Fully Dynamic All Pairs Shortest Paths with Real Edge Weights *

Camil Demetrescu † Giuseppe F. Italiano ‡

Abstract

We present the first fully dynamic algorithm for maintaining all pairs shortest paths in directed graphs with real-valued edge weights. Given a dynamic directed graph G such that each edge can assume at most S different real values, we show how to support updates deterministically in $O(S \cdot n^{2.5} \log^3 n)$ amortized time and queries in optimal worst-case time. No previous fully dynamic algorithm was known for this problem. In the special case where edge weights can only be increased, we give a randomized algorithm with one-sided error which supports updates faster in $O(S \cdot n \log^3 n)$ amortized time.

1 Introduction

All Pairs Shortest Path (APSP) is perhaps one of the most fundamental 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 [24] 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 lead to a recent surge of interest in APSP. Alon, Galil and Margalit [2] were the first to show that fast matrix multiplication can be effectively used for solving APSP on directed graphs with small integer weights: they gave an algorithm whose running time is $\tilde{O}(n^{(3+\omega)/2})$[1] 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, $\omega < 2.376$ [4]. Galil and Margalit [11, 12] and Seidel [22] 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 [23] and Zwick [25] achieve the best known bound for APSP with positive integer edge weights less than C: $\tilde{O}(Cn^\omega)$ for undirected graphs [23] and $O(C^{0.681} n^{2.575})$ for directed graphs [25]. All these subcubic algorithms are based on clever reductions to fast matrix multiplication.

*This work has been partially supported by the IST Programme of the EU under contract n. IST-1999-14.186 (ALCOM-FT), by the Italian Ministry of University and Scientific Research and by CNR, the Italian National Research Council, under contract n. 00.00346.CT26.

†Dipartimento di Informatica e Sistemistica, Università di Roma “La Sapienza”, Roma, Italy. Email: demetres@dis.uniroma1.it. URL: http://www.dis.uniroma1.it/~demetres.

‡Dipartimento di Informatica, Sistemi e Produzione, Università di Roma “Tor Vergata”, Roma, Italy. Email: italiano@info.uniroma2.it. URL: http://www.info.uniroma2.it/~italiano.

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

In this paper we present 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 $\mathcal{P}$ quickly, 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:

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 n the number of vertices in G.

Previous Work. The dynamic maintenance of shortest paths has a long history, and the first papers date back to 1967 [15, 17, 20]. In 1985 Even and Gazit [6] and Rohnert [21] 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 [18, 19] 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.

There seem to be few dynamic shortest path algorithms which are provably faster than recomputing APSP from scratch, and they only work on graphs with small integer weights. In particular, Ausiello *et al.* [3] 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.* [13] 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. King [14] 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.

It seems thus quite natural to ask whether one can efficiently solve fully dynamic shortest path problems on graphs whose weights are not integers, and are not small: i.e., graphs whose edges can have arbitrary (positive and negative) real weights. We remark that, despite over 30 years of research in this area, finding an efficient solution for this problem has remained an elusive goal.

Our Results. The main contribution of this paper is 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, we give a deterministic algorithm that 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. This seems to be the case in many applications. Just to mention one, consider shortest-path routing in computer networks. One important feature of Open Shortest Path First (OSPF) [16], the most commonly used intra-domain Internet routing protocol, is that the link weights, and consequently the shortest path routes, can be changed by network operators: when the traffic is light and network conditions change slowly, the weights could be set proportional to their physical distances. In the presence of congestion, however, the standard heuristic recommended by Cisco is to make the weight of a link inversely proportional to its capacity. In this scenario, link weights are not necessarily integers and are not necessarily small, but can assume real values in a given set.

There are no previously known fully dynamic algorithms for APSP on graphs with arbitrary real weights. The fastest asymptotic static algorithm is by Takaoka [24], and thus our algorithm is faster than recomputing from scratch by a factor of $O(\sqrt{n \log \log n}\,/(S \cdot \log^{3.5} n))$. We can also support within the same time bounds the generalized updates of [14], i.e., decreasing the weight of a set of edges incident to the same vertex and increasing the weight of an arbitrary subset of edges. In the special case where edge weights can only be increased, we show how to support updates faster in $O(S \cdot n \log^3 n)$ amortized time. This algorithm is randomized with one-sided error. The only previous increase-only algorithm [14] works only if edge weights are positive integers less than C and runs in $O(C \cdot n^2)$ amortized time. Our algorithms are based on simple data structures and thus seem amenable to efficient implementations.

To achieve our bounds, we manage to extend to dynamic shortest paths the algebraic framework and the lazy evaluation techniques developed in [5] for the simpler problem of dynamic transitive closure. In particular, we show how to cast fully dynamic all pairs shortest paths into the problem of maintaining polynomials of matrices over the {min,+} semiring. The equivalence between APSP and matrix multiplication on the {min,+} semiring is well known [1]. However, exploiting this equivalence has always been a difficult task as no fast matrix multiplication methods are available on {min,+}. Indeed, many static algorithms for APSP try to shortcut this difficulty by recurring to reductions to fast integer matrix multiplication: the price paid for this is that edge weights need to be small integers. To circumvent this limitation, we develop tools for exploiting directly in the dynamic setting the equivalence between APSP and matrix multiplication on {min,+}: our techniques do not use fast matrix multiplication at all but rather hinge on the dynamic maintenance of products of real-valued matrices in this semiring as the main ingredient for solving dynamic shortest path problems.

2 Dynamic Matrices over the {min,+} Semiring

In this section we consider the problem of reevaluating polynomials of matrices over the {min,+} semiring subject to updates of their variables. Those updates might look strange at first sight: they are defined this way so that they can be useful later on for our original problem of fully dynamic all pairs shortest paths. In the following, we denote by $\mathcal{M}_n$ the set of $n \times n$ real matrices. Given two matrices $A, B \in \mathcal{M}_n$, we denote by $C = A \odot B$ the matrix product over the {min,+} semiring such that $C[x, y] = \min_{1 \leq z \leq n}\{A[x, z] + B[z, y]\}$ and we denote by $D = A \oplus B$ the matrix sum over the {min,+} semiring such that $D[x, y] = \min\{A[x, y], B[x, y]\}$. In the following, we denote by AB the product $A \odot B$ and by $AB[x, y]$ entry (x, y) of matrix AB. Moreover, if X is a matrix, we denote by $I_{X,i}$ and $J_{X,j}$ the matrices equal to X in the i-th row and in the j-th column, respectively, and $+\infty$ in any other entries.

The Problem. Without loss of generality, let

$$P = \bigoplus_{1 \leq a \leq h} T_a$$

be a polynomial with h terms defined over $\mathcal{M}_n$, where each term

$$T_a = \bigodot_{1 \leq b \leq k_a} X_b^a$$

has degree $k_a \leq k$ and variables $X_b^a \in \mathcal{M}_n$ are distinct. We consider the problem of maintaining a data structure `P` for the polynomial P under an intermixed sequence $\sigma = \langle \mathtt{P.Op}_1, \ldots, \mathtt{P.Op}_l \rangle$ of initialization, update, and query operations. Each operation $\mathtt{P.Op}_j$ on the data structure `P` can be either one of the following:

- $\mathtt{P.Init}(Z_1^1, \ldots, Z_k^h)$: perform the initialization $X_b^a \leftarrow Z_b^a$ of the variables of polynomial P, where each $Z_b^a \in \mathcal{M}_n$.
- $\mathtt{P.SetRow}(i, X, X_b^a)$: perform the row update operation $X_b^a \leftarrow X_b^a \oplus I_{X,i}$, where $X \in \mathcal{M}_n$. The operation sets the i-th row of variable X_b^a to the minimum between its current value and the i-th row of matrix X.
- $\mathtt{P.SetCol}(i, X, X_b^a)$: perform the column update operation $X_b^a \leftarrow X_b^a \oplus J_{X,i}$, where $X \in \mathcal{M}_n$. The operation sets the i-th column of variable X_b^a to the minimum between its current value and the i-th column of matrix X.
- $\mathtt{P.LazySet}(X, X_b^a)$: perform the update operation $X_b^a \leftarrow X_b^a \oplus X$, where $X \in \mathcal{M}_n$. The operation sets variable X_b^a to the minimum between its current value and matrix X.
- $\mathtt{P.Reset}(X, X_b^a)$: perform the update operation $X_b^a \leftarrow \max\{X_b^a, X\}$, where $X \in \mathcal{M}_n$.
- $\mathtt{P.Lookup}()$: answer a query about the value of P by returning a matrix $Y \in \mathcal{M}_n$ whose value at time j is defined for any x, y as follows:

$$Y_j[x,y] = \begin{cases} P_j[x,y] & \text{if } P_j[x,y] > Y_{j-1}[x,y] \lor \\ & \quad \mathtt{Op}_j = \mathtt{Init}(Z_1^1, \ldots, Z_k^h) \\ \min\{Y_{j-1}[x,y], (X_1^a \odot \cdots \odot X_b^a)[x,i] + & \text{if } \mathtt{Op}_j = \mathtt{SetCol}(i, X, X_b^a) \lor \\ \qquad (X_{b+1}^a \odot \cdots \odot X_k^a)[i,y]\} & \quad \mathtt{Op}_j = \mathtt{SetRow}(i, X, X_{b+1}^a) \\ \min\{Y_{j-1}[x,y], X_1^a[x,y]\} & \text{if } \mathtt{Op}_j = \mathtt{LazySet}(X, X_1^a) \land k_a = 1 \\ Y_{j-1}[x,y] & \text{otherwise} \end{cases}$$

  and P_i is the value of polynomial P at time i. As $Y \geq P$, answers about the value of P may be affected by one-sided error.

- $\mathtt{P.Witness}(x, y)$: return a tuple $(a, x_0, \ldots, x_{k_a})$ such that $x_0 = x$, $x_{k_a} = y$ and $X_1^a[x_0, x_1] \odot \cdots \odot X_k^a[x_{k_a-1}, x_{k_a}] \leq \mathtt{P.Lookup}()[x, y]$.

Few observations are in order at this point. First, `Reset` operations can be actually viewed as a sort of "undo" operation. Indeed, according to the definition of `Lookup`, if an entry X_b^a[u,v] of a variable was equal to ϵ_1 at time t_1, and at time $t_2 > t_1$ it is increased to $\epsilon_2 > \epsilon_1$ by a `Reset` operation, then a `Lookup` at time t_2+1 answers as if X_b^a[u,v] at time t_1 was already equal to ϵ_2. Second, `Lookup` can make errors because of `LazySet` updates. Indeed, `Lookup` always returns the correct value of P if no `LazySet` operations are performed on terms of degree $k_a > 1$. The effects of `LazySet` operations are propagated to Y by subsequent `SetRow` and `SetCol` operations: this explains the reason for the name `LazySet`. Finally, we note that it is straightforward to extend the results of this section to the case where the same variable may occur in polynomial P more than once.

Our Data Structure. We consider only the case of polynomials of degree $k \le 2$. The case $k > 2$ can be easily supported by using a technique similar to the one presented in [5] for maintaining polynomials over Boolean matrices, i.e., by considering an equivalent representation $\widehat{P}$ for P such that $\widehat{P}$ has degree ≤ 2. We maintain polynomials of degree $k \le 2$ with the following elementary data structures:

- $2h$ matrices $X_1^a, X_2^a \in \mathcal{M}_n$ that store the current value of variables.
- For each term $T_a = X_1^a \odot X_2^a$ of degree 2 and for each pair x, y, three ordered arrays $Left_{x,y}^a$, $Right_{x,y}^a$ and $Prod_{x,y}^a$. We use these arrays to store information about witnesses of products.
- For each pair x, y, an ordered array $Min_{x,y}$ that keeps track of the contribution of each term of P to the value of $P[x, y]$.

Implementation of Operations. We now sketch how the different operations are implemented. We maintain our data structure so that the following invariants hold for any a, x, y, z:

1. $Right_{x,z}^a[y] \ge X_1^a[x, z]$ and $Left_{z,y}^a[x] \ge X_2^a[z, y]$
2. $Prod_{x,y}^a[z] = Right_{x,z}^a[y] + Left_{z,y}^a[x]$
3. $Right_{x,z}^a[y] = +\infty \Leftrightarrow Left_{z,y}^a[x] = +\infty$.
4. $Min_{x,y}[a] = X_1^a[x, y]$ if $k_a = 1$ and $Min_{x,y}[a] = \min_z Prod_{x,y}^a[z]$ if $k_a = 2$.

It is easy to maintain invariants 1–4 throughout any sequence of operations. In particular, we let `Init` set up the data structure such that the invariant 1 is satisfied with equalities for all a, x, y, z. Moreover, if a `SetRow/SetCol` decreases either $X_1^a[x, z]$ or $X_2^a[z, y]$ for some x, z or z, y, then we let invariant 1 be satisfied with equalities for all y or for all x, respectively. If a `Reset` increases $X_1^a[x, z]$, and after the update $Right_{x,z}^a[y] < X_1^a[x, z]$ for some y, then we let $Right_{x,z}^a[y] = X_1^a[x, z]$. Similarly, if $X_2^a[z, y]$ gets increased, then we may have to update $Left_{z,y}^a[x]$ for some x. To support `LazySet`, we only have to adjust arrays $Min_{x,y}$ to let invariant 4 be satisfied in the case where $k_a = 1$. As far as `Lookup` is concerned, we return a matrix Y s.t. $Y[x, y] = \min_a Min_{x,y}[a]$. Supporting `Witness` operations is straightforward.

Analysis. We now discuss the running time of our operations. We say that x has *variability* D if at any time it can assume only D different values. If $x \in \{\epsilon_0, \epsilon_1, \ldots, \epsilon_D\}$, where $\epsilon_0 = +\infty > \epsilon_1 > \epsilon_2 > \ldots > \epsilon_D$, we denote by $\nu(x)$ the index q such that $x = \epsilon_q$. In the analysis, we assume that the entries of each variable of polynomial P have variability $\leq D_P$. We associate any polynomial P of degree 2 with a potential function

$$\Phi_P = \log n \cdot \sum_{a:\, k_a > 1} \sum_{x,z,y} \nu(Right^a_{x,z}[y]) + \nu(Left^a_{z,y}[x]).$$

Notice that Φ_P is maximum whenever both $Right^a_{x,z}[y]$ and $Left^a_{z,y}[x]$ take their minimum possible values, and thus $\Phi_P \leq 2hD_P \cdot n^3 \log n$. Assuming that $x + \infty = +\infty$ for any real value x and observing that the invariant 3 is maintained throughout the operations, we can refine our estimate on Φ_P for polynomials of degree 2:

Lemma 1 *At any time during a sequence of operations on a polynomial P of degree 2:*

$$\Phi_P \leq \log n \cdot \sum_{a:\, k_a > 1} \sum_{x,z,y} \nu(X^a_1[x,z] + X^a_2[z,y]).$$

Theorem 1 *Any* `SetRow`*,* `SetCol` *and* `LazySet` *operation requires $O(n^2)$ time in the worst case. Any* `Init` *costs $O(hn^3 \log n)$ time in the worst case. The cost of any* `Reset` *operation can be charged to Φ_P. Queries are answered in optimal time.*

Proof. We only discuss the case of `Reset` as the others are trivial. First note that increasing the priority of an element in $Prod$, $Left$, $Right$ and Min takes $O(\log n)$ if we use priority queues. If $X^a_1[x,z]$ is increased, then we scan in increasing order of priority all the y's such that $Right_{x,z}[y] < X^a_1[x,z]$. We stop as soon as we find an y such that $Right_{x,z}[y] \geq X^a_1[x,z]$. The case where $X^a_2[z,y]$ is increased is similar. Thus, we spend time only for elements that have to be increased and therefore affect Φ_P. □

Corollary 1 *In any intermixed sequence of operations* `SetRow`*,* `SetCol`*,* `LazySet`*, and* `Reset`*, each update costs $O(D_P \cdot n^2 \log n)$ amortized time. If only* `Reset` *operations are performed in a sequence of length $\Omega(hn^2)$, the amortized cost per operation is $O(D_P \cdot n \log n)$.*

Proof. By Lemma 1, each `SetRow/SetCol` increases Φ_P by at most $2D_P \cdot n^2 \log n$ units, whereas `LazySet` does not affect Φ_P at all. The `Reset`-only bound comes from amortizing $\Phi_P \leq 2hD_P \cdot n^3 \log n$ over $\Omega(hn^2)$ operations. □

In the general case of polynomials of degree k, each term of degree $k_a \leq k$ in P corresponds to k_a^2 terms of degree 2 in the equivalent polynomial $\widehat{P}$ discussed before, and this increases the running time of update operations by at most a factor of k^2. We remark however that in the context of this paper, h and k will be constant.

3 Maintaining All Pairs Shortest Paths

In this section we show how to cast fully dynamic all pair shortest paths into the problem of reevaluating polynomials of matrices over the {min,+} semiring considered in Section 2. Let $G = (V, X)$ be a weighted directed graph such that $V = \{1, \ldots, n\}$, $X \in \mathcal{M}_n$, and $X[u,v]$ is the weight of edge (u,v); we assume that $X[u,v] = +\infty$ if there is no such edge. The Kleene closure of X over the {min,+} semiring, defined as

$$X^* = \bigoplus_{i=0}^{+\infty} X^i$$

yields the distance matrix of G. In the following, we focus on the problem of maintaining dynamically X^* under updates of X: this is equivalent to maintaining all pairs shortest paths on G.

The Problem. Let X be an $n \times n$ real matrix over the closed semiring $\mathcal{M}_n$ with operations $\oplus$ and $\odot$ and let X^* be its Kleene closure. We consider the problem of maintaining a data structure `X` for matrix X under an intermixed sequence $\sigma = \langle \texttt{X.Op}_1, \ldots, \texttt{X.Op}_k \rangle$ of initialization, update, and query operations. Each operation $\texttt{X.Op}_j$ on data structure `X` can be either one of the following:

- $\texttt{X.Init}^*(Y)$: perform the initialization operation $X \leftarrow Y$, where $Y \in \mathcal{M}_n$.
- $\texttt{X.Set}^*(i, Y)$: perform the update $X \leftarrow X \oplus I_{Y,i} \oplus J_{Y,i}$, where $Y \in \mathcal{M}_n$ and $1 \le i \le n$. We call this kind of update an i-centered set operation on X.
- $\texttt{X.Reset}^*(Y)$: perform the update $X \leftarrow \max\{X, Y\}$, where $Y \in \mathcal{M}_n$.
- $\texttt{X.Lookup}^*(x, y)$: return the value of $X^*[x, y]$.
- $\texttt{X.Witness}^*(x, y)$: return a tuple $(x_0, \ldots, x_q)$ such that $\sum_i X[x_i, x_{i+1}] = X^*[x, y]$ and $x_0 = x$, $x_q = y$.

Note that $\texttt{Set}^*$ is allowed to decrease only one row and one column of X, whereas $\texttt{Reset}^*$ can increase any entries of X. Thus, decreasing the weight of edges incident to the same vertex and increasing the weights of an arbitrary subset of edges of G can be easily supported by means of $\texttt{Set}^*$ and $\texttt{Reset}^*$ operations on X.

Our Data Structure. As it is well known [1], the Kleene closure X^* of an $n \times n$ matrix X over a closed semiring with operations $\oplus$ and $\odot$ and unit element I can be computed from scratch by letting $Q_{k+1} = Q_k^2$ where $Q_0 = X \oplus I$. By a simple doubling argument, $Q_k[x, y]$ is the length of a shortest path $x \rightsquigarrow y$ that uses at most 2^k edges, i.e., $Q_k = \bigoplus_{0 \le i \le 2^k} X^i$. Thus, $X^* = Q_{\log n}$ if there are no negative-length cycles.

We now sketch the high-level ideas behind our approach. We maintain $\log n$ levels: level k keeps track of paths with at most 2^k edges. For each level we maintain a polynomial P_k defined on three variables L, R, C, which are updated via `LazySet`, `SetRow` and `SetCol`, respectively. The value of polynomial P_{k-1} is used to update variables of polynomial P_k. More in detail, we maintain:

- $\log n$ polynomials $P_1, \ldots, P_{\log n}$ defined as $P_k(L, R, C) = L \oplus CRL \oplus LCR$ represented with $\log n$ instances of the data structure presented in Section 2; we maintain each P_k so that, with very high probability, $\texttt{P}_k\texttt{.Lookup}()[x, y]$ is less or equal than the length of a shortest path $x \rightsquigarrow y$ that uses at most 2^k edges, i.e., $\texttt{P}_k\texttt{.Lookup}() \le Q_k$;
- a counter $Time$ of the number of $\texttt{Set}^*$ operations performed since the last $\texttt{Init}^*$;
- an array $SetTime$ such that $SetTime[u] = t$ if vertex u has been last considered in a $\texttt{Set}^*$ operation at time t.

Implementation of Operations. We denote by $R_i^\infty \in \mathcal{M}_n$ and $C_i^\infty \in \mathcal{M}_n$ the matrices with all the entries equal to $-\infty$ except for the entries in the i-th row and in the i-th column which are $+\infty$, respectively. Moreover, we denote by $M^\infty \in \mathcal{M}_n$ the matrix whose entries are all $+\infty$. We now describe how the operations are implemented.

Init*

To support $\texttt{Init}^*(Y)$ we first initialize our data structure with the empty graph and then bulk load Y via $\texttt{Set}^*$ operations:

procedure $\texttt{Init}^*(Y)$:
1. $\quad$ $Time \leftarrow 0$
2. $\quad$ **for** $u = 1$ **to** n **do** $SetTime[u] \leftarrow +\infty$
3. $\quad$ **for** $k = 1$ **to** $\log n$ **do** $\texttt{P}_k.\texttt{Init}(M^\infty, M^\infty, M^\infty)$
4. $\quad$ **for** $u = 1$ **to** n **do** $\texttt{Set}^*(u, Y)$

Set*

procedure $\texttt{Set}^*(i, Y)$:
1. $\quad$ $Time \leftarrow Time + 1$
2. $\quad$ $j \leftarrow$ random integer in $[1, n]$
3. $\quad$ $SetTime[i] \leftarrow Time$
4. $\quad$ $SetTime[j] \leftarrow Time$
5. $\quad$ $Y \leftarrow Y \oplus I$
6. $\quad$ **for** $k = 1$ **to** $\log n$ **do** {
7. $\quad\quad$ $\texttt{P}_k.\texttt{LazySet}(Y, L)$
8. $\quad\quad$ $\texttt{P}_k.\texttt{SetRow}(i, Y, R)$
9. $\quad\quad$ $\texttt{P}_k.\texttt{SetCol}(i, Y, C)$
10. $\quad\quad$ $\texttt{P}_k.\texttt{SetRow}(j, Y, R)$
11. $\quad\quad$ $\texttt{P}_k.\texttt{SetCol}(j, Y, C)$
12. $\quad\quad$ **for** $u = 1$ **to** n **do**
13. $\quad\quad\quad$ **if** $Time - SetTime[u] = \lceil (cn \log n)/2^{k-1} \rceil$ **then** {
14. $\quad\quad\quad\quad$ $\texttt{P}_k.\texttt{Reset}(R^\infty_u, R)$
15. $\quad\quad\quad\quad$ $\texttt{P}_k.\texttt{Reset}(C^\infty_u, C)$
16. $\quad\quad\quad$ }
17. $\quad\quad$ $Y \leftarrow \texttt{P}_k.\texttt{Lookup}()$
18. $\quad$ }

We say that a vertex u is *out of fashion* at level k if $Time - SetTime[u] > (cn \log n)/2^{k-1}$, and we say that it is *in fashion* otherwise. When a vertex goes out of fashion, the corresponding column of C and the corresponding row of R are cast-off and raised to $+\infty$ in lines 14–15. Thus, only rows and columns of R and C corresponding to vertices that are in fashion carry information about shortest paths, whereas rows and columns corresponding to vertices that went out of fashion do not participate in this task. As we will show later, the correctness of our approach hinges on the fact that, given any two vertices x and y, we can find with very high probability a vertex z that is in fashion on some level such that the shortest distance from x to y is given either by $C[x,z] + RL[z,y]$ or by $LC[x,z] + R[z,y]$ on that level. The reason for the terms CRL and LCR of degree 3 in polynomials P_k is because z might be more than 2^{k-1} edges far away from either endpoint of the shortest path $x \rightsquigarrow y$. We note that vertices that are in fashion play the same role as the bridging sets of [25] and as the blockers of [14]; the main difference is that vertices that are in fashion are maintained dynamically throughout each operation by a simple time sliding window, while bridging sets and blockers seem hard to maintain dynamically.

Note that $\texttt{Set}^*(i, Y)$ brings into fashion at most two vertices in lines 3–4. The first vertex is i, the center of the update: it is brought into fashion because it may contain the latest information about shortest paths that go through i. The second vertex j is chosen at random, so as to ensure that vertices that are in fashion are well distributed throughout the graph, as shown in the following lemma.

Lemma 2 *At any time during a sequence of updates, on any path that uses* $\geq 2^{k-1}$ *edges there is at least one vertex that is in fashion at level* k *with probability* $p \geq 1 - \frac{1}{n^c}$*, for any constant* $c > 0$.

Proof. At any time during a sequence of updates, a vertex is out of fashion at level k with probability $p_{out} \leq \frac{1}{n^{c/2^{k-1}}}$, for any constant $c > 0$. Indeed, a vertex is out of fashion at level k if it has not been considered in the last $\frac{cn \log n}{2^{k-1}}$ `Set*` operations. As each `Set*` operation chooses a vertex independently at random and brings it into fashion, $p_{out} \leq (1 - \frac{1}{n})^{\frac{cn \log n}{2^{k-1}}} \leq 2^{\log(n^{-c/2^{k-1}})} = \frac{1}{n^{c/2^{k-1}}}$. Thus, the probability that all vertices on a path that uses $\geq 2^{k-1}$ edges are out of fashion is $\leq (p_{out})^{2^{k-1}} \leq \left(\frac{1}{n^{c/2^{k-1}}}\right)^{2^{k-1}} = \frac{1}{n^c}$. □

`Reset*`

Increases of X specified by Y are carried out by propagating `Reset` operations through the levels:

```
procedure Reset*(Y):
    Y ← Y ⊕ I
    for k = 1 to log n do {
        P_k.Reset(Y, L)
        P_k.Reset(Y, R)
        P_k.Reset(Y, C)
        Y ← P_k.Lookup()
    }
```

`Lookup*` and `Witness*`

A query `Lookup*`(x, y) can be answered by returning $\mathtt{P}_{\log n}$.`Lookup()`$[x, y]$. `Witness*` can be easily supported by performing `Witness` queries on the polynomials P_k.

Analysis of Correctness. We prove that, with very high probability, $\mathtt{P}_k$.`Lookup()`$[x, y]$ is less than or equal to the length of a shortest path $x \rightsquigarrow y$ that uses at most 2^k edges.

Theorem 2 *At any time* t *during a sequence of updates,* $X^* \leq \mathtt{P}_k.\mathtt{Lookup}() \leq Q_k$ *with probability* $p \geq 1 - \frac{1}{n^c}$*, for any constant* $c > 0$.

Proof. We proceed by induction on the number of levels. The base is trivial. Assume by induction that the claim holds for $\mathtt{P}_{k-1}$. Let $\pi_{x,y}$ be any path $x \rightsquigarrow y$ of length $\ell(\pi_{x,y})$ that uses q edges, $2^{k-1} < q \leq 2^k$. By Lemma 2, with probability $p \geq 1 - \frac{1}{n^c}$ there is at least one vertex on $\pi_{x,y}$ that is in fashion at level k. Let z be the most recent among them, with $t_z = SetTime[z]$. Then, at time t_z, operations $\mathtt{P}_k$.`SetRow`(z, Y, R), $\mathtt{P}_k$.`SetCol`(z, Y, C) and $\mathtt{P}_k$.`LazySet`(Y, L) synchronized L, column z of C, and row z of R with Y, where $Y = \mathtt{P}_{k-1}$.`Lookup()` at that time. By the inductive hypothesis, this implies that, at time t_z, $C[x, z] \leq Q_{k-1}[x, z]$, $LC[x, z] \leq Q^2_{k-1}[x, z]$, $R[z, y] \leq Q_{k-1}[z, y]$, and $RL[z, y] \leq Q^2_{k-1}[z, y]$. By the definition of `Lookup` given in Section 2 and by noting that $\ell(\pi_{x,y}) \geq \min\{Q_{k-1}[x, z] + Q^2_{k-1}[z, y],\ Q^2_{k-1}[x, z] + Q_{k-1}[z, y]\}$, we have that at time t_z

$$\mathtt{P}_k.\mathtt{Lookup}()[x, y] \leq \min\{C[x, z] + RL[z, y],\ LC[x, z] + R[z, y]\} \leq$$
$$\min\{Q_{k-1}[x, z] + Q^2_{k-1}[z, y],\ Q^2_{k-1}[x, z] + Q_{k-1}[z, y]\} \leq \ell(\pi_{x,y}).$$

Since a vertex z is brought into fashion not only if it is chosen randomly, but also if a z-centered set operation that decreases the weights of edges incident to z is performed, and since z is the most recent vertex on $\pi_{x,y}$, $\ell(\pi_{x,y})$ did not decrease since time t_z. As edge increases are handled via $\mathtt{P}_k$.`Reset` operations on P_k which "undo" the effects of previous set operations, this implies that the above inequalities were valid at time t_z and are still valid at time t. Moreover, since $\mathtt{P}_k$.`LazySet`(Y, L) always updates P_k by considering the contribution of the term L of degree 1, it follows that $\mathtt{P}_k$.`Lookup`$() \leq \mathtt{P}_{k-1}$.`Lookup`$() \leq Q_{k-1}$. Thus, if $\pi^*_{x,y}$ is a path that uses at most 2^k edges and $\ell(\pi^*_{x,y}) = Q_k[x,y]$ at time t, it holds that $\mathtt{P}_k$.`Lookup`$()[x,y] \leq \ell(\pi^*_{x,y}) = Q_k[x,y]$. □

As $Q_{\log n} = X^*$, Theorem 2 implies that $\mathtt{P}_{\log n}$.`Lookup`$() = X^*$ with very high probability, which proves the correctness of our algorithm.

Analysis of Running Time.

Lemma 3 *At any time during a sequence of updates, there are at most $\frac{cn \log n}{2^{k-1}}$ vertices that are in fashion at level k for any constant $c > 0$.*

Proof. The proof easily follows from observing that a vertex u goes out of fashion after $\frac{cn \log n}{2^{k-1}}$ `Set`* operations that do not update $SetTime[u]$. □

Theorem 3 *In any intermixed sequence of operations* `Set`* *and* `Reset`* *on a matrix X whose entries can assume at most S different real values, each update costs $O(S \cdot n^{2.5} \log^3 n)$ amortized time.*

Proof (Sketch). We use a potential function $\Phi = \sum_{k=1}^{\log n} \Phi_{P_k}$, where Φ_{P_k} is defined on polynomial P_k as in Section 2. `Reset`* is implemented via `Reset` operations on the polynomials: by Theorem 1, the cost of each such `Reset` can be charged to Φ_{P_k}. It thus suffices to bound the increases in Φ_{P_k} caused by `Set`* operations.

Any `Set`* operation decreases at most two rows and two columns of R and C, and increases to $+\infty$ the rows and columns corresponding to nodes that are going out of fashion. Consider a vertex z that either goes out of fashion or is brought into fashion by a `Set`* operation. Since the variability D_k of variables C, R, L at level k is $D_k \leq 2^{k-1} \cdot S$, then for any x and y the change in $\nu(C[x,z] + RL[z,y])$ and $\nu(LC[x,z] + R[z,y])$ is $O(2^{k-1} \cdot S)$. Hence, by Lemma 1, $\Delta\Phi_{P_k} \leq O(2^{k-1} \cdot S \cdot n^2 \log n)$. It is possible to bound the increase in $\Delta\Phi_{P_k}$ also as $O(\frac{cn \log n}{2^{k-1}} \cdot S \cdot n^2 \log n)$. Thus, the increase in Φ_{P_k} during any `Set`* operation is bounded by $O(S \cdot n^2 \log n \cdot \min\{2^{k-1}, \frac{cn \log n}{2^{k-1}}\}) = O(S \cdot n^{2.5} \log^2 n)$. □

The bounds of Theorem 3 can be made deterministic. Details will be given in the full paper.

Theorem 4 *If only* `Reset`* *operations are performed in a sequence of length $\Omega(n^2)$, the amortized cost per update is $O(S \cdot n \log^3 n)$.*

Proof. By Lemma 3, on variables C and R at level k there are at most $\frac{cn \log n}{2^{k-1}}$ columns and rows whose entries are not all $+\infty$. Since P_k can be maintained by means of a constant number of polynomials of degree 2, and the variability D_k of variables C, R, L at level k is $D_k \leq 2^{k-1} \cdot S$, Lemma 1 gives $\Phi_{P_k} \leq O(D_k \frac{cn^3 \log^2 n}{2^{k-1}}) = O(S \cdot n^3 \log^2 n)$. By Theorem 1, maintaining $\log n$ polynomials and amortizing over $\Omega(n^2)$ operations yields an $O(S \cdot n \log^3 n)$ update time. □

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison Wesley, 1974.

[2] 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.

[3] 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.

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

[5] 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`.

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

[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] 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.

[14] 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.

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

[16] J. Moy. *OSPF: Anatomy of an Internet Routing Protocol*. Addison-Wesley, 1999.

[17] 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.

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

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

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

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.