INFORMATION AND COMPUTATION **134**, 103–139 (1997)
ARTICLE NO. IC972620

All Pairs Shortest Distances for Graphs with Small Integer Length Edges

Zvi Galil*

Department of Computer Science, Tel-Aviv University and Columbia University

and

Oded Margalit

Department of Computer Science, Tel-Aviv University

There is a way to transform the All Pairs Shortest Distances (APSD) problem where the edge lengths are integers with small ($\leqslant M$) absolute value into a problem with edge lengths in $\{-1, 0, 1\}$. This transformation allows us to use the algorithms we developed earlier ([1]) and yields quite efficient algorithms. In this paper we give new improved algorithms for these problems. For $n = |V|$ the number of vertices, M the bound on edge length, and ω the exponent of matrix multiplication, we get the following results:

1. A directed nonnegative APSD(n, M) algorithm which runs in $O(T(n, M))$ time, where

$$T(n, M) = \begin{cases} M^{(\omega-1)/2} n^{(3+\omega)/2}, & 1 \leqslant M \leqslant n^{(3-\omega)/(\omega+1)} \\ Mn^{(5\omega-3)/(\omega+1)}, & n^{(3-\omega)/(\omega+1)} \leqslant M \leqslant n^{2(3-\omega)/(\omega+1)}. \end{cases}$$

2. A undirected APSD(n, M) algorithm which runs in $O(M^{(\omega+1)/2} n^{\omega} \log(Mn))$ time.

1. INTRODUCTION

In this paper we show how to improve our algorithms for the All Pairs Shortest Distances (APSD) problem for the case where edge lengths are integers. The algorithms are efficient (subcubic) when the edge lengths are of small absolute value.

Let $G = \langle V, E \rangle$ be a directed graph with $n = |V|$ vertices $\{v_1, v_2, ..., v_n\}$ and $m = |E|$ edges. Let $D = \{d_{ij}\}_{i,j=1}^{n}$ be a distance function from the edges to the integers with ∞. ($d_{ij} = \infty$ if and only if $v_i \to v_j$ is not an edge in E).

* Work supported in part by NSF Grant CCR9014605 and NSF CISE Institutional Infrastructure Grant CDA-90-24735.

0890-5401/97 $25.00

A *path* P from v_i to v_j in G is a sequence of vertices $v_i = v_{k_0}, v_{k_1}, \ldots, v_{k_l} = v_j$. The number of edges in the path is defined as l. If $l = 0$ we call the path an *empty path*. The *length* of P is defined as $d_{ij}^P \stackrel{\text{def}}{=} \sum_{r=1}^{l} d_{k_{r-1}k_r}$. For every two vertices v_ρ and v_μ, on the path P, where $\rho = k_{\rho'}$ and $\mu = k_{\mu'}$, $\rho' < \mu'$, we define $d_{\rho\mu}^P$ as the length of the subpath from v_ρ to v_μ: $\sum_{r=\rho'+1}^{\mu'} d_{k_{r-1}k_r}$.

A *cycle* is a path from v_i to v_i. A *negative cycle* is a cycle whose length is negative.

Given a matrix A, we use the notation A_{ij} for the element in the ith row and jth column of A. There are two exceptions, D and D^*, where we use the lower case letters d_{ij} and d_{ij}^* respectively.

The All Pairs Shortest Distances problem is the problem of finding, for every pair of vertices v_i and v_j in V, the length of a shortest path from v_i to v_j. When there is a path from v_i to v_j that contains a negative cycle, we denote $d_{ij}^* = -\infty$. We denote the solution matrix by $D^* = \{d_{ij}^*\}_{i,j=1}^n$.

We use the notation APSD(n, M) for the problems on graphs with n vertices and edge lengths that are either ∞ or integers whose absolute value is bounded by M.

Let A be an integer matrix and a be an integer. The notation $[A \leqslant a]$ stands for a Boolean matrix defined by

$$[A \leqslant a]_{ij} = \begin{cases} 1, & A_{ij} \leqslant a \\ 0, & \text{otherwise.} \end{cases}$$

Similarly define the notations $[A = a]$, $[A \in X]$ for a set X, etc. We also use relations on matrices which are defined elementwise. For example, $A \leqslant B$ holds if and only if $A_{ij} \leqslant B_{ij}$ for all i and j.

Let A and B be $n \times n$ Boolean matrices. Let C be the $n \times n$ Boolean matrix product of A and B be denoted by $C = A \cdot B$ and defined as $C_{ij} = \bigvee_{k=1}^n A_{ik} \wedge B_{kj}$.

The trivial way to compute C takes $\Theta(n^3)$ time. There are faster ways to compute it. Denote the time complexity of the fastest $n \times n$ matrix multiplication over a ring by $M(n)$. Define ω, the exponent of matrix multiplication, to be the infimum over all real numbers x for which matrix multiplication of $n \times n$ matrices can be computed in $O(n^x)$ time. Currently, the best bound on ω is $\omega < 2.376$ (see [3]). Although it is not exactly true (the infimum might not be a minimum) we assume that $M(n) = O(n^\omega)$.

In [1] we gave an $O(n^{(3+\omega)/2} \log(n))$ time algorithm for APSD(n, 1). Using a simple reduction (see Lemma 1 below) we get an $O((Mn)^{(3+\omega)/2} \log(Mn))$ time algorithm for APSD(n, M). This algorithm is subcubic for $M < n^{0.113}$ (using the best known bound on ω). In this paper we improve this algorithm in the case of nonnegative edge lengths to $O(T(n, M))$ time, where

$$T(n, M) = \begin{cases} M^{(\omega-1)/2} n^{(3+\omega)/2}, & 1 \leqslant M \leqslant n^{(3-\omega)/(\omega+1)} \\ Mn^{(5\omega-3)/(\omega+1)}, & n^{(3-\omega)/(\omega+1)} \leqslant M \leqslant n^{2(3-\omega)/(\omega+1)}. \end{cases}$$

Note that the upper bound on M in the second case is not a restriction since for $M > n^{2(3-\omega)/(\omega+1)}$ we have $Mn^{(5\omega-3)/(\omega+1)} > n^3$. The new algorithm is subcubic for $M \leqslant n^{2(3-\omega)/(\omega+1)} \approx n^{0.370}$.

In the undirected case, better algorithms are known for the APSD(n, 1) problem. We designed an $O(n^{\omega}\log(n))$ time algorithm (see Subsection 3.1 below). Independently, Seidel [4] designed a very elegant and simple algorithm with the same time bound. But Seidel's algorithm does not seem to be generalizable even to APSD(n, 2). Both algorithms can be expanded to find the paths as well (the All Pairs Shortest *Path* Problem), but this is beyond the scope of this paper. The simple reduction mentioned above is not applicable to undirected graphs. Nevertheless, we can generalize our algorithm to handle longer edges and derive an algorithm for the undirected APSD(n, M) of time $O(M^{(\omega+1)/2}n^{\omega}\log(Mn))$. (Here edge lengths need not be nonnegative.) The new algorithm is subcubic for $M \leqslant n^{2(3-\omega)/(\omega+1)} \approx n^{0.370}$.

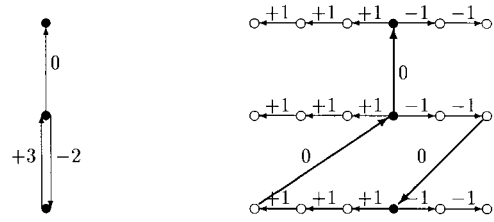
In the rest of this section we review the transformation of an APSD(n, M) problem into an APSD(n', 1) problem. We refer to this transformation in Sections 3 and 4. In Section 2 we review the directed positive APSD(n, 1) algorithm. In Section 3 we give our algorithm for the undirected APSD(n, 1) and generalize it to APSD(n, M), obtaining an $O(M^2n^{\omega}\log(Mn))$ time bound. In Section 4 we show how to pack matrix multiplications and improve the algorithms of the previous sections; we generalize the positive APSD(n, 1) algorithm of Section 2 and obtain a nonnegative APSD(n, M) algorithm; and we improve the undirected APSD(n, M) algorithm of Section 3. In Section 5 we state some open problems.

To solve problems with small (in absolute value) integer edge lengths, transform the given graph into another graph which has more vertices, but only zero or ± 1 length edges.

The simplest way to transform a general APSD(n, M) problem into an APSD(n', 1) is to replace every edge by a path of edges of length 1 by adding new vertices. This transformation has $n' = n + |E|\,(M-1)$ and can yield $n' \geqslant n^2$, which makes this transformation useless, even for $M = 2$. The following lemma uses a more efficient transformation.

LEMMA 1. *Given a graph G which has edge lengths in the interval $[x, y]$, where $x \leqslant 0 \leqslant y$ and $y - x < M$, we can, in time which is linear in the output, compute another graph G' with $M \cdot |V(G)|$ vertices and edge lengths in $\{-1, 0, 1\}$ only. Every shortest distance between a pair of vertices in G can be found as a shortest distance between a corresponding pair of vertices in G'.*

Proof. Transform each vertex v_i in G into M sub-vertices, $\{v_i^k\}_{k=x}^{y}$. Call the sub-vertex v_0^i the *origin vertex* of the vertex v_i. Connect the sub-vertices of v_i in two paths: one with edges of length $+1$ connects the sub-vertices v_i^k with positive k's and the other path with -1 length edges connects the sub-vertices v_i^k with negative k's. Formally, the edges are $\{(v_i^{k-1}, v_i^k)\}_{k=1}^{y}$ with length 1, $\{(v_i^{k+1}, v_i^k)\}_{k=x}^{-1}$ with length -1. For every edge (v_i, v_j) of the original graph G which has length l, connect an edge (v_i^l, v_j^0) with zero edge length. It is easy to see that the shortest distance in G between v_i and v_j, is the same as the shortest distance in G' between the corresponding origin sub-vertices v_i^0 and v_j^0. This transformation enlarges the size of the graph by a factor of M and the running time accordingly. (Figure 1 shows an example where $x = -2$ and $y = 3$.) ∎



The original graph G The transformed graph G'

FIG. 1. Transforming long edges to (signed) uniform ones.

One can avoid using zero edge lengths in the reduction of Lemma 1 by using a ± 1 length edge from one vertex before.

In this paper we sometimes find it simpler to allow empty paths (as in Section 3 and Subsection 4.2) and sometimes (as in Section 2 and Subsection 4.1) we disallow empty paths. We now show that the two versions are essentially equivalent.

LEMMA 2. *The time complexities of the two versions of the APSD(n) problem, one allowing empty paths and the other not allowing them, differ by at most a constant factor.*

Proof. We begin with showing that allowing empty paths is trivial: Denote the solution of the APSD(n) problem when empty paths are not allowed with D^*, and the solution when empty paths are allowed with E^*. Then

$$E^*_{ij} = \begin{cases} D^*_{ij} & i \neq j \\ \min\{0, D^*_{ij}\} & i = j, \end{cases}$$

since the only difference is in the empty paths.

Next we show that we can disallow empty paths as well: Given the input graph D, build another graph E by duplicating every vertex v into two vertices v_1 and v_2 and for every edge $v \to w$ in D, make two edges $v_1 \to w_2$ and $v_2 \to w_2$. Let E^* be the solution for the APSD problem on the graph E allowing empty paths, and let D^* be the solution for the APSD problem on the graph D not allowing empty paths. Then

$$D^*_{ij} = E^*_{i_1 j_2}.$$

This follows from the fact that every nonempty path in D can be translated into a path in E, and every path from v_1 to w_2 in E can be translated into a nonempty path in D. The extra work is $O(n^2)$ plus doubling the size of the graph; both are less than a constant factor. ∎

2. FINDING THE SHORTEST DISTANCES IN THE POSITIVE DIRECTED CASE

In this section we review our algorithm [1] for the directed APSD(n, 1). Here we do not allow empty paths. The main idea is to solve first the small distances ($\leqslant L$) and then extend the solutions in a geometric progression.

Denote $f(\rho) = (3/2)^\rho L$, the integer L is defined later.

1. Using L Boolean matrix multiplications, find d^*_{ij} for all pairs ij with distance of at most L in the graph. This can be done using

$$[D^* \leqslant \rho] = [D \leqslant \rho] \vee ([D = 1] \cdot [D^* \leqslant \rho - 1]).$$

The computation above yields the matrix $D(0)$:

$$D(0)_{ij} = \begin{cases} d^*_{ij}, & d^*_{ij} \leqslant L \\ \infty, & \text{otherwise.} \end{cases}$$

The time complexity of this step is $O(Ln^\omega)$.

2. For $\rho = 0, ..., \lceil \log_{3/2}(n/L) \rceil - 1$, iterate the following step, each time computing the matrix $D(\rho+1)$ from the matrix $D(\rho)$.

(a) For each vertex v_i, find a distance s_i in the interval $[f(\rho)/2, f(\rho)]$ such that there are no more than $2n/f(\rho)$ vertices which are at this shortest distance from v_i. Denote this set by S_i. This can be done in $O(n^2)$ time by inspecting the matrix $D(\rho)$ (we prove it in Lemma 5).

(b) For every two vertices v_i and v_j denote $m_{ij\rho} =^{\text{def}} \min_{v_l \in S_i} \{D(\rho)_{il} + D(\rho)_{lj}\}$ and compute

$$D(\rho+1)_{ij} = \begin{cases} D(\rho)_{ij}, & D(\rho)_{ij} \leqslant f(\rho) \\ m_{ij\rho}, & D(\rho)_{ij} > f(\rho) \text{ and } m_{ij\rho} \leqslant f(\rho+1) \\ \infty, & \text{otherwise.} \end{cases}$$

This can be done in $O(n^3/f(\rho))$ time.

Fact 3. Any sub-path of a shortest path is a shortest path as well.

COROLLARY. 4. *If v_k is on a shortest path from v_i to v_j then $d^*_{ij} = d^*_{ik} + d^*_{kj}$.*

Proof. $d^*_{ij} = d^P_{ij} = d^P_{ik} + d^P_{kj} = d^*_{ik} + d^*_{kj}$. ∎

LEMMA 5. *The algorithm described above is correct and it runs in $O(Ln^\omega + n^3/L)$ time.*

Proof. For the correctness, we show that the matrices $D(k)$ which are computed in the algorithm above satisfy

$$D(\rho)_{ij} = \begin{cases} d^*_{ij}, & d^*_{ij} \leqslant f(\rho) \\ \infty & \text{otherwise.} \end{cases}$$

The proof is by induction on ρ: Step 1 computes $D(0)$ which satisfies this inequality by definition. Suppose that the induction hypothesis holds for ρ; we show that it holds for $\rho+1$. Examine d^*_{ij}. There are three cases:

$d^*_{ij} \leqslant f(\rho)$: From the induction hypothesis, $D(\rho)_{ij} = d^*_{ij} \leqslant f(\rho) \leqslant f(\rho+1)$ and therefore $D(\rho+1)_{ij} = D(\rho)_{ij} = d^*_{ij}$.

$f(\rho) < d^*_{ij} \leqslant f(\rho+1)$: From the induction hypothesis, $D(\rho)_{ij} = \infty > f(\rho)$. There is a vertex v_l on a shortest path from v_i to v_j such that $d^*_{il} = s_i$ (since $d^*_{ij} > f(\rho) \geqslant s_i$). $d^*_{il} \leqslant f(\rho)$ and from Corollary 4, $d^*_{lj} = d^*_{ij} - d^*_{il} \leqslant f(\rho+1) - f(\rho)/2 = f(\rho)$. Therefore $D(\rho+1)_{ij} \leqslant D(\rho)_{il} + D(\rho)_{lj} = d^*_{il} + d^*_{ij} = d^*_{ij}$. Clearly, $D(\rho+1)_{ij} \geqslant d^*_{ij}$ since

$$\forall l,\ D(\rho)_{il} + D(\rho)_{lj} \geqslant d^*_{il} + d^*_{ij} \geqslant d^*_{ij}.$$

So $D(\rho+1)_{ij} = d^*_{ij}$.

$f(\rho+1) < d^*_{ij}$: From the induction hypothesis, $D(\rho)_{ij} = \infty > f(\rho)$. Clearly, $D(\rho)_{ij} \geqslant d^*_{ij}$ and therefore

$$m_{ijp} = \min_{v_l \in S_i} \{D(\rho)_{il} + D(\rho)_{lj}\} \geqslant \min_{v_l \in S_i} \{d^*_{il} + d^*_{ij}\} \geqslant d^*_{ij} > f(\rho+1).$$

Hence, $D(\rho+1)_{ij} = \infty$.

As for the time complexity, Step 1 takes $O(Ln^\omega)$ time since we perform L Boolean matrix multiplications. Step 2 takes

$$O\left(\sum_{\rho=0}^{\lceil \log_{3/2}(n/L) \rceil - 1} (n^2 + n^3/f(\rho))\right) = O(n^3/L)$$

time. ∎

THEOREM 1. *The positive unweighted case of the APSD problem can be solved in* $O(n^v)$ *time, where* $v = (3+\omega)/2 < 3$.

Proof. Apply Lemma 5 with $L = n^{(3-\omega)/2}$. ∎

3. UNDIRECTED APSD

In this section we show a solution for the undirected APSD(n). Here we find it simpler to allow empty paths. First we explain why the directed transformation (Lemma 1) does not work in the undirected case. Next we show how to take care of nonpositive edges. Next we introduce several notations, and then we show the solution for the undirected APSD(n, M). Since the algorithm is quite complicated, we first explain our algorithm for the unweighted case (Subsection 3.1) and only then (Subsection 3.2) we show the full solution and prove its correctness.

The transformation described in Lemma 1 does not work for undirected graphs since it uses directed edges. Even if the original graph is undirected, the transformed one is directed. Trying to use undirected edges in the transformation changes the shortest distances. It may introduce incorrect zero distances. Even for the modified transformation that does not introduce zero length edges, a simple counter example (see Fig. 2) proves that making the edges undirected can distort shortest distance even when $M = 2$. The figure shows the directed transformation of a graph. The shortest distance between the upper vertex and the lower vertex in the original graph is 4. This holds for the (directed) transformed graph. However, making the edges of the transformed graph undirected leads to a path of length 2 between the lower and upper vertices.

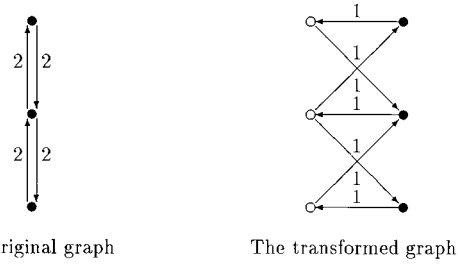


FIG. 2. The transformation does not work for undirected graphs.

We can transform the undirected APSD(n, M) problem into the directed APSD($n, 1$) problem, but we can solve it faster in another way. First note that while for directed graphs, zero length edges and in particular negative edge length add much to the complexity of the algorithm; for undirected graphs it is very easy to get rid of these edges and we can assume without loss of generality positive edge lengths only:

LEMMA 6. *In the undirected problem we can remove nonpositive edges from the input graph in* $O(n^2)$ *time.*

Proof. First we take care of the negative edges. For every negative edge (which gives a negative cycle) find its connected component and remove it from G. The shortest distance between any pair of vertices in the same component is $-\infty$ and between any two vertices in different components is $+\infty$. The time complexity of this step is $O(|E|) = O(n^2)$.

Now for the zero length edges, contract the connected components of the zero length edges in the graph which contains only these edges. Then define the length of a newly formed edge as the minimum over all the original edges which were contracted to it. ▌

An important property of distances in undirected graphs is that the full version of the triangle inequality holds:

LEMMA 7. $d^*_{ik} - d^*_{kj} \leqslant d^*_{ij} \leqslant d^*_{ik} + d^*_{kj}$.

Proof. The second inequality, the original triangle inequality, follows from the definition: d^*_{ij} is the length of the *shortest* path from v_i to v_j, and therefore not longer than any other path, including the concatenation of a shortest path from v_i to v_k and a shortest path from v_k to v_j. This includes all the cases of infinite ($+\infty$) length edges shortest paths.

The first inequality holds only for the undirected case: From the second inequality and the symmetric property of d^* in the undirected case it follows that

$$d^*_{ik} \leqslant d^*_{ij} + d^*_{jk} = d^*_{ij} + d^*_{kj}.$$

Subtracting d^*_{kj} from both sides completes the proof. ▌

Our algorithm handles paths by multiplying matrices, which is equivalent to path concatenation. In the directed case, when only half of the triangle inequality holds, there is no way to have a lower bound on the shortest distance between the endpoints of the concatenated path. In the directed case, we can use the second

half as well, which allows us to work with a more general set of possible shortest distances. In Section 2 we built the small ($\leqslant L$) distances in the directed case one by one by subtracting the sets $[D^* < x+1] \backslash [D^* < x]$. (We use the definition $A \backslash B \stackrel{\text{def}}{=} \{x: x \in A \text{ and } x \notin B\}$ for set subtraction.) This method required to compute linear (in L) number of matrices. Now, in the undirected case, we can hope (due to the other side of the triangle inequality) to do it much faster by building the logarithmic number of sets, for example, the sets [D^* has the ith bit on] and intersecting them. Unfortunately, we could not find a way to build these binary sets. We did, however, manage to solve the problem modulo three.

Digits of numbers in base 2 are called bits. When writing numbers in base 3, we use the term *trit* for the digits. Recall that $r = \lceil \log_3 N \rceil$ and $N = nM$. The output of our algorithm consists of integers which are less than N. For each one of the r trits of the result we compute 3 Boolean matrices which contain all the pairs i, j whose shortest distance has this ternary digit set to 0, 1 or 2. Then, packing these matrices together, we get the shortest distance written in ternary form.

As we stated in the introduction, the general algorithm is quite complicated; therefore we start with some definitions, then we handle the unweighted case. In Subsection 3.1 we give the algorithm for APSD(n, 1). We explain later some typical steps, but we do not prove correctness because it will follow from the correctness of the more general algorithm for weighted graphs given in Subsection 3.2.

We represent a set of numbers by writing a pattern in base 3; the pattern consists of trits (0, 1, or 2) and a special sign x which stands for a "don't care." To make the computation simpler, we add one to the patterns. We use angled braces to distinguish patterns from regular numbers. For example, the pattern $\langle 1\mathrm{x}2 \rangle$ means the set of all the three-trits-long ternary numbers whose most significant trit is 1 and least significant trit is 2 by adding one to them; i.e.,

$$\langle 1\mathrm{x}2 \rangle = \{12, 15, 18\}.$$

We shift these sets by adding a constant. For example,

$$\langle 10\mathrm{xx}0 \rangle - 2 = \{80, 83, 86, 89, 92, 95, 98, 101, 104\}.$$

We use the exponentiation operator to denote a repetition of the same digit. For example,

$$\langle \mathrm{x}^3 0^2 \rangle = \langle \mathrm{xxx}00 \rangle.$$

Note that the three don't care signs do not necessarily represent the same digit. All the patterns we use in this paper have $r = \lceil \log_3 N \rceil$ length, $N = nM$, where the graph has n vertices and the edge lengths are bounded from above by M.

We define operations on sets,

$$X + Y = \{a + b : a \in X, b \in Y\},$$
$$X - Y = \{a - b : a \in X, b \in Y\},$$
$$X + i = \{a + i : a \in X\},$$

and on Boolean matrices,

$$(\neg A)_{ij} = \neg A_{ij},$$
$$(A \vee B)_{ij} = A_{ij} \vee B_{ij},$$
$$(A \wedge B)_{ij} = A_{ij} \wedge B_{ij}.$$

We use $\oplus$ for addition modulo 3 and $\ominus$ for subtraction modulo 3.

3.1. *The Unweighted Undirected Case*

The algorithm we describe in this subsection is built of three main steps. The first step computes a rough estimate (the most significant trit); the second step computes more intermediate results; and the third step combines the outputs of the other steps into the final result. We first give a summary of the algorithm to explain the main ideas behind each step, then we describe what each step does in more details, then sketch the algorithm itself.

1. The first step gives a rough estimate on the shortest distance between any two pairs of nodes: In this step $\log_3 n$ matrices are computed, and the ith matrix captures all the shortest distances that are less than 3^i. (Actually we are computing $3 \log_3 n$ matrices, but we refer to the first out of each three matrices.) In other words, this step gives the most significant trit (MST) of the distance between any pair of nodes.

2. Once the MST of the distance between any pair of vertices is known, one can proceed to improve the resolution of the distances. This is done by computing the rest of the digits by computing the second MST, third MST, and so on. This is done using the following observation. If the shortest distance between two vertices is less than $2 \cdot 3^i$ and more than 3^i, then the ith trit is 1. Similarly, if that distance is more than $2 \cdot 3^i$ and less than 3^{i+1}, then the ith bit should be 2 and so on.

3. The output of the previous step would be $3 \log_3 n$ Boolean matrices, with the $(3i+k)$th matrix denoting when the ith trit of any distance is k or not. The final step computes the distances from these Boolean matrices.

To understand the algorithm better, we visualize $[D^* \in X]$ by drawing the set of integers X on a horizontal line. The line is drawn in precise scale, and every segment represents a single element. A solid segment means that the corresponding element is in the set, a vacant place means that it is missing. A dashed line has a special meaning, which is explained later. Such drawings are used in Figs. 3 and 4.

The algorithm consists of three steps. The first step is similar to the exponentiation step used to solve the transitive closure using logarithmic number of matrix multiplications. The difference is that the transitive closure algorithm computes the matrices $[D^* \in [1+k2^l, (k+1)2^l]]$, for $0 \leqslant l < \log_2(n)$ and $k \in \{0, 1\}$; and the first step of our algorithm computes the matrices $[D^* \in \langle k \mathrm{x}^l \rangle] = [D^* \in [1+k3^l, (k+1)3^l]]$ for $0 \leqslant l < r$ and $k \in \{0, 1, 2\}$, i.e., it works in base 3 instead of base 2.

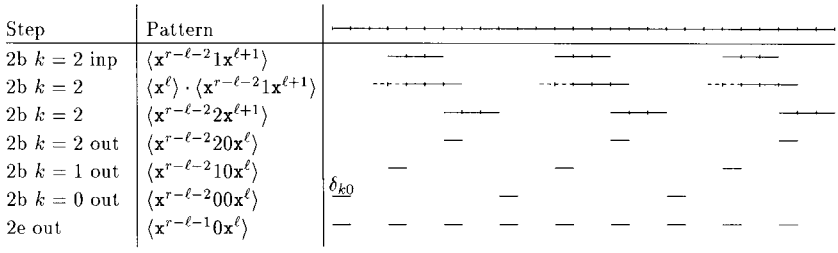


FIG. 3. Some explanations for Step 2.

The motivation for base 3 is explained later. The first step is illustrated by the first 12 lines of Figure 4.

The second step is the heart of our algorithm. This step is the only place where we use the undirectedness of the input graph. In this step we compute the ternary "combs." A ternary comb is a set of integers which correspond to a pattern of the type $\langle x^{r-l-1}kx^{l}\rangle$ for $0\leqslant l<r$ and $k\in\{0,1,2\}$. In Fig. 4 we can see several such combs (in each of the bottom 12 lines in Fig. 4) and understand the origin of the name. The horizontal axis represents the integer numbers in the interval $[1,81]$. Each line stands for a set of integers. The way we derive a line from the lines above it in the second step is not so straight forward as in the first step, and an example is shown in Fig. 3. Explanations for this part are given later.

The third and last step combines the combs to form the solution. The idea is that every comb actually gives us one trit of the shortest distance. The lth trit (counting the least significant as zero) of d^{*}_{ij} is k if and only if $[D^{*}\in\langle x^{r-l-1}kx^{l}\rangle]_{ij}=1$. So gathering the trits of D^{*} is quite easy.

We now sketch the algorithm and then explain why it works.

1. This step is used to generate the basic segments: $\langle kx^{l}\rangle$ for $0\leqslant k\leqslant 2$ and $0\leqslant l<r$. These segments are used to build the combs in the second step.

For $0\leqslant l<r$, compute $[D^{*}\in\langle x^{l}\rangle]$, $[D^{*}\in\langle 1x^{l}\rangle]$ and $[D^{*}\in\langle 2x^{l}\rangle]$. We start with $l=0$ (Substep 1a). Note that $\langle x^{0}\rangle$ is $\langle 0\rangle$. Then we iterate Substeps 1b to 1d r times with $l=0,1,\ldots,r-1$. Actually, we can omit the last iteration of Substep 1d, because it computes the all-ones matrix.

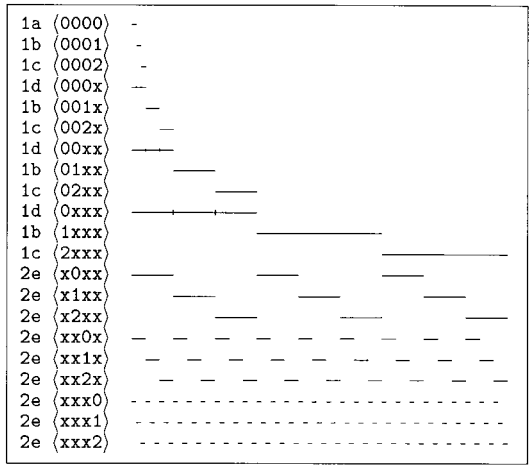


FIG. 4. Illustration of the undirected algorithm.

(a) For $l=0$, $[D^* \in \langle 0 \rangle]$ can be easily computed from the input D,

$$[D^* \in \langle 0 \rangle]_{ij} = [D^* = 1]_{ij} = \begin{cases} 1 & d_{ij} = 1 \\ 0 & \text{otherwise;} \end{cases}$$

i.e., we simply replace the $+\infty$ distances in D with zeros.

(b) $[D^* \in \langle 1\mathrm{x}^l \rangle] = ([D^* \in \langle \mathrm{x}^l \rangle] \cdot [D^* \in \langle \mathrm{x}^l \rangle]) \wedge \neg(I \vee [D^* \in \langle \mathrm{x}^l \rangle])$.

(c) $[D^* \in \langle 2\mathrm{x}^l \rangle] = ([D^* \in \langle \mathrm{x}^l \rangle] \cdot [D^* \in \langle 1\mathrm{x}^l \rangle])$
$\wedge \neg(I \vee [D^* \in \langle \mathrm{x}^l \rangle] \vee [D^* \in \langle 1\mathrm{x}^l \rangle])$.

(d) $[D^* \in \langle \mathsf{x}^{l+1} \rangle] = [D^* \in \langle \mathrm{x}^l \rangle] \vee [D^* \in \langle 1\mathrm{x}^l \rangle] \vee [D^* \in \langle 2\mathrm{x}^l \rangle]$.

2. This step, is the main step of the algorithm. It computes the combs $\langle \mathsf{x}^{r-l-1} k \mathsf{x}^l \rangle$ for $0 \leqslant k \leqslant 2$ and $0 \leqslant l < r$. This step uses the output of the previous step. We explain it in detail later.

Compute $[D^* \in \langle \mathrm{x}^{r-l-1} k \mathsf{x}^l \rangle]$, for $0 \leqslant l < r$ and $k \in 0, 1, 2$. We start with $l = r-1$ (Substep 2a) and iterate the other substeps for $l = r-2, r-3, ..., 0$.

(a) The matrices $[D^* \in \langle k\mathsf{x}^{r-1} \rangle]$ for $k \in \{0, 1, 2\}$ were computed in the previous step (for $l = r-1$).

(b) Compute $[D^* \in \langle \mathrm{x}^{r-l-2} k 0 \mathsf{x}^l \rangle]$ as

$$(([D^* \in \langle \mathrm{x}^l \rangle] \cdot [D^* \in \langle \mathrm{x}^{r-l-2} (k \ominus 1)\, \mathsf{x}^{l+1} \rangle]) \vee \delta_{k0} [D^* \in \langle \mathrm{x}^l \rangle])$$
$$\wedge [D^* \in \langle \mathrm{x}^{r-l-2} k \mathsf{x}^{l+1} \rangle],$$

where δ_{k0} is the Kronecker δ, i.e., one if $k=0$ and zero otherwise.

(c) Compute $[D^* \in \langle \mathrm{x}^{r-l-2} k 1 \mathsf{x}^l \rangle]$ as

$$([D^* \in \langle \mathrm{x}^l \rangle] \cdot [D^* \in \langle \mathrm{x}^{r-l-2} k 0 \mathsf{x}^l \rangle])$$
$$\wedge \neg([D^* \in \langle \mathrm{x}^{r-l-2} k 0 \mathsf{x}^l \rangle] \vee [D^* \in \langle \mathrm{x}^{r-l-2} (k \ominus 1)\, \mathsf{x}^{l+1} \rangle]).$$

(d) Compute $[D^* \in \langle \mathrm{x}^{r-l-2} k 2 \mathsf{x}^l \rangle]$ as

$$[D^* \in \langle \mathrm{x}^{r-l-2} k \mathsf{x}^{l+1} \rangle] \wedge \neg([D^* \in \langle \mathrm{x}^{r-l-2} k 1 \mathsf{x}^l \rangle] \vee [D^* \in \langle \mathrm{x}^{r-l-2} k 0 \mathsf{x}^l \rangle]).$$

(e) Compute $[D^* \in \langle \mathrm{x}^{r-l-1} k \mathsf{x}^l \rangle]$ as $\bigvee_{\rho=0}^{2} [D^* \in \langle \mathrm{x}^{r-l-2} \rho k \mathsf{x}^l \rangle]$.

3. This last step combines the output of the previous steps into a solution for the APSD problem. Once the combs are computed, it is straightforward to compute D^* from them.

Compute

$$D^* = 1 + \sum_{l=0}^{r-1} 3^l \sum_{k=0}^{2} k [D^* \in \langle \mathrm{x}^{r-l-1} k \mathsf{x}^l \rangle].$$

Substep 1a is trivial; a path of length one can only be achieved by a single edge. As for Substep 1b, it is correct because $3^l < d^*_{ij} \leqslant 2 \cdot 3^l$ if and only if $\exists k$, $d^*_{ik}, d^*_{kj} \leqslant 3^l$, and $d^*_{ij} > 3^l$. Recall that we add 1 in the definition of the patterns, so that $\langle \mathrm{x}^l \rangle$ is the interval $[1, 3^l]$ and this simplifies the computations. Substep 1c is similar. Substep 1d trivially follows from the definitions.

Substep 2a follows from the definitions too. Substep 2b is essentially the heart of this algorithm. In this substep we refine the combs. This operation only works in the undirected case. We denoted by $[D^* \in A]$ the Boolean matrix which corresponds to the set A of shortest distances. We take a Boolean matrix which corresponds to the ternary comb with 3^{l+1} length tooth. The case $k = 2$ is shown in the first line of Fig. 3. (k does not affect the figure much, it just fixes the phase of the comb.) Then we multiply it with the 3^l length interval that starts at the origin. The result is shown in the second line of the same figure. The result contains a shift of the comb and a "shadow" (the dashed lines) which we explain next. If for a pair ij d^*_{ij} is in the shifted comb (the solid part) then the ijth entry of the multiplication is one. If, on the other hand, the ijth coordinate of the multiplication is zero, then d^*_{ij} must be outside the interval with its shadow. What this means is that we did not capture the shift operation we wanted to by using the multiplication, but rather added a shadow part which may contain errors. The reason for the errors is that when we concatenate two shortest paths, one from v_i to v_k and the other from v_k to v_j, then the shortest distance between v_i and v_j may be less than the sum of the shortest distances d^*_{ik} and d^*_{kj}, because we can make a detour. However, the situation is not that bad because of the other direction of the triangle inequality: appending a short path cannot decrease the shortest distance by more than its length.

Multiplying $[D^* \in \langle \mathrm{x}^l \rangle]$ by a matrix is equivalent to concatenating to a path of length no more than 3^l. Concatenating such a path (say from v_i to v_k) to another path (say from v_k to v_j) yields a path P (from v_i to v_j) whose length can not be longer by more than 3^l from the length of a shortest path d^*_{ij}. Note that the only place where we use the full version of the triangle inequality was in Substeps 2b and 2c above. Intersecting the multiplication (second line of Fig. 3) with another comb which was computed before (third line of Fig. 3) gives us a third of a comb with smaller tooth: only 3^l (fourth line of Fig. 3). The fifth and sixth lines show the other two thirds of the comb ($k = 0$ and $k = 1$). In the sixth line there is a correction term with δ_{k0} that is needed because the multiplication always shifts the combs to the right and thus the case $k = 0$ need to be taken care of separately. Substep 2c is very similar to Substep 2b. Substeps 2d and 2e are simple: again, following immediately from the definitions. The last line in Fig. 3 is the sum of the three lines above it and it corresponds to Substep 2e.

We need the full version of the triangle inequality to bound the distance from below. Without the other side, we would have the following problem: The pattern $\langle 2\mathrm{x}^{r-1} \rangle$ contains $\langle 2^r \rangle = \{3^r\}$ which is $\geqslant N$. Multiplying $[D^* \in \langle 2\mathrm{x}^{r-1} \rangle]$ by any non-zero matrix may lead to the all one matrix $[D^* \in \langle \mathrm{x}^r \rangle]$ since there is no lower bound, in the directed case, on the possible distances which can result from concatenating a very long path with another (even short) path. Thus, while in the undirected case the shadows are short, in the directed case the shadows can be very

long and cannot be "intersected out." Here we can justify the base 3 computation too. In base 2, the shadows would overlap in such a way that will make it impossible to separate them. Any other base would be fine, and for efficiency reasons we chose the smallest nonbinary base.

THEOREM 2. *The undirected APSD*$(n, 1)$ *problem can be solved in* $O(n^{\omega}\log(n))$ *time.*

Our algorithm is slightly better than Seidel's algorithm because while Seidel's algorithm performs $O(\log(n))$ *integer* matrix multiplications (with entries bounded by n), ours uses $O(\log(n))$ *Boolean* matrix multiplications. Seidel's algorithm is considerably simpler.

We end this subsection by giving an example. To show clearly the combs, we need to have at least 3-digit (ternary) numbers. An example with that many vertices is hard to follow, if all the details are given, so we chose to show a little smaller example shown in Fig. 5. The eight parts are as follows.

1. The original graph. This is the input for Substep 1a.
2. The intermediate multiplication done in Substep 1b.
3. The result of the first application of Substep 1b.
4. The intermediate multiplication done in Substep 1c.
5. The result of the first application of Substep 1c.
6. The result of the first application of Substep 1d.
7. The result of the second application of Substep 1b.
8. The result of the second application of Substep 1c.

The computations in Step 2 are not shown because the graph is too small to show the combs, and they degenerate into single points. The "shadows" which appear in Step 2 can be shown in the transition between item 2 to item 3 and in

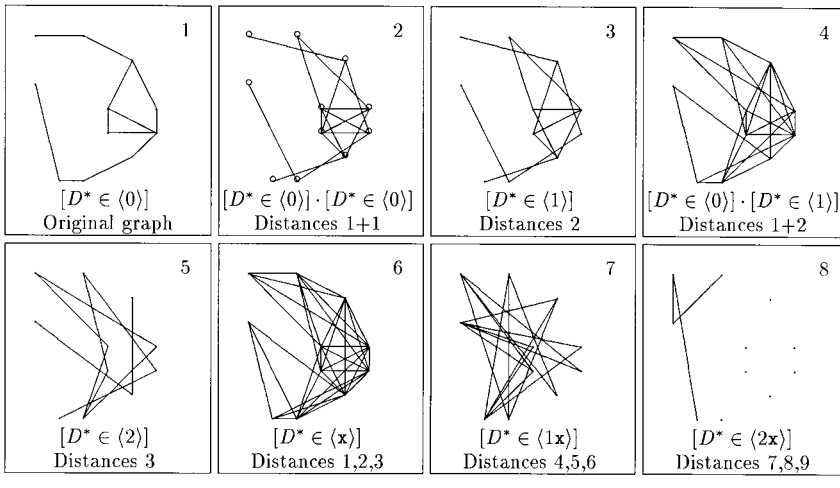


FIG. 5. Example of the unweighted directed algorithm.

the transition between item 4 to item 5 in Fig. 5. It is easy to see that item 2 contains all distance 2, but it also contains some shadow: distances zero (marked as small loops on the nodes) for example. As for the second example (3 to 4) there are shadows, but they are only distance 1, since one cannot get a zero distance from $2+1$ in undirected graphs. To show the whole algorithm we use the x-axis as the interval $[1, n]$, where $n = 3^4 = 81$. We draw, in Fig. 4, the sets as we go along.

3.2. *The Weighted Undirected Case*

In this subsection we describe the general algorithm for the weighted case, and prove its correctness. This algorithm is a generalization of the undirected algorithm described in Subsection 3.1. The algorithm works for weighted graphs, where the weights are integer values up to M. When $M = 1$, the algorithm is exactly the one described in Subsection 3.1.

We start with some observations and definitions, then give the algorithm, describe the difference between it and the unweighted one, and finally prove its correctness. The algorithm again has three steps, here are the differences.

1. This step also gives a rough estimate. In this step and in the entire algorithm we are working on $2M-1$ matrices at once, instead of a single Boolean matrix in the unweighted case.

2. We compute the secondmost trit and the thirdmost trit, but we do not go all the way to the least significant trit, but rather stop at the $\log_3 M$ least significant trit.

3. Combining the output of the previous step requires a correction term to compensate for the missing least significant trits. The new term is derived from the matrices we already computed.

LEMMA 8. *If* $d_{ij} \in \{1, 2, ..., M, \infty\}$ *then for all* k, $[D^* \leqslant k] = \bigvee_{\delta=1}^{M} [D = \delta] \cdot [D^* \leqslant k - \delta]$.

Proof. If $[D^* \leqslant k]_{ij} = 1$ then there exists a path P from v_i to v_j of length no more than k. Let v_l be the first vertex on this path. $1 \leqslant d_{il} = \delta \leqslant M$ and the rest of the path P from v_l to v_j has length no more than $k - \delta$. Hence the right hand side is one.

If, on the other hand, the right hand side is one, then there exists some l and $1 \leqslant \delta \leqslant M$ such that $[D = \delta]_{il} = 1$ and $[D^* \leqslant k - \delta]_{lj} = 1$. Therefore there exists a path from v_i via v_l to v_j whose length is no more than k. ∎

Without loss of generality, M is a power of 3 (we enlarge M to the next power of 3). Note that 1 is a power of 3. The main problem in solving the general case is that while in the uniform case we have

$$[D^* \in (A + B)] \leqslant [D^* \in (A)] \cdot [D^* \in (B)],$$

this does not hold in the general case because it might be that the distance from v_i to v_j is $a + b$, $a \in A$ and $b \in B$, but there is no intermediate vertex v_k such that $d^*_{ik} = a$

and $d^*_{kj} = b$ since any shortest path "jumps" over distance a using a "long" edge (length >1).

However, such a jump cannot be too long: it is bounded by M. So we can find $[D^* \in (A+B)]$ by computing the logical or of M matrix multiplications: $[D^* \in (A+\delta)] \cdot [D^* \in (B-\delta)]$ for δ in some interval of length M. That way, no path can jump itself out of all the matrix multiplications. Our algorithm is iterative, using previous matrices as inputs to later steps. So we must be able to compute a range of M consecutive matrices for each matrix of the uniform case algorithm. We found it useful to maintain $2M-1$ matrices.

The algorithm is very similar to that of Subsection 3.1, but we keep $2M-1$ matrices for each matrix we computed there. Instead of computing $[D^* \in A]$, which was a single matrix, we compute a sequence of $2M-1$ matrices. We call this sequence a *super-matrix* and denote it by $G(A)$,

$$G(A) \stackrel{\text{def}}{=} \{G(A)^m\}_{m=1-M}^{M-1},$$

where

$$G(A)^m = [D^* \in (A+m)]. \tag{1}$$

Operations on super-matrices are performed on each matrix separately. G clearly satisfies the facts

$$G(A \cup B) = G(A) \vee G(B), \tag{2}$$

$$G(A \cap B) = G(A) \wedge G(B), \tag{3}$$

and

$$G(A \backslash B) = G(A) \wedge \neg G(B). \tag{4}$$

The initial step of the algorithm is more complicated. Lemma 9 explains it and proves its time complexity.

LEMMA 9. *The matrices* $\{[D^* = m]\}$ *for* $0 \leqslant m \leqslant M$, *the super-matrix* $G(\langle 0 \rangle)$, *and the super-matrix* $G([1-M, 0])$ *can be somputed in* $O(M^2 n^\omega)$ *time.*

Proof. By definition, $G(\langle 0 \rangle)^m = [D^* \in (\langle 0 \rangle + m)] = [D^* = m+1]$ and it is the zero matrix for $m < -1$. For $m = -1$, $[D^* = m+1 = 0] = I$. For $0 \leqslant m \leqslant M-1$, it can be computed using

$$[D^* = m+1] = [D^* \in [0, m+1]] \wedge \neg [D^* \in [0, m]],$$

and by Lemma 8

$$[D^* \in [0, m+1]] = \bigvee_{\delta=1}^{M} [D = \delta] \cdot [D^* \in [0, m+1-\delta]]. \tag{5}$$

Similarly, $G([1-M, 0])^m = [D^* \in ([m+1-M, m])] = [D^* \in [0, m]]$. The time required for this algorithm is M^2 matrix multiplications. ∎

In the rest of this subsection A and B represent sets of positive integers.

Adapting the algorithm to super-matrices is quite straight forward: The logical operations ($\neg$, $\wedge$, and $\vee$) on super-matrices are performed, as mentioned above, on each matrix separately. The super-matrix multiplication is done as follows, $(G(A) \cdot G(B))^m$ for $1-M \leqslant m < 0$ is defined as

$$\bigvee_{\delta=0}^{M-1} [D^* \in (A+\delta+1-M)] \cdot [D^* \in (B+M-1+m-\delta)] \tag{6}$$

and for $0 \leqslant m \leqslant M-1$ as

$$\bigvee_{\delta=0}^{M-1} [D^* \in (A+\delta)] \cdot [D^* \in (B+m-\delta)]. \tag{7}$$

The main reason for the summation in the definition above is to make sure that no path can jump over the concatenation. $G(A) \cdot G(B)$ is a super-matrix, and it represents, in some sense $G(A+B)$. It is *not* $G(A+B)$, but, as we will see later (Corollary 14), it is close enough. We do not change the definition of G-multiplication, because we want it to generalize the definition of the normal matrix multiplication used in the unweighted case. As we explained there, the matrix multiplication does not only shift the combs, but also creates some shadows. There are two reasons for the different definitions in the two cases ((6) and (7)). The first reason is that a single definition would have been more complex:

$$\bigvee_{\delta=0}^{M-1} [D^* \in (A+\lfloor (m+1-M)/2 \rfloor + \delta)] \cdot [D^* \in (B+\lceil (m+M-1)/2 \rceil - \delta)].$$

The second reason is that the split into two cases will be helpful in Subsection 4.2.

The algorithm has the same three steps as the algorithm of Subsection 3.1. In steps 1 and 2 we replace the matrix $[D^* \in A]$ with the super-matrix $G(A)$. There are additional changes in the algorithm. We give the full algorithm again, and sketch the differences afterwards.

1. For $0 \leqslant l < r$, compute $G(\langle \mathsf{x}^l \rangle)$, $G(\langle 1\mathsf{x}^l \rangle)$ and $G(\langle 2\mathsf{x}^l \rangle)$. We start with $l=0$ (Substep 1a) and iterate r times Substeps 1b to 1d with $l = 0, 1, ..., r-1$.

   (a) For $l=0$ (recall that $\langle \mathsf{x}^0 \rangle$ is $\langle 0 \rangle$), $G(\langle 0 \rangle)$ can be computed from the input D. It is actually the solution for distances $[1, M]$ which can be computed in $O(M^2 n^\omega)$ time (see Lemma 9 for details).

   (b) $G(\langle 1\mathsf{x}^l \rangle) = (G(\langle \mathsf{x}^l \rangle) \cdot G(\langle \mathsf{x}^l \rangle)) \wedge \neg(G([1-M, 0]) \vee G(\langle \mathsf{x}^l \rangle))$.

   (c) $G(\langle 2\mathsf{x}^l \rangle) = (G(\langle \mathsf{x}^l \rangle) \cdot G(\langle 1\mathsf{x}^l \rangle)) \wedge \neg(G([1-M, 0]) \vee G(\langle \mathsf{x}^l \rangle) \vee G(\langle 1\mathsf{x}^l \rangle))$.

   (d) $G(\langle \mathsf{x}^{l+1} \rangle) = G(\langle \mathsf{x}^l \rangle) \vee G(\langle 1\mathsf{x}^l \rangle) \vee G(\langle 2\mathsf{x}^l \rangle)$.

2. Compute $G(\langle \mathsf{x}^{r-l-1}k\mathsf{x}^{l}\rangle)$, for $\log_3 M \leqslant l < r$ and $k \in 0, 1, 2$. We start with $l = r-1$ (Substep 2a) and iterate the other substeps for $l = r-2, r-3, ..., \log_3 M$.

(a) $G(\langle k\mathsf{x}^{r-1}\rangle)$ for $k \in \{0, 1, 2\}$, were computed in the previous step (for $l = r-1$).

(b) Compute $G(\langle \mathsf{x}^{r-l-2}k0\mathsf{x}^{l}\rangle)$ as

$$((G(\langle \mathsf{x}^{l}\rangle)\cdot G(\langle \mathsf{x}^{r-l-2}k\ominus 1)\ \mathsf{x}^{l+1}\rangle)) \vee \delta_{k0}G(\langle \mathsf{x}^{l}\rangle)) \wedge G(\langle \mathsf{x}^{r-l-2}k\,\mathsf{x}^{l+1}\rangle),$$

where δ_{k0} is the Kronecker δ, i.e., one if $k = 0$ and zero otherwise.

(c) Compute $G(\langle \mathsf{x}^{r-l-2}k1\mathsf{x}^{l}\rangle)$ as

$$x^{l}\rangle)\cdot G(\langle \mathsf{x}^{r-l-2}k0\mathsf{x}^{l}\rangle)) \wedge \neg(G(\langle \mathsf{x}^{r-l-2}k0\mathsf{x}^{l}\rangle) \vee G(\langle \mathsf{x}^{r-l-2}(k\ominus 1)\ \mathsf{x}^{l+1}\rangle)).$$

(d) Compute $G(\langle \mathsf{x}^{r-l-2}k2\mathsf{x}^{l}\rangle)$ as

$$G(\langle \mathsf{x}^{r-l-2}k\mathsf{x}^{l+1}\rangle) \wedge \neg(G(\langle \mathsf{x}^{r-l-2}k1\mathsf{x}^{l}\rangle) \vee G(\langle \mathsf{x}^{r-l-2}k0\mathsf{x}^{l}\rangle)).$$

(e) Compute $G(\langle \mathsf{x}^{r-l-1}k\mathsf{x}^{l}\rangle)$ as $\bigvee_{\rho=0}^{2} G(\langle \mathsf{x}^{r-l-2}\rho k\mathsf{x}^{l}\rangle)$.

3. Compute

$$\begin{aligned} D^* = 1 + \sum_{l=\log_3 M}^{r-1} 3^{l} \sum_{k=0}^{2} k[\,D^* \in \langle \mathbb{x}^{r-l-1}k\mathsf{x}^{l}\rangle\,] \\ + \sum_{m=1}^{M-1} \sum_{k=0}^{2} ([\,D^* \in (\langle \mathsf{x}^{r-\log_3 M-1}k\mathsf{x}^{\log_3 M}\rangle + m)\,] \\ \wedge [\,D^* \in \langle \mathsf{x}^{r-\log_3 M-1}k\mathsf{x}^{\log_3 M}\rangle\,]). \end{aligned}$$

We have the matrices needed for this computation, since they were computed in Step 2 as $G(\langle \mathsf{x}^{r-l-1}k\mathsf{x}^{l}\rangle)$ for $l \geqslant \log_3 M$.

- Substep 1a is not so trivial as before.
- Step 2 is performed only for $l \geqslant \log_3 M$.
- Step 3 is changed: a new summation term is added.

Note that for $M = 1$, we get that the super-matrix $G(A)$ is the same as the regular matrix $[\,D^* \in A\,]$, and $\log_3 M = 0$. Therefore Steps 1–3 for this algorithm degenerate to Steps 1–3 in the algorithm of Subsection 3.1 (In Step 3 the last term disappears since the summation is from 1 to 0.) Note that for $M = 1$, $G([\,1-M, 0\,]) = I$.

We now prove the correctness of the modified algorithm. Consequently also the algorithm of Subsection 3.1 is proven correct. First, we prove two lemmas (Lemma 10 and Lemma 11), which enable us to bound the multiplication from below. Then we prove two lemmas (Lemma 12 and Lemma 13) which allow us to bound the multiplication from above. Lastly, we combine them to two corollaries (Corollary 14 and Corollary 15) which enable us to prove correctness.

LEMMA 10. *For* $1-M \leqslant m < 0$,

$$[D^* \in (A+B+m)] \leqslant \bigvee_{\delta=0}^{M-1} [D^* \in (A+\delta+1-M)] \cdot [D^* \in (B+m+M-1-\delta)].$$

Proof. If the ijth coordinate of the left hand side is one, then there exists $a \in A$ and $b \in B$ such that $d^*_{ij} = a+b+m$. Examine a shortest path P from v_i to v_j. Let v_k be the first vertex on P whose distance from v_i is no less than $a+1-M$. As we stated after the proof of Lemma 9, B contains only positive elements. Therefore every $b \in B$, $b > 0$. So v_k exists because $d^*_{ij} = a+b+m \geqslant a+1-M$. Note that k can be i. Let $\delta_0 = d^*_{ik} - a + M - 1$. Clearly $\delta_0 \geqslant 0$. The length of every edge is no more than M, therefore $\delta_0 < M$ (if not, it would contradict the definition of v_k as the *first* vertex such that $d^*_{ik} \geqslant a+1-M$). $d^*_{ik} = a+\delta_0+1-M$ so $[D^* \in (A+\delta_0+1-M)]_{ik} = 1$. Using Corollary 4 we get $d^*_{kj} = d^*_{ij} - d^*_{ik} = a+b+m-(\delta_0+a+1-M) = b+m+M-1-\delta_0$ which implies that $[D^* \in (B+M-1+m-\delta_0)]_{kj} = 1$. So we proved that the ijth coordinate of the right hand side is one too. ∎

LEMMA 11. *For* $0 \leqslant m \leqslant M-1$,

$$[D^* \in (A+B+m)] \leqslant \bigvee_{\delta=0}^{M-1} [D^* \in (A+\delta)] \cdot [D^* \in (B+m-\delta)].$$

Proof. Similar to the previous proof. ∎

LEMMA 12. *For* $1-M \leqslant m < 0$,

$$\bigvee_{\delta=0}^{M-1} [D^* \in (A+\delta+1-M)] \cdot [D^* \in (B+m+M-1-\delta)] \leqslant [D^* \in (B+m+[2-2M-\max(A), \max(A)])].$$

Proof. If the ijth coordinate of the left hand side is one, then there exists an integer δ_0, $0 \leqslant \delta_0 \leqslant M-1$, integers $a \in A$, $b \in B$ and an index k such that $d^*_{ik} = a+\delta_0+1-M$ and $d^*_{kj} = b+M-1+m-\delta_0$. Apply the full version of the triangle inequality (Lemma 7):

$$d^*_{ij} \leqslant d^*_{ik} + d^*_{kj} = a+b+m \leqslant b+m+\max(A), \tag{8}$$

and

$$d^*_{ij} = d^*_{ji} \geqslant d^*_{jk} - d^*_{ki} = b+M-1+m-\delta_0-(a+\delta_0+1-M) \geqslant b+m-\max(A). \tag{9}$$

Combining (8) and (9) yields the desired result. ∎

LEMMA 13. *For* $0 \leqslant m \leqslant M-1$,

$$\bigvee_{\delta=0}^{M-1} [D^* \in (A+\delta)] \cdot [D^* \in (B+m-\delta)] \leqslant [D^* \in (B+m+[2-2M-\max(A), \max(A)])].$$

Proof. If the ijth coordinate of the left hand side is one, then there exists an integer δ_0, $0 \leqslant \delta_0 \leqslant M-1$, integers $a \in A$, $b \in B$ and an index k such that $d^*_{ik} = a+\delta_0$ and $d^*_{kj} = b+m-\delta_0$. Apply the full version of the triangle inequality (Lemma 7):

$$d^*_{ij} \leqslant d^*_{ik} + d^*_{kj} = a+b+m \leqslant b+m+\max(A), \tag{10}$$

and

$$d^*_{ij} = d^*_{ji} \geqslant d^*_{jk} - d^*_{ki} = b+m-\delta_0-(a+\delta_0) \geqslant b+m+2-2M-\max(A). \tag{11}$$

Combining (10) and (11), yields the desired result. ∎

COROLLARY 14.

$$G(A+B)^m \leqslant (G(A)\cdot G(B))^m \leqslant G(B+[2-2M-\max(A), \max(A)])^m.$$

Proof. If $G(A+B)^m_{ij} = 1$ then there exist integers $a \in A$ and $b \in B$ such that $d^*_{ij} = a+b+m$. By Lemma 10 (or Lemma 11) and the definition of G-multiplication (6) (or (7)), we get $(G(A)\cdot G(B))^m_{ij} = 1$.

If $(G(A)\cdot G(B))^m_{ij} = 1$ then by Lemma 12 (or Lemma 13) we get that $[D^* \in (B+m+[2-2M-\max(A), \max(A)])]_{ij} = 1$. ∎

COROLLARY 15. *For any integers* $b \geqslant a > 0$,

$$G([b+1, b+a]) = (G([1, a])\cdot G([b-a+1, b])) \wedge \neg G([1-M, b]).$$

Proof. Denote $A = [1, a]$ and $B = [b-a+1, b]$. By Corollary 14,

$$G([b-a+2, a+b])^m \leqslant (G(A)\cdot G(B))^m \leqslant G([b+3-2M-2a, a+b])^m. \tag{12}$$

Use (4) and (2) to intersect this inequality with $\neg G([1-M, b])^m$. $b+1 \geqslant b-a+2$, and $G([-\infty, -M]) = 0$, so

$$G([b-a+2, a+b]) \wedge \neg G([1-M, b]) = G([b-a+2, -M]) \vee G([b+1, a+b]) = G([b+1, a+b]).$$

$b+1 \geqslant b+3-2M-2a$, hence

$$G([b+3-2M-2a, a+b]) \wedge \neg G([1-M, b]) = G([b+3-2M-2a, -M]) \vee G([b+1, a+b]) = G([b+1, a+b]).$$

Both sides are equal, hence (12)after intersecting with $\neg G([1-M, b])$ becomes an equality. ∎

Now we can prove the correctness of the algorithm.

LEMMA 16. *Step* 1 *is correct.*

Proof. Substep 1a is justified by Lemma 9. We use Corollary 15 to prove the correctness of the following two substeps of Step 1: For Substep 1b use $a=b=3^l$, and for Substep 1c use $a=3^l$ and $b=2\cdot 3^l$. (2) proves that $G([1-M, b])=G([1-M, 0]) \vee G([1, b])$. Substep 1d is an immediate consequence of (2). ∎

Next we prove the correctness of Step 2. The formal detailed proof is quite complicated, but the main idea is pictured in Fig. 3. We slide the rough comb, intersect it with another comb to remove the shadows and the unwanted parts, and get the finer comb. We iterate until we get the finest comb with M-long teeth.

LEMMA 17. *Step* 2 *is correct.*

Proof. The correctness of the first substep follows from the correctness of Step 1. The last two substeps are immediate consequence of (2) and (4). As for the remaining two, we cannot use Corollary 15, since the sets are not contiguous intervals, so we use Corollary 14. Consider Substep 2b: From the definition of the patterns,

$$\begin{aligned} A &\stackrel{\text{def}}{=} \langle \mathrm{x}^l \rangle = [1, 3^l], \\ B &\stackrel{\text{def}}{=} \langle \mathrm{x}^{r-l-2}(k\ominus 1)\,\mathrm{x}^{l+1} \rangle \\ &= \{\alpha 3^{l+2}+\beta : \delta_{k0} \leqslant \alpha < 3^{r-l-2}+\delta_{k0},\ (k-1)\,3^{l+1} < \beta \leqslant k\,3^{l+1}\} \end{aligned} \tag{13}$$

and

$$C \stackrel{\text{def}}{=} \langle \mathrm{x}^{r-l-2} k 0 \mathrm{x}^l \rangle = \{\alpha 3^{l+2}+\beta : 0 \leqslant \alpha < 3^{r-l-2},\ k\,3^{l+1} < \beta \leqslant k\,3^{l+1}+3^l\}.$$

Observe that

$$A+B=\{\alpha 3^{l+2}+\beta : \delta_{k0} \leqslant \alpha < 3^{r-l-2}+\delta_{k0},\ (k-1)\,3^{l+1}+1 < \beta \leqslant k\,3^{l+1}+3^l\}$$

and

$$\begin{aligned} B+[2-2M-\max(A), \max(A)] &= \{\alpha 3^{l+2}+\beta : \delta_{k0} \leqslant \alpha < 3^{r-l-2}+\delta_{k0},\ (k-1)\,3^{l+1} \\ &\quad -3^l+2-2M < \beta \leqslant k\,3^{l+1}+3^l\}. \end{aligned}$$

Using Corollary 14 we get

$$\begin{aligned} G(A+B) \vee \delta_{k0} G(A) &\leqslant G(A)\cdot G(B) \vee \delta_{k0} G(A) \\ &\leqslant G(B+[2-2M-\max(A), \max(A)]) \vee \delta_{k0} G(A). \end{aligned}$$

Using (2) we get (note that when $k=0$ the $\alpha=0$ part is added by the $\vee \delta_{k0} G(A)$ term)

$$\begin{aligned} G(\{\alpha 3^{l+2}+\beta : 0 \leqslant \alpha < 3^{r-l-2}+\delta_{k0}, \\ (1-\delta_{\alpha 0}\delta_{k0})((k-1)\,3^{l+1}+1) < \beta \leqslant k\,3^{l+1}+3^l\}) \\ \leqslant G(A)\cdot G(B) \vee \delta_{k0} G(A) \\ \leqslant G(\{\alpha 3^{l+2}+\beta : 0 \leqslant \alpha < 3^{r-l-2}+\delta_{k0}, \\ (1-\delta_{\alpha 0}\delta_{k0})((k-1)\,3^{l+1}-3^l+2-2M) < \beta \leqslant k\,3^{l+1}+3^l\}). \end{aligned} \tag{14}$$

Note that since $l \geqslant \log_3 M$, $M \leqslant 3^l$, so,

$$(1-\delta_{\alpha 0}\delta_{k0})((k-1)\,3^{l+1}+1) > (k-1)\,3^{l+1}-3^l+2-2M > (k-2)\,3^{l+1}. \tag{15}$$

We intersect (14) with $G(\langle x^{r-l-2} k x^{l+1}\rangle)$, and want to prove that both sides became equal using (3). So we need to prove that the following sets are equal:

$$\begin{aligned} \{\alpha 3^{l+2}+\beta : 0 \leqslant \alpha < 3^{r-l-2}+\delta_{k0}, \\ (1-\delta_{\alpha 0}\delta_{k0})((k-1)\,3^{l+1}+1) < \beta \leqslant k\,3^{l+1}+3^l\} \\ \cap \{\alpha 3^{l+2}+\beta : 0 \leqslant \alpha < 3^{r-l-2}, k\,3^{l+1} < \beta \leqslant k\,3^{l+1}+3^{l+1}\} \end{aligned} \tag{16}$$

and

$$\begin{aligned} \{\alpha 3^{l+2}+\beta : 0 \leqslant \alpha < 3^{r-l-2}+\delta_{k0}, \\ (1-\delta_{\alpha 0}\delta_{k0})((k-1)\,3^{l+1}-3^l+2-2M) < \beta \leqslant k\,3^{l+1}+3^l\} \\ \cap \{\alpha 3^{l+2}+\beta : 0 \leqslant \alpha < 3^{r-l-2}, k\,3^{l+1} < \beta \leqslant k\,3^{l+1}+3^{l+1}\}. \end{aligned} \tag{17}$$

If some integer a is in (17) then

$$a = \alpha 3^{l+2}+\beta = \alpha' 3^{l+2}+\beta', \tag{18}$$

where

$$0 \leqslant \alpha < 3^{r-l-2}+\delta_{k0},\ (1-\delta_{\alpha 0}\delta_{k0})((k-1)\,3^{l+1}-3^l+2-2M) < \beta \leqslant k\,3^{l+1}+3^l$$

and

$$0 \leqslant \alpha' < 3^{r-l-2}, \qquad k\,3^{l+1} < \beta' \leqslant k\,3^{l+1}+3^{l+1}.$$

Using (15) we get

$$|\beta-\beta'| < (k\,3^{l+1}+3^{l+1})-(k-2)\,3^{l+1} \leqslant 2\cdot 3^{l+1}+3^{l+1} = 3^{l+2}. \tag{19}$$

So $\beta = \beta'$ (see (18)) and thus $\alpha = \alpha'$, hence (by choosing the more restrictive bounds) the set in (17) is

$$\{\alpha 3^{l+2} + \beta : 0 \leqslant \alpha < 3^{r-l-2},\ k3^{l+1} < \beta \leqslant k3^{l+1} + 3^l\} = C.$$

Similarly, the set in (16) is also C. Note that we have an equal term on both sides and thus equality:

$$G(C) = G(\langle \mathrm{x}^{r-l-2} k \mathrm{x}^{l+1} \rangle) \wedge (G(A) \cdot G(B) \vee \delta_{k0} G(A)).$$

As for Substep 2c: Let A be as in (13) and let

$$B \stackrel{\text{def}}{=} \langle \mathrm{x}^{r-l-2} k 0 \mathrm{x}^{l+1} \rangle = \{\alpha 3^{l+2} + \beta : 0 \leqslant \alpha < 3^{r-l-2},\ k3^{l+1} < \beta \leqslant k3^{l+1} + 3^l\}.$$

Observe that

$$A + B = \{\alpha 3^{l+2} + \beta : 0 \leqslant \alpha \leqslant 3^{r-l-2},\ k3^{l+1} + 1 < \beta \leqslant k3^{l+1} + 2 \cdot 3^l\}$$

and

$$\begin{aligned} B + [2 - 2M - \max(A), \max(A)] \\ = \{\alpha 3^{l+2} + \beta : 0 \leqslant \alpha < 3^{r-l-2},\ k3^{l+1} - 3^l + 2 - 2M < \beta \leqslant k3^{l+1} + 2 \cdot 3^l\}. \end{aligned}$$

Using Corollary 14 and (2) we get

$$\begin{aligned} &G(\{\alpha 3^{l+2} + \beta : 0 \leqslant \alpha < 3^{r-l-2},\ k3^{l+1} + 1 < \beta \leqslant k3^{l+1} + 2 \cdot 3^l\}) \\ &\quad \leqslant G(A) \cdot G(B) \\ &\quad \leqslant G(\{\alpha 3^{l+2} + \beta : 0 \leqslant \alpha < 3^{r-l-2},\ k3^{l+1} - 3^l + 2 - 2M < \beta \leqslant k3^{l+1} + 2 \cdot 3^l\}). \end{aligned} \tag{20}$$

We apply logical and to (20) with

$$\neg(G(\langle \mathrm{x}^{r-l-2} k 0 \mathrm{x}^l \rangle) \vee G(\langle \mathrm{x}^{r-l-2} (k \ominus 1)\, \mathrm{x}^{l+1} \rangle)). \tag{21}$$

Clearly,

$$\langle \mathrm{x}^{r-l-2} 0 \mathrm{x}^{l+1} \rangle \cup \langle \mathrm{x}^{r-l-2} 1 \mathrm{x}^{l+1} \rangle \cup \langle \mathrm{x}^{r-l-2} 2 \mathrm{x}^{l+1} \rangle$$

is the all one matrix, the union is disjoint, and

$$\langle \mathrm{x}^{r-l-2} k \mathrm{x}^{l+1} \rangle = \langle \mathrm{x}^{r-l-2} k 0 \mathrm{x}^l \rangle \cup \langle \mathrm{x}^{r-l-2} k 1 \mathrm{x}^l \rangle \cup \langle \mathrm{x}^{r-l-2} k 2 \mathrm{x}^l \rangle.$$

So (21) is actually

$$G(\langle \mathrm{x}^{r-l-2} k 1 \mathrm{x}^l \rangle) \vee G(\langle \mathrm{x}^{r-l-2} k 2 \mathrm{x}^l \rangle) \vee G(\langle \mathrm{x}^{r-l-2} (k \oplus 1)\, \mathrm{x}^{l+1} \rangle),$$

which is

$$G([0, 3^r] \cap \{\alpha 3^{l+2} + \beta : -1 \leqslant \alpha < 3^{r-l-2}, k3^{l+1} + 3^l < \beta \leqslant (k+2)\, 3^{l+1}\}).$$

Intersecting the sets (as in (16) and (17)) and using a similar argument on the β's, we get again that both sides are equal, so we get

$$\begin{aligned}
&\neg(G(\langle \mathrm{x}^{r-l-2} k 0 \mathrm{x}^l \rangle) \vee G(\langle \mathrm{x}^{r-l-2}(k \ominus 1)\, \mathrm{x}^{l+1} \rangle)) \wedge (G(A) \cdot G(B)) \\
&\quad = G(\{\alpha 3^{l+2} + \beta : 0 \leqslant \alpha < 3^{r-l-2}, k3^{l+1} + 3^l < \beta \leqslant k3^{l+1} + 2 \cdot 3^l\}) \\
&\quad = G(\langle \mathrm{x}^{r-l-2} k 1 \mathrm{x}^l \rangle). \quad \blacksquare
\end{aligned}$$

In Step 3 we combine all the matrices computed previously into the result. In the unweighted case it is staightforward, since we have the right matrix for each trit. In the more general weighted case, there is a problem with the least significant trits, so we prove (in Lemma 19) that the extra term in Step 3 takes care of these trits.

LEMMA 18. *Step* 3 *is correct.*

Proof. If $d_{ij}^* - 1$, as a ternary number, is $a_{r-1} \cdots a_1 a_0$, then clearly, $[D^* \in \langle \mathrm{x}^{r-l-1} k \mathrm{x}^l \rangle]_{ij} = 1$ if and only if $k = a_l$. This proves the uniform case ($M = 1$), since Step 3 computes d_{ij}^* trit by trit. In the general case, we do not have $[D^* \in \langle \mathrm{x}^{r-l-1} k \mathrm{x}^l \rangle]$ for $l < \log_3 M$. The next lemma shows that these missing terms are computed by the additional term in the new Step 3. $\blacksquare$

LEMMA 19.

$$\begin{aligned}
&\sum_{l=0}^{\log_3 M - 1} 3^l \sum_{k=0}^{2} k[D^* \in \langle \mathrm{x}^{r-l-1} k \mathrm{x}^l \rangle] \\
&\quad = (D^* - 1) (\mathrm{mod}\ M) \\
&\quad = \sum_{m=1}^{M-1} \sum_{k=0}^{2} ([D^* \in (\langle \mathrm{x}^{r-\log_3 M - 1} k \mathrm{x}^{\log_3 M} \rangle + m)] \\
&\qquad \wedge [D^* \in \langle \mathrm{x}^{r-\log_3 M - 1} k \mathrm{x}^{\log_3 M} \rangle]).
\end{aligned}$$

Proof. Consider the ij entry of the matrix identity we want to prove. By Lemma 18, the left hand side is the contribution of the $\log_3 M$ least significant trits of $d_{ij}^* - 1$; i.e., it is equal to $(d_{ij}^* - 1) \pmod M$. We now prove that the right hand side is the same. Observe that

$$\langle \mathrm{x}^{r-\log_3 M - 1} k \mathrm{x}^{\log_3 M} \rangle = \{3\alpha M + kM + \beta : 0 \leqslant \alpha < 3^{r-\log_3 M - 1}, 0 < \beta \leqslant M\},$$

so

$$\begin{aligned}
&\langle \mathrm{x}^{r-\log_3 M - 1} k \mathrm{x}^{\log_3 M} \rangle + m \\
&\quad = \{3\alpha M + kM + \beta : 0 \leqslant \alpha < 3^{r-\log_3 M - 1}, m < \beta \leqslant M + m\}
\end{aligned}$$

Using the simple fact

$$[D^* \in A] \wedge [D^* \in B] = [D^* \in (A \cap B)],$$

we get that

$$\begin{aligned}
&[D^* \in \langle \mathrm{x}^{r-\log_3 M-1} k \mathrm{x}^{\log_3 M} \rangle] \wedge [D^* \in (\langle \mathrm{x}^{r-\log_3 M-1} k \mathrm{x}^{\log_3 M} \rangle + m)] \\
&\quad = [D^* \in (\langle \mathrm{x}^{r-\log_3 M-1} k \mathrm{x}^{\log_3 M} \rangle \cap (\langle \mathrm{x}^{r-\log_3 M-1} k \mathrm{x}^{\log_3 M} \rangle + m))] \\
&\quad = [D^* \in (A_m^k + 1)],
\end{aligned}$$

where

$$A_m^k \stackrel{\text{def}}{=} \{\text{“}a_{r-1} \cdots a_1 a_0\text{”} : a_{\log_3 M} = k \text{ and } \text{“}a_{\log_3 M-1} \cdots a_1 a_0\text{”} \geqslant m\}. \tag{22}$$

We use the quotes in the equation above to emphasis the fact that we mean ternary numbers and not patterns. The summation over k removes the restriction $a_{\log_3 M} = k$ from (22); the summation on m gives a weight m to numbers whose $\log_3 M$ least significant trits has value m: $d_{ij}^* \in (A_m^k + 1)$ only for one k and only if $(d_{ij}^* - 1) \pmod M \geqslant m$. Therefore, summing on m, we get

$$\sum_{m=1}^{M-1} \sum_{k=0}^{2} [D^* \in (A_m^k + 1)] = (D^* - 1) \pmod M. \quad \blacksquare$$

We have completed the correctness proof of the algorithm, and we are left with only its time complexity.

LEMMA 20. *After computing $G(\langle 0 \rangle)$ and $G([1-M, 0])$, the algorithm described above performs $O(\log(Mn))$ G-multiplications and $O(n^2 \log(Mn))$ other operations.*

Proof. Each step involves $\log(Mn)$ computations which consist of a constant number of G-multiplications and a constant number of other matrix operations which are computed in $O(n^2)$ time each. $\blacksquare$

THEOREM 3. *The undirected $APSD(n, M)$ problem can be solved in $O(M^2 n^\omega \log(Mn))$ time.*

Proof. $G(\langle 0 \rangle)$ and $G([1-M, 0])$ can be computed in $O(M^2 n^\omega)$ time (Lemma 9) and every G-multiplication can be computed in $O(M^2 n^\omega)$ time by (6) and (7). Lemma 20 completes the proof. $\blacksquare$

4. FASTER ALGORITHMS FOR LONG EDGES

Previously we used Lemma 1 to transform the weighted input graph into an unweighted one. This would only be used in the directed case. In this section we show how we can solve the problem for weighted graphs directly and more efficiently, by slightly changing the algorithms. The main idea is to pack together several matrix multiplications into larger matrices.

The main packing idea is that our algorithm is computing many sums (logical or) of Boolean matrix multiplications. We can rewrite these sums to have structured sub-expressions. These sub-expressions can be computed simultaneously with a single Boolean matrix multiplication. A single multiplication of two $kn \times kn$ Boolean matrices results in k^2 $n \times n$ Boolean matrices. Each one of them is a sum of k $n \times n$ Boolean matrix multiplications. We reduce the time complexity since $\omega < 3$, so $(kn)^\omega < k^2 kn^\omega$. The problem is that the k^2 computations are dependent. One can save computation this way, by making sure that all k^2 matrices can be fitted into a single $kn \times kn$ Boolean matrix multiplication, and that the result can be used.

In Subsection 4.1 we handle the nonnegative directed case and in Subsection 4.2 we modify the undirected algorithm of Section 3.

4.1. *The Nonnegative Directed Case*

In this section we generalize the algorithm of Section 2 to nonnegative M-bounded edge lengths. First (Lemma 21) we show how to take care of zero distances. Then we show another way to compute $[D^* \leqslant a]$, and prove its correctness (Lemma 22). This new way enables us to pack several Boolean matrix multiplications together (lemma 23 and Corollary 24). Lemmas 25 and 26 prove that we can efficiently compute the matrices needed to start up this process, and Theorem 4 follows.

The main obstacle to the packing is the fact that the way we computed the matrix $[D^* \leqslant a]$ used the matrix $[D^* \leqslant a-1]$, so we could not pack any computation. Lemma 22 allows us to break this dependency a little. But still we need to use recursive computation shown in Fig. 6 to be able to pack these computations. The main idea behind this algorithm is that we do not fully compute the $[D^* \leqslant a]$ matrices, but rather prepare partial sums, which are both easier to pack and useful for further computations.

As we did in Section 2, we do not allow empty paths in this subsection. However this alone does not take care of all the possible zero length edges so we show how to remove the zero length edges. Removing the zero-length edges in the directed case is similar to the undirected case (Lemma 6), but a little more complicated. We prove it in Lemma 21. Lemma 8 that deals with the undirected case, can be transformed into a similar lemma for the directed case as well (Lemma 22).

LEMMA 21. *The nonnegative APSD(n, M) problem can be reduced to the positive APSD(n, M) problem, using $O(Mn^\omega)$ extra work.*

Proof. First, we compute the transitive closure of the zero length edges, giving all the zero length shortest distances. For APSD$(n, 1)$, using two Boolean matrix multiplications, we compute for all i and j whether there is a path in the original graph from v_i to v_j with exactly one nonzero length edge. Next, we construct a new graph whose edges correspond to such paths and have length 1. Then, we solve APSD$(n, 1)$ for the new graph (with no zero length edges) giving D^*, except for the zero entries, which have already been computed. The case of APSD(n, M) is similar; we compute up to $2M$ Boolean matrix multiplications: two for every edge length in G.

More formally, let

$$D' = \min_{1 \leqslant m \leqslant M} \{m[D^* = 0][D = m][D^* = 0]\}.$$

$[D^* = 0] = [D = 0]^*$ and therefore can be computed using one Boolean matrix transitive closure, and D' can be computed using $2M$ additional Boolean matrix multiplications. Furthermore, $D' > 0$ and

$$D^*_{ij} = \begin{cases} 0 & D^*_{ij} = 0 \\ D'^*_{ij} & \text{otherwise.} \end{cases} \blacksquare$$

Therefore we assume from now on that $d_{ij} > 0$. The major time saving in this subsection comes from the fact that we can pack many matrix multiplications together (Lemma 25). The problem is that in order to use this packing we must first compute about M matrices in another way. In this first step, to pack several matrix multiplications together we need to make sure that different computations would have common components. Therefore we choose to compute $[D^* \leqslant a]$ in a very specific way. First we prove that it is correct (Lemma 22), next we show how to use the special structure to efficiently pack some computations (Lemma 23 and Corollary 24).

There are several different ways to compute $[D^* \leqslant a]$. The following lemma enables us to choose various ways to compute it so that we can pack several computations.

LEMMA 22. *If $l_1 \geqslant 1$, $l_2 \geqslant M$ then*

$$[D^* \leqslant (l_1 + l_2)] = [D \leqslant (l_1 + l_2)] \vee \bigvee_{m=0}^{M-1} [D^* \leqslant (l_1 + m)] \cdot [D^* \leqslant (l_2 - m)].$$

Proof. If the ijth coordinates of the right hand side is one then either $d_{ij} \leqslant l_1 + l_2$, in which case $d^*_{ij} \leqslant d_{ij} \leqslant l_1 + l_2$; or there exists $0 \leqslant m \leqslant M-1$ and a vertex v_k such that $d^*_{ik} \leqslant l_1 + m$ and $d^*_{kj} \leqslant l_2 - m$ hence $d^*_{ij} \leqslant d^*_{ik} + d^*_{kj} \leqslant l_1 + l_2$. In any case $[D^* \leqslant (l_1 + l_2)]_{ij} = 1$.

Suppose, on the other hand, that $d^*_{ij} \leqslant l_1 + l_2$. Examine a shortest path P from v_i to v_j. If P is a single edge, $d_{ij} = d^*_{ij} \leqslant l_1 + l_2$ and therefore the ijth coordinates of the right hand side is one. Otherwise we consider two cases and show in each one of them how to choose v_k and m such that

a: $0 \leqslant m \leqslant M-1$,

b: $d^*_{ik} \leqslant l_1 + m$, and

c: $d^*_{kj} \leqslant l_2 - m$.

1. $d^*_{ij} \leqslant l_2$: Let v_k be the second vertex on P. It cannot be v_j since P is not a single edge. Choose $m = d^*_{ik} - 1$. From Fact 3 it follows that $d^*_{ik} = d_{ik}$.

a: $1 \leqslant d_{ik} \leqslant M$ so $0 \leqslant m \leqslant M-1$.

b: $d^*_{ik} = d_{ik} = m + 1 \leqslant l_1 + m$.

c: $d^*_{kj} = d^*_{ij} - d^*_{ik} \leqslant l_2 - d^*_{ik} < l_2 - m$.

2. $d^*_{ij} > l_2$: Let v_k be the first vertex on P which satisfies $d^*_{kj} \leqslant l_2$; there must be such vertex $\neq v_j$ since $d^*_{ij} > l_2 \geqslant M$. Let $m = l_2 - d^*_{kj}$.

a: $d^*_{kj} \leqslant l_2$, so $0 \leqslant m$. v_k is the *first* vertex such that $d^*_{kj} \leqslant l_2$, $d^*_{ij} > l_2$, so it is not v_i. $m(= l_2 - d^*_{kj}) < M$ because otherwise the vertex $v_{k'}$ just before v_k on P would satisfy $d^*_{k'j} \leqslant l_2$ too, so $0 \leqslant m \leqslant M-1$.

b: $d^*_{ik} = d^*_{ij} - d^*_{kj} \leqslant (l_1 + l_2) - d^*_{kj} = l_1 + m$.

c: $d^*_{kj} = l_2 - m$. ∎

Lemma 8 assumes that empty paths are allowed. In this subsection we do not allow empty paths, so we can prove a slightly different condition: for all r,

$$[D^* \leqslant r] = [D \leqslant r] \vee \bigvee_{\delta=1}^{r-1} [D = \delta] \cdot [D^* \leqslant (r-\delta)]. \tag{23}$$

The $[D \leqslant r]$ part is for the single-edge-long paths which were covered by the $[D^* \leqslant (r-\delta)]$ part in Lemma 8.

Both $[D \leqslant r]$ and $[D = \delta]$ can be easily computed from the input matrix D. The obvious way to compute the matrices $\{[D^* \leqslant r]\}_{r=1}^{M}$ is to compute them one after the other. This requires r Boolean matrix multiplications for $[D^* \leqslant r]$ and therefore $\Theta(M^2)$ $n \times n$ Boolean matrix multiplications which takes $\Theta(M^2 n^\omega)$ time. We will save time by packing several Boolean matrix multiplications together. We use two methods to speed up the computation. Both compute several $[D^* \leqslant r]$ matrices together. The first (Lemma 23) will be applied to values of r smaller than M, and the second (Lemma 25) will be applied to larger values of r.

LEMMA 23. *The matrices*

$$\left\{ \bigvee_{\delta=1}^{m} [D = \delta] \cdot [D^* \leqslant (m_0 + m - \delta)] \right\}_{m=1}^{r}$$

can be computed, given the matrices

$$\left\{ \bigvee_{\delta=m-m_0+1}^{m-1} [D = \delta] \cdot [D^* \leqslant (m - \delta)] \right\}_{m=2}^{m_0+r},$$

in $C\sqrt{r}\, B(n\lceil\sqrt{r}\rceil)$ time, where $B(n)$ is the time needed for $n \times n$ Boolean matrix multiplication, and C is some constant defined later.

Proof. Assume without loss of generality that r is a power of 2. We use recursion to compute the matrices, and use induction on r in the proof. If $r = 1$, we compute the single Boolean matrix multiplication needed. To do this we need $[D^* \leqslant m_0]$. If $m_0 = 1$ then $[D^* \leqslant 1] = [D \leqslant 1]$, otherwise it can be computed from the given part for $m = m_0$, using (23). This computation can be done in $B(n)$ time. If $r > 1$, we compute the matrices in three steps. Fig. 6 graphically shows the three steps. Each point in the figure represents an $n \times n$ Boolean matrix. We use the notation $\binom{x}{y}$ for the expression $[D = x] \cdot [D^* \leqslant y]$. The circled numbers mark their region. Region 0 is the part which is assumed to be given; we assume that we know

the logical or of the rows of region 0. Individual matrices are not given and will not be computed. In the ith step, $1 \leqslant i \leqslant 3$, we compute the logical or of the matrices which are on the same row in the ith region. Note that every point outside (to the left or to the right of) the triangle, for example the dashed triangle, contains only zero matrices because it contains matrices like $[D = x] \cdot [D^* \leqslant y]$ where either x or y are nonpositive. Regions 1, 2, and 3 are computed in this order. Using (23) $[D^* \leqslant m]$ can be computed from the logical or of the $m-1$th row of the triangle in $O(n^2)$ time.

1. Compute $\{\bigvee_{\delta=1}^{m} [D = \delta] \cdot [D^* \leqslant (m_0 + m - \delta)]\}_{m=1}^{r/2}$ recursively, with $m_0' = m_0$ and $r' = r/2$. It is easy to verify that the assumptions hold (the given part needed for applying the recursion is indeed part of the given part we have) and therefore we can use the induction hypothesis. So the time complexity of this step is $C\sqrt{r/2}\, B(n\lceil\sqrt{r/2}\rceil)$.

2. Compute $\{\bigvee_{\delta=m-r/2+1}^{m} [D = \delta] \cdot [D^* \leqslant (m_0 + m - \delta)]\}_{m=r/2+1}^{r}$. In this computation we save time by packing multiplications together. Define two new matrices: an $r \times \lceil\sqrt{r/2}\rceil$ matrix X of $n \times n$ matrices defined as $X_{ik} = [D = i + k]$; and a $\lceil\sqrt{r/2}\rceil \times \lceil\sqrt{r/2}\rceil$ matrix Y of $n \times n$ matrices defined as

$$Y_{kj} = \begin{cases} [D^* \leqslant (m_0 + r/2 - k - (j-1)\lceil\sqrt{r/2}\rceil)], & k + (j-1)\lceil\sqrt{r/2}\rceil \leqslant r/2 \\ 0, & \text{otherwise.} \end{cases}$$

Any matrix multiplication of dimension $n_1 n_2 \times m$ by $m \times k$ can be computed by n_1 matrix multiplications of dimension $n_2 \times m$ by $m \times k$. So multiply X by Y as $\lceil\sqrt{r/2}\rceil$ multiplications of a square $\lceil\sqrt{r/2}\rceil n \times \lceil\sqrt{r/2}\rceil n$ Boolean matrix multiplication. The matrix X can be easily computed from D in time proportional to its size. To compute the matrix Y we need $[D^* \leqslant m']$ for $m_0 \leqslant m' \leqslant m_0 + r/2$ and from (23),

$$\begin{aligned} [D^* \leqslant (m_0 + m)] = {} & [D \leqslant m_0 + m] \\ & \vee \bigvee_{\delta=1}^{m} [D = \delta] \cdot [D^* \leqslant (m_0 + m - \delta)] \\ & \vee \bigvee_{\delta=m+1}^{m_0+m-1} [D = \delta] \cdot [D^* \leqslant (m_0 + m - \delta)]. \end{aligned}$$

The first term, $[D \leqslant m_0 + m]$, can be computed from D in $O(n^2)$ time. The second term, $\bigvee_{\delta=1}^{m} [D = \delta] \cdot [D^* \leqslant (m_0 + m - \delta)]$, was computed in Step 1. The last term, $\bigvee_{\delta=m+1}^{m_0+m-1} [D = \delta] \cdot [D^* \leqslant (m_0 + m - \delta)]$, is given (region 0). From Z, the multiplication of X and Y, we derive the logical or of the matrices in the rows of region 2: Let $i(m, j) =^{\text{def}} m - r/2 + (j-1)\lceil\sqrt{r/2}\rceil$.

$$\begin{aligned} & \bigvee_{\delta=m-r/2+1}^{m} [D = \delta] \cdot [D^* \leqslant (m_0 + m - \delta)] \\ & \quad = \bigvee_{j=1}^{\lceil\sqrt{r/2}\rceil} \left(\bigvee_{k=1}^{\min\{\lceil\sqrt{r/2}\rceil,\, r/2-(j-1)\lceil\sqrt{r/2}\rceil\}} [D = i(m,j) + k] \right. \\ & \qquad \left. \cdot [D^* \leqslant (m_0 + m - i(m,j) - k)] \right), \end{aligned}$$

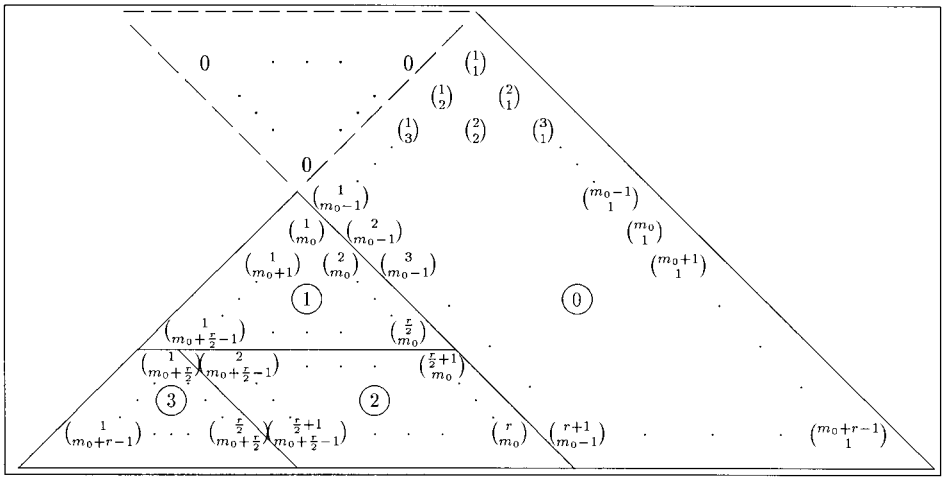


FIG. 6. Efficiently computing short distances for nonuniform edge lengths.

by the definition of X and Y,

$$\bigvee_{\delta=m-r/2+1}^{m} [D=\delta]\cdot[D^*\leqslant(m_0+m-\delta)]$$
$$=\bigvee_{j=1}^{\lceil\sqrt{r/2}\rceil}\bigvee_{k=1}^{\lceil\sqrt{r/2}\rceil} X_{i(m,j)k}\,Y_{kj}=\bigvee_{j=1}^{\lceil\sqrt{r/2}\rceil} Z_{i(m,j)j}.$$

The time complexity of this step is therefore $r^{1.5}n^2$ for preparing X, rn^2 for preparing Y, $\lceil\sqrt{r/2}\rceil\, B(n\lceil\sqrt{r/2}\rceil)$ for computing $Z=X\cdot Y$, and $\lceil\sqrt{r/2}\rceil\, n^2$ for computing the desired output from Z.

3. As done in Step 1, we use induction again here. To do that we show how to build all the needed parts to use the induction hypothesis with smaller r. We combine the triangle of region 1 and the parallelogram of region 2 to enlarge the trapezoid of region 0 to the size needed to compute $\{\bigvee_{\delta=1}^{m-r/2}[D=\delta]\cdot[D^*\leqslant(m_0+m-\delta)]\}_{m=r/2+1}^{r}$ recursively. We choose $m_0'=m_0+r/2$ and $r'=r/2$ so

$$\left\{\bigvee_{\delta=m-m_0'+1}^{m-m_0}[D=\delta]\cdot[D^*\leqslant(m-\delta)]\right\}_{m=2}^{m_0'+r'}$$
$$=\left\{\bigvee_{\delta=m-m_0-r/2+1}^{m-m_0}[D=\delta]\cdot[D^*\leqslant(m-\delta)]\right.$$
$$\left.\vee\bigvee_{\delta=m-m_0+1}^{m-1}[D=\delta]\cdot[D^*\leqslant(m-\delta)]\right\}_{m=2}^{m_0+r}.$$

The second term is given (in region 0). The first term is the zero matrix for $2\leqslant m\leqslant m_0$; it is computed in region 1 for $m_0<m\leqslant m_0+r/2$ ($[D=\delta]$ for negative δ is zero); and it is computed in region 2 for $m_0+r/2<m\leqslant m_0+r$. The time of this step is $rn^2+C\sqrt{r/2}\,B(n\lceil\sqrt{r/2}\rceil)$.

The total time complexity of all the three steps is

$$C\sqrt{r/2}\,B(n\lceil\sqrt{r/2}\rceil)+r^{1.5}n^2+rn^2+\lceil\sqrt{r/2}\rceil\,B(n\lceil\sqrt{r/2}\rceil)+\lceil\sqrt{r/2}\rceil\,n^2 + rn^2+C\sqrt{r/2}\,B(n\lceil\sqrt{r/2}\rceil).$$

Using the fact that $B(n)\approx n^\omega$ with $\omega>2$, we can find large enough C such that the total time complexity will be $C\sqrt{r}\,B(\lceil\sqrt{r}\rceil n)$. ▮

COROLLARY 24. *$\{[D^*\leqslant\rho]\}_{\rho=1}^{M'}$ can be computed in $O(M'^{(\omega+1)/2}n^\omega)$ time.*

Proof. Use Lemma 23 with $m_0=1$ and $r=M'-1$. Note that the "given" part is empty since $m_0=1$. The matrices $\{[D^*\leqslant\rho]\}_{\rho=1}^{M'}$ can be computed from $\{\bigvee_{\delta=1}^{m}[D=\delta]\cdot[D^*\leqslant(m_0+m-\delta)]\}_{m=1}^{r}$ in $O(M'n^2)$ time using (23). This additional term is negligible. ▮

Now that we can prove how to save computation by packing, we only need to show how to efficiently compute the first matrices to start up the process.

LEMMA 25. *If $M'\geqslant M$ and $1\leqslant\alpha\leqslant M$ then the matrices $\{[D^*\leqslant r]\}_{r=M'+\alpha^2+1}^{M'+2\alpha^2}$ can be computed from the matrices $\{[D^*\leqslant r]\}_{r=1}^{M'+\alpha^2}$ in $O(M\alpha^{\omega-1}n^\omega)$ time.*

Proof. Let X be a $\alpha\times M$ matrix of $n\times n$ Boolean matrices where

$$X_{ij}=[D^*\leqslant(i\alpha+j)]$$

and Y be a $M\times\alpha$ matrix of $n\times n$ matrices, where

$$Y_{ij}=[D^*\leqslant(M'+\alpha^2-\alpha-i+j)].$$

Compute the Boolean matrix multiplication $Z=X\cdot Y$. Note that Z is actually an $\alpha\times\alpha$ matrix of $n\times n$ matrices, where

$$Z_{ij}=\bigvee_{m=0}^{M-1}[D^*\leqslant(i\alpha+m)]\cdot[D^*\leqslant(M'+\alpha^2-\alpha+j-m)].$$

Using Lemma 22 with $l_1=i\alpha\geqslant 1$ and $l_2=M'+\alpha^2-\alpha+j\geqslant M$ we get

$$[D^*\leqslant(M'+\alpha^2+(i-1)\alpha+j)]=[D\leqslant(M'+\alpha^2+(i-1)\alpha+j)]\vee Z_{ij}.$$

The matrices X and Y can be computed from $\{[D^*\leqslant r]\}_{r=1}^{M'+\alpha^2}$ since (for X_{ij})

$$1\leqslant\alpha+1\leqslant i\alpha+j\leqslant M'+\alpha^2$$

and (for Y_{ij})

$$1\leqslant M'+\alpha^2-\alpha-M+1\leqslant M'+\alpha^2-\alpha-i+j\leqslant M'+\alpha^2-\alpha-1+\alpha<M'+\alpha^2.$$

The Boolean matrix multiplication which computes Z is of size $\alpha n\times Mn$ times $Mn\times\alpha n$. $\alpha\leqslant M$, so this Boolean matrix multiplication can be computed as M/α

square Boolean matrix multiplications of size $\alpha n \times \alpha n$; i.e., $O((M/\alpha)(\alpha n)^\omega)$, the desired time bound. As a result of the computation of Z above, we get $\{[D^* \leqslant (M' + \alpha^2 + (i-1)\alpha + j)]\}_{i,j=1}^{\alpha}$ which gives $[D^* \leqslant r]$ for $M' + \alpha^2 + 1 \leqslant r \leqslant M' + 2\alpha^2$. ▮

LEMMA 26. *Let M' be an integer such that $M \leqslant M' \leqslant M^2$. The matrices $\{[D^* \leqslant r]\}_{r=1}^{M'}$ and all the entries in D^* which are $\leqslant M'$, can be computed in $O(M(M')^{(\omega-1)/2} n^\omega)$ time.*

Proof. First we use Corollary 24 to compute $\{[D^* \leqslant r]\}_{r=1}^{M+6^2}$. Then we use Lemma 25 several times, each time computing $\{[D^* \leqslant r]\}_{r=1}^{M+2\alpha^2}$ (called the *output* matrices) from previously computed $[D^* \leqslant r]$'s (called the *input* matrices). We do it by choosing $M = M'$ and choosing the parameter α for each application such that the input matrices for the $(l+1)$st step are computed as output matrices from the lth step. We choose α to be $\lfloor 1.1^{l+19} \rfloor$ in the lth application of the lemma. Corollary 24 supplies the input for the first application since $\lfloor 1.1^{1+19} \rfloor = 6$. We apply Lemma 25 until $\alpha^2 \geqslant M'$. Thus in all applications $\alpha^2 < M' \leqslant M^2$ and thus $\alpha \leqslant M$ (as required by the lemma). To show that the applications are valid we show that the output of the lth application of Lemma 25 contains the input necessary for the $(l+1)$st application. The output is $\{[D^* \leqslant r]\}_{r=1}^{M+2\lfloor 1.1^{l+19} \rfloor^2}$ and

$$\begin{aligned} 2\lfloor 1.1^{l+19} \rfloor^2 &\geqslant 2(1.1^{l+19} - 1)^2 = 2 \cdot 1.1^{2(l+19)} - 4 \cdot 1.1^{l+19} + 2 \\ &\geqslant 1.1^{2(l+19)+2} + ((2 - 1.1^2)\, 1.1^{l+19} - 4)\, 1.1^{l+19}. \end{aligned}$$

For $l \geqslant 1$, $(2 - 1.1^2)\, 1.1^{l+19} > 4$, so

$$2\lfloor 1.1^{l+19} \rfloor^2 > 1.1^{2(l+19)+2} \geqslant \lfloor 1.1^{l+19+1} \rfloor^2.$$

So the output of the lth application indeed contains the input for the $l+1$th application. The output of the last application contains the desired matrices since we apply Lemma 25 until $\alpha^2 \geqslant M'$.

The time needed for this computation is

$$O\left(\sum_{l=1}^{\lceil \log_{1.1} \sqrt{M'} \rceil} M \lfloor 1.1^{l+19} \rfloor^{\omega-1} n^\omega \right) = O(M(M')^{(\omega-1)/2} n^\omega).$$

After computing the matrices $\{[D^* \leqslant r]\}_{r=1}^{M'}$, we can easily find all the shortest distances which are $\leqslant M'$ in $O(M'n^2)$ time, which is negligible compared to the other terms in the time bound. ▮

We now use the machinery above to speed up the algorithm of Section 2. It consists of two parts: the first part (Step 1) computes the entries of D^* which are smaller than L. The second part (Step 2) uses the separator trick to compute D^*. In our case, where $M > 1$, we use Lemma 26 to efficiently compute the small entries in D^* and modify the separator trick to work with long edges by choosing S_i to be M consecutive layers instead of one. Choose S_i in the algorithm of Section 2 as the set of vertices whose distance from v_i is in the interval $[s_i, s_i + M - 1]$ for some integer s_i in $[f(k)/2, f(k)]$. The following computation bounds $|S_i|$: There are

$f(k)-f(k)/2-M+2$ different consecutive segments of length M in the interval $[f(k)/2, f(k)]$. Counting their vertices with multiplicity gives no more than M times each one of the n vertices. Therefore, by the pigeon hole principle, there exists s_i such that $|S_i| \leqslant Mn/(f(k)-f(k)/2-M+2)$. Assuming that $f(k) \geqslant L \geqslant 4M$, we get that $f(k)-f(k)/2-M+2 \geqslant f(k)/4$. Therefore the total time of the second step is $O(Mn^3/L)$ (instead of $O(n^3/L)$).

If $n^{(3-\omega)/(\omega+1)} \leqslant M < n^{2(3-\omega)/(\omega+1)}$, choose $L = 4n^{2(3-\omega)/(\omega+1)}$. $4M < L \leqslant M^2$, so we can use Lemma 26 with $M' = L$ to find all entries in D^* which are $\leqslant L$ and the total time for the algorithm will be

$$O(ML^{(\omega-1)/2}n^\omega + Mn^3/L) = O(Mn^{(5\omega-3)/(\omega+1)}).$$

If, on the other hand, $1 \leqslant M < n^{(3-\omega)/(\omega+1)}$, choose $L = 4(Mn)^{(3-\omega)/2}$. In this case, $4M \leqslant L$, so we use Lemma 26. Since $M^2 \leqslant L$, we use Lemma 26 only for computing $\{[D^* \leqslant r]\}_{r=1}^{M^2}$. We compute the other matrices needed for computing the entries in D^* which are smaller than L, namely $\{[D^* \leqslant r]\}_{r=M^2+1}^{L}$, by repeatedly applying Lemma 25 with $M' = (l-1)M^2$, $\alpha = M$ and l going from 1 to $\lceil L/M^2 \rceil$. The time needed for computing all these entries is

$$O(M(M^2)^{(\omega-1)/2}n^\omega + (L/M^2)(Mn)^\omega) = O(LM^{\omega-2}n^\omega)$$

and the total time in this case is

$$O(LM^{\omega-2}n^\omega + Mn^3/L) = O(M^{(\omega-1)/2}n^{(3+\omega)/2}).$$

So we have the following result.

THEOREM 4. *The APSD(n, M) problem on graphs with positive length edges can be solved in $O(T(n, M))$ time, where*

$$T(n, M) = \begin{cases} M^{(\omega-1)/2}n^{(3+\omega)/2}, & 1 \leqslant M \leqslant n^{(3-\omega)/(\omega+1)} \\ Mn^{(5\omega-3)/(\omega+1)}, & n^{(3-\omega)/(\omega+1)} \leqslant M \leqslant n^{2(3-\omega)/(\omega+1)}. \end{cases}$$

COROLLARY 27. *The* APSD(n, M) *problem on graphs with* nonnegative *length edges, can be solved in $O(T(n, M))$ time, where $T(n, M)$ is defined as in Theorem* 5.

Proof. Use Lemma 21 with Theorem 4.

$$\frac{5\omega-3}{\omega+1} > \frac{5\omega-3-(\omega-1)(3-\omega)}{\omega+1} = \omega.$$

So, $Mn^{(5\omega-3)/(\omega+1)} > Mn^\omega$. If $M \leqslant n^{(3-\omega)/(\omega+1)}$ then $n \geqslant M^{(\omega+1)/(3-\omega)}$, so

$$M^{(\omega-1)/2}n^{(3+\omega)/2} \geqslant M^{(\omega-1)/2}n^\omega(M^{(\omega+1)/(3-\omega)})^{(3-\omega)/2} = M^\omega n^\omega > Mn^\omega.$$

Hence, in any case, the $O(Mn^\omega)$ term is negligible. ∎

Substituting $w = 2.376$ yields (all numbers are rounded to three digits after the decimal point)

$$T(n, M) = \begin{cases} M^{0.688} n^{2.688}, & 1 \leqslant M \leqslant n^{0.185} \\ M n^{2.630}, & n^{0.185} \leqslant M \leqslant n^{0.370}. \end{cases}$$

Note that for $M \leqslant n^{2(3-\omega)/(\omega+1)} < n^{0.370}$ the algorithm is subcubic.

4.2. *The Undirected Case*

In this subsection we modify the algorithm of Subsection 3.2. As we did there, we allow empty paths. Examine Theorem 3. We replace the computation of $G(\langle 0 \rangle)$ (Lemma 9) with the faster way we developed for the directed case (any algorithm for the directed case can be applied to the undirected case by transforming each edge into two antiparallel ones). We use Corollary 24 with $M' = 2M$. A G-multiplication is actually a sequence of $2M-1$ matrices. Each one of them is a sum (logical or) of M $n \times n$ matrices which are a product of two $n \times n$ Boolean matrices. The naive way to compute a G-multiplication is to compute all these matrices separately. The G-multiplications are performed faster by packing some of the Boolean matrix multiplications in larger matrices. We give the details in Lemma 28. A short example is given below. In this example we choose $M = 100$ (even though M should be a power of 9). We do this to make the numbers simpler. We build the matrix X which is a $\sqrt{M} \times \sqrt{M}$ matrix of $n \times n$ Boolean matrices. In every entry there is a matrix of the type $[D^* \in (A + x)]$ for some integer x. We give these x's:

$$X = \begin{pmatrix} 0 & 1 & \cdots & 9 \\ 10 & 11 & \cdots & 19 \\ \vdots & \vdots & \ddots & \vdots \\ 90 & 91 & \cdots & 99 \end{pmatrix}.$$

Similarly we define a matrix Y which is a $\sqrt{M} \times (2M - \sqrt{M})$ matrix of $n \times n$ Boolean matrices $[D^* \in (B + y)]$ for some integer y. We give these y's:

$$Y = \begin{pmatrix} 99 & 98 & \cdots & 9 & \cdots & -90 \\ 98 & 97 & \cdots & 8 & \cdots & -91 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 90 & 89 & \cdots & 0 & \cdots & -99 \end{pmatrix}.$$

Then multiply X and Y and get Z, which is given below. The logical or of the framed entries gives one of the matrices of the G-multiplication (for $m = M - 1$). The rest are computed similarly, using the other parallel diagonals. Z is

$$\begin{pmatrix} \boxed{(0 \cdot 99 + \cdots + 9 \cdot 90)} & \cdots & (0 \cdot 89 + \cdots + 9 \cdot 80) & \cdots & (0 \cdot 9 + \cdots + 9 \cdot 0) & \cdots \\ (10 \cdot 99 + \cdots + 19 \cdot 90) & \cdots & \boxed{(10 \cdot 89 + \cdots + 19 \cdot 80)} & \cdots & (10 \cdot 9 + \cdots + 19 \cdot 0) & \cdots \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\ (90 \cdot 99 + \cdots + 99 \cdot 90) & \cdots & (90 \cdot 89 + \cdots + 99 \cdot 80) & \cdots & \boxed{(10 \cdot 89 + \cdots + 19 \cdot 80)} & \cdots \end{pmatrix}.$$

LEMMA 28. *A G-multiplication can be computed in* $O(M^{(\omega+1)/2}n^{\omega})$ *time.*

Proof. Without loss of generality, M is a square of an integer. (Assume M to be a power of 9 instead of a power of 3.) Let X be a $\sqrt{M} \times \sqrt{M}$ matrix of $n \times n$ Boolean matrices defined as

$$X_{ik} = [D^* \in (A + (i-1)\sqrt{M} + k - 1)]$$

and let Y be a $\sqrt{M} \times (2M - \sqrt{M})$ matrix of $n \times n$ Boolean matrices defined as

$$Y_{kj} = [D^* \in (B + M + 1 - k - j)].$$

Note that X can be computed from $G(A)$ since $X_{ik} = G(A)^{(i-1)\sqrt{M}+k-1}$ and for $1 \leqslant i, k \leqslant \sqrt{M}$,

$$1 - M \leqslant 0 \leqslant (i-1)\sqrt{M} + k - 1 \leqslant M - 1.$$

Y can be computed from $G(B)$ since $Y_{kj} = G(B)^{M+1-k-j}$ and for $1 \leqslant k \leqslant \sqrt{M}$, $1 \leqslant j \leqslant 2M - \sqrt{M}$,

$$1 - M \leqslant M + 1 - k - j \leqslant M - 1.$$

Compute the Boolean matrix multiplication $Z =^{\text{def}} X \cdot Y$. This multiplication is done as $2\sqrt{M} - 1$ square Boolean matrix multiplications which takes $O(\sqrt{M}(\sqrt{M}n)^{\omega}) = O(M^{(\omega+1)/2}n^{\omega})$ time. Let $j(i, m) =^{\text{def}} (i-1)\sqrt{M} + M - m$, for $0 \leqslant m \leqslant M - 1$. It satisfies $1 \leqslant j(i, m) \leqslant 2M - \sqrt{M}$, so we can compute

$$\begin{aligned}
\bigvee_{i=1}^{\sqrt{M}} Z_{ij(i,m)} &= \bigvee_{i=1}^{\sqrt{M}} \bigvee_{k=1}^{\sqrt{M}} X_{ik} \cdot Y_{kj(i,m)} \\
&= \bigvee_{i=1}^{\sqrt{M}} \bigvee_{k=1}^{\sqrt{M}} [D^* \in (A + (i-1)\sqrt{M} + k - 1)] \\
&\quad \cdot [D^* \in (B + 1 - k - (i-1)\sqrt{M} + m)] \\
&= \bigvee_{\delta=0}^{M-1} [D^* \in (A + \delta)] \cdot [D^* \in (B + m - \delta)].
\end{aligned}$$

Thus we can compute from Z the positive half of $G(A) \cdot G(B)$, namely $(G(A) \cdot G(B))^m$, for $0 \leqslant m \leqslant M - 1$ as

$$(G(A) \cdot G(B))^m = \bigvee_{i=1}^{\sqrt{M}} Z_{ij(i,m)}.$$

As for the negative half, we use the same Y with a different X': define

$$X'_{ik} = [D^* \in (A + (i-1)\sqrt{M} + k - M)]$$

Note that X' can be computed from $G(A)$ since $X'_{ik} = G(A)^{(i-1)\sqrt{M}+k-M}$ and

$$1 - M \leqslant (i-1)\sqrt{M} + k - M \leqslant 0 \leqslant M - 1.$$

Compute the Boolean matrix multiplication $Z' =^{\text{def}} X' \cdot Y$. This multiplication can be computed in $O(M^{(\omega+1)/2}n^{\omega})$ time. Let $j'(i, m') =^{\text{def}} (i-1)\sqrt{M} - m' + 1$, for $1 - M \leqslant m' \leqslant 0$. It satisfies $1 \leqslant j'(i, m') \leqslant 2M - \sqrt{M}$, so we can compute

$$\begin{aligned}
\bigvee_{i=1}^{\sqrt{M}} Z'_{ij'(i,m')} &= \bigvee_{i=1}^{\sqrt{M}} \bigvee_{k=1}^{\sqrt{M}} X'_{ik} \cdot Y_{kj'(i,m')} \\
&= \bigvee_{i=1}^{\sqrt{M}} \bigvee_{k=1}^{\sqrt{M}} [D^* \in (A + (i-1)\sqrt{M} + k - M)] \\
&\quad \cdot [D^* \in (B + M + 1 - k - ((i-1)\sqrt{M} - m + 1))] \\
&= \bigvee_{\delta'=1-M}^{0} [D^* \in (A + \delta')] \cdot [D^* \in (B + m - \delta')].
\end{aligned}$$

Thus we can compute from Z' the negative half of $G(A) \cdot G(B)$ $(1 - M \leqslant m' \leqslant 0)$ as

$$(G(A) \cdot G(B))^{m'} = \bigvee_{i=1}^{\sqrt{M}} Z'_{ij'(i,m')}.$$

The time complexity of preparing X, Y and X' and the time for computing $G(A) \cdot G(B)$ from Z and Z' is only $O(M^{1.5}n^2)$ since it can easily be done in linear time. ▮

We are now ready to prove an improvement of Theorem 3 of Subsection 3.2.

THEOREM 5. *In the undirected case, there is an algorithm for the* APSD(n, M) *problem which runs in* $O(M^{(\omega+1)/2}n^{\omega}\log(Mn))$ *time.*

Proof. Replace Lemma 9 with Corollary 24 and the naive $O(M^2n^{\omega})$ time estimation of a G-multiplication with Lemma 28. ▮

The largest M for which the algorithm is subcubic is $O(n^{2(3-\omega)/(\omega+1)}/\log(n))$ which for $\omega < 2.376$ is $O(n^{0.370})$.

5. OPEN PROBLEMS AND FURTHER RESEARCH DIRECTIONS

We improved only positive APSD(n, M). We can also handle zero edge lengths. What about negative edge lengths? Can we improve the $O((Mn)^{(3+\omega)/2}\log(Mn))$ time algorithm which comes from applying the transformation of Lemma 1 to the algorithm of [1]?

It seems that the directed cases of the problems are harder than the corresponding undirected cases. Are the directed problems intrinsically more difficult, or is it just because our techniques are not good enough? Can one prove any kind of

nontrivial lower bound? On the other hand, we can try to find some kind of transformation which will take a directed problem into an undirected one. One of the most interesting problems is to find an $\tilde{O}(n^\omega)$ time algorithm for the directed APSD$(n, 1)$ problem, even for the positive case. It seems that the separator trick is quite naive, but we could not improve it in the directed case. We believe that the directed problems *are* harder.

How far can we take M and still have a subcubic algorithm? One can attack this problem from two directions:

1. From below: Try to improve the algorithms of Section 4 so that they will work with larger M's. We always transform our problem into square Boolean matrices. Perhaps using other fast matrix multiplications methods (for example multiplying directly rectangular matrices) will improve our algorithm. We are not using the fact that our matrices are (usually) Boolean. This kind of research will take the fast matrix multiplication algorithms we are using out of their black box.

2. From above: Take a large M and reduce the problem into one with a smaller M. This direction might not improve any result initially. It may help later, if and when improvements are made in the first direction, or it can serve as a kind of lower bound by saying something like: "if one could solve for $M = n^2$ then one would be able to solve for $M = n^{\log(n)}$."

Some of our algorithms will benefit from the fact that the edge lengths are taken from a set of small cardinality. For example, in the equation

$$[D^* \leqslant x] = [D \leqslant x] \vee \bigvee_{m=1}^{M} [D = m] \cdot [D^* \leqslant (x - m)],$$

one can replace the M Boolean matrix multiplications with M', which is the cardinality of the set of possible values of edge lengths. So we can use transformations which enlarge the graph (add vertices) but make the edge lengths more degenerate. One such transformation can multiply the number of vertices by k while leaving only M/k^2 possibilities for edge lengths.

Can we use approximate solutions to improve our algorithms? Some work has been done on approximating the shortest distance:

1. In the undirected case one can partition the graph to several clusters and use this partition to get an approximation for the shortest distances ([2]). This gives us an $O(n^{2+1/k})$ time algorithm which approximates the shortest distances within a factor of $2k$. This seems to be the only thing we can do in $o(n^\omega)$ time. Can we use a similar method to obtain an exact solution?

2. In the positive case one can find all the pairs of vertices in a specific distance in logarithmic number of Boolean matrix multiplications. This easily leads to an $O(n^\omega \log_1 + c(n))$ time algorithm which approximates the shortest distances within a $(1 + c)$ multiplicative factor.

Can we use the methods developed here to solve other problems? For example matching or flow problems. Even a nonoptimal algorithm (an $O(n^{(\omega+3)/2})$ time, for example) would be interesting as a start.

ACKNOWLEDGMENT

The authors thank the anonymous referees for their comments which helped us make this article clearer.

Received September 3, 1992; final manuscript received December 3, 1996

REFERENCES

1. Alon, N., Galil, Z., and Margalit, O. (1991), On the exponent of All Pairs Shortest Path problem, *in* "Proceedings, 32th Annual Symposium on Foundations of Computer Science, 1991," pp. 569–575; a full version to appear in *J. Comput. System Sci.*
2. Awerbuch, personal communication.
3. Coppersmith, D., and Winograd, S. (1981), Matrix multiplication via arithmetic progressions, *in* "Proceedings, 19th Annual Symposium on Theory of Computing," pp. 1–6.
4. Seidel, R. (1992), On the All-Pairs-Shortest-Path problem, *in* "Proceedings, 24th Annual Symposium on Theory of Computing," pp. 745–749.