Strictly Second Shortest Simple Path

Yangxinyu Xie
Supervised by Dr. Vijaya Ramachandran

Summer 2020

1 The Directed Disjoint Shortest Paths Problem **2**
1.1 Directed k Disjoint Shortest Paths Problem is NP-Hard . . . 2
1.2 All Cycles are Positive . . . 2
1.3 G is Planar and Undirected . . . 5

2 The Directed Subgraph Homeomorphism Problem **7**
2.1 "Tree-like" Fixed SHP is in P . . . 7
2.2 General fixed SHP is NP-hard . . . 8
2.3 Pebbling Game and Directed Acyclic Input Graph . . . 12
2.4 Open Problems . . . 13

3 Finding k Cycle-free Shortest Paths **14**

4 Computing Strictly Second Shortest Path **15**
4.1 Strictly Second Shortest Simple Path is NP-Complete . . . 15
4.2 Strictly Second Shortest Path is in P . . . 15

5 Undirected Strictly Second Shortest Simple Path Problem is in P **18**

6 Fine-Grained Complexity for Weighted Directed Sparse Graphs **20**

7 Finding Replacement Paths in Unweighted Directed Graphs in Expected $\tilde{O}(m\sqrt{n})$ Time **21**
7.1 Finding Short Detours . . . 21
7.2 Finding Long Detours . . . 22
7.3 Finding k Shortest Simple Paths . . . 22

Last updated August 20, 2021

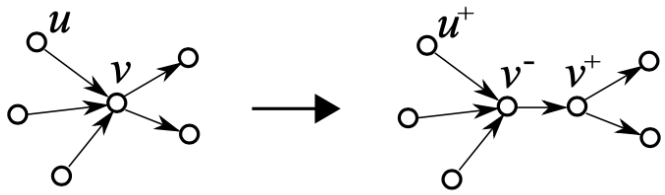


Figure 1.1: Reduction from vertex disjoint to edge disjoint.

1 The Directed Disjoint Shortest Paths Problem

Consider the *Directed k Disjoint Shortest Paths Problem*:
Given a digraph $G = (V, E)$ with a length function $l : E \to \mathbb{R}^+$ and k pairs of vertices $(s_1, t_1), ..., (s_k, t_k) \in V$. Find pairwise disjoint (vertex-disjoint or edge-disjoint) paths $P_1, ..., P_k$ such that P_i is a shortest path from s_i to t_i for $i \in [k]$ if they exists.

Notice that the vertex disjoint version and the edge disjoint version of the k disjoint shortest paths problem are equivalent, as shown by the reduction in the Figure 1.1. Hence, for simplicity, by *directed k disjoint shortest paths problem* we mean the edge disjoint version.

1.1 Directed k Disjoint Shortest Paths Problem is NP-Hard ❖

Theorem 1.1. *The directed k disjoint shortest paths problem are NP-hard even when $k = 2$.*

Proof. Notice that the directed 2 disjoint paths problem is NP-hard [FHW80]. Hence, it suffices to set edge length to 0. □

1.2 All Cycles are Positive ❖

Theorem 1.2 ([BK17])**.** *If the length of each cycle is positive, the directed 2 disjoint paths problem can be solved in polynomial time.*

Proof. Let E_i be the set of edges contained in some **shortest path** from s_i to t_i. This can be obtained efficiently using Dijkstra's algorithm: let $d_i(v)$ denote the distance from s_i to v, then an edge $e = (u, v)$ is in the set E_i if and only if $l(e) = d_i(v) - d_i(u)$.

Claim 1.3. The edge set E_i contains no cycle for $i \in [2]$.

Proof. Suppose on the contrary that there is a cycle $C \in E_i$, then we have

$$l(C) = \sum_{e=(u,v)\in C} l(e) = \sum_{(u,v)\in C} d_i(v) - d_i(u) \tag{1.2.1}$$

which is 0. This contradicts our assumption that the length of each cycle is positive. ■

Notation 1.4. For a set F of directed edges, we let $\overline{F}$ denote the set of directed edges obtained from F by reversing the direction of each edge.

Claim 1.5. Suppose that C is a cycle in $E_1 \cup \overline{E_2}$. Then, $E_1 \cap E(C) \subseteq E_2$ and $E_2 \cap \overline{E(C)} \subseteq E_1$.

Proof. Let C be a cycle in $E_1 \cup \overline{E_2}$. Notice that there exists a natural number $1 \leq r \leq \lfloor |C|/2 \rfloor$ such that C can be partitioned into subpaths $P_1, \overline{Q_1}, P_2, \overline{Q_2}, ..., P_r, \overline{Q_r}$ with $P_i \subseteq E_1, Q_i \subseteq E_2$ for all $i \in [r]$.

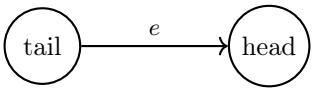


Figure 1.2: Head and tail of an edge

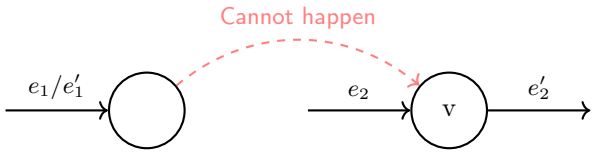


Figure 1.3: Case 1. $e_1 = e_1'$, $\text{head}_{G^*}(e_2) = \text{tail}_{G^*}(e_2') = v$ and there is no path in G^* from $\text{head}_{G^*}(e_1)$ to v.

Suppose for each i, P_i is a path from u_i to v_i and denote $u_{r+1} := u_1$. Then, we have $\overline{Q_i}$ is a path from v_i to u_{i+1}. In other words, Q_i goes from u_{i+r} to v_i. By the definition of d_1 and E_1, we have that

$$\sum_{i=1}^{r} l(P_i) = \sum_{i=1}^{r} d_1(v_i) - d_1(u_i) = \sum_{i=1}^{r} d_1(v_i) - d_1(u_{i+1}) \geq \sum_{i=1}^{r} l(Q_i) \tag{1.2.2}$$

By symmetry, we have

$$\sum_{i=1}^{r} l(Q_i) = \sum_{i=1}^{r} d_2(v_i) - d_2(u_{i+1}) = \sum_{i=1}^{r} d_2(v_i) - d_2(u_i) \geq \sum_{i=1}^{r} l(P_i) \tag{1.2.3}$$

Hence, it must be the case that for all $i, d_1(v_i) - d_1(u_{i+1}) = l(Q_i), d_2(v_i) - d_2(u) = l(P_i)$. Thus, by the definition of E_i, we have for all $i, E(Q_i) \subseteq E_1, E(P_i) \in E_2$. ■

We define the following:

- $E_0 := E_1 \cap E_2$.
- $E_1^* := E_1 \setminus E_0$.
- $E_2^* := E_2 \setminus E_0$.

We apply the following three steps on G and obtain a reduced graph $G^* = (V^*, E^*)$:

- **Remove** all edges in $E \setminus (E_1 \cup E_2)$.
- **Contract** all edges in E_0.
- **Reverse** all edges in E_2^*.
- **Add** s_1', t_1', s_2', t_2' to V^* and $(s_1', s_1), (t_1, t_1'), (t_2', t_2), (s_2, s_2')$ to E^*. This is only for auxiliary purposes.

Claim 1.6. G^* is acyclic.

Proof. Notice that E_1 and E_2 are acyclic. Hence, if there is a cycle $C \in G^*$, then C must be contained in $E_1 \cup \overline{E_2}$, then by claim 1.5, C is entirely contained in E_2, which is acyclic. ■

Notation 1.7. For an edge $e = (u, v) \in G$, we define $\text{head}_G(e) := v$, $\text{tail}_G(e) := u$, as shown in figure 1.2.

We now construct a graph $\mathcal{G}$ with vertex set $W = E_1^* \times \overline{E_2^*}$ and for a pair $(e_1, e_2), (e_1', e_2') \in W$, there is an edge from (e_1, e_2) to (e_1', e_2') in $\mathcal{G}$ if one of the following holds:

- $e_1 = e_1'$, $\text{head}_{G^*}(e_2) = \text{tail}_{G^*}(e_2') = v$ and there is no path in G^* from $\text{head}_{G^*}(e_1)$ to v, as shown in figure 1.3.

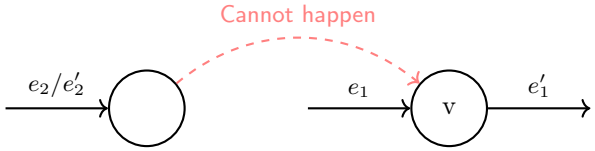


Figure 1.4: Case 2. $e_2 = e_2'$, $\text{head}_{G^*}(e_1) = \text{tail}_{G^*}(e_1') = v$ and there is no path in G^* from $\text{head}_{G^*}(e_2)$ to v.

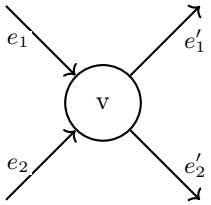


Figure 1.5: Case 3. $\text{head}_{G^*}(e_1) = \text{head}_{G^*}(e_2) = \text{tail}_{G^*}(e_1') = \text{tail}_{G^*}(e_1') = v$.

- $e_2 = e_2'$, $\text{head}_{G^*}(e_1) = \text{tail}_{G^*}(e_1') = v$ and there is no path in G^* from $\text{head}_{G^*}(e_2)$ to v, as shown in figure 1.4.
- $\text{head}_{G^*}(e_1) = \text{head}_{G^*}(e_2) = \text{tail}_{G^*}(e_1') = \text{tail}_{G^*}(e_1') = v$, as shown in figure 1.5.

Claim 1.8. There is a path in $\mathcal{G}$ from $((s_1', s_1), (t_2', t_2))$ to $((t_1, t_1'), (s_2, s_2'))$ if and only if G has two edge disjoint paths P_1 and P_2 such that for $i = 1, 2$, P_i goes from s_i to t_i and $E(P_i) \subseteq E_i$.

Proof. ($\Leftarrow$) Suppose that G has such two edge disjoint paths P_1 and P_2. Then, by constructing G^*, there is are two disjoint paths P_1^* from s_1 to t_1 in G^* with $E(P_1^*) \subseteq E_1 \setminus E_0$ and P_2^* from t_2 to s_2 in G^* with $E(P_2^*) \subseteq E_2 \setminus E_0$. Let $P_1^* = e_1^1 e_1^2 ... e_1^p$ with $p = |P_1^*|$ and $P_2^* = e_2^1 e_2^2 ... e_2^q$ with $q = |P_2^*|$. Because G^* is acyclic by claim 1.6, for each $i \in [p], j \in [q]$, at least one of the following holds:

- There is no path in G^* from $\text{head}_{G^*}(e_1^i)$ to $\text{head}_{G^*}(e_2^j)$, as figure 1.3. Then there is an edge in $\mathcal{G}$ from (e_1^i, e_2^j) to (e_1^i, e_2^{j+1}).
- There is no path in G^* from $\text{head}_{G^*}(e_2^j)$ to $\text{head}_{G^*}(e_1^i)$, as figure 1.4. Then there is an edge in $\mathcal{G}$ from (e_1^i, e_2^j) to (e_1^{i+1}, e_2^j).
- $\text{head}_{G^*}(e_1^i) = \text{head}_{G^*}(e_2^j)$, as figure 1.5. Then there is an edge in $\mathcal{G}$ from (e_1^i, e_2^j) to (e_1^{i+1}, e_2^{j+1}).

Hence, by tracing P_1^* and P_2^* incrementally we obtain a path from $((s_1', s_1), (t_2', t_2))$ to $((t_1, t_1'), (s_2, s_2'))$ in $\mathcal{G}$.

($\Rightarrow$) Suppose that is a path Φ in $\mathcal{G}$ from $((s_1', s_1), (t_2', t_2))$ to $((t_1, t_1'), (s_2, s_2'))$. Hence, by tracing Φ step by step we obtain two disjoint paths P_1^* and P_2^* in G^*. Because G^* is equivalent to G for our purpose, there are two disjoint paths P_1 and P_2 in G.

■

Since $\mathcal{G}$ contains at most $|E|^2$ edges, we can find a path in $\mathcal{G}$ in polynomial time. Notice that E_1 and E_2 only contain edges in the shortest paths, hence we can correctly find the two disjoint shortest paths in polynomial time. □

Corollary 1.9. *If each edge has a positive length, the undirected k-disjoint shortest path problem can be solved in polynomial time.*

Proof. As shown in figure 1.6, it suffices to break e into five edges and set the length of each newly created edge to $l(e)/3$. □

Remark 1.10. A similar scheme to what we have just presented can also be used for the scenario below, given that k_1 and k_2 are fixed and each cycle is postive:

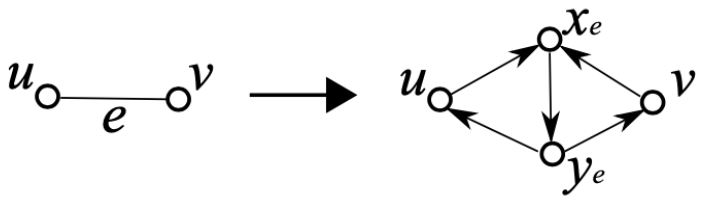


Figure 1.6: Reduction from undirected case to directed case.

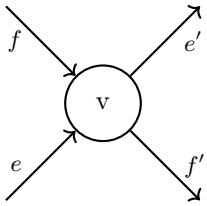


Figure 1.7: Two paths cross

Given a digraph $G = (V, E)$ with a length function $l : E \to \mathbb{R}^+$ and 2 pairs of vertices $(s_1, t_1), (s_2, t_2) \in V$. Find pairwise disjoint paths $P_1^1, ..., P_{k_1}^1, P_2^2, ..., P_{k_2}^2$ such that P_j^i is a shortest path from s_i to t_i for $i \in [2]$ and $j \in [k_i]$ if they exists.

1.3 G is Planar and Undirected ❖

Definition 1.11. A graph G is called **planar** if there is an embedding ϕ of G to a surface $\Sigma \subseteq S_3$ such that Σ homeomorphic to S_2. That is, every embedding of G in S_3 does not contain a link.

To solve the vertex disjoint version of the k disjoint shortest paths problem, we can restrict G on the edge set $E_1 \cup E_2 \cup ... \cup E_k$. In this way, it suffices to find k disjoint paths, which is shown to be solved in polynomial time in [Sch94]. Again, for simplicity, by k-disjoint shortest path problem we mean the edge disjoint version.

Definition 1.12. We say that two edge-disjoint paths P_1 and P_2 crosses each other if $e, e' \in E(P_1), f, f' \in E(P_2)$ and e, f, e', f' are incident to v clockwise as in figure 1.7.

Proposition 1.13. *If k is a fixed constant and G is planar, then the k-disjoint and non-crossing shortest path problem can be solved in polynomial time.*

Proof. We construct a new G' from G by applying the following:

- Reduce G to the case when each terminal node is of degree one. One approach is to simply guess the first and last edge of P_i and delete all other edges incident to the terminals. There are at most $O(|V|^k)$ such possibilities.
  (I personally reckon it is simpler to just replace a terminal by an extra vertex and an edge. For instance, if s_i has degree greater than 1, we can simply add s_i' and edge $e = (s_i', s_i)$ to the graph with zero length.)
- For each vertex of degree $d_v > 3$, we simply replace v by a d_v-gon, with each vertex connected to 1 exterior edge, as shown in figure 1.8.

Doing so, the new planar graph $G' = (V', E')$ has terminals of degree 1 and each vertex in V' has degree at most 3. Therefore, for any two paths with one-degree terminals are edge-disjoint if and only if they are vertex disjoint. Hence, the undirected edge disjoint version of the k-disjoint and non-crossing shortest path problem can be solved in polynomial time. □

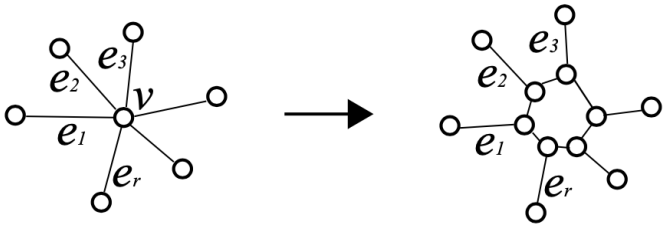


Figure 1.8: v to d_v-gon.

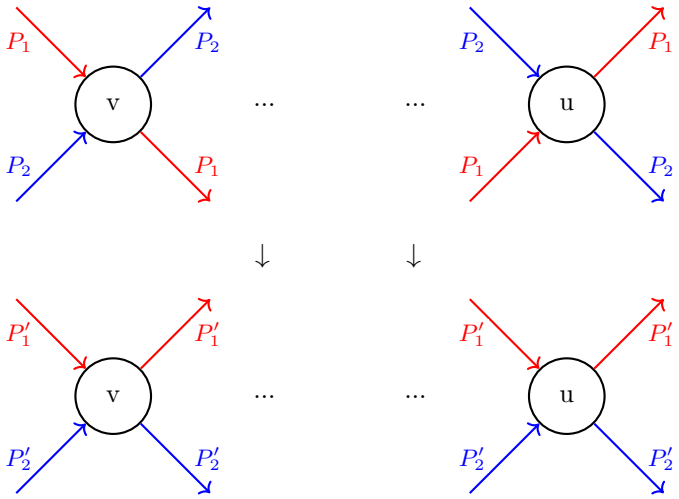


Figure 1.9: Reduce cross

Lemma 1.14. *Suppose that k is a fixed constant and G is planar. There exists a solution $P_1, ..., P_k$ such that for each pair $i, j \in [k], i \neq j$, P_i and P_j cross at most once, if any solution exists.*

Proof. Suppose that for some $i, j \in [k], i \neq j$, P_i and P_j cross at some vertices $v, u, v \neq u$. Hence, because P_i and P_j are disjoint, there exist two disjoint subpath $Q_1 \subseteq P_1, Q_2 \subseteq P_2$ from v to u. Since both P_i and P_j are the shortest path, we must have $l(Q_1) = l(Q_2)$. Hence, simply exchanging Q_1 and Q_2, i.e. letting $P_1' = P_1 \cup Q_2 \setminus Q_1, P_2' = P_2 \cup Q_1 \setminus Q_2$, gives us two new paths that do not cross at u or v, as shown in figure 1.9. By repeating this process, we will have a solution with at most 1 cross. □

Theorem 1.15. *If k is a fixed constant and G is planar, then the k-disjoint shortest path problem can be solved in polynomial time.*

Proof. Notice that each P_i only crosses other paths at at most $r_i \leq k - 1$ vertices. Given these r_i vertices, P_i can be partitioned to r_{i+1} subpaths, each of which parallels with an edge disjoint subpath of some other path P_j. Hence, we can see that the following algorithm runs in polynomial time:

1. For $i = 1, ..., k$, guess an integer $r_i \leq k - 1$ and crossing vertices $u_1^i, u_2^i, ..., u_{r_i}^i$.
2. Find pairwise disjoint shortest paths Q_j^i.
3. Check $P_i = Q_1^i Q_2^i ... Q_{r_i+1}^i$ is a solution.

□

2 The Directed Subgraph Homeomorphism Problem

Definition 2.1 (Graph Homeomorphism)**.** Given a pattern graph P and an input graph G, we say that P is **homeomorphic** to a subgraph $G' \subseteq G$, written as $P \simeq G'$ if there is a map $\varphi : P \to G$ that maps the *nodes* of P to *nodes* of G' and *edges* of P to *simple paths* in G' and the paths in G' corresponding to edges in P must be *pairwise node-disjoint*. We call such a map φ : a **graph homeomorphism** from P to $G' \subseteq G$.

Definition 2.2 (Subgraph Homeomorphism Problem)**.** Given a fixed pattern graph P and an input graph G, the **subgraph homeomorphism problem** (SHP) aims to determine if P is **homeomorphic** to a subgraph $G' \subseteq G$.

Definition 2.3 (Fixed Subgraph Homeomorphism Problem)**.** Given a fixed pattern graph $P = (V_P, E_P)$, an input graph $G = (V_G, E_G)$ and a map $\pi : V_P \to V_G$, the **fixed subgraph homeomorphism problem** (fixed SHP) aims to determine if there is a map $\varphi : P \to G$ that extends π naturally and is a graph homeomorphism from P to $G' \subseteq G$.

Definition 2.4. Let $\mathcal{C}$ denote the set of all directed graphs with a distinguished node called the **root** with the property that

- the root is the head of every edge; Or
- the root is the tail of every edge.

Equivalently, a graph $G \in \mathcal{C}$ if when all loops at the root are deleted and multiple edges between pairs of nodes are merged into single edges, the resulting graph is a *tree of height at most 1*.

2.1 "Tree-like" Fixed SHP is in P ❖

Theorem 2.5 ([FHW80])**.** *For each $P \in \mathcal{C}$, there is a polynomial time algorithm for the fixed SHP with pattern graph P.*

The key idea is that Maximum-Flow Problem can be solved in polynomial time.

Proof. Let $P = (V_P, E_P)$, $G = (V_G, E_G)$ and $\pi : V_P \to V_G$ be given. We prove for the case that $P \in \mathcal{C}$ and all edges are directed away from the root $r \in V_P$.

- **Eliminate self loops at r.** If there are loops at the root r, we can obtain an equivalent problem without loops at r. As shown in figure 2.1, we split r into a new root r' and a new leaf l with the loop edges directed from r' to l and all other incident edges from r incident from r'; in the input graph G, we split $\pi(r)$ into $\pi'(r')$ and $\pi'(l)$, where all edges incident to $\pi(r)$ are now incident to $\pi(l)$ and all edges incident from $\pi(r)$ incident from $\pi(r')$.
- **Create a flow network.** Now we give a capacity function $c : G \to \mathbb{Z}$ as the following
  1. let $\pi(r)$ be a source and set $c(\pi(r)) =$ the out degree of r in P;
  2. for all $v \in V_P, v \neq r$, let $\pi(v)$ be a sink with $c(\pi(v)) =$ the in degree of v in P;
  3. for all $u \in V_G, u \notin \pi(V_P)$, set $c(u) = 1$;
  4. for all edges $e \in E_G$, set $c(e) = 1$.

  It remains to decide if there is a flow in G equal to the capacity of the source. Notice that $P \simeq G' \subseteq G$ if and only if such a flow exists.

□

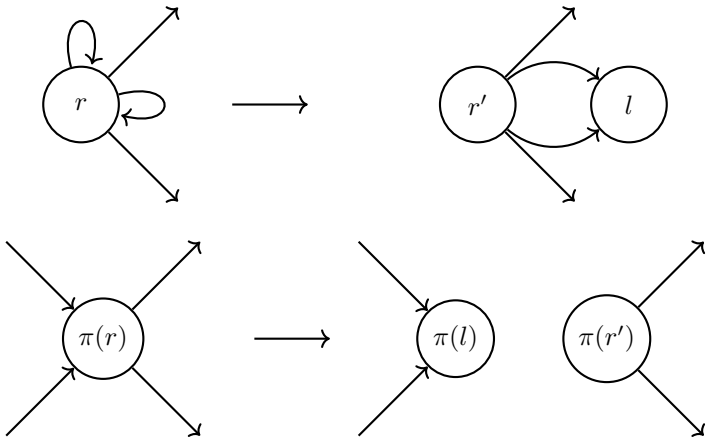


Figure 2.1: There is some loop at $r \in V_P$

2.2 General fixed SHP is NP-hard ❖

Theorem 2.6 ([FHW80])**.** *For each $P \notin \mathcal{C}$, the fixed SHP with pattern graph P is NP-complete.*

To prove this theorem, we first introduce some auxiliary lemmas.

Lemma 2.7. *Suppose P is a subgraph of Q and the fixed SHP is NP-hard with pattern P. Then, the fixed SHP is NP-hard with pattern Q.*

The goal of our proof is the following: given $P = (V_P, E_P)$, $G = (V_G, E_G)$ and $\pi : V_P \to V_G$, we construct a new graph H and a mapping $h : V_Q \to V_H$ in polynomial time such that

$$P \simeq G' \subset G \Leftrightarrow Q \simeq H' \subset H \tag{2.2.1}$$

Proof. Let $Q \setminus P$ denote the graph with edges in Q but not in P, together with the incident vertices. We construct a graph $H := G \cup (Q \setminus P)$ and extend the map h from π naturally. Clearly, $P \simeq G' \subset G \to Q \simeq H' \subset H$.
It remains to show the converse by induction on the number of edges in $Q \setminus P$. If $Q \setminus P = \emptyset$, the statement holds vacuously. Inductively, suppose $Q \setminus P \neq \emptyset$ and $Q \simeq H' \subset H$. Now let $\phi : Q \to H$ be a graph homeomorphism extending h and let ϕ_P be the map restricted to P.

- If for each $e \in E_P, \phi_P(e) \in G$, then there is a subgraph $G' \subset G$ with $P \simeq G'$ and we are done.
- If there is some edge $e \in E_P$ such that $\phi_P(e) \notin G$. Then there is some edge $f' \in \phi_P(e) \subset H$ but $f' \notin G$.
  - Notice that both of the end points of f' must be in G; otherwise, it must be the case $e \in Q \setminus P$.
  - Then there exists an edge $f \in Q \setminus P$ which forced f' to be added to H. This implies that $f' = \phi(f)$; otherwise ϕ fails to map in edges in Q to *pairwise node-disjoint* paths in H. Thus, we have f and e are disjoint and parallel in Q.
  - Let $\varphi : Q \to H$ be such that $\varphi(e) = \phi(f), \varphi(f) = \phi(e)$ and $\varphi(g) = \phi(g)$ for all $g \in E_Q$, we obtain that φ_P is a graph homeomorphism from P to G.

□

Definition 2.8 (Switch)**.** Consider the subgraph in figure 2.2. Suppose that there are two node-disjoint paths passing through the subgraph, one leaving node A and the other entering at B. Then the path leaving at A must have entered at C and the path entering at B must leave at D. There is exactly one additional path though the subgraph and it is either

$$8 \to 9 \to 10 \to 4 \to 11 \quad \text{or} \quad 8' \to 9' \to 10' \to 4' \to 11' \tag{2.2.2}$$

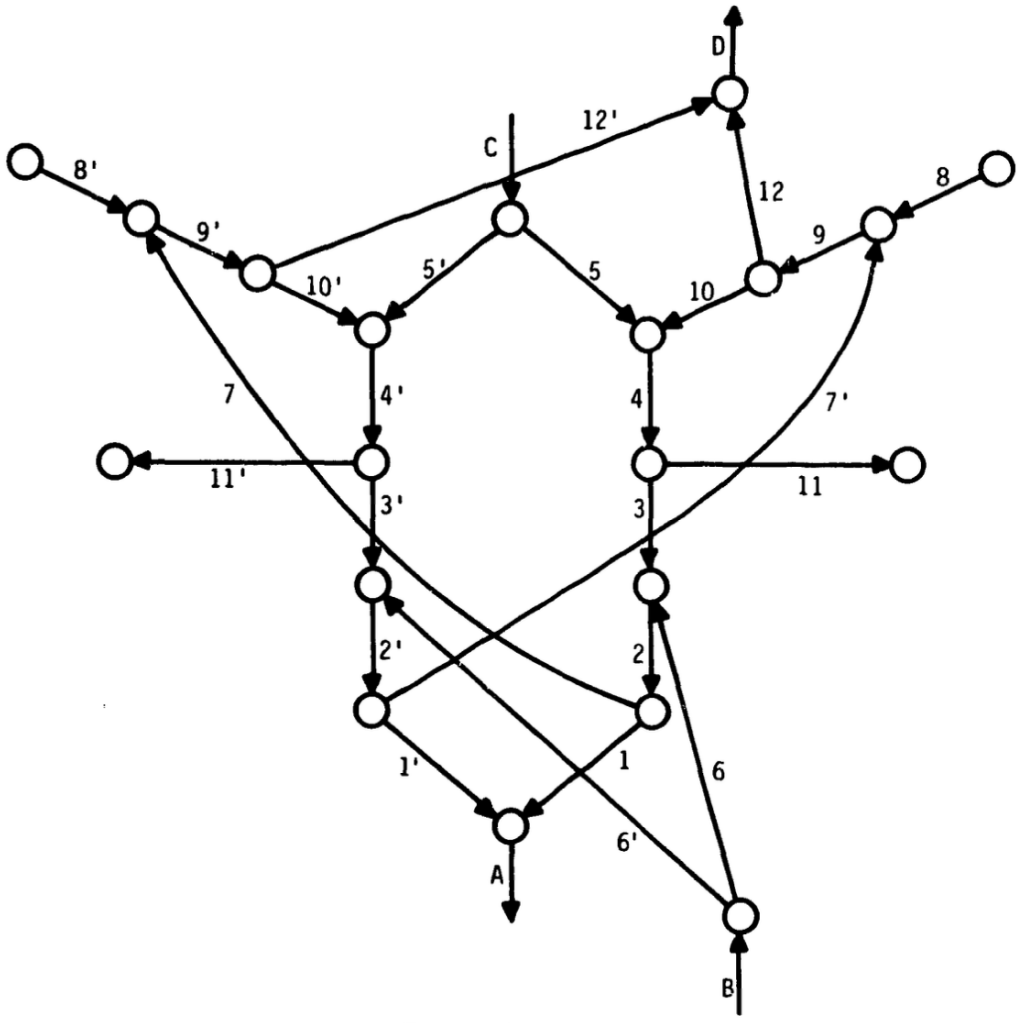


Figure 2.2: Switch.

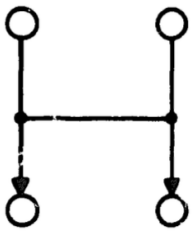


Figure 2.3: Schematic Representation of a Switch

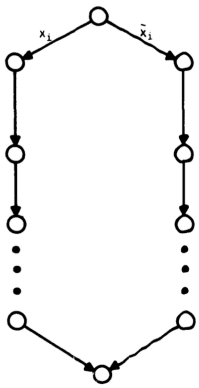


Figure 2.4: Subgraph associated with each literal in G_f

depending on the actual routing of the path leaving A. We call such a subgraph a **switch**. For simplicity, we represent a switch schematically as figure 2.3, where the horizontal line, *which is not an edge*, indicates that at most *one* of the vertical edges can be used.

Lemma 2.9. *Let P consist of two disjoint directed edges and the four incident vertices. Then the the fixed SHP with pattern P is NP-hard.*

This lemma says that finding two disjoint paths in a graph is NP-hard. The main idea is to reduce the 3-SAT (SAT-3-CNF) to the fixed SHP with pattern P.

Proof. Let a 3-CNF formula F with variables $x_1, ..., x_k$ and clauses $t_1, ..., t_l$. We construct a graph G_F:

- For each x_i, we create a copy of subgraph as in figure 2.4. We associate the left column of the vertical edges with literal x_i and the other $\bar{x}_i$. The number of edges in each column is exactly the number of occurrences of its associated literal in F. For each x_i, we have that the bottom node of this subgraph associated with x_i is connected to the top node of that of x_{i+1}. See figure 2.5.
- There are also nodes $n_0, ..., n_l$. For each $i \in [l]$, the three edges from n_i to n_{i+1} corresponds with clause t_i. See figure 2.5.
- There is also an edge from the bottom of the subgraph of x_k to n_0.
- For each literal $y \in t_i$, we replace one of the edges between n_{i-1} to n_i by one of the edges in the column associated with y by a switch.
- Add nodes W, X, Y and Z. The edge from Y is a B-input edge of the first switch and D the output of the last switch; similarly, the edge from W is the C input of the last switch and A the output of the first. *We concatenate two switches by merging edge C and D of the ith switch to edge A and B of the $i+1$th switch.*

Claim 2.10. There are node-disjoint paths from W to X and from Y to Z in G_F if and only if F is satisfiable.

Proof. ($\Leftarrow$) Suppose F is satisfiable. For each literal $y \in t_i$, if y is true, then the path from Y to Z can go through $\bar{y}$. Since each clause has at least 1 literal set to true, there will be at least 1 simple path from n_{i-1} to n_i. By the definition of a switch, This process determines the path from Y to Z and that from W to X.

($\Rightarrow$) Suppose there exist two node-disjoint paths from W to X and from Y to Z in G_F. For each literal $y \in t_i$, if the path from Y to Z passes the column associated to $\bar{y}$, then y is set to true and vice versa. Because for each i, the subpath from n_{i-1} to n_i is in the path from Y to Z, we have that the assignment to each y must satisfy F.

■

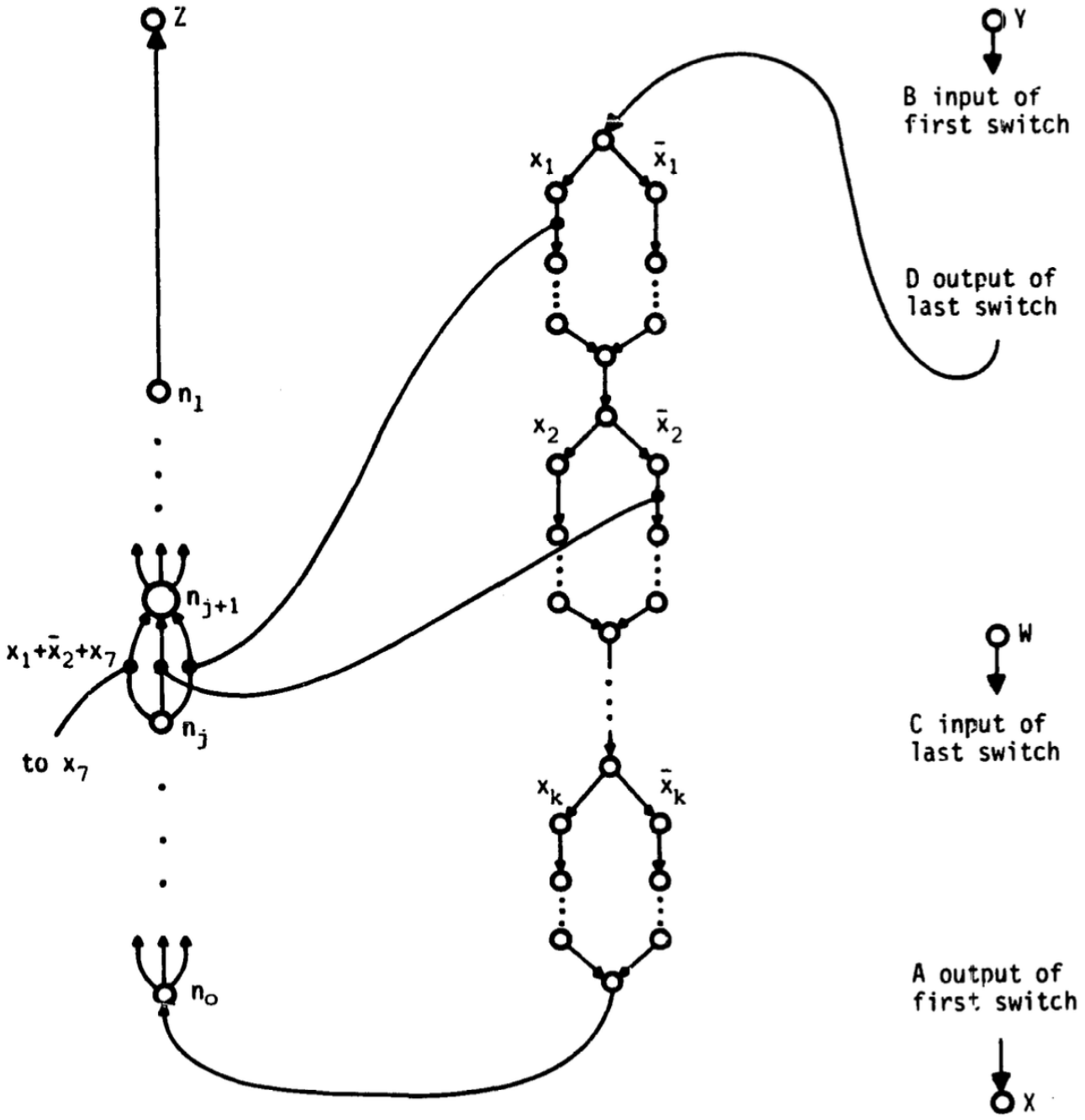


Figure 2.5: An Example of G_f

Because the construction of G_F can be done in polynomial time, we have that the fixed SHP with pattern P is NP-hard. □

proof of Theorem 2.6. Let $P \notin \mathcal{C}$ be given. Because P is not "tree-like", then P must contain one of the following:

1. two edges e_1 and e_2 with disjoint end vertices, one or both of which may be loops. If neither edges are loops, then by lemma 2.9, finding two disjoint paths in G NP-hard. If e_1 is a loop, then identify W with X; if e_2 is a loop, then identify Y with Z.
2. two consecutive edges that forms a path on 3 vertices. In this case, it suffices to identify X with Y.
3. a cycle of length 2. In this case, it suffices to identify W with Z and X with Y.

By lemma 2.7, since each of the three subgraphs in P makes the fixed SHP NP-hard, the fixed SHP is NP-hard with pattern P. □

2.3 Pebbling Game and Directed Acyclic Input Graph ❖

In this section we assume the input graph G is acyclic, which allows us to give the following definition.

Definition 2.11. For a node $v \in G$, we define the **level** of v, denoted $l(v)$, as the length of a longest path in G starting from v.

Definition 2.12 (Pebbling Game)**.** Let $P = (V_P, E_P)$, $G = (V_G, E_G)$ and $\pi : V_P \to V_G$ be given. The rules of the **pebbling game** are given below:

1. For each edge $e_i \in E_P$ there is a corresponding pebble. At the beginning of the game, we place the pebble p_i of e_i on $\pi(\text{tail}(e))$.
2. At any step a pebble p_i may be moved along an edge $(u, v) \in E_G$ if:
   (a) $l(u)$ is the largest among all nodes with a pebble on it;
   (b) v has no pebble on it;
   (c) $v \notin \pi(P)$ unless $v = \pi(\text{head}(e_i))$.
3. If the pebble p_i is moved to $\pi(\text{head}(e_i))$, remove it.

Lemma 2.13. *The pebbling game can be won if and only if $P \simeq G' \subseteq G$.*

Proof. ($\Leftarrow$) Suppose that $P \simeq G' \subseteq G$. Then, by following the disjoint paths in G' according to the specified rules we can win the game.

($\Rightarrow$) Suppose that pebbling game can be won by travelling each p_i through each path q_i. It remains to show that all q_i are disjoint except for the end vertices.
Suppose that the pebbles p_i and p_j intersect at some point v other than the end nodes.

- Note that $v \notin \pi(P)$.
- Because at each step we only move 1 pebble, we can assume that p_i visits v first. Then, p_i leaves v first because p_j cannot reach v is p_i is on v.
- However, this cannot be the case because $l(v)$ cannot be the largest among all nodes with a pebble on it as p_j has not reached v yet. Hence, all q_i must be disjoint.

□

Theorem 2.14 ([FHW80])**.** *For a fixed directed pattern graph P, there is a polynomial time to decide if $P \simeq G' \subseteq G$ when an input DAG G is given.*

Proof. Assume that $\pi : V_P \to V_G$ is given. Let $k = |E_P|, m = |V_P|$, and $n = |V_G|$. Then there exist at most $(n+1)^k$ ways to put k or fewer pebbles on G. Thus, there are at most $(n+1)^k$ configurations of the pebbling game. Therefore, for each pair of configurations $\mathcal{G}$ and $\mathcal{G}'$, we draw an edge $(\mathcal{G}, \mathcal{G}')$ if we can move from $\mathcal{G}$ to $\mathcal{G}$ in one single step according to the rules of the pebbling game. Then, any path finding algorithm can decide if there is a path from the starting configuration to the winning configuration.
Notice that because P is fixed, there are at most $m!\binom{n}{m}$ possible choices for π, which is polynomial in n. □

Remark 2.15. Note that the proof of theorem 1.2 is similar to this one. Note that when P is not fixed, this problem is NP-complete despite whether G is acyclic, as implied in [EIS75].

2.4 Open Problems ❖

As for undirected graphs, we do not know how to construct a switch as in definition 2.8 to prove that the undirected fixed SHP is NP-hard.
It might be the case that undirected fixed SHP is in P. In [LR80], LaPaugh and Rivest have shown that the undirected fixed SHP with a pattern consisting of a cycle of length 3 can be solved in polynomial time; in [Shi78], Shiloach showed that for P consisting of two disjoint edges, the undirected fixed SHP can be solved in polynomial time.
As we have seen in section 1.3, by exploiting the geometric properties of planar graphs, finding two disjoint paths in G when G is planar and undirected can be solved in polynomial time. However, can the general directed fixed SHP be solved in polynomial time?

3 Finding k Cycle-free Shortest Paths

In this section we present a polynomial time algorithm that finds k cycle-free shortest paths in an input graph G with no negative cycles [Yen71]. Given an input graph $G = (V, E)$ with $|V| = n$, a source s and a sink t, we let π^i denote the i-th shortest path from s to t. We don't allow multiples edges between a pair of vertices.

Definition 3.1. We say that a path p from s to t is a **deviation from** π^{i-1} at v if $p \cap \pi^{i-1}$ is a subpath from s to v and for an edge $e = (v, u) \in p$, we have that $e \notin \pi^j$ for any $j \in [i-1]$.

Notation 3.2. Let subpath $p_v^i \subseteq \pi^i$ denote the subpath from s to v in π^i.

In the following algorithm, given $u, v \in V$, we can assume an oracle $\mathcal{O}$ outputs a shortest path from u to v in G if it exists. This can be implemented using Dijkstra's algorithm ($O(m + n \log n)$) or Bellman-Ford algorithm ($O(mn)$).

Algorithm 3.3. Given $k = O(1), G, s, t$, output $\pi^1, ..., \pi^k$.

1. Find the shortest path π^1 from s to v. Let set $S = \emptyset$.
2. **for** $i = 2$ to k:

    for $v \in \pi^{i-1}, v \neq t$:

        Let G' be a copy of G.

        for $j = 1$ to $i - 1$:

            if $p_v^{i-1} = p_v^j$, delete the edge $e = (v, u) \in \pi^j$ from G'.

        Delete p_v^{i-1}, excluding the vertex v, from G'.

        Find a shortest path q_v^i from v to t in G'.

        $S \leftarrow S \cup \{p_v^{i-1} \circ q_v^i\}$. If $|S| > k - i + 1$, delete the longest paths in S.

    Choose the shortest path $\pi \in S$, then set $\pi^i = \pi$.

    $S = S \setminus \{\pi^i\}$.

proof of correctness of Algorithm 3.3. It suffices to show that for each i, ith iteration correctly obtains a shortest path π^i from s to t. Let $i \in [k]$ be given. Then $\pi^1, ..., \pi^{i-1}$ are found and S be a collection of deviations obtained in the ith iteration. Let v be a vertex in $\pi^{i-1}, v \neq t$. Because we delete every edge following v in any of the shortest path $\pi^1, ..., \pi^{i-1}$, $p_v^i \circ q_v^i$ cannot be the same as any of the $\pi^1, ..., \pi^{i-1}$.
Let $l(p)$ denote the length of a path p and let $\hat{p}$ be an ith shortest path. It remains to show that $l(\hat{p}) = l(\pi^i)$. Suppose $\hat{p} \notin S$. If $\hat{p}$ does not share any subpath from s to some $v \in \pi^i$ with any of $\pi^1, ..., \pi^{i-1}$, then there is a path $q_s^i \in S$ such that $l(q_s^i) \leq l(\hat{p})$. On the other hand, let v denote the last vertex in $\hat{p}$ such that $\hat{p} \cap \pi^j = p_v^j$ for some $j \in [i-1]$. Then we have that $l(p_v^j) + l(q_v^j) \leq l(\hat{p})$ and $p_v^j \circ q_v^j \in S$. Hence, $l(\hat{p}) \geq l(\pi^i)$. □

4 Computing Strictly Second Shortest Path

In this section we present the results in [LP97].

4.1 Strictly Second Shortest Simple Path is NP-Complete ❖

Definition 4.1 (Strictly Second Shortest Simple Path Problem)**.** Let $G = (V, E)$ be a weighted directed graph and s, t be two distinct vertices in V such that there is a shortest path p_1 from s to t. The **Strictly Second Shortest Simple Path Problem** computes a simple path p_2 from s to t such that

$$l(p_2) > l(p_1) \tag{4.1.1}$$

and for all simple path p from s to t, $p \neq p_1$,

$$l(p_2) \leq l(p) \tag{4.1.2}$$

Definition 4.2 (Edge-Connecting Simple Cycle Problem)**.** Let $G = (V, E)$ be directed graph and let $e, f \in E, e \neq f$ be given. The **Edge-Connecting Simple Cycle Problem** finds a simple cycle C that passes through e and f, or determine that no such cycle exists.

Remark 4.3. The NP-hardness follow from the hardness of finding two disjoint paths, see Lemma 2.9. It is easy to see that the **Vertex-Connecting Simple Cycle Problem**, which determines a cycle that passes through two given vertices, is also NP-hard.

Theorem 4.4. *The Strictly Second Shortest Simple Path Problem is NP-hard.*

Proof. Given a directed graph $G = (V, E)$ and edges $e_1 = (u_1, v_1), e_2 = (u_2, v_2) \in E$. We define new graph $G' = (V', E')$ with $V' = V$ and $E' = E \cup \{(v_2, u_2)\}$ and a length function

$$l(e) = \begin{cases} 1 & \text{if } e = e_1 \\ 0 & \text{otherwise} \end{cases} \tag{4.1.3}$$

Hence, the shortest path from v_2 to u_2 in G' has weight 0 and the strictly second shortest simple path from v_2 to u_2 includes e_1 and has weight 1. That is, we have found an edge-connecting simple cycle in G: $u_2 \to v_2 \to u_1 \to v_1 \to u_2$. Because the edge-connecting simple cycle problem is NP-hard, the strictly second shortest simple path problem is also NP-hard. □

4.2 Strictly Second Shortest Path is in P ❖

In this section we relax the requirement that such a next-to-shortest path is simple.

Lemma 4.5. *Let $G = (V, E)$ be a weighted directed graph and s, t be two distinct vertices in V. Let p_2 be a strictly second shortest path from s to t and for any edge $e = (u, v) \in p_2$, let p_u be the subpath from s to u and p_v be the subpath from v to t. Then at least one of p_u and p_v is a shortest path.*

Proof. Suppose that there is a path p_2 such that neither p_u nor p_v are shortest paths. Then, there exists p'_u and p'_v with $l(p'_u) < l(p_u)$ and $l(p'_v) < l(p_v)$. Then, we have

$$l(p'_u \circ e \circ p'_v) < l(p'_u \circ e \circ p_v) < l(p_u \circ e \circ p_v) = l(p_2)$$

contradicting that p_2 is a strictly second shortest path. □

The following lemma allows us to find a strictly second shortest path from s to t by enumerating a "non-optimal" edge and find shortest paths connects this edge to s and t, which can be done in polynomial time.

Lemma 4.6. *Let $G = (V, E)$ be a weighted directed graph and s, t be two distinct vertices in V. Let p_2 be a strictly second shortest path from s to t. Then there is an edge $e = (u, v) \in p_2$, with p_u the subpath from s to u and p_v the subpath from v to t, such that p_u and p_v are both shortest paths.*

Proof. Let $e = (u, v) \in p_2$ be an edge such that p_v is a shortest path from v to t but $e \circ p_v$ is not a shortest path from u to t. Such a path must exist because p_2 is not a shortest path. Suppose that p_u is not a shortest path from s to u. Then there exists p'_u with $l(p'_u) < l(p_u)$, which implies

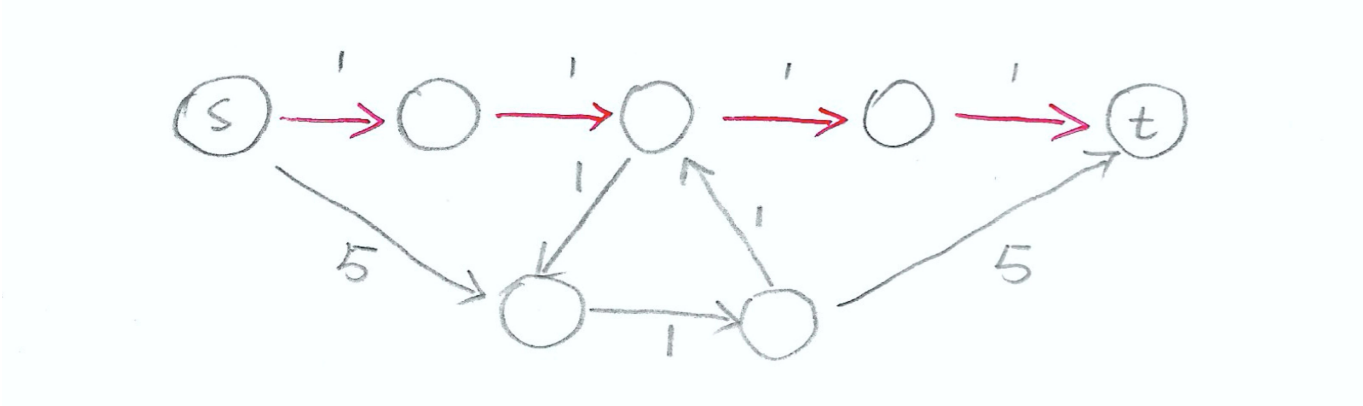
$$l(p'_u \circ e \circ p_v) < l(p_u \circ e \circ p_v)$$

However, because $e \circ p_v$ is not a shortest path, we have that $p'_u \circ e \circ p_v$ cannot be a shortest path from s to t. This contradicts that p_2 is a strictly second shortest path. □

In the following discussion, we restrict our attention to simple graphs with positive weight edges. Given a shortest path DAG $H = (V_H, E_H)$ from s to t in G. That is, each edge $e \in E_H$ is a part of some shortest path π from s to t. It can be implemented using Dijkstra's algorithm ($O(m + n \log n)$) or Bellman-Ford algorithm ($O(mn)$). Suppose that we colour each edge in E_H red.

Observation 4.7. *For any edge $f = (u, v) \in E$, if both $p_1 = \mathcal{O}(s, u)$ and $p_2 = \mathcal{O}(v, t)$ exist and $p^* := p_1 \circ f \circ p_2$ contains a cycle, then f is in the cycle. If f is not in the cycle, then the cycle is a subpath of either p_1 or p_2, which is impossible. By removing the cycle, we obtain a simple path p' induced from p^* by removing the cycle. Because each edge has a positive weight, we have that $w(p') < w(p^*)$.*

Such an observation gives a counterexample that Lemma 4.5 does not apply if p_2 is required to be simple, as shown in the following figure.



Let p be a path from s to t with at least two non-red edges. Let $e_1 = (u_1, v_1), e_2 = (u_2, v_2)$ be the first and last non-red edges on p. We partition p into $p_1 \circ e_1 \circ p_2 \circ e_2 \circ p_3$ where p_1 goes from s to u_1, p_2 goes from v_1 to u_2 and p_3 goes from v_2 to t.

Theorem 4.8. *If p is a strictly second shortest simple path from s to t, then either p_1, p_2, p_3 are shortest paths between their end vertices and or all of the shortest paths from v_1 to u_2 intersects with both p_1 and p_3.*

Proof. Because p_1 and p_3 are both red, we must have p_1 and p_3 are shortest paths between their end vertices. Suppose that p_2 is not a shortest path from v_1 to u_2. Then there is a shortest path p^* from v_1 to u_2. If p^* is disjoint from both p_1 and p_3, then $p_1 \circ e_1 \circ p^* \circ e_2 \circ p_3$ is a shorter path from s to t, which contradicts that p is a strictly second shortest simple path. Suppose $p_1 \circ e_1 \circ p^*$ contains a cycle. Let p' be the induced subpath of $p_1 \circ e_1 \circ p^*$ by removing the cycles, we obtain that $p' \circ e_2 \circ p_3$ is a shorter path from s to t, which contradicts that p is a strictly second shortest simple path. If $p^* \circ e_2 \circ p_3$ contains a cycle, a similar argument applies. □

Question 4.9. If we set v_1 and u_2 as the new source-sink pair, create a new shortest path DAG from v_1 to u_2, colour each edge in the new DAG also to red, we know that if p_2 if not a shortest path, then p_2 must still contain a white edge.

Hence, is it possible to recursively look for the "critical" white edge that lies in some strictly second shortest simple path?

Algorithm 4.10. **Input**: A directed graph $G = (V, E)$, a weight function w, two vertices $s, t \in V$.
Output: A strictly second shortest simple path p from s to t, if p exists.

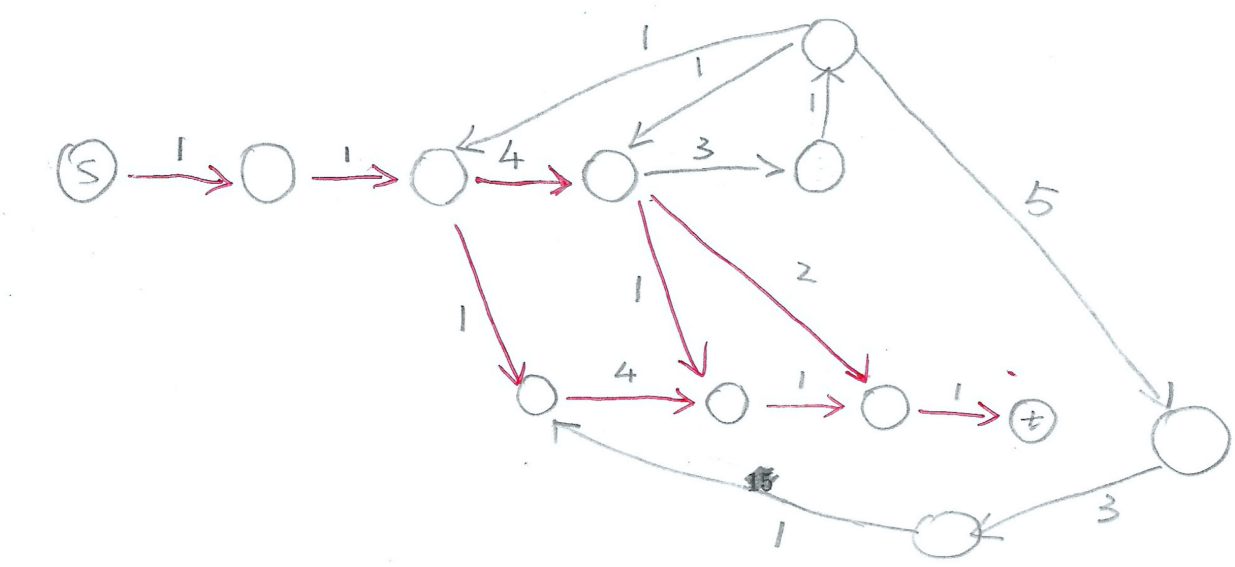
1. Run $H \leftarrow \mathcal{O}(G, s, t)$.
2. Set $E_F \leftarrow \emptyset, E_L \leftarrow \emptyset, p \leftarrow \emptyset, w(p) \leftarrow \infty$.
3. For each edge $e = (u, v) \in (E \setminus E_H)$:

   if $u \in V_H$ and $v \in V_H$:
   i. $(p_1, p_2) \leftarrow \mathcal{P}(G = (V, E \setminus \{e\}), k = 2, (s, u), (v, t))$.
   ii. **if** no `error` occurs and $w(p_1 \circ e \circ p_2) < w(p)$, $p \leftarrow p_1 \circ e \circ p_2$.

   if $u \in V_H$, **then** $E_F = E_F \cup \{e\}$. // The set of all possible "first non-red edges" in a path.

   if $v \in V_H$, **then** $E_L = E_L \cup \{e\}$. // The set of all possible "last non-red edges" in a path.
4. For each pair of edges $e_1 = (u_1, v_1) \in E_F, e_2 = (u_2, v_2) \in E_L, e_1 \neq e_2$:

   (a) $(p_1, p_2, p_3) \leftarrow \mathcal{P}(G = (V, E \setminus \{e_1, e_2\}), k = 3, (s, u_1), (v_1, u_2), (v_2, t))$.
   if no `error` occurs and $w(p_1 \circ e_1 \circ p_2 \circ e_2 \circ p_3) < w(p)$, $p \leftarrow p_1 \circ e_1 \circ p_2 \circ e_2 \circ p_3$;
   else set v_1 and u_2 to the source and the sink and recursively reapply this algorithm.
5. Output p.



Question 4.11. Alternatively, is it true that the number of red-white alternations is not large, say $O(1)$?

5 Undirected Strictly Second Shortest Simple Path Problem is in P

As we have seen in the last section, when 0-weight edges are allowed, the Strictly Second Shortest Simple Path Problem is NP-hard. Hence, we restrict our attention to simple graphs with positive weight edges. In this section we present a somewhat surprising result in [KN04].
Let $G = (V, E)$ be a weighted undirected graph and s, t be two distinct vertices in V.

Lemma 5.1. *Let $w \in V, w \neq s, w \neq t$ be given. There is a polynomial time algorithm to find a shortest path from s to t that passes through w.*

Equivalently, there is a a polynomial time algorithm to find a shortest path from s to t that passes through some edge $e \in E$.

Proof. We first add a vertex r to V and two edges $(s, r), (t, r)$ to E. We create a directed G' by replacing each edge in E by two directed edges with the same end vertices and weight. Give every edge and vertices t and w infinite capacity and any other vertex a capacity of 1. Similar to what we have seen in the proof of Theorem 2.5, finding a shortest path from s to t that passes through w is equivalent to finding the minimum cost flow from w to r where the cost function is equal to the weight function. □

Question 5.2. Is there an analogous lemma for directed graphs?

Notation 5.3. For any vertex $v \in V, v \neq s$, we let $\delta(v)$ to denote the minimum distance between s and v.

Given $u, v \in V, \mathcal{O}$ outputs a shortest path DAG $H = (V_H, E_H)$ from u to v in G if it exists. That is, each edge $e \in E_H$ is a part of some shortest path π from u to v. It can be implemented using Dijkstra's algorithm ($O(m + n \log n)$) or Bellman-Ford algorithm ($O(mn)$).

Lemma 5.4. *Let $H = (V_H, E_H)$ be a shortest path DAG from s to t. Then any path $p \subseteq H$ from s to some vertex $w \in V_H$ is a shortest path from s to w.*

Proof. We prove this lemma by induction. Suppose that every edge in E_H is coloured red and any edge in $E \setminus E_H$ is coloured white.
Suppose q is a simple path from s to t consisting only of red edges. Let $q = e_1 \circ e_2 \circ ... \circ e_k$ be partitioned into a sequence of red edges. We show that for each $v_i :=$head$(e_i), i \in [k]$, $\delta(v_i) := l(e_1 \circ ... \circ e_i)$, which implies that q is a shortest path from s to t.
$i = 1$: Let $v_1 =$head(e_1). If $l(e_1) > \delta(v_1)$, then there is a path q from s to v_1 with $l(q) = \delta(v_1)$. Because e_1 is red, then it is an edge in some shortest path p from s to t. Let p_1 be the subpath of p from v_1 to t. Now, $l(q \circ p_1) < l(p)$, which contradicts that p is a shortest simple path.
$i > 1$: Because head(e_{i-1}) = tail$(e_i) = v_{i-1}$, we have that $\delta(v_{i-1}) = l(e_1 \circ ... \circ e_{i-1})$ by induction hypothesis. Because e_i is red, then it is an edge in some shortest path p from s to t. Hence, applying the same argument, we have that $\delta(v_i) := l(e_1 \circ ... \circ e_i)$. □

Definition 5.5. We say an edge $e = (u, v) \in E$ is a **forward** edge if $(u, v) \in H$ and a **backward** edge if $(v, u) \in H$. Because H is a DAG, an edge cannot both be forward and backward.

Algorithm 5.6. Input: An undirected graph $G = (V, E)$, a weight function w, two vertices $s, t \in V, s \neq t$.
Output: A strictly second shortest simple path p from s to t, if p exists.

1. Run $H \leftarrow \mathcal{O}(G, s, t)$.
2. Set $p \leftarrow \emptyset, l(p) \leftarrow \infty$.
3. **for** each edge $e = (u, v)$ with $(u, v) \notin E_H$ and $(v, u) \notin E_H$:

   Find a shortest path q , that passes through e.

   if q exists and $l(q) < l(p), p \leftarrow q$.
4. **for** each edge pair of edges $e = (u, w)$ with $(u, w) \in E_H$ and $f = (w, v)$ with $(v, w) \in E_H$:

Create G' from G by removing all edges adjacent to w, except for e and f

Find a shortest path q , that passes through w.

if q exists and $l(q) < l(p), p \leftarrow q$.

5. Output p.

Notice that the strictly second shortest simple path either contains some edge which is neither forward nor backward, or consists only of at least 1 backward edge and a number of forward edges. Hence, the correctness of the above algorithm is established.

6 Fine-Grained Complexity for Weighted Directed Sparse Graphs

Lemma 6.1. *In weighted directed graphs,*

$$2\text{-}SiSP \lesssim^{\mathbf{sprs}}_{m+n} Radius \tag{6.0.1}$$

Notation 6.2. Let $d_G(u, v)$ be the distance from u to v in G.

Proof. Let $G = (V, E)$ and a shortest path $P = (s = v_0 \to v_1 \rightsquigarrow v_{l-1} \to v_l = t)$ from s to t be the input.

1. Construct $G' = (V', E')$. Let $V' = V \cup \{z_0, z_1, ..., z_l\}$. To obtain E',
   (a) $E' \leftarrow E$.
   (b) Remove edges in P.
   (c) For each $0 \le i \le l-1$: let edge $e = (z_i, v_i)$ with weight $d_G(s, v_i)$, $f = (v_{i+1}, z_i)$ with weight $d_G(v_{i+1}, t)$ and $g = (z_i, z_{i-1})$ with 0 weight, and then add e, f, g to E'.
2. Construct $G'' = (V'', E'')$ from G'.
   (a) For each $0 \le j \le l-1$:
      i. We replace z_j by vertices z_{j_i} and z_{j_o} and an edge $e = (z_{j_i}, z_{j_o})$ with 0 weight.
      ii. Replace each incoming edge (u, z_j) by (u, z_{j_i}) with the same weight.
      iii. Replace each outgoing edge (z_j, v) by (z_{j_o}, v) with the same weight.
   (b) Let $W_{\max}$ be the largest edge weight in G and $M' = 9nW_{\max}$. For each $0 \le j \le l-1$:
      i. Add vertices y_{j_i}, y_{j_o}.
      ii. Add an edge $e = (y_{j_o}, z_{j_o})$ with weight 0.
      iii. Add an edge $f = (z_{j_i}, y_{j_i})$ with weight $11M'/9$.
   (c) Add vertices A, B and an edge $e = (A, B)$ with weight 0. For each $0 \le j \le l-1$:
      i. Add an edge $f = (y_{j_o}, A)$ with weight 0.
      ii. Add an edge $g = (B, y_{j_o})$ with weight M'.
   (d) For each $0 \le j, k \le l-1, j \ne k$, we could add an edge $e = (y_{j_o}, y_{k_o})$ with weight $2M'/3$. However, this will make G'' dense. Alternatively, we apply **bit encoding**: for each $1 \le r \le \lceil \log n \rceil$ and $s \in \{0, 1\}$:
      i. Create a gadget $C_{r,s}$ consisting $2\lceil \log n \rceil$ vertices.
      ii. If j's r-th bit is equal to s, add an edge from y_{j_o} to $C_{r,s}$ of weight $M'/3$.
      iii. If j's r-th bit is equal not to s, add an edge from $C_{r,s}$ to y_{j_o} of weight $M'/3$.

   This means that, overall, we have added $4\log^2 n$ vertices and at most $2n \log n$ edges.

Claim 6.3. For each $0 \le j \le l-1$, the longest shortest path in G'' from y_{j_o} is to the vertex y_{j_i}.

Proof.
1. Notice that any shortest path from y_{j_o} to any vertex in G or any z's is at most $nW_{\max}$.
2. As (y_{j_o}, A) has weight 0 and (A, B) has weight 0, any shortest path to A or B has weight 0.
3. For each $k \ne j$, the shortest path from y_{j_o} to y_{k_o} has weight M' because the path $y_{j_o} \to A \to B \to y_{k_o}$ has weight M'; the shortest path from y_{j_o} to y_{k_i}, the bit encoding construction gives us a path of weight $2M'/3$.
4. The shortest path from y_{j_o} is to y_{j_i} must contain an edge (z_{j_i}, y_{j_i}), which has weight $11M'/9$.

■

Claim 6.4. The shortest path from z_{j_o} to z_{j_i} corresponds to the replacement path for the edge (v_j, v_{j+1}) lying on P.

Proof. Suppose not and let $P_j(s \rightsquigarrow v_h \rightsquigarrow v_k \rightsquigarrow t)$ denote the replacement path from s to t for the edge (v_j, v_{j+1}). Notice the path $z_{j_o} \to z_{j-1_i} \rightsquigarrow z_{h_o} \to v_h \circ P_j(v_h \rightsquigarrow v_k) \circ v_k \to z_{k_i} \to z_{k_o} \rightsquigarrow z_{j_i}$ has weight equal to that of P_j by construction. ■

Claim 6.5. One of the vertices among y_{j_o}'s is a centre of G''.

Proof. 1. Because each vertex in G or each z's cannot reach y_{j_o} for any j, it cannot be a centre.

2. As (B, y_{j_o}) has weight M' and the path from y_{j_o} to y_{k_i} has weight $2M'/3$, any shortest path from A or B to any y_{k_i} has weight $5M'/3$, which is greater than $11M'/9$.

■

□

7 Finding Replacement Paths in Unweighted Directed Graphs in Expected $\tilde{O}(m\sqrt{n})$ Time

The **replacement paths** problem, when given a unweighted directed graph $G = (V, E)$ and a shortest path P from s to t, seeks, for each edge $e \in P$, a shortest path from s to t that avoids e. In this section, we introduce the notion of **detour** that will help construct a randomised algorithm that outputs replacement paths in expected $\tilde{O}(m\sqrt{n})$ time.

Definition 7.1. Let $P(s,t)$ be a simple path from s to t. A simple path $D(u,v)$ is a **detour** of $P(s,t)$ if $D(u,v) \cap P(s,t) = \{u, v\}$ and u precedes v on $P(s,t)$.

Let $L = O(n)$ be a given parameter. We say that a detour is *short* if its length is at most L and that it is *long* otherwise. In the following two subsections, we will show that we can find short detours in $\tilde{O}(mL)$ time while finding suitable long detours in $\tilde{O}(mn/L)$ time. By setting $L = O(\sqrt{n})$, we get the desired time bound.

7.1 Finding Short Detours ❖

For a shortest path $P = (V_P, E_P) = (u_0 \to u_1 \to ... \to u_l)$, we construct an *auxiliary graph* G^A from G as the following:

1. Let $k = \lfloor l/2L \rfloor$.
2. Take $G^A \leftarrow G \setminus E_P$. Set the weight of every edge to 1.
3. Add a vertex r and edges (r, u_{2iL}) for $0 \le i \le k$, each with weight iL to G^A.

We introduce some notations.

Notation 7.2. For two vertices $u, v \in V$, we let

1. $\delta(u,v) :=$ distance from u to v in G.
2. $\delta^-(u,v) :=$ distance from u to v in $G \setminus E_P$.
3. $\delta^A(u,v) :=$ distance from u to v in G^A.

The idea is that, for each iteration $0 \le i \le 2L-1$, by running Dijkstra's algorithm, from r, on G^A, we find all the best short detours that start in each one of the vertices $u_i, u_{2L+i}, ..., u_{2kL+i}$. Hence, for an edge (u_i, u_{i+1}), it suffices to find indices $i - L \le a \le i$ and $i < b < i + L$ and concatenate a detour between a and b with parts of P. Thus, we can return a best short replacement path in $\tilde{O}(mL)$ time. The following theorem tells us that small distance between r and some vertex u_{2iL+j} for some $1 \le j \le L$ implies a short detour containing u_{2iL} and u_{2iL+j}. Notice that it suffices to show the case for $u_0, u_{2L}, ..., u_{2kL}$.

Theorem 7.3. *If*

$$\delta^A(r, u_{2iL+j}) \le (i+1)L,$$

where $0 \le i \le k$ *and* $1 \le j \le L$, *then*

$$\delta^-(u_{2iL}, u_{2iL+j}) = \delta^A(r, u_{2iL+j}) - iL.$$

Otherwise, we must have

$$\delta^-(u_{2iL}, u_{2iL+j}) > L.$$

Proof. For simplicity we let $v_i = u_{2iL}, v_{ij} = u_{2iL+j}$. Suppose $\delta^A(r, v_{ij}) \leq (i+1)L$. Consider a shortest path from r to $v_{i,j}$ in G^A and let (r, v_q) be the first edge on the path. If $q < i$, then

$$\delta^A(r, v_{ij}) = qL + \delta^-(v_q, v_{i,j}) \geq qL + 2(i-q)L + j \geq (i+1)L + j > (i+1)L \tag{7.1.1}$$

If $q \geq i$, then

$$\delta^A(r, v_{ij}) = qL + \delta^-(v_q, v_{i,j}) > (i+1)L \tag{7.1.2}$$

On the other hand, if $\delta^-(v_i, v_{ij}) \leq L$, then

$$\delta^A(r, v_{ij}) = w(r, v_i) + \delta^-(v_i, v_{ij}) \leq (i+1)L \tag{7.1.3}$$

where $w(r, v_i)$ is the weight of the edge (r, v_i). □

7.2 Finding Long Detours ❖

By following simple sampling lemma, we have that for each pair of vertices u and v on the path P for which the shortest detour $D(u, v)$ from u to v is of length at least L, $D(u, v) \cap R \neq \emptyset$ with probability at least $1 - n^{2-c}$.

Lemma 7.4 (Sampling Lemma)**.** *Let q be an integer and V be the vertex set of G. Let $D_1, D_2, ..., D_q \subseteq V$ be vertex sets, each of size at least L. If $R \subseteq V$ is a random subset obtained by selecting each vertex, independently, with probability $(c \ln n/L)$ for some constant c, then with probability $\geq 1 - q \cdot n^{-c}$, we have $D_i \cap R \neq \emptyset$, for all $1 \leq i \leq q$.*

We now describe the intuition for the randomised algorithm finding long detours.

1. Sample a random set R where each vertex $v \in V$ is chosen with probability $(4 \ln n/L)$. Then
$$\mathbb{E}[|R|] = \tilde{O}(n/L).$$
2. For each $r \in R$ and $v \in V$, compute $\delta^-(r, v)$ and $\delta^-(v, r)$. This step can be done in expected $\tilde{O}(mn/L)$ time using BFS.
3. Recall that $D(u, v) \cap R \neq \emptyset$ with high probability with each pair of u and v. For the edge (u_i, u_{i+1}) that passes through some $r \in R$, it suffices to find indices $0 \leq a \leq i$ and $i < b \leq l$ that minimises $a + \delta^-(u_a, r)$ and $\delta^-(r, u_b) + (l - b)$. These can be maintained in two priority queues.

7.3 Finding k Shortest Simple Paths ❖

The improvement from Yen's algorithm $\tilde{O}(kn(m+n))$ to $\tilde{O}(km\sqrt{n})$ is very intuitive. Simply supplanting the replacement path finding part in Yen's algorithm by the improved randomised result we have just shown, the desired time bound is obtained.

References

[BK17] Kristóf Bérczi and Yusuke Kobayashi. The directed disjoint shortest paths problem. In *25th Annual European Symposium on Algorithms (ESA 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[EIS75] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of time table and multi-commodity flow problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 184–193. IEEE, 1975.

[FHW80] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111 – 121, 1980.

[KN04] Ilia Krasikov and Steven D Noble. Finding next-to-shortest paths in a graph. *Information processing letters*, 92(3):117–119, 2004.

[LP97] Kumar N Lalgudi and Marios C Papaefthymiou. Computing strictly-second shortest paths. *Information processing letters*, 63(4):177–181, 1997.

[LR80] Andrea S LaPaugh and Ronald L Rivest. The subgraph homeomorphism problem. *Journal of Computer and System Sciences*, 20(2):133–149, 1980.

[Sch94] Alexander Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM Journal on Computing*, 23(4):780–788, 1994.

[Shi78] Yossi Shiloach. *The two paths problem is polynomial.* Stanford University. Computer Science Department, 1978.

[Yen71] Jin Y Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.