ALGORITHMS FOR LINEAR FRACTIONAL SHORTEST PATH PROBLEM WITH TIME WINDOWS

Jinshyang Roan

Department of Business Administration, Soochow University, Taipei, Taiwan

The shortest path problem is one of the most important issues among network problems. In this paper, an extension of this problem is considered wherein each arc possesses two parameters (such as cost and time) and each node constrained by a time window. The object in this problem is to find a path between two specified nodes that minimizes cost per unit time without violating the time window constraints. Since such an objective involves minimizing the fraction of two linear objectives, it is called a linear fractional shortest path problem with time windows (LFSPTW).

The time windows here are hard time windows. In order to avoid the encouragement of arriving early and wait to increase the denominator, there is a penalty $Y$ will be charged to every unit waiting time. An optimal algorithm and a heuristic algorithm to solve the problem are proposed and compared. A sensitivity analysis is also carried out to study the effect of $Y$ on the performance of these two algorithms.

Keywords: network flows, shortest path, algorithm, time window

INTRODUCTION

In the past decade, the convenience of computer networks has created tremendous effects on so many areas. It attracts so many attentions from practical and academic research projects. On the other hand, the shortest path problem is one of the most common problems encountered in the analysis of networks. The shortest path problem is defined as one of finding a path between two designated points of a network having a minimum total length, cost, time, etc. In transportation networks, nodes can represent cities and towns, while the arcs represent the roads connecting the cities and towns. Of course, each arc costs some money and takes some time to traverse. When both cost and time are considered, a path that takes the least of time may cost a lot of money to traverse. On the other hand, if a cheapest path is chosen, it’s possible that it could take a long time. As a result, finding a path that minimizes the cost per unit time becomes an interesting issue. The idea of minimizing cost per unit time can be applied to routing transportation facilities (Bhatia, Swarup, and Puri, 1976), to routing general cargo containership, and to routing hazardous materials (Lindner-Dutton, Batta, and Karwan, 1991; Mirchandani, Turnquist, and Zografos, 1991). Since such an objective involves minimizing the fraction of two linear objectives, it is called a linear fractional shortest path problem (LFSP).
In early 1980s, time window constraints are becoming a popular way to model opening hours of customers, preferred delivery time, etc. in many scheduling and routing problems. The shortest path problem with time windows (SPPTW), first presented by Desrosiers, Pelletier, and Soumis (1983), consists of finding the least cost route between two nodes in a network while visiting each chosen node i within a specified time interval \([a_i, b_i]\). Desrosiers, Soumis, and Desrochers (1984), Gallo and Pallottino (1986) and Desrochers and Soumis (1988a,b) proposed algorithms to solve SPPTW. However, time is the only parameter considered in such a problem.

The purpose of this paper is to find a path that minimizes cost per unit time under time window constraint on each node visited. A model will be formulated. An optimal algorithm and a heuristic algorithm to solve the linear fractional shortest path problem with time windows (LFSPTW) will be proposed and compared.

**MODEL FORMULATION**

The linear fractional shortest path problem with time windows (LFSPTW) consists of finding the least cost per unit time route between two nodes in a network while visiting each chosen node i within a specified time interval \([a_i, b_i]\). Let G(N,A) denote a directed network consisting of a finite set of nodes \(N=\{1, \ldots, n\}\) representing the tasks, a finite set of arcs \(A \subseteq N \times N\), and a source \(s\) and a sink \(q\). Each arc is denoted by an ordered pair \((i, j)\). We call i the tail node and j the head node. Let \(d_{ij}\), a positive integer, denote the time it takes to transverse the arc \((i, j)\) plus the duration of task i. Let \(C_{ij}\), also a positive integer, denote the cost of traversing arc \((i, j)\) plus the cost of task i. Given the time windows \([a_i, b_i]\) for each task, all arcs \((i, j)\) are such that if task \(i\) is executed at the earliest possible time (i.e. at \(a_i\)), task \(j\) may then begin before the end of its time window i.e. all arcs \((i, j)\) respect the condition:

\[
a_i + d_{ij} \leq b_j
\]

Because of the objective here is to minimize cost per unit time, in order to avoid the encouragement of arriving early and wait to increase the denominator, there is a penalty \(Y\) will be charged to every unit waiting time. The problem considered can then be described mathematically as:

Minimize

\[
\sum_{(i,j) \in A} \left\{ C_{ij}x_{ij} + Y \cdot \left[ a_j - (T_i + d_{ij}) \right] \cdot h_{ij}x_{ij} \right\} \over \sum_{(i,j) \in A} \left\{ d_{ij}x_{ij} + \left[ a_j - (T_i + d_{ij}) \right] \cdot h_{ij}x_{ij} \right\}
\]

subject to

\[
\sum_{j \in N} x_{ij} - \sum_{k \in N} x_{ki} = \begin{cases} +1 & i = s \\ 0 & i \in N, i \neq s \text{ or } q \\ -1 & i = q \\ \end{cases}
\]

\[
x_{ij} = 0 \quad \text{if} \ T_i + d_{ij} > T_j, \ (i,j) \in A
\]

\[
h_{ij} = \begin{cases} 1 & \text{if} \ T_i + d_{ij} \leq a_j \\ 0 & \text{otherwise} \\ \end{cases}
\]

\[
x_{ij} = 0 \text{ or } 1 \quad \text{if, } j = 1,2, \ldots, n
\]

where \(T_i\) denotes the time a feasible path takes from source node \(s\) to node \(i\).

**ALGORITHMS**

Here, an optimal algorithm, called Multiple labeling algorithm, and a heuristic algorithm to solve the linear fractional
shortest path problem with time window are presented.

**MULTIPLE LABELING ALGORITHM**

This algorithm is a modification of the Label-correcting algorithm to solve the shortest path problem. This algorithm is referred to as multiple labeling because unlike the labeling algorithms to solve the simple shortest problem, a node can have many labels associated with it. Here, a label is associated with each path. Specifically, the label consists of two elements, the total cost and the total time associated with the corresponding path. In the simple shortest path problem, a path associated with a small cost or time dominates other paths with larger cost or time. Hence, these other paths can be discarded based on these values. However, to solve (LFSPTw) a label cannot be discarded based on cost to time ratios alone. For instance, in Figure 1, after arc (1,3) is traversed, the ratio of the label associated with this path p1 is $\frac{4}{5}$. There is another path p2 consisting of arcs (1,2) and (2,3). The ratio of the label associated with this path is $\frac{7}{10}$. If path p1 is extended by arc (3,4), then the resulting ratio of the path is $\frac{6}{11}$, while if p2 is also extended by the arc (3,4) the ratio obtained is $\frac{9}{16}$. Since $\frac{6}{11}$ is less than $\frac{9}{16}$, path p1 cannot be discarded.

For node j, let $C_k(j)$ denote the total cost of kth label and $T_k(j)$ denote total time of kth label. The ratio associated with the kth label of node j is $R_k(j) = C_k(j)/T_k(j)$. The multiple labeling algorithm is now presented. In the algorithm stated below, LB(i) is the number of labels associated with node i, P(•) denotes the predecessor list function that can be used to trace back to the previous node, and PP(•) is the predecessor label function that can be used to determine the correct label of previous node. List(•) is the eligible list. List(i) equal to 0 implies that all arcs out of node i have been examined, while List(i) equal to 1 implies that at least one of the arcs out of node i have to be examined again.

Multiple Labeling Algorithm

**STEP 1: [Initialization]**

1. Read the data in ladder form and convert it to forward star form.
2. Set $C_1(s) := 0$ and $C_0(i) := \infty$ for all other nodes in the network. (s: source node)
3. Set $T_1(i) := 0$ and $R_1(i) := 0$ for all the nodes in the network.
4. Set $L_B(s) := 1$ and $L_B(i) := 0$ for all other nodes in the network.
5. Set $P_1(i) := 0$ for all nodes in the network.
6. Set $P_P(i) := 1$ for all nodes in the network.
7. Set $List(s) := 1$ and $List(i) := 0$ for all other nodes in the network.

**STEP 2: [Node, Label selection]**
Select a label of node i whose $List(i) = 1$.
If no such node exist, GO TO STEP 4.

**STEP 3: [Ratio, Tree and List Update]**

1. For each arc in the forward star of
node i determine if
\[ T_m(i) + T_{ij} < b_j, \text{ then} \]
\[ T'(j) = \max \{ a_j, T_m(i) + T_{ij} \} ; \]
if \( T_m(i) + T_{ij} \geq a_j, \text{ then} \)
\[ C'(j) = C_m(i) + C_{ij} ; \text{otherwise,} \]
\[ C'(j) = C_m(i) + C_{ij} + Y[a_j - (T_m(i) + T_{ij})] \]
for those labels of node i that have not yet
been examined.

Rearrange all the remaining labels of
node j.

Set \( k = \text{LB}(j) \)
Set \( C_{k+1}(j) = C'(j) \)
Set \( T_{k+1}(j) = T'(j) \)
Set \( R_{k+1}(j) = C'(j) / T'(j) \)
Set \( P_{k+1}(j) = i \)
Set \( P_{P_{k+1}(j)} = m \)
Set \( \text{LB}(j) = \text{LB}(j) + 1 \)
Set \( \text{List}(j) = 1 \) if \( \text{List}(j) = 0 \).

(2) Set \( \text{List}(i) = 0 \).

(3) GO TO STEP 2.

STEP 4: [Termination]

(1) Find \( R_m(q) \) which has the minimum
ratio.

(2) If \( R_m(q) = \infty \), no path exists from
node s to node q.

(3) Use predecessor list function \( P \) and
predecessor label function \( PP \) to
trace path from q to s.

Heuristic Algorithm

As we can see, using the optimal algorithm,
the number of labels of a node may be
numerous when a network is large. If the
number of labels can be reduced, the solving
algorithm will be less complicate. The
heuristic algorithm presented here is assumed
that any node on a feasible path can only
possess two labels. Label 1 is used to trace
the path that uses least time, while label 2 is
used to trace the path that uses most time.
The heuristic algorithm is now presented
below:

Heuristic Algorithm

STEP 1: [Initialization]

(1) Read the data in ladder form and
convert it to forward star form.

(2) Set \( C_1(s) := 0 \) and \( R_0(i) := \infty \) for all
other nodes in the network. (s:
source node)

(3) Set \( T_1(i) := \infty \) and \( T_2(i) := 0 \) for all
the nodes in the network.

(4) Set \( \text{LB}(s) := 1 \) and \( \text{LB}(i) := 2 \) for all
other nodes in the network.

(5) Set \( \text{P}_1(i) := 0 \) for all nodes in the
network.

(6) Set \( \text{P}_1(i) := 1 \) for all nodes in the
network.

(7) Set \( \text{List}(s) := 1 \) and \( \text{List}(i) := 0 \) for
all other nodes in the network.

STEP 2: [Node, Label selection]
Select a label of node i whose \( \text{List}(i) = 1 \).
If no such node exist, GO TO STEP 4.

STEP 3: [Ratio, Tree and List Update]

(1) For each arc in the forward star of
node i determine if
\[ T_m(i) + T_{ij} < b_j, \text{ then} \]
\[ T'(j) = \max \{ a_j, T_m(i) + T_{ij} \} ; \]
if \( T_m(i) + T_{ij} \geq a_j, \text{ then} \)
\[ C'(j) = C_m(i) + C_{ij} ; \text{otherwise,} \]
\[ C'(j) = C_m(i) + C_{ij} + Y[a_j - (T_m(i) + T_{ij})] \]
for those labels of node i that have not yet
been examined.

(2) If \( T'(j) > T_2(j) \), then
Set \( T_2(j) = T'(j) \)
Set \( C_2(j) = C'(j) \)
Set \( R_2(j) = C'(j)/T'(j) \)
Set \( P_2(j) = i \)
Set \( PP_2(j) = m \)
Set \( List(j) = 1 \) if \( List(j) = 0 \).
If \( T'(j) < T_2(j) \), then
Set \( T_1(j) = T'(j) \)
Set \( C_1(j) = C'(j) \)
Set \( R_1(j) = C'(j)/T'(j) \)
Set \( P_1(j) = i \)
Set \( PP_1(j) = m \)
Set \( List(j) = 1 \) if \( List(j) = 0 \).

(3) Set \( List(i) = 0 \).
(4) GO TO STEP 2.

STEP 4: [Termination]
(1) Find \( R_m(q) \) which has the minimum ratio.
(2) If \( R_m(q) = \infty \), no path exists from node \( s \) to node \( q \).
(3) Use predecessor list function \( P \) and predecessor label function \( PP \) to trace path from \( q \) to \( s \).

COMPUTATIONAL STUDY

The principle goal of the computational experiment is to study the efficiency of this algorithm in a simulated network environment. Hence, we need to generate networks that depict structures encountered when solving the linear fractional shortest path problem with time window. Also, we need to generate enough problems of sufficient variety so that there is validity to any conclusions that are drawn.

The network problems were generated by a network generator program. The program creates random connected networks with a specified number of nodes \( N \) and arcs \( M \). The program first creates a spanning tree rooted at node 1 (i.e. a tree in which all other nodes in the network are accessible from node 1). This is accomplished by starting with a tree containing only the root, and augmenting it at every stage by an arc \( (a,b) \), where \( a \) is a node chosen at random from those nodes already in the tree and \( b \) is chosen at random from those nodes not in the tree. The final \( M - N + 1 \) arcs are then obtained by randomly choosing pairs of nodes from the set of all possible node pairs. Starting with a spanning tree rooted at node 1 insures that all nodes are accessible from that node at least. In practice this has been true as well from other nodes chosen as root (Gilsinn and Witzgall, 1973). The costs and times of these arcs follow a uniform distribution. The network is then represented in a forward star form.

The test problems were generated by varying two parameters: the number of nodes and the number of arcs in the network. For the evaluation of both algorithms, we set the number of nodes to equal 30, 50, and 80. The number of arcs for the 30-node problems was set to equal 80, 120, and 200. The number of arcs for the 50-node problems was set to equal 150, 250, and 500. The number of arcs for the 80-node problems was set to equal 250. All of these networks are acyclic. For each of these node-arc combinations we set the arc length in the range \([1,50]\). The width of the time window at each node is 25,
which is the average of the arc length. The reason of choosing the average is to ensure all the possible situations when a path reaches a node can happen. In other words, node i can be visited before the earliest possible time a_i, within the time window, or after the latest possible time b_i. The penalty of unit waiting time is 3. However, a sensitivity analysis of \( Y \) is also carried out. Thus 7 different node-arc combinations were identified. In order to avoid generating an unusually difficult or easy network, which would make the results of the study imprecise, for each, identified combination, we randomly generated 20 different networks. This gives us 20 replications for each combination. Hence, the running times, the number of iterations and number of labels reported are an average of the 20 individual times gathered.

The algorithm was coded in C. All testing was undertaken on a Pentium 200. The results of the computational runs are presented in four tables. Table 1 shows the average running time for the 20 replications for the Multiple Labeling Algorithm and the Heuristic Algorithm. Table 2 shows the average number of labels for the 20 replications for the Multiple Labeling Algorithm and the Heuristic Algorithm. The average number of labels is obtained by adding up all the labels considered for each node and dividing by the number of nodes. Table 3 presents results obtained by both algorithms. Table 4 presents the sensitivity analysis of \( Y \).

According to Table 1, it can be easily seen that the Multiple Labeling Algorithm takes much more time than the Heuristic Algorithm. When the size of networks becomes larger, the time taken by the Multiple Labeling Algorithm to solve a network increases dramatically, while the time taken by the Heuristic Algorithm increases slightly.

### Table 1

Running Time (in seconds) for the 20 Replications for the Multiple Labeling Algorithm and the Heuristic Algorithm

<table>
<thead>
<tr>
<th>nodes</th>
<th>30</th>
<th>50</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>arcs</td>
<td>80</td>
<td>120</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>250</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple Labeling Algorithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Highest</td>
<td>0.44</td>
<td>0.78</td>
<td>4.52</td>
</tr>
<tr>
<td>Lowest</td>
<td>0.22</td>
<td>0.57</td>
<td>2.88</td>
</tr>
<tr>
<td>Average</td>
<td>0.291</td>
<td>0.641</td>
<td>3.487</td>
</tr>
<tr>
<td>Heuristic Algorithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Highest</td>
<td>0.06</td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td>Lowest</td>
<td>0.05</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>Average</td>
<td>0.058</td>
<td>0.0713</td>
<td>0.098</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Labels for the 20 Replications for the Multiple Labeling Algorithm and the Heuristic Algorithm</td>
</tr>
</tbody>
</table>

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
<td>50</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arcs</td>
<td>80</td>
<td>120</td>
<td>200</td>
<td>150</td>
<td>250</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple Labeling Algorithm</td>
<td>Highest</td>
<td>1998</td>
<td>2922</td>
<td>10124</td>
<td>11258</td>
<td>33148</td>
</tr>
<tr>
<td></td>
<td>Lowest</td>
<td>722</td>
<td>1888</td>
<td>7763</td>
<td>7938</td>
<td>19132</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>1115.2</td>
<td>2454.4</td>
<td>8524.7</td>
<td>9059.8</td>
<td>28103</td>
</tr>
<tr>
<td>Heuristic Algorithm</td>
<td>Highest</td>
<td>59</td>
<td>59</td>
<td>59</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>Lowest</td>
<td>59</td>
<td>59</td>
<td>59</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>59</td>
<td>59</td>
<td>59</td>
<td>99</td>
<td>99</td>
</tr>
</tbody>
</table>

Table 2 presents results describing the amount of computer memory required by the Multiple Labeling Algorithm and the Heuristic Algorithm. It is clear that the memory requirements of the Multiple Labeling Algorithm are dynamic with the requirements increasing rapidly as the problem size increases. In comparison, the memory requirements of the Heuristic Algorithm are static and increase linearly with number of nodes of problem.

In Table 3, the heuristic solution obtained by the Heuristic Algorithm is closer to per unit waiting time penalty $Y (= 3)$, while the Multiple Labeling Algorithm obtains a better solution. This is because the Heuristic Algorithm tends to visit a node before its latest possible time. Consequently, it has a better chance to be "punished".

<table>
<thead>
<tr>
<th>Table 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratios for the 20 Replications for the Multiple Labeling Algorithm and the Heuristic Algorithm</td>
</tr>
</tbody>
</table>

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
<td>50</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arcs</td>
<td>80</td>
<td>120</td>
<td>200</td>
<td>150</td>
<td>250</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple Labeling Algorithm</td>
<td>Highest</td>
<td>2.3361</td>
<td>2.0002</td>
<td>1.9665</td>
<td>2.2235</td>
<td>2.1327</td>
</tr>
<tr>
<td></td>
<td>Lowest</td>
<td>1.5266</td>
<td>1.4212</td>
<td>1.3143</td>
<td>1.4928</td>
<td>1.3215</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>1.8855</td>
<td>1.6835</td>
<td>1.6164</td>
<td>1.9395</td>
<td>1.8091</td>
</tr>
<tr>
<td>Heuristic Algorithm</td>
<td>Highest</td>
<td>2.9692</td>
<td>2.9041</td>
<td>2.9207</td>
<td>2.8976</td>
<td>2.9405</td>
</tr>
<tr>
<td></td>
<td>Lowest</td>
<td>2.1567</td>
<td>2.2171</td>
<td>2.2218</td>
<td>2.4443</td>
<td>2.5273</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>2.5986</td>
<td>2.6747</td>
<td>2.6959</td>
<td>2.6707</td>
<td>2.7894</td>
</tr>
</tbody>
</table>
As network density increases, the solution obtained by the Multiple Labeling Algorithm decreases. This is due to the more possible paths can be chose when density becomes larger. On the other hand, the heuristic solution obtained by the Heuristic Algorithm increases, as network density becomes larger. The reason is the more possible paths there are, the better chance the Heuristic Algorithm can find a path visit a node earlier, as well as be punished.

In Table 4, the sensitivity analysis of $Y$ is undertaken when number of nodes is 50 and number of arcs is 250. The solutions obtained by the Multiple Labeling Algorithm and the Heuristic Algorithm are shown.

According to Table 4, the heuristic solution obtained by the Heuristic Algorithm is closer to the optimal solution obtained by the Multiple Labeling Algorithm when $Y$ is smaller. When $Y$ is 0, the heuristic solution obtained equals to the optimal solution occasionally. As $Y$ increases, the difference between two solutions becomes larger.

TABLE 4

<table>
<thead>
<tr>
<th></th>
<th>0.0</th>
<th>0.3</th>
<th>0.8</th>
<th>1.0</th>
<th>3.0</th>
<th>10.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Multiple</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Labeling</td>
<td>Highest</td>
<td>0.1032</td>
<td>0.3179</td>
<td>0.7923</td>
<td>0.8124</td>
<td>2.1327</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Lowest</td>
<td>0.0097</td>
<td>0.2819</td>
<td>0.6928</td>
<td>0.6485</td>
<td>1.3215</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>0.0541</td>
<td>0.2992</td>
<td>0.7537</td>
<td>0.7544</td>
<td>1.8091</td>
</tr>
<tr>
<td><strong>Heuristic</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td>Highest</td>
<td>0.1301</td>
<td>0.3799</td>
<td>0.8574</td>
<td>1.0235</td>
<td>2.9405</td>
</tr>
<tr>
<td></td>
<td>Lowest</td>
<td>0.0097</td>
<td>0.3031</td>
<td>0.7743</td>
<td>0.9260</td>
<td>2.5273</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>0.0614</td>
<td>0.3361</td>
<td>0.8050</td>
<td>0.9748</td>
<td>2.7895</td>
</tr>
</tbody>
</table>
CONCLUSION

This paper has proposed a problem called linear fractional shortest path problem with time windows, where each arc possesses two parameters (e.g. cost and time) and each node constrained by a time window. The object is to find a path that minimizes cost per unit time between two specified nodes without violating time window constraints. Because of the character of fraction, two algorithms are presented. One is the Multiple Labeling Algorithm, which assumed each node can possesses more than one label, can solve the problem optimally. The other is the Heuristic Algorithm, which assumed each node possesses no more than two labels in order to reduce the complexity.

The performance of the algorithms was tested on several randomly generated networks with varying sizes. Based on the computational results it is clear to see the trade-off between the time taken to solve the problem and the accuracy of the solution. The average solution ratio obtained by the Heuristic Algorithm is close to per unit waiting time penalty $Y$. Also, we found the difference between the solution ratios obtained by the algorithms becomes smaller when $Y$ decreases.

The algorithms presented here are based on label-correcting algorithm. A modification from the algorithms proposed to solve shortest path problem with time windows (SPPTW) is one area of possible future research. The time windows here are assumed to be hard time windows. Another area of possible future research is assuming the time windows are soft. One can pay some cost to enter a node before its earliest possible time. These issues have been included in the authors' future research plans.

REFERENCES


