tree data structures. Demetrescu and Italiano [5] proposed a fully dynamic algorithm for maintaining APSP on directed graphs with arbitrary real weights. Given a directed graph G, subject to dynamic operations, and such that each edge weight can assume at most S different *real* values, their algorithm supports each update in $O(S \cdot n^{2.5} \log^3 n)$ amortized time and each query in optimal worst-case time. We remark that the sets of possible weights of two different edges need not be necessarily the same: namely, any edge can be associated with a different set of possible weights. The only constraint is that throughout the sequence of operations, each edge can assume at most S different real values, which seems to be the case in many applications. Differently from [19], this method uses dynamic reevaluation of products of real-valued matrices as the kernel for solving

dynamic shortest paths. Finally, the same authors [3] have studied some combinatorial properties of graphs that make it possible to devise a different approach to dynamic all pairs shortest paths problems. This approach yields a fully dynamic algorithm for general directed graphs with non-negative real-valued edge weights that supports any sequence of operations in $O(n^2 \log^3 n)$ amortized time per update and unit worst-case time per distance query, where n is the number of vertices. Shortest paths can be reported in optimal worst-case time. The algorithm is deterministic, uses simple data structures, and appears to be very fast in practice. Using the same approach, Thorup [27] has shown how to achieve $O(n^2(\log n + \log^2((m+n)/n)))$ amortized time per update and $O(mn)$ space. His algorithm works with negative weights as well. In [28], Thorup has shown how to achieve worst-case bounds at the price of a higher complexity: in particular, the update bounds become $\widetilde{O}(n^{2.75})$, where $\widetilde{O}(f(n))$ denotes $O(f(n) \cdot \text{polylog } n)$.

An extensive computational study on dynamic all pairs shortest path problems appears in [4].

1.3 Organization of the paper

The remainder of this paper is organized as follows. In Section 2 we describe some combinatorial properties of path problems on directed graphs and we abstract some data structures that are at the base of algorithmic techniques for dynamic path problems. Next, we focus on the newest dynamic shortest path algorithms in Section 3. In Section 4 we list some concluding remarks and open problems.

2 Algorithmic Techniques

In this section we describe some algorithmic techniques that are the kernel of the best known algorithms for maintaining shortest paths.

2.1 Long Paths

In this section we discuss an intuitive combinatorial property of long paths. Namely, if we pick a subset S of vertices at random from a graph G, then a sufficiently long path will intersect S with high probability.

To the best of our knowledge, this property was first given in [12], and later on it has been used may times in designing efficient algorithms for transitive closure and shortest paths (see e.g., [5,19,29,30]). The following theorem is from [29].

Theorem 2.1 *Let $S \subseteq V$ be a set of vertices chosen uniformly at random. Then the probability that a given simple path has a sequence of more than $\frac{cn \log n}{|S|}$ vertices, none of which are from S, for any $c > 0$, is, for sufficiently large n, bounded by $1/n^{\alpha c}$ for some positive α.*

As shown in [30], it is possible to choose set S deterministically by a reduction to a hitting set problem [2,21]. A similar technique has also been used in [19].

2.2 Locality

Recently, Demetrescu and Italiano [3] proposed a new approach to dynamic path problems based on maintaining classes of paths characterized by local properties, i.e., properties that hold for all proper subpaths, even if they may not hold for the entire paths. They showed that this approach can play a crucial role in the dynamic maintenance of shortest paths. For instance, they considered a class of paths defined as follows:

Definition 2.2 A path π in a graph is *locally shortest* if and only if every proper subpath of π is a shortest path.

This definition is inspired by the optimal-substructure property of shortest paths: all subpaths of a shortest path are shortest. However, a locally shortest path may not be shortest.

The fact that locally shortest paths include shortest paths as a special case makes them a useful tool for computing and maintaining distances in a graph. Indeed, paths defined locally have interesting combinatorial properties in dynamically changing graphs. For example, it is not difficult to prove that the number of locally shortest paths that may change due to an edge weight update is $O(n^2)$ if updates are partially dynamic, i.e., increase-only or decrease-only:

Theorem 2.3 *Let G be a graph subject to a sequence of increase-only or decrease-only edge weight updates. Then the amortized number of paths that start or stop being locally shortest at each update is $O(n^2)$.*

Unfortunately, Theorem 2.3 may not hold if updates are fully dynamic, i.e., increases and decreases of edge weights are intermixed. To cope with pathological sequences, a possible solution is to retain information about the history of a dynamic graph, considering the following class of paths:

Definition 2.4 A *historical shortest path* (in short, *historical path*) is a path that has been shortest at least once since it was last updated.

Here, we assume that a path is updated when the weight of one of its edges is changed. Applying the locality technique to historical paths, we derive locally historical paths:

Definition 2.5 A path π in a graph is *locally historical* if and only if every proper subpath of π is historical.

Like locally shortest paths, also locally historical paths include shortest paths, and this makes them another useful tool for maintaining distances in a graph:

Lemma 2.6 *If we denote by SP, LSP, and LHP respectively the sets of shortest paths, locally shortest paths, and locally historical paths in a graph, then at any time the following inclusions hold: $SP \subseteq LSP \subseteq LHP$.*

Differently from locally shortest paths, locally historical paths exhibit interesting combinatorial properties in graphs subject to fully dynamic updates. In particular,

it is possible to prove that the number of paths that become locally historical in a graph at each edge weight update depends on the number of historical paths in the graph.

Theorem 2.7 *(Demetrescu and Italiano [3]) Let G be a graph subject to a sequence of update operations. If at any time throughout the sequence of updates there are at most $O(h)$ historical paths in the graph, then the amortized number of paths that become locally historical at each update is $O(h)$.*

To keep changes in locally historical paths small, it is then desirable to have as few historical paths as possible. Indeed, it is possible to transform every update sequence into a slightly longer equivalent sequence that generates only a few historical paths. In particular, there exists a simple *smoothing* strategy that, given any update sequence Σ of length k, produces an operationally equivalent sequence $F(\Sigma)$ of length $O(k \log k)$ that yields only $O(\log k)$ historical shortest paths between each pair of vertices in the graph. We refer the interested reader to [3] for a detailed description of this smoothing strategy. According to Theorem 2.7, this technique implies that only $O(n^2 \log k)$ locally historical paths change at each edge weight update in the smoothed sequence $F(\Sigma)$.

As elaborated in [3], locally historical paths can be maintained very efficiently. Since by Lemma 2.6 locally historical paths include shortest paths, this yields the fastest known algorithm for fully dynamic all pairs shortest paths.

2.3 Tools for Trees

In this section we describe a tree data structure for keeping information about dynamic path problems. The first appearance of this tool dates back to 1981, when Even and Shiloach [8] showed how to maintain a breadth-first tree of an undirected graph under any sequence of edge deletions; they used this as a kernel for decremental connectivity on undirected graphs. Later on, Henzinger and King [13] showed how to adapt this data structure to fully dynamic transitive closure in directed graphs. Recently, King [19] designed an extension of this tree data structure to weighted directed graphs for solving fully dynamic all pairs shortest paths.

The Problem.

The goal is to maintain information about breadth-first search (BFS) on an undirected graph G undergoing deletions of edges. In particular, in the context of dynamic path problems, we are interested in maintaining BFS trees of depth up to d, with $d \leq n$. For the sake of simplicity, we describe only the case where deletions do not disconnect the underlying graph. The general case can be easily handled by means of "phony" edges (i.e., when deleting an edge that disconnects the graph, we just replace it by a phony edge).

It is well known that BFS partitions vertices into levels, so that there can be edges only between adjacent levels. More formally, let r be the vertex where we start BFS, and let level ℓ_i contains vertices encountered at distance i from r ($\ell_0 = \{r\}$): edges incident to a vertex at level ℓ_i can have their other endpoints either at level

ℓ_{i-1}, ℓ_i, or ℓ_{i+1}, and no edge can connect vertices at levels ℓ_i and ℓ_j for $|j - i| > 1$. Let $\ell(v)$ be the level of vertex v.

Given an undirected graph $G = (V, E)$ and a vertex $r \in V$, we would like to support any intermixed sequence of the following operations:

- `Delete`(x, y): delete edge (x, y) from G.
- `Level`(u): return the level $\ell(u)$ of vertex u in the BFS tree rooted at r (return $+\infty$ if u is not reachable from r).

In the remainder of this paper, to indicate that an operation `Y`() is performed on a data structure `X`, we use the notation `X.Y`().

Data Structure.
We maintain information about BFS throughout the sequence of edge deletions by simply keeping explicitly those levels. In particular, for each vertex v at level ℓ_i in T, we maintain the following data structures: UP(v), SAME(v) and DOWN(v) containing the edges connecting v to level ℓ_{i-1}, ℓ_i, and ℓ_{i+1}, respectively. Note that for all $v \neq r$, UP(v) must contain at least one edge (i.e., the edge from v to its parent in the BFS tree). In other words, a non-empty UP(v) witnesses the fact that v is actually entitled to belong to that level. This property will be important during edge deletions: whenever UP(v) gets emptied because of deletions, v looses its right to be at that level and must be demoted at least one level down.

Implementation of Operations.
When edge (x, y) is being deleted, we proceed as follows. If $\ell(x) = \ell(y)$, simply delete (x, y) from SAME(y) and from SAME(x). The levels encoded in UP, SAME and DOWN still capture the BFS structure of G. Otherwise, without loss of generality let $\ell(x) = \ell_{i-1}$ and $\ell(y) = \ell_i$. Update the sets UP, SAME and DOWN by deleting x from UP(y) and y from DOWN(x). If UP(y) $\neq \emptyset$, then there is still at least one edge connecting y to level ℓ_{i-1}, and the levels will still reflect the BFS structure of G after the deletion.

The main difficulty is when UP(y) $= \emptyset$ after the deletion of (x, y). In this case, deleting (x, y) causes y to loose its connection to level ℓ_{i-1}. Thus, y has to drop down at least one level. Furthermore, its drop may cause a deeper landslide in the levels below. This case can be handled as follows.

We use a FIFO queue Q, initialized with vertex y. We will insert a vertex v in the queue Q whenever we discover that UP(v) $= \emptyset$, i.e., vertex v has to be demoted at least one level down. We will repeat the following demotion step until Q is empty:

Demotion Step :
(i) Remove the first vertex in Q, say v.
(ii) Delete v from its level $\ell(v) = \ell_i$ and tentatively try to place v one level down, i.e., in ℓ_{i+1}.
(iii) Update the sets UP, SAME and DOWN consequently:
  (a) For each edge (u, v) in SAME(v), delete (u, v) from SAME(u) and insert (u, v) in DOWN(u) and UP(v) (as UP(v) was empty,

this implies that UP(v) will be initialized with the old set SAME(v)).

(b) For each edge (v, z) in DOWN(v), move edge (v, z) from UP(z) to SAME(z) and from DOWN(v) to SAME(v); if the new UP(z) is empty, insert z in the queue Q. Note that this will empty DOWN(v).

(c) If UP(v) is still empty, insert v again into Q.

Analysis.
It is not difficult to see that applying the Demotion Step until the queue is empty will maintain correctly the BFS levels. Level queries can be answered in constant time. To bound the total time required to process any sequence of edge deletions, it suffices to observe that each time an edge (u, v) is examined during a demotion step, either u or v will be dropped one level down. Thus, edge (u, v) can be examined at most $2d$ times in any BFS levels up to depth d throughout any sequence of edge deletions. This implies the following theorem.

Theorem 2.8 *Maintaining BFS levels up to depth d requires $O(md)$ time in the worst case throughout any sequence of edge deletions in an undirected graph with m initial edges.*

This means that maintaining BFS levels requires d times the time needed for constructing them. Since $d \leq n$, we obtain a total bound of $O(mn)$ if there are no limits on the depth of the BFS levels.

◁ ◇ ▷

As it was shown in [13,19], it is possible to extend the BFS data structure presented in this section to deal with directed graphs with small integer edge weights. In this case, a shortest path tree is maintained in place of BFS levels: after each edge deletion or edge weight increase, the tree is reconnected by essentially mimicking Dijkstra's algorithm rather than BFS. Details can be found in [19].

3 Dynamic Shortest Paths

In this section we survey the best known algorithms for fully dynamic shortest paths. We start with a formal definition of the fully dynamic all pairs shortest paths problem. Next, we survey the algorithm by King [19], whose main ingredients are the long paths property of Section 2.1 and the tools for trees described in Section 2.3. This methods yields $O(n^{2.5}\sqrt{C \cdot \log n})$ amortized time per update and $O(1)$ per query in graphs with positive integer weights less than C. Finally, we describe the algorithm by Demetrescu and Italiano [3], which is based on the locality properties described in Section 2.2. Using this method, updates are supported in $O(n^2 \log^3 n)$ amortized time and distance queries are answered in $O(1)$ in graphs with non-negative real edge weights.

The Problem.
Let $G = (V, E)$ be a weighted directed graph. We consider the problem of maintaining a data structure for G under an intermixed sequence of update and query operations of the following kinds:

- `Decrease`(v, w): decrease the weight of edges incident to v in G as specified by a new weight function w. We call this kind of update a v-CENTERED decrease in G.
- `Increase`(w): increase the weight of edges in G as specified by a new weight function w.
- `Query`(x, y): return the distance between x and y in G.

We consider generalized update operations where we modify a whole set of edges, rather than a single edge. Also, we do not address the issue of returning actual paths between vertices, and we just consider the problem of answering distance queries.

3.1 Tree-based Dynamic Shortest Paths with Stitching

In this section we describe how to maintain all pairs shortest paths in a directed graph with non-negative integer edge weights less than C in $O(n^{2.5}\sqrt{C \log n})$ amortized time per update operation. The algorithm that we describe has been designed by King [19] and it builds on the tree data structure presented in Section 2.3 and on the long paths property described in Section 2.1.

Data Structure.
Given a weighted directed graph G, we maintain for each vertex v:

- a shortest paths tree Out_v of G of depth $d \leq \sqrt{nC \log n}$ rooted at v;
- a shortest paths tree In_v of $\widehat{G}$ of depth $d \leq \sqrt{nC \log n}$ rooted at v, where $\widehat{G}$ is equal to G, except for the orientation of edges, which is reversed;
- a set S containing $\sqrt{nC \log n}$ vertices of G chosen uniformly at random, referred to as "blockers";
- a complete weighted directed graph G_S with vertex set S such that, with very high probability, the weight of (x, y) in G_S is equal to the distance between x and y in G. For the long paths property given in Section 2.1, these weights are no greater than d.
- an integer distance matrix *dist*.

We maintain the trees with instances of the data structure presented in Section 2.3, adapted to deal with weighted directed graphs and to include only short paths, i.e., vertices of distance up to d from the root. Information about longer paths will be obtained by stitching together these short paths.

Implementation of Operations.
The main idea of the algorithm is to exploit the long paths property of Section 2.1, maintaining dynamically only shortest paths of (weighted) length up to $\sqrt{C \log n}$

with the data structure presented in Section 2.3, and stitching together these paths to update longer paths using any static $O(n^3)$ all pairs shortest paths algorithm on a contraction with $O(\sqrt{nC\log n})$ vertices of the original graph. Operations are realized as follows:

- `Decrease`(v, w): rebuild In_v and Out_v to update G_S, i.e., paths of length up to d. Apply the algorithm `Stitch` below to update longer paths.
- `Increase`(w): update edges with increased weight in any In_v and Out_v that contain them, and then update G_S, i.e., paths of length up to d. Apply the algorithm `Stitch` below to update longer paths.
- `Query`(x, y): return $dist(x, y)$.

Details about the stitching algorithm are given below:

- `Stitch`():

(i) Let $dist^{(d)}(x, y)$ be the distance from x to y of length up to d, obtained from all the trees In_v and Out_v.

(ii) Compute the distances $dist()$ between all vertices in S using any static $O(n^3)$ APSP algorithm on G_S.

(iii) Compute the distances from vertices in V to vertices in S. This can be done for a pair $x \in V$ and $s \in S$ by computing

$$dist(x, s) \leftarrow \min\{dist^{(d)}(x, s),\ \min_{s' \in S}\{dist^{(d)}(x, s') + dist(s', s)\}.$$

(iv) Compute the distances between vertices in V. This can be done for a pair $x, y \in V$ by computing

$$dist(x, y) \leftarrow \min\{dist^{(d)}(x, y),\ \min_{s \in S}\{dist(x, s) + dist^{(d)}(s, y)\}.$$

Analysis.
The stitching algorithm is dominated by the last step, which takes time $O(n^2|S|) = O(n^2(n \log n/d))$. Shortest path trees of length up to d can be maintained in $O(n^2 d)$ amortized time with the data structure of Section 2.3. Choosing $d = \Theta(\sqrt{nC\log n})$ yields an amortized update bound of $O(n^{2.5}\sqrt{C\log n})$.

Theorem 3.1 *Any* `Decrease` *and* `Increase` *operation requires* $O(n^{2.5}\sqrt{C\log n})$ *amortized time, and* `Query` *can be answered in constant time.*

The set S can be computed deterministically, as illustrated in [19]. This makes the whole algorithm deterministic with the same bounds.

3.2 Locality-based Dynamic Shortest Paths

In this section we address the algorithm by Demetrescu and Italiano [3], who devised the first deterministic near-quadratic update algorithm for fully dynamic all-pairs shortest paths. This algorithm is also the first solution to the problem in its generality. The algorithm is based on the notions of historical paths and locally historical

paths in a graph subject to a sequence of updates, as discussed in Section 2.2.

The main idea is to maintain dynamically the locally historical paths of the graph in a data structure. Since by Lemma 2.6 shortest paths are locally historical, this guarantees that information about shortest paths is maintained as well.

To support an edge weight update operation, the algorithm implements the smoothing strategy mentioned in Section 2.2 and works in two phases. It first removes from the data structure all maintained paths that contain the updated edge: this is correct since historical shortest paths, in view of their definition, are immediately invalidated as soon as they are touched by an update. This means that also potentially uniform paths that contain them are invalidated and have to be removed from the data structure. As a second phase, the algorithm runs an all-pairs modification of Dijkstra's algorithm [6], where at each step a shortest path with minimum weight is extracted from a priority queue and it is combined with existing historical shortest paths to form new potentially uniform paths. At the end of this phase, paths that become potentially uniform after the update are correctly inserted in the data structure.

The update algorithm spends constant time for each of the $O(zn^2)$ new potentially uniform path (see Theorem 2.7). Since the smoothing strategy lets $z = O(\log n)$ and increases the length of the sequence of updates by an additional $O(\log n)$ factor, this yields $O(n^2 \log^3 n)$ amortized time per update. The interested reader can find further details about the algorithm in [3].

Using the same approach, but with a different smoothing strategy, Thorup [27] has shown how to achieve $O(n^2(\log n + \log^2(m/n)))$ amortized time per update and $O(mn)$ space. His algorithm works with negative weights as well.

4 Conclusions and Open Problems

In this paper we have surveyed the newest developments in the area of fully dynamic algorithms for shortest paths. Throughout the paper, we have attempted to present all the algorithmic techniques within a unifying framework by abstracting the algebraic and combinatorial properties and the data-structural tools that lie at their foundations.

This bulk of recent work has raised some new and perhaps intriguing questions. First, can we reduce the space usage for dynamic shortest paths to $O(n^2)$? Second, and perhaps more importantly, can we solve efficiently fully dynamic *single-source* reachability and shortest paths on general graphs? Finally, are there any general techniques for making increase-only algorithms fully dynamic? Similar techniques have been widely exploited in the case of fully dynamic algorithms on undirected graphs [14,15,17].

References

[1] G. Ausiello, G.F. Italiano, A. Marchetti-Spaccamela, and U. Nanni. Incremental algorithms for minimal length paths. *Journal of Algorithms*, 12(4):615–38, 1991.

[2] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[3] C. Demetrescu and G. F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the Association for Computing Machinery (JACM)*, 51(6):968–992, 2004.

[4] C. Demetrescu and G. F. Italiano. Experimental analysis of dynamic all pairs shortest path algorithms. *To appear in ACM Transactions on Algorithms*, 2005. Special issue devoted to the best papers selected from the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04).

[5] C. Demetrescu and G.F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. *To appear in Journal of Computer and System Sciences*, 2005.

[6] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

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

[8] S. Even and Y. Shiloach. An on-line edge-deletion problem. *Journal of the ACM*, 28:1–4, 1981.

[9] J. Fakcharoemphol and S. Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. In *Proc. of the 42nd IEEE Annual Symposium on Foundations of Computer Science (FOCS'01), Las Vegas, Nevada*, pages 232–241, 2001.

[10] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Semi-dynamic algorithms for maintaining single source shortest paths trees. *Algorithmica*, 22(3):250–274, 1998.

[11] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms*, 34:351–381, 2000.

[12] D. H. Greene and D.E. Knuth. *Mathematics for the analysis of algorithms*. Birkhäuser, 1982.

[13] M. Henzinger and V. King. Fully dynamic biconnectivity and transitive closure. In *Proc. 36th IEEE Symposium on Foundations of Computer Science (FOCS'95)*, pages 664–672, 1995.

[14] M. Henzinger and V. King. Maintaining minimum spanning forests in dynamic graphs. *SIAM Journal on Computing*, 31(2):364–374, 2001.

[15] M.R. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, 1999.

[16] M.R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, August 1997.

[17] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48:723–760, 2001.

[18] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[19] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS'99)*, pages 81–99, 1999.

[20] P. Loubal. A network evaluation procedure. *Highway Research Record 205*, pages 96–109, 1967.

[21] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[22] J. Murchland. The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph. Technical report, LBS-TNT-26, London Business School, Transport Network Theory Unit, London, UK, 1967.

[23] G. Ramalingam and T. Reps. An incremental algorithm for a generalization of the shortest path problem. *Journal of Algorithms*, 21:267–305, 1996.

[24] G. Ramalingam and T. Reps. On the computational complexity of dynamic graph problems. *Theoretical Computer Science*, 158:233–277, 1996.

[25] V. Rodionov. The parametric problem of shortest distances. *U.S.S.R. Computational Math. and Math. Phys.*, 8(5):336–343, 1968.

[26] H. Rohnert. A dynamization of the all-pairs least cost problem. In *Proc. 2nd Annual Symposium on Theoretical Aspects of Computer Science, (STACS'85), LNCS 182*, pages 279–286, 1985.

[27] M. Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT'04)*, pages 384–396, 2004.

[28] M. Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the 37th ACM Symposium on Theory of Computing (STOC 2005)*, 2005.

[29] J.D. Ullman and M. Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM Journal on Computing*, 20(1):100–125, 1991.

[30] U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002.