Computers & Operations Research 40 (2013) 2975–2982

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Finding the K shortest paths in a time-schedule network with constraints on arcs

Wen Jin, Shuiping Chen, Hai Jiang *

Department of Industrial Engineering, Tsinghua University, Beijing 100084, China

ARTICLE INFO

Available online 17 July 2013

Keywords:
Time-constrained network
Time-schedule network
K shortest paths
Constrained shortest paths

ABSTRACT

We study a new variant of the time-constrained shortest path problem (TCSPP), that is, the K shortest paths problem in a time-schedule network with constraints on arcs. In such networks, each arc has a list of pre-specified departure times, and traversal along the arc can only take place at one of those departure times. We develop a solution algorithm that finds the K shortest looping paths in $O(m\ \log(nr) + Kmr^2\eta)$ time, where n is the number of nodes, m is the number of arcs, r is the maximum number of departure times on an arc, and η is the maximum in-degree of a node. Computational experiments show that our algorithm outperforms existing ones adapted to solve the same problem.

1. Introduction

The time-constrained shortest path problem (TCSPP) is an important generalization of the shortest path problem and has attracted much research interest in recent years [1]. In time-constrained networks, nodes and arcs can be traversed subject to pre-specified time constraints. There are two common forms of time constraints in the literature. The first one is called time-window constraint, where a list of time intervals is defined for each node or arc, and node or arc traversal can only take place during these time intervals. The second one is named time-schedule constraints, where node or arc traversal can only occur at pre-specified times. Clearly, the second form is a special case of the first one. Previous study has examined TCSPPs in time-window networks with constraints on nodes [1] and arcs [2–4], and in time-schedule networks with constraints on nodes [5,6]. Yet, no existing research has studied the TCSPP in time-schedule networks with constraints on arcs, although this type of networks is widely encountered in practice. For example, a node in a time-schedule network may correspond to a bus stop and an arc between two nodes may correspond to a scheduled bus service. Arcs then can only be traversed at times when the buses are scheduled to depart.

Although the shortest path problem in a time-schedule network with constraints on arcs can be addressed by a few modifications to classical shortest path algorithms [2,5], the extension to finding the K shortest paths requires more efforts and has many potential applications. For example, a passenger may want to know the shortest 5 ways that she can arrive at her destination by bus; or, a shipper may want to know the shortest 10 alternative paths that can get her shipment from the origin to the destination by freight rail. Furthermore, if we view the K shortest paths as a set and compute its standard summary statistics, such as mean, variance, range, quartiles, and percentiles, it can be very helpful to network planners to plan, design and operate a transportation system.

The objective of this paper is to determine the K shortest paths between a given origin s and a given destination d in a time-schedule network with constraints on arcs. In our approach, an auxiliary network is first constructed where the destination node d is replaced by a set of nodes denoted as $\tilde{\mathcal{N}}$. By defining the K shortest paths problem between a node and a set of nodes, we show that finding the K shortest paths between node s and node d in the original network can be transformed to finding the K shortest paths between node s and nodes in $\tilde{\mathcal{N}}$ in the auxiliary network. In the auxiliary network, we develop an algorithm that runs in $O(m\ \log(nr) + Kmr^2\eta)$ time, where n is the number of nodes, m is the number of arcs, r is the maximum number of departure times on an arc, and η is the maximum in-degree of a node. In our approach, we modify Dijkstra's algorithm so as to be able to find the shortest path in time-schedule networks. We then develop a path enumeration algorithm and embed it into the modified Dijkstra's algorithm to find the K shortest paths. Computational experiments show that our algorithms outperform existing ones adapted to solve the same problem.

The rest of this paper is organized as follows. Section 2 reviews related literature. In Section 3, we introduce time-schedule networks with constraints on arcs and formally define the K shortest paths problem in such networks. Our solution approach is presented in Section 4 and computational results are reported in Section 5.

* Corresponding author. Tel.: +86 10 62796513.
E-mail address: haijiang@tsinghua.edu.cn (H. Jiang).

0305-0548/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2013.07.005

Finally, we conclude our discussion and outline possible future research directions in Section 6.

2. Literature review

Classical K shortest paths problems determine the first K minimum cost paths in a network where arc costs are constant and there are no restrictions on node or arc traversal. Ref. [7] classifies K shortest paths algorithms into two categories: (a) deviation algorithms [8–10], which find loopless paths; and (b) labeling algorithms [11–14], which may find looping paths. In networks with time constraints, classical K shortest path algorithms cannot be directly applied. In the following text, we briefly review related literature on the TCSPP.

In time-window networks with constraints on nodes, [1] finds the first K shortest paths. It introduces the concept of "path route" and "path", where a path route is a list of consecutive nodes without time constraints information while a path is a complete solution from the source to the destination with departure time and arrival time at each node of the path route. The algorithm performs in two steps. The first step finds a path route between the source node and the destination node, and the corresponding total time. This path route is added into a set Q. In the second step, a graph related to the path route with minimum total time in set Q is constructed to find all corresponding paths with the same total time. Yen's deviation algorithm [8] is used to generate loopless path routes. The complexity of this algorithm is $O(Kn^3r)$, where n is the number of nodes in the network and r is the maximum number of windows associated with a node.

In time-window networks with constraints on arcs, [2] considers a real life application of traffic lights, which has a repeated sequence of time-window constraints along each arc, called traffic-light constraint, to simulate the operations of traffic-light control encountered at street intersections. A modified Dijkstra's algorithm [15] is used to find the shortest path in this traffic-light network. Ref. [3] extends the work of [2] to find the first K shortest paths in a traffic-light network by modifying the deletion algorithm [13,14]. Later, [4] studies the problem of finding K shortest looping paths with waiting time in a traffic-light network. The algorithm iteratively finds path routes and their corresponding paths. Martins' deletion algorithm [13] is used to generate looping path routes, and a related graph is constructed to enumerate all paths. The time complexity of the above K shortest paths algorithms is $O(K^2n^3r)$, where n is the number of nodes in the network and r is the maximum number of windows associated with a node.

In time-schedule networks with constraints on nodes, [5] proposes an algorithm to find minimum time paths. The authors use a modified Dijkstra's algorithm to solve minimum total time problems. Later, [6] develops algorithms to find the K-th shortest path as well as the K shortest paths. The authors essentially create a time-expanded representation of the original network. Since the network is acyclic, all the nodes are topologically ordered. They identify the K-th shortest path by counting the number of paths from the source to each node.

While the aforementioned research focuses on the shortest or K shortest paths problem, [16] develops an algorithm to find the first K minimum cost paths without loops in time-schedule networks with constraints on nodes. The authors enlarge the network to a time-expanded network by adding nodes with time-subscripts for all time-schedule nodes. Yen's deviation algorithm [8] is applied in the enlarged network to find the first K minimum cost loopless paths. Ref. [17] studies the K shortest paths problem in a time varying network where the departure from the origin and the arrival at the destination are constrained within specified time windows.

The review of existing literature shows that we can distinguish various types of TCSPPs depending on the following criteria: (a) time constraints placed on nodes vs. on arcs; (b) time-window vs. time-schedule constraints, where the time-schedule constraints can be viewed as a special case of the time-window constraints with zero window width; (c) shortest (fastest) vs. minimum cost path problems. In shortest path problems, the cost of an arc is the travel time of that arc, while in minimum cost path problems, link costs can take general forms; (d) one path vs. K paths; and (e) looping vs. loopless paths, where in a looping path, nodes can be visited more than once. These five criteria can be classified into two groups: network characteristics defining the basic characteristics of the time-constrained network, that is, criteria (a) and (b), and problem objectives defining the goals of the problems, that is, criteria (c), (d), and (e). Table 1 classifies existing literature on TCSPPs. In this paper, we study the K shortest path problem in time-schedule networks with constraints on arcs.

3. Problem definition

Let $\mathcal{G}=(\mathcal{N},\mathcal{A})$ be a time-schedule network with constraints on arcs, where $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of arcs. Let $|\mathcal{N}|=n$ and $|\mathcal{A}|=m$. Each arc $(u,v)\in\mathcal{A}$ has a non-negative travel time $c(u,v)$ and a list of pre-specified departure times. These departure times are denoted as $TS(u,v)=(ts^1_{(u,v)}, ts^2_{(u,v)}, \ldots, ts^{r(u,v)}_{(u,v)})$, where $ts^i_{(u,v)}$ is the i-th departure time on arc (u,v) and $r(u,v)$ is the total number of departure times on arc (u,v). These departure times are in ascending order. Let r be the maximum number of departure times on an arc. For a node $u\in\mathcal{N}$, $x(u)$ denotes the set of nodes having an arc entering node u and $y(u)$ denotes the set of nodes that are the end of an arc outgoing from node u. Fig. 1 shows an example time-schedule network. In this example, $\mathcal{N}=\{s,A,B,C,D,d\}$ and $\mathcal{A}=\{(s,A),(s,B),(A,C),(A,D),(B,A),(B,D),(C,B),(C,d),(D,C),(D,d)\}$.

Table 1
Classification of existing literature on the Time-Constrained Shortest Path Problem (TCSPP).

Problem objectives		Time-window network		Time-schedule network	
		Constraints on nodes	Constraints on arcs	Constraints on nodes	Constraints on arcs
Shortest path problem	One path	[1]	[2]	[5]	This paper
	K paths		[3,4]	[6]	
Minimum cost path problem	One path				
	K paths			[16]	

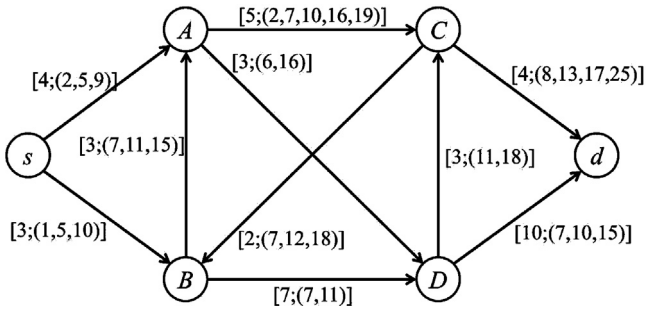

Fig. 1. An example time-schedule network with constraints on arcs. Each arc (u,v) is associated with a tuple $[c(u,v); TS(u,v)]$.

Each arc $(u,v)\in\mathcal{A}$ is associated with a tuple $[c(u,v);TS(u,v)]$. For example, [4;(2,5,9)] on arc (s, A) means $c(s,A)=4$, $TS(s,A)=(ts^1_{(s,A)}, ts^2_{(s,A)}, ts^3_{(s,A)})=(2,5,9)$, and $r(s,A)=3$. For node A, we have $x(A)=\{s,B\}$ and $y(A)=\{C,D\}$.

The time-schedule constraints placed on arcs work as follows: once arriving at node u, we have the option to immediately proceed to node $v\in y(u)$ along arc (u, v) if the arrival time at node u is among the departure times in $TS(u,v)=(ts^1_{(u,v)}, ts^2_{(u,v)}, \ldots, ts^{r(u,v)}_{(u,v)})$. If not, we must wait until the next departure time in $TS(u,v)$. Note that we are not obligated to leave at the next departure time; instead, we can choose to stay at node u for a longer time and depart at later departure times in $TS(u,v)$. In our example, $TS(C,B)=(7,12,18)$. If we arrive at node C at time 5, we need to wait a period of 2 time units and depart at 7 along arc (C, B). Alternatively, we can also choose to wait another 5 time units at C to depart at time 12 along arc (C, B).

Given two distinct nodes s and d in $\mathcal{G}$. A path from node s to node d specifies the sequence of nodes visited and the corresponding departure time at each intermediate node. For example, $[s_2;A_7;C_{13};d;17]$ denotes a path that departs node s at time 2 (and enters node A at time 6), then departs node A at time 7 (and enters node C at time 12), and finally departs node C at time 13 and enters destination d at time 17. The subscript indicates the departure time at each node, while the last number indicates the final arrival time at the destination node. Path $[s_2;A_7;C_{17};d;21]$ is another path that traverses the same set of nodes as path $[s_2;A_7;C_{13};d;17]$, but arrives at d at a later time. Two paths can also be concatenated by the operator $\oplus$ to form a longer path. For example, let $P_1=[s_2;A_7;C;12]$ and $P_2=[C_{13};d;17]$. Since the arrival time of P_1 at node C is 12, which is smaller than the departure time of P_2 from node C, we can concatenate these two paths to form a complete path from s to d as $P_1\oplus P_2=[s_2;A_7;C;12]\oplus[C_{13};d;17]=[s_2;A_7;C_{13};d;17]$. However, if $P_1=[s_5;A_{10};C;15]$, $P_1\oplus P_2$ is infeasible because the arrival time of P_1 at node C is now 15, which is greater than the departure time of P_2 from node C.

In the scope of this research, a path can visit a node more than once at different times. Now we formally define the K shortest paths problem studied in this paper as follows:

Definition 1. Let s and d be two distinct nodes in $\mathcal{G}$. The K shortest paths problem between s and d determines the first K paths from s to d in non-decreasing order of their arrival times.

4. The solution approach

In our solution approach, we first modify the original network $\mathcal{G}$ to create an auxiliary network $\mathcal{G}'$ to facilitate the development of the algorithm. We then transform the K shortest paths problem in $\mathcal{G}$ to an equivalent problem in $\mathcal{G}'$. We finally develop and integrate a modified Dijkstra's algorithm and a path enumeration algorithm to find the K shortest paths in $\mathcal{G}'$.

4.1. The auxiliary network

To facilitate the development of the solution algorithm, we create an auxiliary time-schedule network $\mathcal{G}'=(\mathcal{N}',\mathcal{A}')$ by replacing node d in $\mathcal{G}$ with $|x(d)|$ nodes, that is, node d is removed and for each node $u\in x(d)$, we add a node $\tilde{u}$ to $\mathcal{G}'$. These new nodes are denoted as $\tilde{\mathcal{N}}=\{\tilde{u}|u\in x(d)\}$. Arc (u, d) is also replaced by arc $(u,\tilde{u})$ in $\mathcal{G}'$ and the set of such arcs is referred to as $\tilde{\mathcal{A}}=\{(u,\tilde{u})|u\in x(d)\}$. $\mathcal{G}'$ is formally defined as follows:

1. $\mathcal{N}'=\mathcal{N}|\{d\}\cup\tilde{\mathcal{N}}$.
2. $\mathcal{A}'=\mathcal{A}\backslash\{(u,d)|u\in x(d)\}\cup\tilde{\mathcal{A}}$.

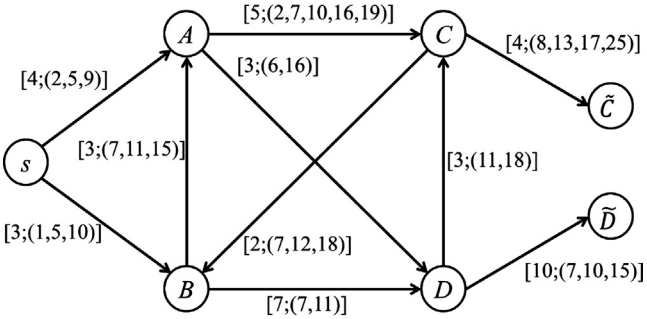

Fig. 2. The auxiliary network for the network shown in Fig. 1.

3. For arcs in $\mathcal{A}\backslash\{(u,d)|u\in x(d)\}$, their associated travel times and departure time lists remain the same as those in $\mathcal{G}$. For each arc $(u,\tilde{u})\in\tilde{\mathcal{A}}$, $c(u,\tilde{u})$ equals $c(u,d)$ in $\mathcal{G}$. Its corresponding departure time list $TS(u,\tilde{u})$ equals $TS(u,d)$ in $\mathcal{G}$.

It is not difficult to see that the network transformation takes $O(m+n)$ time. Fig. 2 shows the auxiliary network $\mathcal{G}'$ for the network shown in Fig. 1. In this example, $x(d)=\{C,D\}$ in $\mathcal{G}$, therefore in the auxiliary network, we replace node d with nodes $\tilde{C}$ and $\tilde{D}$. This means $\tilde{\mathcal{N}}=\{\tilde{C},\tilde{D}\}$ and $\mathcal{N}'=\mathcal{N}n\{d\}\cup\tilde{\mathcal{N}}=\{s,A,B,C,D,\tilde{C},\tilde{D}\}$. Arcs (C,d) and (D,d) are replaced by arcs $(C,\tilde{C})$ and $(D,\tilde{D})$. That is, we have $\tilde{\mathcal{A}}=\{(C,\tilde{C}),(D,\tilde{D})\}$ and $\mathcal{A}'=\mathcal{A}\backslash\{(u,d)|u\in x(d)\}\cup\tilde{\mathcal{A}}=\{(s,A),\ (s,\ B),\ (A,C),(A,D),(B,A),(B,D),(C,B),(C,\tilde{C}),(D,C),(D,\tilde{D})\}$. The travel times and departure time lists on arcs $(C,\tilde{C})$ and $(D,\tilde{D})$ are the same as those on arcs (C,d) and (D,d), respectively.

Definition 2. For a path P from node s to node d in $\mathcal{G}$. Suppose that P starts from node s at time t_0, subsequently visits node n^1 at time t_1, node n^2 at time t_2, …, node n^α at time t_α (α is the number of intermediate nodes), and finally arrives at node d at time $t_{\alpha+1}$, that is, $P=[s_{t_0};n^1_{t_1};n^2_{t_2};\ldots;n^\alpha_{t_\alpha};d;t_{\alpha+1}]$. Its corresponding path in $\mathcal{G}'$ is defined as $P'=[s_{t_0};n^1_{t_1};n^2_{t_2};\ldots;n^\alpha_{t_\alpha};\tilde{n}^\alpha;t_{\alpha+1}]$.

Note that P and P' visit the same sequence of nodes with the same departure times at those nodes, and arrive at their respective destinations at the same time. For example, for path $P=[s_2;A_7;C_{13};d;17]$ in $\mathcal{G}$, its corresponding path in $\mathcal{G}'$ is $P'=[s_2;A_7;C_{13};\tilde{C};17]$. Everything is the same between P and P' except the destination nodes. Now, for a path P' in $\mathcal{G}'$, we also define its corresponding path in $\mathcal{G}$:

Definition 3. For a path $P'=[s_{t_0};n^1_{t_1};n^2_{t_2};\ldots;n^\alpha_{t_\alpha};\tilde{n}^\alpha;t_{\alpha+1}]$ in $\mathcal{G}'$ that starts from s and arrive at $\tilde{n}^\alpha\in\tilde{\mathcal{N}}$, its corresponding path in $\mathcal{G}$ is defined as $P=[s_{t_0};n^1_{t_1};n^2_{t_2};\ldots;n^\alpha_{t_\alpha};d;t_{\alpha+1}]$.

Definitions 2 and 3 imply that there is a one-to-one correspondence between a path from s to d in $\mathcal{G}$ and a path from s to a node in $\tilde{\mathcal{N}}$ in $\mathcal{G}'$. Now we define the K shortest paths problem between node s and nodes in $\tilde{\mathcal{N}}$ in $\mathcal{G}'$.

Definition 4. Let s be a node and $\tilde{\mathcal{N}}$ be a set of nodes, where $s\notin\tilde{\mathcal{N}}$. The K shortest paths problem between s and $\tilde{\mathcal{N}}$ determines the first K paths from node s to nodes in $\tilde{\mathcal{N}}$ in non-decreasing order of their arrival times.

Note that although these paths all start from node s, they can end at different nodes in $\tilde{\mathcal{N}}$. For example, the first shortest path may end at node $\tilde{w}_1\in\tilde{\mathcal{N}}$, while the second shortest path may end at node $\tilde{w}_2\in\tilde{\mathcal{N}}$ ($\tilde{w}_1\neq\tilde{w}_2$).

Proposition 1. *If we find the first K shortest paths between s and $\tilde{N}$ in $\mathcal{G}'$, their corresponding paths in $\mathcal{G}$ are the K shortest paths between s and d in $\mathcal{G}$.*

Proof. This can be proved by contradiction. Suppose Φ' is the set of the K shortest paths between s and $\tilde{N}$ in $\mathcal{G}'$ and t' is the maximum final arrival time of these paths. Let Φ be the set of their corresponding paths in $\mathcal{G}$. This indicates that the maximum final arrival time of the paths in Φ is also t'. We just need to show that we cannot find another path $P \notin \Phi$ in network $\mathcal{G}$, whose final arrival time at node d is less than t'.

Assume that there exists such a path $P=[s_{t_0}; n^1_{t_1}; n^2_{t_2}; \ldots; n^\alpha_{t_\alpha}; d; t_{\alpha+1}]$, where $t_{\alpha+1} < t'$, and P is not in set Φ. Its corresponding path in $\mathcal{G}'$ is $P'=[s_{t_0}; n^1_{t_1}; n^2_{t_2}; \ldots; n^\alpha_{t_\alpha}; \tilde{n}^\alpha; t_{\alpha+1}]$. Therefore, we have identified a path P' that arrives earlier than t' but does not belong to Φ'. This contradicts the assumption that Φ' is the set of the K shortest paths between s and $\tilde{N}$ in $\mathcal{G}'$. □

Proposition 1 allows us to transform the K shortest paths problem between s and d in the original network to the K shortest paths problem between s and $\tilde{N}$ in the auxiliary network.

4.2. Modified Dijkstra's algorithm for time-schedule networks

In a time-schedule network with constraints on arcs, Dijkstra's algorithm cannot be applied directly and has to be modified. Without loss of generality, we assume that the earliest departure time at s is 0. Before we present the modified algorithm, we introduce the following additional notation for node u:

$t(u)$ the earliest arrival time at node u when leaving node s at time 0. $t(u)$ is also referred to as the time label of node u;

$p(u)$ the predecessor node on the shortest path from node s to node u;

$q(u)$ index of the departure time in $T(p(u),u)$ taken by the shortest path from node s to node u while traversing arc $(p(u),u)$. For example, when the shortest path traverses arc (v, u) via its third departure time to reach node u. We have $p(u)=v$ and $q(u)=3$;

S: set of nodes that have been reached and that are candidates for the selection of the next node.

The modified Dijkstra's algorithm is presented in Algorithm 1. On Lines 5 and 6, we select the node in S with minimum time label and remove it from S. This operation is also referred to as permanently labeling node u. On Line 7, we scan the nodes in $y(u)$. On Line 8, we find the smallest departure time index i such that $t^i_{(u,v)} \geq t(u)$. On Line 9, we check if we can decrease the time label of node v by traversing arc (u, v) at its i-th departure time. If so, we update $p(v)$, $q(v)$ and $t(v)$ on Lines 10 through 13.

Algorithm 1. The modified Dijkstra's algorithm.

```
1  Initialization: t(s)←0, p(s)←*, q(s)←*, S←{s}
2                  t(v)←∞, p(v)←*, q(v)←*, ∀v∈N'\{s}
3  begin
4    repeat
5      u←arg min_{w∈S} t(w)
6      S←S\{u}
7      for all v∈y(u) do
8        i←arg min_j {ts^j_(u,v) | ts^j_(u,v) ≥ t(u), 1≤j≤r(u,v)}
9        if t(v) > ts^i_(u,v) + c(u,v) then
10         p(v)←u
11         q(v)←i
12         t(v)←ts^i_(u,v) + c(u,v)
13         S←S∪{v}
14       end
15     end
16   until S = φ
17 end
```

Table 2 shows the steps when we apply the modified Dijkstra's algorithm to the auxiliary network shown in Fig. 2. Column 1 shows the iteration number. Column 2 shows the elements in set S at the beginning of each iteration. Column 3 shows the node with minimum time label in S. This node is selected, permanently labeled, and removed from S. In Columns 4 through 10, we show the values of $p(v)$, $q(v)$, and $t(v)$ for each node $v \in \{s, A, B, C, D, \tilde{C}, \tilde{D}\}$ after we execute Lines 9 through 14. When we initialize the algorithm, $S=\{s\}$. For node s, we have $p(s)=*$, $q(s)=*$, and $t(s)=0$. For all other nodes $v \in \mathcal{N}'\backslash\{s\}$, we have $p(v)=*$, $q(v)=*$, and $t(v)=\infty$. In Iteration 1, $S=\{s\}$ at the beginning of this iteration. Node s gets selected and removed from S, that is, it is permanently labeled. We scan nodes in $y(s)=\{A,B\}$. For node A, the earliest departure time index we can take to traverse arc (s, A) is 1 and the corresponding departure time is $t^1_{(s,A)}=2$. Since $ts^1_{(s,A)} + c(s,A) = 2+4 = 6 < t(A) = \infty$, we update the labels at node A as follows: $p(A)=s$, $q(A)=1$, and $t(A)=6$. Similarly, we can get $p(B)=s$, $q(B)=1$, and $t(B)=t^1_{(s,B)} + c(s,B) = 1+3 = 4$. Nodes A and B both get inserted into S and at the end of this iteration, $S=\{A,B\}$. In the second iteration, we start with $S=\{A,B\}$ and since $t(B) < t(A)$, node B is selected and removed from S. We scan nodes in $y(B)=\{A,D\}$. For node A, the earliest departure time index we can take to traverse arc (B, A) is 1 and the corresponding departure time is $t^1_{(B,A)}=7$. Since $t(A)=6 < t^1_{(B,A)} + c(B,A) = 7+3 = 10$, we leave the labels on node A unchanged. For node D, the earliest departure time index we can take to traverse arc (B, D) is 1 and the corresponding departure time is $t^1_{(B,D)}=7$. Since $t(D)=\infty > t^1_{(B,D)} + c(B,D) = 7+7 = 14$, we update the labels on node D as follows: $p(D)=B$, $q(D)=1$, and $t(D)=14$. This process continues until S becomes empty. In this table, when a node v is permanently labeled in one iteration, we omit the values of $p(v)$, $q(v)$, and $t(v)$ in

Table 2
The steps of the modified Dijkstra's algorithm when applied to the auxiliary network shown in Fig. 2 to find shortest paths from s to all other nodes.

Iteration	S	u	s	A	B	C	D	$\tilde{C}$	$\tilde{D}$
Initialization	{s}		*, *, 0	*, *, ∞	*, *, ∞	*, *, ∞	*, *, ∞	*, *, ∞	*, *, ∞
1	{s}	s	*, *, 0	s, 1, 6	s, 1, 4	*, *, ∞	*, *, ∞	*, *, ∞	*, *, ∞
2	{A, B}	B		s, 1, 6	s, 1, 4	*, *, ∞	B, 1, 14	*, *, ∞	*, *, ∞
3	{A, D}	A		s, 1, 6		A, 2, 12	A, 1, 9	*, *, ∞	*, *, ∞
4	{D, C}	D				A, 2, 12	A, 1, 9	*, *, ∞	D, 2, 20
5	{C, $\tilde{D}$}	C				A, 2, 12		C, 2, 17	D, 2, 20
6	{$\tilde{D}$, $\tilde{C}$}	$\tilde{C}$						C, 2, 17	D, 2, 20
7	{$\tilde{D}$, $\tilde{C}$}	$\tilde{D}$							D, 2, 20
8	{$\tilde{C}$, $\tilde{D}$}	∅							

subsequent iterations. For example, Column 4 becomes blank starting from Iteration 2.

Proposition 2. *The time complexity of the modified Dijkstra's algorithm is $O(m \log(nr))$ if set S is implemented as a heap.*

Proof. Each time, it takes $O(\log n)$ to select and remove a node u with the minimum time label from S. The total time for node selection is $O(n \log n)$. When updating labels for each $v \in y(u)$, it takes $O(\log r)$ to scan the departure time list on arc (u, v) and $O(\log n)$ to insert v into S if v is not in S or update v if v is already in S. The time for updating labels is $|y(u)|(\log n + \log r)$ for node u and the total time for label updates is $O(m \log(nr))$. Hence, the overall time complexity is $O(m \log(nr))$. □

4.3. Our algorithm for the K shortest paths problem

Now that we know how to find the shortest path in a time-schedule network with constraints on arcs, we use a simple example to illustrate how to find the K shortest paths. Suppose that our goal is to find the first 20 paths. We construct the auxiliary network according to the steps in Section 4.1 and apply the modified Dijkstra's algorithm presented in Section 4.2 until we permanently label a node in $\tilde{\mathcal{N}}$. This is the shortest path between node s and nodes in $\tilde{\mathcal{N}}$. Let P_1 be this path and suppose it arrives at node $\tilde{w}_1 \in \tilde{\mathcal{N}}$ via arc $(w_1, \tilde{w}_1)$. Note that the construction of the auxiliary network ensures that $p(\tilde{w}_1) = w_1$. For example, in Fig. 2, $p(\tilde{C}) = C$. Let us assume that P_1 takes the j_1-th departure in $TS(w_1, \tilde{w}_1)$, that is, $q(\tilde{w}_1) = j_1$. The arrival time at node $\tilde{w}_1$ is then $t(\tilde{w}_1) = t^{j_1}_{(w_1,\tilde{w}_1)} + c(w_1, \tilde{w}_1)$. Since there may be paths other than P_1 that take the j_1-th departure along arc $(w_1, \tilde{w}_1)$ and arrive at time $t(\tilde{w}_1)$, we need to find those paths as well. Let us denote the set of such paths (including path P_1) as $R^{j_1}_{(w_1,\tilde{w}_1)}$ and enumerate all paths in $R^{j_1}_{(w_1,\tilde{w}_1)}$ according to the path enumeration procedure to be presented in Section 4.3.1. Suppose there are 8 paths in $R^{j_1}_{(w_1,\tilde{w}_1)}$ and we can output these paths as the first 8 paths that arrive at node $\tilde{w}_1$ at time $t(\tilde{w}_1)$. They may traverse different nodes with different departure times from those nodes, but what is in common is they all traverse arc $(w_1, \tilde{w}_1)$ via its j_1-th departure and finally arrive at node $\tilde{w}_1$.

Since $8 < 20$, we need to find more paths. The next earliest arrival time at node $\tilde{w}_1$ takes the $(j_1 + 1)$-th departure over arc $(w_1, \tilde{w}_1)$ and arrives at time $ts^{j_1+1}_{(w_1,\tilde{w}_1)} + c(w_1, \tilde{w}_1) = T$. This set of paths is denoted as $R^{j_1+1}_{(w_1,\tilde{w}_1)}$. If T is the next earliest arrival time at nodes in $\tilde{\mathcal{N}}$, we can enumerate paths in $R^{j_1+1}_{(w_1,\tilde{w}_1)}$ as the next set of shortest paths. To check whether T is indeed the next earliest arrival time at nodes in $\tilde{\mathcal{N}}$, we continue the modified Dijkstra's algorithm until we permanently label a second node in $\tilde{\mathcal{N}}$. Let this node be $\tilde{w}_2$ and the shortest path from node s to $\tilde{w}_2$ be P_2. Suppose that P_2 traverses arc $(w_2, \tilde{w}_2)$ through the j_2-th departure time, that is, $q(\tilde{w}_2) = j_2$. The arrival time of P_2 is then $t(\tilde{w}_2) = ts^{j_2}_{(w_2,\tilde{w}_2)} + c(w_2, \tilde{w}_2)$. By comparing T and $t(\tilde{w}_2)$, we know the next earliest arrival time at nodes in $\tilde{\mathcal{N}}$. If $T \le t(\tilde{w}_2)$, we enumerate all paths in $R^{j_1+1}_{(w_1,\tilde{w}_1)}$ to find the set of paths with the next earliest arrival time; otherwise, we enumerate all paths in $R^{j_2}_{(w_2,\tilde{w}_2)}$. Suppose that $T > t(\tilde{w}_2)$ and there are 15 paths in $R^{j_2}_{(w_2,\tilde{w}_2)}$. Since it is a little more than what we require, we only need to output 12 of them as the solution. This way, we successfully generate the first 20 paths in the network.

4.3.1. The function to enumerate paths in $R^j_{(w,\tilde{w})}$

Following our previous notation, $R^j_{(w,\tilde{w})}$ refers to the set of paths arriving at node $\tilde{w}$ through the j-th departure along arc $(w, \tilde{w})$. First of all, we know all of these paths should arrive at node w no later than $t^j_{(w,\tilde{w})}$. Otherwise, they are not able to traverse arc $(w, \tilde{w})$ by its j-th departure and finally arrive at $\tilde{w}$ at time $t^j_{(w,\tilde{w})} + c(w, \tilde{w})$. Hence, paths in $R^j_{(w,\tilde{w})}$ take the following form $P = [s_{t_0}; n^1_{t_1}; n^2_{t_2}; \ldots; n^\alpha_{t_\alpha}; w_{t^j_{(w,\tilde{w})}}; \tilde{w}; t^j_{(w,\tilde{w})} + c(w, \tilde{w})]$, where $n^1, n^2, \ldots, n^\alpha$ are the intermediate nodes and $t_0, t_1, \ldots, t_\alpha$ are the corresponding departure times. If we define $\hat{P} = [s_{t_0}; n^1_{t_1}; n^2_{t_2}; \ldots; n^\alpha_{t_\alpha}; w; t_\alpha + c(n^\alpha, w) = \hat{t}]$ and $P^0 = [w_{t^j_{(w,\tilde{w})}}; \tilde{w}; t^j_{(w,\tilde{w})} + c(w, \tilde{w})]$, P can be written as $P = \hat{P} \oplus P^0$. Note that we must have $\hat{t} \le t^j_{(w,\tilde{w})}$; otherwise, $\hat{P} \oplus P^0$ is infeasible.

Hence, to find all paths in $R^j_{(w,\tilde{w})}$, we just need to find all paths in the following form $\hat{P} = [s_{t_0}; n^1_{t_1}; n^2_{t_2}; \ldots; n^\alpha_{t_\alpha}; w; \hat{t}]$, where $\hat{t} \le t^j_{(w,\tilde{w})}$. That is, we need to find all paths arriving at node w no later than $t^j_{(w,\tilde{w})}$. These paths can then be concatenated with P^0 to form complete paths from s to $\tilde{w}$ that take the j-th departure while traversing arc $(w, \tilde{w})$. This process can be accomplished by calling the recursive function Backtrack() shown in Algorithm 2, that is, by calling BackTrack$(w, t^j_{(w,\tilde{w})}, P^0)$. Note that since $w = p(\tilde{w})$, BackTrack$(w, t^j_{(w,\tilde{w})}, P^0)$ can be written as BackTrack$(p(\tilde{w}), t^j_{(p(\tilde{w}),\tilde{w})}, P^0)$ and P^0 can be written as $[p(\tilde{w})_{t^j_{(p(\tilde{w}),\tilde{w})}}; \tilde{w}; t^j_{(w,\tilde{w})} + c(p(\tilde{w}), \tilde{w})]$.

In function BackTrack(u, b, p), u indicates the node from which we backtrack to find all incoming arcs, b is the latest allowable arrival time at u, and p stores a path that can be followed from u to reach the destination node. It behaves similar to depth-first search except that when we examine node $v \in x(u)$, we need to scan the list of departure times along arc (v, u) that can arrive at node u by time b. On Line 2, we scan node v in $x(u)$. On Lines 3 and 4, if the departure time $t^i_{(v,u)}$ falls between $t(v)$ and $b - c(v, u)$, we prefix path p with path $[v_{ts^i_{(v,u)}}; u; ts^i_{(v,u)} + c(v, u)]$, which departs node v, traverses arc (v, u) via its i-th departure, and arrives at u at time $ts^i_{(v,u)} + c(v, u)$. Between Lines 6 and 10, we output p if we reach the source node; otherwise, we continue the backtrack procedure from node v.

Algorithm 2. BackTrack(u,b,p).

```
1  begin
2    for v∈x(u) do
3     for i = 1, ..., r(v, u) do
4       if t(v)≤ts^i_(v,u)≤b−c(v, u) then
5
6         p←[v_(ts^i_(v,u)); u; ts^i_(v,u) + c(v, u)]⊕p
7         if v = s then
8
9           output p
10        else
11
12          BackTrack(v, ts^i_(v,u), p)
13        end
        end
      end
    end
14 end
```

Proposition 3. *The time complexity of BackTrack for outputting K paths is $O(Kmr^2\eta)$, where η is the maximum in-degree of a node in $\mathcal{G}'$.*

Proof. We first get the time complexity for outputting one path. Since we have $O(m)$ arcs in $\mathcal{G}'$ and each of these arcs has at most r possible departures, a path from s to $\tilde{w}$ may contain at most $O(mr)$ nodes. When we backtrack from a node, we examine all arcs entering this node and the departure time lists on those arcs, this requires $O(r\eta)$ time. Hence, the overall time to output one path is $O(mr^2\eta)$ and the overall time to output K paths is $O(Kmr^2\eta)$. □

4.3.2. Algorithm statement

In Algorithm 3, we present the overall algorithm to find the K shortest paths. If we ignore the statement between Lines 16 and

23, it essentially is the same as the modified Dijkstra's algorithm shown earlier in Algorithm 1. Line 16 checks if we have reached a node in $\tilde{\mathcal{N}}$. If so, we enumerate paths in $R^{q(u)}_{(p(u),u)}$. This is achieved by calling BackTrack$(p(u), ts^{q(u)}_{(p(u),u)}, P^0)$, where $P^0 = [p(u)_{ts^{q(u)}_{(p(u),u)}}; u; ts^{q(u)}_{(p(u),u)} + c(p(u),u)]$. On Line 18, we check if we can traverse arc $(p(u),u)$ at a later time to arrive at node u. If so, we update $t(u)$ by increasing its value to $ts^{q(u)+1}_{(p(u),u)} + c(p(u),u)$, the next earliest arrival time at node u along arc $(p(u),u)$. In addition, $q(u)$ is also increased by 1. Finally, on Line 21 we insert u back into set S, so that the next earliest arrival time at node u can be compared to the time labels of other nodes in $\tilde{\mathcal{N}}$ to determine the next set of shortest paths.

Algorithm 3. Our algorithm to find the K shortest paths.

1 **Initialization**: $t(s)\leftarrow 0, p(s)\leftarrow *, q(s)\leftarrow *, S\leftarrow\{s\}, k\leftarrow 0$
2 $t(v)\leftarrow\infty, p(v)\leftarrow *, q(v)\leftarrow *, \forall v\in\mathcal{N}'\backslash\{s\}$
3 **begin**
4 **repeat**
5 $u\leftarrow \arg\min_{w\in S} t(w)$
6 $S\leftarrow S\backslash\{u\}$
7 **for** *all* $v\in y(u)$ **do**
8 $i\leftarrow \arg\min_i\{ts^i_{(u,v)} | ts^i_{(u,v)}\geq t(u)\}$
9 **if** $t(v) > ts^i_{(u,v)} + c(u,v)$ **then**
10 $p(v)\leftarrow u$
11 $q(v)\leftarrow i$
12 $t(v)\leftarrow ts^i_{(u,v)} + c(u,v)$
13 $S\in S\cup\{v\}$
14 **end**
15 **end**
16 **if** $u\in\tilde{\mathcal{N}}$ **then**
17 *Enumerate paths in* $R^{q(u)}_{(p(u),u)}$ *by calling the BackTrack function*
18 **if** $q(u) < r(p(u),u)$ **then**
19 $t(u)\leftarrow ts^{q(u)+1}_{(p(u),u)} + c(p(u),u)$
20 $q(u)\leftarrow q(u)+1$
21 $S\in S\cup\{u\}$
22 **end**
23 **end**
24 **until** $S=\phi$ *or* K *path have been found*
25 **end**

We want to emphasize the significance of increasing $t(u)$, the time label of node u, on Line 19 and insert it back into S. Recall that when we illustrate the idea of our algorithm in the beginning of Section 4.3, we need to compare T to $t(\tilde{w}_2)$ to determine whether paths in $R^{j_1+1}_{(w_1,\tilde{w}_1)}$ or paths in $R^{j_2}_{(w_1,\tilde{w}_1)}$ arrive at $\tilde{\mathcal{N}}$ earlier. The value of T in that example actually corresponds to the updated value of $t(\tilde{w}_1)$ in this algorithm. Hence, comparing T and $t(\tilde{w}_2)$ is equivalent to comparing the updated $t(\tilde{w}_1)$ and $t(\tilde{w}_2)$. By increasing the value of $t(\tilde{w}_1)$ and inserting $\tilde{w}_1$ back into S, it allows us to embed such comparison into the node selection process of the modified Dijkstra's algorithm on Line 5 of Algorithm 3.

Table 3 shows the steps when we apply our K shortest paths algorithm to the example network shown in Fig. 2. Iterations 1 through 5 are no different from those shown in Table 2. At Iteration 6, node $\tilde{C}$ is selected and removed from S. There are no nodes in $y(\tilde{C})$, so we skip Lines 7 through 15 in Algorithm 3. Because $\tilde{C}\in\tilde{\mathcal{N}}$, Lines 16 through 22 are executed: since the labels at node $\tilde{C}$ are: $p(\tilde{C})=C$, $q(\tilde{C})=2$, $t(\tilde{C})=17$, we enumerate paths in $R^{q(\tilde{C})}_{(p(\tilde{C}),\tilde{C})} = R^2_{(C,\tilde{C})}$. There is only one path (shown in Column 11), so $k=1$. Because $q(\tilde{C})=2 < r(C,\tilde{C})=3$, we update $t(\tilde{C})$ to $ts^{q(\tilde{C})+1}_{(p(\tilde{C}),\tilde{C})} + c(p(\tilde{C}),\tilde{C}) = ts^3_{(C,\tilde{D})} + c(C,\tilde{C}) = 17+4=21$ and increase the value of $q(\tilde{C})$ from 2 to 3. Node $\tilde{C}$ is inserted back into S. Note that at the beginning of Iteration 6, we have $p(\tilde{C})=$C, $q(\tilde{C})=2$, and $t(\tilde{C})=17$ (shown in Iteration 5); while at the end of this iteration, we have $p(\tilde{C})=C$, $q(\tilde{C})=3$, and $t(\tilde{C})=21$ (shown in Iteration 6).

At the beginning of Iteration 7, we have $S=\{\tilde{C},\tilde{D}\}$. Node $\tilde{D}$ is selected and removed from S. There are no nodes in $y(\tilde{D})$, so we skip Lines 7 through 15 in Algorithm 3. Because $\tilde{D}\in\tilde{\mathcal{N}}$, Lines 16 through 22 are executed: Since the labels at node $\tilde{D}$ are: $p(\tilde{D})=D$, $q(\tilde{D})=2$, $t(\tilde{D})=20$, we enumerate paths in $R^2_{(D,\tilde{D})}$. There is one such path, so $k=2$. Because $q(\tilde{D})=2 < r(D,\tilde{D})=3$, we update $t(\tilde{C})$ to 25 and increase the value of $q(\tilde{D})$ to 3. Node $\tilde{D}$ is inserted back into S. Note that at the beginning of Iteration 7, we have $p(\tilde{D})=D$, $q(\tilde{D})=2$, and $t(\tilde{D})=20$ (shown in Iteration 6); while at the end of this iteration, we have $p(\tilde{D})=D$, $q(\tilde{D})=3$, and $t(\tilde{D})=25$ (shown in Iteration 7).

At the beginning of Iteration 8, we have $S=\{\tilde{C},\tilde{D}\}$. Node $\tilde{C}$ is selected and removed from S. There are no nodes in $y(\tilde{C})$, so we skip Lines 7 through 15 in Algorithm 3. Because $\tilde{C}\in\tilde{\mathcal{N}}$, Lines 16 through 22 are executed: since the labels at node $\tilde{C}$ are: $p(\tilde{C})=C$, $q(\tilde{C})=3$, $t(\tilde{C})=21$, we enumerate paths in $R^3_{(C,\tilde{C})}$. There are 5 paths, so $k=7$. Because $q(\tilde{C})=r(C,\tilde{C})=3$, Lines 18 through 22 are skipped.

At the beginning of Iteration 9, we have $S=\{\tilde{D}\}$. Node $\tilde{D}$ is selected and removed from S. There are no nodes in $y(\tilde{D})$, so we skip Lines 7 through 15 in Algorithm 3. Because $\tilde{D}\in\tilde{\mathcal{N}}$, Lines 16 through 22 are executed: since the labels at node $\tilde{D}$ are: $p(\tilde{D})=D$, $q(\tilde{D})=3$, $t(\tilde{C})=25$, we enumerate paths in $R^3_{(D,\tilde{D})}$. There are 2 paths, so $k=9$. Because $q(\tilde{D})=r(D,\tilde{D})=3$, Lines 18 through 22 are skipped.

At the beginning of Iteration 10, $S=\varnothing$, the algorithm terminates. We can also terminate the algorithm when the number of paths enumerated reaches K.

Proposition 4. *The total time complexity of the algorithm for finding K shortest looping paths in a time-schedule network with constraints on arcs is $O(m\ \log(nr) + Kmr^2\eta)$.*

Proof. The time complexity of the modified Dijkstra's algorithm is $O(m\log(nr))$. The BackTrack function to find K shortest paths takes time $O(Kmr^2\eta)$. Hence, the total time complexity of the algorithm is $O(m\log(nr) + Kmr^2\eta))$. □

5. Computational experiments

The goal of the computational experiments is to assess the empirical performance of our algorithm against existing algorithms that can be adapted to solve the K shortest paths problem with constraints on arcs. In particular, we benchmark against modified versions of the algorithms proposed by [3,6] for finding looping paths in two types of networks. The modifications of the algorithms are:

- Algorithm A is based on [3], which solves the K shortest paths problem in a traffic-light network. In traffic-light networks, for each pair of incoming and outgoing arcs at a node, there is a constraint that consists of a repeated sequence of time windows specifying when this pair of arcs can be traversed. This type of constraint aims at simulating the operation of traffic-light control at an intersection. The authors modify Martin's deletion algorithm [13] in an enlarged network by adding not only artificial nodes and arcs but also the time-windows associated with the arcs. In Algorithm A, we borrow their idea and create an enlarged network by adding not only artificial nodes and arcs but also time-schedules associated with the

Table 3
Illustration of the proposed algorithm on the network shown in Fig. 2.

Iter.	S	u	s	A	B	C	D	$\tilde{C}$	$\tilde{D}$	Enumerate paths and update labels	k
Init.	$\{s\}$		$*, *, 0$	$*, *, \infty$	$*, *, \infty$	$*, *, \infty$	$*, *, \infty$	$*, *, \infty$	$*, *, \infty$		0
1	$\{s\}$	s	$*, *, 0$	$s, 1, 6$	$s, 1, 4$	$*, *, \infty$	$*, *, \infty$	$*, *, \infty$	$*, *, \infty$		0
2	$\{A, B\}$	B		$s, 1, 6$	$s, 1, 4$	$*, *, \infty$	$B, 1, 14$	$*, *, \infty$	$*, *, \infty$		0
3	$\{A, D\}$	A		$s, 1, 6$		$A, 2, 12$	$A, 1, 9$	$*, *, \infty$	$*, *, \infty$		0
4	$\{D, C\}$	D				$A, 2, 12$	$A, 1, 9$	$*, *, \infty$	$D, 2, 20$		0
5	$\{C, \tilde{D}\}$	C				$A, 2, 12$		$C, 2, 17$	$D, 2, 20$		0
6	$\{\tilde{D}, \tilde{C}\}$	$\tilde{C}$						$C, 3, 21$	$D, 2, 20$	$R^2_{(C,\tilde{C})} = \{[s_2; A_7; C_{13}; d; 17]\}$ labels on $\tilde{C}$ are updated.	1
7	$\{\tilde{D}, \tilde{C}\}$	$\tilde{D}$						$C, 3, 21$	$D, 3, 25$	$R^2_{(D,\tilde{D})} = \{[s_2; A_6; D_{10}; d; 20]\}$ labels on $\tilde{D}$ are updated.	2
8	$\{\tilde{C}, \tilde{D}\}$	$\tilde{C}$						$C, 3, 21$	$D, 3, 25$	$R^3_{(C,\tilde{C})} = \{[s_2; A_6; D_{11}; C_{17}; d; 21], [s_2; B_7; A_{10}; C_{17}; d; 21], [s_2; A_7; C_{17}; d; 21], [s_2; A_{10}; C_{17}; d; 21], [s_5; A_{10}; C_{17}; d; 21]\}$ cannot update $t(\tilde{C})$.	7
9	$\{\tilde{D}\}$	$\tilde{D}$						$C, 3, 21$	$D, 3, 25$	$R^3_{(D,\tilde{D})} = \{[s_2; A_6; D_{15}; d; 25], [s_1; B_7; D_{15}; d; 25]\}$ cannot update $t(\tilde{D})$.	9
10	$\varnothing$										

Table 4
Computational results in two types of networks. Run times are reported in milliseconds.

Network size	K	Alg. A		Alg. B		This paper		Reduction (%)	
		Mean	Standard error	Mean	Standard error	Mean	Standard error	Alg. A	Alg. B
Grid network									
	2	5.131	0.151	22.892	0.841	0.870	0.155	83.046	96.200
$n=50*50$	20	12.724	1.200	22.928	0.838	1.619	0.209	87.278	92.940
	100	124.374	13.325	23.053	0.837	4.135	0.465	96.676	82.064
	2	12.368	0.311	74.114	2.986	2.872	0.492	76.777	96.125
$n=75*75$	20	30.797	3.184	74.171	2.990	4.118	0.561	86.627	94.447
	100	218.146	40.680	74.409	2.986	8.776	1.020	95.977	88.205
	2	22.672	0.601	175.901	6.883	6.652	1.089	70.658	96.218
$n=100*100$	20	55.090	6.044	175.911	6.881	8.749	1.174	84.120	95.027
	100	339.717	77.019	176.228	6.876	17.044	1.968	94.983	90.328
Random network									
	2	0.784	0.099	2.806	0.342	0.230	0.034	70.612	91.791
$n=2000$	20	1.399	0.111	2.825	0.342	0.388	0.042	72.300	86.281
	100	5.978	0.399	3.009	0.344	0.577	0.053	90.355	80.836
$n=5000$	2	1.981	0.281	6.646	0.690	0.558	0.108	71.848	91.608
	20	2.270	0.259	6.796	0.695	0.976	0.167	57.029	85.646
	100	5.916	0.389	7.103	0.693	1.360	0.205	77.005	80.846
	2	3.900	0.553	13.450	1.500	1.218	0.294	68.766	90.943
$n=10\,000$	20	4.235	0.534	13.745	1.503	2.089	0.395	50.674	84.803
	100	8.119	0.589	14.271	1.502	2.885	0.466	64.463	79.781

arcs. Then Martin's deletion algorithm is applied to find the K shortest paths.

- Algorithm B is based on [6], which finds K shortest paths in a time-schedule network with constraints on nodes. The authors create a time-expanded representation of the original network. Each node in the original network has several node copies in the time-expanded network, where each copy corresponds to a departure time in the time-schedule at that node. We adapt this algorithm to work with time-schedule network with constraints on arcs by adding artificial nodes for each outgoing arc so that time-schedule constraints on arcs can be modeled by time-schedule constraints on nodes. The algorithm developed in [6] is then applied to find the K shortest paths.

We conduct our computational experiments on two types of networks:

- Grid network: a $p \times p$ grid network consists of $n = p^2$ nodes and these nodes are numbered consecutively from left to right and from top to bottom. Arcs join a given node to the nodes immediately above, below, to the left, and to the right of that node; and
- Random network: a random network consists of n nodes and m arcs. Arcs are generated by joining two distinct nodes chosen randomly from the n nodes. In our tests, we set $m = 10n$.

For each arc (u, v), its cost $c(u, v)$ is uniformly distributed between 1 and 5. We allow elements in $TS(u, v)$ to spread over a wide range to ensure that when we arrive at a node u, our arrival time $t(u)$ is no more than the largest element (or the latest departure time) in $TS(u, v)$, which permits us to traverse arc (u, v) if needed. The list of departure times in $TS(u, v)$ is uniformly distributed between 0 and the length of each window, and the average time interval between successive departure times along each arc is 2.

The three K shortest paths algorithms for time-schedule network with constraints on arcs are implemented in C++, and the same functions are used to perform common operations as far as is

possible. Experiments are performed on a workstation with a Pentium 4 2.5 GHz processor and 8 GB of RAM. For each network size, 100 origin-destination pairs are randomly generated and a series of K shortest paths problems are solved for $K=2$, 20, and 100. The average run times and the standard errors are reported in Table 4. Column 1 shows the numbers of nodes in the test networks. Column 2 shows the value of K for each test problem. Columns 3 through 8 report the run times and standard errors of Algorithms A and B, and the algorithm developed in this paper, respectively. The average run times for all three algorithms go up when the size of the test network and K grow. In addition, the coefficient of variation, that is, the ratio between the standard error and the average run time for each test case is around 10%, suggesting that the performance of these algorithms is relatively stable. Columns 9 and 10 show the reduction in run time with respect to Algorithms A and B, respectively. It is observed that the algorithm developed in this paper runs much faster than the other two algorithms. For most of the large instances, the reduction is over 80%, which clearly demonstrates the superiority of our algorithm.

6. Conclusions

In this paper, we investigate the K shortest paths problem in a time-schedule network with constraints on arcs, a variant of the TCSPP not yet studied in the literature. The complexity of our algorithm is $O(m\ \log(nr) + Kmr^2\eta)$, where $O(m\ \log(nr))$ is the time used by the modified Dijkstra's algorithm to find shortest paths and $O(Kmr^2\eta)$ is the time used to enumerate all paths. Computational experiments show that our algorithm outperforms existing algorithms adapted to solve the same problem. One possible extension of the research is to consider networks containing the two types of time constraints (time-window and time-schedule) on both arcs and nodes simultaneously. This represents a broader and more practical situation in reality.

Acknowledgments

We would like to thank the anonymous referees for their helpful suggestions which allowed us to improve the quality of this paper.

References

[1] Chen YL, Yang HH. Finding the first k shortest paths in a time-window network. Computers and Operations Research 2004;31(4):499–513.
[2] Chen YL, Yang HH. Shortest paths in traffic-light networks. Transportation Research Part B: Methodological 2000;34(4):241–53.
[3] Yang HH, Chen YL. Finding k shortest looping paths in a traffic-light network. Computers and Operations Research 2005;32(3):571–81.
[4] Yang HH, Chen YL. Finding k shortest looping paths with waiting time in a time-window network. Applied Mathematical Modelling 2006;30(5):458–65.
[5] Chen YL, Tang k. Minimum time paths in a network with mixed time constraints. Computers and Operations Research 1998;25(10):793–805.
[6] Chen YL, Tang K. Finding the kth shortest path in a time-schedule network. Naval Research Logistics (NRL) 2005;52(1):93–102.
[7] Martins E, Pascoal M. The k shortest paths problem—a survey. Technical Report, University of Coimbra, Portugal; 2000. URL ⟨http://www.mat.uc.pt/~eqvm/CISUC/COM/claio2000.ps.gz⟩.
[8] Yen J. Finding the k shortest loopless paths in a network. Management Science 1971;17(11):712–6.
[9] Eppstein D. Finding the k shortest paths. SIAM Journal of Computing 1998;28(2):652–73.
[10] Martins E, Pascoal M, Santos J. A new algorithm for ranking loopless paths. Technical Report, Univ. de Coimbra; 1997.
[11] Dreyfus S. An appraisal of some shortest-path algorithms. Operations Research 1969;17(3):395–412.
[12] Shier D. Computation experience with an algorithm for finding the k shortest paths in a network. Journal of Research of the NBS 1974;78:139–64.
[13] Martins E. An algorithm for ranking paths that may contain cycles. European Journal of Operational Research 1984;18(1):123–30.
[14] Azevedo J, Costa MEOS, Madeira JJES, Martins EQV. An algorithm for the ranking of shortest paths. European Journal of Operational Research 1993;69(1):97–106.
[15] Dijkstra E. A note on two problems in connexion with graphs. Numerische Matematik 1959;1:269–71.
[16] Chen YL, Rinks D, Tang K. The first k minimum cost paths in a time-schedule network. Journal of the Operational Research Society 2001;52(1):102–8.
[17] Androutsopoulos K, Zografos K. Solving the k-shortest path problem with time windows in a time varying network. Operations Research Letters 2008;36(6):692–5.