

Anarchy in the APSP: Algorithm and Hardness for Incorrect Implementation of Floyd-Warshall

Jaehyun Koo   

MIT EECS and CSAIL, Cambridge, MA, USA

Abstract

The celebrated Floyd-Warshall algorithm efficiently computes the all-pairs shortest path, and its simplicity made it a staple in computer science classes. Frequently, students discover a variant of this Floyd-Warshall algorithm by mixing up the loop order, ending up with the incorrect APSP matrix. This paper considers a computational problem of computing this *incorrect* APSP matrix. We will propose efficient algorithms for this problem and prove that this incorrect variant is APSP-complete.

2012 ACM Subject Classification Theory of computation → Shortest paths; Theory of computation → Problems, reductions and completeness

Keywords and phrases fine-grained complexity, recreational algorithms

Digital Object Identifier 10.4230/LIPIcs.FUN.2024.19

1 Introduction

The Floyd-Warshall algorithm computes the all-pairs shortest path (APSP) in a directed weighted graph [4, 9, 2]. Known in the computer science community for over 60 years, Floyd-Warshall is still one of the most efficient algorithms for the APSP problem, where it has a runtime of $O(n^3)$ for a graph with n vertices. No algorithms have improved this runtime by a polynomial factor in general graphs, which motivates the APSP Conjecture by [10].

Another remarkable characteristic of this algorithm is its simplicity - the standard implementation of this algorithm is a short triply nested loop, as shown below:

■ **Algorithm 1** The Floyd-Warshall Algorithm (KIJ Algorithm).

```
A ← Adjacency matrix of the graph
Ensure:  $A[i, i] = 0$  for all  $1 \leq i \leq n$ 
Ensure:  $G$  has no negative cycles
for  $k \leftarrow 1, 2, \dots, n$  do
  for  $i \leftarrow 1, 2, \dots, n$  do
    for  $j \leftarrow 1, 2, \dots, n$  do
       $A[i, j] \leftarrow \min(A[i, j], A[i, k] + A[k, j])$ 
```

Unfortunately, due to the algorithm being too simple, some students write a variant of the Floyd-Warshall algorithm, either by mistake or as a deliberate attempt to rectify the loop order into a *natural* lexicographic order.¹

¹ We are unaware of any publication over this variant except [7], but some Internet forums and even the lecture material discuss it. Examples are: <https://www.quora.com/Why-is-the-order-of-the-loops-in-Floyd-Warshall-algorithm-important-to-its-correctness>, <https://cs.stackexchange.com/questions/9636/why-doesnt-the-floyd-warshall-algorithm-work-if-i-put-k-in-the-innermost-loop>, <https://codeforces.com/blog/entry/20882>, <https://cs.nyu.edu/~siegel/JJ10.pdf>.



© Jaehyun Koo;

licensed under Creative Commons License CC-BY 4.0

12th International Conference on Fun with Algorithms (FUN 2024).

Editors: Andrei Z. Broder and Tami Tamir; Article No. 19; pp. 19:1–19:11

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

19:2 Algorithm for Incorrect Implementation of Floyd-Warshall

■ **Algorithm 2** The Variant of Floyd-Warshall Algorithm (IJK Algorithm).

```

A ← Adjacency matrix of the graph G
Ensure: A[i, i] = 0 for all 1 ≤ i ≤ n
Ensure: G has no negative cycles
for i ← 1, 2, ..., n do
  for j ← 1, 2, ..., n do
    for k ← 1, 2, ..., n do
      A[i, j] ← min(A[i, j], A[i, k] + A[k, j])

```

Algorithm 2 do not compute the correct APSP matrix. For example, let A be the adjacency matrix, and the M_1 and M_2 be the resulting matrix computed by Algorithm 1 and Algorithm 2 on A . The following choice of A makes $M_1[2, 1]$ and $M_2[2, 1]$ different:

$$A = \begin{bmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & 1 \\ 1 & \infty & 0 & \infty \\ \infty & \infty & 1 & 0 \end{bmatrix}, M_1 = \begin{bmatrix} 0 & \infty & \infty & \infty \\ 3 & 0 & 2 & 1 \\ 1 & \infty & 0 & \infty \\ 2 & \infty & 1 & 0 \end{bmatrix}, M_2 = \begin{bmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & 2 & 1 \\ 1 & \infty & 0 & \infty \\ 2 & \infty & 1 & 0 \end{bmatrix}$$

Since Algorithm 2 cannot compute the APSP correctly, it is natural to regard it as a novice mistake and move on. But we encounter surprisingly nontrivial questions when we try to understand what’s happening. For example, in a sparse graph with no negative edges, we can compute the APSP problem using Dijkstra’s algorithm [3] for computing a single-source shortest path in each vertex. Using the Fibonacci heap from [5] yields an $O(nm + n^2 \log n)$ time algorithm for n -vertex, m -edge graphs. Hence, we have an efficient way to obtain the resulting matrix by Algorithm 1, but for Algorithm 2, despite being a seemingly *novice* version, it is unclear how to obtain such matrix efficiently. To this end, we formally define the INCORRECT-APSP problem as follows:

► **Definition 1** (INCORRECT-APSP). *Given a weighted graph with n vertices without negative cycles, compute the matrix returned by Algorithm 2.*

We note that calling Algorithm 2 as an *incorrect* algorithm might be unfair. Indeed, a recurring theme in art and fashion is to take a seemingly *correct* version of a piece and twist it cleverly so that people can find something new from a familiar composition. Even in theoretical computer science, such attempts are not new: For example, a famous *Bogosort* [8] is a sorting algorithm deliberately engineered to perform worse. A more similar example where people consider a variant of a well-known algorithm also exists [7, 6]. In this way, depending on the inspiration we drew, we can call the Algorithm 2 as either IMPROVISED-APSP (Jazz music reference), PUNK-APSP (Rock music reference), or BOGO-APSP (Bogosort reference). However, we will (unfortunately) call the problem INCORRECT-APSP to avoid possible confusion.

A significant inspiration for this work is an arXiv preprint by [7], which considers the exact problem of INCORRECT-APSP . In the preprint, the authors proved that running Algorithm 2 three times on the given adjacency matrix will compute a correct APSP distance matrix. In other words, the INCORRECT-APSP problem is not entirely incorrect - even if the student does not know the correct loop order, they can run the algorithm three times and obtain a correct APSP matrix. While this result itself is funny (who cares about the loop order if you can just run three times?), it raises an intriguing question about the

hardness of the INCORRECT-APSP problem, as their result implies that INCORRECT-APSP is at least as hard as the APSP problem. In other words, a subcubic implementation of INCORRECT-APSP can imply a breakthrough! So, is this seemingly *novice* mistake of APSP made the problem strictly harder? Or can we find a subcubic reduction to prove that the INCORRECT-APSP problem is equivalent to the APSP problem?

1.1 Our Results

Our results in this paper are the following:

► **Theorem 2.** *Given a weighted graph with n vertices and m edges, we can solve the INCORRECT-APSP in $O(nT_{SSSP}(n, m))$ time, where $T_{SSSP}(n, m)$ is the time to execute a single-source shortest path (SSSP) algorithm in a graph with n vertices and m edges.*

► **Theorem 3.** *INCORRECT-APSP is subcubic equivalent to APSP.*

Theorem 2 concerns a notable special case where n iteration of single-source shortest path (SSSP) algorithm is known to outperform the Floyd-Warshall algorithm on the APSP problem. For the INCORRECT-APSP problem, it is unclear whether it can be optimized using the SSSP-based approach. We show this is possible and present an algorithm matching the APSP problem's corresponding bounds. Note that the current record for the SSSP algorithm is $T_{SSSP}(n, m) = O(m \log^2 n \log(nM) \log \log n)$ ([1]). Under the assumption where G do not contain negative weight edges, $T_{SSSP}(n, m) = O(n \log n + m)$ ([5]).

Theorem 3 implies that both INCORRECT-APSP and APSP have a subcubic algorithm or both do not, meaning that we have a new member in the list of APSP-complete problems where the subcubic solution to any of them implies a breakthrough.

1.2 Organization of our paper

In Section 2, we list the definitions and carefully formalize Algorithm 2 in graph theoretic notions. In Section 3, we prove that the INCORRECT-APSP problem is equivalent to the path minimization problem with certain constraints. In Section 4, we prove Theorem 2 by solving the path minimization problem by combining single-source shortest path problem and dynamic programming. In Section 5, we prove Theorem 3 by devising an algorithm that solves the INCORRECT-APSP in subcubic time under the subcubic APSP oracle.

2 Preliminaries

2.1 Notations

For integers i and j , we use $[i, j]$ to denote the set $\{i, i + 1, \dots, j\}$. Let $G = (V, E, w)$ be a weighted directed graph, where $V = [1, n]$ is the set of vertices, $E \subseteq V \times V$ is a set of edges where an element $(i, j) \in E$ represents a directed edge from vertex i to vertex j , and $w : E \rightarrow [-M, M]$ is the weight function of edges. As we represent vertices as integers, we can say that a vertex is *smaller* or *larger* than the other vertex by comparing the integer. We heavily rely on such notation, as our algorithm iterates V in the order of these integers.

We can use another representation for the directed weighted graphs if we do not rely on the graph's sparsity. For a graph $G = (V, E, w)$ with $V = [1, n]$, an *adjacency matrix* $A(G)$ of G is a $n \times n$ matrix such that:

$$A(G)[i, j] = \begin{cases} 0 & \text{if } i = j \\ w(i, j) & \text{if } i \neq j, (i, j) \in E \\ \infty & \text{otherwise} \end{cases}$$

Note that $A(G)[i, j]$ may not accurately represent G if it has negative-weight loops, but this is not a problem since we will invoke Algorithm 1 or Algorithm 2 only if G has no negative cycles. Here, ∞ is an element where $\infty + x = \infty$ holds for all $x \in [-M, M]$.

2.2 Definitions

Algorithm 1 and Algorithm 2 both try to recognize certain kinds of paths in a graph: Starting from the paths with at most one edge, it tries to build up a new paths that is a concatenation of two paths. Here, we will formalize it so that we can view it as a minimization problem over certain types of paths. In Section 3, we will argue that there is a simple characterization of paths realized by Algorithm 2.

We formally define a concept of *path realization*, a generic description of paths realized by the algorithms above.

► **Definition 4.** A path $P = \{p_0, p_1, \dots, p_m\}$ is **realized** by a sequence of 3-tuple $T = \{(i_1, j_1, k_1), (i_2, j_2, k_2), \dots, (i_l, j_l, k_l)\}$, if and only if $m \leq l$, or there exists a pair of integer d, x such that:

- $1 \leq d \leq l$
- $1 \leq x \leq m - 1$
- $(i_d, j_d, k_d) = (p_0, p_m, p_x)$
- $\{p_0, p_1, \dots, p_x\}$ is realized by $\{(i_1, j_1, k_1), (i_2, j_2, k_2), \dots, (i_{d-1}, j_{d-1}, k_{d-1})\}$.
- $\{p_x, p_{x+1}, \dots, p_m\}$ is realized by $\{(i_1, j_1, k_1), (i_2, j_2, k_2), \dots, (i_{d-1}, j_{d-1}, k_{d-1})\}$.

We list some immediate corollaries that can be shown by structural induction on Definition 4. While the corollaries themselves are somewhat trivial, they help us to organize our theorems in a purely mathematical way.

► **Corollary 5.** Let $T_{kij}(n)$ be a sequence of length n^3 consisting of 3-tuples, where the $an^2 + bn + c + 1$ -th element in the sequence is $(b + 1, c + 1, a + 1)$ for all $0 \leq a, b, c \leq n - 1$. Given a $n \times n$ adjacency matrix of G , where G has no negative cycle, Algorithm 1 will return a matrix M , where $M[i, j]$ is the minimum total weight of all simple paths from i to j that is realized by $T_{kij}(n)$.

Proof. It suffices to prove the statement for all paths, as the graph does not contain any negative cycles, and we can turn any non-simple paths into simple paths without increasing their total length.

For any path from i to j realized by $T(n)$, $M[i, j]$ is at most the weight of such paths. Note that the triple nested loop in Algorithm 1 exactly iterates the list $T_{kij}(n)$ and performs a *relaxation* operation of $A[i, j] = \min(A[i, j], A[i, k] + A[k, j])$ for each 3-tuple $(i, j, k) \in T_{kij}(n)$. Hence, we can show this by induction over Definition 4. Specifically, we can prove the following: For all path P of length k realized by a sequence of 3-tuples of length l , $M[i, j]$ is always at most the weight of P after applying the relaxation operation for first l elements of $T_{kij}(n)$. Then, we can apply induction over (k, l) .

Conversely, we can also prove that, for any $1 \leq i, j \leq n$, there exists a path of weight at most $M[i, j]$, which is realized by $T(n)$, which also follows the identical induction as above. ◀

► **Corollary 6.** Let $T_{ijk}(n)$ be a sequence of length n^3 consisting of 3-tuples, where the $an^2 + bn + c + 1$ -th element in the sequence is $(a + 1, b + 1, c + 1)$ for all $0 \leq a, b, c \leq n - 1$. Given a $n \times n$ adjacency matrix of G , where G has no negative cycle, Algorithm 2 will return a matrix M , where $M[i, j]$ is the minimum total weight of all simple paths from i to j that is realized by $T_{ijk}(n)$.

Proof. You can proceed identically as in Corollary 5. ◀

The following provides some characterizations of paths that we will use in the later stage of the paper.

► **Definition 7.** A path $P = \{p_0, p_1, \dots, p_k\}$ with $k \geq 1$ is **increasing** if for all $1 \leq i \leq k$ it holds that $p_{i-1} < p_i$.

► **Definition 8.** A path $P = \{p_0, p_1, \dots, p_k\}$ with $k \geq 1$ is **decreasing** if for all $1 \leq i \leq k$ it holds that $p_{i-1} > p_i$.

► **Definition 9.** A path $P = \{p_0, p_1, \dots, p_k\}$ with $k \geq 1$ is **valley** if for all $1 \leq i \leq k-1$ it holds that $p_i \leq \min(p_0, p_k)$.

► **Definition 10.** A path $P = \{p_0, p_1, \dots, p_k\}$ with $k \geq 1$ is **proper** if there is no index $1 \leq i \leq k-2$ such that $p_i > \min(p_0, p_k)$ and $p_{i+1} > \min(p_0, p_k)$.

3 Main Theorem

The main theorem of our paper is as follows:

► **Theorem 11.** In a graph with n vertices, a nonempty simple path $P = \{p_0, p_1, \dots, p_k\}$ is realized by $T_{ijk}(n)$ if and only if one of the following holds:

- $p_0 < p_k$, and there exists an index $0 \leq i \leq k$ such that $\{p_0, p_1, \dots, p_i\}$ is proper, $\{p_i, p_{i+1}, \dots, p_k\}$ is increasing, and $p_i \geq p_0$.
- $p_0 > p_k$, and there exists an index $0 \leq i \leq k$ such that $\{p_0, p_1, \dots, p_i\}$ is decreasing, $\{p_i, p_{i+1}, \dots, p_k\}$ is proper, and $p_i \geq p_k$.

By combining Theorem 11 with Corollary 6, we can yield an explicit characterization for the output of Algorithm 2.

► **Corollary 12.** Given a $n \times n$ adjacency matrix of G where G has no negative cycle, Algorithm 2 will return a matrix M , where $M[i, j]$ is:

- If $i < j$, the minimum possible total weight of a path $P = \{p_0 = i, p_1, \dots, p_k = j\}$ such that there exists an index $0 \leq x \leq k$ such that $\{p_0, p_1, \dots, p_x\}$ is proper, $\{p_x, p_{x+1}, \dots, p_k\}$ is increasing, and $p_x \geq p_0$.
- If $i = j$, 0.
- If $i > j$, the minimum possible total weight of a path $P = \{p_0 = i, p_1, \dots, p_k = j\}$ such that $\{p_0, p_1, \dots, p_x\}$ is decreasing, $\{p_x, p_{x+1}, \dots, p_k\}$ is proper, and $p_x \geq p_0$.

Proof. By Corollary 6, Algorithm 2 returns a matrix where $M[i, j]$ is the minimum total weight of all simple paths from i to j realized by $T_{ijk}(n)$, which is of above form by Theorem 11. ◀

3.1 High-level ideas

Before proceeding to the proof of Theorem 11, let's play with several examples to motivate intuition. Recall the standard proof of Algorithm 1, where one proves the following lemma by the induction on t :

► **Lemma 13** (Key lemma of [4]). For all $0 \leq t \leq n$, a path $P = \{p_0, p_1, \dots, p_k\}$ is realized by the tn^2 -length prefix of $T_{kij}(n)$ if and only if $p_i \leq t$ for all $1 \leq i \leq k-1$.

19:6 Algorithm for Incorrect Implementation of Floyd-Warshall

This proof strategy surely does not work for Algorithm 2, but it does work identically for the specific type of path, which we define as a valley path:

► **Definition 9.** A path $P = \{p_0, p_1, \dots, p_k\}$ with $k \geq 1$ is **valley** if for all $1 \leq i \leq k - 1$ it holds that $p_i \leq \min(p_0, p_k)$.

However, while all valley paths are realized by $T_{ijk}(n)$, this is still not an exhaustive characterization. Consider the path $P = \{3, 101, 1, 102, 2\}$, which is not a valley path but is realized by $T_{ijk}(n)$ as the subsequence $\{(1, 2, 102), (3, 1, 101), (3, 2, 1)\}$ of $T_{ijk}(n)$ can realize P . The reason is that the algorithm can glue some large vertices into P before it starts to realize the valley path. For this specific counterexample, one can observe that these large vertices should be adjacent to small vertices to get glued inside a valley path, which motivates the definition of a proper path.

► **Definition 10.** A path $P = \{p_0, p_1, \dots, p_k\}$ with $k \geq 1$ is **proper** if there is no index $1 \leq i \leq k - 2$ such that $p_i > \min(p_0, p_k)$ and $p_{i+1} > \min(p_0, p_k)$.

With some modification of proofs in Lemma 13, we can see that all proper paths are realized by $T_{ijk}(n)$. However, this definition is slightly strict: For example, the path $P = \{1, 2, 3, 4\}$ is realized by $\{(1, 3, 2), (1, 4, 3)\}$ but does not fit in a definition of valley path. More generally, we can append any increasing paths in the back of the proper path and prepend any decreasing paths in the front of the proper path. By addressing these cases, we do reach an appropriate characterization of paths realized by $T_{ijk}(n)$, that fits nicely to the inductive argument given in the proof of Theorem 11.

3.2 Proof of the Main Theorem

► **Theorem 11.** In a graph with n vertices, a nonempty simple path $P = \{p_0, p_1, \dots, p_k\}$ is realized by $T_{ijk}(n)$ if and only if one of the following holds:

- $p_0 < p_k$, and there exists an index $0 \leq i \leq k$ such that $\{p_0, p_1, \dots, p_i\}$ is proper, $\{p_i, p_{i+1}, \dots, p_k\}$ is increasing, and $p_i \geq p_0$.
- $p_0 > p_k$, and there exists an index $0 \leq i \leq k$ such that $\{p_0, p_1, \dots, p_i\}$ is decreasing, $\{p_i, p_{i+1}, \dots, p_k\}$ is proper, and $p_i \geq p_k$.

Proof (\Leftarrow). Let n be the number of vertices, and let $T_{ijk}(n, i, j)$ be the prefix of $T_{ijk}(n)$ up to and including the element (i, j, n) .

We first show that all proper paths from i to j are realized by $T_{ijk}(n, i, j)$. We perform an induction over (i, j) . Consider a proper path $P = \{p_0, p_1, \dots, p_k\}$, and assume that there is no index $1 \leq i \leq k - 1$ such that $p_i \leq \min(p_0, p_k)$. Then, since there could be no two entries with $p_i > \min(p_0, p_k)$, it holds that $k \leq 2$, and we are done. Otherwise, take the maximum p_x such that $p_x < \min(p_0, p_k)$. By maximality, both $\{p_0, p_1, \dots, p_x\}$ and $\{p_x, p_{x+1}, \dots, p_k\}$ are proper paths, and by inductive hypothesis, they are realized in $T_{ijk}(p_0, p_x)$ and $T_{ijk}(p_x, p_k)$. Hence, by Definition 4, P is realized.

As $P = \{p_0, p_1, \dots, p_k\}$ from p_0 to p_k are realized by $T_{ijk}(n, p_0, p_k)$, we can see $\{p_0, p_1, \dots, p_k, x\}$ are realized by $T_{ijk}(n, p_0, x)$ for any $x > p_k$, and similarly, $\{x, p_0, p_1, \dots, p_k\}$ are realized by $T_{ijk}(n, x, p_k)$ for any $x > p_0$. Using this fact, we can use a similar induction to prove that we can append an increasing path in the back or a decreasing path in the front. ◀

Proof (\rightarrow). We use the induction on the length l of the prefix of $T_{ijk}(n)$ to show that the path realized by a length- l prefix of $T(n)$ is always in one of such patterns. This claim is true for $l = 0$. Assume $l \geq 1$ and let (i, j, k) be the last element of $T(n)$. Note that we can

assume $i \neq j, j \neq k, k \neq i$, since if one of them holds, we have no new realized paths or a non-simple path.

We list all possible cases and show we can always find such patterns. Here, we use $*$ to denote that any index from 1 to n can be there.

- $i < j, k < i$: We consider a path in a form of $i \rightarrow$ (decreasing) \rightarrow (proper) $\rightarrow k \rightarrow$ (proper) \rightarrow (increasing) $\rightarrow j$. Let z be the first vertex in the increasing part of this path, where $i \leq z$ (as $i < j$ such vertex exists). The path between i to z is proper, and the path between z to j is increasing.
- $i < j, i < k < j$: All entries of $(k, j, *)$ are not in the current prefix, so the only path from k to j realized now is the trivial path $\{k, j\}$. We can append this in the increasing part of our path.
- $i < j, j < k$: All entries of $(i, k, *)$ and $(k, j, *)$ are not in the current prefix, so the path we consider is exactly $\{i, k, j\}$ which is proper.
- $j < i, k < j$: We consider a path in a form of $i \rightarrow$ (decreasing) \rightarrow (proper) $\rightarrow k \rightarrow$ (proper) \rightarrow (increasing) $\rightarrow j$. Let z be the last vertex in the decreasing part of this path, where $j \leq z$ (as $j < i$ such vertex exists). The path between i to z is decreasing, and the path between z to j is proper.
- $j < i, j < k < i$: All entries of $(i, k, *)$ are not in the current prefix, so the only path from i to k realized now is the trivial path $\{i, k\}$. We can prepend this in the decreasing part of our path.
- $j < i, i < k$: All entries of $(i, k, *)$ and $(k, j, *)$ are not in the current prefix, so the path we consider is exactly $\{i, k, j\}$ which is proper. ◀

4 Algorithm for Incorrect-APSP Problem

In this section, we prove the following theorem:

► **Theorem 2.** *Given a weighted graph with n vertices and m edges, we can solve the INCORRECT-APSP in $O(nT_{SSSP}(n, m))$ time, where $T_{SSSP}(n, m)$ is the time to execute a single-source shortest path (SSSP) algorithm in a graph with n vertices and m edges.*

For this, we will try to solve the optimization problem described in Corollary 12. Here, it is useful to observe the following:

► **Observation 14.** *Let $rev(P)$ be the reverse of the path. A path P is realized by $T_{ijk}(n)$ if and only if $rev(P)$ is realized by $T_{ij^k}(n)$.*

Proof. If P satisfies the property from Theorem 11, the reverse also satisfies the property from Theorem 11. ◀

By Observation 14, it suffices to find $M[i, j]$ for all $i < j$, as the other case can be solved by reversing all edges and repeating the same algorithm.

We will fix i and devise an algorithm that computes $M[i, j]$ for all $j \geq i$. To start, we find a minimum-length proper path to all j . By the description in Corollary 12, we only need to compute the proper path ending in $j \geq i$. By Definition 10, a path is proper if it does not contain two adjacent vertices p_x, p_{x+1} of index greater than $\min(i, j)$ unless one of the vertices is the last vertex of the path.

Since we have $\min(i, j) = i$, this is equivalent to not using an edge where both endpoints have an index greater than i , except where the edge is the last on the path. As we fixed i , we can apply SSSP algorithm on G where edges with endpoints greater than i are removed. Let $PROPEREXCEPTLAST[j]$ be the distance from i to j on such graph.

19:8 Algorithm for Incorrect Implementation of Floyd-Warshall

Then, for the last move, we simulate the move for all vertex. Specifically, we initialize $\text{PROPER}[j] = \text{PROPEREXCEPTLAST}[j]$ for all $1 \leq j \leq n$, and for all edge $(u, v) \in E$, we set $\text{PROPER}[v] = \min(\text{PROPER}[u], \text{PROPEREXCEPTLAST}[u] + w(u \rightarrow v))$. In the end, $\text{PROPER}[j]$ holds the shortest proper path from i to j for all $j \geq i$.

Finally, we may need to append the increasing path at the back of each proper path. We can use dynamic programming as the increasing path consists of edges headed to the larger indexed vertex. For computing the $M[i, j]$, we have two choices:

- Assume $p_x = p_k$ and simply take the shortest proper path from i to j .
- Otherwise, we pick an edge $(k \rightarrow j)$ and use the edge as a final edge in the optimal path. Here, $k < j$ should hold, and there should be an edge $(k, j) \in E$. The remaining path from i to k is a proper path appended with an increasing path computed by $M[i, k]$.

In Algorithm 3, we present a pseudocode of the above algorithm.

■ **Algorithm 3** Computing $M[i, j]$ for fixed i , all $j \geq i$.

```

 $G' = (G \text{ without edges where both vertices have index greater than } i)$ 
 $\text{PROPER} = \text{SSSP}(G', i)$ 
 $\text{PROPEREXCEPTLAST} = \text{PROPER}$ 
for  $(u, v) \in E$  do
     $\text{PROPER}[v] = \min(\text{PROPER}[u], \text{PROPEREXCEPTLAST}[u] + w(u \rightarrow v))$ .
for  $j \leftarrow i, i + 1, \dots, n$  do
     $M[i, j] = \text{PROPER}[j]$ 
    for  $k$  incident to  $j$  do
        if  $i \leq k < j$  then
             $M[i, j] = \min(M[i, j], M[i, k] + w(k \rightarrow j))$ 

```

As all procedures except the SSSP use $O(n + m)$ computation, the algorithm's running time is dominated by the SSSP, which leads to the runtime of $O(nT_{\text{SSSP}}(n, m))$.

5 Subcubic equivalence between APSP and Incorrect-APSP

In this section, we will prove Theorem 3. For a computational problem A, B , we say $A \leq_3 B$ if there is a subcubic reduction from A to B as in [10]. The following result is known:

► **Theorem 15** (Theorem 1 of [7]). $\text{APSP} \leq_3 \text{INCORRECT-APSP}$.

In the first subsection, we will recite the proof of Theorem 15 from [7]. We emphasize that this is not our original contribution: Our goal is to make this paper self-contained and to rephrase the statement of [7] in terms of subcubic reduction.

In the second subsection, we complement Theorem 15 by proving the following theorem, hence closing the gap:

► **Theorem 16.** $\text{INCORRECT-APSP} \leq_3 \text{APSP}$.

Given that Theorem 16 and Theorem 15 are true, the proof of Theorem 3 follows by definition of subcubic equivalence in [10].

► **Theorem 3.** INCORRECT-APSP is subcubic equivalent to APSP .

5.1 Proof of Theorem 15

► **Lemma 17.** *Let T^i be a sequence T repeated for i times and P be any simple path in a graph with n vertices. $(T_{ijk}(n))^3$ realizes P .*

Proof. Given a path $P = \{p_0, p_1, \dots, p_k\}$, let m be an integer in $[0, k]$ where $p_m = \max_{i=0}^k p_i$. We call p_i a *skyscraper*, if it satisfies the following:

- $i < m$, and there exists no $j < i$ such that $p_j > p_i$.
- $i = m$.
- $i > m$, and there exists no $j > i$ such that $p_j > p_i$.

It is helpful to observe that p_0 and p_k are always a skyscraper.

Let $0 = i_0 < i_1 < \dots < i_j = m < i_{j+1} < \dots < i_l = k$ be a list of indices where p_{i_s} is a skyscraper for all $0 \leq s \leq l$. For all $1 \leq s \leq l$, it holds that the subpath $p_{i_{s-1}}, p_{i_{s-1}+1}, \dots, p_{i_s}$ is proper - in fact, there can't be even a single vertex with value larger than $\min(p_{i_{s-1}}, p_{i_s})$ since that will add such vertex into a list of skyscraper, contradicting our assumption that i_{s-1} and i_s are adjacent. By Theorem 11, we can see that the subpath $p_{i_{s-1}}, p_{i_{s-1}+1}, \dots, p_{i_s}$ is realized by $(T_{ijk}(n))^1$.

Next, we will prove that both subpath p_0, p_1, \dots, p_m and p_m, p_{m+1}, \dots, p_k are realized in $(T_{ijk}(n))^2$. Given that all subpaths between $p_{i_{s-1}}$ and p_{i_s} are realized in $(T_{ijk}(n))^1$, the remaining skyscrapers are either increasing or decreasing according to its relative location per p_m . The statement holds as we can realize the increasing and decreasing path by Theorem 11.

Finally, to realize P in $(T_{ijk}(n))^3$, we need (p_0, p_k, p_m) in $T_{ijk}(n)$ which we definitely have. ◀

► **Theorem 15** (Theorem 1 of [7]). $APSP \leq_3 INCORRECT-APSP$.

Proof. By Definition 3.1 in [10], it suffices to design a subcubic algorithm for APSP which calls the oracle to compute INCORRECT-APSP for an $n \times n$ matrix, at most a polylogarithmic times.

We take an input graph, construct an adjacency matrix, call INCORRECT-APSP, call INCORRECT-APSP again in the returned matrix, call INCORRECT-APSP again in the returned matrix, and return the output of INCORRECT-APSP. This algorithm calls an oracle for 3 times and runs in quadratic time, which satisfies all requirements to be a subcubic reduction. The correctness follows by Lemma 17. ◀

5.2 Proof of Theorem 16

Let $A \odot B$ be a min-plus matrix product of two $n \times n$ matrix A and B . The following lemma shows that we can convert a proper path minimization into a valley path minimization problem. Note that the definition of G^2 is equivalent to the usual definition of graph powers, defined as a power of adjacency matrix (here, the multiplication operator is \odot).

► **Lemma 18.** *Given a graph G , a proper path of G from i to j with weight w exists if and only if a valley path of G^2 from i to j with weight w exists. Here, G^2 is a complete directed graph on the same set of vertices with G , where the weight of the edge from i to j is the minimum weight path from i to j using at most 2 edges.*

Proof. Every proper path of G translates to a valley path of G^2 , as all vertices that are not i, j and have an index at most $\min(i, j)$ are either adjacent in the path or have at most one intermediate vertex in the path. Conversely, given a valley path of G^2 , we can turn it into a path in G by replacing an edge with two edges and a vertex. Those new are the only vertex that can violate the $p_i \leq \min(p_0, p_k)$ condition, but they are not adjacent by construction. ◀

19:10 Algorithm for Incorrect Implementation of Floyd-Warshall

We will devise an algorithm that uses subcubic oracle for APSP-complete problems to compute the resulting matrix $M[i, j]$. Note that we will only demonstrate how to compute $M[i, j]$ for $i \leq j$ - as in Observation 14, we can compute the opposite part by computing the transpose of $A(G)$ in quadratic time and running the same algorithm. We use the result from [10] that APSP and min-plus matrix multiplication are subcubic equivalent.

Let's first find a matrix for minimum proper paths in G , which by Lemma 18 is equivalent to minimum valley paths in G^2 . Let $V_l[i, j]$ be a minimum weight of the valley path from i to j , which includes all valley paths with at most 2^l edges. Note that this definition includes all valley paths with $\leq 2^l$ edges, but it is not limited to such valley paths - it is, however, limited to valley paths. Here, $V_0 = A(G^2) = A(G)^2$, and we can compute this using min-plus matrix multiplication.

To compute the entry $V_l[i, j]$, we fix the vertex k in the middle of the (imaginary) path P . As k is in the middle of P , and since P has at most 2^l edges, the path to the left and the right of k has at most 2^{l-1} edges.

Let's say a valley path from i to j is *ascending* if $i < j$ and *descending* otherwise. The path to the left of k can be divided into a block of descending paths, and the path to the right of k can be divided into a block of ascending paths. To see this, consider all vertex v such that the subpath from v to k does not have any vertex greater than v , which we refer to as a *skyscraper*. Then, for each adjacent skyscraper, the subpath they form is a valley, as otherwise, we find a skyscraper in between them.

Conversely, it's easy to see that a concatenation of ascending valley paths, followed by descending valley paths, forms a valley path. Hence, a valley path of at most 2^l edges is equivalent to a sequence of descending and ascending valley paths, each with at most 2^{l-1} edges.

The minimum path from i to j that is a sequence of descending valley paths of at most 2^{l-1} edges, can be computed with the APSP oracle: We can provide an adjacency matrix of $V_{l-1}[i, j]$, where all entries with $i < j$ are overwritten to ∞ . Conversely, we can compute the sequence of ascending paths with the APSP oracle by overwriting $V_{l-1}[i, j]$ with $I > j$ to ∞ . Then, we can combine those two patterns by a single min-plus matrix multiplication.

Finally, we need to append an increasing path in the back of the path. By supplementing the adjacency matrix of G where we overwrite all entries with $i > j$ to ∞ , we can compute the increasing path of minimum cost for all pairs using an APSP oracle. We can obtain the desired answer by multiplying this with the valley path matrix.

In Algorithm 4, we present a pseudocode of the above algorithm.

As we make at most $O(\log n)$ calls to the subcubic oracles, Algorithm 4 is subcubic, which proves Theorem 16.

■ **Algorithm 4** Computing $M[i, j]$ for all $1 \leq i \leq j \leq n$.

Ensure: $A \odot B$ is subcubic

Ensure: $APSP(G)$ is subcubic

$V_0 = A(G) \odot A(G)$

for $l \leftarrow 1, 2, \dots, \lceil \log_2 n \rceil$ **do**

 DESCENDINGVALLEY = V_{l-1}

 ASCENDINGVALLEY = V_{l-1}

for $i \leftarrow 1, 2, \dots, n$ **do**

for $j \leftarrow 1, 2, \dots, n$ **do**

if $i < j$ **then**

 DESCENDINGVALLEY[i, j] = ∞

else if $i > j$ **then**

 ASCENDINGVALLEY[i, j] = ∞

$V_l = APSP(\text{DESCENDINGVALLEY}) \odot APSP(\text{ASCENDINGVALLEY})$

 VALLEY = $V_{\lceil \log_2 n \rceil}$

$G' = (G \text{ with edges } u \rightarrow v \text{ such that } u < v)$

 ANSWER = VALLEY $\odot APSP(A(G'))$

for $i \leftarrow 1, 2, \dots, n$ **do**

for $j \leftarrow 1, 2, \dots, n$ **do**

if $i \leq j$ **then**

$M[i, j] = \text{ANSWER}[i, j]$

References

- 1 K. Bringmann, A. Cassis, and N. Fischer. Negative-weight single-source shortest paths in near-linear time: Now faster! In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 515–538, Los Alamitos, CA, USA, November 2023. IEEE Computer Society. doi:10.1109/FOCS57990.2023.00038.
- 2 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.
- 3 Edsger W Dijkstra. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, pages 287–290. 2022.
- 4 Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.
- 5 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, July 1987. doi:10.1145/28869.28874.
- 6 Stanley P. Y. Fung. Is this the simplest (and most surprising) sorting algorithm ever?, 2021. arXiv:2110.01111.
- 7 Ikumi Hide, Soh Kumabe, and Takanori Maehara. Incorrect implementations of the floyd-warshall algorithm give correct solutions after three repeats, 2019. arXiv:1904.01210.
- 8 Eric S Raymond. *The new hacker's dictionary*. Mit Press, 1996.
- 9 Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, January 1962. doi:10.1145/321105.321107.
- 10 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5), August 2018. doi:10.1145/3186893.