All Pairs Shortest Paths in Undirected Graphs with Integer Weights

Avi Shoshan * *Uri Zwick* *

Abstract

We show that the All Pairs Shortest Paths *(APSP) problem for* undirected *graphs with* integer *edge weights taken from the range* $\{1, 2, \ldots, M\}$ *can be solved using only a logarithmic number of* distance products *of matrices with elements in the range* $\{1, 2, \ldots, M\}$. *As a result, we get an algorithm for the APSP problem in such graphs that runs in* $\tilde{O}(Mn^\omega)$ *time, where* n *is the number of vertices in the input graph,* M *is the largest edge weight in the graph, and* $\omega < 2.376$ *is the exponent of matrix multiplication. This improves, and also simplifies, an* $\tilde{O}(M^{(\omega+1)/2}n^\omega)$ *time algorithm of Galil and Margalit.*

1. Introduction

The *All Pairs Shortest Paths* (APSP) problem is one of the most fundamental algorithmic graph problems. The APSP problem for directed or undirected graphs with real weights can be solved using classical methods, in $O(mn + n^2 \log n)$ time (Dijkstra [4], Johnson [10], Fredman and Tarjan [7]), or in $O(n^3((\log\log n)/\log n)^{1/2})$ time (Fredman [6], Takaoka [12]). Thorup [14],[15] recently showed that for undirected graphs with integer weights, the problem can actually be solved in $O(mn)$ time. In the last decade, it was shown that fast algorithms for algebraic matrix multiplication can be used to obtain faster algorithms for solving the APSP problem in dense graphs with small integer edge weights. Galil and Margalit [8],[9], and Seidel [11], obtained $\tilde{O}(n^\omega)$ time algorithms for solving the APSP problem for unweighted *undirected* graphs. Seidel's algorithm is simpler. The algorithm of Galil and Margalit can be extended to handle small integer weights. The currently best upper bound on ω, the exponent of matrix multiplication, is $\omega < 2.376$, obtained by Coppersmith and Winograd [3].

*Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail addresses: avi@compugen.co.il and zwick@math.tau.ac.il. Work supported in part by **the basic research foundation** administrated by the **Israel academy of sciences and humanities**.

The algorithm of Seidel computes products of integer matrices. The algorithm of Galil and Margalit uses only Boolean products. Their algorithm shows that the APSP problem for unweighted undirected graphs can be solved using a logarithmic number of Boolean matrix products.

It is not difficult to see that solving the APSP problem, even for unweighted undirected graphs, is at least as hard as Boolean matrix multiplication (see, e.g., [5]). Thus, the running times of the algorithms of Galil and Margalit, and of Seidel, for unweighted undirected graphs are essentially optimal.

What happens with *weighted* undirected graphs? There seems to be no obvious way of using Seidel's elegant technique on weighted graphs. The algorithm of Galil and Margalit [8],[9], on the other hand, can be used to solve the APSP problem for undirected graphs with edge weights in the range $\{1, 2, \ldots, M\}$ in $\tilde{O}(M^{(\omega+1)/2} \cdot n^\omega)$ time. Their algorithm is quite complicated, however. It works by finding, in an ingenious way, the individual *trits*, i.e., the base 3 digits, of each distance in the graph. As in the unweighted case, the algorithm of Galil and Margalit uses only Boolean matrix multiplications. More specifically, their algorithm uses $\tilde{O}(M^{(\omega+1)/2} \log n)$ Boolean products of $n \times n$ matrices, where n is the number of vertices in the graph.

In this paper, we simplify and improve the algorithm of Galil and Margalit [8],[9] and obtain an algorithm for the APSP problem whose running time is $\tilde{O}(Mn^\omega)$. There are two main differences between our algorithm and the algorithm of Galil and Margalit. The first is that we compute the *bits* of the individual distances, not the trits. This simplifies the algorithm quite a bit. The second, and more major difference, is that instead of the 'low level' Boolean matrix multiplications, we use *distance products* of matrices. (Distance products are also known as min/plus products.)

To solve the APSP problem for undirected graphs with weights from the range $\{1, 2, \ldots, M\}$, our new algorithm uses only $O(\log(Mn))$ distance products of $n \times n$ matrices with elements in the range $\{1, 2, \ldots, M\}$. As each such distance product can be computed in $\tilde{O}(Mn^\omega)$ time, where ω is the exponent of matrix multiplication (see [1], [13] or [16]), the whole algorithm runs in $\tilde{O}(Mn^\omega)$ time, as stated.

If $A = (a_{ij})$ and $B = (b_{ij})$ are two $n \times n$ matrices, then their distance product $C = A \star B$ is an $n \times n$ matrix $C = (c_{ij})$ such that $c_{ij} = \min_{k=1}^{n} \{a_{ik} + b_{kj}\}$, for $1 \le i, j \le n$.

It is easy to see that distance products are intimately related to the computation of distances. A weighted graph $G = (V, E)$ on n vertices can be encoded as an $n \times n$ matrix $D = (d_{ij})$ in which d_{ij} is the weight of the edge (i, j), if there is such an edge in the graph, and $d_{ij} = +\infty$, otherwise. We also let $d_{ii} = 0$, for $1 \le i \le n$. It is easy to see that D^n, the n-th power of D with respect to distance products, is a matrix that contains the distances between all pairs of vertices in the graph (assuming there are no negative cycles). The matrix D^n can be easily computed using $\lceil \log_2 n \rceil$ distance products. This is done by repeatedly squaring the matrix D. Note, however, that even if the finite elements of D are taken from the small range $\{1, 2, \ldots, M\}$, the elements of the matrices D^{2^i} may lie in a much larger range. The last distance product, for example, may involve elements as high as $nM/2$. Computing distance products of matrices with such large elements is too costly.

The main contribution of this paper is the demonstration that the APSP problem for *undirected* graphs with *integer* edge weights can be solved using only a logarithmic number of distance products of matrices whose elements are taken from the *same range* as the original edge weights. It is easy to see that the APSP problem is at least as hard as the computation of a single distance product. This result, therefore, is optimal, up to a logarithmic factor. All efforts for getting improved algorithms for the APSP problem for undirected graphs with integer weights can now be devoted to obtaining improved algorithms for the computation of distance products.

The range preserving reduction from the undirected APSP problem to distance products uses, in an essential way, the fact the graphs are *undirected*. In addition to the *triangle inequality*, which states that for every three vertices x, y and z we have $\delta(x,y) \le \delta(x,z) + \delta(z,y)$, and holds also in directed graphs, we also use the *inverse triangle inequality*, which states that for every three vertices x, y and z in an undirected graph we have $\delta(y,z) - \delta(x,z) \le \delta(x,y)$. This inequality holds in undirected graphs, as $\delta(x,y) = \delta(y,x)$, but does not hold, in general, in directed graphs. (Here $\delta(x,y)$ denotes the distance from x to y in the graph.)

No range preserving reduction is known from the *directed* APSP problem to distance products. Improving results of Alon, Galil and Margalit [1] and Takaoka [13], the second author [16] (see also [17]) obtained recently an $O(M^{0.681} n^{2.575})$ algorithm for solving the APSP problem in directed graphs with edge weights taken from the range $\{-M, \ldots, 0, \ldots, M\}$. (The constants 0.681 and 2.575 are derived from the currently best exponents of *rectangular matrix multiplications*.) There is quite a large gap, therefore, between the complexities of the currently best algorithms for the undirected APSP and the directed APSP problems in graphs with integer edge weights. It is interesting to note, however, that the running time of both the $\tilde{O}(Mn^{\omega})$ time algorithm for undirected graphs presented here, and the $O(M^{0.681} n^{2.575})$ time algorithm for directed graphs presented in [16] are sub-cubic for $M < n^{3-\omega}$, though for every $M < n^{3-\omega}$ the algorithm for undirected graphs is faster.

The $\tilde{O}(Mn^{\omega})$ time algorithm presented here finds the *exact* distances in the graph. An $\tilde{O}(n^{\omega} \log M)$ time algorithm for finding approximate distances with a stretch of at most $1+\epsilon$, for any fixed $\epsilon > 0$, is given in Zwick [16]. This algorithm works even on directed graphs. An $\tilde{O}(mn^{1/2})$ time algorithm for finding approximate distances between all pairs of vertices with an *additive* error of at most $2M$, where m is the number of edges in the graph, and an $\tilde{O}(n^2)$ time algorithm for finding all the distances with an additive error of at most $O(M \log n)$ are given by Cohen and Zwick [2]. These two algorithms do not use fast matrix multiplication and work again only on undirected graphs.

The rest of this paper is organized as follows. In the next section we present a simple $\tilde{O}(n^{\omega})$ time algorithm, called **USP**, for finding all the distances in unweighted undirected graphs. The algorithm is a simplification of the algorithm given for the unweighted case by Galil and Margalit [8],[9]. The algorithm computes the distances by computing their individual bits. The algorithm uses only $O(\log n)$ Boolean matrix products. The resulting algorithm is quite simple. In Section 3, we then present an algorithm, called **WUSP**, for finding all the distances in weighted undirected graphs. The algorithm is obtained from algorithm **USP** by replacing the Boolean products by distance products. The algorithms described in Sections 2 and 3 find distances. In Section 4 we show that once the distances in a graph were found, a concise description of shortest paths between all pairs of vertices in the graph can easily be found. We then end, in Section 5 with some concluding remarks and open problems.

2. Unweighted undirected graphs

An algorithm, called **USP**, for solving the APSP problem for unweighted undirected graphs is given in Figure 1. The algorithm receives an $n \times n$ matrix A which is the *adjacency matrix* of the input graph $G = (V, E)$, i.e., $a_{ij} = 1$, if $(i, j) \in E$, and $a_{ij} = 0$, otherwise. (We assume that $V = \{1, 2, \ldots, n\}$.) It returns an $n \times n$ matrix Δ, such that $\Delta_{ij} = \delta(i, j)$ is the distance from i to j in the graph. We assume, without loss of generality, that the graph $G = (V, E)$ is connected so that for every $1 \le i, j, \le n$, we have $0 \le \delta(i, j) < n$.

If A and B are two $n \times n$ Boolean matrices, we let $A \vee B$ and $A \wedge B$ be the matrices obtained by computing the element-wise *or*, or the element-wise *and* of the matrices A and B. We also let $\neg A$ denote the matrix obtained by complementing all the elements of A. Finally, we let $A \cdot B$ denote the Boolean product of the two matrices A and B, i.e., $(A \cdot B)_{ij} = \bigvee_k a_{ik} \wedge b_{kj}$.

Algorithm **USP** finds the individual bits in the binary representation of all the distances in the graph. It starts by finding the most significant bits and then proceeds to find the less significant ones. As each distance in the graph is in the range $\{0, 1, \ldots, n-1\}$, each distance can be represented using about $\ell = \lceil \log n \rceil$ bits. The algorithm is therefore composed of about $\ell = \lceil \log n \rceil$ steps. In the first step, the algorithm finds the most significant bits of the distances. This is quite easy to do as the most significant bit in the distance $\delta(i, j)$ is 1 if and only if $\delta(i, j) \geq 2^{\lceil \log_2 n \rceil - 1}$. The most significant bits are thus easily extracted from the matrix $A_{\ell-1} = A^{2^{\ell-1}-1}$ that can be easily found using about $\log n$ Boolean matrix multiplications. In general, the algorithm maintains a sequence of matrices A_k and B_k such that $A_k = A^{2^k - 1}$ and $B_k = A^{2^k}$, for $0 \leq k \leq \ell$, where A is the adjacency matrix of the graph. Computing the other bits in the binary representation of the distances is slightly more difficult. The way this is done is explained in the rest of this Section.

Before explaining the operation of algorithm **USP** in more detail and proving its correctness, we introduce some notation and prove two basic lemmas.

Let $X \subseteq \{0, 1, \ldots, n\}$ be a set of possible distances. We let δ_X denote the $n \times n$ matrix such that

$$(\delta_X)_{ij} = \begin{cases} 1 & \text{if } \delta(i,j) \in X \\ 0 & \text{otherwise} \end{cases} .$$

We also let $[\delta < a]$, $[\delta \leq a]$, denote the matrices $\delta_{[0,a)}$ and $\delta_{[0,a]}$. More generally, we let $[cond(\delta)]$ denote the matrix $\delta_{\{x | cond(x)\}}$. We also let $[a, b] = \{a, a+1, \ldots, b\}$ and $[a, b) = \{a, a+1, \ldots, b-1\}$. The first lemma we need is the triangle inequality for undirected graphs.

Lemma 2.1 (Triangle inequality) *For every three vertices* $i, j, k \in V$ *in an* undirected *graph we have:*

$$|\delta(i,k) - \delta(k,j)| \;\leq\; \delta(i,j) \;\leq\; \delta(i,k) + \delta(k,j) \;.$$

Proof: The inequality $\delta(i,j) \leq \delta(i,k) + \delta(k,j)$ holds as the concatenation of any path from i to k with any path from k to j gives a path from i to j. To get the other inequality, we start with $\delta(i,k) \leq \delta(i,j) + \delta(j,k)$. As the graph is undirected, $\delta(j,k) = \delta(k,j)$ and thus $\delta(i,k) - \delta(k,j) \leq \delta(i,j)$. In the same way we get $\delta(k,j) - \delta(i,k) \leq \delta(i,j)$ which gives the result. $\square$

If X and Y are two sets of distances, we let

$$\begin{aligned} X + Y &= \{x + y \mid x \in X, y \in Y\} \;, \\ X \circ Y &= \bigcup_{x \in X, y \in Y} [|x - y|, x + y] \;. \end{aligned}$$

Lemma 2.2 *Let X and Y be sets of distances. Then, with respect to any unweighted undirected graph,*

$$\delta_X \vee \delta_Y \;=\; \delta_{X \cup Y} \quad , \quad \delta_X \wedge \delta_Y \;=\; \delta_{X \cap Y} \quad ,$$

$$\neg\delta_X \;=\; \delta_{X^c} \quad , \delta_{X+Y} \;\leq\; \delta_X \cdot \delta_Y \;\leq\; \delta_{X \circ Y} \quad .$$

Proof: The first three equalities are obvious. We prove the last relation. We begin by showing that $\delta_{X+Y} \leq \delta_X \cdot \delta_Y$. Suppose that $(\delta_{X+Y})_{ij} = 1$, for some $1 \leq i, j \leq n$. This implies that $\delta(i,j) = x + y$ for some $x \in X$ and $y \in Y$. Consider a shortest path p from i to j. Let k be the vertex on p whose distance from i is x. It follows that $\delta(i,k) = x$ and $\delta(k,j) = y$. Thus $(\delta_X)_{ik} = 1$ and $(\delta_Y)_{kj} = 1$, and it follows that $(\delta_X \cdot \delta_Y)_{ij} = 1$, as required.

We next show that $\delta_X \cdot \delta_Y \leq \delta_{X \circ Y}$. Suppose that $(\delta_X \cdot \delta_Y)_{ij} = 1$, for some $1 \leq i, j \leq n$. This implies that there exists $1 \leq k \leq n$ such that $\delta(i,k) = x \in X$ and $\delta(k,j) = y \in Y$. By the triangle inequality, it follows that $|x - y| \leq \delta(i,j) \leq x + y$. By the definition of $X \circ Y$, we get that $\delta(i,j) \in X \circ Y$ and therefore $(\delta_{X \circ Y})_{ij} = 1$, as required. $\square$

Algorithm **USP** is composed of three stages. The operations performed in each one of these stages are described in the following three lemmas.

Lemma 2.3 *After stage 1 of algorithm* **USP**, *for every* $0 \leq k \leq \ell = \lceil \log_2 n \rceil$, *we have*

$$\begin{aligned} A_k &= [\delta < 2^k] \;, \\ B_k &= [\delta \leq 2^k] \;. \end{aligned}$$

Proof: The lemma follows easily by induction using Lemma 2.2 and the following easily verified relations:

$$\begin{aligned} [0, 2^k) + [0, 2^k] &= [0, 2^k) \circ [0, 2^k] &= [0, 2^{k+1}) \;, \\ [0, 2^k] + [0, 2^k] &= [0, 2^k] \circ [0, 2^k] &= [0, 2^{k+1}] \;. \end{aligned}$$

$\square$

Lemma 2.4 *After stage 2 of algorithm* **USP**, *for every* $1 \leq k \leq \ell = \lceil \log_2 n \rceil$, *we have*

$$\begin{aligned} C_k &= [0 \leq (\delta \bmod 2^{k+1}) < 2^k] \;, \\ D_k &= [0 \leq (\delta \bmod 2^{k+1}) \leq 2^k] \;, \\ P_k &= [(\delta \bmod 2^{k+1}) = 0] \;, \\ Q_k &= [(\delta \bmod 2^{k+1}) = 2^k] \;. \end{aligned}$$

```
algorithm USP(A)

1. ℓ ← ⌈log₂ n⌉
   A₀ ← I
   B₀ ← A ∨ I
   for k ← 1 to ℓ do
       A_k ← A_{k−1} · B_{k−1}
       B_k ← B_{k−1} · B_{k−1}
   end

2. C_ℓ ← 1 ; D_ℓ ← 1
   P_ℓ ← I ; Q_ℓ ← 0
   for k ← ℓ − 1 downto 0
       C_k ← [(P_{k+1} · A_k) ∧ C_{k+1}] ∨ [(Q_{k+1} · A_k) ∧ ¬C_{k+1}]
       D_k ← [(P_{k+1} · B_k) ∧ C_{k+1}] ∨ [(Q_{k+1} · B_k) ∧ ¬C_{k+1}]
       P_k ← P_{k+1} ∨ Q_{k+1}
       Q_k ← D_k ∧ ¬C_k
   end

3. Δ ← Σ_{k=0}^{ℓ} 2^k (¬C_k)
```

Figure 1. Finding all the distances in an unweighted undirected graph.

Proof: We prove the Lemma by induction on k. It is easy to verify that the claim holds for $k = \ell$. We now show that if the claim holds for $k+1$, where $0 \leq k < \ell$, then it also holds for k. Using the relations of Lemma 2.2 it is not difficult to see that

$$[0 \leq (\delta \bmod 2^{k+2}) < 2^k] \;\leq\; P_{k+1} \cdot A_k \leq [-2^k < (\delta \bmod 2^{k+2}) < 2^k] \;,$$

$$[2^{k+1} \leq (\delta \bmod 2^{k+2}) < 2^{k+1} + 2^k] \;\leq\; Q_{k+1} \cdot A_k \leq [2^{k+1} - 2^k < (\delta \bmod 2^{k+2}) < 2^{k+1} + 2^k] \;,$$

where in the first relation we assume that for every integer x, the function $(x \bmod 2^r)$ returns values in the range $(-2^{r-1}, 2^{r-1}]$, while in the second relation we assume that it returns values in the range $[0, 2^r)$. By the induction hypothesis, $C_{k+1} = [0 \leq (\delta \bmod 2^{k+2}) < 2^{k+1}]$. As a consequence we get that

$$(P_{k+1} \cdot A_k) \wedge C_{k+1} = [0 \leq (\delta \bmod 2^{k+2}) < 2^k] \;,$$
$$(Q_{k+1} \cdot A_k) \wedge \neg C_{k+1} = [2^{k+1} \leq (\delta \bmod 2^{k+2}) < 2^{k+1} + 2^k] \;.$$

Thus,

$$C_k = [(P_{k+1} \cdot A_k) \wedge C_{k+1}] \vee [(Q_{k+1} \cdot A_k) \wedge \neg C_{k+1}] = [0 \leq (\delta \bmod 2^{k+1}) < 2^k] \;,$$

as required. The proof that D_k is assigned the correct value is almost identical. All the sharp inequalities above simply have to be replaced by weak inequalities. Finally, it is easy to see that P_k and Q_k are also assigned the correct values. This completes the proof. □

It is not difficult to see that $\neg C_k$ gives the k-th bit, counting from the right, in the binary representation of the distances in the graph. Finally, in the third stage, the algorithm packs the individual bits of the distances and obtains the matrix $\Delta = \delta$. As a consequence we get:

Theorem 2.5 *Algorithm* **USP** *finds all the distances in an unweighted undirected graph using $O(\log n)$ Boolean matrix multiplications and its total running time is therefore $\tilde{O}(n^\omega)$, where $\omega < 2.376$ is the exponent of matrix multiplication.*

3. Weighted undirected graphs

In this section we present an algorithm for finding all the distances in undirected graphs with integer edge weights. The algorithm, called **WUSP**, is given in Figure 2. The algorithm receives an $n \times n$ matrix D containing the weights of the edges in the input graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$. Thus, d_{ij} is the weight of the edge (i, j), if $(i, j) \in E$, and $d_{ij} = +\infty$, otherwise. We also assume that $d_{ii} = 0$, for $1 \leq i \leq n$. We assume that all the edge weights are taken from the range $\{1, 2, \ldots, M\}$. (We can easily allow edges of weight 0. We simply have to contract these edges. We cannot allow edges of negative weight. As the graph is undirected, a negative edge would immediately create a negative cycle.)

Algorithm **WUSP** is an extension of algorithm **USP**. The Boolean matrix products used in **WUSP** are replaced in **WUSP** by *distance products*. Recall that if A and B are two $n \times n$ matrices, we let $A \star B$ be an $n \times n$ matrix such that $(A \star B)_{ij} = \min_{k=1}^{n} \{a_{ik} + b_{kj}\}$, for $1 \leq i, j \leq n$. As shown in [1], the distance product of two $n \times n$ matrices with elements taken from the set $\{1, 2, \ldots, M, +\infty\}$ can be computed in $\tilde{O}(Mn^{\omega})$ time.

The distances in the input graph $G = (V, E)$ are all in the range $[0, (n-1)M]$. Algorithm **WUSP** computes the $\lceil \log_2 n \rceil$ most significant bits of each one of the distances, as well as the remainder that each distance leaves modulo M. Clearly, this gives enough information to reconstruct the distances. For simplicity we assume that M is a power of 2, i.e., $M = 2^m$, for some $m \geq 1$.

The distance product of two matrices whose finite elements are taken from $[1, M]$ is a matrix whose finite elements are from $[1, 2M]$. As we do not want the range of the elements in our matrices to increase, we define the following two operations. If A is an $n \times n$ matrix and a, b are two numbers, we let *clip*(A, a, b) and *chop*(A, a, b) be the two $n \times n$ matrices such that

$$\mathit{clip}(A, a, b)_{ij} = \begin{cases} a & \text{if } a_{ij} < a \\ a_{ij} & \text{if } a \leq a_{ij} \leq b \\ +\infty & \text{if } a_{ij} > b \end{cases}$$

$$\mathit{chop}(A, a, b)_{ij} = \begin{cases} a_{ij} & \text{if } a \leq a_{ij} \leq b \\ +\infty & \text{otherwise} \end{cases} .$$

In addition, if A and B are two $n \times n$ matrices, we let $A \bigwedge B$, $A \bar{\bigwedge} B$ and $A \bigvee B$ be the three $n \times n$ matrices such that

$$(A \bigwedge B)_{ij} = \begin{cases} a_{ij} & \text{if } b_{ij} < 0 \\ +\infty & \text{otherwise} \end{cases}$$

$$(A \bar{\bigwedge} B)_{ij} = \begin{cases} a_{ij} & \text{if } b_{ij} \geq 0 \\ +\infty & \text{otherwise} \end{cases}$$

$$(A \bigvee B)_{ij} = \begin{cases} a_{ij} & \text{if } a_{ij} \neq +\infty \\ b_{ij} & \text{if } a_{ij} = +\infty, b_{ij} \neq +\infty \\ +\infty & \text{if } a_{ij} = b_{ij} = +\infty \end{cases} .$$

The operations $A \bigwedge B$, $A \bar{\bigwedge} B$ and $A \bigvee B$, for numerical matrices A and B, are "analogs" of the Boolean operations $A \wedge B$, $A \wedge \neg B$ and $A \vee B$ for Boolean matrices A and B. They are used in stage 3 of **WUSP**.

Finally, if $C = (c_{ij})$ is a matrix, we let $(C \geq 0)$ be the Boolean matrix such that $(C \geq 0)_{ij} = 1$, if $c_{ij} \geq 0$, and $(C \geq 0)_{ij} = 0$, otherwise. The Boolean matrix $(0 \leq P \leq M)$ is defined in a similar way. This notation is used in stage 4 of **WUSP**.

With these definitions, all the operations performed by algorithm **WUSP** are well defined, though we are still far from explaining what the algorithm is actually doing and why it is correct. To that end we need a few more definitions.

Let $X, Y \subseteq [0, Mn]$ be two sets of possible distances. We define

$$\begin{aligned} X + Y &= \{x + y \mid x \in X, y \in Y\} , \\ X \bullet Y &= \bigcup_{x \in X, y \in Y} [x - y - 2M, x + y] . \end{aligned}$$

Note that $X + Y$ is defined exactly as in the unweighted case, while the definition of $X \bullet Y$ is similar, but not identical to the definition of $X \circ Y$.

In the previous section, we defined for every set $X \subseteq [0, n]$ a Boolean matrix δ_X. Here we define for every set $X \subseteq [0, Mn]$ that satisfies certain conditions a matrix Δ_X whose elements are in the range $[-M, M] \cup \{+\infty\}$.

Let $X = \cup_{r=1}^{p} [a_r, b_r]$, where $a_r \leq b_r$, for $1 \leq r \leq p$ and $b_r < a_{r+1}$, for $1 \leq r < p$. We define *sep*(X), the *separation* of X, to be $\min_{r=2}^{p} (a_r - b_{r-1})$. We let Δ_X be an $n \times n$ matrix such that for every $1 \leq i, j \leq n$ we have

$$(\Delta_X)_{ij} = \begin{cases} -M & \text{if } a_r \leq \delta(i,j) \leq b_r - M \\ & \text{for some } 1 \leq r \leq p \\ & \text{(the clipped zone)} \\ \delta(i,j) - b_r & \text{if } b_r - M < \delta(i,j) \leq b_r + M \\ & \text{for some } 1 \leq r \leq p \\ & \text{(the exact zone)} \\ +\infty & \mathit{otherwise} \text{ (the infinity zone)} \end{cases}$$

Note that Δ_X is well defined only if for every $1 < r \leq p$ we have $b_{r-1} + M < \min\{a_r, b_r - M + 1\}$. In particular, we must have *sep*$(X) > M$. We also let $X^* = \cup_{r=1}^{p} [\min\{a_r, b_r - M + 1\}, b_r + M]$. Note that $(\Delta_X)_{ij} < +\infty$ if and only if $\delta(i,j) \in X^*$.

Algorithm **WUSP** tries to 'imitate' algorithm **USP**, using distance products instead of Boolean products. The elements in the matrices whose distance product is computed will always taken from a small enough range. The A_k and B_k matrices of **USP** are replaced by the A_k matrices of **WUSP**which represent distances at the range $[0, 2^{k+m} + M]$, instead of distances in the range $[0, 2^k]$, as in **USP**. More precisely, $(A_k)_{ij} = -M$ if $\delta(i,j) < 2^{k+m} - M$, $(A_k)_{ij} = \delta(i,j) - 2^{k+m}$, if $2^{k+m} - M \leq \delta(i,j) < 2^{k+m} + M$, and $(A_k)_{ij} = +\infty$, otherwise. Note that all the finite elements of A_k are in the range $\{-M, \ldots, 0, \ldots, M\}$. The matrices A_k are computed in stages 1 and 2 of algorithm **WUSP**. To ensure that the elements of A_k are never out of the allowed range, we use the *clip*() operation. The matrices C_k and D_k of **USP** are replaced by the matrices C_k of **WUSP**. The matrices P_k and Q_k of **WUSP** are analogs of the matrices P_k and Q_k of **USP**. The exact definition of the matrices C_k, P_k and Q_k appears in Lemma 3.5.

The full version of the triangle inequality (Lemma 2.1) continues to hold in the weighted case. We also need the following "window" lemma. Let p be a path from i to j. If x and y are two vertices on p, we let $\delta_p(x, y)$ be the distance between x and y on p. We now claim:

Lemma 3.1 ("Window" Lemma) *Let p be a path between i and j. For each "window" $0 \le a < b \le \delta_p(i, j)$ such that $b - a \ge M - 1$, there exists a vertex k, such that $a \le \delta_p(i, k) \le b$.*

Proof: As all the edge weights in the graph are in the range $[1, M]$, we get that if x and y are two consecutive vertices on p, then $\delta_p(i, y) - \delta_p(i, x) \le M$. The lemma follows immediately. □

We now present a lemma which is an extension for the weighted case of Lemma 2.2 from the previous section. For simplicity, we present the Lemma only for the case $Y = [0, y]$, for some y, which is the case we need in order to prove the correctness of algorithm **WUSP**.

Lemma 3.2 *For every two sets of distances X,Y such that $Y = [0, y]$, where $y \ge M$, and sep$(X) > 2y + 3M$ we have*

$$\Delta_{X \bullet Y} \;\le\; \textit{clip}(\Delta_X \star \Delta_Y, -M, M) \;\le\; \Delta_{X+Y} \;.$$

Proof: Assume that $X = \cup_{r=1}^{p}[a_r, b_r]$, so that $X + Y = \cup_{r=1}^{p}[a_r, b_r + y]$ and $X \bullet Y = \cup_{r=1}^{p}[a_r - y - 2M, b_r + y]$. The condition *sep*$(X) > 2y + 3M$ ensures that $b_{r-1} + y \le a_r - y - 2M$, for $2 \le r \le p$, so the intervals in the representations above are indeed disjoint. We next verify that the matrices Δ_{X+Y} and $\Delta_{X \bullet Y}$ are well defined. To check that $\Delta_{X \bullet Y}$ is well defined, we have to verify that $b_{r-1} + y + M < a_r - y - 2M$, for $2 \le r \le p$. This is equivalent to the condition *sep*$(X) > 2y + 3M$. As $X + Y \subseteq X \bullet Y$, Δ_{X+Y} is also well defined.

We next show that $\Delta_X \star \Delta_Y \le \Delta_{X+Y}$. The inequality *clip*$(\Delta_X \star \Delta_Y, -M, M) \le \Delta_{X+Y}$ will then follow as the clipping operation only changes entries that are below $-M$ to $-M$, and values that are larger than M to $+\infty$, while in Δ_{X+Y} there are no entries below $-M$, and no finite entries larger than M. We show, therefore, that for every $1 \le i, j, \le n$ we have $(\Delta_X \star \Delta_Y)_{ij} \le (\Delta_{X+Y})_{ij}$. If $(\Delta_{X+Y})_{ij} = +\infty$, there is nothing to prove. We assume therefore that $(\Delta_{X+Y})_{ij} < +\infty$. This implies that $\delta(i, j) \in [a_r, b_r + y + M]$, for some $1 \le r \le p$. The proof is now broken into four different cases:

Case 1: $\delta(i, j) \in [a_r, b_r]$

In this case $(\Delta_{X+Y})_{ij} = -M$. We take $k = j$ and then $(\Delta_X)_{ik} \le 0$ and $(\Delta_Y)_{kj} = -M$ and thus $(\Delta_X \star \Delta_Y)_{ij} \le (\Delta_X)_{ik} + (\Delta_Y)_{kj} \le -M$, as required.

Case 2: $\delta(i, j) \in [b_r + 1, b_r + y - M]$

Here again $(\Delta_{X+Y})_{ij} = -M$. In this case we choose a vertex k on a shortest path from i to j such that $\delta(i, k) \in [b_r - M + 1, b_r]$. If $b_r - M + 1 \le 0$, we can take $k = i$, otherwise, the existence of k follows from the "window" lemma (Lemma 3.1). As k lies on a shortest path from i to j, we get that $\delta(i, j) = \delta(i, k) + \delta(k, j)$. As $\delta(i, k)$ belongs to the exact zone of Δ_X, we get that $(\Delta_X)_{ik} = \delta(i, k) - b_r \le 0$. As $\delta(k, j) = \delta(i, j) - \delta(i, k)$, we get that $\delta(k, j) < y$. We consider two subcases. If $\delta(k, j) \le y - M$, then $(\Delta_Y)_{kj} = -M$ and thus $(\Delta_X \star \Delta_Y)_{ij} \le (\Delta_X)_{ik} + (\Delta_Y)_{kj} \le -M$, as required. If $y - M < \delta(k, j) < y$, then $(\Delta_Y)_{kj} = \delta(k, j) - y$ and thus $(\Delta_X \star \Delta_Y)_{ij} \le (\Delta_X)_{ik} + (\Delta_Y)_{kj} \le (\delta(i, k) - b_r) + (\delta(k, j) - y) = \delta(i, j) - b_r - y \le -M$, as required.

Case 3: $\delta(i, j) \in [b_r + y - M + 1, b_r + y]$

In this case $\delta(i, j)$ is in the exact zone of Δ_{X+Y} and so we have $(\Delta_{X+Y})_{ij} = \delta(i, j) - b_r - y$. We choose again a vertex k on a shortest path from i to j such that $\delta(i, k) \in [b_r - M + 1, b_r]$, and thus $(\Delta_X)_{ik} = \delta(i, k) - b_r$. As $\delta(k, j) = \delta(i, j) - \delta(i, k)$, we get that $\delta(k, j) \in [y - M + 1, y + M - 1]$. Thus $\delta(k, j)$ is in the exact zone of Δ_Y and we have $(\Delta_Y)_{kj} = \delta(k, j) - y$. Thus,

$$\begin{aligned}(\Delta_X \star \Delta_Y)_{ij} &\le (\Delta_X)_{ik} + (\Delta_Y)_{kj} \\ &= (\delta(i, k) - b_r) + (\delta(k, j) - y) \\ &= \delta(i, j) - b_r - y \;=\; (\Delta_{X+Y})_{ij} \;.\end{aligned}$$

Case 4: $\delta(i, j) \in [b_r + y + 1, b_r + y + M]$

Here again $(\Delta_{X+Y})_{ij} = \delta(i, j) - b_r - y$. The vertex k on a shortest path from i to j is now chosen so that $\delta(i, k) \in [b_r + 1, b_r + M]$. Again $(\Delta_X)_{ij} = \delta(i, j) - b_r$. As $\delta(k, j) = \delta(i, j) - \delta(i, k)$, we get that $\delta(k, j) \in [y - M + 1, y + M - 1]$. Thus $\delta(k, j)$ is still in the exact zone of Δ_Y so $(\Delta_Y)_{kj} = \delta(k, j) - y$, and

$$\begin{aligned}(\Delta_X \star \Delta_Y)_{ij} &\le (\Delta_X)_{ik} + (\Delta_Y)_{kj} \\ &= (\delta(i, k) - b_r) + (\delta(k, j) - y) \\ &= \delta(i, j) - b_r - y \\ &= (\Delta_{X+Y})_{ij} \;.\end{aligned}$$

This completes the proof that $\Delta_X \star \Delta_Y \le \Delta_{X+Y}$ and thus *clip*$(\Delta_X \star \Delta_Y, -M, M) \le \Delta_{X+Y}$.

We next show that $\Delta_{X \bullet Y} \le$ *clip*$(\Delta_X \star \Delta_Y, -M, M)$. As $X \bullet Y = \cup_{r=1}^{p}[a_r - y - 2M, b_r + y]$, we get that $(X \bullet Y)^* = \cup_{r=1}^{p}[a_r - y - 2M, b_r + y + M]$. Recall that $(\Delta_{X \bullet Y})_{ij} < +\infty$ if and only if $\delta(i, j) \in (X \bullet Y)^*$.

Let $1 \le i, j \le n$. We have to show that $(\Delta_{X \bullet Y})_{ij} \le$ *clip*$(\Delta_X \star \Delta_Y, -M, M)_{ij}$. Let $1 \le k \le n$ be such that $(\Delta_X \star \Delta_Y)_{ij} = (\Delta_X)_{ik} + (\Delta_Y)_{kj}$. If $(\Delta_X)_{ik} = +\infty$ or $(\Delta_Y)_{kj} = +\infty$ then we are done. We can assume, therefore, that $(\Delta_X)_{ik} < +\infty$, $(\Delta_Y)_{kj} < +\infty$. We consider again four different cases:

```
algorithm WUSP(D)

1. ℓ ← ⌈log₂ n⌉
   m ← log₂ M
   for k ← 1 to m + 1 do
       D ← clip(D ⋆ D, 0, 2M)
   end

2. A₀ ← D − M
   for k ← 1 to ℓ do
       A_k ← clip(A_{k−1} ⋆ A_{k−1}, −M, M)
   end

3. C_ℓ ← −M
   P_ℓ ← clip(D, 0, M)
   Q_ℓ ← +∞
   for k ← ℓ − 1 downto 0 do
       C_k ← [clip(P_{k+1} ⋆ A_k, −M, M) ⋀ C_{k+1}] ⋁ [clip(Q_{k+1} ⋆ A_k, −M, M) ⋀̄ C_{k+1}]
       P_k ← P_{k+1} ⋁ Q_{k+1}
       Q_k ← chop(C_k, 1 − M, M)
   end

4. for k ← 1 to ℓ do
       B_k ← (C_k ≥ 0)
   end
   B₀ ← (0 ≤ P₀ < M)
   R ← P₀ mod M
   Δ ← M · Σ_{k=0}^{ℓ} 2^k B_k + R
```

Figure 2. Finding all the distances in an undirected graph with weights from $\{1, \dots, M\}$.

Case 1: $(\Delta_X)_{ik} = -M$ and $(\Delta_Y)_{kj} = -M$
We get that $\delta(i,k) \in [a_r, b_r - M]$, for some $1 \le r \le p$, and that $\delta(k,j) \in [0, y-M]$. Using both sides of the triangle inequality, we get that $\delta(i,j) \in [a_r - y + M, b_r + y - 2M]$. $\delta(i,j)$ is therefore in the clipped zone of $X \bullet Y$ and therefore $(\Delta_{X \bullet Y})_{ij} = -M \le \mathit{clip}(\Delta_X \star \Delta_Y, -M, M)_{ij}$.

Case 2: $(\Delta_X)_{ik} = -M$ and $(\Delta_Y)_{kj} = \delta(k,j) - y > -M$
We get that $\delta(i,k) \in [a_r, b_r - M]$, for some $1 \le r \le p$, and that $\delta(k,j) \in [y - M + 1, y + M]$. Using both sides of the triangle inequality, we get that $\delta(i,j) \in [a_r - y - M, b_r + y] \subset (X \bullet Y)^*$. Thus $(\Delta_{X \bullet Y})_{ij} < +\infty$. If $(\Delta_{X \bullet Y})_{ij} = -M$, then we are done. We may assume, therefore, that $(\Delta_{X \bullet Y})_{ij} = \delta(i,j) - b_r - y > -M$, and thus $\delta(i,j) \in [b_r + y - M + 1, b_r + y]$. We now have to show that $\delta(i,j) - b_r - y = (\Delta_{X \bullet Y})_{ij} \le (\Delta_X)_{ik} + (\Delta_Y)_{kj} = -M + \delta(k,j) - y$, or equivalently, that $\delta(i,j) - \delta(k,j) \le b_r - M$. This follows from the triangle inequality $\delta(i,j) - \delta(k,j) \le \delta(i,k) \le b_r - M$.

Case 3: $(\Delta_X)_{ik} = \delta(i,k) - b_r > -M$ and $(\Delta_Y)_{kj} = -M$
This case is similar to the previous case. We get that $\delta(i,k) \in [b_r - M + 1, b_r + M]$, for some $1 \le r \le p$, and that $\delta(k,j) \in [0, y - M]$. Using both sides of the triangle inequality, we get that $\delta(i,j) \in [b_r - y - 2M + 1, b_r + y] \subset (X \bullet Y)^*$. Thus $(\Delta_{X \bullet Y})_{ij} < +\infty$. If $(\Delta_{X \bullet Y})_{ij} = -M$, then we are done. We may assume, therefore, that $(\Delta_{X \bullet Y})_{ij} = \delta(i,j) - b_r - y > -M$, and thus $\delta(i,j) \in [b_r + y - M + 1, b_r + y]$. We now have to show that $\delta(i,j) - b_r - y = (\Delta_{X \bullet Y})_{ij} \le (\Delta_X)_{ik} + (\Delta_Y)_{kj} = \delta(i,k) - b_r - M$, or equivalently, that $\delta(i,j) - \delta(i,k) \le y - M$. This follows from the triangle inequality $\delta(i,j) - \delta(i,k) \le \delta(k,j) \le y - M$.

Case 4: $(\Delta_X)_{ik} = \delta(i,k) - b_r > -M$ and $(\Delta_Y)_{kj} = \delta(k,j) - y > -M$
We get that $\delta(i,k) \in [b_r - M + 1, b_r + M]$, for some $1 \le r \le p$, and that $\delta(k,j) \in [y - M + 1, y + M]$. As $\mathit{clip}(\Delta_X \star \Delta_Y, -M, M)_{ij} < +\infty$, we get that $(\Delta_X)_{ik} + (\Delta_Y)_{kj} = \delta(i,k) - b_r + \delta(k,j) - y \le M$. Thus, $\delta(i,j) \le \delta(i,k) + \delta(k,j) \le b_r + y + M$. Using the other side of the

triangle inequality, we get that $\delta(i,j) \geq \delta(i,k) - \delta(k,j) \geq (b_r - M + 1) - (y - M) = b_r - y - 2M + 1$. Thus $\delta(i,j) \in [b_r - y - 2M + 1, b_r + y + M] \subset (X \bullet Y)^*$. Thus $(\Delta_{X \bullet Y})_{ij} < +\infty$. If $(\Delta_{X \bullet Y})_{ij} = -M$, then we are done. We may assume, therefore, that $(\Delta_{X \bullet Y})_{ij} = \delta(i,j) - b_r - y > -M$. Thus,

$$\begin{aligned}(\Delta_{X \bullet Y})_{ij} &= \delta(i,j) - b_r - y \\ &\leq \delta(i,k) - b_r + \delta(k,j) - y \\ &= (\Delta_X)_{ik} + (\Delta_Y)_{kj} \ .\end{aligned}$$

This completes the proof of the lemma. □

We are now finally able to explain the operations of algorithm **WUSP** and prove its correctness.

Lemma 3.3 *After stage 1 of algorithm* **WUSP** *we have* $D = clip(\delta, 0, 2M)$.

Proof: In stage 1, the matrix D with the original edge weights of the graph is raised to the $\log_2 M + 1$ power, with entries that become larger than $2M$ immediately replaced by $+\infty$. This clearly gives all the distances in the graph that are at most $2M$. □

Lemma 3.4 *After stage 2 of algorithm* **WUSP***, for every* $0 \leq k \leq \ell$*, we have*

$$A_k = \Delta_{[0, 2^{k+m}]} \ .$$

Proof: A_0 is initialized correctly. The claim for larger values of k follows by induction using Lemma 3.2 and the simple fact that $[0, 2^k] \bullet [0, 2^k] = [0, 2^k] + [0, 2^k] = [0, 2^{k+1}]$, for every $k \geq 0$. □

We now let:

$$\begin{aligned}X_k &= \{ x \mid 0 \leq (x \bmod 2^{k+m+1}) \leq 2^{k+m} \} \ , \\ Y_k &= \{ x \mid (x \bmod 2^{k+m+1}) = 0 \} \ , \\ Z_k &= \{ x \mid (x \bmod 2^{k+m+1}) = 2^{k+m} \} \ .\end{aligned}$$

The sets X_k, Y_k and Z_k, as defined above are infinite. We will only be interested in elements of these sets that are potential distances, i.e., elements smaller than $(n-1)M$. We now have:

Lemma 3.5 *After stage 3 of algorithm* **WUSP***, we have*

$$\begin{aligned}C_k &= \Delta_{X_k} \quad , \quad \text{for every } 1 \leq k \leq \ell \ , \\ P_k &= \Delta_{Y_k} \quad , \quad \text{for every } 0 \leq k \leq \ell \ , \\ Q_k &= \Delta_{Z_k} \quad , \quad \text{for every } 1 \leq k \leq \ell \ .\end{aligned}$$

Proof: We use downward induction on k. It is easy to check that C_ℓ, P_ℓ and Q_ℓ are initialized correctly. We next show that if the claims hold for $k+1$, where $1 \leq k < \ell$, then they also hold for k. We need the following simple relations:

$$\begin{aligned}Y_{k+1} + [0, 2^{k+m}] &= \\ \{ x \mid 0 \leq (x \bmod 2^{k+m+2}) &\leq 2^{k+m} \} \ , \\ Z_{k+1} + [0, 2^{k+m}] &= \\ \{ x \mid 2^{k+m+1} \leq (x \bmod 2^{k+m+2}) &\leq 2^{k+m+1} + 2^{k+m} \} \ , \\ Y_{k+1} \bullet [0, 2^{k+m}] &= \\ \{ x \mid -2^{k+m} - 2^{m+1} \leq (x \bmod 2^{k+m+2}) &\leq 2^{k+m} \} \ , \\ Z_{k+1} \bullet [0, 2^{k+m}] &= \\ \{ x \mid 2^{k+m+1} - 2^{k+m} - 2^{m+1} &\leq (x \bmod 2^{k+m+2}) \\ &\leq 2^{k+m+1} + 2^{k+m} \} \ ,\end{aligned}$$

where, in the relations involving Y_{k+1} we assume that the function $(x \bmod 2^r)$ returns values in the range $(-2^{r-1}, 2^{r-1}]$ while in the other relations, we assume that the values returned are in the range $[0, 2^r)$. Using Lemma 3.2 we get that

$$\begin{aligned}\Delta_{Y_{k+1} \bullet [0, 2^{k+m}]} &\leq clip(P_{k+1} \star A_k, -M, M) \\ &\leq \Delta_{Y_{k+1} + [0, 2^{k+m}]} \ .\end{aligned}$$

Note that for $k \geq 1$, we have $y = 2^{k+m} > M$ and $sep(Y_{k+1}) = 2^{k+m+2} > 2y + 3M + 1$ and Lemma 3.2 is applicable. We next claim that

$$\begin{aligned}C_k^0 &= clip(P_{k+1} \star A_k, -M, M) \bigwedge C_{k+1} \\ &= \Delta_{Y_{k+1} + [0, 2^{k+m}]} \ .\end{aligned}$$

Recall, from the definition of $A \bigwedge B$, that $(A \bigwedge B)_{ij} < +\infty$ only if $b_{ij} < 0$. Consider an element $(C_k^0)_{ij}$ of C_k^0. If $\delta(i,j) \in Y_{k+1} + [0, 2^{k+m}] + [0, M]$, then $(\Delta_{Y_{k+1} \bullet [0, 2^{k+m}]})_{ij} = (\Delta_{Y_{k+1} + [0, 2^{k+m}]})_{ij}$, and therefore $(C_k^0)_{ij} = (\Delta_{Y_{k+1} + [0, 2^{k+m}]})_{ij}$, as required. On the other hand, if $\delta(i,j) \notin Y_{k+1} + [0, 2^{k+m}] + [0, M]$, then it is easy to check that $(C_{k+1})_{ij} \geq 0$, and therefore $(C_k^0)_{ij} = +\infty = (\Delta_{Y_{k+1} + [0, 2^{k+m}]})_{ij}$, again as required.
Using very similar arguments, we get that

$$\begin{aligned}C_k^1 &= clip(Q_{k+1} \star A_k, -M, M) \bar{\bigwedge} C_{k+1} \\ &= \Delta_{Z_{k+1} + [0, 2^{k+m}]} \ .\end{aligned}$$

It is easy to verify that $\Delta_X \bigvee \Delta_Y = \Delta_{X \cup Y}$, provided that the *distance* between X and Y is at least M, i.e., if $x \in X$ and $y \in Y$, then $|x - y| \geq M$. As

$$(Y_{k+1} + [0, 2^{k+m}]) \cup (Z_{k+1} + [0, 2^{k+m}]) = X_k \ ,$$

and as the distance condition on these two sets is satisfied, we get that

$$C_k = C_k^0 \bigvee C_k^1$$

$$\begin{aligned} &= \Delta_{Y_{k+1}+[0,2^{k+m}]} \bigvee \Delta_{Z_{k+1}+[0,2^{k+m}]} \\ &= \Delta_{X_k} \ , \end{aligned}$$

as required. As

$$Y_k = Y_{k+1} \cup Z_{k+1} \ ,$$

and as the distance between Y_{k+1} and Z_{k+1} is large enough, we get that

$$P_k \ = \ P_{k+1} \bigvee Q_{k+1} \ = \ \Delta_{Y_{k+1}} \bigvee \Delta_{Z_{k+1}} \ = \ \Delta_{Y_k} \ .$$

This relation also holds for $k = 0$. Finally, the operation $chop(C_k, 1-M, M)$ extracts the entries of C_k that belong to the exact zone. It is easy to verify that the matrix obtained is the desired Q_k. □

In stage 4, algorithm **WUSP** extracts the individual bits of the distances, and their remainder modulo M, and reconstructs all the distances.

Lemma 3.6 *After stage 4 of algorithm* **WUSP**, *we have*

$$\begin{array}{rcl} B_k &=& [(\delta \bmod 2^{k+m+1}) \geq 2^{k+m}], \\ && \text{for every } 0 \leq k \leq \ell \ , \\ R &=& (\delta \bmod M) \ , \\ \Delta &=& \delta \ . \end{array}$$

Proof: For $k \geq 1$, we have $(B_k)_{ij} = 1$ if and only if $(C_k)_{ij} \geq 0$. By Lemma 3.5, this happens when $(\delta(i,j) \bmod 2^{k+m+1}) \geq 2^{k+m}$, as required. Special care is taken in the computation of B_0. This is done as the claim regarding C_k made in Lemma 3.5 holds only for $k \geq 1$. For $k = 0$ we have $(B_0)_{ij} = 1$ if and only if $0 \leq (P_0)_{ij} < M$. By the claim made in Lemma 3.5 regarding P_k, a claim which holds also for $k = 0$, we get that $(B_0)_{ij} = 1$ if and only if $(\delta(i,j) \bmod 2^{m+1}) \geq 2^m$, as required. For the same reason, we get that $R = (\delta \bmod M)$. It is easy to see that $(B_k)_{ij}$ is just the $(k+m)$-th bit in the binary representation of $\delta(i,j)$, where $M = 2^m$. The claim that $\Delta_{ij} = \delta(i,j)$, for every $1 \leq i,j \leq n$, follows immediately. □

Algorithm **WUSP** performs only $O(\log n + \log M) = O(\log(Mn))$ distance products of matrices whose finite elements are either in the range $[0, 2M]$ (this is the case in stage 1), or in the range $[-M, M]$ (this is the case in stages 2 and 3). It is not difficult to see that each such distance product can be reduced to a constant number of distance products of matrices with elements in the range $[1, M]$. All other operations performed by **WUSP** take only $O(n^2(\log n + \log M))$ time. We thus get:

Theorem 3.7 *Algorithm* **WUSP** *finds all the distances in a weighted undirected graph whose edge weights are in the range* $\{1, 2, \ldots, M\}$ *using* $O(\log(Mn))$ *distance products of matrices whose finite elements are in the range* $\{1, 2, \ldots, M\}$ *and its total running time is therefore* $\tilde{O}(Mn^{\omega})$*, where* $\omega < 2.376$ *is the exponent of matrix multiplication.*

4. Finding shortest paths

Once all the distances in the graph were found, we can find a concise representation of shortest paths between all pairs of vertices in the graph using only three *witnessed* Boolean matrix products, in the unweighted case, or three *witnessed* distance products in the weighted case. For the unweighted case, this was shown by Seidel [11]. We extend his technique to the weighted case.

Let A and B two $n \times n$ matrices . An $n \times n$ matrix W is said to be a matrix of *witnesses* for the distance product $A \star B$ if for every $1 \leq i,j \leq n$ we have $1 \leq w_{ij} \leq n$ and $(A \star B)_{ij} = a_{i,w_{ij}} + b_{w_{ij},j}$. As shown in Zwick [16], a matrix of witnesses for the distance product $A \star B$ of two $n \times n$ matrices with finite elements in the range $\{1, 2, \ldots, M\}$ can also be found in $\tilde{O}(Mn^{\omega})$.

Let $G = (V, E)$ be an *undirected* graph on n vertices with edge weights in the range $\{1, 2, \ldots, M\}$. Let D be a matrix holding the edge weights of the graph, but this time with $+\infty$ on the main diagonal,that is $D_{ii} = +\infty$ for any i. Let Δ be the matrix of distances in the graph. To represent the shortest paths in the graph we use a matrix S, called a matrix of *successors*, such that for every $1 \leq i,j \leq n$, there is a shortest path from i to j that starts with the edge (i, s_{ij}). Using the matrix S, a shortest path from i to j in G can be reconstructed in time which is linear in the number of edges used in this shortest path.

Let Δ' be an $n \times n$ matrix such that

$$\Delta'_{ij} = \begin{cases} (\delta(i,j) \bmod 3M) & \text{if } (\delta(i,j) \bmod 3M) \geq M \ , \\ +\infty & \text{otherwise .} \end{cases}$$

Let W be a matrix of witnesses for the distance product $D \star \Delta'$. In the next lemma we show that if $(\delta(i,j) \bmod 3M) \geq 2M$ then there is a shortest path from i to j that starts with the edge (i, w_{ij}), so we can let $s_{ij} \leftarrow w_{ij}$.

Lemma 4.1 *If* $2M \leq \Delta'_{ij} < 3M$*, then there is a shortest path from* i *to* j *that starts with the edge* (i, w_{ij}).

Proof: It is easy to see that if w_{ij} is a witness for the element $\Delta_{ij} = (D \star \Delta)_{ij} = \min_k\{d_{ik} + \Delta_{kj}\}$, i.e., if $(D \star \Delta)_{ij} = d_{i,w_{ij}} + \Delta_{w_{ij},j}$, then w_{ij} can serve as a successor for the pair i, j, i.e., $\delta(i,j) = \delta(i.w_{ij}) + \delta(w_{ij}, j)$ and there is a shortest path from i to j that starts with the edge (i, w_{ij}). We compute instead a witness w_{ij} for the element $(D \star \Delta')_{ij}$. We show now that if $2M \leq \Delta'_{ij} < 3M$, then this witness w_{ij} is also a witness for the element $(D \star \Delta)_{ij}$.

Suppose that $\Delta_{ij} = \delta(i,j) = a \cdot 3M + \Delta'_{ij}$, for some a, where $2M \le \Delta'_{ij} < 3M$. If $(i,k) \in E$, then $0 < \delta(i,k) \le M$ and by the triangle inequality, we get that $\delta(i,j) - M \le \delta(k,j) \le \delta(i,j) + M$. It follows therefore that $a \cdot 3M + M \le \Delta_{kj} < (a+1) \cdot 3M + M$ and thus $\Delta'_{kj} < +\infty$ if and only if $a \cdot 3M + M \le \Delta_{kj} \le (a+1) \cdot 3M$, in which case $\Delta'_{kj} = \Delta_{kj} - a \cdot 3M$. In other words,

$$(D \star \Delta')_{ij} \;=\; \min_{k \,|\, \Delta_{kj} < (a+1)\cdot 3M} \{\, d_{ik} + \Delta_{kj} \,\} - a \cdot 3M \ .$$

On the other hand, it is easy to see that the minimum $(D \star \Delta)_{ij} = \min_k\{d_{ik} + \Delta_{kj}\}$ is attained by a k for which $\Delta_{kj} < \Delta_{ij} < (a+1) \cdot 3M$. Thus

$$(D \star \Delta)_{ij} \;=\; \min_{k \,|\, \Delta_{kj} < (a+1)\cdot 3M} \{\, d_{ik} + \Delta_{kj} \,\} \ .$$

It follows, therefore, that any witness for $(D \star \Delta')_{ij}$ is also a witness for $(D \star \Delta)_{ij}$, and vice versa. □

Successors for the other pairs, i.e., pairs for which $0 \le (\delta(i,j) \bmod 3M) < M$, or $M \le (\delta(i,j) \bmod 3M) < 2M$, can be easily found using two additional distance products. We thus get:

Theorem 4.2 *Given the distances in a weighted undirected graph on n vertices with edge weights in the range $\{1, 2, \ldots, M\}$, a concise description of shortest paths between all pairs of vertices can be found using a constant number of distance products of $n \times n$ matrices whose finite elements are in the range $\{1, 2, \ldots, M\}$.*

5. Concluding remarks

We showed that the All Pairs Shortest Paths (APSP) problem in directed graphs with edge weights taken from the range $\{1, 2, \ldots, M\}$ can be solved using $O(\log(Mn))$ distance products of two $n \times n$ matrices with elements in the range $\{1, 2, \ldots, M\}$. As each such distance product can be computed in $\tilde{O}(Mn^{\omega})$ time, we got an $\tilde{O}(Mn^{\omega})$ time algorithm for the APSP problem, improving a previous $\tilde{O}(M^{(\omega+1)/2}n^{\omega})$ time algorithm of Galil and Margalit [8],[9].

The obvious open problem is whether there exist better algorithms for computing distance products. The distance product of two $n \times n$ matrices can be computed in $O(n^3)$ time using the naive algorithm, or in $\tilde{O}(Mn^{\omega})$ time using fast matrix multiplication algorithms. Can the distance product of two $n \times n$ matrices with elements in the range $\{1, 2, \ldots, M\}$ be computed in $O(n^{3-\epsilon} \log M)$ time, for some fixed $\epsilon > 0$?

The algorithms given here work only for *undirected* graphs. Is the APSP problem for directed graphs really harder than the corresponding problem for undirected graphs?

References

[1] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54:255–262, 1997.

[2] E. Cohen and U. Zwick. All-pairs small-stretch paths. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, Louisiana*, pages 93–102, 1997.

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

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

[5] D. Dor, S. Halperin, and U. Zwick. All pairs almost shortest paths. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, Burlington, Vermont*, pages 452–461, 1996. Full version submitted to SIAM Journal on Computing.

[6] M. Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5:49–60, 1976.

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

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

[9] Z. Galil and O. Margalit. All pairs shortest paths for graphs with small integer length edges. *J. Comput. Syst. Sci.*, 54:243–254, 1997.

[10] D. Johnson. Efficient algorithms for shortest paths in sparse graphs. *Journal of the ACM*, 24:1–13, 1977.

[11] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51:400–403, 1995.

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

[13] T. Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica*, 20:309–318, 1998.

[14] M. Thorup. Undirected single source shortest paths in linear time. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, Miami Beach, Florida*, pages 12–21, 1997.

[15] M. Thorup. Floats, integers, and single source shortest paths. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science, Paris, France*, pages 14–24, 1998.

[16] U. Zwick. All pairs shortest paths in weighted directed graphs – exact and almost exact algorithms. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, California*, pages 310–319, 1998.

[17] U. Zwick. All pairs lightest shortest paths. In *Proceedings of the 31th Annual ACM Symposium on Theory of Computing, Atlanta, Georgia*, pages 61–69, 1999.