Alcom-FT Technical Report Series
ALCOMFT-TR-02-1

Improved Bounds and New Trade-Offs for Dynamic All Pairs Shortest Paths *

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

Giuseppe F. Italiano ‡
Dipartimento di Informatica, Sistemi e Produzione
Università di Roma "Tor Vergata", Roma, Italy

Abstract

Let G be a directed graph with n vertices, subject to dynamic updates, and such that each edge weight can assume at most S different arbitrary real values throughout the sequence of updates. We present a new algorithm for maintaining all pairs shortest paths in G in $O(S^{0.5} \cdot n^{2.5} \log^{0.5} n)$ amortized time per update and in $O(1)$ worst-case time per distance query. This improves over previous bounds. We also show how to obtain query/update trade-offs for this problem, by introducing two new families of algorithms. Algorithms in the first family achieve an update bound of $\widetilde{O}(S \cdot k \cdot n^2)$ [1] and a query bound of $\widetilde{O}(n/k)$, and improve over the best known update bounds for k in the range $(n/S)^{1/3} \leq k < (n/S)^{1/2}$. Algorithms in the second family achieve an update bound of $\widetilde{O}\left(S \cdot k \cdot n^2\right)$ and a query bound of $\widetilde{O}(n^2/k^2)$, and are competitive with the best known update bounds (first family included) for k in the range $(n/S)^{1/6} \leq k < (n/S)^{1/3}$.

1 Introduction

All Pairs Shortest Path (APSP) is perhaps one of the most fundamental algorithmic graph problems. The fastest static algorithm for APSP on graphs with arbitrary real weights is achieved with the Fibonacci heaps of Fredman and Tarjan [8] and has a running time of $O(mn + n^2 \log n)$, where m is the number of edges and n is the number of vertices in the graph. This is $\Omega(n^3)$ in the worst case. Fredman [7] and later Takaoka [26] showed how to break this cubic barrier: the best asymptotic bound is by Takaoka, who showed how to solve APSP in $O(n^3\sqrt{\log\log n / \log n})$ time.

The quest for faster subcubic algorithms has led to many new results for APSP. In particular, Alon *et al.* [1] were the first to show that fast matrix multiplication can be effectively used for solving APSP on directed graphs with small integer weights: they

*Work partially supported by the IST Programme of the EU under contract n. IST-1999-14.186 (ALCOM-FT) and by CNR, the Italian National Research Council, under contract n. 01.00690.CT26.

†Email: `demetres@dis.uniroma1.it`. URL: `http://www.dis.uniroma1.it/~demetres`.

‡Email: `italiano@info.uniroma2.it`. URL: `http://www.info.uniroma2.it/~italiano`.

[1] Throughout the paper, we use $\widetilde{O}(f(n))$ to denote $O(f(n)\mathrm{polylog}(n))$.

gave an algorithm whose running time is $\tilde{O}(n^{(3+\omega)/2})$ for APSP on directed graphs whose weights are integers in the set $\{-1, 0, 1\}$, where ω is the best known exponent for matrix multiplication: currently, $\omega < 2.376$ [2]. Galil and Margalit [11, 12] and Seidel [24] gave $\tilde{O}(n^{\omega})$ algorithms for solving APSP for unweighted undirected graphs whose weights are integers in the set $\{-1, 0, 1\}$. Shoshan and Zwick [25] and Zwick [28] achieve the best known bound for APSP with positive integer edge weights less than C: $\tilde{O}(Cn^{\omega})$ for undirected graphs [25] and $O(C^{0.681}n^{2.575})$ for directed graphs [28]. All these subcubic algorithms are based on clever reductions to fast matrix multiplication.

In this paper we consider fully dynamic algorithms for maintaining all pairs shortest paths on directed graphs with *real-valued* edge weights. 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 $\mathcal{P}$, and must 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 the fully dynamic 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:

`Increase`(w): increase the edge weights as specified by the cost function $w : E \to \mathcal{R} \cup \{+\infty\}$. Note that the same operation can increase the cost of many edges simultaneously.

`Decrease`(v, w): decrease the weights of edges incident to vertex v as specified by the cost function $w : E \to \mathcal{R} \cup \{+\infty\}$. Note that the same operation can decrease the cost of many edges incident to v simultaneously.

`Distance`(x, y): output the shortest distance from x to y.

`Path`(x, y): report a shortest path from x to y, if any.

We note that edge deletions are a special case of `Increase`, whenever the new weights are set to $+\infty$. Similarly, edge insertions are a special case of `Decrease`. Throughout the paper, we denote by n the number of vertices in G.

Previous Work. The dynamic maintenance of shortest paths has been investigated for over three decades, as the first papers date back to 1967 [18, 19, 22]. In 1985 Even and Gazit [5] and Rohnert [23] 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 [20, 21] 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.* [9, 10] 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.

Recently several dynamic shortest path algorithms, which are provably faster than recomputing APSP from scratch, were proposed. In particular, Henzinger *et al.* [15] 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 edge insertion or deletion. This bound has been improved by Fakcharoemphol and Rao in [6], who 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 edge operation. King [16] 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 $O(C^{0.5} \cdot n^{2.5}\log^{0.5} n)$ per update. The space required is $O(n^3)$: a simple method for reducing space to $O(n^2)$ is shown in [17]. Finally, Demetrescu and Italiano [4] showed how to maintain APSP on directed graphs with arbitrary real weights. In particular, given a direct graph G, subject to dynamic operations, and such that each edge weight can assume at most S different *real* values, they give a deterministic algorithm that supports each update in $O(S \cdot n^{2.5}\log^3 n)$ amortized time and each query in $O(1)$ worst-case time.

The best known bounds for dynamic all pairs shortest paths on general graphs are thus $O(1)$ worst-case time per distance query and $\widetilde{O}(C^{0.5} \cdot n^{2.5})$ amortized time per update in case of integer weights in the range $[0, C]$ [16], or $\widetilde{O}(S \cdot n^{2.5})$ amortized time per update in case of arbitrary real-valued weights (with at most S different values per edge) [4]. Note that update bounds of [4] are slightly worse than the update bounds of [16] when restricted to positive integer edge weights, and that in both cases there is a large gap in the running times of distance queries and updates. In this scenario, it seems quite natural to ask whether the bounds in [4] can be improved so as to match the bounds in [16] and whether it is possible to reduce the (high) update bounds at the expenses of increasing the distance query times. The latter question seems particularly important in many applications, in which after a distance query one is typically required to exhibit an actual shortest path realizing that distance. Since reporting a shortest path might require as much as $\Omega(n)$ time, then one might as well afford to slow down the distance queries in order to speed the updates up. We remark that, despite many decades of research in the area of dynamic graph problems, there seem to be very few query/update trade-offs available in the literature (see e.g., [3, 14]).

Our Results. In this paper, we present a new fully dynamic shortest path algorithm for directed graphs and arbitrary real weights (with at most S different values per edge weight), which achieves $O(S^{0.5} \cdot n^{2.5}\log^{0.5} n)$ amortized time per update and $O(1)$ worst-case time per query. This improves over [4] and matches the bounds of [16] in the special case of integer weights. Furthermore, we make a first step towards the study of effective query/update trade-offs in dynamic shortest paths problems. In particular, we introduce two new families of fully dynamic algorithms for maintaining APSP on directed graphs, which are able to obtain the following bounds:

1 Algorithms in the first family achieve an update bound of $\widetilde{O}(S \cdot k \cdot n^2)$ and a query bound of $\widetilde{O}(n/k)$, for any integer k in the range $(n/S)^{1/3} \leq k \leq n$. Note that this improves the best known update bounds (above algorithm included) whenever k is in the range $(n/S)^{1/3} \leq k < (n/S)^{1/2}$. The fastest update time for this family of algorithms can be obtained by choosing $k = (n/S)^{1/3}$: this yields $\widetilde{O}(S^{2/3} \cdot n^{7/3}) = \widetilde{O}(S^{0.667} \cdot n^{2.333})$ amortized time per update and $\widetilde{O}(S^{1/3} \cdot n^{2/3}) = \widetilde{O}(S^{0.333} \cdot n^{0.667})$

worst-case time per query.

2 Algorithms in the second family are able to achieve an update bound of $\widetilde{O}\,(S \cdot k \cdot n^2)$ and a query bound of $\widetilde{O}(n^2/k^2)$, for any parameter k in the range $(n/S)^{1/6} \leq k \leq n$. This is competitive with the best known update bounds (first family included) whenever k is in the range $(n/S)^{1/6} \leq k < (n/S)^{1/3}$. The fastest update time for this family of algorithms is obtained for $k = (n/S)^{1/6}$: in this case, we get $\widetilde{O}(S^{5/6} \cdot n^{13/6}) = \widetilde{O}(S^{0.833} \cdot n^{2.167})$ amortized time per update and $\widetilde{O}(S^{1/3} \cdot n^{5/3}) = \widetilde{O}(S^{0.333} \cdot n^{1.667})$ worst-case time per query.

All our algorithms are able to report a shortest path with ℓ edges in time $O(\ell)$ plus the time for the corresponding distance query. Like other dynamic shortest paths algorithms, their space usage is $O(n^3)$. To achieve our results, we find a way to combine in a novel fashion the matrix techniques of [4] and the path decomposition techniques of [16]. We remark that our algorithms do not need fast matrix multiplication, use simple data structures, and thus seem amenable to efficient implementations.

2 Short Paths and Long Paths

All our algorithms maintain track of paths according to their length. In particular, they keep explicit information about shortest paths with at most k edges. This can be done in $O(S \cdot k \cdot n^2 \log^2 n)$ amortized time per update, where S is the number of different real values that each edge can assume, and in $O(1)$ worst-case time per distance query.

We now sketch few details about how these bounds can be achieved. We denote the operations as `k-Decrease`, `k-Increase`, and `k-Query`. We use a standard logarithmic path decomposition by maintaining $\log k$ levels: level i includes information about all (but not only) the shortest paths with at most 2^i edges. The main idea is to keep $\log k$ polynomials $P_i(Y) = Y^3$, $1 \leq i \leq \log k$, over the $\{\min, +\}$ semiring with instances of the data structure presented in [4]. The degree 3 for the polynomial $P_i(Y) = Y^3$ is used in a dynamic setting exactly as described in [3] for the simpler problem of fully dynamic transitive closure. This yields the following bounds:

Theorem 1 *Any* `k-Decrease` *and* `k-Increase` *operation requires $O(S \cdot k \cdot n^2 \log^2 n)$ amortized time and any* `k-Query` *is answered in $O(1)$ worst-case time.*

For paths with more than k edges, we exploit the following combinatorial property: if a subset H of vertices is picked at random from a graph G, then a sufficiently long path will intersect H with high probability. This property appears already in [13], and later on it has been used may times in designing efficient algorithms for transitive closure and shortest paths (see e.g., [4, 14, 16, 27, 28]). The following theorem is from [27].

Theorem 2 *Let $H \subseteq V$ be a set of vertices chosen uniformly at random. Then the probability that a given simple path has a sequence of more than $\frac{cn}{|H|} \log n$ vertices, none of which are from H, for any $c > 0$, is, for sufficiently large n, bounded by $2^{-\alpha c}$ for some positive α.*

3 Dynamic Shortest Paths with $O(1)$ Queries

In this section we show how to improve the bounds of [4] for dynamic all pairs shortest paths in directed graphs where each edge can assume at most S different real values. In particular, we achieve an amortized update time of $O(S^{0.5} \cdot n^{2.5} \log^{0.5} n)$ and a query time of $O(1)$ in the worst case. If the shortest path has at most $(n/S)^{1/2}$ edges, then the query is answered correctly; otherwise it is answered correctly with probability $1 - 1/n^c$, for any constant $c > 0$. These bounds can be made deterministic.

Data Structure. Given a weighted directed graph $G = (V, E, w)$, we denote by X the weight matrix of G. We recall that the Kleene closure X^* of X is the distance matrix of G. We maintain the following elementary data structures.

1. A set $H \subseteq V$ of vertices chosen uniformly at random, with $|H| = \frac{cn}{k} \log n$, for any constant $c > 0$, with $c \log n \leq k \leq n$.

2. An $n \times n$ matrix D such that $D[x, y]$ is the shortest distance from $u \in V$ to $v \in V$ in G using at most k edges. We maintain D using an instance of the data structure given in Section 2. We denote by A the $n \times |H|$ matrix obtained from D by considering only the columns corresponding to vertices in H. We denote by B the $|H| \times |H|$ matrix obtained from D by considering only the rows and the columns corresponding to vertices in H. Finally, we denote by C the $|H| \times n$ matrix obtained from D by considering only the rows corresponding to vertices in H.

3. An $n \times |H|$ matrix $AB^* = A \cdot B^*$.

4. An $n \times n$ matrix $AB^*C = A \cdot B^* \cdot C$.

Our data structure uses the same path decomposition as in [16], i.e., $X^* = AB^*C \oplus D$, where $\oplus$ is defined as follows: $(X \oplus Y)[u, v] = \min_h \{X[u, h] + Y[h, v]\}$. The definitions of matrices A, B, C, and D imply that: $A[u, h]$ is the shortest distance from $u \in V$ to $h \in H$ in G using at most k edges, $B^*[h, h']$ is the shortest distance from $h \in H$ to $h' \in H$ in G, and $C[h', v]$ is the shortest distance from $h' \in H$ to $v \in V$ in G using at most k edges. By Theorem 2, $AB^*[u, h']$ gives the shortest distance from $u \in V$ to $h' \in H$, and $(AB^*C \oplus D)[u, v]$ gives the shortest distance from u to v in G: both these distances are correct with high probability.

Implementation of Operations. Matrix D is maintained dynamically using the operations described in Section 2. Matrices B^*, AB^*, and AB^*C are recomputed from scratch, while queries about entries of X^* are computed by accessing AB^*C and D. Operations can be supported as follows.

- `Decrease`(v, w): update D by calling `k-Decrease`(v, w). Next, recompute B^*, AB^*, and AB^*C from scratch.

- `Increase`(w): update D by calling `k-Increase`(w). Next, recompute B^*, AB^*, and AB^*C from scratch.

- `Query`(x, y): return $\min \{D[x, y], (AB^*C)[x, y]\}$.

Analysis. We now discuss the running time of our algorithm.

Theorem 3 *Any* `Decrease` *and* `Increase` *operation can be supported in* $O(S^{0.5} \cdot n^{2.5} \log^{0.5} n)$ *amortized time, while any* `Query` *operation can be answered in* $O(1)$ *worst-case time.*

Proof. Recall that $|H| = \frac{cn}{k} \log n$. The bound for updates is given by summing up the times required for updating D, and for recomputing B^*, AB^*, and AB^*C from scratch.

1) D: By Theorem 1, the cost of updating D is $\widetilde{O}(S \cdot k \cdot n^2)$.

2) B^*: Recomputing B^* from scratch can be done trivially in $O(|H|^3) = O((n^3/k^3) \cdot \log^3 n)$ using any cubic static algorithm for APSP.

3) AB^*: Recomputing $A \cdot B^*$ from scratch takes time $O(n \cdot |H|^2) = O((n^3/k^2) \cdot \log^2 n)$.

4) AB^*C: Recomputing $AB^* \cdot C$ from scratch takes time $O(n^2 \cdot |H|) = O((n^3/k) \cdot \log n)$.

Choosing $k = (\frac{n}{S} \cdot \log n)^{1/2}$ yields the claimed bounds. □

4 Trading off Updates and Queries

In this section we show how to achieve new trade-offs in dynamic shortest paths problems. In particular, we present two families of fully dynamic algorithms for maintaining all pairs shortest paths in directed graphs where each edge can assume at most S different real values.

4.1 Family $\mathcal{F}_1$

For the first family, we achieve an update time of $\widetilde{O}(S \cdot k \cdot n^2)$ and a query time of $\widetilde{O}(n/k)$, for any $(n/S)^{1/3} \leq k \leq n$. If the shortest path has at most k edges, then the query is answered correctly; otherwise it is answered correctly with probability $1 - 1/n^c$, for any constant $c > 0$.

Data Structure. We maintain the same data structures as in Section 3, with the only difference that now we do not maintain explicitly the matrix AB^*C:

1. A set $H \subseteq V$ of vertices chosen uniformly at random, with $|H| = \frac{cn}{k} \log n$, for any constant $c > 0$, with $c \log n \leq k \leq n$.

2. An $n \times n$ matrix D such that $D[x, y]$ is the shortest distance from $u \in V$ to $v \in V$ in G using at most k edges. We maintain D using an instance of the data structure given in Section 2. We denote by A the $n \times |H|$ matrix obtained from D by considering only the columns corresponding to vertices in H. We denote by B the $|H| \times |H|$ matrix obtained from D by considering only the rows and the columns corresponding to vertices in H. Finally, we denote by C the $|H| \times n$ matrix obtained from D by considering only the rows corresponding to vertices in H.

3. An $n \times |H|$ matrix $AB^* = A \cdot B^*$.

Implementation of Operations. Once again, information about paths with more than k edges is encoded implicitly in the product of matrices $AB^* \cdot C$, while distances realized with at most k edges are maintained explicitly in D. As in Section 3, matrix D is maintained dynamically using the operations described in Section 2, while matrix AB^* is recomputed from scratch. The main difference is that matrix B^* is now updated more efficiently by plugging in the algorithm of Section 3. Since the product of matrices $AB^* \cdot C$ is not recomputed explicitly, queries about entries of X^* are now answered by accessing on demand matrices AB^* and C. In more details, operations are supported as follows.

- `Decrease`(v, w): update D by calling `k-Decrease`(v, w). Next, perform a v-centered decrease on B^* using the algorithm of Section 3. Finally, recompute AB^* from scratch.

  Note that when $v \notin H$, we need to maintain H dynamically. This can be done similarly to [4]. Namely, add v to H; to keep H random, add another random vertex to H; to prevent H from getting bigger, remove the two oldest vertices from H, and update B^* accordingly.

- `Increase`(w): update D by calling `k-Increase`(w), perform an increase operation on B^* using the algorithm of Section 3, and then recompute AB^* from scratch.

- `Query`(x, y): return

$$\min \left\{ D[x,y], \min_{h \in H} \{ AB^*[x,h] + C[h,y] \} \right\}.$$

Analysis. We now discuss the running time of our algorithm.

Theorem 4 *Let* $(n/S)^{1/3} \le k \le n$. *Any* `Decrease` *and* `Increase` *operation can be supported in* $\widetilde{O}(S \cdot k \cdot n^2)$ *amortized time, while any* `Query` *operation can be answered in* $\widetilde{O}(\frac{n}{k})$ *worst-case time.*

Proof. We first observe that $|H| = \frac{cn}{k} \log n$ and that entries of D (and thus those of B) cannot assume more than $S \cdot k$ different real values. The bound for updates is given by summing up the times required for updating D, updating B^*, and recomputing AB^*.

1. By Theorem 1, the cost of updating D is $\widetilde{O}(S \cdot k \cdot n^2)$.

2. As shown in Section 3, maintaining all pairs shortest paths in a graph with N vertices and edges taking at most s different values requires $\widetilde{O}(s^{0.5} \cdot N^{2.5})$ amortized time. Since entries of B cannot assume more than $s = S \cdot k$ different values and B has size $|H| \times |H|$, then the time required for updating B^* is $\widetilde{O}(S^{0.5} \cdot k^{0.5} \cdot n^{2.5}/k^{2.5}) = \widetilde{O}(S^{0.5} \cdot n^{2.5}/k^2)$, which is $\widetilde{O}(S \cdot k \cdot n^2)$ for $k \ge (n/S)^{1/6}$.

3. Recomputing AB^* from scratch takes time $O(n \cdot |H|^2) = \widetilde{O}(n^3/k^2)$, which is $\widetilde{O}(S \cdot k \cdot n^2)$ for $k \ge (n/S)^{1/3}$.

Thus for any $k \ge (n/S)^{1/3}$ the running time of `Decrease` and `Increase` operations is dominated by updates in D, which is $\widetilde{O}(S \cdot k \cdot n^2)$ amortized. To conclude the proof, we observe that queries require $O(|H|) = \widetilde{O}\left(\frac{n}{k}\right)$ worst-case time per operation. □

Choosing $k = (n/S)^{1/3}$ yields the following bounds.

Corollary 1 *Any* `Decrease` *and* `Increase` *operation can be supported in* $\widetilde{O}(S^{2/3} \cdot n^{7/3})$ *amortized time, while any* `Query` *operation can be answered in* $\widetilde{O}(S^{1/3} \cdot n^{2/3})$ *worst-case time.*

4.2 Family $\mathcal{F}_2$

For the second family, we extend the update time of $\widetilde{O}\left(S \cdot k \cdot n^2\right)$ to the larger range $(n/S)^{1/6} \leq k \leq n$, at the price of increasing the query time to $\widetilde{O}(n^2/k^2)$. Once again, if the shortest path has at most k edges, then the query is answered correctly; otherwise it is answered correctly with probability $1 - 1/n^c$, for any constant $c > 0$.

The main difference with the algorithms in family $\mathcal{F}_1$ is that we avoid the recomputation of AB^* during the updates. Namely, we maintain matrices D and B^* exactly as in Section 4.1, while imposing more burden on queries about entries of X^*, which now have to be computed as follows:

- `Query`(x, y): return

$$\min\left\{D[x,y], \min_{h,h' \in H}\{A[x,h] + B^*[h,h'] + C[h',y]\}\right\}.$$

Theorem 5 *Let* $n^{1/6} \leq k \leq n$. *Any* `Decrease` *and* `Increase` *operation can be supported in* $\widetilde{O}\left(S \cdot k \cdot n^2\right)$ *amortized time, while any* `Query` *operation can be answered in* $\widetilde{O}\left(n^2/k^2\right)$ *worst-case time.*

Proof. Following the same lines as in the proof of Theorem 4, we can show that the running time of an update is given by the cost of updating D, which is $\widetilde{O}(S \cdot k \cdot n^2)$, plus the cost of updating B^*, which is $\widetilde{O}(S^{1/2} \cdot n^{2.5}/k^2)$. For $k \geq (n/S)^{1/6}$, the update in D is dominating, and thus the overall update bound is $\widetilde{O}(S \cdot k \cdot n^2)$. Queries require $O(|H|^2) = \widetilde{O}\left(n^2/k^2\right)$ worst-case time per operation. □

Choosing $k = (n/S)^{1/6}$ yields the following bounds.

Corollary 2 *Any* `Decrease` *and* `Increase` *operation can be supported in* $\widetilde{O}(S^{5/6} \cdot n^{13/6})$ *amortized time, while any* `Query` *operation can be answered in* $\widetilde{O}(S^{1/3} \cdot n^{5/3})$ *worst-case time.*

5 Conclusions

In this paper, we have presented a new fully dynamic shortest path algorithm for directed graphs and arbitrary real weights (with at most S different values per edge weight), which supports updates in $O(S^{0.5} \cdot n^{2.5} \log^{0.5} n)$ amortized time and distance queries in $O(1)$ worst-case time. We have also introduced two new families of algorithms that obtain effective query/update trade-offs for the same problem. Besides improving any of our bounds, there are several issues that seem worth further investigation. First, is it possible to achieve quadratic update time while maintaing costant query time? Note that constant update time and quadratic query time is trivially obtainable by just doing nothing at each update and by performing a single-source shortest path computation at each query. Second, is there any dynamic all pairs shortest path algorithm for general graphs that is able to achieve subquadratic times for both updates and queries? We remark that this is possible for the simpler problem of fully dynamic transitive closure on directed acyclic graphs [3].

References

[1] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, April 1997.

[2] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[3] C. Demetrescu and G.F. Italiano. Fully dynamic transitive closure: Breaking through the $O(n^2)$ barrier. In *Proc. of the 41st IEEE Annual Symposium on Foundations of Computer Science (FOCS'00)*, pages 381–389, 2000. Full paper available at the URL: `http://arXiv.org/abs/cs.DS/0104001`.

[4] C. Demetrescu and G.F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. In *Proc. of the 42nd IEEE Annual Symposium on Foundations of Computer Science (FOCS'01), Las Vegas, Nevada*, pages 260–267, 2001.

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

[6] 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.

[7] M. L. Fredman. New bounds on the complexity of the shortest path problems. *SIAM Journal on Computing*, pages 87–89, 1976.

[8] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their use in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.

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

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

[11] Z. Galil and O. Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134(2):103–139, 1 May 1997.

[12] Z. Galil and O. Margalit. All pairs shortest paths for graphs with small integer length edges. *Journal of Computer and System Sciences*, 54(2):243–254, April 1997.

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

[14] 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.

[15] 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.

[16] 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.

[17] V. King and M. Thorup. A space saving trick for directed dynamic transitive closure and shortest path algorithms. In *Proceedings of the 7th Annual International Computing and Combinatorics Conference (COCOON), LNCS 2108*, pages 268–277, 2001.

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

[19] 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.

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

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

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

[23] 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.

[24] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, December 1995.

[25] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science: October 17–19, 1999, New York City, New York,*, pages 605–614, 1999.

[26] T. Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Information Processing Letters*, 43(4):195–199, September 1992.

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

[28] U. Zwick. All pairs shortest paths in weighted directed graphs - exact and almost exact algorithms. In *Proc. of the 39th IEEE Annual Symposium on Foundations of Computer Science (FOCS'98)*, pages 310–319, Los Alamitos, CA, November 8–11 1998.