Improved Time Bounds for All Pairs Non-decreasing Paths in General Digraphs

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

Yong Gu
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
guyong12@mails.tsinghua.edu.cn

Le Zhang
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
le-zhang12@mails.tsinghua.edu.cn

Abstract

We present improved algorithms for solving the All Pairs Non-decreasing Paths (APNP) problem on weighted digraphs. Currently, the best upper bound on APNP is $\tilde{O}(n^{(9+\omega)/4}) = O(n^{2.844})$, obtained by Vassilevska Williams [TALG 2010 and SODA'08], where $\omega < 2.373$ is the usual exponent of matrix multiplication. Our first algorithm improves the time bound to $\tilde{O}(n^{2+\omega/3}) = O(n^{2.791})$. The algorithm determines, for every pair of vertices s, t, the minimum last edge weight on a non-decreasing path from s to t, where a non-decreasing path is a path on which the edge weights form a non-decreasing sequence. The algorithm proposed uses the combinatorial properties of non-decreasing paths. Also a slightly improved algorithm with running time $O(n^{2.78})$ is presented.

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

Keywords and phrases Graph algorithms, Matrix multiplication, Non-decreasing paths

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.44

Funding This work was partially supported by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61361136003.

Acknowledgements The authors thank the anonymous reviewers for the constructive comments.

1 Introduction

Given a digraph with arbitrary real weights, a non-decreasing path is a path on which the edge weights form a non-decreasing sequence. Two of the problems studied on non-decreasing paths are the *Single Source Non-decreasing Paths* (SSNP) problem and the *All Pairs Non-decreasing Paths* (APNP) problem. The problem of SSNP was first studied by Minty [13]. The motivation is a train scheduling problem, as reviewed in [22]. Every train stop is mapped to a vertex. A train from stop v_1 with departure time t_1 to stop v_2 with arrival time t_2 is mapped to a vertex v with two edges (v_1, v), (v, v_2), of which the weights are t_1, t_2 resp. Now a trip from s to t is possible only when there exists a non-decreasing path from s to t in the

[1] 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. 44; pp. 44:1–44:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

constructed digraph. As said in [22], for SSNP a folklore modification of Dijkstra's algorithm [5], implemented using Fibonacci heaps [8], gives the running time of $O(m + n \log n)$, where m, n are the number of edges and vertices resp. In the word RAM model the first linear-time algorithm was given by Vassilevska Williams [22]. With a slight modification, the algorithm also runs in $O(m \log \log n)$ time in the standard addition-comparison model.

A restriction of APNP in vertex-weighted digraphs is computationally equivalent to the problem of *Maximum Witness for Boolean Matrix Multiplication* (MWBMM) [22]. (Note that the complexity of computing MWBMM is at least $\Omega(n^\omega)$ [17].) An algorithm of $O(n^{2+\mu})$ time for the latter problem was given by Czumaj et al. [3], where μ satisfies the equation $\omega(1, \mu, 1) = 1 + 2\mu$ and $\omega(1, \mu, 1)$ is the exponent of the multiplication of an $n \times n^\mu$ matrix by an $n^\mu \times n$ matrix. Currently, the best available bounds on $\omega(1, \mu, 1)$ by Le Gall and Urrutia [11] imply that $\mu < 0.5286$. The first truly sub-cubic algorithm for edge-weighted APNP was also presented in [22]. The algorithm originally runs in $\tilde{O}(n^{(15+\omega)/6}) = O(n^{2.896})$ time[2] based on an $O(n^{2+\omega/3})$-time $(\min, \leq)$-product ($\min_k\{B[k, j] \mid A[i, k] \leq B[k, j]\}$ for $(\min, \leq)(A, B)$) algorithm from [21], where ω is the exponent of square matrix multiplication. The best upper bound on ω is currently $\omega < 2.373$ [12, 23]. By using a faster $O(n^{(3+\omega)/2})$-time algorithm from [6] for $(\min, \leq)$-product, the result can be improved to $\tilde{O}(n^{(9+\omega)/4})$ as indicated in the abstract. These two algorithms for $(\min, \leq)$-product are from [21] and [6] resp. The faster algorithm of [6] utilizes a simple technique called *row-balancing*, which we introduce in details in Section 2.

A closely related problem of APNP is the *All Pairs Bottleneck Paths* (APBP) problem, where the bottleneck weight of a path is the smallest weight of an edge on the path. Intuitively for a digraph with non-negative edge weights, APBP determines, for every pair of vertices s, t, the maximum amount of flow that can be routed from s to t along any single path. As indicated in [22], APNP is at least as hard as APBP. We briefly explain it here. Consider an $O(n^{3-\delta})$-time algorithm for APNP with $0 < \delta \leq 1$. Now to compute the $(\min, \leq)$-product of two $n \times n$ matrices A, B, a tripartite digraph $G' = (V_1 \cup V_2 \cup V_3, E')$ can be constructed as follows. The edge from $i \in V_1$ to $k \in V_2$ is represented by $A[i, k]$; the edge from $k \in V_2$ to $j \in V_3$ is similarly represented by $B[k, j]$. Hence the $(\min, \leq)$-product is solved in $O(n^{3-\delta})$ time. The $(\max, \min)$-product ($\max_k \min\{A[i, k], B[k, j]\}$ for $(\max, \min)(A, B)$) is a combination of two variants of $(\min, \leq)$-product. Therefore it can also be computed in $O(n^{3-\delta})$ time. If $(\max, \min)$-product is computable in $O(n^{3-\delta})$ time, then APBP is computable in $O(n^{3-\delta})$ time, as $(\mathbb{R}, \min, \max, \infty, -\infty)$ is a closed semiring [21]. There are several results on the APBP problem. On vertex-weighted digraphs, Shapira et al. [17] showed that APBP can be solved in $O(n^{2+\mu})$ time. They also demonstrated the computational equivalence (up to constant factors) among vertex-weighted APBP and several other problems. The forementioned problem of MWBMM is one of them. The first truly sub-cubic algorithm for edge-weighted APBP was given by Vassilevska et al. [21], with the running time of $O(n^{2+\omega/3})$. Later an improved algorithm was proposed in [6], which runs in $O(n^{(3+\omega)/2})$ time. There are several variants of APBP. One is the *All Pairs Bottleneck Shortest Paths* (APBSP) problem, which for every pair of vertices u, v, determines a path with the maximum bottleneck weight among all the shortest paths from u to v. The shortest paths here are measured w.r.t. the unweighted distances. On edge-capacitated digraphs, Vassilevska et al. [21] gave an algorithm for APBSP with the running time $\tilde{O}(n^{(15+\omega)/6})$, which was improved to $\tilde{O}(n^{(3+\omega)/2})$ in [6]. On vertex-capacitated digraphs, Shapira et al. [17] presented an $\tilde{O}(n^{(8+\mu)/3})$-time algorithm. Also the result was later improved by [6].

[2] Here $\tilde{O}(\cdot)$, as usual, hides poly-logarithmic factors.

Fast matrix multiplication algorithms have numerous applications in other graph problems as well. We list here only a subset of them, which includes finding a maximum triangle in vertex-weighted graphs [19, 20, 4], to obtain the All Pairs Shortest Paths [9, 10, 16, 18, 1, 25, 7], finding minimum weight cycles in directed or undirected graphs with integral edge weights [14, 24, 2, 15].

Our Results: We give faster algorithms for solving APNP on digraphs. The results are listed in Theorem 1, which follows directly from Theorem 18 and Theorem 21.

▶ **Theorem 1.** *Let $G = (V, E, w)$ be a real edge-weighted digraph. There exists a* deterministic *algorithm which solves the problem of APNP in $\tilde{O}(n^{2+\omega/3}) = O(n^{2.791})$ time. There also exists another slightly faster* deterministic *algorithm which runs in $O(n^{2.78})$ time using rectangular matrix multiplications.*

A High-level Overview: The problem of APNP can be solved by running (essentially) $(n-1)$ steps of the $(\min, \leq)$-product of the adjacency matrix of the input digraph. We utilize the fact that the adjacency matrix in the computation is always *fixed*, and therefore when the rows of it all have a bounded number of $(< \infty)$ entries, we find a simple (and faster) alternative to the repeated applications of the $(\min, \leq)$-product. To make use of this simple alternative, we partition the input digraph into many sparse subgraphs, and compute APNP by considering the edges one subgraph by one subgraph. However, even in a sparse subgraph the number of $(< \infty)$ entries in a row might be still as large as $\Omega(n)$. Thus we further classify a row as type *low* or *high*, where a high row corresponds to a high out-degree vertex in the subgraph. The simple alternative is used to replace the $(\min, \leq)$-product for the submatrix consisting of low rows. Also it computes the portion of a non-decreasing path until the *first* vertex of high out-degree. The remaining portion from this high out-degree vertex to the destination can be constructed by the relevant queries on a slightly modified data structure from [22]. However, to get an efficient algorithm we should first make sure of the queries worthy of asking, which is actually a weaker problem of the existence of non-decreasing paths.

The paper is organized as follows. We introduce the preliminaries required in the next section. Then in Section 3 we propose an algorithm for a simple case, which is also a sub-routine of the improved algorithms presented in Section 4.

2 Preliminaries

Given a real edge-weighted digraph $G = (V, E, w)$, where $w : E \to \mathbb{R}$ is a weight function defined on its edges, the All-Pairs Non-decreasing Paths (APNP) problem asks to determine, for every pair of vertices u, v, the minimum last edge weight on a non-decreasing path from u to v (∞ if such a path does not exist). Typically the output is in tabular form, i.e. a matrix R with $R[i, j]$ corresponding to the minimum last edge weight of a non-decreasing path from u_i to u_j. Also conventionally the entries $R[i, i]$'s are set as $-\infty$ [22]. The matrix R is called the "APNP matrix" of G. Given a matrix A, the i-th row, the j-th column of A are denoted as $A[i, \cdot]$, $A[\cdot, j]$ resp. Also for two matrices A, B of the same size, the entry-wise minimum of them is denoted as $\min(A, B)$. Given a path p in G, if vertex k_1 appears earlier on p than k_2, then the portion of p from k_1 to k_2 is denoted as $p[k_1, k_2]$. Notations $p(k_1, k_2]$, $p[k_1, k_2)$, $p(k_1, k_2)$ represent $p[k_1, k_2]$ excluding k_1, k_2, both k_1 and k_2 resp. The following special matrix products are used in later algorithms.

▶ **Definition 2** (Various Products). Given two $n \times n$ matrices A and B over a totally ordered set, the dominance product $A \circledS B$ is defined as

$$(A \circledS B)[i,j] = |\{k \mid A[i,k] \leq B[k,j]\}|.$$

The $(\min, \leq)$-product $A \circledcirc B$ is defined as

$$(A \circledcirc B)[i,j] = \begin{cases} \min_k\{B[k,j] \mid A[i,k] \leq B[k,j]\} & \text{if } \exists k, A[i,k] \leq B[k,j], \\ \infty & \text{otherwise.} \end{cases}$$

Next we review a simple technique called *row-balancing* [6]. Basically by row-balancing, a matrix is decomposed into a sparse matrix and a dense one, where the finite entries of the dense matrix are uniformly re-distributed across the rows.

▶ **Definition 3** ([6] Row-Balancing). Let A be an $n \times p$ matrix with m finite elements. Depending on context, the other elements will either all be ∞ or all be $-\infty$. We assume the former below. The row-balancing of A, or $\mathbf{rb}(A)$, is a pair (A', A'') of $n \times p$ matrices, each with at most $k = \lceil m/n \rceil$ elements in each row. The row-balancing is obtained by the following procedure: First, sort all the finite elements in the i-th row of A in non-increasing or non-decreasing order depending on context , and divide this list into several parts $T_i^1, T_i^2, \ldots, T_i^{a_i}$ such that all parts except the last one contain k elements and the last part $(T_i^{a_i})$ contains at most k elements. Let A' be the submatrix of A containing the last parts:

$$A'[i,j] = \begin{cases} A[i,j] & \text{if } A[i,j] \in T_i^{a_i}, \\ \infty & \text{otherwise.} \end{cases}$$

Since the remaining parts have exact k elements, there can be at most $m/k \leq n$ of them. We assign each part to a distinct row in A'', i.e., we choose an arbitrary mapping $\rho : [n] \times [p/k] \to [n]$ such that $\rho(i,q) = i'$ if T_i^q is assigned row i'; it is undefined if T_i^q doesn't exist. Let A'' be defined as:

$$A''[i',j] = \begin{cases} A[i,j] & \text{if } \rho^{-1}(i') = (i,q) \text{ and } (i,j) \in T_i^q, \\ \infty & \text{otherwise.} \end{cases}$$

Thus, every finite $A[i,j]$ in A has a corresponding element in either A' or A'', which is also in the j-th column. The column-balancing of A, or $\mathbf{cb}(A)$, is similarly defined as (A'^T, A''^T), where $(A', A'') = \mathbf{rb}(A^T)$.

This simple technique is very useful for computing $A \circledS B$, $A \circledcirc B$, $(\max, \min)(A, B)$, and several new hybrid products defined in [6]. Below is a theorem from [6], which shows how to compute $A \circledS B$ using $\mathbf{cb}(A)$.

▶ **Theorem 4** ([6] Sparse Dominance Product). *Let A and B be two $n \times n$ matrices where the number of non-(∞) values in A is m_1 and the number of non-$(-\infty)$ values in B is m_2. Then $A \circledS B$ can be computed in time $O(m_1 m_2/n + n^\omega)$.*

There is a symmetric problem of computing non-decreasing paths with maximum first edge weights. Note that the maximum first edge weight on a non-decreasing path from i to i is defined as ∞, and for $i \neq j$, if no non-decreasing path exists, the maximum first edge weight is defined as $-\infty$. The time complexity of the single source version is given below.

▶ **Theorem 5** ([22] Maximum First Edge Weight). *Given a digraph G with n vertices, m edges, and a vertex s, there exists an algorithm which in $O(m \log n)$ time outputs the maximum first edge weight on a non-decreasing path from s to v for every $v \in V$.*

We need a slightly modified auxiliary data structure from [22] to efficiently compute a non-decreasing path with the minimum last edge weight starting from a subset of the out-edges of the source vertex. A paraphrased proof is given below.

▶ **Theorem 6.** *Given a digraph $G = (V, E, w)$ with n vertices, m edges, and a vertex s, there exists an algorithm which in $O(m \log n)$ time constructs a balanced binary search tree $T(t)$ for every $t \in V$. With $T(t)$, given a weight value w', the algorithm can determine in $O(\log n)$ time the minimum last edge weight of a non-decreasing path from s to t, starting from any out-edge e of s with $w(e) \geq w'$.*

Proof. For every vertex v, add an attribute $d[v]$ and a list $L(v)$, where $d[s] = -\infty$, $d[v] = \infty$ if $v \neq s$, and $L(v) = \varnothing$ initially. Next start a search resembling DFS from s, where differently the edges of s are explored in *reverse* sorted order, and the search only follows non-decreasing paths. Also for every edge (u, v), once explored, we first run $d[v] \leftarrow \min\{d[v], w(u, v)\}$ and then remove (u, v) from G. Lastly, vertices can be repeatedly visited. Consider the time when this recursive search backtracks to the initial search at s. The algorithm will explore the next *unexplored* out-edge of s in the reverse sorted order. We can thus partition the search into different phases, each of which corresponds to the search starting from an unexplored out-edge of s until the algorithm backtracks to s. Consider the end of a specific phase corresponding to an out-edge e of s. For every v of which $d[v]$ becomes *strictly* smaller in this phase, append $(w(e), d[v])$ to $L(v)$. At the end of the whole search, for every v, scan $L(v)$, where for consecutive elements of equal $w(e)$, only the last one is retained. Next transform $L(v)$ into a balanced binary search tree $T(v)$ keyed by $w(e)$.

Given a weight value w', the value required is the $d[v]$ attribute of the predecessor found. ◀

We last review a standard technique called *bridging set*, which is repeatedly used in the literature [6, 22, 25, 17]. The following lemma is one from [22]. The set of size $\frac{n \log n}{L}$ constructed is an L-bridging set.

▶ **Lemma 7** ([22, 25])**.** *Given a collection of N subsets of $\{1, \ldots, n\}$, each of size L, one can find in deterministic $O(NL)$ time a set of $\frac{n \log n}{L}$ elements of $\{1, \ldots, n\}$ hitting every one of the subsets.*

Model of Computation: We use the standard addition-comparison computational model. The only operations performed on real numbers are comparisons in this paper.

3 Warm Up: A Simple Case

As a warm up for our main algorithms, we consider a simple algorithm, which is efficient if the out-edges are uniformly distributed across the vertices. The main purpose is to give an idea of how the combinatorial properties of non-decreasing paths are utilized, and the difficulty of extending the algorithm to the general case.

Given a digraph $G = (V, E, w)$, let $L(E)$ be a sorted list of the edges in E. We evenly divide $L(E)$ into t parts such that each part has at most $\lceil n^2/t \rceil$ edges. Each part corresponds to a subgraph of G. Therefore there are t subgraphs. Denote them as G_r for $1 \leq r \leq t$. The edge weights in G_{r-1} is no greater than those in G_r.

Consider the adjacency matrix A_j of G_j. By saying a *simple* case, we mean the number of $(< \infty)$ elements of any row in any A_j is bounded by d, which has $n/t \leq d \leq n$. This assumption is *only* used in Theorem 15 later. We use the notation $G_{\leq r}$ to represent the

subgraph induced by the ($< \infty$) entries of A_j for $1 \leq j \leq r$. The notation $G_{<r}$ can be inferred similarly. The proposed algorithm consists of t iterations. In the r-th iteration, the APNP matrix of $G_{<r}$ is extended to the APNP matrix of $G_{\leq r}$. Consider the APNP matrix R of $G_{<r}$. Except a technical issue of edges of equal weights straddling across different A_r's (which is handled later in this section), one can verify that by running $R \leftarrow \min(R, R \otimes A_r)$ for $(n-1)$ steps, the matrix R will be the APNP matrix of $G_{\leq r}$. Intuitively by n' steps of $R \leftarrow \min(R, R \otimes A_r)$, the algorithm considers all the non-decreasing paths containing at most n' edges of G_r. The combinatorial properties of non-decreasing paths bring the following observations in the extension from $G_{<r}$ to $G_{\leq r}$.

The APNP matrices of $G_{<r}$, $G_{\leq r}$ are denoted as R, R' resp. If $R[i,j] < \infty$, then the non-decreasing path in G from i to j with the minimum last edge weight is already computed, for existing non-decreasing paths with minimum last edge weights cannot be improved by introducing edges of no smaller weights.

▶ **Observation 8** (An Entry Only Computed Once)**.** If $R[i,j] < \infty$, then $R'[i,j] = R[i,j]$.

The above observation indicates that the exact value of an entry of the APNP matrix is computed in at most one extension among all the extensions.

By allowing non-decreasing paths with more edges of G_r, minimum last edge weights of non-decreasing paths never become larger. Therefore we have the following observation.

▶ **Observation 9** (Non-increasing of Entries)**.** In the process of the $(n-1)$ steps of $R \leftarrow \min(R, R \otimes A_r)$, the entry $R[i,j]$ is non-increasing.

Generally, given a non-decreasing path p from s to t with the minimum last edge weight, we can only claim subpaths starting from s (*prefixes*) can be replaced with non-decreasing paths with minimum last edge weights. However, as shown below, except the technicality on equal weights handled later, the claim also holds for certain other subpaths.

▶ **Observation 10** (Two Parts of A Path)**.** Consider a non-decreasing path p in $G_{\leq r}$ with the minimum last edge weight. W.l.o.g. the path p can be split into two portions p_1, p_2 lying within $G_{<r}$, G_r resp. Any prefix of p_2 can be replaced with a non-decreasing path in G_r with the minimum last edge weight.

We use the following definition to capture the entries of interest in R'.

▶ **Definition 11.** An entry $R'[i,j]$ is *new* w.r.t. R if $R'[i,j] < \infty$ but $R[i,j] = \infty$. Otherwise, it is *old*.

Use n_r to denote the number of new entries of the APNP matrix of $G_{\leq r}$ w.r.t. the APNP matrix of $G_{<r}$. The following observation is then obvious.

▶ **Observation 12** (Bounded Number of New Entries)**.** $\sum_{1 \leq r \leq t} n_r \leq n^2$.

We also need a different view of computing $C = \min(A, A \otimes B)$, where A, B are $n \times n$ matrices. The matrix C is the entry-wise minimum of A and the n matrices $A[\cdot, k] \otimes B[k, \cdot]$ for $1 \leq k \leq n$. An algorithm for computing C using this view is given in Table 1. Now we present the intuition of the algorithm for the simple case.

The Intuition: The matrices R, R' are defined as before. Among all the non-decreasing paths from i to j in $G_{\leq r}$ with the minimum last edge weight, consider one path p with the least number of edges in G_r. As in Observation 10 (Two Parts of A Path), w.l.o.g. p is a concatenation of subpaths $p[i,k_1]$, $p[k_1,j]$. Subpaths $p[i,k_1]$, $p[k_1,j]$ are within $G_{<r}$, G_r resp. The entry $R'[i,k_1]$ is old, whereas the entries $R'[i,k']$'s are new for $k' \in p(k_1,j)$

Algorithm 1 An algorithm for computing $C = \min(A, A \otimes B)$.

(S1) Initialize C as A. Construct a sorted list $L(B[k,\cdot])$ of the $(< \infty)$ elements of $B[k,\cdot]$ in non-increasing order, for every k.
(S2) For every $A[\cdot,k]$, and for every $A[i,k]$ within:
  (S21) Scan $L(B[k,\cdot])$ from head to tail until the first element of $L(B[k,\cdot])$ which is less than $A[i,k]$.
  (S22) $C[i,j] \leftarrow \min\{C[i,j], B[k,j]\}$ for every $B[k,j]$ scanned, excluding the first element of $L(B[k,\cdot])$ which is less than $A[i,k]$.

due to p having the least number of edges in G_r. Therefore during the $(n-1)$ steps of $R \leftarrow \min(R, R \otimes A_r)$, the old entries effectively change R *only* in the first step. The changes of R in the later $(n-2)$ steps are contributed only by the *new* entries. In the view of the algorithm in Table 1, the entry (i,k') of R is only compared with the k'-th row of A_r. As the number of $(<\infty)$ elements of the k'-th row of A_r is assumed to be bounded, the cost brought by all the new entries is bounded due to Observation 9 (Non-increasing of Entries) and Observation 12 (Bounded Number of New Entries).

Besides the algorithm in Table 1, we also need an "ordinary" way to compute $R \otimes A_r$ as shown below.

▶ **Lemma 13** (The First Edge). *Given the APNP matrix R of $G_{<r}$, the product $R \otimes A_r$ can be computed in $\tilde{O}(n^\omega + n_r \cdot \frac{n}{t})$ time.*

Proof. According to Observation 8 (An Entry Computed Only Once), only the new entries of $R \otimes A_r$ w.r.t. R are of concern. To get these, we first determine the set S of (i,j)'s with $(R \otimes A_r)[i,j] < \infty$ but $R[i,j] = \infty$. Again to determine whether $(R \otimes A_r)[i,j] < \infty$, we only need to compute $R \circledcirc A_r$, where a slight difference is that here only the $(<\infty)$ entries of A_r are considered. As the $(<\infty)$ entries of R are no greater than those of A_r, the product $R \circledcirc A_r$ is a matrix product of two Boolean matrices corresponding to the $(<\infty)$ entries of R, A_r resp., which is computable in $O(n^\omega)$ time.

To determine the exact value of $(R \otimes A_r)[i,j] < \infty$, we use an idea similar to one in [6]. Let $(A'_r, A''_r) = \mathbf{cb}(A_r)$. Compute $R \circledcirc A'_r$ and $R \circledcirc A''_r$ in $O(n^\omega)$ time. The value of $(R \otimes A_r)[i,j]$ is $R[i,\cdot] \otimes A_r[\cdot,j]$. In $\mathbf{cb}(A_r)$, the column $A_r[\cdot,j]$ is divided into a_j parts $T_j^1, \ldots, T_j^{a_j}$. To determine the right q' where $(R \otimes A_r)[i,j] \in T_j^{q'}$, check $(R \circledcirc A'_r)[i,j] > 0$, and if it does not hold, search for the largest q with $(R \circledcirc A''_r)[i, \rho(j,q)] > 0$. Note that T_j^q's with $q < a_j$ are assigned to columns $\rho(j,q)$'s in matrix A''_r. Once q' is known, the exact value of $(R \otimes A_r)[i,j]$ is returned by an exhaustive enumeration within $T_j^{q'}$. The total time for the new entries of $R \otimes A_r$ is $\tilde{O}(n^\omega + n_r \cdot \frac{n}{t})$, as each entry of $R \circledcirc A''_r$ is checked no more than once, and $T_j^{q'}$'s are of size $O(n/t)$. ◀

A Technicality on Equal Weights: To let the edges of a non-decreasing path first come from $G_{<r}$, then from G_r, we need to handle the case in which the edges of equal weight straddle across different A_r's. We split out these special edges and merge them into single matrices. Consider one such a matrix A'. The APNP matrix R' of the corresponding subgraph of A' can be computed by a transitive closure algorithm in $O(n^\omega)$ time. Suppose the APNP matrix before processing A' is R. Next we run $R \leftarrow \min(R, R \otimes R')$, which is similar to Lemma 13 (The First Edge), but now the finite entries of R' are all equal to the single value. Therefore the cost is $O(n^\omega)$. As there are only $O(t)$ such A''s, the total cost for the technicality is only $O(t \cdot n^\omega)$.

Algorithm 2 The algorithm for the simple case.

(S1) An APNP matrix R is initialized as $R[i,i] = -\infty$, and $R[i,j] = \infty$ for $i \neq j$.

(S2) $\forall r, 1 \leq r \leq t$, the APNP matrix R of $G_{<r}$ is extended to the APNP matrix of $G_{\leq r}$. Construct a sorted list $L(A_r[k,\cdot])$ of the $(< \infty)$ elements of $A_r[k,\cdot]$ in non-increasing order, for $1 \leq k \leq n$.

(S21) Run $R \leftarrow \min(R, R \circledcirc A_r)$ as in Lemma 13 (The First Edge). Let S be the set of the entries of R which get *strictly* smaller. Any entry *first* added to S, say $R[i,k]$, is associated with a pointer pointing to the head of $L(A_r[k,\cdot])$.

(S22) Run the following for $(n-2)$ steps or until $S = \varnothing$.

i. For every $R[i,k] \in S$:

A. Starting from the position of $L(A_r[k,\cdot])$ pointed to by the pointer associated with $R[i,k]$, move the pointer element by element until the first element of $L(A_r[k,\cdot])$ which is less than $R[i,k]$.

B. $R[i,j] \leftarrow \min\{R[i,j], A_r[k,j]\}$, for every element $A_r[k,j]$ of $L(A_r[k,\cdot])$ scanned, excluding the first element of $L(A_r[k,\cdot])$ which is less than $R[i,k]$.

ii. Re-initialize S as the set of the entries of R which get *strictly* smaller in this step.

iii. Any entry *first* added to S, say $R[i,k]$, is associated with a pointer pointing to the head of $L(A_r[k,\cdot])$.

The formal algorithm for the simple case is given in Table 2. The analysis for it is given in Theorem 15. Also by Observation 8 (An Entry Only Computed Once), the set S in Table 2 has the following property which is useful in the proof.

▶ **Observation 14.** The set S in Table 2 contains only the new entries of the APNP matrix of $G_{\leq r}$ w.r.t. the APNP matrix of $G_{<r}$.

▶ **Theorem 15.** *If the number of $(< \infty)$ elements in any row of A_r is bounded by d, then the steps S21 and S22 in Table 2 can find the APNP matrix of $G_{\leq r}$ in $\tilde{O}(n^\omega + n_r \cdot (d + \frac{n}{t}))$ time in every iteration. So the total time is $\tilde{O}(t \cdot n^\omega + n^2 \cdot (d + \frac{n}{t}))$.*

Proof. To show the correctness, we only need to prove that the step S22 in Table 2 is equivalent to $(n-2)$ steps of the algorithm in Table 1. This is almost obvious, as the entries not getting *strictly* smaller during the previous step of $R \leftarrow \min(R, R \circledcirc A_r)$ do not contribute anything in the current step of $R \leftarrow \min(R, R \circledcirc A_r)$. Also if S in the step S22 of Table 2 is empty, we can stop earlier than $(n-2)$ steps, as later steps of $R \leftarrow \min(R, R \circledcirc A_r)$ give the same R.

Note that the cost of the step S22 in Table 2 is charged to the scanning induced by the entries in S. By Observation 9 (Non-increasing of Entries) and Observation 14, the time of the step S22 of Table 2 in r-th iteration is $\tilde{O}(n_r \cdot d)$. Also by Lemma 13 (The First Edge), the step S21 of Table 2 takes $\tilde{O}(n^\omega + n_r \cdot \frac{n}{t})$ time, thus prove the theorem. ◀

So if the out-edges are uniformly distributed across all vertices, i.e., the number of $(< \infty)$ elements of any row in any A_r is always bounded by $O(n/t)$ for any t, then we can get a $\tilde{O}(n^{(3+\omega)/2})$ time APNP algorithm by setting $t = n^{(3-\omega)/2}$.

Generally a row of an A_r could have as many as $\Omega(n)$ $(< \infty)$ elements. Rows with a large number of $(< \infty)$ elements correspond to vertices of high out-degree. A new approach for them is given in the next section.

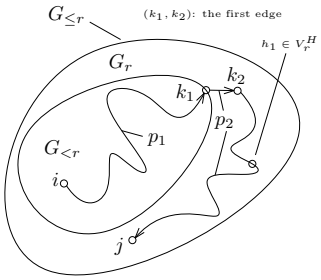

Figure 1 An illustration of the extension of the APNP matrix from $G_{<r}$ to $G_{\le r}$ in the general case.

4 Improved Algorithms for APNP

We move to the general case, in which the number of $(< \infty)$ elements in a row might not be bounded by $O(n/t)$. For ease of analysis, the edge set is divided into n^t subsets, each with at most $\lceil n^2/n^t \rceil = \lceil n^{2-t} \rceil$ edges. The vertices of G_r are classified as *high* out-degree with out-degree $> n^{1-t+s}$, or *low* out-degree otherwise, where $s > 0$ is a parameter to be chosen. Denote the sets of high out-degree, low out-degree vertices of G_r as V_r^H, V_r^L resp. Note that $|V_r^H| = O(n^{1-s})$.

The Intuition: As illustrated in Figure 1, among all the non-decreasing paths in $G_{\le r}$ from i to j with the minimum last edge weight, consider one path p with the least number of edges in G_r. By running the step S21 of Table 2, we construct the first edge (k_1, k_2). Thanks to p having the least number of edges in G_r, entries (i, k') of the APNP matrix of $G_{\le r}$ for $k' \in p[k_2, j)$ are new w.r.t. the APNP matrix of $G_{<r}$. The hard case is when there exists a vertex from V_r^H on $p[k_2, j)$. Consider the first such vertex h_1 from V_r^H. The important observation is that the edges on $p[k_2, h_1]$ are the out-edges of *low* out-degree vertices. Therefore, if we run the step S22 of Table 2, but differently only on the *low* out-degree vertices, then the portion $p[i, h_1]$ is successfully constructed. The last portion $p[h_1, j]$ can be constructed by Theorem 6, as $p[h_1, j]$ represents a non-decreasing path starting with an edge of weight no smaller than the last edge weight of $p[i, h_1]$. However, to get an efficient algorithm, we could not afford to construct $p[h_1, j]$ if it did not exist. To determine the existence of $p[h_1, j]$, we replace $p[h_1, j]$ with a non-decreasing path in G_r from h_1 to j with the *maximum* first edge weight. Then the existence of $p[h_1, j]$ is reduced to whether $p[i, h_1]$ can be concatenated with this replacement of $p[h_1, j]$.

The algorithm for the general case is described in Table 3. For the matrix H_1^N in Table 3, we have the following properties which are crucial for the correctness and an efficient algorithm.

▶ **Observation 16.** Consider one path p with the least number of edges in G_r among all the non-decreasing paths in $G_{\le r}$ from i to j with the minimum last edge weight. Let $p[k_1, j]$ be the portion of p in G_r. If $p[k_1, j] \neq \varnothing$, and there exists a vertex from V_r^H on $p(k_1, j)$, then the last edge weight of $p[i, h_1]$ is stored in $H_1^N[i, h_1]$, where h_1 is the first vertex in V_r^H on $p(k_1, j)$. For the matrix H_1^N, the number of $(< \infty)$ entries of H_1^N is no greater than n_r.

For the matrix X, the number of $X[i, j] = 1$ with $R[i, j] = \infty$ is no more than n_r. Such (i, j)'s of X are associated with non-decreasing paths never appearing in $G_{<r}$. Hence these (i, j)'s in the APNP matrix of $G_{\le r}$ are new w.r.t. R. By Theorem 6, the running time of the step S24 therefore is as follows.

Algorithm 3 The algorithm for the general case.

(S1) An APNP matrix R is initialized as $R[i,i] = -\infty$, and $R[i,j] = \infty$ for $i \neq j$.

(S2) $\forall r, 1 \leq r \leq n^t$, the APNP matrix R of $G_{<r}$ is extended to the APNP matrix of $G_{\leq r}$.

- **(S21)** Split out a sub-matrix A_r^L of A_r, which contains only the rows with the number of ($< \infty$) elements no greater than n^{1-t+s}. Initialize a matrix R' as R.
  - **i.** Run the steps S21, S22 in Table 2 with inputs R', A_r and A_r^L. The step S22 of Table 2 only works on A_r^L instead of A_r.
- **(S22)** With Theorem 5 (Maximum First Edge Weight), compute an $n^{1-s} \times n$ matrix H_2 with $H_2[i,j]$ representing the maximum first edge weight on a non-decreasing path in G_r from $i \in V_r^H$ to j.
- **(S23)** Group the entries of R' from V to V_r^H that are *new* w.r.t. R as an $n \times n^{1-s}$ matrix H_1^N. Construct a Boolean matrix X with $X[i,j] = 1$ if $(H_1^N \circledS H_2)[i,j] > 0$.
- **(S24)** Initialize a matrix R'' as R. For every $h_1 \in V_r^H$, use Theorem 6 to build the auxiliary data structure for h_1 in G_r, i.e., a set of $T(h_1, j)$'s for all $j \in V$.
  For every $X[i,j] = 1$ with $R[i,j] = \infty$:
  - **i.** For every $h_1 \in V_r^H$:
    - **A.** Query $T(h_1, j)$ with $w' = H_1^N[i, h_1]$. Get the minimum last edge weight w'' of a non-decreasing path from h_1 to j in G_r, starting from any out-edge e of h_1 with $w(e) \geq w'$.
    - **B.** $R''[i,j] \leftarrow \min(R''[i,j], w'')$.
- **(S25)** $R \leftarrow \min(R', R'')$.

▶ **Observation 17.** The step S24 of Table 3 has the running time of $\tilde{O}(n_r \cdot n^{1-s} + n^{1-s} \cdot n^{2-t})$.

▶ **Theorem 18.** *Given a real edge-weighted digraph on n vertices, the APNP matrix can be computed in $\tilde{O}(n^{2+\omega/3}) = O(n^{2.791})$ time.*

Proof. We show the APNP matrix R is correctly extended from $G_{<r}$ to $G_{\leq r}$. Among all the non-decreasing paths in $G_{\leq r}$ from i to j with the minimum last edge weight, consider one path p with the least number of edges in G_r. We show p can be constructed by the algorithm. As in Observation 10 (Two Parts of A Path), the path p consists of two subpaths $p[i, k_1]$, $p[k_1, j]$, lying within $G_{<r}$, G_r resp. An illustration is given in Figure 1. If $p[k_1, j] = \varnothing$, then $R[i,j]$ is an old entry. Therefore p is already constructed. Consider the case in which $p[k_1, j] \neq \varnothing$. Let (k_1, k_2) be the first edge on $p[k_1, j]$. If there is no vertex in V_r^H on $p[k_2, j)$, the circumstance is similar to the simple case of Section 3. The step S21 successfully constructs p under this circumstance. The only case left is when there exists a vertex in V_r^H on $p[k_2, j)$. Consider the first such vertex h_1. Due to Observation 16, the subpath $p[i, h_1]$ is successfully constructed by the step S21. The last portion $p[h_1, j]$ is constructed by the step S24, as $X[i,j] = 1$ and $R[i,j] = \infty$ in this case.

Next we proceed to the analysis of the time complexity. The step S1 costs $O(n^2)$. According to Theorem 15, the step S21 needs the running time of $\tilde{O}(n^\omega + n_r \cdot n^{1-t+s})$ to compute R'. By Theorem 5 (Maximum First Edge Weight), the step S22 takes $\tilde{O}(n^{1-s} \cdot n^{2-t})$ time, as $|V_r^H| = O(n^{1-s})$ and G_r has $O(n^{2-t})$ edges. The step S23 involves the computation of $H_1^N \circledS H_2$, for which Theorem 4 (Sparse Dominance Product) is used. Due to Observation 16, the cost for it is $\tilde{O}(n^\omega + n_r \cdot n^{1-s})$. Note that the number of ($> -\infty$) entries of H_2 is no greater than n^{2-s}. Also H_1^N, H_2 are *expanded* to $n \times n$ matrices in Theorem 4. For easy understanding, one can do $\mathbf{rb}(H_2)$ rather than $\mathbf{cb}(H_1^N)$ in the proof of Theorem 4, which can be found in [6]. The step S24 takes $\tilde{O}(n_r \cdot n^{1-s} + n^{1-s} \cdot n^{2-t})$ time according to

Observation 17. The step S25 costs $O(n^2)$. Summing up all these costs, we have the total running time within $\tilde{O}(\cdot)$ as follows.

$$\begin{aligned}&\sum_{1\le r\le n^t}(n^\omega + n_r\cdot n^{1-t+s}) + n^{3-s}\\ &+\sum_{1\le r\le n^t}(n^\omega + n_r\cdot n^{1-s})\\ &+\sum_{1\le r\le n^t}(n_r\cdot n^{1-s} + n^{1-s}\cdot n^{2-t})\\ =&n^{t+\omega} + n^{3-t+s} + n^{3-s},\end{aligned}$$

where by choosing $t = 2s$, and $s = 1 - \frac{\omega}{3}$, we have the final result $n^{2+\omega/3}$. ◀

4.1 A Slight Improvement via Rectangular Matrix Multiplication

The algorithm in Table 3 builds auxiliary data structures of Theorem 6 for all vertices in V_r^H. We can lessen this cost a little bit at the price of considering more vertices like h_1 in Figure 1.

The Intuition: As illustrated in Figure 1, formerly we consider the first vertex h_1 in V_r^H on $p[k_2, j)$. Now we look at more vertices from V_r^H on $p[k_2, j)$. Consider the second such vertex h_2. Similarly the edges on $p(h_1, h_2]$ are the out-edges of *low* out-degree vertices. Also thanks to p having the least number of edges in G_r, entries (i, k') of the APNP matrix of $G_{\le r}$ for $k' \in p(h_1, h_2)$ are new w.r.t. the APNP matrix of $G_{<r}$. Therefore, if we had constructed the first edge of $p[h_1, h_2]$, the cost for constructing later edges on $p[h_1, h_2]$ could be charged to the step S22 in Table 2. The idea for the improvement is to look at the first n^q vertices from V_r^H on $p[k_2, j)$. Name them as $h_1, \ldots, h_{n^q}$. If $p[k_2, j)$ contains no more than n^q vertices from V_r^H, the path p is then fully constructed. If $p[k_2, j)$ contains more than n^q vertices from V_r^H, we could sample only $O(n^{1-s-q}\log n)$ vertices from V_r^H, such that w.h.p. at least one vertex in $\{h_1, \ldots, h_{n^q}\}$ is hit. Say one such vertex is h. Note that $p[i, h]$ is already constructed. We then use the same auxiliary data structures as in Table 3 to construct $p[h, j]$. Also we use the same algorithm as in Table 3 to first determine the existence of $p[h, j]$. The difference is that the construction of $p[h, j]$ only involves the computation on $O(n^{1-s-q}\log n)$ vertices, instead of $O(n^{1-s})$ vertices as before. The improvement comes from this difference.

The algorithm is given in Table 4, which needs to compute $H_1^N \circledcirc A_r^H$ for matrices H_1^N, A_r^H defined within. The time complexity of the computation is given as follows, for which the idea is similar to one in [6]. Note that Theorem 4 (Sparse Dominance Product) can be extended to handling two matrices of sizes $n \times n^{1-s}$ and $n^{1-s} \times n$. The time complexity similar to the one in Theorem 4 is $O(m_1 m_2/n^{1-s} + n^{\omega(1,1-s,1)})$.

▶ **Lemma 19.** *The computation for $H_1^N \circledcirc A_r^H$ takes $\tilde{O}(n^{\omega(1,1-s,1)+q} + n_r\cdot n^{1-t-q+s})$ time.*

Proof. Let $L(A_r^H)$ be the sorted list of $(<\infty)$ entries of A_r^H. We divide $L(A_r^H)$ into n^q parts, so each part contains at most $\lceil n^{2-t}/n^q\rceil$ entries. We have n^q matrices $A_{r,1}^H, \ldots, A_{r,n^q}^H$ corresponding to each part. The entries of H_1^N with values no greater than $\max\{e \mid e \in A_{r,n^q}^H\}$ are partitioned into (n^q+1) matrices $H_{1,0}^N, H_{1,1}^N, \ldots, H_{1,n^q}^N$. The matrix $H_{1,p}^N$ with $1 \le p \le n^q$ consists of the entries $H_1^N[i, j]$'s satisfying $\min\{e \mid e \in A_{r,p}^H\} < H_1^N[i, j] \le \max\{e \mid e \in A_{r,p}^H\}$. The matrix $H_{1,0}^N$ consists of the entries $H_1^N[i, j]$'s satisfying $H_1^N[i, j] \le \min\{e \mid e \in A_{r,1}^H\}$.

We compute $H_1^N \circledcirc A_{r,p}^H$ for $1 \le p \le n^q$. Note that the following holds.

$$H_1^N \circledcirc A_{r,p}^H = (H_{1,0}^N + \cdots + H_{1,p}^N) \circledcirc A_{r,p}^H,$$

Algorithm 4 A slightly faster algorithm via rectangular matrix multiplication.

(S1) An APNP matrix R is initialized as $R[i,i] = -\infty$, and $R[i,j] = \infty$ for $i \neq j$.

(S2) $\forall r, 1 \leq r \leq n^t$, the APNP matrix R of $G_{<r}$ is extended to the APNP matrix of $G_{\leq r}$. Split A_r into two sub-matrices A_r^H, A_r^L, representing out-edges of high, low out-degree vertices resp.

- **(S21)** Initialize a matrix R' as R.
  - **i.** Run the steps S21, S22 in Table 2 with inputs R', A_r and A_r^L. The step S22 of Table 2 only works on A_r^L instead of A_r.
- **(S22)** Run the following for n^q steps:
  - **i.** Group the entries of R' from V to V_r^H that are *new* w.r.t. R as an $n \times n^{1-s}$ matrix H_1^N.
  - **ii.** $R' \leftarrow \min(R', H_1^N \circledcirc A_r^H)$. Determine the set S of the entries of R' which get *strictly* smaller in this step.
  - **iii.** Run the step S22 of Table 2 with inputs R', S and A_r^L.
- **(S23)** Sample uniformly at random $O(n^{1-s-q} \cdot \log n)$ vertices from V_r^H. Run the steps S22–S25 in Table 3, but only on the sampled vertices, rather than V_r^H.

where $(H_{1,0}^N + \cdots + H_{1,p-1}^N) \circledcirc A_{r,p}^H$ is computable in $O(n^{\omega(1,1-s,1)})$ time, as the entries in $(H_{1,0}^N + \cdots + H_{1,p-1}^N)$ are no greater than $\min\{e \mid e \in A_{r,p}^H\}$. Use $|H_{1,p}^N|$ to represent the number of $(< \infty)$ entries in $H_{1,p}^N$. We use Theorem 4 (Sparse Dominance Product) to compute $H_{1,p}^N \circledcirc A_{r,p}^H$ in $\tilde{O}(n^{\omega(1,1-s,1)} + |H_{1,p}^N| \cdot n^{2-t-q}/n^{1-s})$ time, as $A_{r,p}^H$ only has n^{1-s} rows. The total time for computing $H_1^N \circledcirc A_{r,p}^H$ for $1 \leq p \leq n^q$ is $\tilde{O}(n^{\omega(1,1-s,1)} \cdot n^q + n_r \cdot n^{2-t-q}/n^{1-s})$, as $\sum_{1 \leq p \leq n^q} |H_{1,p}^N| \leq n_r$.

Let $((A_{1,p}^H)', (A_{1,p}^H)'') = \mathbf{cb}(A_{1,p}^H)$. The computation of $H_1^N \circledcirc (A_{r,p}^H)'$, $H_1^N \circledcirc (A_{r,p}^H)''$ for $1 \leq p \leq n^q$ also takes $\tilde{O}(n^{\omega(1,1-s,1)} \cdot n^q + n_r \cdot n^{2-t-q}/n^{1-s})$ time.

For an entry $(H_1^N \circledcirc A_r^H)[i,j]$ to be determined, we find the smallest p with $(H_1^N \circledcirc A_{r,p}^H)[i,j] > 0$. Then $(H_1^N \circledcirc A_r^H)[i,j] \in A_{r,p}^H$. The remaining steps are similar to Lemma 13 (The First Edge). The total time is $\tilde{O}(n^{\omega(1,1-s,1)} \cdot n^q + n_r \cdot n^{2-t-q}/n^{1-s} + n_r \cdot n^{2-t-q}/n)$, as $A_{r,p}^H$ has n columns and only the new entries of $H_1^N \circledcirc A_r^H$ are of interest, of which the number is at most n_r. ◀

The step S23 in Table 4 constructs an L-bridging set with $L = n^q$. We use Lemma 7 to compute it *deterministically* as follows. Consider p as in the intuition of Section 4.1. The N subsets in Lemma 7 are p's with exactly n^q vertices from V_r^H on $p[k_2, j)$'s, where $j \in V_r^H$. To get the subset related to p, we need to extract the vertices in V_r^H on $p[k_2, j)$. This is solvable in $O(n^q)$ time if we know the predecessors in V_r^H of the vertices on $p(k_2, j]$.

We notice that for a specific i, just before the Step 23 of Table 4, for the j's of which $R'[i,j]$'s are new w.r.t. R, the edges corresponding to these $R'[i,j]$ form a forest, i.e., we know the predecessors in V (rather than in V_r^H) of these j's. Given such a forest, a DFS will give us the required predecessors in V_r^H of these j's. The cost of DFS for different i's is $O(n_r)$, as there are no more than n_r new $R'[i,j]$'s w.r.t. R. Thus the following holds.

▶ **Observation 20.** An L-bridging set with $L = n^q$ in the step S23 of Table 4 can be constructed *deterministically* in $\tilde{O}(n^{1-s} \cdot n^q)$ time.

▶ **Theorem 21.** *The APNP matrix of a real edge-weighted digraph with n vertices is computable in deterministic $O(n^{2.78})$ time.*

Proof. The correctness proof is almost similar to the one in Theorem 18. There we consider the first vertex h_1 in V_r^H, whereas here we consider the vertex h in V_r^H hit by the bridging

set. According to Observation 20, the total time for the construction of the bridging sets is $\tilde{O}(n^{t+1-s+q}) = \tilde{O}(n^{2+t-2s})$ (as $q \leq 1-s$), which is absorbed by the other terms.

Now we turn to the analysis of time complexity. The step S21 is the same as before, taking $\tilde{O}(n^{\omega} + n_r \cdot n^{1-t+s})$ time. The second step of S22 takes $\tilde{O}(n^{\omega(1,1-s,1)+q} + n_r \cdot n^{1-t-q+s})$ time, as shown in Lemma 19. The third step of S22 in one iteration of the step S2 is charged to the step S21, resulting in $\tilde{O}(n_r \cdot n^{1-t+s})$ time. The other steps are similar to Theorem 18, for which the time is $\tilde{O}(n^{\omega} + n_r \cdot n^{1-s-q} + n^{1-s-q} \cdot n^{2-t})$. Summing up all of them, we have the total running time within $\tilde{O}(\cdot)$ as follows.

$$n^{t+\omega} + n^{3-t+s} + n^{3-s-q} + n^{t+\omega(1,1-s,1)+2q},$$

where $\omega(1, 1-s, 1) \leq 2 + (1-s)(\omega - 2)$ if rectangular matrix multiplications are reduced to square matrix multiplications. By setting $s = \frac{3-\omega}{\omega+1}$, $t = \frac{1}{2}(3 + s - \omega)$, and $q = t - 2s$, we have the final result $n^{\frac{1}{2}(3+\frac{3-\omega}{\omega+1}+\omega)} = O(n^{2.78})$. ◀

References

1. Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997.
2. Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of Baur-Strassen's theorem: Shortest cycles, diameter, and matchings. *J. ACM*, 62(4):28, 2015.
3. Artur Czumaj, Mirosław Kowaluk, and Andrzej Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Theoretical Computer Science*, 380(1-2):37–46, 2007.
4. Artur Czumaj and Andrzej Lingas. Finding a heaviest triangle is not harder than matrix multiplication. In *Proceedings of the 18th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 986–994. SIAM, 2007.
5. Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
6. Ran Duan and Seth Pettie. Fast algorithms for $(\max, \min)$-matrix multiplication and bottleneck shortest paths. In *Proceedings of the 20th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–391. SIAM, 2009.
7. Pavlos Eirinakis, Matthew Williamson, and K. Subramani. On the Shoshan-Zwick algorithm for the all-pairs shortest path problem. *J. Graph Algorithms Appl.*, 21(2):177–181, 2017.
8. Michael L. Fredman and Robert E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.
9. Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134(2):103–139, 1997.
10. Zvi Galil and Oded Margalit. All pairs shortest paths for graphs with small integer length edges. *Journal of Computer and System Sciences*, 54(2):243–254, 1997.
11. François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proceedings of the 29th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018.
12. François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 296–303. ACM, 2014.
13. George J. Minty. A variant on the shortest-route problem. *Operations Research*, 6(6):882–883, 1958.

14 Liam Roditty and Virginia Vassilevska Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In *Proceedings of the 52nd annual IEEE symposium on Foundations of Computer Science*, pages 180–189. IEEE, 2011.

15 Piotr Sankowski and Karol Węgrzycki. Improved distance queries and cycle counting by Frobenius normal form. In *Proceedings of the 34th Symposium on Theoretical Aspects of Computer Science*, 2017.

16 R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

17 Asaf Shapira, Raphael Yuster, and Uri Zwick. All-pairs bottleneck paths in vertex weighted graphs. *Algorithmica*, 59(4):621–633, 2011.

18 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th annual IEEE symposium on Foundations of Computer Science*, pages 605–614. IEEE, 1999.

19 Virginia Vassilevska and Ryan Williams. Finding a maximum weight triangle in $n^{3-\delta}$ time, with applications. In *Proceedings of the 38th annual ACM Symposium on Theory of Computing*, pages 225–231. ACM, 2006.

20 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. Finding the smallest H-subgraph in real weighted graphs and related problems. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming*, pages 262–273. Springer-Verlag, 2006.

21 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory of Computing*, 5(9):173–189, 2009.

22 Virginia Vassilevska Williams. Nondecreasing paths in a weighted graph or: How to optimally read a train schedule. *ACM Trans. Algorithms*, 6(4):70:1–70:24, 2010.

23 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith–Winograd. In *Proceedings of the 44th annual ACM Symposium on Theory of Computing*, pages 887–898. ACM, 2012.

24 Raphael Yuster. A shortest cycle for each vertex of a graph. *Information Processing Letters*, 111(21):1057–1061, 2011.

25 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.