ELSEVIER

European Journal of Operational Research 144 (2003) 565–580

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Discrete Optimization

Minimization of travel time and weighted number of stops in a traffic-light network

Yen-Liang Chen [a,*], Hsu-Hao Yang [b,1]

[a] *Department of Information Management, National Central University, Chung-Li 320, Taiwan, ROC*
[b] *Department of Industrial Management, National Chinyi Institute of Technology, Taiping 411, Taiwan, ROC*

Received 3 April 2000; accepted 11 December 2001

Abstract

The time-constrained shortest path problem is an important generalization of the shortest path problem. Recently, a model called traffic-light control model was introduced by Chen and Yang [Transport. Res. B 34 (2000) 241] to simulate the operations of traffic-light control in a city. The constraints of the model consist of a repeated sequence of time windows, and each window allows only certain routes to pass through a node. In this paper, we introduce a new kind of network called on–off time-switch network in which an arc is associated with a sequence of windows with status "on" or "off" analogous to "go" or "wait". We show that both networks have the same mathematical structure in the sense that a path in one network corresponds to a path in the other one. Since Chen and Yang have developed algorithms to find the minimum total time path in the previous paper, we include one more criterion in this paper: weighted number of stops. To solve this bi-criteria path problem, we transform the traffic-light network into the on–off time-switch network, which allows us to take advantages of the special structure to design more efficient algorithms. By this transformation, finding the bi-criteria shortest path in the traffic-light network can be done in time $O(\#Wn^3)$, where n is the number of nodes and $\#W$ is a given maximum number of weighted stops.

Keywords: Shortest path; Traffic light; Time window; Road network

1. Introduction

Studying network problems subject to time constraints is increasingly gaining popularity, especially in a variety of transportation applications. Some examples include shortest path problem (Desrochers and Soumis, 1988; Chen and Tang, 1997, 1998), traveling salesman problem (Baker, 1983; Dumas et al., 1995), vehicle routing problem (Baker, 1982; Kolen et al., 1987; Balakrishnan, 1993; Russell, 1995; Bramel and

* Corresponding author. Tel.: +886-3-4267266; fax: +886-3-425-4604.
E-mail addresses: ylchen@im.mgt.ncu.edu.tw (Y.-L. Chen), yanghh@chinyi.ncit.edu.tw (H.-H. Yang).
[1] Tel.: +886-4-3924505x7616; fax: +886-4-3934620.

0377-2217/03/$ - see front matter
PII: S0377-2217(02)00148-0

Simchilevi, 1996), traffic network (Fu and Rilett, 1998), and pickup and delivery problem (Dumas et al., 1991). Among them, shortest path problem and vehicle routing problem are probably the most widely studied ones.

Basically, the shortest path problem is concerned with finding the path with minimum distance, time, or cost from an origin to a destination through a connected network. It is a classical and important problem in the area of combinatorial optimization because of its numerous applications. Readers are referred to Bodin et al. (1982), Deo and Pang (1984), Ahuja et al. (1993) and Golden and Magnanti (1977) for more comprehensive discussions of these issues.

In the presence of time constraints, the shortest path problem needs to consider when a node in the network can be visited in search of a solution. Time window has been a common form of time constraints that requires that a node can be visited only in a specified time interval (Baker, 1982; Kolen et al., 1987; Desrochers and Soumis, 1988; Balakrishnan, 1993; Russell, 1995; Bramel and Simchilevi, 1996). Two kinds of time windows appear commonly. The first one is the *hard* time window where solution is infeasible if we cannot visit the node during the window period (Kolen et al., 1987; Russell, 1995; Bramel and Simchilevi, 1996). The other one is the *soft* time window where a cost penalty is incurred if we visit the node outside its time window (Balakrishnan, 1993). In addition, if the time windows degenerate into time points, which is a discrete version of the time window constraint, we call it *time schedule* constraint (Chen and Tang, 1997, 1998, 2001). This kind of constraint assumes that each node has a list of pre-specified departure times and requires that departure from a node can take place only at one of these departure times.

Although the time-constrained problems have been studied extensively, one member of this family received surprisingly little attention and even seemed to have been ignored. That is, finding the shortest path of a city with traffic-light control in a number of crossroads. Consider Fig. 1(a) that shows a sample crossroad. Suppose the crossroad has a light control that is a repeated sequence of four different windows. Fig. 1(b) (or (c)) indicates the allowable routes in the first (or the second) window. The third and fourth windows in essence resemble the first and second ones but slightly differ in orientations: the former is between north and south, while the latter is between east and west.

To model the problem in Fig. 1, one may consider the soft time window to be suitable since it specifies the time interval to pass through the crossroad. However, we cannot apply the single soft time window because it does not consider the orientations nor contain a repeated sequence of different time windows. To solve this problem, Chen and Yang (2000) proposed a new kind of network, called traffic-light network, to formulate a modern city subject to traffic-light control constraints and developed a polynomial algorithm for finding the shortest path. In this paper, we will consider one more criterion that occurs frequently in practice, namely, the number of stops in a path. To justify this new criterion, note that more stops often introduce uncertainty and cause tour scheduling to be less manageable. Furthermore, in terms of transportation alone, more stops are less cost-effective. Finally, excessive number of stops may make travelers frustrated.

In this paper, we introduce a new kind of network, called *on–off time-switch* network, and show that both networks have the same mathematical structure. That is, if a path appears in one network, there exists its counterpart in the other one. Moreover, both paths have the same total time and the same number of stops. Because of this property, an optimum path in the traffic-light network can be found by solving its counterpart in the on–off time-switch network. The transformation allows us to develop the efficient algorithm that is easier based on the on–off time-switch network. In consequence, our solution procedure contains two major parts. First, we transform the original network into the corresponding on–off time-switch network. Second, we solve the bi-criteria path problem in the on–off time-switch network.

This paper is organized as follows. In Section 2, we define the traffic-light network and the on–off time-switch network. In Section 3, we propose a method for transforming a traffic-light network into an on–off time-switch network. In Section 4, we study a bi-criteria shortest problem in an on–off time-switch network. The criteria considered are the total time and the weighted number of stops. We use the weighted number of

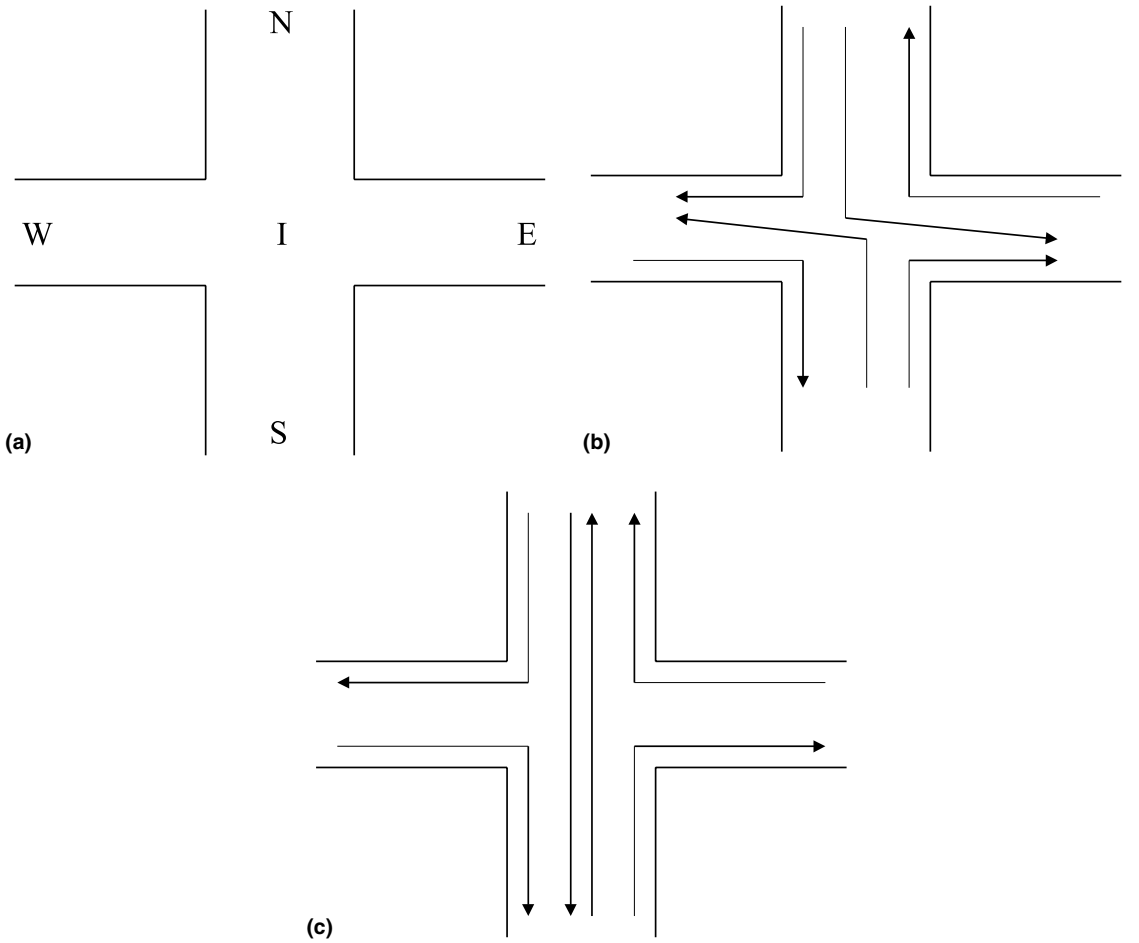

Fig. 1. (a) A sample crossroad. (b) The allowable routes in the first window. (c) The allowable routes in the second window.

stops instead of number of stops as the second criterion to reflect the importance of the road. Finally, Section 5 contains the conclusion, limitation of the paper and future research directions. We provide the proofs of lemmas and theorems in Appendix A.

2. Definitions of networks

We let $N = (V, A, t, s, d)$ denote a traffic-light network, where V is the node set of crossroads, A is the arc set of roads in the city, $t(u, v)$ is the travel time from node u to node v, s is the source node and d is the destination node. A path is said to be *efficient* if no other paths with the same or fewer number of stops but a smaller total time exist in the network. Our goal is to find all efficient paths from node s to node d in N where some nodes are subject to the traffic-light control. By the definition of the efficient path, we have at most one efficient path for a given number of stops. Since the maximum number of stops in a path is limited in practice, processing the set of efficient paths can be done comfortably. Therefore, after finding all the efficient paths, we can select the best one that reflects our preferences. For example, we can assign different weight to each criterion, and then choose the one with minimum weighted sum.

Assume that $V = V_1 \cup V_2$ and $\{s, d\} \in V_1$, where V_1 is the node set without window restriction and V_2 is the node set with window restriction. For notational purpose, let each node $u \in V_2$ have r time windows $w_{u,1}, w_{u,2}, \ldots, w_{u,r}$. Since these windows form a repeated sequence, we assume that $w_{u,0} = w_{u,r}, w_{u,(k\times r)+i} = w_{u,i}$ for any nonnegative integers k and i, and use $d_{u,i}$ to specify the length of window $w_{u,i}$. Further, we associate the window $w_{u,i}$ with a set of node-pairs $NP_{u,i}$. A node-pair $\langle x, y\rangle$ in $NP_{u,i}$ denotes the ith window of node u to visit node y from node x, namely, $NP_{u,i}$ is the set of allowable routes in the ith time window of node u. Consider Fig. 2 that is the network representation of Fig. 1(a). For node I, we attach four time windows $w_{I,1}, w_{I,2}, w_{I,3}$ and $w_{I,4}$. Window $w_{I,1}$ has a set of node-pairs $NP_{I,1} = \{\langle \mathrm{N}, \mathrm{W}\rangle, \langle \mathrm{N}, \mathrm{E}\rangle, \langle \mathrm{S}, \mathrm{W}\rangle, \langle \mathrm{S}, \mathrm{E}\rangle, \langle \mathrm{W}, \mathrm{S}\rangle, \langle \mathrm{E}, \mathrm{N}\rangle\}$, $\mathrm{NP}_{I,2}$ of $w_{I,2} = \{\langle \mathrm{N}, \mathrm{W}\rangle, \langle \mathrm{N}, \mathrm{S}\rangle, \langle \mathrm{S}, \mathrm{N}\rangle, \langle \mathrm{S}, \mathrm{E}\rangle, \langle \mathrm{W}, \mathrm{S}\rangle, \langle \mathrm{E}, \mathrm{N}\rangle\}$, $NP_{I,3}$ of $w_{I,3} = \{\langle \mathrm{W}, \mathrm{S}\rangle, \langle \mathrm{W}, \mathrm{N}\rangle, \langle \mathrm{E}, \mathrm{S}\rangle, \langle \mathrm{E}, \mathrm{N}\rangle, \langle \mathrm{S}, \mathrm{E}\rangle, \langle \mathrm{N}, \mathrm{W}\rangle\}$, and $NP_{I,4}$ of $w_{I,4} = \{\langle \mathrm{W}, \mathrm{S}\rangle, \langle \mathrm{W}, \mathrm{E}\rangle, \langle \mathrm{E}, \mathrm{W}\rangle, \langle \mathrm{E}, \mathrm{N}\rangle, \langle \mathrm{S}, \mathrm{E}\rangle, \langle \mathrm{N}, \mathrm{W}\rangle\}$. Note that the node-pair set $NP_{I,1}$ (or $NP_{I,2}$) contains all the routes in Fig. 1(b) (or Fig. 1(c)). As shown, the network models the light control of a city. By expressing each route as an entering arc (x, u) plus a leaving arc (u, y), we can group all these allowable routes, denoted by $\langle x, y\rangle$, together to form a node-pair set of the corresponding time window. Therefore, the problem of how to find the quickest path to pass through a number of traffic-light controls can be answered by solving the shortest path problem in the present network.

In contrast, we associate a repeated sequence of time windows to an arc rather than a node in an on–off time-switch network. The windows have no orientations, which means we can leave for the next node if we are in "on" windows. To denote the beginning time of the operation sequence, we use an offset in the time-switch. Fig. 3 shows such an example. Suppose the length of "on" is 5 units of time, the length of "off" is 4, and the offset is at time 1. Further suppose the operation sequence is "on" followed by "off", then again followed by "on", and so on. If we reach the arc at 7, then the earliest leaving time from this node is 10. On the contrary, if we reach the arc at 4, we can leave immediately.

Let $N = (V, A, t, TL, s, d)$ be an on–off time-switch network, where $G = (V, A)$ is a directed graph without multiple arcs and self-loops, V and A are the sets of all the nodes and arcs in the network, $t(u, v)$ is the travel time of arc (u, v), s and d are the source and destination nodes. Each arc (u, v) in the network is associated

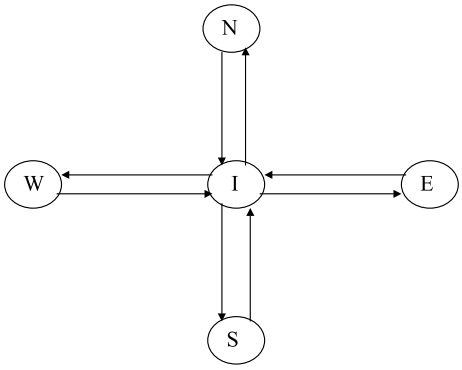

Fig. 2. The network representation of Fig. 1(a).

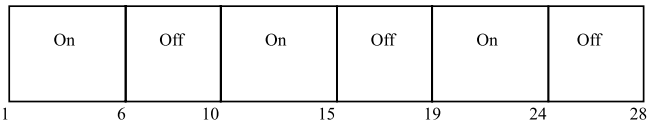

Fig. 3. An example of on–off time-switch.

with a time-list $TL(u,v) = (o, t_1, t_2, \ldots, t_r)$, where o is the offset and t_i is the length of the ith time window. The offset o allows different arcs to start at different times. Without loss of generality, we assume that

(1) r is an even number and
(2) t_i is an "on" (or "off") window if i is an odd (or even) number.

From $TL(u,v)$, we can derive the sequence of window times as $TS(u,v) = (ts_{g(1)}(u,v), ts_{w(1)}(u,v), ts_{g(2)}(u,v), ts_{w(2)}(u,v), \ldots)$ by using the following expression, where $ts_{g(j)}(u,v)$ (or $ts_{w(j)}(u,v)$) is the starting time of the jth "on" (or "off") window:

$$T_0 = 0 \text{ and } T_i = T_{i-1} + t_i = \sum_{j=1}^{i} t_j \quad \text{for } i = 1 \text{ to } r,$$

$$ts_{g(1)}(u,v) = o,$$

$$ts_{g(j)}(u,v) = o + \lfloor (2j-2)/r \rfloor \times T_r + T_{2j-2-\lfloor (2j-2)/r \rfloor \times r} \quad \text{for } j > 1,$$

$$ts_{w(j)}(u,v) = o + \lfloor (2j-1)/r \rfloor \times T_r + T_{2j-1-\lfloor (2j-1)/r \rfloor \times r} \quad \text{for } j \geqslant 1.$$

For example, if $TL(u,v) = (2,5,4,1,2)$, then $TS(u,v) = (2,7,11,12,14,19,23,24,26,\ldots)$, i.e., $ts_{g(1)}(u,v) = 2$, $ts_{w(1)}(u,v) = 7$, $ts_{g(2)}(u,v) = 11$, $ts_{w(2)}(u,v) = 12$, and so on.

3. Transforming traffic-light networks into on–off time-switch networks

In this section, we will show that these two networks are equivalent in the sense that a path appearing in one network always corresponds to a path in the other one with the same number of stops and total time. Let $N = (V_1 \cup V_2, A, t, s, d)$ be a traffic-light network. For each node $u \in V_2$, an offset wo_u and r time windows $w_{u,1}, w_{u,2}, \ldots, w_{u,r}$ of the lengths $d_{u,1}, d_{u,2}, \ldots, d_{u,r}$ are attached. Recall a node-pair $\langle x, y \rangle$ in $NP_{u,i}$ denotes the ith window of node u to reach node y from node x. To transform, we need to do the following:

1. Create the arcs and nodes in the on–off time-switch network. For paths in each network, a one-to-one correspondence exists.
2. Attach a time-switch to each arc in the on–off time-switch network. The attachment ensures that the total time and number of stops of a path in one network equal those in the other.

Creating the on–off time-switch network contains four steps (refer to Fig. 4).

1. For arc (u,v), where $u \in V_1$ and $v \in V_1$, create arc (u,v) with the same travel time.
2. For arc (u,v), where $u \in V_1$ and $v \in V_2$, create arc $(u, {}^{u}v)$ with the same travel time.
3. For arc (u,v), where $u \in V_2$ and $v \in V_1$, create arc $({}^{x}u, v)$ with the same travel time for all nodes x having arc (x,u).
4. For arc (u,v), where $u \in V_2$ and $v \in V_2$, create arc $({}^{x}u, {}^{u}v)$ with the same travel time for all nodes x having arc (x,u).

To see how this transformation works, consider Fig. 6(a), where $\{s,d\} \subseteq V_1, \{A,B,C,D\} \subseteq V_2$ and the number along each arc is the arc's travel time. By the preceding transformation, we create Fig. 6(b) (we temporarily ignore the time-switches associated with arcs). For every path in Fig. 6(a), say path

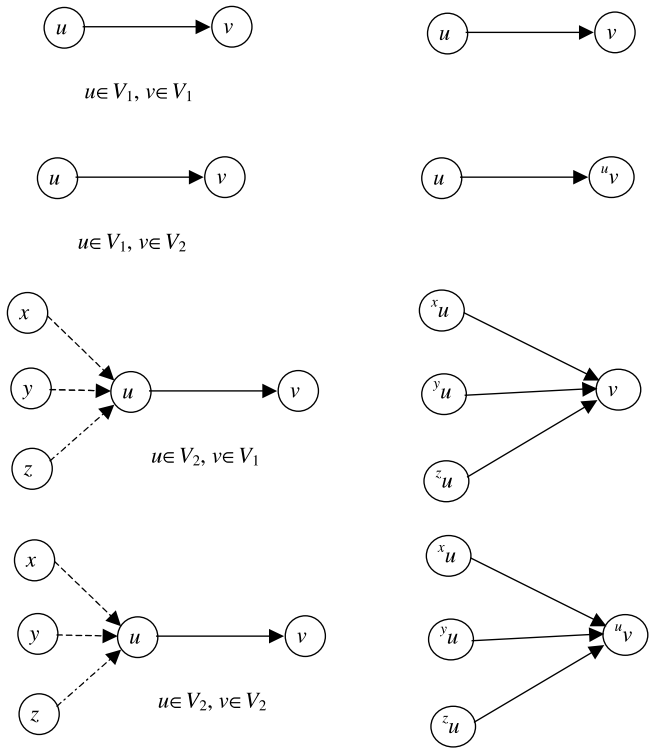

Fig. 4. Create the on–off time-switch network.

(s, A, C, D, d), we can find a corresponding path $(s, {}^{s}A, {}^{A}C, {}^{C}D, d)$ in Fig. 6(b). On the other hand, for every path in Fig. 6(b), say $(s, {}^{s}C, {}^{C}B, {}^{B}D, d)$, we can also find a corresponding path (s, C, B, D, d) in Fig. 6(a).

Next, we will consider the problem of how to construct the time-switches for the arcs in the on–off time-switch network. Consider Fig. 5 where we construct the time-switches for node C in V_2. Suppose node C has five different windows and two of them contain the node-pair $\langle B, D\rangle$. If windows $w_{C,1}$ and $w_{C,2}$ contain this pair, then arc $({}^{B}C, {}^{C}D)$ can go in $[w_{C,1}, w_{C,2}]$ but must wait in $[w_{C,3}, w_{C,4}, w_{C,5}]$. Viewing $[w_{C,1}, w_{C,2}]$ as the first window and $[w_{C,3}, w_{C,4}, w_{C,5}]$ the second one, we can pass arc $({}^{B}C, {}^{C}D)$ in the first but must wait in the second. This situation is shown on the top of Fig. 5. In addition, we may have three other cases as follows:

1. Assume windows $w_{C,2}$ and $w_{C,3}$ contain $\langle B, D\rangle$. This means that arc $({}^{B}C, {}^{C}D)$ can go in $[w_{C,2}, w_{C,3}]$, while must wait in $[w_{C,1}]$ and $[w_{C,4}, w_{C,5}]$. Recall that the number of windows must be even. To do that, we add an "on" window of zero preceding $[w_{C,1}]$. As a result, the arc $({}^{B}C, {}^{C}D)$ has four windows, i.e., $[0]$, $[w_{C,1}]$, $[w_{C,2}, w_{C,3}]$ and $[w_{C,4}, w_{C,5}]$.
2. Assume windows $w_{C,1}$ and $w_{C,5}$ contain $\langle B, D\rangle$. Similarly, we add an "off" window of zero following $[w_{C,5}]$. The windows are thus: $[w_{C,1}]$, $[w_{C,2}, w_{C,3}, w_{C,4}]$, $[w_{C,5}]$ and $[0]$.
3. Assume windows $w_{C,2}$ and $w_{C,5}$ contain $\langle B, D\rangle$. In this case, $[w_{C,1}]$, $[w_{C,2}]$, $[w_{C,3}, w_{C,4}]$ and $[w_{C,5}]$ happen to be in reverse order of on–off. We add an "on" window of zero preceding $[w_{C,1}]$ and an "off" window of zero following $[w_{C,5}]$. Then, the windows are: $[0]$, $[w_{C,1}]$, $[w_{C,2}]$, $[w_{C,3}, w_{C,4}]$, $[w_{C,5}]$ and $[0]$.

Finally, if the node considered is in V_1, we attach a time-switch $(0, \infty, 0)$ to all the arcs emanating from the node.

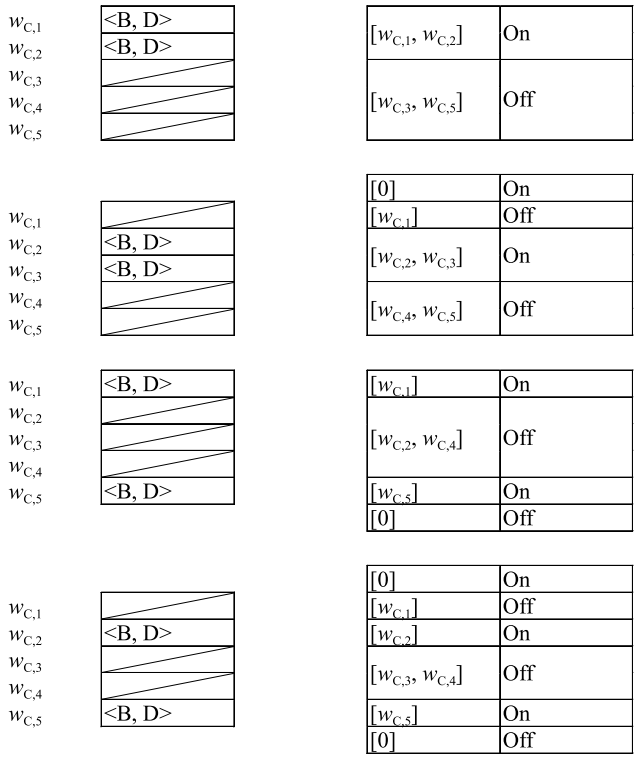

Fig. 5. Create the on–off time-switch for a node in V_2.

Applying the transformation to the network in Fig. 6(a), we obtain the arcs attached with time-switches as shown in Fig. 6(b). For example, arc $({}^{A}C, {}^{C}D)$ has a time list $(4, 2, 5)$ because $\langle A, D\rangle$ is allowed in window $w_{C,1}$ but prohibited in window $w_{C,2}$. Note that $({}^{s}C, {}^{C}D)$ has a time list $(4, 0, 2, 5, 0)$ because $\langle s, D\rangle$ is allowed in window $w_{C,2}$ but prohibited in window $w_{C,1}$. As the second example, $({}^{B}D, d)$ has a time list $(2, 2, 6)$ because $\langle B, d\rangle$ is allowed in window $w_{D,1}$ but prohibited in windows $w_{D,2}$ and $w_{D,3}$. On the contrary, $({}^{C}D, d)$ has a time list $(2, 0, 2, 6, 0)$ because $\langle C, d\rangle$ is prohibited in window $w_{D,1}$ but allowed in windows $w_{D,2}$ and $w_{D,3}$.

To see why the path in one network is equivalent to that in the other one, consider (s, A, C, d) in Fig. 6(a) and $(s, {}^{s}A, {}^{A}C, d)$ in Fig. 6(b). Table 1 summarizes the traveling sequence of these paths and shows that both paths have the same total time and the same number of stops.

The algorithm below constructs the time-switches for the arcs in the present network. For ease of presentation, we number all the nodes in $V_1 \cup V_2$ from 1 to n, where $n = |V_1 \cup V_2|$. Given $w_{u,1}$, $w_{u,2}, \ldots, w_{u,r}$ and their $NP_{u,1}, NP_{u,2}, \ldots, NP_{u,r}$, the following algorithm builds on–off time-switch for a node u in V_2.

Algorithm 1 (*Build-time-switch* (u))

1. For $k = 1$ to r
   For each node-pair $\langle x, y\rangle$ in $NP_{u,k}$ do
   Insert $w_{u,k}$ into the end of the queue associated with arc $({}^{x}u, {}^{u}y)$.
2. For $x = 1$ to n
   For $y = 1$ to n

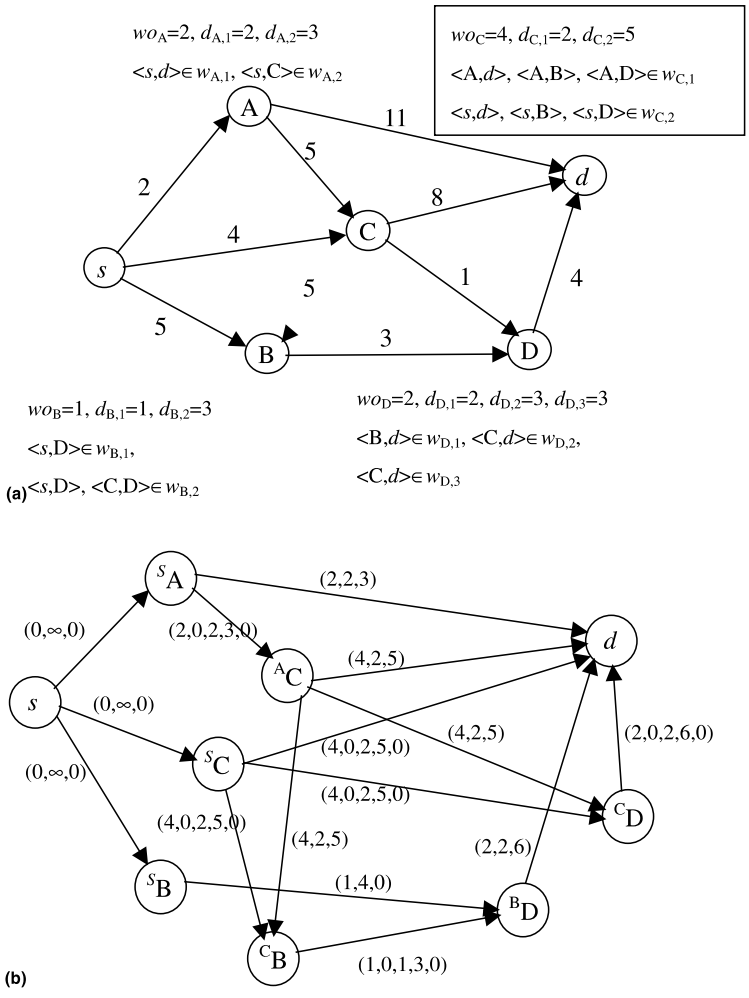

Fig. 6. (a) The original traffic-light network. (b) The constructed on–off time-switch network.

Examine the queue attached with arc $({}^{x}u, {}^{u}y)$ by the following:

2.1. If the queue is empty then delete arc $({}^{x}u, {}^{u}y)$ and exit.

2.2. Examine the first element in the queue, say $w_{u,b}$.
If $w_{u,b} \neq w_{u,1}$
then output an "on" window [0]
output an "off" window $[w_{u,1}, \ldots, w_{u,b-1}]$.

2.3. Find the continuous elements in the queue, say $w_{u,b}, w_{u,b+1}, w_{u,b+2}, \ldots, w_{u,b+z}$.
Output an "on" window $[w_{u,b}, w_{u,b+1}, w_{u,b+2}, \ldots, w_{u,b+z}]$.

2.4. If the queue is empty and $w_{u,b+z} \neq w_{u,r}$
then output an "off" window $[w_{u,b+z+1}, w_{u,b+z+2}, \ldots, w_{u,r}]$ and exit.
If the queue is empty and $w_{u,b+z} = w_{u,r}$
then output an "off" window [0] and exit.
Let the next element in the queue be $w_{u,e}$ (note that $e \neq b + z + 1$).

Table 1
Traveling sequence of paths in two networks

Time	Path														
	s	$\rightarrow$	A	$\rightarrow$	C	$\rightarrow$	d	s	$\rightarrow$	${}^{s}A$	$\rightarrow$	${}^{A}C$	$\rightarrow$	d
0	Leave							Leave						
2			Arrive							Arrive				
4			Leave							Leave				
9					Arrive							Arrive		
11					Leave							Leave		
19							Arrive							Arrive

Output an "off" window $[w_{u,b+z+1}, w_{u,b+z+2}, w_{u,b+z+3}, \ldots, w_{u,e-1}]$.
Let $w_{u,b} \leftarrow w_{u,e}$; go to step 2.3.

To analyze the time complexity of Algorithm 1, we assume that the maximum number of windows (i.e., r) in a single iteration of the repeated window sequence is a constant.

Lemma 1. *The time complexity of Algorithm* 1 *is* $O(n^2)$, *where n is the number of nodes in the network.*

In sum, we can transform a traffic-light network into the corresponding on–off time-switch network by the following algorithm.

Algorithm 2 (*Transformation*)
1. For arc (u, v), where $u \in V_1$ and $v \in V_1$, create arc (u, v) with the same travel time.
2. For arc (u, v), where $u \in V_1$ and $v \in V_2$, create arc $(u, {}^{u}v)$ with the same travel time.
3. For arc (u, v), where $u \in V_2$ and $v \in V_1$, create arc $({}^{x}u, v)$ with the same travel time for all nodes x having arc (x, u).
4. For arc (u, v), where $u \in V_2$ and $v \in V_2$, create arc $({}^{x}u, {}^{u}v)$ with the same travel time for all nodes x having arc (x, u).
5. For every node u in V_1, attach $(0, \infty, 0)$ to all the arcs emanating from node u.
6. Use Algorithm 1 to attach windows to every node u in V_2.

With Lemma 1, we can obtain the time complexity to transform and the size of the transformed network as follows.

Lemma 2. *The time of Algorithm* 2 *is* $O(n^3)$, *and the transformed network has* $O(n^2)$ *nodes and* $O(n^3)$ *arcs, where n is the number of nodes in the network.*

4. Finding bi-criteria shortest paths in a traffic-light network

The result of previous section shows that our problem reduces to finding a bi-criteria shortest path in an on–off time-switch network. Since there are two objectives in our model, we may have a number of different solutions depending on how we define our decision scenario. Hence, instead of defining what the optimal path is, we choose to enumerate all efficient paths for the following two reasons:

(1) We know that the solution of a reasonable decision scenario must be in the efficient path set, since an inefficient path is dominated by at least one efficient path.

(2) The size of the efficient path set is often small. This implies that choosing the optimal path from the efficient path set is not a great concern.

As described, an arc in an on–off time-switch network is analogous to transit an intersection of roads. For example, the arc $({}^{A}C, d)$ denotes to transit from node A to node d via node C. In practice, the importance of transition may vary from one to the other depending on whether the road is major or minor. To reflect this factor, we attach each arc (u, v) with a nonnegative integer, $weight(u, v)$, to denote its relative importance. Because of this reflection, we will change to use the weighted number of stops as the second criterion. For example, let (s, A, B, C, d) be a path and we stop on arcs (A, B) and (B, C). If $weight(A, B) = 1$ and $weight(B, C) = 2$, the weighted number of stops of path (s, A, B, C, d) is $1 + 2 = 3$.

Let $N = (V, A, t, TL, s, d)$ be as defined above, and let $\#w$ denote the weighted number of stops. Our goal is to find all efficient paths from node s to node d for $0 \leqslant \#w \leqslant \#W$, where $\#W$ is a given constant. Let the total time of a path to the node u denote the time to arrive at node u. Because we may wait for a while before we start traveling the arc (u, v), we use the leaving time of arc (u, v) to represent the earliest time to travel. The following symbols will be used in the algorithm below.

$P(u, \#w)$: the path from node s to node u satisfying: (1) the arrival time of u is minimal; (2) it has $\#w$ weighted stops.

$reverse(u, \#w)$: the node that precedes node u in the path $P(u, \#w)$.

$arrival(u, \#w)$: the earliest time to reach node u from a path with $\#w$ weighted stops.

$leaving(u, v, \#w)$: the earliest time to leave node u for v from a path with $\#w$ weighted stops.

When reaching node u at $arrival(u, \#w)$, we can leave for node v immediately if we are in "on" window, or we must wait until the next "on" window. Let the stop be weighted by $weight(u, v)$. Thus, we can compute the earliest leaving time of arc (u, v) as follows:

Find a value of j such that $ts_{g(j-1)}(u, v) \leqslant arrival(u, \#w) < ts_{w(j)}(u, v)$.

If the above value of j can be found, then $leaving(u, v, \#w) = arrival(u, \#w)$.

Otherwise, find a value of j such that $ts_{w(j-1)}(u, v) \leqslant arrival(u, \#w) < ts_{g(j)}(u, v)$.

Then, set $leaving(u, v, \#w + weight(u, v)) = ts_{g(j)}(u, v)$.

On the basis of the label-setting algorithm (Dijkstra, 1959), we present the following algorithm to find all the efficient paths from node s to node d. We refer to it as *multiple* labeling because a node can be associated with as many as $\#W$ labels. In the simple shortest path problem, a path with a larger value of label (i.e., time) is discarded. However, to solve our problem, a label cannot be discarded simply based on the value of time, but because it is less efficient.

Algorithm 3 (*Multiple labeling*)

1. Set $arrival(s, 0) = 0$.
   Create a table T with $|N|$ rows and $\#W$ columns, where entry $T(i, k)$ is for node i with k weighted stops.
   Let $T(s, 0) = (\text{on}, 0)$ and all other entries of T as (on, ∞), where the first element denotes the status and the second is the arrival time.
2.
   2.1. If each entry in T either has the status "off" or has the time ∞ then go to step 3.
   2.2. Among all "on" entries in T, choose the one with minimum arrival time.
   Let it be $T(u, \#w)$ with the value (on, $arrival(u, \#w)$).
   Set $T(u, \#w) = (\text{off}, arrival(u, \#w))$.
   Delete all entries $T(u, k)$ from T where $k > \#w$ and statuses as "on".
   2.3. For each node v adjacent to u,
   find a value of j such that $ts_{g(j-1)}(u, v) \leqslant arrival(u, \#w) < ts_{w(j)}(u, v)$.
   If the preceding value of j can be found,
   then $arrival(v, \#w) = \min\{arrival(u, \#w) + t(u, v), arrival(v, \#w)\}$,

If the status of $T(v,\#w)$ is "on" and $arrival(v,\#w)$ becomes smaller
then set $T(v,\#w) = (\text{on}, arrival(v,\#w))$ and $reverse(v,\#w) = (u,\#w)$.
Otherwise, find a value of j such that $ts_{w(j-1)}(u,v) \leqslant arrival(u,\#w) < ts_{g(j)}(u,v)$.
Then, let $\#r = \#w + weight(u,v)$,
$arrival(v,\#r) = \min\{ts_{g(j)}(u,v) + t(u,v),\ arrival(v,\#r)\}$.
If $T(v,\#r)$ is "on" and $arrival(v,\#r)$ becomes smaller,
then set $T(v,\#r) = (\text{on}, arrival(v,\#r)), reverse(v,\#r) = (u,\#w)$.

2.4. Go to step 2.1.

3. For $\#w$ from 0 to $\#W$ do
3.1. Find the optimal path $P(d,\#w)$ by traversing backward through $reverse(d,\#w)$.
3.2. Set the total time of path $P(d,\#w)$ to be $arrival(d,\#w)$.

4. Stop.

Example 1. Fig. 7(a) shows a network N, where the numbers inside the brackets are the travel times and the numbers inside the parentheses are the time-switches. In Fig. 7(a), assume that $weight({}^{s}A,{}^{A}C) = weight({}^{A}C,{}^{C}D) = weight({}^{A}C,{}^{C}B) = 2$, and 1 otherwise. Given the data, we can derive the on–off time sequences of all of the nodes as follows:

$$TS(s,{}^{s}A) = TS(s,{}^{s}C) = TS(s,{}^{s}B) = (\underline{0,\infty}),$$

$$TS({}^{s}A,d) = (\underline{2,4},\underline{7,9},\underline{12,14},\ldots),$$

$$TS({}^{s}A,{}^{A}C) = (\underline{2,2,4,7},\underline{7,7,9,12},\ldots),$$

$$TS({}^{C}A,d) = TS({}^{C}A,{}^{C}D) = TS({}^{C}A,{}^{C}B) = (\underline{4,6},\underline{11,13},\underline{18,20},\ldots),$$

$$TS({}^{s}C,d) = TS({}^{s}C,{}^{C}D) = TS({}^{s}C,{}^{C}B) = (\underline{4,4,6,11},\underline{11,11,13,18},\ldots),$$

$$TS({}^{s}B,{}^{B}D) = (\underline{1,5},\underline{5,9},\underline{9,13},\ldots),$$

$$TS({}^{C}B,{}^{B}D) = (\underline{1,1,2,5},\underline{5,5,6,9},\ldots),$$

$$TS({}^{B}D,d) = (\underline{2,4},\underline{10,12},\underline{18,20},\ldots),$$

$$TS({}^{C}D,d) = (\underline{2,2,4,10},\underline{10,10,12,18},\ldots).$$

Suppose we want to find all the efficient paths subject to $\#W = 2$. Fig. 7(b) shows the result of the first iteration of the algorithm, and Fig. 7(c)–(e) give the results of the remaining ones. For clarity, only the labels denoting arrival times of $\#w = 0$, 1 and 2 for each node are shown. In the beginning, labels of all nodes are ∞, ∞, ∞, except the node s with 0, ∞, ∞. Recall that an entry can be deleted or set as "off". If it is deleted (i.e., dominated), we will no longer consider it in the later iteration. On the other hand, we set the entry to "off" if it is selected. Since the status of an undeleted label can be "on" or "off", we use an underline to show the status of "off". In comparison, the label with a double strikeout line is a deleted one. Each iteration will select the minimum arrival time label from all the entries with status of "on". We summarize the entries selected and deleted in each iteration in Table 2. Finally, all undeleted labels in Fig. 7(e) have either statuses as "off" or arrival times as ∞, so we stop the algorithm.

According to the algorithm, we find two efficient paths: $P(d,0) = (s,{}^{s}A,d)$ with total time 13, and $P(d,1) = (s,{}^{s}C,{}^{C}D,d)$ with total time 11. It is interesting to observe how the trade-off between the travel time and number of stops reveals. Recall our model has reflected the relative importance of the road. From the result, path $P(d,0)$ takes more time than path $P(d,1)$ does, but has fewer weighted number of stops.

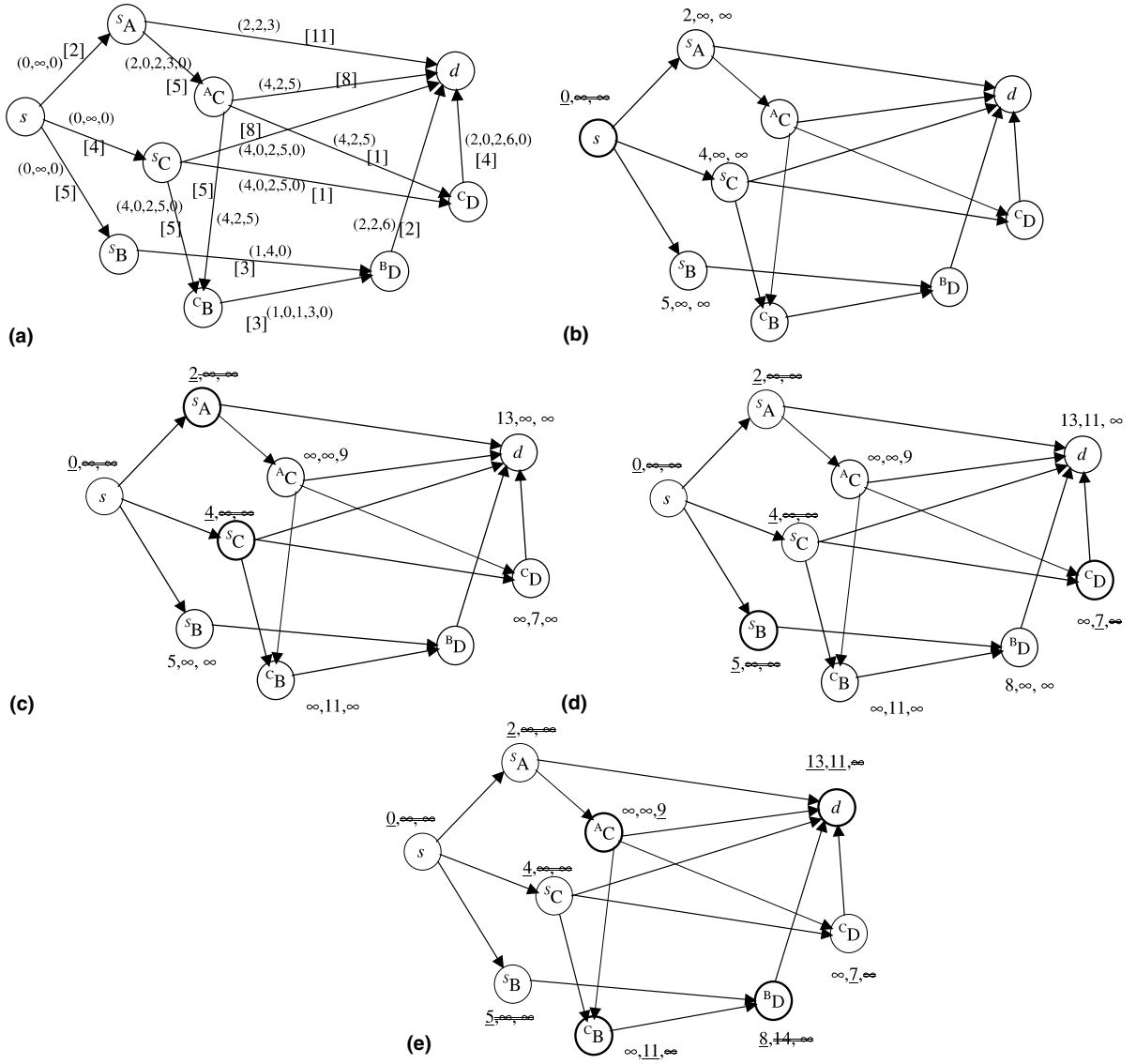

Fig. 7. (a) The original on–off time-switch network N. (b) The first iteration of the multiple labeling algorithm. (c) The second and third iterations. (d) The fouth and the fifth iterations. (e) From the sixth iteration to the final iteration.

From a practical viewpoint, it is not a surprising result because a road may constitute part of a *detour* that takes more time but encounters fewer number of traffic signals. As we stated at the beginning of this section, the solution depends on the decision scenario so that we can select the one by our preferences. In this case, we can either travel faster or experience fewer numbers of stops, but not both. Apparently, the trade-off explains why we find the efficient paths rather than optimal ones.

To prove Algorithm 3, we need to prove two things: (1) the selected entry $T(u,\#w)$ in each iteration of step 2 is efficient, i.e., no other entry in T having $arrival(u,k) \leqslant arrival(u,\#w)$ where $k < \#w$, and (2) if $arrival(u,\#w)$ is efficient, the algorithm will finally set $T(u,\#w) = (\text{off}, arrival(u,\#w))$ in some iteration of step 2. These two elements are presented in the following lemmas.

Table 2
The execution of Algorithm 3

Iteration no.	$T(u,\#w)$ selected	$arrival(u,\#w)$	Entries deleted
1	$T(s,0)$	0	$T(s,1)$, $T(s,2)$
2	$T({}^{s}A,0)$	2	$T({}^{s}A,1)$, $T({}^{s}A,2)$
3	$T({}^{s}C,0)$	4	$T({}^{s}C,1)$, $T({}^{s}C,2)$
4	$T({}^{s}B,0)$	5	$T({}^{s}B,1)$, $T({}^{s}B,2)$
5	$T({}^{C}D,1)$	7	$T({}^{C}D,2)$
6	$T({}^{B}D,0)$	8	$T({}^{B}D,1)$, $T({}^{B}D,2)$
7	$T({}^{A}C,2)$	9	–
8	$T({}^{C}B,1)$	11	$T({}^{C}B,2)$
9	$T(d,1)$	11	$T(d,2)$
10	$T(d,0)$	13	–

Lemma 3. *The selected entry $T(u,\#w)$ in step 2.2 is efficient.*

Lemma 4. *If $arrival(u,\#w)$ is efficient, then the algorithm will finally set $T(u,\#w) = (\text{off}, arrival(u,\#w))$ in some iteration of step 2.*

Combining Lemmas 3 and 4, Algorithm 3 is correct. To analyze its time complexity, we need the following lemma.

Lemma 5. *Given $arrival(u,\#w) + t(u,v)$ in step 2.3, $arrival(v,\#w)$ or $arrival(v,\#w + weight(u,v))$ can be determined in time $O(\log r)$, where r is the number of windows in an iteration of the repeated window sequence associated with node v.*

With Lemma 5, we obtain the time complexity of Algorithm 3.

Lemma 6. *The time complexity of Algorithm 3 is $O(\#W|A|\log r + \#W|V|\log|V|)$, where $|A|$ and $|V|$ are the numbers of arcs and nodes in the constructed on–off time-switch network, respectively.*

Finally, by Lemma 6, we derive the following theorem.

Theorem 1. *The bi-criteria shortest path problem in a traffic-light network can be solved in time $O(\#Wn^3)$, where n is the number of nodes in the original traffic-light network.*

5. Conclusions

This paper studies minimizing total travel time and weighted number of stops in a network subject to the traffic-light constraints that simulate the operations of a practical light control. To solve this bi-criteria problem, we first transform the traffic-light network into a counterpart network named on–off time-switch network and show that a shortest path in one network is equivalent to that in the other. In addition to maintaining the same mathematical structure, the on–off time-switch network is a simpler platform to develop algorithms for solving problems. As more transportation problems are studied, we hope the on–off time-switch network may provide a gateway for developing efficient algorithms. The second contribution is that polynomial algorithms are developed to find such paths that minimize total travel time and weighted number of stops.

The major limitation of the paper is that the proposed algorithm does not consider limited capacity during green nor queuing effects at the stop-line. Our model implies that all the waiting vehicles can immediately start regardless of the number of queued vehicles. In the real-world problems, the number of vehicles able to pass the signal during green depends on the capacity. This limitation underestimates delays that may be significant with traffic demand approaching capacity and in over-saturated conditions.

Finally, we mention some possible extensions of the paper. In addition to bi-criteria, more criteria may be included. For example, since the total time of a path is its traveling time plus waiting time, a natural extension is to consider the problem subject to a combination of goals such as the total time, the traveling time, the waiting time and the number of stops. Another possible extension is to consider the vehicle routing problem in a traffic-light network. As an example, we may assume that some nodes are required to be served, some node is the depot and a vehicle route should complete within a given threshold. Under the circumstance, the problem is to dispatch the minimum number of vehicles with a given capacity to complete the service in a traffic-light network.

Acknowledgements

The authors are grateful to the anonymous referees for their helpful comments. The second author was supported in part by National Science Foundation grant no. 89-2416-H-167-009.

Appendix A

Proof of Lemma 1. Step 1 iterates r times. In each iteration, every node-pair $\langle x, y\rangle$ in $NP_{u,k}$ is inserted into the queue. Thus, each iteration has at most $O(n^2)$ insertions, and the total time for step 1 is $O(rn^2) = O(n^2)$. Step 2 examines all $O(n^2)$ queues. Since the number of "on" windows in the queue plus the number of "off" windows not in the queue is r, the total time for processing the queue is $O(r)$. Therefore, the total time of step 2 is $O(rn^2) = O(n^2)$. □

Proof of Lemma 2. Since we create $n-1$ nodes for each node in V_2, the total number of nodes in the time-switch network is $O(n^2)$. For every arc (u, v) in the traffic-light network, we create one arc in steps 1 and 2 but create at most n arcs in steps 3 and 4. Therefore, the maximum number of arcs in time-switch network is bounded from above by $O(n^3)$. In step 6, we execute one time of Algorithm 1 for every node in V_2. By Lemma 1, the time of step 6 requires $O(n^3)$. Taken together, the lemma is proved. □

Proof of Lemma 3. By definition, if $T(u, \#w)$ is not efficient, then there exists an entry $T(u, k)$ satisfying $k < \#w$ and $arrival(u, k) \leqslant arrival(u, \#w)$. Since the algorithm uses an approach similar to that of Dijkstra's shortest path algorithm (Dijkstra, 1959) to select the next entry, the arrival times of the selected entries in step 2 are output in nondecreasing order. Therefore, entry $T(u, k)$ must be selected and output before $T(u, \#w)$. In this case, we would have deleted $T(u, \#w)$. This contradiction shows that no other entry in T satisfying $arrival(u, k) \leqslant arrival(u, \#w)$ where $k < \#w$. □

Proof of Lemma 4. If $arrival(u, \#w)$ is efficient, then there exists an efficient path from s to u with $\#w$ weighted stops. Besides, for every intermediate node x, the subpath from s to node x must also be efficient. Otherwise, we can replace the subpath from s to x by an efficient one, and the resulting path will be at least as good as the original one. □

We now use induction to prove that every efficient label will finally be set in some iteration of step 2. At the start, we set $T(s, 0) = (\text{off}, arrival(s, 0))$. Assume this is true for the remaining iterations before we select

$arrival(u, \#w)$. Let x be the node preceding node u in the efficient path from s to u with $\#w$ weighted stops. Therefore, the label of x is either $T(x, \#w - weight(x,u))$ or $T(x, \#w)$. Without loss of generality, assume it is $T(x, \#w - weight(x,u))$. When we process the label $T(x, \#w - weight(x,u))$, the entry $T(u, \#w)$ must exist and have the status as "on". If $T(u, \#w)$ has been deleted, it means in some earlier iteration of step 2 we had selected another entry $T(u,k)$, where $k < \#w$, with a smaller or the same total time as $arrival(x, \#w - weight(x,u))$ and therefore $arrival(u, \#w)$ is not efficient. This contradiction indicates that entry $T(u, \#w)$ must exist. Besides, $T(u, \#w)$ must have status "on". Otherwise, it means $arrival(u, \#w)$ has a smaller or the same total time as $arrival(x, \#w - weight(x,u))$, because we process the labels in nondecreasing order of arrival times. This is also a contradiction. Therefore, we will set the label of u as $T(u, \#w) = (\text{on}, arrival(u, \#w))$ when we select and process the label $T(x, \#w - weight(x,u))$.

After that, we will select $T(u, \#w)$ in step 2.2 of some later iteration. At this point, the label $T(u, \#w)$ must remain the same as that we set in the iteration for $T(x, \#w - weight(x,u))$. Otherwise, there is another path that is more efficient than the efficient path, and this is a contradiction. Therefore, this proves that we will finally set $T(u, \#w)$ as (off, $arrival(u, \#w)$) in step 2.2 and the arrival time is correct.

Proof of Lemma 5. Let $T = arrival(u, \#w) + t(u,v)$. Suppose $T_i = T_{i-1} + t_i$ for $i = 1$ to r. The most time-consuming task is to find a j such that $ts_{g(j-1)}(u,v) \leqslant T < ts_{w(j)}(u,v)$ or $ts_{w(j-1)}(u,v) \leqslant T < ts_{g(j)}(u,v)$. This can be done by the following procedure:

1. Set $T' = (T - o) \bmod T_r$. Set $r' = \lfloor (T-o)/T_r \rfloor$.
2. Use binary search to locate where the location of T' is in the sequence of ordered points $T_0, T_1, T_2, \ldots, T_r$. Assume that the location is $T_{2i-2} \leqslant T' < T_{2i-1}$.
3. Set $j = (r/2) \times r' + i$.

Obviously, the procedure above can be done in time $O(\log r)$ because of the binary search. Thus, the lemma is proved. However, this time complexity does not include the time to compute $T_i = T_{i-1} + t_i$ for $i = 1$ to r. Fortunately, this computation is done once for each node, and hence the total time for computing T_i in the whole algorithm is $O(|V|r)$, without affecting the final complexity of the algorithm. □

Proof of Lemma 6. Suppose we use Fibonacci heap to store all the entries $T(v, \#w)$ of T satisfying $T(v, \#w) = (\text{on}, value)$, where $value \neq \infty$. Note that, the advantage of using Fibonacci heap is that insertion, decrease-value and find-minimum operations can all be done in $O(1)$ amortized time, and delete-minimum operation in amortized time $O(\log h)$, where h is the number of elements in the heap (Fredman and Tarjan, 1987). Thus, the delete-minimum operation can be done in time $O(\log(\#W|V|)) = O(\log|V|)$, since the heap stores at most $\#W|V|$ elements.

Note that step 2.2 of Algorithm 3 iterates at most $\#W|V|$ times, because each iteration sets one entry to be "off" and there are $\#W|V|$ entries in the table. Thus, the find-minimum operation of step 2.2 can be done in time $O(\#W|V|)$. To set $T(u, \#w)$ to "off" in step 2.2, the operation needs to delete the minimum element from the heap. Therefore, the whole algorithm spends time $O(\#W|V|\log|V|)$ for this operation.

In addition, step 2.2 needs to delete all entries $T(u,k)$ for $k > \#w$. To implement this operation, two things should be done: (1) we must remove all these entries from the table, and (2) if some of them are already in the heap then we need to delete them from the heap. Part (1) needs time at most $O(\#W|V|)$ because there are $\#W|V|$ entries in the table. Note that, part (2) can be done by a decrease-value operation followed by another delete-minimum operation. Therefore, each operation of part (2) can be finished in time $O(\log|V|)$, and hence the total time for part (2) is $O(\#W|V|\log|V|)$. Putting together, we observe that step 2.2 can be done in time $O(\#W|V|\log|V|)$.

Concerning 2.3, note that we examine every arc at most $\#W$ times. For each examination of arc (u,v) in step 2.3, we need to determine the value of $arrival(v, \#w)$ or $arrival(v, \#w + weight(u,v))$, and then decrease

the value of the corresponding entry in the heap. The former can be done in time $O(\log r)$ by Lemma 5, and the latter can be done in time $O(1)$. Therefore, the total time required for step 2.3 is $O(\#W|A|(\log r))$. Combining the times for steps 2.2 and 2.3 together, we have the total time complexity as $O(\#W|A|\log r + \#W|V|\log|V|)$. □

Proof of Theorem 1. Lemma 2 shows the transformed on–off time-switch network has $O(n^2)$ nodes and $O(n^3)$ arcs. Applying Lemmas 2–6 and assuming r is a constant, we have the total time $O(\#Wn^3)$. □

References

Ahuja, R.K., Magnanti, T.L., Orlin, J.B., 1993. Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Englewood Cliffs, NJ.

Baker, E., 1982. Vehicle routing with time window constraints. Logistics & Transportation Review 18, 385–401.

Baker, E., 1983. An exact algorithm for the time-constrained traveling salesman problem. Operations Research 31, 938–945.

Balakrishnan, N., 1993. Simple heuristics for the vehicle routing problem with soft time windows. Journal of the Operational Research Society 44, 279–287.

Bodin, L.D., Golden, B.L., Assad, A.A., Ball, M.O., 1982. Routing and scheduling of vehicles and crews: The state of the art. Computers & Operations Research 10, 63–211.

Bramel, J., Simchilevi, D., 1996. Probabilistic analyses and practical algorithms for the vehicle routing problem with time windows. Operations Research 44, 501–509.

Chen, Y.L., Tang, K., 1997. Shortest paths in time-schedule networks. International Journal of Operations and Quantitative Management 3, 157–173.

Chen, Y.L., Tang, K., 1998. Minimal time paths in a network with mixed time constraints. Computers & Operations Research 25, 793–805.

Chen, Y.L., Tang, K., 2001. The first k minimum cost paths in a time-schedule network. Journal of Operational Research Society 52, 102–108.

Chen, Y.L., Yang, H.H., 2000. Shortest paths in traffic-light networks. Transportation Research B 34, 241–253.

Deo, N., Pang, C., 1984. Shortest path algorithms: Taxonomy and annotation. Networks 14, 275–323.

Desrochers, M., Soumis, F., 1988. A reoptimization algorithm for the shortest path problem with time windows. European Journal of Operational Research 35, 242–254.

Dijkstra, E.W., 1959. A note on two problems in connection with graphs. Numerische Mathematik 1, 269–271.

Dumas, Y., Desrosiers, J., Gelinas, E., Solomon, M., 1995. An optimal algorithm for the traveling salesman problem with time windows. Operations Research 43, 367–371.

Dumas, Y., Desrosiers, J., Soumis, F., 1991. The pickup and delivery problem with time windows. European Journal of Operational Research 54, 7–22.

Fredman, M.L., Tarjan, R.E., 1987. Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM 34, 596–615.

Fu, L., Rilett, L.R., 1998. Expected shortest paths in dynamic and stochastic traffic networks. Transportation Research B 32, 499–516.

Golden, B.L., Magnanti, T.L., 1977. Deterministic network optimization: A bibliography. Networks 7, 149–183.

Kolen, A.W.J., Kan, A.H.G.R., Trienekens, H.W.J.M., 1987. Vehicle routing with time windows. Operations Research 35, 266–273.

Russell, R.A., 1995. Hybrid heuristics for the vehicle-routing problem with time windows. Transportation Sciences 29, 156–166.