Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computers & Operations Research 32 (2005) 571–581

computers & operations research

www.elsevier.com/locate/dsw

Finding K shortest looping paths in a traffic-light network

Hsu-Hao Yang[a,*], Yen-Liang Chen[b]

[a]Department of Industrial Engineering & Management, National Chinyi Institute of Technology, Taiping, Taichung 411, Taiwan, Republic of China

[b]Department of Information Management, National Central University, Chung-Li, Taiwan 320, Republic of China

Abstract

In this paper we consider a traffic-light network where every node in the network is associated with the constraint consisting of a repeated sequence of time windows. The constraint aims to simulate the operations of traffic-light control in an intersection of roads. With this kind of constraints in place, directions of routes when passing through the intersections can be formally modeled. The objective of this paper is to find the first K shortest looping paths in the traffic-light network. An algorithm of time complexity of $O(rK^2|V_1|^3)$ is developed, where r is the number of different windows of a node and $|V_1|$ is the number of nodes in the original network.

© 2003 Elsevier Ltd. All rights reserved.

Keywords: Shortest path; Traffic-light; Looping path

1. Introduction

A shortest path problem (SPP) deals with finding a path with minimum time, distance, or cost from a source node to a destination node through a connected network. It is an important issue because of its wide range of applications in transportations (Swersey and Ballard [1]) and communications (Chen and Chin [2]). We refer readers to Bodin et al. [3], Deo and Pang [4], and Golden and Magnanti [5] for more details.

The time-constrained shortest path problem (TCSPP) generalizes the SPP and has been the focus of study over the years. Time window appears to be a common form of time constraint that assumes that a node can be visited only in a specified time interval (Desrochers et al. [6];

* Corresponding author. Tel.: +886-4-3924505; fax: +886-4-3934620.
E-mail addresses: yanghh@chinyi.ncit.edu.tw (H.-H. Yang), ylchen@mgt.ncu.edu.tw (Y.-L. Chen).

0305-0548/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2003.08.004

Dumas et al. [7]; Kohl and Madsen [8]). In other words, a time window defines the earliest time and the latest time that the node is available. In a recent paper, Chen and Yang [9] introduced a new kind of time constraint called traffic-light constraint that associates each node with a repeated sequence of different windows in which directions of routes are also taken into account. By this formulation, finding the quickest path in a city with a number of traffic-light controls is equivalent to solving a SPP in the formulated traffic-light network. Since the introduction of the traffic-light network, the same authors have proposed several extensions. For example, to reflect what occurs in practice, Chen and Yang [10] considered "weighed number of stops" as an additional criterion and solved this bi-criteria problem in polynomial time if a maximum weighted number of stops is given. Another extension is to find K shortest paths instead of one path only. One of reasons to do so is that certain constraints may be difficult to define or hard to optimize; a common strategy is to compute several paths and then choose among them by considering the other criteria (Eppstein [11]).

Traditionally, the first K shortest paths found can be members of two major classes: (1) simple paths (paths without repeated nodes and arcs), and (2) looping paths (paths with repeated nodes and arcs). These two classes of paths have long been studied together because they are not only similar (finding K paths) in general but also complementary (simple versus looping) in particular. This fact motivated the development of not only algorithms for finding the first K simple paths (Yang and Chen [12]), but also algorithms for finding the first K looping paths. Moreover, the *path deletion* algorithm proposed in this paper differs from the *path partition* algorithm of Yang and Chen [12] in that, to find the next shortest path given one is available, the former algorithm removes the entire path while the latter algorithm partitions the path into a set of sub-paths.

As we stated above, the first K shortest paths found can be classified as simple paths or looping paths. Regardless of the network under consideration, the efforts required to find simple paths appear to be harder than those to find looping paths. In the first class, Yen [13] proposed a very efficient algorithm that finds the first K simple paths in a general network in $O(K|V|^3)$ time, where $|V|$ is the number of nodes. Katoh et al. [14] improved the time bound to be $O(K(|A|+|V|\log|V|))$ for an undirected network, where $|A|$ is the number of arcs. Recently, Hadjiconstantinou and Christofides [15] presented an efficient implementation of a *KSP* algorithm based on the method of Katoh et al. [14]; they suggested that the *KSP* algorithm is suitable for finding a large number of simple paths between any pair of nodes. In the second class, Dreyfus [16] developed an efficient algorithm that found the K shortest paths from one node to each one of the other nodes in the time of $O(K|V|\log|V|)$, once the shortest tree has been determined. Fox [17] gave an algorithm to run in $O(|V|^2+K|V|\log|V|)$ time. Using a concept of path deletion, Martins [18] developed an algorithm with the worst case to be $O(K^3|V|)$. Later, this worst case time was improved to be $O(K^2|A|)$ by Azevedo et al. [19] using an efficient computational implementation. According to Azevedo et al. [19], however, comparative computational experiments showed that their algorithm outperformed that of Dreyfus [16] despite its worse complexity. For a given pair of nodes, Eppstein [11] used an implicit representation of paths to significantly improve the time to be $O(|A|+|V|\log|V|+K|V|)$.

The rest of this paper is organized as follows. In Section 2, we describe the traffic-light network and develop an algorithm for finding the first K shortest looping paths in the present network. We also provide the time complexity of the algorithm in this section. We include the conclusion and directions for future research in Section 3.

2. The problem and solution

In this section, we first define the traffic-light network in Section 2.1. In Section 2.2, we introduce algorithms of Martins [18] and Azevedo et al. [19] that closely relate to our algorithm. We present our algorithm with an example, and show the complexity of the algorithm in Section 2.3.

2.1. Problem definition

To be consistent, we follow the notations of Chen and Yang [9]. Let $N=(V_1\cup V_2, A, WL, t, s, d)$ denote a traffic-light network, where V_1 is the node set without window constraints, V_2 is the node set with window constraints, A is the arc set without multiple arcs and self-loops, $t(u,v)$ is the travel time of arc $(u,v)\in A$. For each node $u\in V_2$, it is associated with a window-list $WL(u)=(ws_u, w_{u,1}, w_{u,2},\dots,w_{u,r})$, where ws_u is the starting time of the first window and $w_{u,i}$ is the ith time window of node u for $i=1$ to r. Each window $w_{u,i}$ is associated with a duration $d_{u,i}$ and a set of node-triplets $NT_{u,i}$, where a node-triplet $\langle x,u,y\rangle$ is in $NT_{u,i}$ if visiting node y from node x is allowed in window $w_{u,i}$. If we represent windows using a repeated sequence and by assuming $w_{u,0}=w_{u,r}$, we have the relationship that $w_{u,(k\times r)+i}=w_{u,i}$ for any nonnegative integers k and i, where $i\leqslant r$. In this context, the sequence of the windows describes the whole phasing of the traffic-light signal.

Since a node u in V_1 can be regarded as a node in V_2 by associating it with a window of infinite duration and containing all possible node-triplets, we assume that all the nodes are in the set V_2 for ease of presentation. Consider Fig. 1a that illustrates the traffic-light network, where the number beside each arc is the arc's travel time. We also show each node's, say u, duration $d_{u,i}$ and its node-triplets $NT_{u,i}$ wherever appropriate. For example, the first window of node C starts at time 3; the duration of window $w_{C,1+2i}$ is 2 and the duration of window $w_{C,2+2i}$ is 4 where i is a nonnegative integer. The triplet $\langle A,C,d\rangle$ is the allowable route in window $w_{C,1+2i}$; $\langle B,C,d\rangle$ and $\langle D,C,d\rangle$ are allowable in window $w_{C,2+2i}$. Therefore, if at node C from node A, we can visit node d only in window $w_{C,1+2i}$, and so on. Chen and Yang [9] have developed an algorithm, which labels arcs rather than nodes, to find the shortest path as (s,A,D,d) with total time 11 (Fig. 1b). Because the algorithm labels arcs, each arc (u,v) in A is associated with a label *arrived*(u,v) to denote the earliest time to arrive at node v through arc (u,v). We briefly explain how we compute the total time 11 in Fig. 1b. Because there are two arcs leaving node s, arcs (s,A) and (s,B), we need to compute the values of *arrived*(s,A) and *arrived*(s,B), which are simply 4 and 5, respectively. Now consider node A. Although we visit node A at time 4, we cannot leave for node B or node C until the beginning time of window $w_{A,2}$ that contains the node-triplets $\langle s,A,B\rangle$ and $\langle s,A,C\rangle$. Because this time is 5, *arrived*$(A,B)=5+3=8$ and *arrived*$(A,C)=5+5=10$. In contrast, we can leave for node D immediately because time 4 falls within window $w_{A,1}$, and hence *arrived*$(A,D)=4+1=5$. Similarly, consider node B where *arrived*$(s,B)=5$. Since the first time period of $\langle s,B,C\rangle$ and $\langle s,B,D\rangle$ in window $w_{B,1}$ is from 2 to 4, we must wait until the second time period of $w_{B,1}$, which is 8. Therefore, *arrived*$(B,C)=8+2=10$ and $(B,D)=8+7=15$. Continuing the same way, we eventually find the shortest path is (s,A,D,d) with time 11. For more details, see Chen and Yang [9].

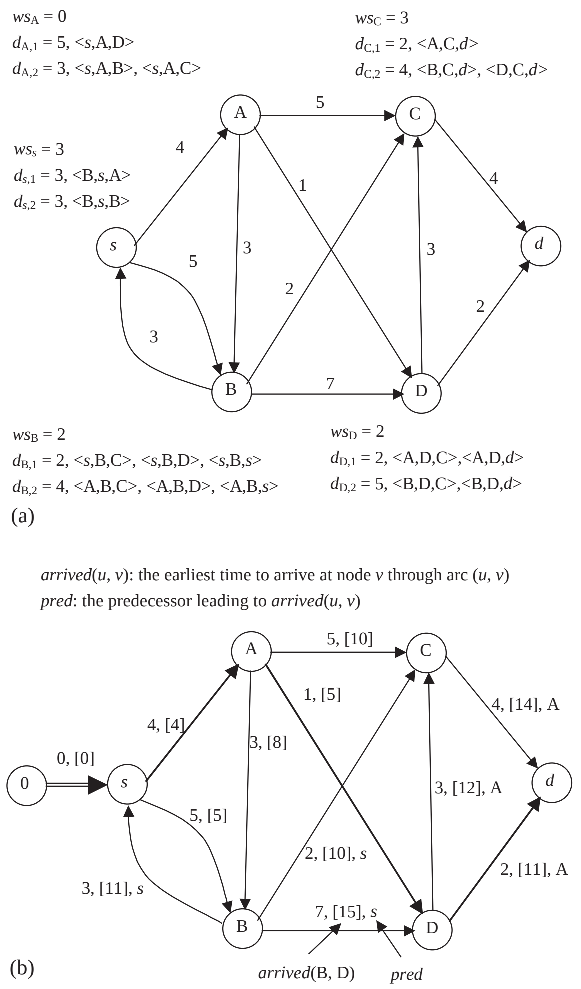
$ws_A = 0$
$d_{A,1} = 5$, <s,A,D>
$d_{A,2} = 3$, <s,A,B>, <s,A,C>
$ws_C = 3$
$d_{C,1} = 2$, <A,C,d>
$d_{C,2} = 4$, <B,C,d>, <D,C,d>
$ws_s = 3$
$d_{s,1} = 3$, <B,s,A>
$d_{s,2} = 3$, <B,s,B>
$ws_B = 2$
$d_{B,1} = 2$, <s,B,C>, <s,B,D>, <s,B,s>
$d_{B,2} = 4$, <A,B,C>, <A,B,D>, <A,B,s>
$ws_D = 2$
$d_{D,1} = 2$, <A,D,C>,<A,D,d>
$d_{D,2} = 5$, <B,D,C>,<B,D,d>
(a)
arrived(u, v): the earliest time to arrive at node v through arc (u, v)
pred: the predecessor leading to arrived(u, v)
5, [10]
1, [5]
4, [14], A
4, [4]
3, [8]
0, [0]
3, [12], A
5, [5]
2, [10], s
2, [11], A
3, [11], s
7, [15], s
arrived(B, D)
pred
(b)

Fig. 1. (a) The traffic-light network and (b) the shortest path p in the traffic-light network.

2.2. The algorithms by Martins and Azevedo et al.

Since the algorithms proposed by Martins [18] and Azevedo et al. [19] form the basis of our algorithm, we will describe their algorithms and complexities before we proceed. For each network N, Martins' algorithm [18] executes two algorithms: (1) a shortest path algorithm to find the shortest path p, and (2) a path deletion algorithm to generate a new network N'. The main idea of this path deletion algorithm is that once a shortest path was determined, the path would be removed from the network. In fact, by adding new well-defined nodes and arcs, this path deletion algorithm results in an enlarged network where all the paths but the deleted one can be determined. That is, given a shortest path p in the network N, the path deletion algorithm will delete p from the network in such a way that a new network N' is generated where all paths but p in N' can be determined from N. In terms of finding the first K shortest paths given N_1 as the initial network, this means that a

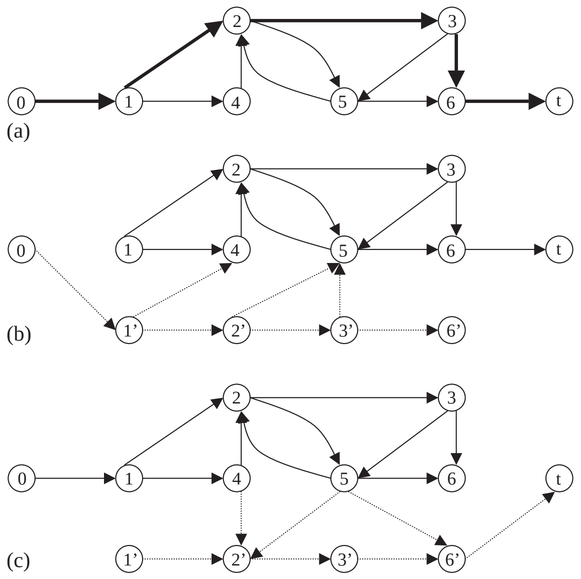

Fig. 2. (a) The sample network N with the shortest path p, (b) delete p from N to generate N' using Martins' algorithm and (c) delete p from N to generate N' using the algorithm of Azevedo et al.

sequence of networks $\{N_1, N_2, \ldots, N_i \ldots, N_k\}$ will be generated where the ith shortest path p_i will be determined from N_i. In total, Martins' algorithm [18] executes the shortest path algorithm K times and the path deletion algorithm $K-1$ times for a given network to find the K shortest paths; the worst case of the algorithm is $O(K^3|V|)$.

To show how the path deletion algorithm works, assume $p=(0=v_0, s=v_1, \ldots, v_{m-2}, v_{m-1}=d, v_m=t)$ is a shortest path in some N, where $m \geqslant 3$. Following the definitions of Azevedo et al. [19], for a node u, let $O(u) = \{(u,v) \in A \mid v \in N\}$ be the set of its outgoing arcs, and for a node v, let $I(v) = \{(u,v) \in A \mid u \in N\}$ be the set of its incoming arcs. To obtain the enlarged network N' by deleting p, Martins' path deletion algorithm [18] adds new nodes, adds set of *outgoing* arcs of these new nodes, and deletes the *first* arc of p. The algorithm is as follows:

1. The set $\{v'_1, v'_2, \ldots, v'_{m-1}\}$ of new nodes is added to N.
2. The set of outgoing arcs of node v_0 is updated as

$$O(v_0) = O(v_0) - \{(v_0, v_1)\} \cup \{(v_0, v'_1)\}.$$

3. The set of outgoing arcs of each new node is defined as

$$O(v'_i) = \{(v'_i, v) \mid (v_i, v) \in O(v_i) \text{ and } v \neq v_{i+1}\} \cup \{(v'_i, v'_{i+1})\}, \quad \forall i \in \{1, \ldots, m-2\}, \text{ and}$$

$$O(v'_{m-1}) = \{(v'_{m-1}, v) \mid (v_{m-1}, v) \in O(v_{m-1})\}.$$

Consider a simple network N in Fig. 2a, where the shortest path $p = (0, 1, 2, 3, 6, t)$ is shown by bold arcs. Applying the algorithm produces the network N' as shown in Fig. 2b. The set of new nodes added in step 1 is $\{1', 2', 3', 6'\}$. The first arc of p, i.e., the arc $(0, 1)$, is deleted and replaced by $(0, 1')$ in step 2. Finally, the set of outgoing arcs added in step 3 is $\{(1', 4), (2', 5), (3', 5), (1', 2'), (2', 3'),$

$(3', 6')\}$. Note that we classify the arc set into two subsets: (1) the subset that emanates from new nodes to old nodes, i.e., $\{(1', 4), (2', 5), (3', 5)\}$, and (2) the subset that emanates from new nodes to new nodes, i.e., $\{(1', 2'), (2', 3'), (3', 6')\}$. As we will see, the latter subset remains present in the algorithm described below. Therefore, it is the former subset that makes the difference between Martins' algorithm [18] and the algorithm of Azevedo et al. [19].

Aimed at improving the performance of Martins' algorithm [18], the path deletion algorithm of Azevedo et al. [19] deletes the *last* arc of p and adds the set of incoming arcs of new nodes. To see why doing so can improve the performance, remember that Martins' algorithm [18] executes the shortest path algorithm K times and the path deletion algorithm $K - 1$ times. By removing the last arc of p and adding incoming arcs of new nodes, the algorithm of Azevedo et al. [19] can save $K - 1$ executions of the shortest path algorithm. The saving derives from the fact that after deleting p from N to generate N', only the labels of those newly added arcs in N' are to be determined. This is owing to the property that the labels of the arcs in N will remain permanent in N'. Moreover, the sub-path of a deleted path p in N will be explicitly determined in N'. (See Azevedo et al. [19] for more details.) Therefore, the worst case of the algorithm of Azevedo et al. [19] reduces to $O(K^2|A|)$. The path deletion algorithm is as follows:

1. The set $\{v'_1, v'_2, \ldots, v'_{m-1}\}$ of new nodes is added to N.
2. The set of incoming arcs of each new node is defined as

$$I(v'_1) = \{(v, v'_1) \mid (v, v_1) \in I(v_1) \text{ and } v \neq v_0\}, \text{ and}$$

$$I(v'_i) = \{v, v'_i\} \mid (v, v_i) \in I(v_i) \text{ and } v \neq v_{i-1}\} \cup \{(v'_{i-1}, v'_i)\}, \text{ for any } i \in \{2, \ldots, m-1\}.$$

3. The set of incoming arcs of node t is updated as

$$I(t) = I(t) - \{(v_{m-1}, t)\} \cup \{(v'_{m-1}, t)\}.$$

The result of applying this algorithm to the network in Fig. 2a is shown in Fig. 2c. The set of incoming arcs of new nodes added by step 2 is $\{(4, 2'), (5, 2'), (5, 6'), (1', 2'), (2', 3'), (3', 6')\}$, where the subset $\{(1', 2'), (2', 3'), (3', 6')\}$ is the same as that of Martins' algorithm [18] as we mentioned earlier. Finally, the last arc of p, i.e., the arc $(6, t)$, is deleted and replaced by $(6', t)$ in step 3. In short, the major differences between two path deletion algorithms lie in: (1) delete first arc versus last arc, (2) add outgoing arcs versus incoming arcs of new nodes.

2.3. The algorithm

Having described the path deletion algorithms for a typical network, we present our path deletion algorithm for the traffic-light network. As mentioned in Azevedo et al. [19], we also assume that a super-initial node 0 and a super-terminal node t with zero arcs $(0, s)$ and (d, t) are added to N_1 to allow the possible repetition of the initial node s and terminal node d. Let the shortest path p be $(0 = v_0, s = v_1, \ldots, v_{m-2}, v_{m-1} = d, v_m = t)$ as defined earlier. To obtain the enlarged network by deleting p, our path deletion algorithm chooses to delete the last arc of p. Recall that each node in a traffic-light network is associated with a set of node-triplets, therefore, our path deletion algorithm needs to add not only nodes and arcs but also the node-triplets of the nodes. Given p and N, the enlarged network N' is obtained by the algorithm below, where notations are:

$V(V')$ is the node set of $N(N')$,
$A(A')$ is the arc set of $N(N')$,
$NT_{v,i}(NT'_{v,i})$ is the set of node-triplets of the ith window of node v in $N(N')$,
v'_j is the added new node in N' corresponding to v_j in N, and
u_w is a node in N or N'.

Network Enlargement Algorithm

```
1.  (Add new nodes)
    Let $V' = V$, $A' = A$, and $NT'_{v,i} = NT_{v,i}$.
    For $j = 1, \ldots, m-1$, add a node $v'_j$ to $V'$.
2.  (Add incoming arcs of $v'_j$ from nodes in $N$, but exclude arcs $(v'_{j-1}, v'_j)$)
    For $j = 1, \ldots, m-1$
      For $w = 1, \ldots, |V|$
        If arc $(u_w, v_j) \in A$ but $u_w \neq v_{j-1}, A' = A' \cup (u_w, v'_j)$; set $t(u_w, v'_j) = t(u_w, v_j)$,
          $NT'_{u_w,i} NT'_{u_w,i} \cup \{\langle x, u_w, v'_j\rangle \mid \langle x, u_w, v_j\rangle \in NT_{u_w,i}$ and $x \in V\}$ for $i = 1$ to $r$.
3.  (Add arcs $(v'_j, v'_{j+1})$)
    For $j = 1, \ldots, m-2$
          $A' = A' \cup (v'_j, v'_{j+1})$, set $t(v'_j, v'_{j+1}) = t(v_j, v_{j+1})$.
          $NT'_{v'_j,i} = \{\langle x, v'_j, v'_{j+1}\rangle \mid x \neq v_{j-1}$ and $x \in V$ and $\langle x, v_j, v_{j+1}\rangle \in NT_{v_j,i}\}$
            $\cup\{\langle v'_{j-1}, v'_j, v'_{j+1}\rangle \mid \langle v_{j-1}, v_j, v_{j+1}\rangle \in NT_{v_j,i}\}$ for $i = 1$ to $r$.
    $NT'_{v'_{m-1},1} = \{\langle x, v'_{m-1}, v'_m\rangle \mid x \in V'\}$ and let the duration of this window be infinite.
4.  Delete arc $(v_{m-1}, t)$ from $A'$ and add arc $(v'_{m-1}, t)$ to $A'$. Set $t(v'_{m-1}, t) = t(v_{m-1}, t)$.
```

Example 1. To illustrate this algorithm, reconsider Fig. 1b where the shortest path is $p_1 = (s, \mathrm{A}, \mathrm{D}, d)$. Initially, the set of new nodes $\{s', \mathrm{A}', \mathrm{D}'$ and $d'\}$ is added to the network N as shown in Fig. 3a.

Beginning step 2, consider j equals 1 (i.e., $v_1 =$ node s). When $w = 1$ in the inner loop, new arc (B, s') is added to the set of incoming arcs of s' because $(\mathrm{B}, s) \in A$ but $\mathrm{B} \neq v_0$ (node 0). In addition, since (B, s') is added to the set of outgoing arcs of node B, the set of node-triplet $NT'_{\mathrm{B},1}$ is changed to $\{\langle s,\mathrm{B},\mathrm{C}\rangle, \langle s,\mathrm{B},\mathrm{D}\rangle, \langle s,\mathrm{B},s\rangle\} \cup \{\langle s,\mathrm{B},s'\rangle\} = \{\langle s,\mathrm{B},\mathrm{C}\rangle, \langle s,\mathrm{B},\mathrm{D}\rangle, \langle s,\mathrm{B},s\rangle, \langle s,\mathrm{B},s'\rangle\}$, and $NT'_{\mathrm{B},2}$ is changed to $\{\langle \mathrm{A},\mathrm{B},\mathrm{C}\rangle, \langle \mathrm{A},\mathrm{B},\mathrm{D}\rangle, \langle \mathrm{A},\mathrm{B},s\rangle\} \cup \{\langle \mathrm{A},\mathrm{B},s'\rangle\} = \{\langle \mathrm{A},\mathrm{B},\mathrm{C}\rangle, \langle \mathrm{A},\mathrm{B},\mathrm{D}\rangle, \langle \mathrm{A},\mathrm{B},s\rangle, \langle \mathrm{A},\mathrm{B},s'\rangle\}$. For brevity, these sets of node-triplets are not shown in the figure. When $j = 2$, no arc is added. When $j = 3$, new arc $(\mathrm{B}, \mathrm{D}')$ is added; $NT'_{\mathrm{B},1}$ is updated to $\{\langle s,\mathrm{B},\mathrm{C}\rangle, \langle s,\mathrm{B},\mathrm{D}\rangle, \langle s,\mathrm{B},s\rangle, \langle s,\mathrm{B},s'\rangle, \langle s,\mathrm{B},\mathrm{D}'\rangle\}$ and $NT'_{\mathrm{B},2}$ to $\{\langle \mathrm{A},\mathrm{B},\mathrm{C}\rangle, \langle \mathrm{A},\mathrm{B},\mathrm{D}\rangle, \langle \mathrm{A},\mathrm{B},s\rangle, \langle \mathrm{A},\mathrm{B},s'\rangle, \langle \mathrm{A},\mathrm{B},\mathrm{D}'\rangle\}$. The result of this step is shown in Fig. 3b.

The task of step 3 is to add the arcs from node v'_j to node v'_{j+1} that was not finished in step 2 because more care should be taken to update the sets of node-triplets. When $j = 1$ in step 3, the new arc to be added is $(v'_1 v'_2)$, which is the arc (s', A'). Since $\mathrm{B} \neq v_0$ and $\langle \mathrm{B}, s, \mathrm{A}\rangle \in NT_{s,1}, NT'_{s',1}$ is newly created as $\{\langle \mathrm{B}, s', \mathrm{A}'\rangle\}$, thus finishes the execution for $j = 1$. For $j = 2$, the new arc to be added is (v'_2, v'_3), which is the arc $(\mathrm{A}', \mathrm{D}')$; $NT'_{\mathrm{A}',1}$ is newly created as $\{\langle s', \mathrm{A}', \mathrm{D}'\rangle\}$. Similarly, when $j = 3$, arc (D', d') is added; $NT'_{\mathrm{D}',1} = \{\langle \mathrm{A}', \mathrm{D}', d'\rangle\}$ and $NT'_{\mathrm{D}',2} = \{\langle \mathrm{B}, \mathrm{D}', d'\rangle\}$. Once the loop is finished, we set $NT'_{d',1} = \{\langle D', d', t\rangle, \langle \mathrm{C}, d', t\rangle\}$ to be infinite and show the result in Fig. 3c.

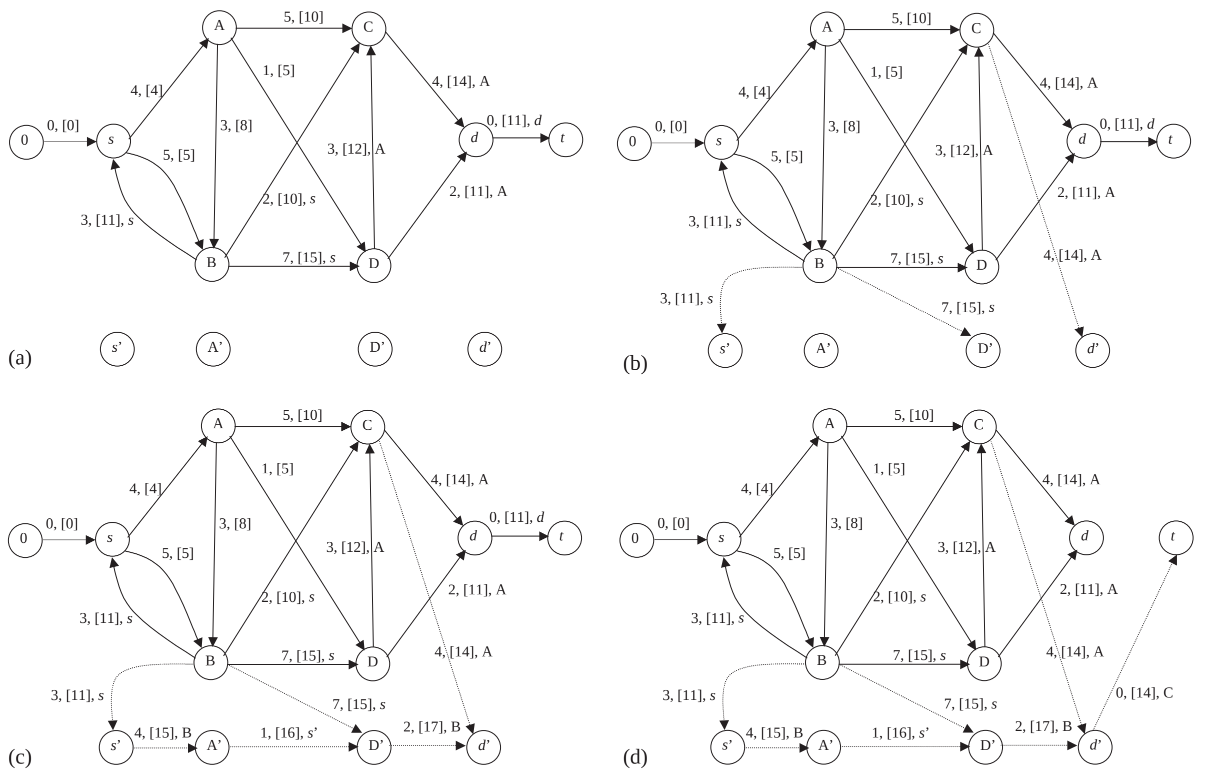

Fig. 3. (a) Add the set of new nodes $\{s', A', D', d'\}$ to the network, (b) add incoming arcs of new nodes (emanating from old nodes), (c) add incoming arcs of new nodes (emanating from new nodes) and (d) replace the last (d, t) by (d', t).

Finally, the arc (d, t) is deleted and the arc (d', t) is added in step 4. Fig. 3d shows the final result and Table 1 summarizes the added nodes and sets of node-triplets after the execution of the algorithm.

To see how the $K - 1$ executions of the shortest path algorithm can be avoided, note that the labels of all arcs in Figs. 1b remain unchanged (i.e., permanent) in Fig. 3d. The property that labels remain permanent arises from the definition of the attached labels *arrived*(u, v), which denotes the earliest time to reach node v through arc (u, v). On the one hand, since all paths in the old network remain present in the augmented network, the labels will not be increased. On the other hand, nor will the labels be decreased because the algorithm does not create any new path to connect the old arcs or nodes. As a result, the labels remain unchanged. To compute the labels of all new arcs, we need to execute two steps: (1) compute the labels of the arcs from the nodes in V to the nodes in $V' - V$, and (2) compute the label of each arc (v'_j, v'_{j+1}) sequentially along the newly-created (we will use *augmented* interchangeably) path. These two steps are described in the following procedure.

```
1.  For every node $u$ in $V$
        For every node $w$ in $V$ and $(w, u)$ in $A$
            For every node $v$ in the augmented path and $(u, v)$ in $A' - A$
                Update $arrived(u, v)$ if it can be reduced by coming from arc $(w, u)$.
```

Table 1
Nodes and node-triplets added after executing Network Enlargement Algorithm

Node	$d_{u,I}$	Original $NT_{u,i}$			Added $NT'_{u,i}$	
s	$d_{s,1}$	$\langle B, s, A\rangle$				
	$d_{s,2}$	$\langle B, s, B\rangle$				
s'	$d_{s',1}$				$\langle B, s', A'\rangle$	
A	$d_{A,1}$	$\langle s, A, D\rangle$				
	$d_{A,2}$	$\langle s, A, B\rangle$	$\langle s, A, C\rangle$			
A′	$d_{A',1}$				$\langle s', A', D'\rangle$	
B	$d_{B,1}$	$\langle s, B, C\rangle$	$\langle s, B, D\rangle$	$\langle s, B, s\rangle$	$\langle s, B, D'\rangle$	$\langle s, B, s'\rangle$
	$d_{B,2}$	$\langle A, B, C\rangle$	$\langle A, B, D\rangle$	$\langle A, B, s\rangle$	$\langle A, B, D'\rangle$	$\langle A, B, s'\rangle$
C	$d_{C,1}$	$\langle A, C, d\rangle$			$\langle A, C, d'\rangle$	
	$d_{C,2}$	$\langle B, C, d\rangle$	$\langle D, C, d\rangle$		$\langle B, C, d'\rangle$	$\langle D, C, d'\rangle$
D	$d_{D,1}$	$\langle A, D, C\rangle$	$\langle A, D, d\rangle$			
	$d_{D,2}$	$\langle B, D, C\rangle$	$\langle B, D, d\rangle$			
D′	$d_{D',1}$				$\langle A', D', d'\rangle$	
	$d_{D',2}$				$\langle B, D', d'\rangle$	

```
2.   For every arc $(v'_j, v'_{j+1})$ in the augmented path
         For every node $w$ in $V \cup \{v'_{j-1}\}$ and $(w, v'_j)$ in $A' - A$
             Update $arrived(v'_j, v'_{j+1})$ if it can be reduced by coming from arc $(w, v'_j)$.
```

To illustrate these two steps, reconsider Figs. 3b and c. In step 1, we will compute the labels of the arcs from the nodes in V to the nodes in $V' - V$, including *arrived*(B, s'), *arrived*(B, D′) and *arrived*(C, d'). Take *arrived*(C, d'), which is 14 in Fig. 3b, for example. Since arc (C, d') can be reached through arc (A, C) or arc (B, C) or arc (D, C), we need to know which one arrives earlier. If through arc (A, C), the arrival time as 14; if through (B, C), the time is $11 + 4 = 15$; if through (D, C), the time is $12 + 4 = 16$. The result indicates that reaching arc (C, d') by coming from arc (A, C) will give the minimum value of *arrived*(C, d'). As for step 2, recall that we need to compute the label of each arc (v'_j, v'_{j+1}) along the augmented path, i.e., *arrived*(s, A′), *arrived*(A′, D′) and *arrived*(D′, d'). Consider *arrived*(D′, d') = 17 in Fig. 3c. Since arc (D′, d') can be reached through either arc (A′, D′) or arc (B, D′), we need to compute their arrival times. The arrival time of coming through arc (B, D′) is $15 + 2 = 17$, but through arc (A′, D′) is $16 + 2 = 18$. Therefore, the value of *arrived*(D′, d') is 17 that goes through arc (A′, D′).

Only the subset of arcs is scanned because the algorithm does not completely scan each arc using this two-step procedure. We use the following lemmas to analyze the time complexity of the network enlargement algorithm and label update operation.

Lemma 1. *Every newly added node has at most $|V_1|$ adjacent nodes with arcs going into it, i.e., no matter how many number of iterations have been done, the in-degree of a node is bounded from above by this value, where $|V_1|$ is the number of nodes of network N_1.*

Proof. If arc (w, u) is on the shortest path, we will create node u' and arcs $\{(x, u') \mid (x, u) \text{ in } A\} \cup \{(w', u')\}$. So, the in-degree of node u' is the same as that of node u. □

Lemma 2. *The time complexity of executing Network Enlargement Algorithm for the kth iteration is $O(rk|V_1|^3)$, where r is the number of different windows of a node and $|V_1|$ is the number of nodes of network N_1.*

Proof. Since the most time-consuming part is in step 2, we analyze only the complexity of this part and omit the other steps for clarity. Step 2 iterates at most m times, where m is the number of nodes of the augmented path in the kth network that may contain as many as $k|V_1|$ nodes in the worst case. Because each node on the path contains at most $|V_1|$ adjacent nodes with arcs entering it, there are at most $k|V_1|^2$ newly added arcs (u_w, v'_j). During the iterations, we examine each arc (u_w, v_j) exactly once to check whether (u_w, v_j) exists. If it does, we add arc, travel time, and update node-triplets. Among them, each operation of adding arc and travel time requires $O(1)$ time. As for updating node-triplets, note that for a certain v_j in the outer loop, at most $|V_1|$ node-triplets will be present for the node u_w in the inner loop due to Lemma 1. Since a node u_w contains at most r windows, the time of the inner loop is $O(r|V_1|)$. Therefore, the time of step 2 is $O(rk|V_1|^3)$, and so does the whole algorithm. □

Lemma 3. *The time complexity to update labels in the kth iteration is $O(\log rk|V_1|^3)$, where r is the number of different windows of a node and $|V_1|$ is the number of nodes of network N_1.*

Proof. In step 1, node v is a node appearing in the augmented path. Each node v contains at most $O(|V_1|)$ incoming arcs like (u, v) by Lemma 1; in turn, each node u contains at most $O(|V_1|)$ incoming arcs like (w, u). Since the length of the augmented path is at most $O(k|V_1|)$, step 1 needs $O(k|V_1|^3)$ times to compute *arrived*(u, v). As for step 2, we need $O(|V_1|k)$ times to compute *arrived*(v'_j, v'_{j+1}), for all of the arcs in the augmented path must be computed in order. Since each computation of *arrived*(u, v) can be done in time of $O(\log r)$ by Lemma 1 of Chen and Yang [9], the total time required to update the labels is $O(\log rk|V_1|^3)$. □

With the algorithms to enlarge network and update labels in place, the algorithm to find the first K shortest looping paths is as follows.

K Shortest Looping Algorithm

```
1. Find $p_1$ by using the algorithm of Chen and Yang [9].
2. For $k = 2$ to $K$
       Use Network Enlarge Algorithm.
       Update labels of added nodes and arcs.
```

Lemma 4. *The time complexity of the K Shortest Looping Algorithm is $O(rK^2|V_1|^3)$, where r is the number of different windows of a node and $|V_1|$ is the number of nodes of network N_1.*

Proof. Step 1 can be done in time of $O(r|V_1|^3)$. Step 2 iterates $K - 1$ times and each iteration consists of two parts, namely, enlarging network and updating labels. Since each enlarging network can be done in $O(rk|V_1|^3)$ as shown in Lemma 2, the total time for $K - 1$ iterations is $O(rK^2|V_1|^3)$. To update labels, by Lemma 3 it can be done in time $O(\log rk|V_1|^3)$, and hence the total time for $K - 1$ iterations is $O(\log rK^2|V_1|^3)$. As a result, the total time of the algorithm is $O(rK^2|V_1|^3)$. □

3. Conclusions

In this paper, we develop an efficient algorithm for enumerating the first K shortest looping paths in a traffic-light network. Because the complexity of the proposed algorithm is shown to be polynomial, the algorithm can be applied to solve problems in reasonable time. One possible extension of the research is to consider the situation where one can choose to wait for some time on a node and leave later. Dealing with this issue would be challenging because of enumeration of all possible paths resulting from the varying length of time to wait.

Acknowledgements

The authors are grateful to the anonymous referees for their helpful comments. The first author was supported in part by National Science Foundation grant no. 91-2416-H-167-004.

References

[1] Swersey AJ, Ballard W. Scheduling school buses. Management Science 1984;30:844–53.
[2] Chen YL, Chin YH. The quickest path problem. Computers & Operations Research 1990;17:153–61.
[3] Bodin LD, Golden BL, Assad AA, Ball MO. Routing and scheduling of vehicles and crews: the state of the art. Computers & Operations Research 1982;10:63–211.
[4] Deo N, Pang C. Shortest path algorithms: taxonomy and annotation. Networks 1984;14:275–323.
[5] Golden BL, Magnanti TL. Deterministic network optimization: a bibliography. Networks 1977;7:149–83.
[6] Desrochers M, Desrosiers J, Solomon MM. A new optimization algorithm for the vehicle routing problem with time windows. Operations Research 1992;40:342–54.
[7] Dumas Y, Desrosiers J, Gelinas E, Solomon MM. An optimal algorithm for the traveling salesman problem with time windows. Operations Research 1995;43:367–71.
[8] Kohl N, Madsen OBG. An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation. Operations Research 1997;45:395–406.
[9] Chen YL, Yang HH. Shortest paths in traffic-light networks. Transportation Research: Part B 2000;34:241–53.
[10] Chen YL, Yang HH. Minimization of travel time and weighted number of stops in a traffic-light network. European Journal of Operational Research 2003;144(3):565–80.
[11] Eppstein D. Finding the k shortest paths. SIAM Journal on Computing 1998;28:652–73.
[12] Yang HH, Chen YL. The first k shortest unique-arc walks in a traffic-light network. Mathematical and Computer Modelling, 2003, submitted for publication.
[13] Yen JY. Finding the k shortest loopless paths in a network. Management Science 1971;17:712–6.
[14] Katoh N, Ibaraki T, Mine H. An efficient algorithm for k shortest simple paths. Networks 1982;12:411–27.
[15] Hadjiconstantinou E, Christofides N. An efficient implementation of an algorithm for finding k shortest simple paths. Networks 1999;34:88–101.
[16] Dreyfus S. An appraisal of some shortest path algorithms. Operations Research 1969;17:395–412.
[17] Fox BL. Data structures and computer science techniques in operations research. Operations Research 1978;26: 686–717.
[18] Martins EQV. An algorithm for ranking paths that may contain cycles. European Journal of Operational Research 1984;18:123–30.
[19] Azevedo JA, Santos Costa MEO, Silvestre Madeira JJER, Martins EQV. An algorithm for the ranking of shortest paths. European Journal of Operational Research 1993;69:97–106.