All Pairs Lightest Shortest Paths

Uri Zwick *

Abstract

Two vertices in a weighted directed graph may be connected by many shortest paths. Although all these paths are shortest in terms of weight, the number of edges on them may vary substantially. This leads us to consider the *All Pairs Lightest Shortest Paths* (APLSP) problem. A solution to this problem is a representation of shortest paths between all of pairs of vertices in the graph such that each of these shortest paths uses a minimal, or a close to minimal, number of edges. We present the following algorithms for obtaining exact or approximate solutions to the APLSP problem:

- An $\tilde{O}(n^{2+\nu})$ time algorithm for exactly solving the APLSP problem for directed graphs with integer weights of small absolute value, where n is the number of vertices in the graph and $\nu < 0.747$ is the solution of the equation $\omega(1,\nu,1) = 3\nu$, where $\omega(1,\nu,1)$ is the exponent of the multiplication of an $n \times n^{\nu}$ matrix by an $n^{\nu} \times n$ matrix.
- An $\tilde{O}(n^{2+\mu})$ time algorithm, where $\mu < 0.575$ is the solution of the equation $\omega(1,\mu,1) = 1+2\mu$, for finding shortest paths between all pairs of vertices in a directed graph on n vertices with integer weights of small absolute value such that each shortest path found uses at most $(1+\epsilon)$ times the minimal number of edges required, for any fixed $\epsilon > 0$.
- An $\tilde{O}(n^{\omega}\log W)$ time algorithm, where $\omega = \omega(1,1,1) < 2.376$ is the the exponent of square matrix multiplication, for finding paths between all pairs of vertices in a directed graph on n vertices with *non-negative* real edge weights, such that each path returned is of stretch at most $(1+\epsilon)$ and uses at most $(1+\epsilon)$ times the minimal number of edges required, for every fixed $\epsilon > 0$. (In the time bound, W is the largest edge weight in the graph after the edge weights are scaled so that the smallest non-zero edge weight in the graph is 1.)

These results extend results obtained recently by the author for the APSP problem.

*Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail address: `zwick@math.tau.ac.il`.
Work supported in part by THE BASIC RESEARCH FOUNDATION administrated by THE ISRAEL ACADEMY OF SCIENCES AND HUMANITIES.

1 Introduction

The *All Pairs Shortest Paths* (APSP) problem is one of the most fundamental algorithmic graph problems. The input to this problem is a *weighted* directed graph. The output is a representation of shortest paths between all pairs of vertices in the graph. The *length* of a path in the graph is the sum of the weights of the edges along the path. A shortest path from a vertex i to a vertex j is a path of minimal length going from i to j. The length of a shortest path from i to j is called the *distance* from i to j and it is denoted by $\delta(i,j)$. There may be many shortest paths from i to j in the graph. The number of edges along these shortest paths may vary considerably. Among all shortest paths connecting i to j it is natural to be interested in a shortest path that uses a smallest number of edges. We say that such a shortest paths is a *lightest shortest path* from i to j. [1] We denote the number of edges along a lightest shortest path from i to j by $\eta(i,j)$.

Most classical algorithms for of the APSP problem can be easily adapted to solve the *All Pairs Lightest Shortest Paths* (APLSP) problem. The APLSP problem for directed graph with arbitrary real weights can therefore be solved in $O(mn + n^2 \log n)$ time, where n is the number of vertices, and m is the number of edges in the graph (Dijkstra [Dij59], Johnson [Joh77], Fredman and Tarjan [FT87]).

The running time of the above mentioned algorithm may be as high as $\Omega(n^3)$. If the edge weights

[1] A note on the terminology: It may have been more appropriate to refer to shortest paths as lightest paths, as the length of a path is defined in terms of the edge weights. A lightest path that uses a minimal number of edges would then be called a *shortest lightest path*. We decided against it as we did not want to redefine the term shortest paths.

of the graph are integers of small absolute value, then fast matrix multiplication algorithms can be used to speed up the operation of algorithms for the APSP problem. Seidel [Sei95] and Galil and Margalit [GM97a],[GM97b] give $\tilde{O}(n^\omega)$ algorithms for solving the APSP problem for *undirected* graphs with small integer weights, where $\omega < 2.376$ is the exponent of square matrix multiplication. (The algorithm of Seidel [Sei95] works only for unweighted graphs.) For *directed* graphs with small integer weights, Alon, Galil and Margalit [AGM97] give an algorithm whose complexity is $\tilde{O}(n^{(3+\omega)/2})$. (Note that $(3+\omega)/2 < 2.688$.) The author [Zwi98] recently improved this result and obtained an algorithm that solves the directed version of the APSP problem, for graphs with small integer weights, in $\tilde{O}(n^{2+\mu})$ time, where μ is the solution of the equation $\omega(1,\mu,1) = 1+2\mu$, where $\omega(1,\mu,1)$ is the exponent of the multiplication of an $n \times n^\mu$ matrix by an $n^\mu \times n$ matrix. The currently best available bounds on $\omega(1,\mu,1)$, obtained by Coppersmith and Winograd [CW90] and by Coppersmith [Cop97], imply that $\mu < 0.575$. The main ingredient used in [Zwi98] is *bridging sets*. The use of bridging sets makes it possible to replace large square matrix products by much smaller rectangular matrix products, thereby speeding up the operation of the algorithm.

The algorithms of [GM97a],[GM97b],[AGM97] and [Zwi98] solve the APSP problem. They do *not* solve the APLSP problem, however. There is simple reduction from the APSP problem to the APLSP problem. Let d_{ij} be the weight of the edge (i,j) in the graph (if there is no such edge then $d_{ij} = +\infty$). If all the edge weights in the graph are integers, then solving the APSP problem with the modified weights $d'_{ij} = n{\cdot}d_{ij}+1$ immediately gives a solution to the APLSP problem. There is one problem, however, that prevents us from using this reduction in conjunction with the algorithms of [GM97a],[GM97b],[AGM97] and [Zwi98]. Namely, the modified weights are *large*, even if the original weights are *small*. The stated time bounds, $\tilde{O}(n^\omega)$, $\tilde{O}(n^{(3+\omega)/2})$ and $\tilde{O}(n^{2+\mu})$ of these algorithms assume, for example, that the weights are polylogarithmic in n. The running time of the algorithm of [Zwi98], for example, is sub-cubic only if the absolute values of the weights are at most $n^{3-\omega}$.

The first algorithm presented in this paper is an algorithm for solving the APLSP problem for directed graphs with small integer weights. The running time of this algorithm is $\tilde{O}(n^{2+\nu})$, where $\nu < 0.747$ is the solution of the equation $\omega(1,\nu,1) = 3\nu$. (Again, this time bound assumes that the absolute values of the weights is at most polylogarithmic. The exact dependency of the running time on the size of the weights will be stated later.) The main ingredient, in addition to bridging sets, used to obtain this algorithm is *double distance products*. At present, solving the APLSP problem for directed graphs with small integer weights is more expensive than just solving the APSP problem.

We can get faster algorithms for the APLSP problem if we are willing to settle for shortest paths that are $(1+\epsilon)$-light, for an arbitrarily small $\epsilon > 0$. A shortest path from i to j is $(1+\epsilon)$-*light* if it uses at most $(1+\epsilon)\cdot\eta(i,j)$ edges. Note that we are still insisting here on genuine shortest paths. The second algorithm presented in the paper finds $(1+\epsilon)$-light shortest paths between all pairs of vertices in a graph on n vertices in $\tilde{O}(n^{2+\mu}/\epsilon)$ time, where $\mu < 0.575$. Up to polylogarithmic factors, this algorithm is therefore as fast as the algorithm given in [Zwi98] for the APSP problem.

An interesting aspect of this second algorithm is that it is *deterministic*. This algorithm is in fact simpler than the deterministic algorithm given in [Zwi98] for the APSP problem. [2]

Finally, if we are willing to settle for $(1+\delta)$-light $(1+\epsilon)$-short paths, we can get an even better running time. Furthermore, the algorithm would now work on graph with arbitrary *non-negative* real weights, and not only when the weights are small integers. A path from i to j is said to be $(1+\delta)$-light $(1+\epsilon)$-short if its length is at most $(1+\epsilon)\delta(i,j)$ and it uses at most $(1+\delta)\eta(i,j)$ edges. (Note that a $(1+\delta)$-light $(1+\epsilon)$-short path may actually contain less than $\eta(i,j)$ edges.) The third algorithm given in this paper runs in $O((n^\omega \log W)/(\epsilon\delta))$ time, where $\omega < 2.376$ is the exponent of square matrix multiplication and W is the largest edge weight in the graph after the edge weights are scaled so that the smallest non-zero edge weight in the graph is 1. It finds $(1+\delta)$-light $(1+\epsilon)$-short paths between all pairs of vertices in the graph.

2 Distance products

It is well known (see, e.g., [AHU74] or [CLR90]), that the APSP problem is closely related to the *min-plus* product, or *distance product*, as we call it here, of matrices. Let A be an $n \times m$ and B be an $m \times n$ matrix. The *distance product* of A and B, denoted $A \star B$, in an $n \times n$ matrix C such that

$$c_{ij} = \min_{k=1}^{m} \{a_{ik} + b_{kj}\} \ , \ \text{for } 1 \le i,j \le n \ .$$

An $n \times n$ matrix W is said to be a matrix of *witnesses* for the distance product $C = A \star B$ if

$$c_{ij} = a_{i,w_{ij}} + b_{w_{ij},j} \ , \ \text{for } 1 \le i,j \le n \ .$$

[2] [Zwi98] contains a deterministic algorithm only for the unweighted case. The more complicated deterministic algorithm for the weighted case will appear in the journal version of that paper.

The following lemma is implicit in Alon, Galil and Margalit [AGM97] (see also [Tak98] and [Zwi98]):

Lemma 2.1 *The witnessed distance product of an $n \times n^r$ matrix by an $n^r \times n$ matrix, whose elements are taken from the set $\{1-M, \ldots, 0, \ldots, M-1\} \cup \{+\infty\}$, can be computed in $\tilde{O}(\min\{\, Mn^{\omega(1,r,1)}\,,\, n^{2+r}\,\})$ time.*

Here $\omega(1,r,1)$ is the number of algebraic operations needed to compute the product of an $n \times n^r$ matrix by an $n^r \times n$ matrix. Note that $\omega = \omega(1,1,1)$ is the famous exponent of (square) matrix multiplication. The best bound on ω is currently $\omega < 2.376$ (Coppersmith and Winograd [CW90]). It is easy to see that a product of $n \times n^r$ matrix by an $n^r \times n$ matrix can be broken into $n^{2(1-r)}$ products of $n^r \times n^r$ matrices, and can therefore by computed in $O(n^{2+r(\omega-2)})$ time. It follows, therefore, that $\omega(1,r,1) \le 2 + r(\omega - 2)$. Better bounds are known. Coppersmith [Cop97] showed that the product of $n \times n^{0.294}$ matrix by an $n^{0.294} \times n$ matrix can be computed using $\tilde{O}(n^2)$ arithmetical operations. Let $\alpha = \sup\{\, r : \omega(1,r,1) = 2 + o(1)\}$. It follows from Coppersmith's result that $\alpha \ge 0.294$. Note that if $\omega = 2 + o(1)$, then $\alpha = 1$. An improved bound for $\omega(1,r,1)$, for $\alpha \le r \le 1$ can be obtained by combining the bounds $\omega(1,1,1) < 2.376$ and $\omega(1,\alpha,1) = 2 + o(1)$. The following lemma is taken from Huang and Pan [HP98]:

Lemma 2.2 *Let $\omega = \omega(1,1,1) < 2.376$ and let $\alpha = \sup\{\, r : \omega(1,r,1) = 2 + o(1)\} > 0.294$. Then*

$$\omega(1,r,1) \le \begin{cases} 2 + o(1) & \text{if } 0 \le r \le \alpha, \\ 2 + \frac{\omega-2}{1-\alpha}(r-\alpha) + o(1) & \text{if } \alpha \le r \le 1. \end{cases}$$

To find witnesses, the algorithm alluded to in Lemma 2.1 uses extensions of the algorithm of Alon and Naor [AN96] for finding witnesses for Boolean matrix multiplications.

A simple algorithm for computing *approximate* distance products using a method called *adaptive scaling* is described in [Zwi98]:

Lemma 2.3 *Let A be an $n \times n^r$ matrix and B be an $n^r \times n$ matrix, both with elements taken from $\{0, 1, \ldots, M-1\} \cup \{+\infty\}$. Let $C = A \star B$. Then, for any $\epsilon > 0$, a matrix C' such that $c_{ij} \le c'_{ij} \le (1+\epsilon)c_{ij}$, for every $1 \le i,j \le n$, can be computed in $\tilde{O}((n^{\omega(1,r,1)} \log M)/\epsilon)$ time.*

In the sequel we let $(C, W) \leftarrow$ **DIST-PROD**(A, B, M) denote the algorithm for computing distance products whose complexity is given in Lemma 2.1. We assume that this algorithm replaces all the elements in the matrices A and B whose absolute values are at least M by $+\infty$ before performing the product. The matrix W returned by **DIST-PROD** is a matrix of witnesses.

```
algorithm DDP(A1, B1, A2, B2, M1, M2)
A ← 2M2 · A1 + A2
B ← 2M2 · B1 + B2
(C, W) ← DIST-PROD(A, B, 2M1M2)
C1 ← ⌊C/(2M2)⌋
C2 ← C mod (2M2)
return (C1, C2, W)
```

Figure 1: An algorithm for computing double distance products.

3 Double distance products

Distance products can be used to find shortest paths. To find lightest, or almost lightest, shortest paths we introduce the notion of *double distance products*.

Definition 3.1 (Double Distance Products) *Let A and A' be two $n \times m$ matrices, and let B and B' be two $m \times n$ matrices. The double distance product (C, C') of the pairs (A, A') and (B, B'), denoted $(C, C') = (A, A') \star (B, B')$ is defined as follows:*

$$c_{ij} \;=\; \min_k \{a_{ik} + b_{kj}\} \quad , \quad \text{for } 1 \le i,j \le n \,,$$

$$c'_{ij} \;=\; \min_{k \,:\, a_{ik}+b_{kj}=c_{ij}} \{a'_{ik} + b'_{kj}\} \quad , \quad \text{for } 1 \le i,j \le n \,.$$

An $n \times n$ matrix W is a matrix of witnesses for the double distance product $(C, C') = (A, A') \star (B, B')$ if

$$c_{ij} = a_{i,w_{ij}} + b_{w_{ij},j} \quad , \quad \text{for } 1 \le i,j \le n \,,$$

$$c'_{ij} = a'_{i,w_{ij}} + b'_{w_{ij},j} \quad , \quad \text{for } 1 \le i,j \le n \,.$$

We can also define double distance products as follows. Each element c_{ij} in the distance product of $C = A \star B$ has a corresponding set of witnesses $W_{ij} = \{\, k \mid c_{ij} = a_{ik} + b_{kj}\,\}$. Then $c'_{ij} = \min_{\, k \in W_{ij}} \{\, a'_{ik} + b'_{kj}\,\}$.

Algorithm **DDP**, shown in Figure 1, computes the double distance product $(C_1, C_2) = (A_1, A_2) \star (B_1, B_2)$ and a corresponding matrix W of witnesses by reducing it to an ordinary distance product. It assumes that the elements of A_1 and B_1 belong to the set $\{1 - M_1, \ldots, 0, \ldots, M_1 - 1\} \cup \{+\infty\}$, and that the elements of A_2 and B_2 belong to the set $\{0, \ldots, M_2 - 1\}$. Algorithm **DDP** uses algorithm **DIST-PROD** to compute ordinary distance products.

Using *adaptive scaling* (see [Zwi98]) we can obtain two algorithms, **ADDP** and **AADDP**, for computing approximate double distance products. Algorithm **ADDP** is given in Figure 2. Algorithm **AADDP**, given in Figure 3, is very similar to **ADDP**. Algorithm **ADDP** adaptively scales the matrices A_2 and B_2 and then uses **DDP** to compute an exact distance product.

```
algorithm ADDP(A_1, B_1, A_2, B_2, M_1, M_2, ε)
C_2 ← (+∞) , W ← (+∞) , R ← 2^⌈log_2(4/ε)⌉
for r ← (log_2 R) to ⌈log_2 M_2⌉ do
begin
  A'_2 ← SCALE(A_2, 2^r, R)
  B'_2 ← SCALE(B_2, 2^r, R)
  (C_1, C̄_2, W') ← DDP(A_1, B_1, A'_2, B'_2, M_1, 3R)
  C̄_2 ← (2^r/R) · C̄_2
  for every 1 ≤ i,j ≤ n do
  if c̄_ij^(2) ≤ 2^(r+1) and c̄_ij^(2) ≤ c_ij^(2) then
  w_ij ← w'_ij , c_ij^(2) ← a_(i,w_ij)^(2) + b_(w_ij,j)^(2)
end
return (C_1, C_2, W)
```

Figure 2: An algorithm for computing approximate double distance products.

Algorithm **AADDP**, on the other hand, adaptively scales the matrices A_1 and B_1 and then calls **ADDP**. If $(C_1, C_2) = (A_1, A_2) \star (B_1, B_2)$ is the exact double distance product of (A_1, A_2) and (B_1, B_2), then **ADDP** returns a triple (C'_1, C'_2, W') such that $C'_1 = C_1$ and C'_2 is an approximation to C_2. Algorithm **AADDP**, on the other hand, returns a triple (C''_1, C''_2, W'') such that both C''_1 and C''_2 are approximate. Both **ADDP** and **AADDP** use the simple scaling algorithm given in Figure 4. More exact specifications of **DDP**, **ADDP** and **AADDP** appear in the following Lemma.

Lemma 3.2 *Let A_1, A_2 be two $n \times n^r$ matrices with elements from $\{1-M_1, \ldots, 0, \ldots, M_1 - 1\} \cup \{+\infty\}$, and let B_1, B_2 be two $n^r \times n$ matrices with elements from $\{0, \ldots, M_2 - 1\}$. Let $(C_1, C_2, W) = (A_1, A_2) \star (B_1, B_2)$.*

(i) Algorithm $\mathbf{DDP}(A_1, A_2, B_1, B_2, M_1, M_2)$ *computes* (C_1, C_2, W) *in* $\tilde{O}(\min\{\, M_1 M_2 \cdot n^{\omega(1,r,1)}\,,\, n^{2+r}\,\})$ *time.*

(ii) Algorithm $\mathbf{ADDP}(A_1, A_2, B_1, B_2, M_1, M_2, \epsilon)$ *runs in*

$$\tilde{O}(\min\{\, M_1 \log M_2 \cdot n^{\omega(1,r,1)}/\epsilon\,,\, n^{2+r}\,\})$$

time and returns matrices $(\bar{C}_1, \bar{C}_2, W)$, where $\bar{C}_1 = (\bar{c}_{ij}^{(1)})$ and $\bar{C}_2 = (\bar{c}_{ij}^{(2)})$, such that $\bar{C}_1 = A_1 \star B_1$, and such that

$$c_{ij}^{(2)} \le \bar{c}_{ij}^{(2)} \le (1+\epsilon) c_{ij}^{(2)} \;,$$
$$\bar{c}_{ij}^{(1)} = a_{i,w_{ij}}^{(1)} + b_{w_{ij},j}^{(1)} \;,$$
$$\bar{c}_{ij}^{(2)} = a_{i,w_{ij}}^{(2)} + b_{w_{ij},j}^{(2)} \;,$$

for every $1 \le i, j \le n$.

(iii) Algorithm $\mathbf{AADDP}(A_1, A_2, B_1, B_2, M_1, M_2, \epsilon, \delta)$ *runs in*

$$\tilde{O}(\min\{\, (\log M_1)(\log M_2) \cdot n^{\omega(1,r,1)}/(\epsilon\delta)\,, n^{2+r}\,\})$$

time and returns matrices $(\bar{C}_1, \bar{C}_2, W)$, where $\bar{C}_1 = (\bar{c}_{ij}^{(1)})$ and $\bar{C}_2 = (\bar{c}_{ij}^{(2)})$, such that

$$c_{ij}^{(1)} \le \bar{c}_{ij}^{(1)} \le (1+\epsilon) c_{ij}^{(1)} \;,$$
$$\bar{c}_{ij}^{(2)} \le (1+\delta) c_{ij}^{(2)} \;,$$
$$\bar{c}_{ij}^{(1)} = a_{i,w_{ij}}^{(1)} + b_{w_{ij},j}^{(1)} \;,$$
$$\bar{c}_{ij}^{(2)} = a_{i,w_{ij}}^{(2)} + b_{w_{ij},j}^{(2)} \;,$$

for every $1 \le i, j \le n$.

```
algorithm AADDP(A_1, B_1, A_2, B_2, M_1, M_2, ε, δ)
C_1 ← (+∞) , C_2 ← (+∞)
W ← (+∞) , R ← 2^⌈log_2(4/ε)⌉
for r ← (log_2 R) to ⌈log_2 M_1⌉ do
begin
  A'_1 ← SCALE(A_1, 2^r, R)
  B'_1 ← SCALE(B_1, 2^r, R)
  (D_1, D_2, W') ←
            ADDP(A'_1, B'_1, A_2, B_2, 3R, M_2, δ)
  D_1 ← (2^r/R) · D_1
  for every 1 ≤ i,j ≤ n do
  if d_ij^(1) ≤ 2^(r+1) and d_ij^(1) ≤ c_ij^(1) then
  w_ij ← w'_ij , c_ij^(1) ← a_(i,w_ij)^(1) + b_(w_ij,j)^(1) , c_ij^(2) ← d_ij^(2)
end
return (C_1, C_2, W)
```

Figure 3: An algorithm for computing doubly approximate double distance products.

```
algorithm SCALE(A, M, R)
a'_ij ← { ⌈R a_ij / M⌉   if 0 ≤ a_ij ≤ M
        { 3R             otherwise
Return A'
```

Figure 4: A simple scaling algorithm.

The proof of Lemma 3.2(i) is immediate. Parts (ii) and (iii) of Lemma 3.2 are proved using essentially the same arguments used to prove Lemma 5.1 of [Zwi98]. We omit these simple proofs from this extended abstract.

4 Lightest Shortest Paths

A very simple randomized algorithm, called **RAND-LIGHT-SHORT-PATH**, or **RLSP** for short, for finding lightest shortest paths between

```
algorithm RAND-LIGHT-SHORT-PATH(D)
M ← max{ |d_ij| : d_ij ≠ +∞}
for every 1 ≤ i,j ≤ n do f_ij ← d_ij , g_ij ← 1 , w_ij ← j
for ℓ ← 1 to ⌈log_{3/2} n⌉ do
begin
  s ← (3/2)^ℓ
  B ← RAND({1,2,...,n},(9 ln n)/s)
  (F',G',W') ← DDP(F[*,B],F[B,*] , G[*,B],G[B,*] , sM,s)
  for every 1 ≤ i,j ≤ n do
  if f'_ij < f_ij then f_ij ← f'_ij , g_ij ← g'_ij , w_ij ← b_{w'_ij}
end
return (F,G,W)
```

Figure 5: A randomized algorithm for finding lightest shortest paths.

all pairs of vertices in a graph is given in Figure 5. The algorithm is a simple modification of algorithm **RAND-SHORT-PATH** of [Zwi98]. The main difference is that double distance products are used here instead of ordinary distance products.

Algorithm **RLSP** uses the call $B \leftarrow$ **RAND**(V,p) to produce a random set of vertices to which each vertex belongs, independently, with probability p. If $p \geq 1$, then $B \leftarrow V$. With very high probability, this random set is a *bridging set* (see below). The algorithm then uses algorithm **DDP** to compute the double distance product $(F[*,B],G[*,B]) \star (F[B,*],G[B,*])$, where $F[*,B]$ denotes the matrix composed of the columns of F whose indices belong to the set B and $F[B,*]$ denotes the matrix composed of the rows of B whose indices belong to B. The matrices $G[*,B]$ and $G[B,*]$ are defined similarly.

Theorem 4.1 *Algorithm* **RAND-LIGHT-SHORT-PATH** *finds, with very high probability, lightest shortest paths between all pairs of vertices in a graph on n vertices in which all the edge weights are integers with absolute values at most $M = n^t$, where $t \leq 3-\omega$, in $\tilde{O}(n^{2+\nu(t)})$ time, where $\nu = \nu(t)$ satisfies $\omega(1,\nu,1) = 3\nu - t$.*

The bound on $\omega(1,r,1)$ given in Lemma 2.2 implies that the complexity of **RLSP** is

$$\tilde{O}(n^{2+\frac{2-\alpha\omega}{5-\omega-3\alpha}} \cdot M^{\frac{1-\alpha}{5-\omega-3\alpha}}) ,$$

which is at most $\tilde{O}(n^{2.747} \cdot M^{0.406})$.

The correctness of algorithm **RLSP** follows from arguments similar to the arguments used to prove the correctness of algorithm **RAND-SHORT-PATH** in [Zwi98]. We will see an extension of these arguments in the proof of Theorem 5.2 of the next section. We consentrate here on the complexity analysis of **RLSP**.

The time taken by the ℓ-th iteration is dominated by the time needed to compute a double distance product $(F_1,G_1)\star(F_2,G_2)$, where F_1 and G_1 are $n\times m$ matrices, and F_2 and G_2 are $m \times n$ matrices, where $m = O((n\log n)/s)$. The finite elements of F_1 and F_2 are of absolute value at most sM. The elements of G_1 and G_1 are all at most s. (More precisely, elements in F_1 and F_2 that are larger than sM, as well as elements in these matrices that correspond to elements in G_1 and G_2 that are larger than s are replaced by $+\infty$, and elements in G_1 and G_2 that are larger than s are replaced by s.) . If $M = n^t$ and $s = n^{1-r}$, then the time taken by this double distance product, according to Lemma 3.2(i) is $\tilde{O}(\min\{s^2M \cdot n^{\omega(1,r,1)}, n^{2+r}\}) = \tilde{O}(\min\{n^{t+\omega(1,r,1)+2(1-r)}, n^{2+r}\})$. Note that $\omega(1,r,1) + 2(1-r)$ is decreasing in r while $2+r$ is increasing in r. The running time of an iteration is maximized when $t + \omega(1,r,1) + 2(1-r) = 2+r$, or equivalently, when $\omega(1,r,1) = 3r - t$. As there are only $O(\log n)$ iterations, the running time of **RLSP** is as claimed.

If the input graph $G = (V,E)$ does not contain cycles of non-positive weight, then the matrix W returned by **RLSP** can be easily used to obtain lightest shortest paths. If there is a path from i to j, i.e., if $f_{ij} \neq +\infty$, then a call to **PATH**(W,i,j), where **PATH** is the procedure given in Figure 6, finds a lightest shortest path from i to j. This lightest shortest path is obtained by concatenating a lightest shortest path from i to w_{ij}, found recursively, and a lightest shortest path from w_{ij} to j, also found recursively. The running time of **PATH**(W,i,j) is proportional to the number of edges on the path from i to j encoded by W. If the graph does contain non-positive cycles, then the situation is slightly more complicated. The details wiil appear in the full version of this paper.

In the sequel we also use a procedure **SUB-PATH**(W,i,j,s) which starts to reconstruct

```
algorithm PATH(W, i, j)

if w_ij = j then return ⟨i, j⟩
else return PATH(W, i, w_ij) . PATH(W, w_ij, j)
```

Figure 6: Finding a (lightest) shortest path from i to j.

a path from i to j but stops if it finds out that this path contains more than s edges. It runs in $O(s)$ time and returns a set of at most $s+1$ vertices that lie on the path from i to j encoded by W.

Algorithm **RAND-LIGHT-SHORT-PATH** implicitly used the notion of *bridging sets*:

Definition 4.2 (Bridging sets) *Let $G=(V,E)$ be a weighted directed graph and let $s \geq 1$. A set of vertices B is said to be an s-*bridging set *if for every two vertices $i,j \in V$ such that $\eta(i,j) \geq s$, i.e., if all shortest paths from i to j use at least s edges, there exists $k \in B$, such that $\delta(i,j) = \delta(i,k) + \delta(k,j)$. The set B is said to be a* strong s-bridging set *if for every two vertices $i,j \in V$ such that $\eta(i,j) \geq s$, there exists $k \in B$, such that $\delta(i,j) = \delta(i,k) + \delta(k,j)$ and $\eta(i,j) = \eta(i,k) + \eta(k,j)$.*

Recall that $\eta(i,j)$ is the minimal number of edges on a shortest path from i to j. It is not difficult to see that if s is an integer then we can replace the condition $\eta(i,j) \geq s$ in the above definition by the condition $\eta(i,j) = s$. The correctness of algorithm **RAND-LIGHT-SHORT-PATH** relies on the following lemma whose proof is implicit in [Zwi98].

Lemma 4.3 *Let $G=(V,E)$ be a weighted directed graph on n vertices and let $s \geq 1$. If B is a random set obtained by running* **RAND**$(\{1,2,\ldots,n\},(3\ln n)/s)$, *i.e., if each vertex of V is added to B independently with probability $(3\ln n)/s$, then with high probability B is a* strong *s-bridging set.*

Proof: Let i and j be two vertices such that $\eta(i,j)=s$. Let p be a lightest shortest path from i to j. As there are more than s vertices on the path p, the probability that none of the vertices on p would belong to B is at most

$$\left(1-\frac{3\ln n}{s}\right)^s \;\leq\; e^{-3\ln n} = n^{-3}\,.$$

As there are less than n^2 pairs of vertices in the graph, the probability of failure is at most $n^2 \cdot n^{-3} = 1/n$. □

A very simple deterministic algorithm for finding bridging sets, called **FIND-BRIDGE** was given in [Zwi98]. For completeness, it is also brought here in Figure 7. Algorithm **FIND-BRIDGE** uses algorithm **HITTING-SET** to find a small set B that *hits* all the sets in the collection $\mathcal{C}$. Algorithm **HITTING-SET** uses the greedy heuristic and its running time is linear in the sum of the sizes of the sets in the collection $\mathcal{C}$.

```
algorithm FIND-BRIDGE(W, s)

C ← φ

for every 1 ≤ i, j ≤ n do
    U ← SUB-PATH(W, i, j, s)
    if |U| = s + 1 then C ← C ∪ {U}

end

B ← HITTING-SET(C)

return B
```

Figure 7: A deterministic algorithm for constructing s-bridging sets.

Lemma 4.4 *If the matrix W can be used to reconstruct shortest paths between all pairs of vertices $i,j \in V$ for which $\eta(i,j) \leq s$, then algorithm* **FIND-BRIDGE** *finds an s-bridging set of size at most $n(2\ln n+1)/s$. The running time of* **FIND-BRIDGE** *is $O(n^2 s)$. Furthermore, if the paths reconstructed using W are* lightest *shortest paths then the set returned by* **FIND-BRIDGE** *is a* strong *s-bridging set.*

A proof of Lemma 4.4 can again be found in [Zwi98]. A deterministic algorithm for finding lightest shortest paths, called **LIGHT-SHORT-PATH**, or **LSP** for short, is obtained by replacing the call to **RAND** in **RAND-LIGHT-SHORT-PATH** by

```
if s ≤ n^{1/2} then B ← FIND-BRIDGE(W, ⌊s/3⌋)
```

We compute new bridging sets only when $s \leq n^{1/2}$ as computing bridging sets for larger values of s may consume too much time. (Recall that the running time of **FIND-BRIDGE** is $O(n^2 s)$.) The algorithm remains correct as an s-bridging set is also an s'-bridging set for every $s' \geq s$. The use of a bridging set of size $\Theta(n^{1/2}\log n)$ in the remaining iterations does not change the overall running time of the algorithm.

5 Almost-lightest shortest paths

A deterministic algorithm for finding $(1+\epsilon)$-light shortest paths between all pairs of vertices in a graph is given in Figure 8. Algorithm **ALMOST-LIGHT-SHORT-PATH**, or **ALSP** for short, differs from algorithm **LSP** of the previous section in two respects. The first is that approximate double distance products are now used. The second is that **FIND-BRIDGE** is now used to find an $\lfloor s/(9\log n)\rfloor$-bridging set, instead of an $\lfloor s/3\rfloor$-bridging set. Why this is important will become clear in the proof of Lemma 5.1.

```
algorithm ALMOST-LIGHT-SHORT-PATH(D)
M ← max{ |d_ij| : d_ij ≠ +∞}
for every 1 ≤ i,j ≤ n do f_ij ← d_ij , g_ij ← 1 , w_ij ← j
for ℓ ← 1 to ⌈log_{3/2} n⌉ do
begin
  s ← (3/2)^ℓ
  if s ≤ n^{1/2} then B ← FIND-BRIDGE(W, ⌊s/(9 log n)⌋)
  (F', G', W') ← ADDP(F[*, B], F[B, *], G[*, B], G[B, *], sM, s, ε/(4 log n))
  for every 1 ≤ i,j ≤ n do
  if f'_ij < f_ij then f_ij ← f'_ij , g_ij ← g'_ij , w_ij ← b_{w'_ij}
end
return (F, G, W)
```

Figure 8: A deterministic algorithm for finding almost-lightest shortest paths.

Lemma 5.1 *After the ℓ-th iteration of* **ALSP** *we have, for every $1 \le i,j \le n$, if $\eta(i,j) \le s = (3/2)^\ell$ then $f_{ij} = \delta(i,j)$ and $g_{ij} \le (1+\epsilon_\ell)\eta(i,j)$, where $\epsilon_0 = 0$ and $\epsilon_\ell = \frac{\epsilon}{4\log n} + (1+\frac{1}{4\log n})\epsilon_{\ell-1} \le \epsilon$, and the matrix W can be used to reconstruct a shortest path from i to j that uses at most g_{ij} edges.*

Proof: We prove the lemma by induction on ℓ. The basis of the induction is easily established. We show now that if the claim holds for $\ell-1$ then it also holds for ℓ.

Let i and j be two vertices such that $\eta(i,j) \le s$. If $\eta(i,j) \le 2s/3$ then the clain follows directly from the induction hypothesis. We may assume, therefore, that $2s/3 \le \eta(i,j) \le s$. Let p be a lightest shortest path from i to j. Let I and J be two vertices on p separated by exactly $\lfloor s/(9\log n)\rfloor$ edges and such that i and I, and J and j are separated on p by at most $s/2$ edges, see Figure 9.

Let q be the shortest path from I to J obtained by running **PATH**(W,I,J). As $\eta(I,J) = \lfloor s/(9\log n)\rfloor$, then by the induction hypothesis, the number of edges on q, denoted $\eta(q)$, is at most $g_{IJ} \le (1+\epsilon_{\ell-1})\eta(I,J)$. The set B found by the call to **FIND-BRIDGE** in the ℓ-th iteration contains a vertex $k \in B$ that lies on the path q. It is easy to see that

$$\eta(I,k) + \eta(k,J) \;\le\; \eta(q) \;\le\; (1+\epsilon_{\ell-1})\eta(I,J)\ ,$$

and that

$$\begin{aligned}
\eta(i,k)+\eta(k,j) &\le \eta(i,I) + (\eta(I,k)+\eta(k,J)) + \eta(J,j) \\
&\le \eta(i,I) + (1+\epsilon_{\ell-1})\eta(I,J) + \eta(J,j) \\
&\le \eta(i,j) + \epsilon_{\ell-1}\eta(I,J) \\
&\le (1+\frac{\epsilon_{\ell-1}}{6\log n})\eta(i,j)\ .
\end{aligned}$$

The last inequality above follows from the fact that $\eta(i,j) \ge 2s/3$ while $\eta(I,J) = \lfloor s/(9\log n)\rfloor$.

As

$$\begin{aligned}
\eta(i,k) &\le \eta(i,I) + \eta(I,k) \\
&\le \eta(i,I) + (1+\epsilon_{\ell-1})\eta(I,J) \;\le\; 2s/3\ ,
\end{aligned}$$

we get, by the induction hypothesis, that $f_{ik} = \delta(i,k)$ and $g_{ik} \le (1+\epsilon_{\ell-1})\eta(i,k)$. Similarly, we get that $f_{kj} = \delta(k,j)$ and $g_{kj} \le (1+\epsilon_{\ell-1})\eta(k,j)$. As $k \in B$ lies on a shortest path from i to j, we get that after the computation of the approximate double distance product of the ℓ-th iteration we have

$$f_{ij} \;\le\; f_{ik} + f_{kj} \;=\; \delta(i,k) + \delta(k,j) \;=\; \delta(i,j)\ .$$

As f_{ij} is always the length of a path from i to j in the graph, we always have $f_{ij} \ge \delta(i,j)$. Thus, at the end of the ℓ-th iteration we have $f_{ij} = \delta(i,j)$, as required. Similarly, we also have

$$\begin{aligned}
g_{ij} &\le (1+\frac{\epsilon}{4\log n}) \cdot (g_{ik} + g_{kj}) \\
&\le (1+\frac{\epsilon}{4\log n})(1+\epsilon_{\ell-1}) \cdot (\eta(i,k) + \eta(k,j)) \\
&\le (1+\frac{\epsilon}{4\log n})(1+\epsilon_{\ell-1})(1+\frac{\epsilon_{\ell-1}}{6\log n}) \cdot \eta(i,j) \\
&\le (1+\frac{\epsilon}{4\log n} + (1+\frac{1}{4\log n})\epsilon_{\ell-1}) \cdot \eta(i,j) \\
&\le (1+\epsilon_\ell) \cdot \eta(i,j)\ .
\end{aligned}$$

It is easy to show by induction that for every $1 \le \ell \le 2\log n$ we have $\epsilon_\ell \le \frac{\ell}{2\log n}\epsilon$. As $\log_{3/2} n \le 2\log n$, we get that $\epsilon_\ell \le \epsilon$ during all the iterations of the algorithm. □

The complexity analysis of algorithm **ALSP** is similar to the complexity analysis of **RLSP** performed in the previous section. We thus get:

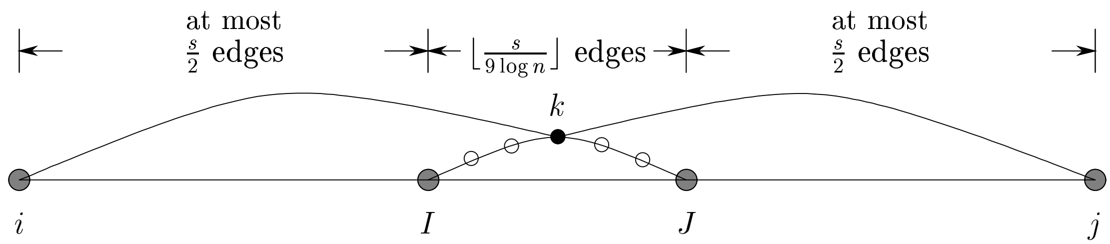

Figure 9: The vertices i and j are connected by a shortest path that uses at most s edges.

Theorem 5.2 *Algorithm* **ALMOST-LIGHT-SHORT-PATH** *runs in* $\tilde{O}(n^{2+\mu(t)})$ *time, where* $\mu = \mu(t)$ *satisfies the equation* $\omega(1, \mu, 1) = 1 + 2\mu - t$*, and* $M = n^t$ *is an upper bound on the absolute value of the edge weights in the graph. It finds* $(1+\epsilon)$*-light shortest paths between all pairs of vertices in the graph.*

The bound on $\omega(1, r, 1)$ given in Lemma 2.2 implies that the complexity of **ALSP** is

$$\tilde{O}(n^{2+\frac{1-\alpha(\omega-1)}{4-\omega-2\alpha}} \cdot M^{\frac{1-\alpha}{4-\omega-2\alpha}}) \ ,$$

which is at most $\tilde{O}(n^{2.575} \cdot M^{0.681})$.

6 Almost-lightest almost-shortest paths

A deterministic algorithm, **ALMOST-LIGHT-ALMOST-SHORT-PATH**, or **ALASP** for short, for finding almost-lightest almost shortest paths is given in Figure 10. The algorithm is simpler than the algorithms given in the preceding sections. No bridging sets are needed this time.

Using arguments similar to the ones used in the proof of Lemma 5.1, but much simpler, we can get:

Lemma 6.1 *After the* ℓ*-th iteration of* **ALASP** *we have, for every* $1 \le i, j \le n$*, if* $\eta(i,j) \le s = (3/2)^\ell$ *then* $f_{ij} = (1+\epsilon_\ell)\delta(i,j)$ *and* $g_{ij} \le (1+\delta_\ell)\eta(i,j)$*, where* $\epsilon_0 = \delta_0 = 0$ *and* $1+\epsilon_\ell \le (1+\frac{\epsilon}{4\log n})(1+\epsilon_{\ell-1}) \le 1+\epsilon$*,* $1+\delta_\ell \le (1+\frac{\delta}{4\log n})(1+\delta_{\ell-1}) \le 1+\delta$*, and the matrix* W *can be used to reconstruct a path from* i *to* j *of length at most* f_{ij} *that uses at most* g_{ij} *edges.*

As a consequence, we get:

Theorem 6.2 *If all the edges weights are in the range* $\{0, 1, \ldots, M-1\}$*, then algorithm* **ALASP** *runs in* $\tilde{O}((n^\omega \log M)/(\epsilon\delta))$ *time, where* $\omega < 2.376$ *is the exponent of square matrix multiplication. It finds* $(1+\delta)$*-light* $(1+\epsilon)$*-short paths between all pairs of vertices in the graph.*

Algorithm **ALASP** can be easily extended to work on graphs with non-negative real weights. All we have to do is to scale the weights so that the smallest non-zero weight is $1/\epsilon$, round the weights up, and apply **ALASP** with the resulting integer weights.

7 Shortest lightest paths and other extensions

So far we have been interested in shortest paths that are as light as possible. We may also be interested in lightest paths, i.e., paths that use as few edges as possible, which are as short as possible. The algorithms presented here can be easily adapted to the solve the corresponding *All Pairs Shortest Lightest Paths* (APSLP) problem. The results can be further generalized to the case in which each edge e has k weights $w_1(e), w_2(e), \ldots, w_2(e)$ associated with it and among all paths connecting two vertices i and j we are interested in a path p for which $(w_1(p), w_2(p), \ldots, w_k(p))$ is as small as possible with respect to the lexicographic ordering.

8 Concluding remarks

The *All Pairs Lightest Shortest Paths* (APLSP) problem is a very naturally defined problem. In many settings it can be easily reduced, without any loss of efficiency, to the *All Pairs Shortest Paths* (APSP) problem. This, however, does not seem to be the case if the edge weights of the graph are integers of small absolute value. We have presented several algorithms designed to solve this version of the APLSP problem.

We were led to consider the APLSP problem while trying to obtain a deterministic version of the simple randomized algorithm for solving the APSP problem in directed graphs with small integer weights given in [Zwi98]. While it is possible to derandomize this algorithm directly, it turns out that a much simpler deterministic algorithm, for the weighted case, is obtained if the randomized algorithm is first converted

```
algorithm ALMOST-LIGHT-ALMOST-SHORT-PATH(D, ε, δ)

M ← max{ d_ij : d_ij ≠ +∞}
for every 1 ≤ i, j ≤ n do f_ij ← d_ij , g_ij ← 1 , w_ij ← j

for ℓ ← 1 to ⌈log_2 n⌉ do
begin
  (F', G', W') ← AADDP(F, F, G, G , Mn, n , ε/(4 log n), δ/(4 log n))
  for every 1 ≤ i, j ≤ n do
  if f'_ij < f_ij then f_ij ← f'_ij , g_ij ← g'_ij , w_ij ← w'_ij
end

return (F, G, W)
```

Figure 10: Almost-lightest almost-shortest paths.

into an algorithm that finds almost-lightest shortest paths. The resulting deterministic algorithm, **ALSP**, is described in Section 5.

Many open problems still remain. Is the APLSP problem for directed graphs with small integer weights really more difficult than the APSP problem for such graphs? Is the APSP problem for directed graphs really more difficult than the APSP problem for *undirected* graphs?

References

[AGM97] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54:255–262, 1997.

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1974.

[AN96] N. Alon and M. Naor. Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16:434–449, 1996.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to algorithms*. The MIT Press, 1990.

[Cop97] D. Coppersmith. Rectangular matrix multiplication revisited. *Journal of Complexity*, 13:42–49, 1997.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[FT87] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.

[GM97a] Z. Galil and O. Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134:103–139, 1997.

[GM97b] Z. Galil and O. Margalit. All pairs shortest paths for graphs with small integer length edges. *Journal of Computer and System Sciences*, 54:243–254, 1997.

[HP98] X. Huang and V.Y. Pan. Fast rectangular matrix multiplications and applications. *Journal of Complexity*, 14:257–299, 1998.

[Joh77] D.B. Johnson. Efficient algorithms for shortest paths in sparse graphs. *Journal of the ACM*, 24:1–13, 1977.

[Sei95] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51:400–403, 1995.

[Tak98] T. Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica*, 20:309–318, 1998.

[Zwi98] U. Zwick. All pairs shortest paths in weighted directed graphs – exact and almost exact algorithms. In *Proceedings of the 39rd Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, California*, 1998. 310–319.