Approximating All-Pair Bounded-Leg Shortest Path and APSP-AF in Truly-Subcubic Time

Ran Duan[1]
Institute for Interdisciplinary Information Sciences, Tsinghua University, China
duanran@mail.tsinghua.edu.cn

Hanlin Ren
Institute for Interdisciplinary Information Sciences, Tsinghua University, China
rhl16@mails.tsinghua.edu.cn

Abstract

In the bounded-leg shortest path (BLSP) problem, we are given a weighted graph G with nonnegative edge lengths, and we want to answer queries of the form "what's the shortest path from u to v, where only edges of length $\leq L$ are considered?". A more general problem is the APSP-AF (all-pair shortest path for all flows) problem, in which each edge has two weights – a length d and a capacity f, and a query asks about the shortest path from u to v where only edges of capacity $\geq f$ are considered.

In this article we give an $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-3/2}\log W)$ time algorithm to compute a data structure that answers APSP-AF queries in $O(\log(\epsilon^{-1}\log(nW)))$ time and achieves $(1+\epsilon)$-approximation, where $\omega < 2.373$ is the exponent of time complexity of matrix multiplication, W is the upper bound of integer edge lengths, and n is the number of vertices. This is the first truly-subcubic time algorithm for these problems on dense graphs. Our algorithm utilizes the $O(n^{(\omega+3)/2})$ time max-min product algorithm [Duan and Pettie 2009]. Since the all-pair bottleneck path (APBP) problem, which is equivalent to max-min product, can be seen as all-pair reachability for all flow, our approach indeed shows that these problems are almost equivalent in the approximation sense.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Graph Theory, Approximation Algorithms, Combinatorial Optimization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.42

1 Introduction

The shortest path problem is one of the most fundamental problems in algorithmic graph theory. In this paper we study one of its variants, the apBLSP(all-pair *bounded-leg* shortest path), and a more general problem, the APSP-AF(all-pair shortest path for all flows) problem.

In apBLSP, we are given a weighted graph G, and we want to find the shortest path from u to v when only using edges with length no more than L. Answering one such query is easy: just discard edges with length $> L$ and run a shortest-path algorithm. So we consider that there are many queries (u, v, L), and we need to preprocess the graph G to answer these queries efficiently.

The apBLSP problem is a special case of the APSP-AF problem. In the APSP-AF problem, we're given a directed graph G, and each edge has a *length* d and a *capacity* f. Let W and K be the maximum length and maximum capacity, respectively. We assume that all lengths are nonnegative integers. The length of a path is the sum of lengths of all its edges, while the capacity of a path is the minimum capacity over its edges. The goal

[1] R. Duan is supported by a China Youth 1000-Talent grant.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 42; pp. 42:1–42:12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is to compute from G a data structure that handles such queries: given $u, v \in V(G)$ and $1 \leq f \leq K$, determine the shortest path (in terms of length) from u to v with capacity at least f. We call the length of this shortest path "distance from i to j under flow constraint f". Given an instance of the apBLSP problem, we can give the capacity of edges in the order reversely as the order of their lengths, then the apBLSP is easily reducible to APSP-AF.

We consider data structures whose query algorithm uses *no* arithmetic operations. In other words, such a data structure would precompute the answers for all possible queries and store them up. As observed in [7], any such data structure requires $\Omega(n^4)$ space to report the exact answer. It's easy to come up with an $O(n^4)$ exact algorithm: let $\delta(i, j, f)$ be the distance from i to j under flow constraint f, we add edges in descending order of their capacities, and when we add an edge e from i to j with length d and capacity f, we set

$$\delta(i', j', f) \leftarrow \min\{\delta(i', j', f_0), \delta(i', i, f_0) + d + \delta(j, j', f_0)\}, \forall i', j' \in V,$$

where f_0 is the capacity of the last edge we added before e. In this sense, the exact version of APSP-AF is not very interesting. In the approximation setting, what we would like to compute is actually a *df*-matrix[2] A, which is a matrix whose entries are sets of pairs (d, f). The answer to the query (i, j, f) is simply $\min\{d : (d, f') \in A_{ij}, f' \geq f\}$, and can be computed in $O(\log |A_{ij}|)$ time by binary search. The data structure achieves a stretch of $(1 + \epsilon)$ if for any possible query (i, j, f), the returned value is in $[D, (1 + \epsilon)D]$ where D is the actual APSP-AF distance.

Related work. The bounded-leg shortest path problem was firstly considered by Bose *et al.* [4], where they showed that a data structure of size $O(\epsilon^{-1}n^2 \log n)$ can be constructed in $O(n^5)$ time to $(1 + \epsilon)$-approximate the bounded-leg length in Euclidean graphs; if explicit paths are required, their data structure needs $O(\epsilon^{-1}n^3 \log n)$ size. They also gave a data structure of size $O(n^{5/2})$ computable in $O(n^{5/2})$ time to support exact apBLSP queries in planar graphs. Roditty and Segal [11] showed a $(1 + \epsilon)$-approximate data structure for general graphs, which has size $\tilde{O}(n^{2.5})$ and can be computed in $\tilde{O}(n^4)$ time.[3] They also obtained a $(1 + \epsilon)$-approximate data structure for apBLSP in any l_p-metric, which has size $O(n^2 \log n)$ and is computable in $O(n^3(\log^3 n + \log^2 n \cdot \epsilon^{-d}))$ time. In [7], Duan and Pettie improved the time complexity for general graphs to $O(n^3 \log^3 n)$ and the space complexity to $O(\epsilon^{-1}n^2 \log n)$. For planar directed graphs, they also gave a data structure with size $O(kn^{1+1/k})$ that answers bounded-leg reachability queries in $\tilde{O}(n^{\frac{k-1}{2k}})$ time, for any integer $k \geq 2$.

The APSP-AF problem was introduced by Shinn and Takaoka [12, 13, 14, 15]. In graphs with unit edge lengths, they showed an $O(\sqrt{K}n^{(\omega+9)/4})$ exact algorithm. In general graphs, they showed exact algorithms of running time $O(Kn^3)$, $\tilde{O}(\sqrt{K}W^{(\omega+5)/4}n^{(\omega+9)/4})$ and $O(m^2n + \min(nW, mn^2 \log \frac{W}{mn}))$ respectively.

We can also view APSP-AF as an *offline* version of partially dynamic shortest path problem: edge insertions and distance queries appear by descending order of f, and we know the whole sequence of insertions and queries at the beginning of algorithm. Most works on dynamic shortest path problem are *online*: the sequence of operations and queries must be processed in order. In *incremental* setting, Ausiello *et al.* [1] showed an algorithm to handle all pair distance queries and edge insertions in a digraph in $O(Wn^3 \log(nW))$ time. In *decremental* setting, Bernstein [3] showed a $(1+\epsilon)$-approximate algorithm with a total running

[2] See section 2.2.
[3] $\tilde{O}$ hides poly$(\log n)$ factor.

time of $\tilde{O}(mn\epsilon^{-1}\log R)$ in weighted digraphs, where R is the ratio of the largest weight that was ever seen in the graph to the smallest such weight. For unweighted digraphs, Baswana, Hariharan and Sen [2] showed a randomized algorithm with $O(\min(n^{3/2}m\sqrt{\log n}, n^3\log^2 n))$ total update time which returns exact answers w.h.p, and a $(1+\epsilon)$-approximate algorithm with $O(nm\epsilon^{-2}\log n + n^2\epsilon^{-1}\sqrt{m\log n})$ total update time. More results can be found in surveys [6, 10]. However, the online setting has its disadvantages: in both incremental and decremental settings, even a reachability oracle in a truly-subcubic total time would refute the OMV conjecture [9].

Our algorithm relies on fast algorithms for max-min product. The first truly-subcubic time algorithm for max-min product was discovered by Vassilevska, Williams and Yuster [16], which has running time $O(n^{2+\omega/3})$. Subsequently, Duan and Pettie [8] improved this algorithm to $O(n^{(\omega+3)/2})$ time.

Our contribution. We show that a data structure for APSP-AF within stretch $(1+\epsilon)$ can be computed in $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-3/2}\log W)$ time for any n-vertex graph and any $\epsilon > 0$, where $\omega < 2.373$ is the exponent of time complexity of matrix multiplication [5]. This is the first truly-subcubic time (*i.e.* $O(n^{3-\delta}\mathrm{poly}(\epsilon^{-1}, \log W))$ for some $\delta > 0$) algorithm to approximate apBLSP or APSP-AF problem in general graphs when $m = \Theta(n^2)$ and $W = \Omega(n)$. Our data structure uses $O(n^2\epsilon^{-1}\log n \log(nW))$ space.

We also establish the equivalence between approximating APSP-AF and computing max-min product. In particular, it's shown in Section 3 that if the max-min product of two matrices can be computed in $T(n)$ time, then a $(1+\epsilon)$-approximate data structure for APSP-AF can be computed in $\tilde{O}(T(n)\epsilon^{-2}\log W)$ time. This is optimal up to $\mathrm{poly}(\epsilon^{-1}, \log n, \log W)$ factors in the sense that APSP-AF approximation is at least as hard as max-min product. In fact, max-min product is reducible to the special case of *bounded-leg reachability problem* where the goal is only to query if u can reach v via edges of capacity $\geq f$. [4] Consider computing the max-min product of two $n \times n$ matrices A, B. We construct a directed graph $G = (V, E)$ with $3n$ vertices $V = \{u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n, w_1, w_2, \ldots, w_n\}$. For every i, j, link an edge from u_i to v_j with capacity A_{ij} and link an edge from v_i to w_j with capacity B_{ij}. Let C be the max-min product of A and B, then $C_{ij} \geq f$ iff w_j is reachable from u_i by edges of capacity $\geq f$.

2 Preliminaries

We denote $[n] = \{1, \ldots, n\}$, $[n]_0 = [n] \cup \{0\}$, $\min(\varnothing) = +\infty, \max(\varnothing) = -\infty$. Rows and columns of matrices are numbered from 1 to n. Let $\omega < 2.373$ be the exponent of time complexity of matrix multiplication. For a set of $m \times n$ matrices $\{A_i\}$, $\min\{A_i\}$ is the entry-wise minimum of them. Let W, K be the upper bound of lengths and capacities of edges, respectively; *i.e.* for any edge, its length is in $[W]_0$ and its capacity is in $[K]$.

2.1 Matrix products

We introduce the *distance product* and *max-min product* for two matrices, denoted as $\star$ and $\circledvee$ respectively.

[4] It should be "edges of length $\leq L$" for bounded-leg reachability problem; However this constraint is equivalent.

▶ **Definition 2.1.** For two matrices A, B of size $n \times n$, we define:

$$(A \star B)_{ij} = \min_{k\in[n]}\{A_{ik} + B_{kj}\};$$
$$(A \circledcirc B)_{ij} = \max_{k\in[n]}\{\min\{A_{ik}, B_{kj}\}\}.$$

A simpler version of our algorithm in Section 3, which runs in $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-2}\log W)$, makes use of max-min product algorithm [8] for two matrices.

▶ **Lemma 2.2** ([8]). *There is an algorithm that, given two $n \times n$ matrices A and B, compute $A \circledcirc B$ in $O(n^{(\omega+3)/2})$ time.*

We also need the following lemma, stating that the distance product of matrices with large entries can be approximated by several distance products of matrices with small entries:

▶ **Lemma 2.3** (Lemma 5.1 of [17]). *Suppose A, B are two matrices whose entries are in $[M]_0 \cup \{+\infty\}$, $C = A \star B$. Let R be a power of 2, and* Scale(A, M, R) *be a matrix A' such that*

$$A'_{ij} = \begin{cases} \lceil RA_{ij}/M \rceil & \text{if } 0 \le A_{ij} \le M \\ +\infty & \text{otherwise} \end{cases}.$$

Define C' as:

$$C' = \min_{\lfloor \log_2 R\rfloor \le r \le \lceil \log_2 M\rceil} \{(2^r/R)\cdot(\textsc{Scale}(A, 2^r, R) \star \textsc{Scale}(B, 2^r, R))\}$$

Then for any i, j, $C_{ij} \le C'_{ij} \le (1 + \frac{4}{R})C_{ij}$.

2.2 The distance/flow pairs and their operations

We introduce the notion of *df-pairs* in [12]. Our algorithm is based on manipulations of *df*-pairs and *df*-matrices.

▶ **Definition 2.4.** A *df-pair* is a pair (d, f) where d represents distance and f represents capacity or flow. We may assume $f \in [K]$.

A *df-set* is a set of *df*-pairs.

A *df-matrix* is a matrix whose entries are *df*-sets. If for every *df*-pair $(d, f) \in A_{ij}$, $d \in [R]_0$, then we say A is *within distance R*.

▶ **Definition 2.5** ([12]). The *merit* order $<_m$ is defined as $(d_1, f_1) >_m (d_2, f_2)$ iff $d_1 < d_2 \wedge f_1 \ge f_2$. For a *df*-set S, define $\mathcal{C}(S) = \{(d, f) : \nexists (d', f') \in S, (d', f') >_m (d, f)\}$.

▶ Remark. Given a *df*-set S sorted by distances, $\mathcal{C}(S)$ can be computed in $O(|S|)$ time.

We can see if $(d_1, f_1) >_m (d_2, f_2)$, then (d_2, f_2) is not useful since its distance is larger but flow is no larger. The operator $\mathcal{C}$ can be used to delete redundant elements of a *df*-set.

▶ **Definition 2.6** ([12]). For two *df*-sets, define their addition and multiplication as

$$S_1 + S_2 = \mathcal{C}(S_1 \cup S_2)$$
$$S_1 \cdot S_2 = \mathcal{C}(\{(d_1 + d_2, \min(f_1, f_2)) : (d_1, f_1) \in S_1, (d_2, f_2) \in S_2\}).$$

The multiplication of two *df*-sets S_1, S_2 can be understood as we try to link the paths pair-wisely in them, so the distance is the sum of their distances, and the flow is the minimum of their flows.

▶ **Definition 2.7.** For a *df*-set S, define $S(f) = \min\{d : (d, f') \in S, f' \geq f\}$. Similarly, for a *df*-matrix A, define $A(f)$ be the matrix whose entries are in $\mathbb{R} \cup \{+\infty\}$ such that $(A(f))_{ij} = A_{ij}(f)$.

For two *df*-sets S, S', we say $S = S'$ if $\forall f \in [K]$, $S(f) = S'(f)$. The relation $=$ is an equivalence relation. We say S' is a $(1+\epsilon)$-approximation of S, denoted as $S' \approx_\epsilon S$, if $S(f) \leq S'(f) \leq (1+\epsilon)S(f)$ for any $f \in [K]$. Similarly, for two *df*-matrices A, A' of size $n \times n$, say $A = A'$ if $A_{ij} = A'_{ij}$ for all $i, j \in [n]$, and say A' is a $(1+\epsilon)$-approximation of A, denoted as $A' \approx_\epsilon A$, if A'_{ij} is a $(1+\epsilon)$-approximation of A_{ij} for all $i, j \in [n]$.

▶ **Definition 2.8.** The *product* of two *df*-matrices A, B, denoted as $A \star B$, is defined as $(A \star B)_{ij} = \sum_{k\in[n]} A_{ik} \cdot B_{kj}$. We define $A^1 = A$ and $A^p = A^{p-1} \star A$ for integer $p > 1$.

Intuitively, this product can be understood in the following way: suppose we have a 3-layer graph $G = (V, E)$ where $V = \{u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n, w_1, w_2, \ldots, w_n\}$. Let A be the *df*-matrix representing edges among $\{u_i\}$ and $\{v_k\}$: for all $(d, f) \in A_{ik}$, there is an edge with length d and capacity f from u_i and v_k. Similarly let B be the *df*-matrix representing edges among $\{v_k\}$ to $\{w_j\}$. Let $C = A \star B$, then each element $(d, f) \in C_{ij}$ corresponds to a path from u_i to w_j with total length d and capacity f.

We have the following facts:

▶ **Fact 2.9.** *Let S_1, S_2 be two df-sets and $f \in [K]$. Then $(\mathcal{C}(S))(f) = S(f)$, $(S_1 + S_2)(f) = \min\{S_1(f), S_2(f)\}$ and $(S_1 \cdot S_2)(f) = S_1(f) + S_2(f)$.*

Proof. These are immediate from definition. ◀

▶ **Fact 2.10.** *For three df-matrices A, B, C and $\epsilon \geq 0$, $C \approx_\epsilon A \star B$ if and only if $C(f) \approx_\epsilon A(f) \star B(f)$ for any $f \in [K]$. In particular when $\epsilon = 0$, $C = A \star B$ iff $C(f) = A(f) \star B(f)$ for all $f \in [K]$.*

Proof. For all i, j, f, from Fact 2.9,

$$(A \star B)_{ij}(f) = (\sum_{k\in[n]} A_{ik} \cdot B_{kj})(f) = \min_{k\in[n]}\{A_{ik}(f) + B_{kj}(f)\} = (A(f) \star B(f))_{ij},$$

and

$$\begin{aligned}
&C \approx_\epsilon (A \star B)\\
\iff &\forall i, j, f,\ C_{ij}(f)/(A \star B)_{ij}(f) \in [1, 1+\epsilon]\\
\iff &\forall i, j, f,\ C_{ij}(f)/(A(f) \star B(f))_{ij} \in [1, 1+\epsilon]\\
\iff &\forall f,\ C(f) \approx_\epsilon A(f) \star B(f).
\end{aligned}$$

◀

▶ **Fact 2.11.** *Let A, B, C be df-matrices. If B is a $(1+\epsilon_1)$-approximation of A and C is a $(1+\epsilon_2)$-approximation of B, then C is a $(1+\epsilon_1)(1+\epsilon_2)$-approximation of A.*

Proof. For any i, j, f, $A_{ij}(f) \leq B_{ij}(f) \leq C_{ij}(f) \leq (1+\epsilon_2)B_{ij}(f) \leq (1+\epsilon_1)(1+\epsilon_2)A_{ij}(f)$. ◀

▶ **Fact 2.12.** *Let A, B be df-matrices and $\epsilon > 0$. If $B \approx_\epsilon A$, then $B \star B \approx_\epsilon A \star A$.*

Proof. For any i, j, f, from Fact 2.10, $(A \star A)(f) = A(f) \star A(f)$, and:

$$\begin{aligned}
(A \star A)(f)_{ij} =& \min_{k\in[n]}\{A(f)_{ik} + A(f)_{kj}\}\\
\leq& \min_{k\in[n]}\{B(f)_{ik} + B(f)_{kj}\} = (B \star B)(f)_{ij}\\
\leq& (1+\epsilon) \min_{k\in[n]}\{A(f)_{ik} + A(f)_{kj}\}\\
=& (1+\epsilon)(A \star A)(f)_{ij}.
\end{aligned}$$

◀

3 Main Algorithm

In this section we introduce an $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-2}\log W)$ algorithm for approximating APSP-AF problem within a stretch of $(1+\epsilon)$. The idea is not hard. Suppose M is the *df*-matrix corresponding to G, and $D = M^n$, then D is the exact *df*-matrix representing all-pair distances for all flows. To approximate D, we only need an algorithm that approximates the product of *df*-matrices by a stretch of $(1+\Theta(\epsilon/\log n))$. By Lemma 2.3, this can be done by exact *df*-matrix product algorithms that only handles distances no more than $R = O(\epsilon^{-1}\log n)$, which turns out to be expressible as $O(R^2)$ max-min products.

3.1 Exact product for small distances

The following algorithm shows that the product of two *df*-matrices A and B can be reduced to $O(R^2)$ max-min products if the distances are integers between 0 and R.

Algorithm 1 Exact product for two *df*-matrices.

function EXACT-PROD(A, B, R)
  $C_{ij} \leftarrow \varnothing$
  for d_0 in $[R]_0$ **do**
    $A_{ij}^{(d_0)} \leftarrow \max\{f : (d,f) \in A_{ij}, d \le d_0\}$
    $B_{ij}^{(d_0)} \leftarrow \max\{f : (d,f) \in B_{ij}, d \le d_0\}$
  for d_1, d_2 in $[R]_0^2$ **do**
    $C' \leftarrow A^{(d_1)} \mathbin{\textcircled{\vee}} B^{(d_2)}$
    $C_{ij} \leftarrow C_{ij} \cup \{(d_1+d_2, C'_{ij})\}$
  $C_{ij} \leftarrow \{\mathcal{C}(C_{ij})\}$; **return** C

▶ **Lemma 3.1.** EXACT-PROD *correctly returns the product $A \star B$ of two df-matrices A, B which are both within distance R.*

Proof. Let $C = \text{EXACT-PROD}(A, B, R)$. For any $i, j \in [n]$ and $f \in [K]$,

$$\begin{aligned}
&(A(f) \star B(f))_{ij} \\
=& \min_{k\in[n]}\{A(f)_{ik} + B(f)_{kj}\} \\
=& \min_{k\in[n]}\{\min\{d_1 : (d_1,f_1) \in A_{ik}, f_1 \ge f\} + \min\{d_2 : (d_2,f_2) \in B_{kj}, f_2 \ge f\}\} \\
=& \min_{d_1,d_2}\{d_1+d_2 : \exists k \in [n], A_{ik}^{(d_1)} \ge f, B_{kj}^{(d_2)} \ge f\} \\
=& \min_{d_1,d_2}\{d_1+d_2 : \left(A^{(d_1)} \mathbin{\textcircled{\vee}} B^{(d_2)}\right)_{ij} \ge f\} \\
=& \min\{d' : (d',f') \in C_{ij}, f' \ge f\} = (C(f))_{ij},
\end{aligned}$$

thus $C = A \star B$ by Fact 2.10. ◀

3.2 Approximate product for arbitrary distances

For two *df*-matrices A, B within distance M, we can compute an approximation of $A \star B$ by applying Lemma 2.3. Given a parameter R which is a power of 2, the following algorithm computes a $(1+\frac{4}{R})$-approximation of $A \star B$.

Algorithm 2 Approximate product of two *df*-matrices.

function SCALE(a, M, R)
  $a' = \begin{cases} \lceil R \cdot a/M \rceil & 0 \le a \le M \\ +\infty & \text{otherwise} \end{cases}$
  return a'
function APPROX-PROD(A, B, M, R)
  $C_{ij} \leftarrow \varnothing$
  for $r \leftarrow \lfloor \log_2 R \rfloor$ to $\lceil \log_2 M \rceil$ **do**
    $A^{(r)}_{ij} \leftarrow \mathcal{C}(\{(\text{SCALE}(d, 2^r, R), f) : (d, f) \in A_{ij}\})$
    $B^{(r)}_{ij} \leftarrow \mathcal{C}(\{(\text{SCALE}(d, 2^r, R), f) : (d, f) \in B_{ij}\})$
    $C^{(r)} = \text{EXACT-PROD}(A^{(r)}, B^{(r)}, R)$
    $C_{ij} \leftarrow C_{ij} \cup \{((2^r/R) \cdot d, f) : (d, f) \in C^{(r)}_{ij}\}$
  $C_{ij} \leftarrow \{\mathcal{C}(C_{ij})\}$; **return** C

▶ **Lemma 3.2.** *Let A, B be two df-matrices within distance M, and R a power of 2, $C =$ APPROX-PROD(A, B, M, R), then $C \approx_{\frac{4}{R}} A \star B$. Moreover, for each i, j, $|C_{ij}| = O(R \log M)$.*

Proof. Fix $f \in [K]$. Then $A^{(r)}(f) = \text{SCALE}(A(f), 2^r, R)$, $B^{(r)}(f) = \text{SCALE}(B(f), 2^r, R)$. By Lemma 3.1,

$$C(f) = \min_{\lfloor \log_2 R \rfloor \le r \le \lceil \log_2 M \rceil} \{(2^r/R) \cdot (A^{(r)}(f) \star B^{(r)}(f))\}.$$

From Lemma 2.3, $C(f) \approx_{\frac{4}{R}} A(f) \star B(f)$ for all f. The lemma is immediate from Fact 2.10. The "moreover" part holds since $|C_{ij}| \le \sum_{r=\lfloor \log_2 R \rfloor}^{\lceil \log_2 M \rceil} |C^{(r)}_{ij}| \le 2R\lceil \log_2 M \rceil$ for any i, j. ◀

3.3 Main procedure

Consider an instance G of APSP-AF problem, we represent G as a *df*-matrix A: A_{ij} is the set of all (d, f) such that there is an edge from i to j with length d and capacity f. Given $\epsilon > 0$, the following algorithm computes a *df*-matrix that approximates answers of APSP-AF problem within a stretch of $1 + \epsilon$.

Algorithm 3 the main procedure

function APSP-AF-APPROX(A, W, ϵ)
  $M \leftarrow nW$
  $R \leftarrow 4\lceil \log_2 n \rceil / \ln(1 + \epsilon)$
  $R \leftarrow 2^{\lceil \log_2 R \rceil}$
  $\tilde{D}^{(0)}_{ij} \leftarrow A_{ij}$
  $\tilde{D}^{(0)}_{ii} \leftarrow \{(0, K)\}$
  for $t = 1$ to $\lceil \log_2 n \rceil$ **do**
    $\tilde{D}^{(t)} \leftarrow \text{APPROX-PROD}(\tilde{D}^{(t-1)}, \tilde{D}^{(t-1)}, M, R)$
  return $\tilde{D}^{(\lceil \log_2 n \rceil)}$

▶ **Theorem 3.3.** *For a graph G, let A be the df-matrix representing G as above, D be the df-matrix such that $D(f)_{ij}$ is the shortest distance from i to j under flow constraint f in G, and $\tilde{D} =$ APSP-AF-APPROX(A, W, ϵ). Then $\tilde{D} \approx_\epsilon D$. Moreover,* APSP-AF-APPROX *runs in $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-2} \log W)$ time, and $\tilde{D}$ occupies $O(n^2\epsilon^{-1} \log n \log(nW))$ space.*

Proof. The time bottleneck of APSP-AF-Approx is $O(\log(nW))$ invocations of Exact-Prod, which runs in $\tilde{O}(n^{(\omega+3)/2}R^2)$ time. From Lemma 3.2, for each i, j, we have $|\tilde{D}_{ij}| = O(R\log(nW))$. Since $R = O(\epsilon^{-1}\log n)$, the "moreover" part is proved.

Let $D^{(t)}_{ij}(f)$ be the shortest distance from i to j under flow constraint f, where only paths of $\le 2^t$ edges are considered. Then $D^{(t)} = D^{(t-1)} \star D^{(t-1)}$, $\tilde{D}^{(t)} \approx_{\frac{4}{R}} \tilde{D}^{(t-1)} \star \tilde{D}^{(t-1)}$ (by Lemma 3.2), and $D^{(0)} = \tilde{D}^{(0)}$. We can prove by induction that $\tilde{D}^{(t)}$ is a $(1+\frac{4}{R})^t$-approximation of $D^{(t)}$. Base case ($t = 0$) is obvious; suppose this is true for t_0, let $\hat{D}^{(t_0+1)} = \tilde{D}^{(t_0)} \star \tilde{D}^{(t_0)}$, then $\hat{D}^{(t_0+1)}$ is a $(1+\frac{4}{R})^{t_0}$-approximation of $D^{(t_0+1)}$ by induction hypothesis and Fact 2.12. Since $\tilde{D}^{(t_0+1)} \approx_{\frac{4}{R}} \hat{D}^{(t_0+1)}$, $\tilde{D}^{(t_0+1)}$ is a $(1+\frac{4}{R})^{(t_0+1)}$-approximation of $D^{(t_0+1)}$ by Fact 2.11.

Since

$$\left(1+\frac{4}{R}\right)^{\lceil\log_2 n\rceil} \le \left(1+\frac{\ln(1+\epsilon)}{\lceil\log_2 n\rceil}\right)^{\lceil\log_2 n\rceil} \le 1+\epsilon,$$

$\tilde{D} = \tilde{D}^{(\lceil\log_2 n\rceil)}$ is a $(1+\epsilon)$-approximation of D. ◀

3.4 Computing witnesses

Our algorithm can be adapted to answer path queries in addition to distance queries: given $i, j \in [n]$ and $f \in [K]$, we not only find the approximated distance from i to j under flow constraint f, but also find an actual path with that distance. Suppose the path contains ℓ vertices, then a query takes $O(\ell \log\log_{1+\epsilon}(nW))$ time to report the whole path.

First we notice that the max-min product algorithm in [8] can be modified to return *witnesses* of max-min product:

▶ **Lemma 3.4** ([8]). *For two $n \times n$ matrices A, B, a pair of matrices (C, W) can be computed in $O(n^{(\omega+3)/2})$ time, where $C = A \otimes B$ and for any $i, j \in [n]$, suppose $k = W_{ij}$, then $C_{ij} = \min\{A_{ik}, B_{kj}\}$. We call W a* witness *of $A \otimes B$.*

We attach a node in the graph as additional information to every *df*-pair we encounter. For a *df*-pair representing a path p from i to j, suppose node x is attached to it, then x is on p. Moreover, if $x = i$ or $x = j$, then p contains at most one edge. More precisely:

- In Algorithm 1, we compute a witness matrix W besides computing $C' = A^{(d_1)} \otimes B^{(d_2)}$. Each *df*-pair $(d_1 + d_2, C'_{ij})$ in line 8 is attached with W_{ij};
- In Algorithm 2, line 10, the node attached with $((2^r/R)\cdot d, f)$ is the same as the node attached with (d, f);
- In Algorithm 3, line 6, the node attached with $(0, K) \in \tilde{D}^{(0)}_{ii}$ is i.

It is easy to see from above modifications that, suppose node x is attached with $(d, f) \in \tilde{D}_{ij}$, then there is a path from i to j with distance $\le d$, flow $\ge f$ which passes through node x.

We give a simple recursive algorithm that, based on nodes attached to *df*-pairs in the data structure, find the whole path for a query (u, v, f).

Algorithm 4 Path querying

```
1: function QUERY(u, v, f)
2:     (d', f') ← arg min{d' : (d', f') ∈ D̃_uv, f' ≥ f}
3:     x ← the node attached with (d', f')
4:     if x = u or x = v then
5:         return edge u → v
6:     else
7:         return QUERY(u, x, f)+QUERY(x, v, f)
```

4 Faster Implementation of Exact Product

In this section we show how to multiply two *df*-matrices within distance R in $\tilde{O}(n^{(\omega+3)/2}R^{3/2})$ time. This implies an $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-3/2}\log W)$ time algorithm for the original APSP-AF approximation problem.

The idea is to look into details of the max-min product algorithm in [8]. The task of computing max-min product of two matrices can be composed into $O(t)$ Boolean matrix multiplications and $O(n^3/t)$ extra work, so in our case we have $O(tR^2)$ matrix multiplications and $O(R^2n^3/t)$ extra work. However, these $O(tR^2)$ matrix multiplications can be expressed as $O(t)$ distance products of matrices whose elements are in $[R]$, thus can be accelerated by Zwick's algorithm for distance product [17] to run in $\tilde{O}(tRn^\omega)$ time.

▶ **Lemma 4.1** ([17]). *The distance product of two $n \times n$ matrices whose entries are in $[M]$ can be computed in $\tilde{O}(Mn^\omega)$ time.*

4.1 Row-balancing and column-balancing

The max-min product algorithm in [8] uses the concept of row-balancing and column-balancing. The function of row (column)-balancing is to rearrange the entries of a sparse matrix such that every row (column) has a moderate number of entries. We adapt this technique to *df*-matrices here.

▶ **Definition 4.2.** Let A be an $n \times n$ *df*-matrix where $m = \sum_{i,j\in[n]} |A_{ij}|$. We define the *row-balancing* of A, denoted by $\mathbf{rb}(A)$, as a pair of $n \times n$ *df*-matrices (A', A''), where each row of A' or A'' contains at most $k = \lceil m/n \rceil$ *df*-pairs in total. More precisely, we first sort all *df*-pairs in the i-th row of A by ascending order of f values, and partition them into blocks of size $\le k$: $T_i^1, T_i^2, \ldots, T_i^{a_i}$. The last block $T_i^{a_i}$ has *at most* k *df*-pairs, and other blocks have *exactly* k *df*-pairs. We define A' to contain all entries in the last block: $A'_{ij} = A_{ij} \cap T_i^{a_i}$. Since $\sum_{i\in[n]}(a_i - 1) \le m/k \le n$, we can rearrange all $T_i^j (1 \le j < a_i)$ into n distinct rows, by selecting an injection $\rho : \{(i, j) : i \in [n], j \in [a_i - 1]\} \to [n]$. Then we define $A''_{ij} = A_{i'j} \cap T_{i'}^q$ where $(i', q) = \rho^{-1}(i)$.

The *column-balancing* of A, denoted by $\mathbf{cb}(A)$, is a pair of $n \times n$ *df*-matrices (A', A'') such that $(A'^\top, A''^\top) = \mathbf{rb}(A^\top)$.

4.2 Dominance product

▶ **Definition 4.3.** The *dominance product* of two *df*-matrices A, B, denoted as $A \circledcirc B$, is defined as $(A \circledcirc B)_{ij} = \min\{d_1 + d_2 : (d_1, f_1) \in A_{ik}, (d_2, f_2) \in B_{kj}, f_1 \le f_2\}$.

There is also a "3-layer graph interpretation" of the dominance product of two *df*-matrices: consider a graph $G = (V, E)$ where $V = \{u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n, w_1, w_2, \ldots, w_n\}$. Let

A be the *df*-matrix representing edges among $\{u_i\}$ and $\{v_k\}$: for all $(d, f) \in A_{ik}$, there is an edge with length d and capacity f from u_i and v_k. Similarly let B be the *df*-matrix representing edges among $\{v_k\}$ to $\{w_j\}$. Let $C = A \circledcirc B$, then C_{ij} is the minimum distance of a path from u_i to w_j, where it's required that the capacity of the first edge is no more than the capacity of the second edge.

▶ **Lemma 4.4.** *Given two df-matrices A, B within distance R, let $m_1 = \sum_{i,j\in[n]} |A_{ij}|$, $m_2 = \sum_{i,j\in[n]} |B_{ij}|$. The dominance product $A \circledcirc B$ can be computed in $\tilde{O}(m_1 m_2/n + Rn^\omega)$ time.*

Proof. Let $(A', A'') = \mathbf{cb}(A)$, the column-balancing of A. Define two (scalar) matrices $\tilde{A}$ and $\tilde{B}$:

$$\tilde{A}_{ik} = \min\{d : (d, f) \in A''_{ik}\}, \tilde{B}_{kj} = \min\{d : (d, f) \in B_{k'j}, f \geq \text{max flow of } T^{q'}_{k'}\},$$

where $(k', q') = \rho^{-1}(k)$. Note that $(\tilde{A} \star \tilde{B})_{ij}$ is the smallest $d_1 + d_2$ such that:

- $(d_1, f_1) \in A_{ik'}$, $(d_2, f_2) \in B_{k'j}$;
- $(d_1, f_1) \in T^{q'}_{k'}$, $q' < a_{k'}$, and f_2 dominates all flows in $T^{q'}_{k'}$.

What we haven't considered are pairs $(d_1, f_1) \in A_{ik'}$ and $(d_2, f_2) \in B_{k'j}$ such that (d_1, f_1) is contained in the largest part of column k', or f_2 is smaller than the largest flow in $T^{q'}_{k'}$, that is, f_1 and f_2 locate in the same "block". In either cases, for any (d_2, f_2) there are only $O(m_1/n)$ entries in A' or A'' to compare. The time complexity for distance product is $\tilde{O}(Rn^\omega)$, and the time complexity for the rest pairs are $O(m_1 m_2/n)$. ◀

4.3 Faster exact product

▶ **Theorem 4.5.** *Given two df-matrices A, B within distance R, their product $A \star B$ can be computed in $\tilde{O}(n^{(\omega+3)/2} R^{3/2})$ time.*

Proof. Let $D^{(d)}_{ij} = \max\{f_1 : d_1 + d_2 = d, f_1 \leq f_2, (d_1, f_1) \in A_{ik}, (d_2, f_2) \in B_{kj}\}$ and $D'^{(d)}_{ij} = \max\{f_2 : d_1 + d_2 = d, f_1 \geq f_2, (d_1, f_1) \in A_{ik}, (d_2, f_2) \in B_{kj}\}$, then $(A \star B)_{ij} = \mathcal{C}\left(\bigcup_d \left\{(d, \max\{D^{(d)}_{ij}, D'^{(d)}_{ij}\})\right\}\right)$. We show $D^{(d)}$ for all $0 \leq d \leq 2R$ can be computed in $\tilde{O}(n^{(\omega+3)/2} R^{3/2})$ total time; $D'^{(d)}$ is similar.

Sort all *df*-pairs of A and B by their f values in increasing order, and then partition this sorted list into t parts $L_1, L_2, \ldots, L_t$, where each part has $O(n^2 R/t)$ elements. Let $A^{(r)}_{ij} = A_{ij} \cap L_r$, $B^{(r)}_{ij} = B_{ij} \cap L_r$ for $1 \leq r \leq t$. For each r we let $(A'^{(r)}, A''^{(r)}) = \mathbf{rb}(A^{(r)})$, and compute $A^{(r)} \circledcirc B$, $A'^{(r)} \circledcirc B$, $A''^{(r)} \circledcirc B$. For each d, i, j, we can compute $D^{(d)}_{ij}$ as follows:

1. We first determine which part $D^{(d)}_{ij}$ lies in, *i.e.* find the largest r such that $(A^{(r)} \circledcirc B)_{ij} \leq d$;
2. If $(A'^{(r)} \circledcirc B)_{ij} \leq d$, then we only need to consider k where $A'^{(r)}_{ik} \neq \varnothing$ to determine $D^{(d)}_{ij}$;
3. Otherwise, we find the largest q such that $(A''^{(r)} \circledcirc B)_{\rho(i,q),j} \leq d$, and determine $D^{(d)}_{ij}$ by looking through the q-th part of i-th row of $A^{(r)}$.

The above procedure takes $O(nR/t)$ time for each d, i, j. We compute $A^{(r)} \circledcirc B$ in $\tilde{O}(n^3 R^2/t^2 + Rn^\omega)$ time as follows. Let $f' = \min\{f : (d, f) \in L_{r+1}\}$, then $A^{(r)} \circledcirc B = \min\{A^{(r)} \circledcirc B^{(r)}, A^{(r)}(0) \star B(f')\}$. The $A^{(r)} \circledcirc B^{(r)}$ part considers comparisons inside L_r, which we compute in $\tilde{O}((n^2 R/t)^2/n + Rn^\omega)$ time by Lemma 4.4; the $A^{(r)}(0) \star B(f')$ part represents comparisons between L_r and $\bigcup_{r<r'\leq t} L_{r'}$, and is simply a distance product computable in $\tilde{O}(Rn^\omega)$ time. We can compute $A'^{(r)} \circledcirc B$ and $A''^{(r)} \circledcirc B$ similarly as we did in $A^{(r)} \circledcirc B$. Therefore, we spend $\tilde{O}(n^3 R^2/t + tRn^\omega)$ time in total for all $1 \leq r \leq t$. The theorem follows by setting $t = n^{(3-\omega)/2} R^{1/2}$. ◀

4.4 Computing witnesses

We modify this algorithm to support path queries as in Section 3.4. We call $k \in [n]$ is a witness of $D^{(d)}_{ij}$ if there is $(d_1, f_1) \in A_{ik}$, $(d_2, f_2) \in B_{kj}$ such that $d_1 + d_2 = d$, $f_1 \le f_2$ and $D^{(d)}_{ij} = f_1$. For any d, i, j, a witness of $D^{(d)}_{ij}$ can be found by step 2,3 when computing $D^{(d)}_{ij}$. Similarly we can find witnesses of $D'^{(d)}_{ij}$. For each d, i, j, if $D^{(d)}_{ij} \ge D'^{(d)}_{ij}$, attach the witness of $D^{(d)}_{ij}$ to the df-pair $(d, D^{(d)}_{ij})$; otherwise attach the witness of $D'^{(d)}_{ij}$ to the df-pair $(d, D'^{(d)}_{ij})$. The time complexity remains the same.

5 Conclusions

Our work shows that the apBLSP and APSP-AF problem can be approximated within a stretch of $(1+\epsilon)$ in $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-3/2}\log W)$ time. Also, a faster algorithm for max-min product would imply a faster algorithm for approximating APSP-AF problem. In this sense, since approximating APSP-AF is at least as hard as max-min product, our algorithm is optimal up to $\tilde{O}(\mathrm{poly}(\epsilon^{-1})\log W)$ factors.

We think the main open problem left by our work is to further improve the dependence on ϵ. Can the APSP-AF problem be approximated in, say, $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-1}\log W)$ time? There is an $\epsilon^{-1/2}$ gap here, and we might need a more refined approach to fill this gap.

References

1. Giorgio Ausiello, Giuseppe F Italiano, Alberto Marchetti Spaccamela, and Umberto Nanni. Incremental algorithms for minimal length paths. *Journal of Algorithms*, 12(4):615–638, 1991.
2. Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *Journal of Algorithms*, 62(2):74–92, 2007.
3. Aaron Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. *SIAM Journal on Computing*, 45(2):548–574, 2016.
4. Prosenjit Bose, Anil Maheshwari, Giri Narasimhan, Michiel Smid, and Norbert Zeh. Approximating geometric bottleneck shortest paths. *Computational Geometry*, 29(3):233–249, 2004.
5. Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.
6. Camil Demetrescu and Giuseppe F Italiano. Dynamic shortest paths and transitive closure: Algorithmic techniques and data structures. *Journal of Discrete Algorithms*, 4(3):353–383, 2006.
7. Ran Duan and Seth Pettie. Bounded-leg distance and reachability oracles. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 436–445. Society for Industrial and Applied Mathematics, 2008.
8. Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Twentieth Acm-Siam Symposium on Discrete Algorithms, SODA 2009, New York, Ny, Usa, January*, pages 384–391, 2009.
9. Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30. ACM, 2015.
10. Daniel P Martin. Dynamic shortest path and transitive closure algorithms: A survey. *arXiv preprint arXiv:1709.00553*, 2017.

11 Liam Roditty and Michael Segal. On bounded leg shortest paths problems. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 775–784. Society for Industrial and Applied Mathematics, 2007.

12 Tong-Wook Shinn and Tadao Takaoka. Efficient graph algorithms for network analysis. In *First International Conference on Resource Efficiency in Interorganizational Networks-ResEff 2013*, page 236, 2013.

13 Tong-Wook Shinn and Tadao Takaoka. Combining all pairs shortest paths and all pairs bottleneck paths problems. In *Latin American Symposium on Theoretical Informatics*, pages 226–237. Springer, 2014.

14 Tong-Wook Shinn and Tadao Takaoka. Combining the shortest paths and the bottleneck paths problems. In *Proceedings of the Thirty-Seventh Australasian Computer Science Conference-Volume 147*, pages 13–18. Australian Computer Society, Inc., 2014.

15 Tong-Wook Shinn and Tadao Takaoka. Variations on the bottleneck paths problem. *Theoretical Computer Science*, 575:10–16, 2015. Special Issue on Algorithms and Computation.

16 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All-pairs bottleneck paths for general graphs in truly sub-cubic time. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 585–589. ACM, 2007.

17 Uri Zwick. All pairs shortest paths in weighted directed graphs – exact and almost exact algorithms. In *Foundations of Computer Science, 1998. Proceedings. Symposium on*, pages 310–319, 1998.