Vertex-relax proof

You

November 7, 2025

1 Our solution

We propose a shortest-path algorithm that exploits the strongly disconnected structure of directed graphs. The key idea is to decompose the graph into strongly connected components (SCCs) and process them in topological order, computing distances within each component and performing inter-component relaxations only through a reduced set of meaningful intermediate vertices rather than across all vertices. This approach extends the Floyd–Warshall relaxation principle to the SCC level, eliminating relaxations that are guaranteed to fail since updates are performed only through boundary vertices that can potentially yield shorter paths.

1.1 Formal Definition of Shortest Path via SCCs.

Let G be a directed graph partitioned into t strongly connected components $C_1, C_2, \ldots, C_t$, ordered topologically such that there are no edges from C_j to C_i for $j > i$. Further, we denote the union $C_i \cup C_{i+1} \cup \cdots \cup C_t$ by $C^{(i\ldots t)}$. Furthermore, for nodes $s, d \in C^{(i\ldots t)}$ let $D_{s,d}^{(i\ldots t)}$ denote the length of shortest path from s to d containing only nodes from $C^{(i\ldots t)}$. We shorten the notation $D_{s,d}^{(i\ldots i)}$ to $D_{s,d}^{(i)}$. Let $X \subseteq C^{(i\ldots t)}$ be a subset of vertices allowed as intermediate vertices on a path from s to d.

Lemma 1.1. *${}^{X\cup\{k\}}D_{s,d}^{(i\ldots t)}$ is the length of the shortest path in the subgraph $C^{(i\ldots t)}$ whose internal vertices belong only to the set X.*

Using this notation, we can formulate the Floyd–Warshall algorithm as follows:

Algorithm 1 Floyd–Warshall algorithm

for each node $k \in C^{(i\ldots t)}$ **do**
  for each node $s \in C^{(i\ldots t)}$ **do**
    for each node$d \in C^{(i\ldots t)}$ **do**
      ${}^{X\cup\{k\}}D_{s,d}^{(i\ldots t)} \leftarrow \min\left({}^{X}D_{s,d}^{(i\ldots t)}, {}^{X}D_{s,k}^{(i\ldots t)} + {}^{X}D_{k,d}^{(i\ldots t)}\right)$
    end for
  end for
end for

1.2 Version 1

In the Floyd–Warshall algorithm, only the order of the outer loop (the k-loop) is critical, as it iterates through the intermediate vertices. The order of the two inner loops (the source s and destination d) is irrelevant; they can be nested in any order as long as both are contained within the outer k-loop. Thus, the Floyd–Warshall algorithm can be represented as follows:

Algorithm 2 Decomposition of Floyd—Warshall algorithm

```
1:  for each node k ∈ C^(i...t) do
2:      for each node s ∈ C^(i) do
3:          for each node d ∈ C^(i) do
4:              Relax
5:          end for
6:          for each node d ∈ C^(i+1...t) do
7:              Relax
8:          end for
9:      end for
10:     for each node s ∈ C^(i+1...t) do
11:         for each node d ∈ C^(i) do
12:             Relax
13:         end for
14:         for each node d ∈ C^(i+1...t) do
15:             Relax
16:         end for
17:     end for
18: end for
```

We observe that the relaxations in line 4 occur only within a single component and can therefore be computed independently. The relaxations in line 12 never succeed, since there are no existing paths between the selected sources and destinations. Relaxations in line 7 can be resolved in the previous step, as all relevant sources, destinations, and intermediate vertices already belong to the set $C^{(i+1\ldots t)}$. The algorithm can therefore be rewritten as shown in Algorithm 3.

Algorithm 3 Version 1

```
1:  for i = t to 1 do
2:      for each node k ∈ C^(i) do
3:          for each node s ∈ C^(i) do
4:              for each node d ∈ C^(i) do
5:                  Relax
6:              end for
7:          end for
8:      end for
9:      for each node k ∈ C^(i...t) do
10:         for each node s ∈ C^(i) do
11:             for each node d ∈ C^(i+1...t) do
12:                 Relax
13:             end for
14:         end for
15:     end for
16: end for
```

1.3 Version 2

Let T denote the tails of the edges in $\mathrm{Out}(C_i)$ and H their heads. Algorithm 3 can be decomposed by first adding vertices $k \notin (T \cup H)$ to the set X, as shown in Algorithm 4.

Algorithm 4 Decomposition of version 1

```
1: for i = t to 1 do
2:     APSP inside the component
3:     for each node k ∉ (T ∪ H) do
4:         for each node s ∈ C^(i) do
5:             for each node d ∈ C^(i+1...t) do
6:                 Relax
7:             end for
8:         end for
9:     end for
10:    for each node k ∈ (T ∪ H) do
11:        for each node s ∈ C^(i) do
12:            for each node d ∈ C^(i+1...t) do
13:                Relax
14:            end for
15:        end for
16:    end for
17: end for
```

We observe that when vertices $k \notin (T \cup H)$ are added to X, relaxations in line 6 will fail, since all corresponding paths are unreachable.

Lemma 1.2. $X \in (C_i \setminus T) \cup (C^{(i+1\ldots t)} \setminus H) \implies {}^{X}D_{s,d}^{(i\ldots t)} = \infty$

The improved version is therefore presented in Algorithm 5.

Algorithm 5 Version 2

```
1: for i = t to 1 do
2:     APSP inside the C^(i)
3:     for each node k ∈ (T ∪ H) do
4:         for each node s ∈ C^(i) do
5:             for each node d ∈ C^(i+1...t) do
6:                 Relax
7:             end for
8:         end for
9:     end for
10: end for
```

1.4 Version 3

We additionally observe that sources in $C_i \setminus T$ do not yet have precomputed paths to vertices in H, and similarly, vertices in T do not yet have paths to destinations in $C^{(i+1\ldots t)}$. A decomposition of Algorithm 5 is presented in Algorithm 6.

Algorithm 6 Decomposition of version 2

1: **for** $i = t - 1$ to 1 **do**
2: APSP inside the $C^{(i)}$
3: **for** each node $k \in H$ **do**
4: **for** each node $s \in (C^{(i)} \setminus T)$ **do**
5: **for** each node $d \in C^{i+1\ldots t}$ **do**
6: *Relax*
7: **end for**
8: **end for**
9: **for** each node $s \in T$) **do**
10: **for** each node $d \in C^{i+1\ldots t}$ **do**
11: *Relax*
12: **end for**
13: **end for**
14: **end for**
15: **for** each node $k \in T$ **do**
16: **for** each node $s \in C^{(i)}$ **do**
17: **for** each node $d \in C^{i+1\ldots t}$ **do**
18: *Relax*
19: **end for**
20: **end for**
21: **end for**
22: **end for**

We note that relaxations in line 6 will not succeed since paths in these cases remain unreachable (equal to ∞). Furthermore, we can restrict destination vertices to those reachable from component C_i. Adding a pre-computation phase required by Tarjan's algorithm, the final version can be written as Algorithm 7.

Algorithm 7 Final version

1: Tarjan(G)
2: **for** $i = t - 1$ to 1 **do**
3: APSP inside the $C^{(i)}$
4: **for** each node $k \in H$ **do**
5: **for** each node $s \in T$ **do**
6: **for** each node $d \in$ reachable $C^{i+1\ldots t}$ **do**
7: *Relax*
8: **end for**
9: **end for**
10: **end for**
11: **for** each node $k \in T$ **do**
12: **for** each node $s \in C^{(i)}$ **do**
13: **for** each node $d \in$ reachable $C^{i+1\ldots t}$ **do**
14: *Relax*
15: **end for**
16: **end for**
17: **end for**
18: **end for**

We conclude this section with the time complexity analysis. We exclude the time required to compute APSP within each component. For each of the t components, the algorithm iterates through all tails in T, relaxing them through all heads in H toward all reachable destinations in $C^{(i+1\ldots t)}$ (denoted C_d). Next, all sources in the component are relaxed through all tails toward all reachable destinations. The resulting time complexity is:

$$t \times (T \times H \times C_d + C_s \times T \times C_d),$$

which simplifies to:

$$t \times C_d \times T \times (H + C_s).$$