Find-Reassemble-Path Algorithm for Finding Node Disjoint Paths in Telecommunication Networks with Two Technologies

Student: Chia-Ching Li

Advisor: Kwang-Fu Li

Jun, 2001

Abstract

Consider a network with two different types of arcs, each type stands for one transmission technology. Each arc in the graph is associated with a transmission cost and each node in the graph may be associated with a transition cost. Transition cost is concerned with the technologies and occurs whenever a message enters node i on an arc (j, i) belonging to one type of transmission technologies and leaves node i on an arc (i, k) belonging to the other type of transmission technologies. The problem we are focusing on is finding two node disjoint paths with minimum cost in the network. In 1995, Jongh, Gendreau, and Labbe showed that it is strong NP-complete and then provided heuristic solutions for the problem. In this paper, we provide a new algorithm, called Find-Reassemble-Path algorithm, to find two node disjoint paths with lower cost.

1 Introduction

Due to the rapid development of communication protocols, it is common to encounter that two different transmission technologies coexist in a network. In these networks, whenever a message is transmitted from sources to destinations, the message often needs to be routed over links of different transmission technologies. It may encounter some costs when the message passes through links of different transmission technologies. Hence, not only transmission costs but also transition costs should be taken into consideration. Transition costs occur only when messages pass through a link of one technology to a link of the other technology.

Reliability is an important issue in communications. We can provide more than one routes in networks from sources to sinks to strengthen the reliability. In fact, when two arc disjoint paths are provided, the messages transmitted from the source to the sink won't be lost if a single link of the network fails. Similarly, when two node disjoint paths are provided, the messages won't be lost if a single node of the network runs out of order.

Two arc disjoint paths with minimum cost can be found after we transform each node i into two nodes i_1 and i_2, as shown in Figure 1, and add two arcs (i_1, i_2) and (i_2, i_1) with costs equal to the transition costs [1]. After the transformation, we can easily find the optimal two arc disjoint paths by the procedure provided by Suurballe and Tarjan [6]. In this paper, we emphasize on finding two node disjoint paths from the source to the sink with minimum cost in a network with two types of transmission technologies. Jongh, Gendreau and Labbe proved that it is NP-Hard to find two node disjoint paths with minimum cost from the source to the sink in a network with two types of transmission technologies [1]. They also provided polynomial time heuristics for finding two node disjoint paths from the source to the sink. They used Dijkstra's algorithm [3][5] to find the first shortest path and delete all nodes on the path as well as arcs incident to the nodes. And they used Dijkstra's algorithm again to find the second shortest path if the source and the sink are connected in the remaining graph. Although the algorithm can be easily executed, the source and the sink may become disconnected if some nodes are removed. Even if the remaining graph is connected, the removal of the first shortest path may also affect the selections of better pair of node disjoint paths. If the remaining graph is not connected, they ignore transition costs and find a pair of node disjoint paths in the network and add transition costs when necessary.

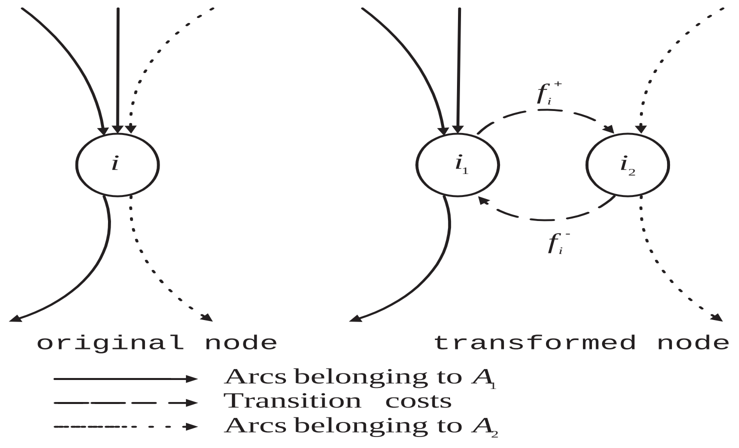


Figure 1: Transition of node i.

In this paper, we present a new algorithm, Find-Reassemble-Path algorithm, for finding two node disjoint paths with lower cost than the original heuristics if the remaining graph is connected. We will reverse the directions of arcs on the first shortest path, instead of deleting all nodes on the path and the arcs incident to the nodes, then find the second path on the reversed graph. Finally, we reassemble these two paths to two node disjoint paths.

This paper is consisted of five sections. In section 1, we briefly describe some methods of finding disjoint paths. In section 2, we describe the methods of finding node disjoint paths provided by Jongh, Gendreau, and Labbe. In section 3, Find-Reassemble-Path algorithm is proposed and discussed. In section 4, we show that Find-Reassemble-Path algorithm can find a pair of node disjoint paths with lower cost if the remaining graph is connected. We will also analyze the complexity of Find-Reassemble-Path algorithm. Finally, in section 5, some conclusion remarks are discussed.

2 Backgrounds

Dijkstra's algorithm is a famous algorithm for finding the shortest path. The Dijkstra's algorithm is a labeling process that records a tentative distance $d(v)$ and a tentative predecessor $p(v)$ for each node $v \in G$, where G is a network with one technology. Let s be the source and t be the sink in G. Each arc $(i,j) \in G$ has a cost c_{ij}. Initially, $d(s) = 0$ and $d(v) = \infty$ if $v \neq s$, each node is unlabeled and $p(v)$ is undefined for every node v. The Dijkstra's algorithm repeats the following step until there is no unlabeled node.

Labeling Step:
Choose an unlabeled vertex i with minimum distance $d(i)$. Make i labeled.
For each arc $(i,j) \in G$, if $d(i) + c_{ij} < d(j)$, change $d(j)$ to $d(i) + c_{ij}$ and $p(j)$ to i.

When the algorithm terminates, $d(i)$ is the distance between node s and node i for every node i reachable from s. And the set of arcs $\{(p(i), i)|\ i \neq s \text{ with } d(i) \text{ finite}\}$ defines a shortest path directed tree rooted from s.

Consider a directed graph $G = (V, A)$ with two transmission technologies, where V denotes the vertex set and A denotes the arc set. A is partitioned into two disjoint subsets A_1 and A_2. Each one represents one transmission technology. Each arc $(i,j) \in G$ is associated with a cost c_{ij}, called the transmission cost. There may be two more positive costs f_i^+ and f_i^- (called the transition costs) with respect to each node $i \in G$. The first cost f_i^+ occurs when a chosen path enters i on an arc $(j,i) \in A_1$ and leaves i on an arc $(i,k) \in A_2$. Similarly, the second cost f_i^- occurs when a chosen path enters i on an arc $(j,i) \in A_2$ and leaves i on an arc $(i,k) \in A_1$. The problem we are interested is to find two node disjoint paths from the source to the destination with minimum cost. This is a NP-Complete problem [1]. Heuristic methods to solve this problem were provided by Jongh, Gendreau, Labbe. We call the method JGL algorithm for convenience. JGL algorithm is briefly described as follows. First, they find the first path and deleting nodes on the paths as well as arcs incident to nodes on this path. If the remaining graph is connected, they proceed to find the second path on the graph. If the remaining graph is not connected, they transformed each node, except the source and the sink, to two nodes as shown in Figure

2. In the transformed network, arc disjoint paths are also node disjoint. They used the procedure provided by Suurballe and Tarjan to find two arc disjoint path in the transformed graph [6], then add transition cost to the solution when necessary.

Heuristic 1

Step 1. Transform each node of the network, except the source and the sink , as shown in Figure 2.

Step 2. Find two minimum cost arc disjoint paths in this transformed network.

Step 3. Add the transition costs to the solution when necessary, i.e., each time the flow passes from one technology to another in the solution of **Step 2**.

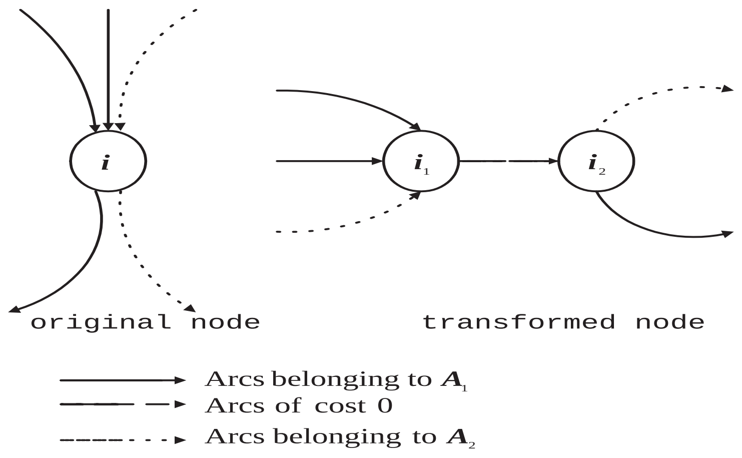


Figure 2: Transformation of node i if the remaining graph is not connected.

Heuristic 2

Step 1. Transform each node of the network, except the source and the sink, as shown in Figure 1.

Step 2. Determine a first minimum cost path from the source to the sink. (Ties are broken in favor of the path with the smallest number of arcs and transitions. If ties still subsist, a single technology path is chosen).

Step 3. Delete all of the intermediate nodes of this first path from the graph, as well as the arcs incident to these nodes.

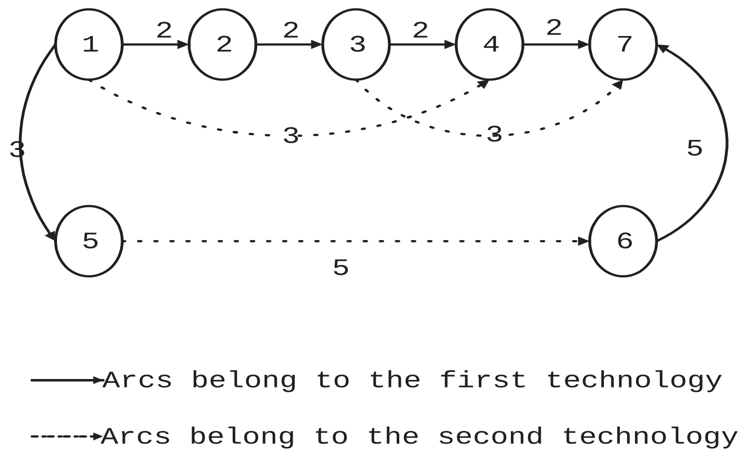
Step 4. If the remaining graph is connected (i.e., if there is at least one path from the source to the sink), determine as second path a minimum cost path. (Ties are broken in favor of the path with the smallest number of arcs and transitions. If ties still subsist, a single technology path is chosen).

Heuristic 3

Step 1. Execute Steps 1 to 3 of Heuristic 2.

Step 2. If the remaining graph is not connected then go to Step 3, else determine as second path a minimum cost path. Stop.

Step 3. Execute Heuristic 1.



1
2
3
4
7
5
6
2
2
2
2
3
3
3
5
5
Arcs belong to the first technology
Arcs belong to the second technology

Figure 3: An example that the remaining graph is connected.

Consider the network in Figure 3, f_i^+ is 3 and f_i^- is 4 at each node i.

Transform the graph in Figure 3 into a new graph as shown in Figure 3a. Apply Dijkstra's algorithm to find the first shortest path $P_1 = 1 \rightarrow 2_1 \rightarrow 3_1 \rightarrow 4_1 \rightarrow 7$.

Then remove all nodes on P_1 and all arcs incident to P_1. The remaining graph is shown in Figure 3b. Proceed to find $P_2 = 1 \rightarrow 5_1 \rightarrow 5_2 \rightarrow 6_2 \rightarrow 6_1 \rightarrow 7$. Thus we find a pair of node disjoint paths, 1→2→3→4→7 and 1→5→6→7, with cost 28. But 1→2→3→7 and 1→4→7 are better solution in the graph.

When JGL algorithm is executed, all nodes on the first path P_1 and arcs incident to these nodes are removed. Therefore, if there exists two node disjoint paths that both have common nodes with P_1 and these two paths have lower cost, the above heuristics are impossible to find this pair of node disjoint paths. In other words, JGL algorithm finds a pair of node disjoint paths still with higher cost.

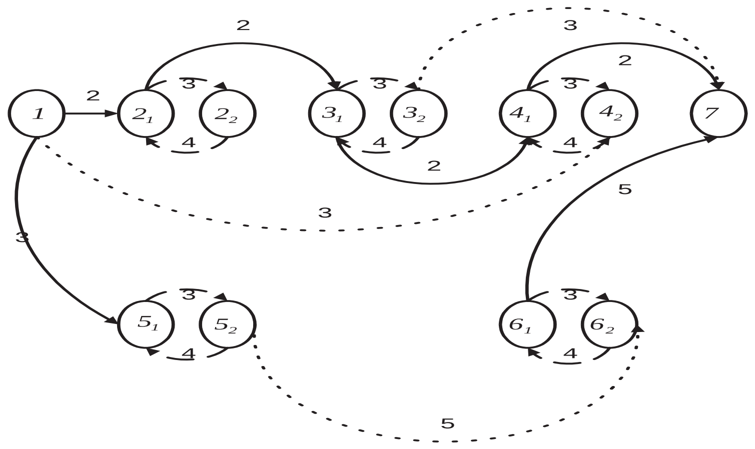


Figure 3a: The transformed graph.

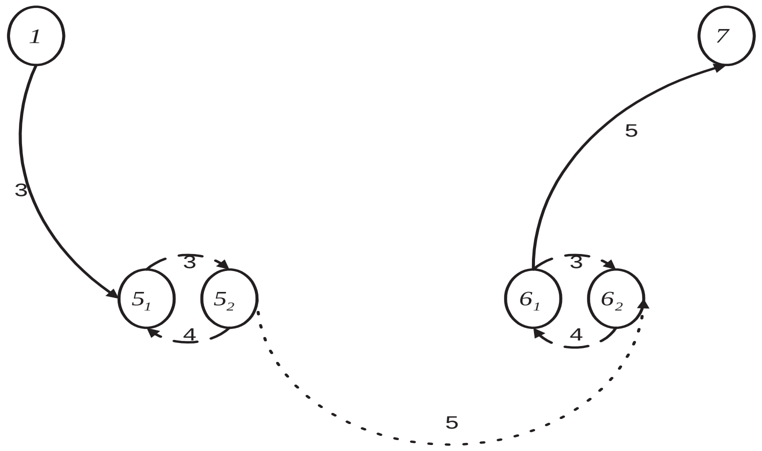


Figure 3b: The remaining graph.

When removing the nodes on P_1 and arcs incident to the nodes , the graph may become disconnected. In this situation, each node except the source and the sink is transformed to two nodes as shown in Figure 2. They ignore the transition costs and find two arc disjoint path in the transformed graph. At last, they add transition costs when necessary. But the addition of the transition costs may add some expensive costs, i.e., we may find a poor solution.

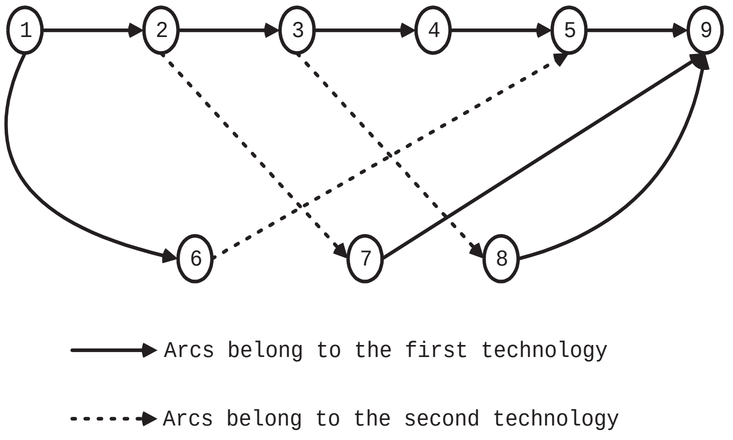

1
2
3
4
5
9
6
7
8
Arcs belong to the first technology
Arcs belong to the second technology

Figure 4: An example that the remaining graph is not connected.

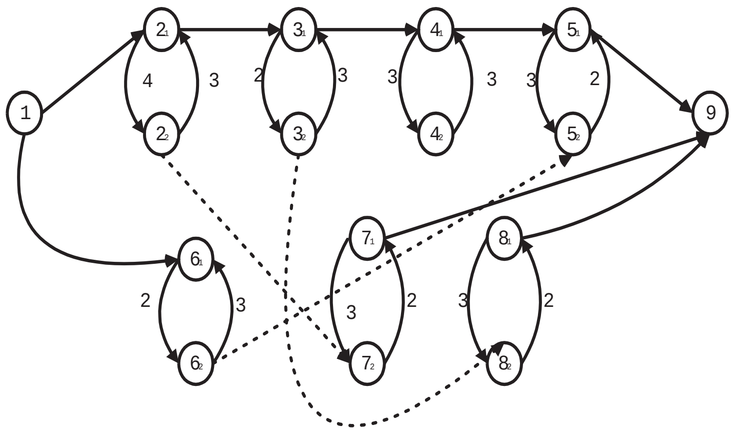


Figure 4a: The transformed graph.

In the following network (see Figure 4), each arc has cost 1 and $f_2^+ = 4$, $f_3^+ = f_5^- = f_6^+ = f_7^- = f_8^- = 2$. First, transform the graph in Figure 4 into the graph in Figure 4a. and

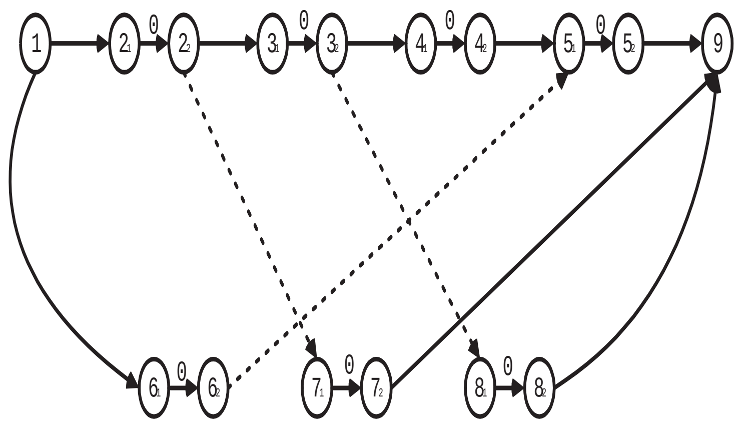


Figure 4b: Transformation of nodes when the remaining graph is not connected.

we'll find the first path $P_1 = 1 \to 2_1 \to 3_1 \to 4_1 \to 7$. After deleting all nodes on P_1 and all arcs incident to nodes on P_1, the remaining graph is disconnected. Thus, we return to the original graph, transform each node of the graph as shown in Figure 4b. Use the procedure provided by Suurballe and Tarjan [6], we can find a pair of arc disjoint paths with minimum cost in this graph, they are $1 \to 2_1 \to 2_2 \to 7_1 \to 7_2 \to 9$ and $1 \to 6_1 \to 6_2 \to 5_1 \to 5_2 \to 9$. In fact, two arc disjoint paths in Figure 4b are also node disjoint. Hence, two node disjoint paths found are $1 \to 2 \to 7 \to 9$ and $1 \to 6 \to 5 \to 9$ and the total cost is 16. But the paths $1 \to 2 \to 3 \to 8 \to 9$ and $1 \to 6 \to 5 \to 9$ are better solutions.

In the following sections, we will provide a new algorithm to find two node disjoint paths. The major difference is that we reverse the directions of all arcs on P_1 instead of removing the nodes and arcs incident to nodes on P_1. We will prove that Find-Reassemble-Path algorithm finds a pair of node disjoint paths with lower cost than the one using JGL algorithm if the remaining graph is connected.

3 Find-Reassemble-Path Algorithm for Finding Two Node Disjoint Paths

In this section, we provide a new algorithm, called Find-Reassemble-Path algorithm, for finding two node disjoint paths. There are four major steps in this algorithm. The first two steps are called the Find-Path and the last two steps are called the Reassemble-Path.

Find-Path

Step 1

(**1.1**) Transform each node of the network, except the source and the sink, as shown in Figure 1.

(**1.2**) Use Dijkstra's algorithm to determine the first minimum cost path, called P_1, from the source to the sink.

(**1.3**) Reverse all directions of arcs on P_1.

Let G_r be the graph obtained from G by reversing the arcs on P_1 is reversed. In the network, s is the source and t is the sink.

We define some symbols as follows.

c_i^1: the cost of the shortest path among all paths from the source s to the node i with the last arc belongs to the subset A_1.

c_i^2: the cost of the shortest path among all paths from the source s to the node i with the last arc belongs to the subset A_2.

p_i^1: the tentative predecessor of node i in the shortest path among all paths from the source s to the node i with the last arc belongs to the subset A_1.

p_i^2: the tentative predecessor of node i in the shortest path among all paths from the source s to the node i with the last arc belongs to the subset A_2.

g_i^1: the tentative grandfather of node i in the shortest path among all paths from the source s to the node i with the last arc belongs to the subset A_1.

g_i^2: the tentative grandfather of node i in the shortest path among all paths from the source s to the node i with the last arc belongs to the subset A_2.

Initially, for each node $i \neq s$, $c_i^1 = \infty$, $c_i^2 = \infty$, p_i^1, p_i^2, g_i^1, and g_i^2 are undefined. For each node, these data are kept and may further be recalculated. We also define $C = \{c_j^1 | j \in V \backslash \{s\}\} \cup \{c_j^2 | j \in V \backslash \{s\}\}$, where V is the vertex set.

In Step 2, we'll register c_i^1, c_i^2, p_i^1, p_i^2, g_i^1, and g_i^2 in each node i, called labeling. Whether a node can be labeled by i depends on the location of i, and the parents of i. There are four cases: (i) $i \notin P_1$, (ii) $i \in P_1$ and $p_i \notin P_1$, (iii) $i \in P_1$ and $p_i = s$, (iv) $i \in P_1$ and $p_i \in P_1$ $(p_i \neq s)$. In the first case, i can label each node $(i, j) \in G_r$. In the second case and in the third case, i can only label the vertex k in front of i on the first path P_1. In the fourth case, i can label (1) the vertex k just in front of i on the first path P_1, (2) each vertex $k \notin P_1$ that $(i, k) \in G_r$, (3) the sink t if $(i, t) \in G_r$.

Step 2

```
(2.1) Label each node $i$ that $(s, i) \in G_r$,
    if $(s, i) \in A_1$ then $c_i^1 = c_{si}$, $p_i^1 = s$,
    if $(s, i) \in A_2$ then $c_i^2 = c_{si}$, $p_i^2 = s$.
(2.2) Find a node $i$ with minimum cost in $C$, and delete this cost from $C$.
    If $i = t$, Stop.
     (2.2.1) $case1$ : The minimum cost is $c_i^1$,
      $(i)$ If $i \notin P_1$ : Label each vertex $j$ that $(i, j) \in G_r$,
        1° If $(i, j) \in A_1$,
          if $c_j^1 > c_i^1 + c_{ij}$, then $c_j^1 = c_i^1 + c_{ij}$, $p_j^1 = i$, $g_j^1 = p_i^1$.
        2° If $(i, j) \in A_2$,
          if $c_j^2 > \min\{c_i^1 + f_i^+ + c_{ij}, c_i^2 + c_{ij}\}$, then $c_j^2 = \min\{c_i^1 + f_i^+ + c_{ij}, c_i^2 + c_{ij}\}$, $p_j^2 = i$,
                  if $c_i^1 + f_i^+ + c_{ij} < c_i^2 + c_{ij}$, then $g_j^2 = p_i^1$,
                  if $c_i^1 + f_i^+ + c_{ij} \geq c_i^2 + c_{ij}$, then $g_j^2 = p_i^2$.
      $(ii)$ If $i \in P_1$, $p_i^1 \notin P_1$, only label the vertex $k$
          which is just in front of $i$ on $P_1$ $(k \neq s)$.
        1° If $(i, k) \in A_1$,
```

if $c_k^1 > c_i^1 + c_{ik}$, then $c_k^1 = c_i^1 + c_{ik}$, $p_k^1 = i$, $g_k^1 = p_i^1$.

2° If $(i,k) \in A_2$,

if $c_k^2 > c_i^1 + f_i^+ + c_{ik}$, then $c_k^2 = c_i^1 + f_i^+ + c_{ik}$, $p_k^2 = i$, $g_k^2 = p_i^1$.

(iii) If $i \in P_1, p_i^1 = s$, only label the vertex k

which is just in front of i on P_1 $(k \neq s)$.

1° If $(i,k) \in A_1$,

if $c_k^1 > c_i^1 + c_{ik}$, then $c_k^1 = c_i^1 + c_{ik}$, $p_k^1 = i$, $g_k^1 = p_i^1$.

2° If $(i,k) \in A_2$,

if $c_k^2 > c_i^1 + f_i^+ + c_{ik}$, then $c_k^2 = c_i^1 + f_i^+ + c_{ik}$, $p_k^2 = i$, $g_k^2 = p_i^1$.

(iv) If $i \in P_{1,}$ $p_i^1 \in P_1$ $(p_i^1 \neq s)$,

(1) label the vertex k which is just in front of i on P_1,

1° If $(i,k) \in A_1$,

if $c_k^1 > c_i^1 + c_{ik}$, then $c_k^1 = c_i^1 + c_{ik}$, $p_k^1 = i$, $g_k^1 = p_i^1$.

2° If $(i,k) \in A_2$,

if $c_k^2 > c_i^1 + f_i^+ + c_{ik}$, then $c_k^2 = c_i^1 + f_i^+ + c_{ik}$, $p_k^2 = i$, $g_k^2 = p_i^1$,

(2) label each vertex $k \notin P_1$ that $(i,k) \in G_r$,

1° If $(i,k) \in A_1$,

if $c_k^1 > c_i^1 + c_{ik}$, then $c_k^1 = c_i^1 + c_{ik}$, $p_k^1 = i$, $g_k^1 = p_i^1$.

2° If $(i,k) \in A_2$,

if $c_k^2 > c_i^1 + f_i^+ + c_{ik}$, then $c_k^2 = c_i^1 + f_i^+ + c_{ik}$, $p_k^2 = i, g_k^2 = p_i^1$.

(3) label the sink t if $(i,t) \in G_r$,

1° If $(i,t) \in A_1$,

if $c_t^1 > c_i^1 + c_{it}$, then $c_t^1 = c_i^1 + c_{it}$, $p_t^1 = i$, $g_t^1 = p_i^1$.

2° If $(i,t) \in A_2$,

if $c_t^2 > c_i^1 + f_i^+ + c_{it}$, then $c_t^2 = c_i^1 + f_i^+ + c_{it}$, $p_t^2 = i$, $g_t^2 = p_i^1$.

$(\mathbf{2.2.2})$ $case2$: The minimum cost is c_i^2,

(i) If $i \notin P_1$: Label each vertex j that $j \in G_r$.

1° If $(i,j) \in A_1$,

if $c_j^1 > \min\{c_i^2 + f_i^- + c_{ij}, c_i^1 + c_{ij}\}$, then $c_j^1 = \min\{c_i^2 + f_i^- + c_{ij}, c_i^1 + c_{ij}\}$, $p_j^1 = i$,

if $c_i^2 + f_i^- + c_{ij} < c_i^1 + c_{ij}$, then $g_j^1 = p_i^2$,

if $c_i^2 + f_i^- + c_{ij} \geq c_i^1 + c_{ij}$, then $g_j^1 = p_i^1$.

2° If $(i,j) \in A_2$,

if $c_j^2 > c_i^2 + c_{ij}$, then $c_j^2 = c_i^2 + c_{ij}$, $p_j^2 = i$, $g_j^2 = p_i^2$.

(ii) If $i \in P_1$, $p_i^2 \notin P_1$, only label the vertex k

which is just in front of i on P_1 $(k \neq s)$.

1° If $(i,k) \in A_1$,

if $c_k^1 > c_i^2 + f_i^- + c_{ik}$, then $c_k^1 = c_i^2 + f_i^- + c_{ik}$, $p_k^1 = i$, $g_k^1 = p_i^2$.

2° If $(i,k) \in A_2$,

if $c_k^2 > c_i^2 + c_{ik}$, then $c_k^2 = c_i^2 + c_{ik}$, $p_k^2 = i$, $g_k^2 = p_i^2$.

(iii) If $i \in P_1, p_i^2 = s$, only label the vertex k

which is just in front of i on P_1 $(k \neq s)$.

1° If $(i,k) \in A_1$,

if $c_k^1 > c_i^2 + f_i^- + c_{ik}$, $c_k^1 = c_i^2 + f_i^- + c_{ik}$, $p_k^1 = i$, $g_k^1 = p_i^2$.

2° If $(i,k) \in A_2$,

if $c_k^2 > c_i^2 + c_{ik}$, $c_k^2 = c_i^2 + c_{ik}$, $p_k^2 = i$, $g_k^2 = p_i^2$.

(iv) If $i \in P_{1,}$ $p_i^2 \in P_1$ $(p_i^2 \neq s)$,

(1) label the vertex k which is just in front of i on P_1,

1° If $(i,k) \in A_1$,

if $c_k^1 > c_i^2 + f_i^- + c_{ik}$, then $c_k^1 = c_i^2 + f_i^- + c_{ik}$, $p_k^1 = i$, $g_k^1 = p_i^2$.

2° If $(i,k) \in A_2$,

if $c_k^2 > c_i^2 + c_{ik}$, then $c_k^2 = c_i^2 + c_{ik}$, $p_k^2 = i$, $g_k^2 = p_i^2$.

(2) label each vertex $k \notin P_1$ that $(i,k) \in G_r$,

1° If $(i,k) \in A_1$,

if $c_k^1 > c_i^2 + f_i^- + c_{ik}$, then $c_k^1 = c_i^2 + f_i^- + c_{ik}$, $p_k^1 = i$, $g_k^1 = p_i^2$.

2° If $(i,k) \in A_2$,

if $c_k^2 > c_i^2 + c_{ik}$, then $c_k^2 = c_i^2 + c_{ik}$, $p_k^2 = i$, $g_k^2 = p_i^2$.

(3) label the sink t if $(i,t) \in G_r$,

1° If $(i,t) \in A_1$,

if $c_t^1 > c_i^2 + f_i^- + c_{it}$, then $c_t^1 = c_i^2 + f_i^- + c_{it}$, $p_t^1 = i$, $g_t^1 = p_i^2$.

2° If $(i,t) \in A_2$,

if $c_t^2 > c_i^2 + c_{it}$, then $c_t^1 = c_i^2 + c_{it}, p_t^1 = i, g_t^1 = p_i^2$.

Example 3.1. Consider the network as shown in Figure 5, for every node i in the graph, $f_i^+ = 3$ and $f_i^- = 4$.

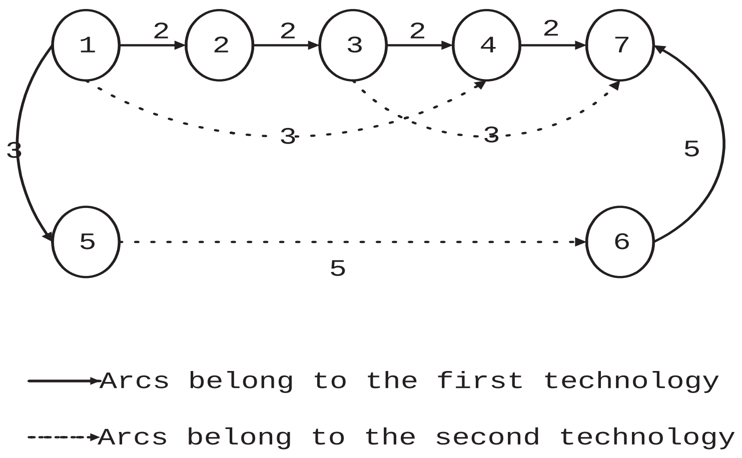

1
2
3
4
7
5
6
2
2
2
2
3
3
3
5
5
Arcs belong to the first technology
Arcs belong to the second technology

Figure 5: An example executing Find-Reassemble-Path algorithm

When applying Find-Path, the first shortest path is $P_1 = 1 \to 2 \to 3 \to 4 \to 7$, then reverse the directions of arcs on P_1 as shown in Figure 5a. If we continue to find the second path, we will find $P_2 = 1 \to 4 \to 3 \to 7$. In this example, we can easily reassemble P_1 and P_2

into two paths $1 \to 2 \to 3 \to 7$ and $1 \to 4 \to 7$. Thus node disjoint paths with cost 19 is found. At times it is not so obvious to reassemble P_1 and P_2 into two node disjoint paths, we need a general method, as the Reassemble-Path, to reassemble node disjoint paths.

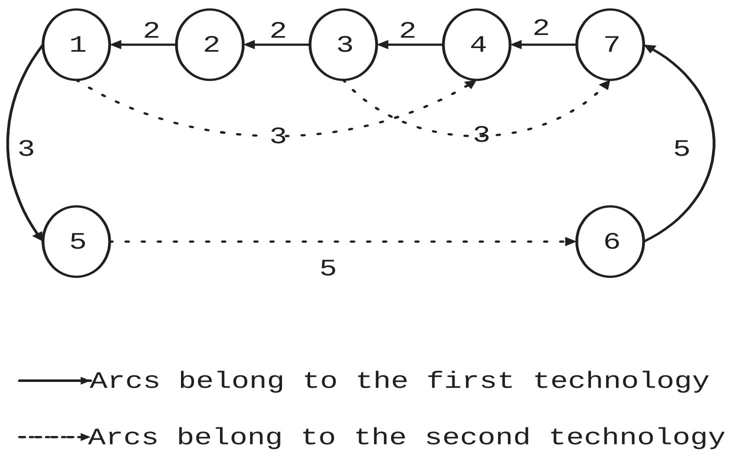

1
2
3
4
7
5
6
2
2
2
2
3
3
3
5
5
Arcs belong to the first technology
Arcs belong to the second technology

Figure 5a: The graph that directions of arcs on the first path is reversed.

Example 3.2. Consider the graph as shown in Figure 6, each arc of the graph has cost 1, and $f_2^+ = 4$, $f_2^- = 3$, $f_3^+ = 2$, $f_3^- = 3$, $f_4^+ = 3$, $f_4^- = 3$, $f_5^+ = 3$, $f_5^- = 2$, $f_6^+ = 2$, $f_6^- = 3$, $f_7^+ = 3$, $f_7^- = 2$, $f_8^+ = 3$, $f_8^- = 2$.

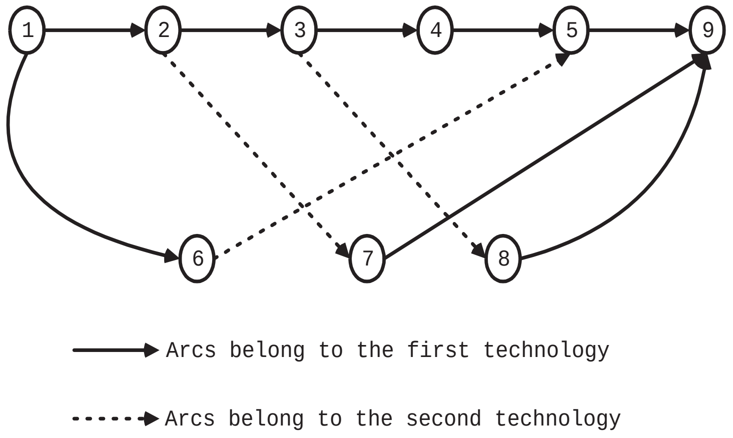

1
2
3
4
5
9
6
7
8
Arcs belong to the first technology
Arcs belong to the second technology

Figure 6: Another example executing Find-Reassemble-Path algorithm.

Applying the Find-Path, the first path P_1 =1→2→3→4→5→9. Reverse the directions of arcs on P_1 as shown in Figure 6a. We can find the second path P_2 =1→6→5→4→3→8→9.

P_1 and P_2 can be reassembles to two paths 1→6→5→9 and 1→2→3→8→9 with cost 15.

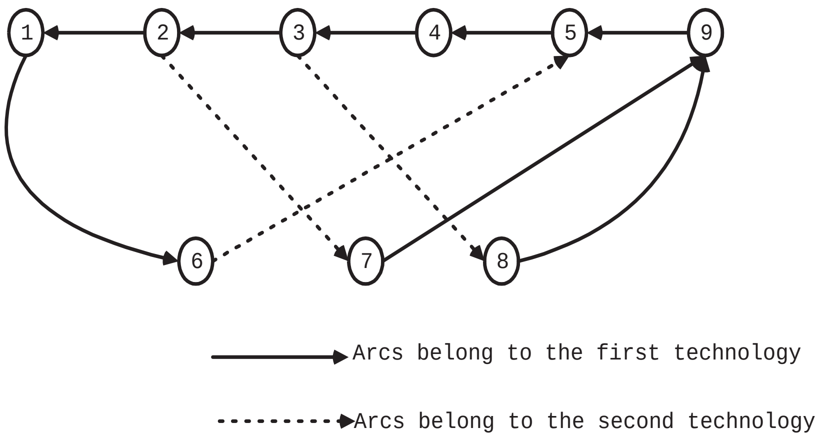

Arcs belong to the first technology
Arcs belong to the second technology

Figure 6a: The reversed graph.

Example 3.3 Consider a graph as shown in Figure 7. At each node i, $f_i^+ = 10$ and $f_i^- = 5$. Transform the graph in Figure 7 into a new graph as shown in Figure 7a. In Figure 7a, the cost of arc (i_1, i_2) is f_i^+ and the cost of arc (i_2, i_1) is f_i^-. Apply Dijkstra's algorithm we can find P_1 =s→2_1 →2_2 →3_2 →3_1 →4_1 →4_2 →5_2 →5_1 →6_1 →7_1 →8_1 →t.

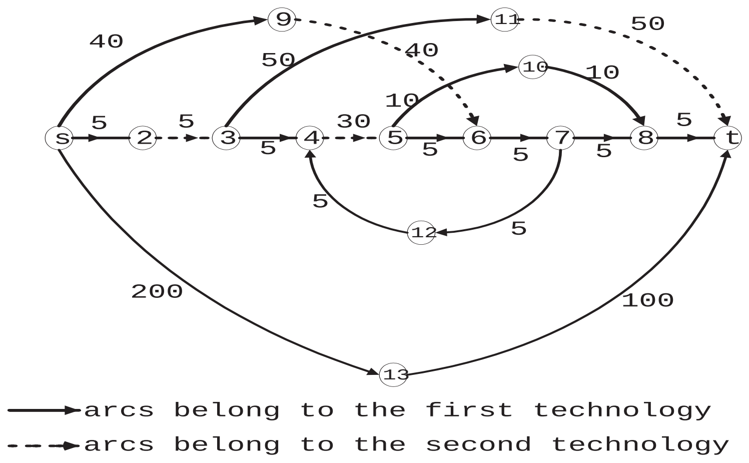

arcs belong to the first technology
arcs belong to the second technology

Figure 7: An example executing Reassemble-Path.

Delete the nodes on P_1 except s and t and the arcs incident to these nodes, then the remaining graph is shown in Figure 7b. Keep on applying Dijkstra's algorithm, we can find

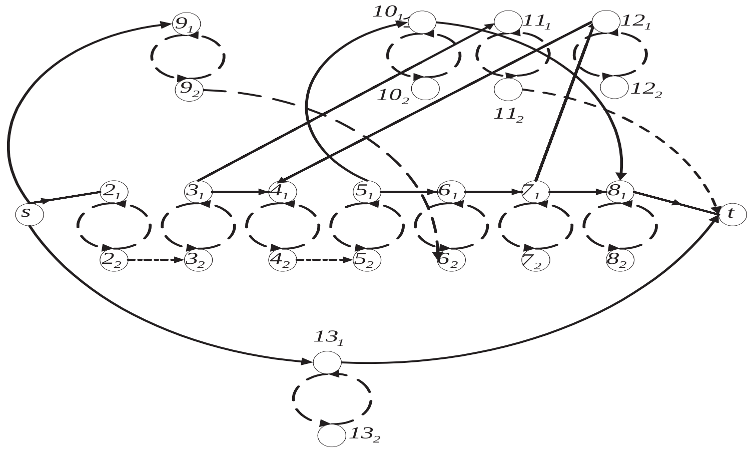


Figure 7a: The transformed graph.

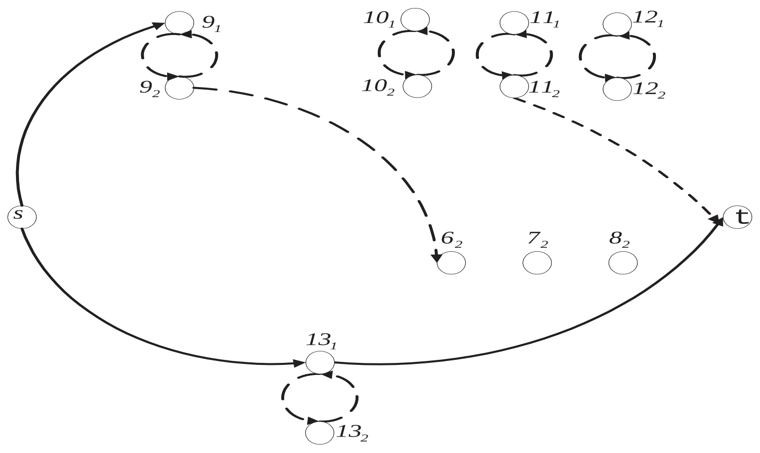
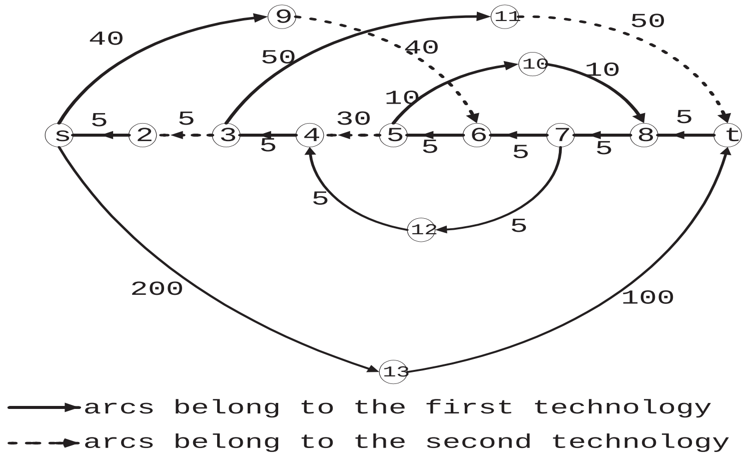


Figure 7b: The remaining graph.

P_2 =1→13_1 →t. Hence we obtain a pair of node disjoint paths with cost 295. Now, use Find-Path, the first path P_1=s→2→3→4→5→6→7→8→t,



40
50
40
50
10
10
5
5
30
5
5
5
5
5
5
200
100
s
2
3
4
5
6
7
8
t
9
10
11
12
13
arcs belong to the first technology
arcs belong to the second technology

Figure 7c: The reversed graph.

and reverse the direction of each arc on P_1, the reversed graph is shown in Figure 7c. Proceeding on Step 2 of Find-Path, the second path P_2 =s→9→6→5→10→8→7→12→4→3→11→t is obtained. There are 3 overlapping parts of P_1 and P_2 and it is not so obvious to reassemble the paths to two node disjoint paths. Thus, we provide Reassemble-Path as follows.

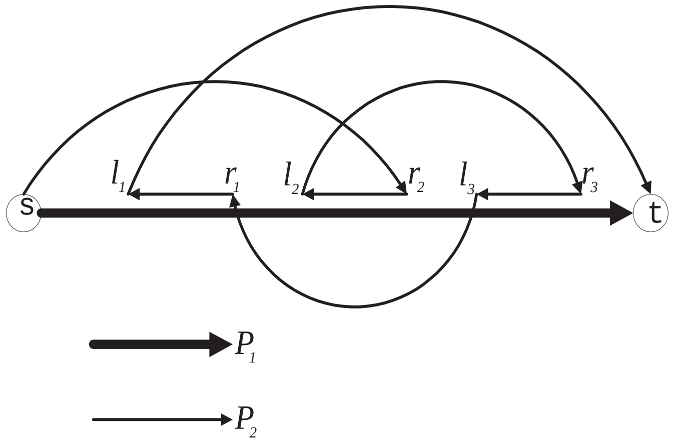


Figure 8: An illustration of paths with overlapping parts.

Let overlapping parts of P_1 and P_2 are called O_1, O_2,... (in order of the distance to s from near to far) and the endnode of O_i closer to s is called l_i and the endnode closer to t is called r_i for each node i. And let v_i be the node on P_2 that meet P_1 when we traverse P_2 from l_i for each node i (Actually v_k is r_j for some j).

Step 3

(**3.1**) Build a tree routed form s. Add r_i, the first node of P_2 that overlaps with P_1 (except s), to the tree at s.

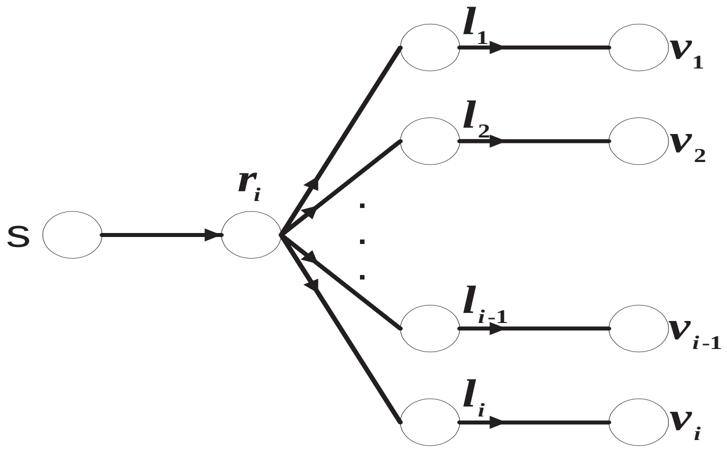


Figure 9a: Steps of building the tree.

(**3.2**)
- Add l_1, l_2, ..., l_i to the tree at r_i. $L = \{l_1, l_2, ..., l_i\}$
- Add v_1 to the tree at l_1, v_2 to the tree at l_2, ..., v_i to the tree at l_i.
- If $v_k = t$, stop building the tree.
- If v_k is closer to s than r_i, delete this branch (v_k is said dead).

(**3.3**)
- For each endnode v_i (except dead one) on the tree, if there are m l_j, called l_{i_1}, l_{i_2}, ..., l_{i_m}, between v and v_g (v_g is the grandfather of v on the tree), for $k = l_{i_1}, l_{i_2}, ..., l_{i_m}$, if $l_k \notin L$,

(1) add l_k to the tree at v_i, and add v_k to the tree at l_k.

(2) add l_k to L.

(3) if $v_k = t$, stop building the tree.

(4) If v_k is closer to s than v, delete this branch (v_k is said dead).

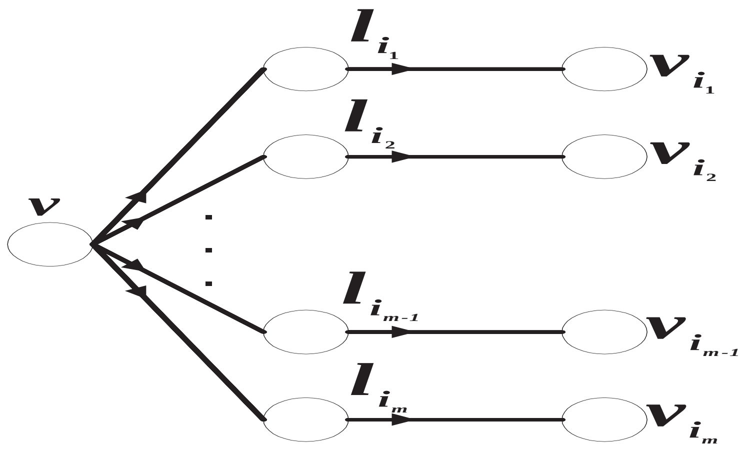


Figure 9b: Steps of building the tree.

(**3**.**4**) Repeat (**3**.**3**).

The branch reach t on the tree can be reassembled into one pair of node disjoint paths. These two paths both start from s. P_1 and P_2 appear alternately in the two paths. We will show how to reassemble a pair of node disjoint paths in the following steps.

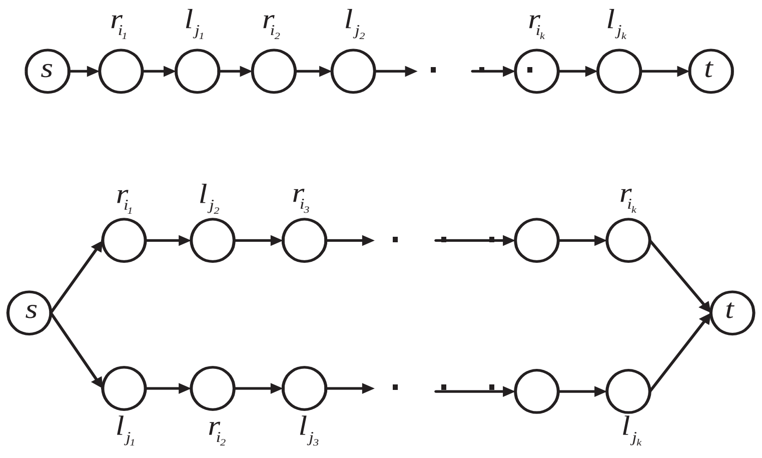


Figure 10: Steps of reassembling two node disjoint paths.

Step 4

For each branch as illustrated in Figure 10, do the following steps:

(4.1) · Add the part of P_1 between s and l_{j_1} to the first path.

· Add the part of P_2 between s and r_{i_1} to the second path.

(4.2) For n=2, 3, ..., k,

· Add the part of P_1 between $r_{i_{n-1}}$ and l_{j_n} to $r_{i_{n-1}}$.

· Add the part of P_2 between $l_{j_{n-1}}$ and l_{i_n} to $l_{j_{n-1}}$.

(4.3) · Add the part of P_1 between r_{i_k} and t to r_{i_k}.

· Add the part of P_2 between l_{j_k} and t to l_{j_k}.

Now, use Reassemble-Path to reassemble P_1 and P_2 in Example 3.3 to two node disjoint paths.

Example 3.3.(Continue) The endnodes of each overlapping part of P_1 and P_2 are (3,4), (5,6) and (7,8), so $l_1 = 3$, $r_1 = 4$, $l_2 = 5$, $r_2 = 6$, $l_3 = 7$, $r_3 = 8$.

Build a tree according to these nodes (see Figure 11a). And the paths we find are $Path2 = s \rightarrow 9 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow t$ and $Path1 = s \rightarrow 2 \rightarrow 3 \rightarrow 11 \rightarrow t$ (As shown in Figure 11b). The total cost of these two paths is 110+135=245.

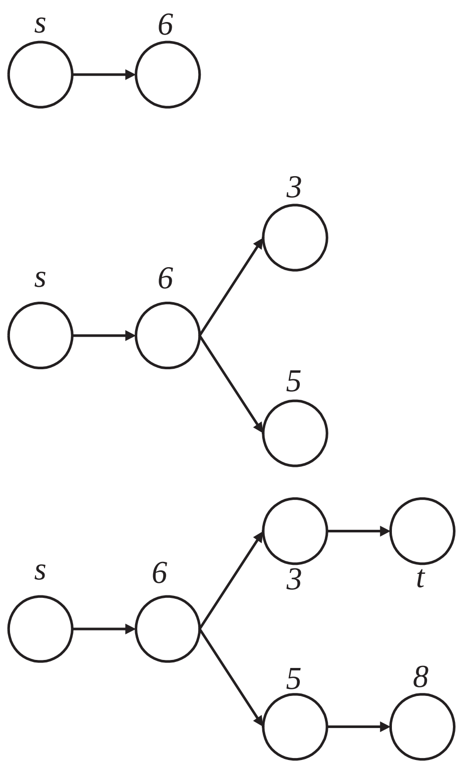


Figure 11a: Building a tree.

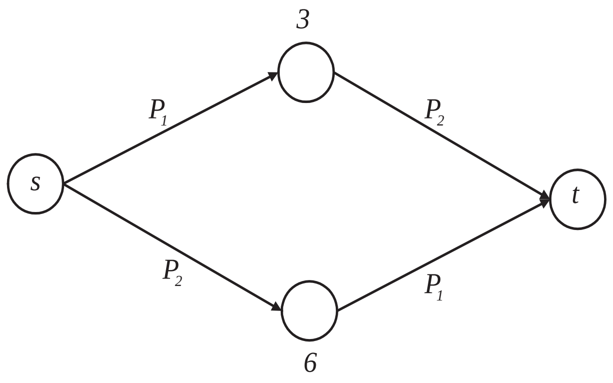


Figure 11b: Reassembled paths.

4 Validity of the Find-Reassemble-Path Algorithm and the Complexity of the Algorithms

In this section, we will show that the Find-Reassemble-Path algorithm finds a pair of node disjoint paths with lower cost than JGL algorithm if the remaining graph is connected. We will also discuss the complexity of this algorithm.

Theorem 4.1. Find-Reassemble-Path algorithm finds two node disjoint paths with lower cost if the remaining graph in JGL algorithm is connected.

Proof:

(i) If the paths P_1 and P_2 found in Find-Path are node disjoint, it is trivial that Find-Reassemble-Path algorithm finds two node disjoint paths with the same cost as JGL algorithm.

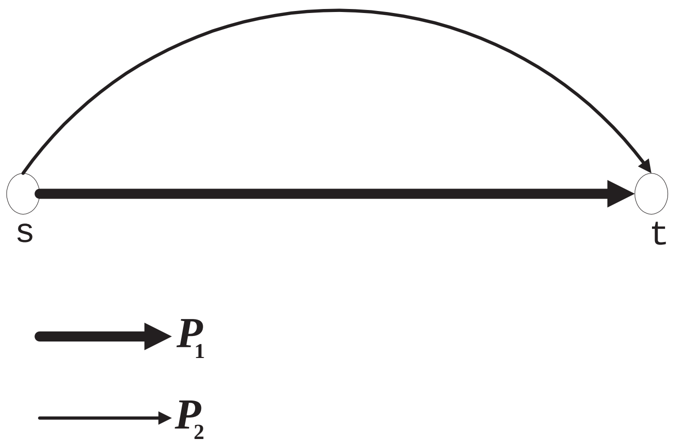


Figure 12a.

(ii) According to the previous section, if P_1 and P_2 found in Find-Path are not node disjoint, some segments of P_1 and P_2 will not be used to reassemble node disjoint paths, thus transmission cost in the paths reassembled is lower than transmission cost in P_1 and P_2.

Let the arc enters l_i on P_1 be a_1, and the arc leaves l_i on P_2 be a_2. If a_1 and a_2 belong to the same technology, connecting a_1 and a_2 doesn't have transition cost. If a_1 and a_2 belong

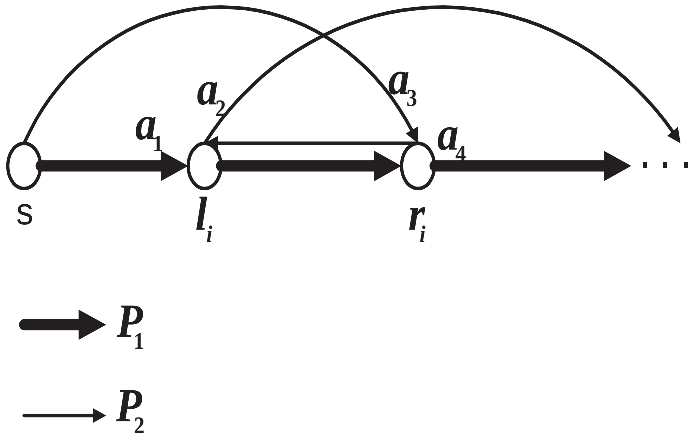


Figure 12b.

to different technologies, without loss of generality we assume that $a_1 \in A_1$ and $a_2 \in A_2$. Connecting a_1 and a_2 has transition cost $f_{l_i}^+$. If the arc closest to l_i in the overlapping part between l_i and r_i belong to A_1, there will be transition cost $f_{l_i}^+$ in P_2 when we traverse P_2. Similarly, if the arc closest to l_i in the overlapping part between l_i and r_i belong to A_2, there will be transition cost $f_{l_i}^+$ in P_1 when we traverse P_1. Hence, as we connect a_1 and a_2, the transition cost won't be increased.

Let the arc enters r_i on P_2 be a_3, and the arc leaves r_i on P_1 be a_4. If a_3 and a_4 belong to the same technology, connecting a_3 and a_4 doesn't have transition cost. If a_3 and a_4 belong to different technologies, connecting a_3 and a_4 has transition cost $f_{l_i}^-$. Like a_1 and a_2, transition cost will not be increase as we connect a_3 and a_4.

Therefore, transition cost in the paths reassembled is lower than transition in P_1 and P_2. □

If the remaining graph in JGL algorithm is not connected. Find-Reassemble-Path may find two node disjoint paths with higher cost than the paths JGL algorithm finds. In the following graph, node 1 is the source and node 8 is the sink and $f_i^+ = f_i^- = 1$ at each node i. When using JGL algorithm, the first path P_1 =1→2→3→4→8. Node 1 and node 8 will be disconnected if we deleting P_1. Then we transform each node as shown in Figure 13a, we can find two arc disjoint paths (also node disjoint),1→2→5→8 and1→6→4→8, with minimum cost in the transformed graph. Thus they find two node disjoint paths with cost 16. But if we use Find-Reassemble-Path algorithm, we will find 1→2→3→7→8 and 1→6→4→8 with cost 17.

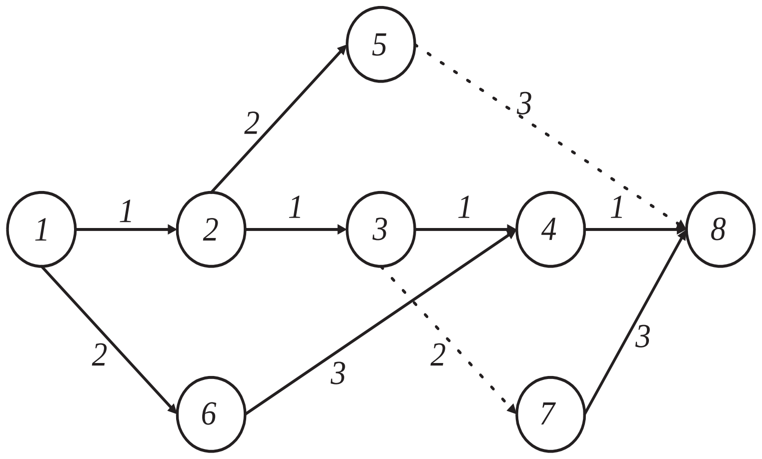


Figure 13: An example that JGL algorithm finds better paths.

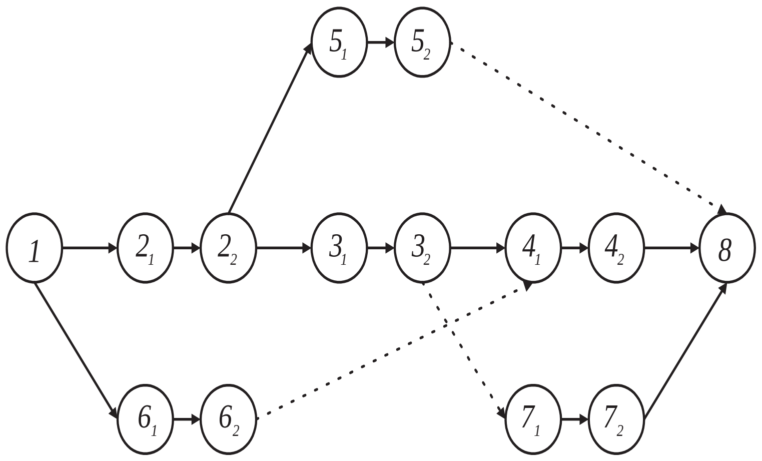


Figure 13a: Transformed graph.

If the remaining graph in JGL algorithm is not connected, JGL algorithm finds a pair of node disjoint paths without considering transition cost and add transition cost to these paths if necessary. It is not certain if there will be transition cost at each node on these paths, so sometimes JGL algorithm finds better paths, sometimes Find-Reassemble-Path algorithm finds better paths. So if the remaining graph is not connected, we execute both algorithms and choose the paths with lower cost. Thus, we can find a pair of node disjoint paths with lower cost than JGL algorithm.

Now, we start to compute the complexity of the Find-Reassemble-Path algorithm. Assume that there are n nodes in the graph. In Step **1.1**, we transform each node except the source s and the sink t into two nodes. Therefore, there are $N = 2n - 2$ nodes in the transformed network. Apply the Dijkstra's algorithm, choose a node i with minimum distance d_i and start labeling the node dominated by i. While labeling there are at most $N-1$ comparisons about the distance. And at most all of the $N-1$ nodes could be chosen as nodes with minimum distance while labeling. So the complexity of Dijkstra's algorithm is $O(N^2)$. In Step **1.3**, reversing the direction of each arc in P_1 only needs linear steps. Thus we know that the total number of steps needed in Step 1 is $O(n^2)$. In Step 2, we also choose a node with minimum cost. Since we record two costs, c_i^1 and c_i^2, in each node i, each node is chosen at most twice as a node with minimum cost. Thus when we choose a node with minimum cost, there will be no more than $2n$ steps. Also each time when we choose a node with minimum cost, the labeling step won't exceed n comparisons of cost. Thus Step 2 is also $O(n^2)$. Hence the complexity of Find-Path is $O(n^2)$.

In step 3,different l_i link to different v_i so no more than n l_i will be add to the tree before we reach t because there is no more than n nodes in the network. Thus step 3 is $O(n)$. And using this branch to reassemble path in step 4 is $O(n)$. Thus the complexity of Reassemble-Path is $O(n^2)$.

5 Conclusions

In this paper, we provide an algorithm that can find a pair of node disjoint paths with lower cost. The first path is the same as Jongh, Gendreau, and Labbe found. The difference is on the second path, we find it using a Dijkstra-like calculation. In the process, we add some constraints during the labeling process in order to construct node disjoint paths. Then using Reassemble Path to reassemble these two paths into two node disjoint paths.

When JGL algorithm finds solutions of node disjoint paths, they deleted all nodes on the first path and arcs incident to the nodes on the first path and find the second path if the remaining graph is connected. We prove that Find-Reassemble-Path can find better paths in this situation. If the remaining graph is not connected they ignore the transition costs. They find a pair of two node disjoint paths without considering the transition cost, then they add transition costs to the path when the paths pass from arcs of one technology to another technology. It is not sure which algorithm finds better solution if the remaining graph is disconnected. In this situation, we choose the better paths found from JGL algorithm and Find-Reassemble-Path algorithm. Thus, we can always find a pair of node disjoint paths with lower cost than JGL algorithm.

References

[1] De Jongh, Gendreau, and Labbe, ″Finding disjoint routes in telecommunications networks with two technologies,″ Operations Research 47, 81-92, 1999.

[2] Even S., Itai A., Shamir A., ″On the complexity of timetable and multi-commodity flow problems,″ SIAM J. Computing 5, 691-703, 1976.

[3] Gould, Rould, ″Graph Theory,″ The Benjamin/Cummings Publishing Company, INC, California, 1988.

[4] Li C. L., McCormick S. T., Simchi-Levi D., ″ The complexity of finding two disjoint paths with min-max objective function,″ Discrete Applied Math 26, 105-115, 1990.

[5] Perl Y., Shiloach Y., ″Finding two disjoint paths between two pairs of vertices in a graph,″ J. ACM 25, 1-9, 1978.

[6] Suurballe J. W., Tarjan R. E., ″A quick method for finding shortest pairs of disjoint paths,″ Networks 14, 325-336, 1984.