Jumbled Floyd-Warshall from a dynamic programming perspective

Emily Fox*

November 5, 2025

Abstract

Given an n-vertex directed graph with real edge weights and no negative weight cycles, the algorithm commonly called Floyd-Warshall solves the all-pairs shortest paths problem in $O(n^3)$ time. The algorithm consists of three nested loops over the vertices surrounding a simple operation that uses the loops' parameters to update estimates of pairwise distances. The correctness of the algorithm depends upon on using a particular ordering of the loops relative to the parameters of the operation updating distance estimates.

In a 2019 preprint, Hide, Kumabe, and Maehara gave the surprising result that incorrect orderings of the loops still lead to correct results if the entire collection of iterations is repeated three times. In this note, we give an alternative presentation of their observation. Our presentation resembles the way Floyd-Warshall is often presented in college classes, showing it as a refinement of a dynamic programming algorithm.

1 Introduction

⟨⟨**Write a real introduction.**⟩⟩ This note presents the results of Hide, Kumabe, and Maehara [HKM19] ◁◁◁◁◁ as the refinement of two dynamic programming algorithms for all-pairs shortest paths (APSP).

2 Notation and definitions

Let $G = (V, E)$ be a simple directed graph with real edge weights $w : E \to \mathbb{R}$. Let $n = |V|$. We assume there are no negative weight cycles in G. For simplicity, we extend E to include exactly one edge $u{\to}v$ per ordered pair $(u, v) \in V \times V$ by setting $w(v{\to}v) = 0$ for all $v \in V$ and $w(u{\to}v) = \infty$ for all $u \neq v$ such that $u{\to}v$ was not originally a member of E. For each pair $(u, v) \in V \times V$, let $dist(u, v)$ denote the length of the shortest path from u to v in G.

We say a path from u to v ***passes through*** a third vertex x if the path both enters *and* leaves x.

Given any two ordered triples (a, b, c) and (a', b', c') of integers, we say $(a', b', c') < (a, b, c)$ if and only if the former is *lexicographically* smaller than the latter, i.e., if and only if $a' < a$ *or* $a' = a$ and $b' < b$ *or* $a' = a$, $b' = b$, and $c' < c$.

3 Subproblems and evaluation orders

Number the vertices arbitrary from 1 to n. From here on, we typically won't make any distinction between a vertex v and its number. We define a collection of ***restricted distance functions***

*Siebel School of Computing and Data Science, University of Illinois Urbana-Champaign; ekfox@illinois.edu.

that give the length of a shortest path subject to some restrictions on which vertices it passes through. Each of the functions takes two vertices $u, v \in \{1, \ldots, n\}$, and an integer $x \in \{0, \ldots, n\}$ as parameters. They each output the length of some path from u to v whose specification depends on the function in question.

- $\boldsymbol{dist_1(u, v, x)}$ denotes the length of the shortest path from u to v that only passes through vertices numbered at most $\min\{u, v, x\}$.
- $\boldsymbol{dist_{2,\ell}(u, v, x)}$ denotes the length of the shortest path from u to v that only passes through vertices numbered at most $\min\{u, x\}$.
- $\boldsymbol{dist_{2,r}(u, v, x)}$ denotes the length of the shortest path from u to v that only passes through vertices numbered at most $\min\{v, x\}$.
- $\boldsymbol{dist_3(u, v, x)}$ denotes the length of the shortest path from u to v that only passes through vertices numbered at most x.

Note that $dist_{2,\ell}$ and $dist_{2,r}$ have symmetric definitions, and $dist_3$ is simply the standard restricted distance function used in dynamic programming based presentations of Floyd-Warshall. Each of these functions can be evaluated using recurrences that are independent of the others', and the recursive cases for all of these recurrences are based on the common decision of whether we should include x in the best relevant path from u to v.

The set of feasible evaluation orders for solving these independent recurrences reduces as one goes further down the list of functions. We are particularly interested in the following four evaluation orders listed from least generally useful to most:

- $\boldsymbol{A}$: A triply nested for loop over increasing values of u, v, x ordered from outermost loop to innermost.
- $\boldsymbol{B_\ell}$: A triply nested for loop over increasing values of u, x, v ordered from outermost loop to innermost.
- $\boldsymbol{B_r}$: A triply nested for loop over increasing values of v, x, u ordered from outermost loop to innermost.
- $\boldsymbol{C}$: A triply nested for loop over increasing values of x, u, v ordered from outermost loop to innermost.

Evaluation order C is the only one that leads to a correct implementations of Floyd-Warshall, because it is the only one that can be used to solve $dist_3$ for all triples of vertices u, v, x. However, the other ones *can* be used after all by repeatedly applying them. The reason is that we can state alternative recurrences for the latter functions that use the results of *earlier* functions in the list.

3.1 Solving $dist_1$

We can evaluate $dist_1$ using the following recurrence:

$$dist_1(u, v, x) = \begin{cases} w(u \to v) & \text{if } x = 0 \\ dist_1(u, v, x-1) & \text{if } x \geq \min\{u, v\} \\ \min\left\{ \begin{array}{c} dist_1(u, v, x-1), \\ dist_1(u, x, x-1) + dist_1(x, v, x-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

Notice how in every recursive use of $dist_1$, the condition $x < \min\{u, v\}$ implies *all* of the parameters are non-increasing during recursive calls. In principle, we can use any nesting of for loops over u, v, and x to evaluate $dist_1$, as long as the values of the parameters are increasing. In particular, we can even use evaluation order A.

3.2 Solving $dist_{2,\ell}$ and $dist_{2,r}$

We can evaluate $dist_{2,\ell}$ using a recurrence that is nearly identical to the one we used last time:

$$dist_{2,\ell}(u,v,x) = \begin{cases} w(u{\to}v) & \text{if } x = 0 \\ dist_{2,\ell}(u,v,x-1) & \text{if } x \geq u \\ \min\left\{ \begin{array}{c} dist_{2,\ell}(u,v,x-1), \\ dist_{2,\ell}(u,x,x-1) + dist_{2,\ell}(x,v,x-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

However, we're a bit more restricted this time around on what evaluation orders are feasible. The restriction stems from the possibility that $x > v$ when we need to access the value $dist_{2,\ell}(u,x,x-1)$. We can no longer use evaluation order A. However, the first and third parameters remain non-increasing during recursive calls, so we *can* use evaluation order B_ℓ.

The above recurrence is not the only way to compute $dist_{2,\ell}$, though! We can instead rely on $dist_1$ as follows:

$$dist_{2,\ell}(u,v,x) = \begin{cases} w(u{\to}v) & \text{if } x = 0 \\ dist_{2,\ell}(u,v,x-1) & \text{if } x \geq u \\ \min\left\{ \begin{array}{c} dist_{2,\ell}(u,v,x-1), \\ dist_{\mathbf{1}}(u,x,x-1) + dist_{2,\ell}(x,v,x-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

Note how the first function evaluation in the second min case now uses $dist_1$. *If we first precompute all values* $dist_1(u,v,x)$, we are free to use any evaluation order to compute values of $dist_{2,\ell}$, including evaluation order A.

The computation of $dist_{2,r}$ is symmetric, resulting in a working evaluation order of B_r using an independent recurrence. There is also a recurrence that is dependent on having precomputed $dist_1$, and this second recurrence works with any evaluation order including A.

3.3 Solving $dist_3$

Finally, we return to the standard recurrence used to present Floyd-Warshall:

$$dist_3(u,v,x) = \begin{cases} w(u{\to}v) & \text{if } x = 0 \\ \min\left\{ \begin{array}{c} dist_3(u,v,x-1), \\ dist_3(u,x,x-1) + dist_3(x,v,x-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

There is no relationship between x and either of u or v, so we are forced to use an evaluation order that iterates over x in the outer loop. Evaluation order C is the natural choice.

However, we can also define an alternative recurrence that relies on $dist_{2,\ell}$:

$$dist_3(u,v,x) = \begin{cases} w(u{\to}v) & \text{if } x = 0 \\ \min\left\{ \begin{array}{c} dist_3(u,v,x-1), \\ dist_3(u,x,x-1) + dist_{2,\ell}(x,v,x-1) \end{array} \right\} & \text{otherwise} \end{cases}$$

While the possibility of $x > v$ still prevents us from iterating over v in the outermost loop, we can now use evalution order B_ℓ.

Finally, if we rely on both $dist_{2,\ell}$ *and* $dist_{2,r}$:

$$dist_3(u,v,x) = \begin{cases} w(u \to v) & \text{if } x = 0 \\ \min \begin{cases} dist_3(u,v,x-1), \\ dist_{2,r}(u,x,x-1) + dist_{2,\ell}(x,v,x-1) \end{cases} & \text{otherwise} \end{cases}$$

Now any order is fine as long as x is increasing. In particular, we can use evaluation order A again.

4 Three algorithms

The observations in the previous section suggest the following three dynamic programming algorithms for APSP.

- We can evaluate each of the functions $dist_1$, $dist_{2,\ell}$, $dist_{3,r}$ and $dist_3$ in sequence using evaluation order A. Later functions in the sequence use the recurrences that rely on earlier functions in the sequence.

- We can evaluate the functions $dist_{2,\ell}$ and $dist_3$ in sequence using evaluation order B_ℓ in a similar manner.

- Finally, we can evaluate $dist_3$ *only* using evaluation order C.

All three algorithms described above run in $\Theta(n^3)$ time and use $\Theta(n^3)$ space for their memoization data structures.

Algorithms textbooks often present this final algorithm evaluating $dist_3$ using evaluation order C as Floyd-Warshall [CLRS22, Rou22]. That said, the original presentation of the Floyd-Warshall algorithm [Flo62] does not use a memoization table indexed by all three parameters. Instead, the table is parameterized only by u and v while the x parameter is essentially ignored in all indexing operations. Correctness follows from the two-dimensional table of distances always storing values that are *at most* each value of $dist_3(u,v,x)$ after iteration x of the outer loop. These same textbooks [CLRS22, Rou22] leave this space-saving refinement as an exercise.

It turns out we can perform this trick with *all three* algorithms presented above by reusing the same table through each stage of the process. In fact, we can reduce the number of runs in the first algorithm to three by merging the computations of $dist_{2,\ell}$ and $dist_{2,r}$ into a single run. In order to more easily relate the progress of the resulting algorithms to the functions discussed in the previous section, they are *all* described below with an outer for loop over variable i that runs for one to three iterations total, ending with iteration $i = 3$. Other than the single iteration "loop" surrounding the loops over the vertices, the third algorithm APSP-C is equivalent to the common $\Theta(n^2)$-space presentation of Floyd-Warshall.

```
APSP-A(G) :
  for u ← 1 to n
      for v ← 1 to n
          dist[u, v] ← w(u→v)

  for i ← 1 to 3
      for u ← 1 to n
          for v ← 1 to n
              for x ← 1 to n
                  if dist[u, v] > dist[u, x] + dist[x, v]
                      dist[u, v] ← dist[u, x] + dist[x, v]
```

```
APSP-B(G) :
  for u ← 1 to n
      for v ← 1 to n
          dist[u, v] ← w(u→v)

  for i ← 2 to 3
      for u ← 1 to n
          for x ← 1 to n
              for v ← 1 to n
                  if dist[u, v] > dist[u, x] + dist[x, v]
                      dist[u, v] ← dist[u, x] + dist[x, v]
```

```
APSP-C(G) :
  ⟨⟨i.e., Floyd-Warshall⟩⟩
  for u ← 1 to n
      for v ← 1 to n
          dist[u, v] ← w(u→v)

  for i ← 3 to 3
      for x ← 1 to n
          for u ← 1 to n
              for v ← 1 to n
                  if dist[u, v] > dist[u, x] + dist[x, v]
                      dist[u, v] ← dist[u, x] + dist[x, v]
```

At all times, the values $dist[u, v]$ contain the length of *some* walk from u to v. However, we still need to show that these pseudocode procedures actually compute the lengths of the *shortest* walks by the time they terminate. We do so using a sequence of lemmas, all of which state that at every moment during the algorithms' execution, each $dist[u, v]$ value is no greater than the most update-to-date $dist_\bullet(u, v, x)$ value available at that moment.

We prove the first of these lemmas in detail below. The rest of the proofs are nearly identical and ⟨⟨**will**⟩⟩ appear in the appendix. ◁◁◁◁◁

Lemma 4.1. *Let $u, v, x \in V$. Immediately after iteration $(1, u, v, x)$ of* APSP-A(G)*, we have $dist[u, v] \leq dist_1(u, v, x)$.*

Proof: Assume for any $u', v', x' \in V$ with $(u', v', x') < (u, v, x)$ that immediately after iteration $(1, u', v', x')$, we have $dist[u', v'] \leq dist_1(u', v', x')$.

If $x = 1$, then $dist[u, v] = w(u{\to}v) = dist_1(u, v, 0)$ at the beginning of iteration $(1, u, v, x)$. Otherwise, the induction hypothesis guarantees $dist[u, v] \leq dist_1(u, v, x - 1)$ immediately after iteration $(1, u, v, x{-}1)$. Either way, $dist[u, v] \leq dist_1(u, v, x)$ at the beginning of iteration $(1, u, v, x)$.

Suppose $x \geq \min\{u, v\}$. During iteration $(1, u, v, x)$, value $dist[u, v]$ remains at most

$$dist_1(u, v, x-1) = dist_1(u, v, x).$$

Now, suppose $x < \min\{u, v\}$. If $x = 1$, then $dist[u, x] = w(u \to x) = dist_1(u, x, x-1)$ at the beginning of iteration $(1, u, v, x)$, and it has not increased since then. Otherwise, by the induction hypothesis, $dist[u, x] \leq dist_1(u, x, x-1)$ at the end of iteration $(1, u, x, x-1)$, and it, again, has not increased since then. Either way, $dist[u, x] \leq dist_1(u, x, x-1)$ at the beginning of iteration $(1, u, v, x)$. Similarly, $dist[x, v] \leq dist_1(x, v, x-1)$ at the beginning of the iteration. By the end of iteration $(1, u, v, x)$, value $dist[u, v]$ is at most

$$\begin{aligned}\min\left\{\begin{array}{c} dist[u, v], \\ dist[u, x] + dist[x, v] \end{array}\right\} &\leq \min\left\{\begin{array}{c} dist_1(u, v, x-1), \\ dist_1(u, x, x-1) + dist_1(x, v, x-1) \end{array}\right\} \\ &= dist_1(u, v, x).\end{aligned}$$

In both cases, $dist[u, v] \leq dist_1(u, v, x)$ at the end of iteration $(1, u, v, x)$.

□

Lemma 4.2. *Let $u, v, x \in V$. Immediately after iteration $(2, u, v, x)$ of* APSP-A(G)*, we have $dist[u, v] \leq \min\{dist_{2,\ell}(u, v, x), dist_{2,r}(u, v, x)\}$.*

Lemma 4.3. *Let $u, v, x \in V$. Immediately after iteration $(3, u, v, x)$ of* APSP-A(G)*, we have $dist[u, v] \leq dist_3(u, v, x)$.*

Corollary 4.4. APSP-A(G) *computes all-pairs shortest paths in G in $O(n^3)$ time.*

Lemma 4.5. *Let $u, v, x \in V$. Immediately after iteration $(2, u, x, v)$ of* APSP-B(G)*, we have $dist[u, v] \leq dist_{2,\ell}(u, v, x)$.*

Lemma 4.6. *Let $u, v, x \in V$. Immediately after iteration $(3, u, x, v)$ of* APSP-B(G)*, we have $dist[u, v] \leq dist_3(u, v, x)$.*

Corollary 4.7. APSP-B(G) *computes all-pairs shortest paths in G in $O(n^3)$ time.*

Lemma 4.8. *Let $u, v, x \in V$. Immediately after iteration $(3, x, u, v)$ of* APSP-C(G)*, we have $dist[u, v] \leq dist_3(u, v, x)$.*

Corollary 4.9. APSP-C(G) *computes all-pairs shortest paths in G in $O(n^3)$ time.*

References

[CLRS22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 4th edition, 2022.

[Flo62] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, June 1962.

[HKM19] Ikumi Hide, Soh Kumabe, and Takanori Maehara. Incorrect implementations of the floyd-warshall algorithm give correct solutions after three repeats. *CoRR*, abs/1904.01210, 2019.

[Rou22] Tim Roughgarden. *Algorithms Illuminated Omnibus Edition.* Cambridge University Press, 2022.