Available online at www.sciencedirect.com
SCIENCE @ DIRECT®
ELSEVIER
Mathematical and Computer Modelling 40 (2004) 1453–1464
MATHEMATICAL AND COMPUTER MODELLING
www.elsevier.com/locate/mcm

The First K Shortest Unique-Arc Walks in a Traffic-Light Network

HSU-HAO YANG*
Institute of Production System Engineering and Management
National Chinyi Institute of Technology
Taiping, 411, Taiwan, Republic of China
yanghh@chinyi.ncit.edu.tw

YEN-LIANG CHEN
Department of Information Management
National Central University
Chung-Li, Taiwan 320, Republic of China
ylchen@mgt.ncu.edu.tw

(Received May 2003; revised and accepted July 2004)

Abstract—In this article, we present an algorithm to find the first K shortest unique-arc walks in a traffic-light network. Each node of the present network is associated with a repeated sequence of different windows to model operations of traffic signals and intersection movements. Unlike conventional simple or looping paths, we refer to the paths in this paper as unique-arc walks because they may include repeated nodes but will exclude repeated arcs. Using the heap structure, we develop an algorithm to find the first K shortest unique-arc walks in time $O(Kr|V|^3|A|)$, where $|V|$ is the number of nodes, $|A|$ is the number of arcs, and r represents the number of different windows associated with a node.

Keywords—Shortest path, Traffic-light, Walk.

1. INTRODUCTION

A classical shortest path problem is concerned with finding the path with the minimum time, distance, or cost from a source node to a destination through a connected network. It is an important problem because of its numerous applications and generalizations in transportation [1], communications [2], and many other areas. Readers are referred to several reviews on the shortest path problem [3–5]. To meet practical needs, a time-constrained shortest path problem generalizes the shortest path problem by incorporating temporal factors. The time window has been a common form of time constraint that assumes that a node can be visited only in a specified time

*This author was supported in part by National Science Foundation Grant No. 90-2416-H-167-003. Author to whom all correspondence should be addressed.
The authors are grateful to the anonymous referees for their helpful comments.

0895-7177/05/$ - see front matter Typeset by $\mathcal{AMS}$-TEX
doi:10.1016/j.mcm.2005.01.004

interval [6–9]. In other words, a time window defines the earliest time and the latest time during which the node is available.

Most shortest path algorithms calculate optimal paths, but do not explicitly consider intersection movements that can be significant in congested street networks [10]. To reflect intersection movements in practice and model operations of traffic signals, Chen and Yang [11] introduced a new kind of time constraint (*traffic-light* constraint) by associating each node with a repeated sequence of different windows. The result of this formulation is that the problem of passing through a number of traffic signals as quickly as possible reduces to a shortest path problem. Since Chen and Yang [11] have developed a polynomial-time algorithm for finding the shortest path in the previous paper, this paper extends their study to find the first K shortest paths in the present network. The motivation of this extension arises from its practical applications. As pointed out by Eppstein [12], one of the reasons to find K shortest paths is that certain constraints may be difficult to define or hard to optimize; a common strategy is to compute several paths and then choose among them by considering the other criteria. Since traffic-light constraints model traffic signals that contain cyclic periods of stop (red periods), we could consider other criteria as well as travel time to choose the optimal path. For example, suppose the *stop times* of two paths are different but the arrival times are the same, which means two paths spend different actual travel times. In this case, we can find paths in ascending order of arrival time first, and then choose a path by taking the length of stop time into account. Another likely criterion is fewer *numbers of stops* that could alleviate the degree of traveling discomfort and be in the interest of vehicles. The last example arises from the fact that we could spend several time-windows at the same traffic signal waiting for the queue to clear. This may not be common for some regions; however, it is a real-life application in major cities such as Taipei City in Taiwan. For example, the *2001 Highway Capacity Manual in Taiwan* [13] indicates that under a normal situation, the approximate discharging rate of a passenger car unit (PCU) per hour is 1950 vehicles. The situation designates seven conditions, one of which is that the intersection is located in a metropolitan area (for example, Taipei City). To know how the discharging rate affects the length of a queue, consider the *2003 Traffic Flow Data* [14] that collected four time intervals in Taipei City: 7 A.M. to 8 A.M., 8 A.M. to 9 A.M., 5 P.M. to 6 P.M., and 6 P.M. to 7 P.M. Take the intersection of MinQuan East Road and RueiGuang Road for example. On the day that the data were collected, the numbers of westbound (from MinQuan East Road) vehicles at the four time intervals were: 2696, 2629, 2225, and 2632. These numbers, according to Dion *et al.* [15, p. 106], suggest that a growing residual queue is to form because the number of vehicles reaching the intersection exceeds the number of vehicles that can be served by the traffic signal.

The result of introducing the queue length as an explicit constraint would be hard to optimize the network model and likely be computationally intractable. Instead, computing several paths can help to choose the path where the queue length is treated as a constraint that is ignored in optimization. All this explains that it is logical to compute not only the optimal path but also a set of alternative ones, upon which we can choose the one while considering other criteria. Moreover, finding a set of paths allows us to perform sensitivity analysis of the optimal solution for various problem parameters [12], such as the length of green or red period of the traffic signals.

To reduce the conflict between accuracy and tractability, in this paper, we assume that travel times between successive time-windows are deterministic in the sense that they will not be affected by traffic conditions. In general, deterministic travel times were extensively used in the conventional shortest path problems and K shortest path problems. In particular, they were also used to compute shortest path for street networks [10] and real road networks [16]. Moreover, according to Fu and Rilett [17], who considered travel times as a stochastic process, the use of shortest path algorithms in dynamic and stochastic traffic networks is incorrect. Nevertheless, they suggested that from a practical perspective, the shortest path algorithms might be acceptable if the change of travel times as a function of time in the network is moderate. (Stochastic travel time is not our concern in this paper.)

In literature, the first K shortest paths found can be members of two major classes:

(1) simple paths (paths without repeated nodes and arcs), and

(2) looping paths (paths with repeated nodes and arcs).

Regardless of the network under consideration, the efforts required to find simple paths appear to be harder than those to find looping paths. In the first class, Yen [18] proposed a very efficient algorithm that finds the first K simple paths in a general network in $O(K|V|^3)$ time, where $|V|$ is the number of nodes. Katoh *et al.* [19] improved the time bound to be $O(K(|A|+|V|\log V))$ for an undirected network, where $|A|$ is the number of arcs. In the second class, Dreyfus [20] developed an efficient algorithm that finds the K shortest paths from one node to each one of the other nodes in the time of $O(K|V|\log|V|)$. Fox [21] gave an algorithm to run in $O(|V|^2+K|V|\log|V|)$ time. Recently, Eppstein [12] used an implicit representation of paths to significantly improve the time bound to be $O(|A|+|V|\log V+KV)$.

In this paper, we will focus on finding efficient paths, where a path P is *efficient* if there does not exist another path P', such that P' is formed by adding some nodes or arcs to P, but with a smaller total time than that of P. Note that the total time of a path in the traffic-light network contains two parts:

(1) the travel time of the arcs, and

(2) the stop time of the nodes waiting for the right direction.

Therefore, two properties arise. First, a simple path may not necessarily be an efficient path, which means that the total time of a path with repeated nodes may be smaller than that of the path without repeated nodes. Let $\langle x,u,y\rangle$ denote a directional route that travels from node x to node u and leaves for node y. Then, the total time of the path (A, B, C, D, E) will be larger than that of the path (A, B, H, B, C, D, E) if the stop time of $\langle A,B,C\rangle$ is longer than that of the sum of $\langle A,B,H\rangle$, $\langle B,H,B\rangle$, and $\langle H,B,C\rangle$. That is, instead of leaving for node C directly from Node B, we can save time by turning to the intermediate Node H. The example suggests that the conventional first K simple paths may no longer be the first K efficient paths because adding repeated nodes could save the travel time. Second, all of the arcs of an efficient path in a traffic-light network must be unique, meaning that the total time of a path without repeated arcs is at least as good as that of the path with repeated arcs. For example, the path $(s,\dots,A,B,C,\dots,A,B,D,\dots,d)$ passes through the arc (A, B) twice. Then, we can generate the other path $(s,\dots,A,B,D,\dots,d)$ with total time as good as the original one by stopping at Node B and waiting for the first right window to leave for Node D. In this case, the first K paths without repeated arcs are efficient because including repeated arcs will increase the travel time. The two properties show that the paths enumerated in this paper are neither simple nor looping. We will refer to this kind of path as *unique-arc walk* in the remainder of the paper to reflect that arcs are unique but nodes can be repeated.

The rest of this paper is organized as follows. In Section 2, we introduce the traffic-light network, develop the algorithm for finding the first K shortest unique-arc walks, and provide the time complexity of the algorithm. Section 3 includes the conclusion and directions for future research.

2. SOLUTION ALGORITHM

2.1. Problem Definition

For convenience, we follow the notations used in Chen and Yang [11]. Let $N=(V_1\cup V_2,A,WL,t,s,d)$ denote a traffic-light network, where V_1 is the node set without window constraints, V_2 represents the node set with window constraints, A is the arc set without multiple arcs and self-loops, $t(u,v)$ is the travel time of arc $(u,v)\in A$. For each node $u\in V_2$, it is associated with a window-list $WL(u)=(ws_u,w_{u,1},w_{u,2},\dots,w_{u,r})$, where ws_u is the starting time of the first window and $w_{u,i}$ is the i^{th} time window of node u for $i=1$ to r. Each window $w_{u,i}$ is associated with a duration $d_{u,i}$ and a set of node-triplets $NT_{u,i}$, where a node-triplet $\langle x, u, y \rangle$ is in $NT_{u,i}$ if visiting node y from node x is allowed in window $w_{u,i}$. Using a repeated sequence to represent windows and assuming $w_{u,0} = w_{u,r}$, $w_{u,(k\times r)+i}$ is equal to $w_{u,i}$ for any nonnegative integers k and i, where $i \le r$. In this context, the sequence of the windows describes the whole phasing of the traffic signals.

Since a Node u in V_1 can be treated as a node in V_2 by associating it with a window of infinite duration and containing all possible node-triplets, we assume that all the nodes are in set V_2 for ease of presentation. Consider Figure 1, where the number beside each arc is the arc's travel time. We also show each node's duration $d_{u,i}$ and node-triplets $NT_{u,i}$ wherever appropriate. For example, the first window of Node C starts at time 3; the duration of window $w_{C,1+2i}$ is two time units and the duration of window $w_{C,2+2i}$ is four time units where i is a nonnegative integer. The triplet $\langle A, C, d \rangle$ is the allowable route in the window $w_{C,1+2i}$, while $\langle B, C, d \rangle$ and $\langle D, C, d \rangle$ are allowable in the window $w_{C,2+2i}$. Therefore, at Node C coming from Node A (or Nodes B, D), we can visit Node d only in the window $w_{C,1+2i}$ (or $w_{C,2+2i}$). By Chen and Yang [11], the shortest unique-arc walk in Figure 1 is (s, A, D, d) with total time 8.

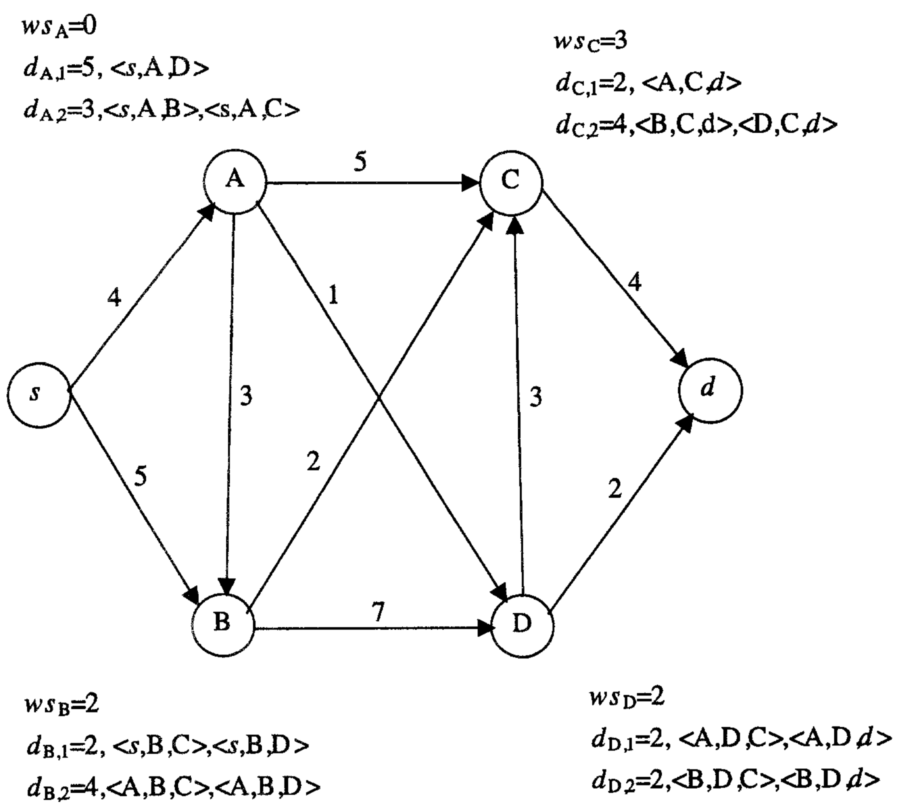


Figure 1. The traffic-light network.

2.2. The Framework of the Algorithm

To find the first K shortest unique-arc walks, our algorithm works as follows. Let P_c be the set of all the walks from s to d in N. Initially, we find the first shortest unique-arc walk $P_1 = (s = v^0, v^1, \dots, v^m = d)$. Then, $P_c - \{P_1\}$ is the set of walks containing all the walks in P_c, but P_1. Define $P_c^{(i)}$, $i = 1, 2, \dots, m$, as the subset of the walks in $P_c - \{P_1\}$ that includes the subwalk $(v^i, v^{i+1}, \dots, v^m = d)$ but excludes the arc, (v^{i-1}, v^i). In this context, we define that for $P_c^{(i)}$, $(v^i, v^{i+1}, \dots, v^m = d)$ is the *in-subwalk* and (v^{i-1}, v^i) is an *out-arc*. It can be easily verified that $P_c - \{P_1\}$ can be partitioned into m disjoint walk subsets $P_c^{(1)}$, $P_c^{(2)}, \dots, P_c^{(m)}$.

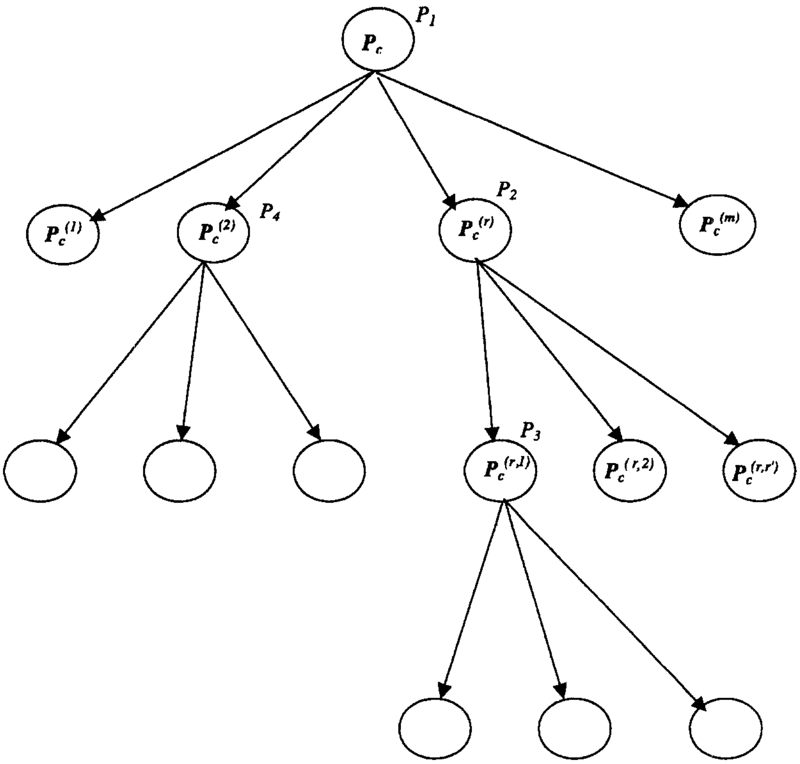


Figure 2. The procedure to compute the first K shortest unique-arc walks.

Let P_2 denote the second shortest unique-arc walk and is in $P_c^{(r)}$. Then, we partition $P_c^{(r)} - \{P_2\}$ into disjoint subsets at the same way we partition $P_c - \{P_1\}$. The subsets obtained from the partitioning of $P_c^{(r)} - \{P_2\}$, together with $P_c^{(1)}$, $P_c^{(2)}, \dots, P_c^{(r-1)}$, $P_c^{(r+1)}, \dots, P_c^{(m)}$, constitute a partition of $P_c - \{P_1, P_2\}$. Following the same procedure, we can generate the walks successively in nondecreasing order. The procedure is illustrated in Figure 2, where each node denotes a subset and the label inside the node is the name of the subset. Furthermore, the arcs emanating from a node denote a partition of this walk set into different subsets, and the walk name, say P_4, beside a node $P_c^{(2)}$ indicates that the walk P_4 is found from the walk set $P_c^{(2)}$.

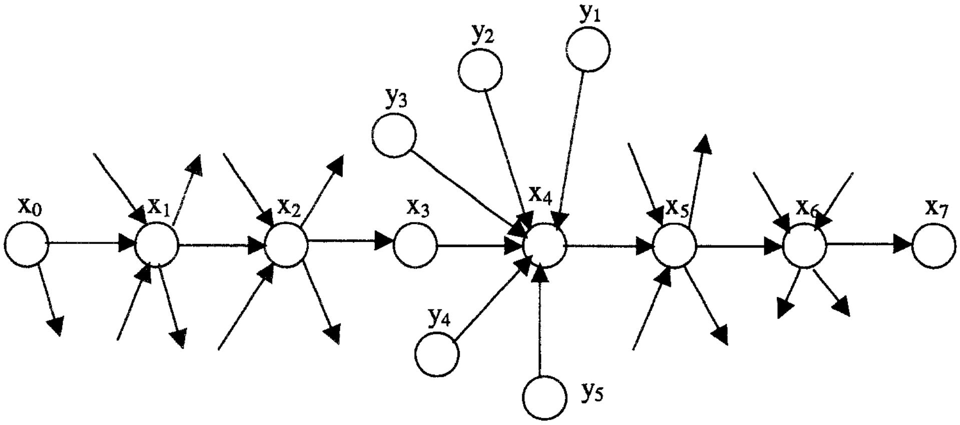
Recall that all of the walks in the walk set $P_c^{(i)}$, for $1 \le i \le m$, contain an in-subwalk from node v^i to d and exclude an out-arc (v^{i-1}, v^i). Suppose P_2 is in $P_c^{(r)}$, and let P_2 be denoted by $(s = u^0, u^1, \dots, u^{r'} = v^r, v^{r+1}, \dots, v^m = d)$. The walk set $P_c^{(r)} - \{P_2\}$ can be further partitioned into r' disjoint subsets. We define $P_c^{(r,i)}$ as the subset of walks in $P_c^{(r)} - \{P_2\}$ with the in-subwalk $(u^i, u^{i+1}, \dots, u^{r'} = v^r)$ and without the out-arc, (u^{i-1}, u^i). By including the original in-subwalk and excluding the out-arc of $P_c^{(r)}$, $P_c^{(r,i)}$ is the subset of walks with the in-subwalk $(u^i, u^{i+1}, \dots, u^{r'} = v^r, v^{r+1}, \dots, v^m = d)$ and without the out-arcs (u^{i-1}, u^i) and (v^{r-1}, v^r). Repeatedly applying the procedure leads to the following property.

PROPERTY 1. Let P be a walk subset in the partition of $P_c - \{P_1, P_2, \dots, P_z\}$. Then, all the walks in P contain an in-subwalk from a node u^* to d and exclude an out-arc set.

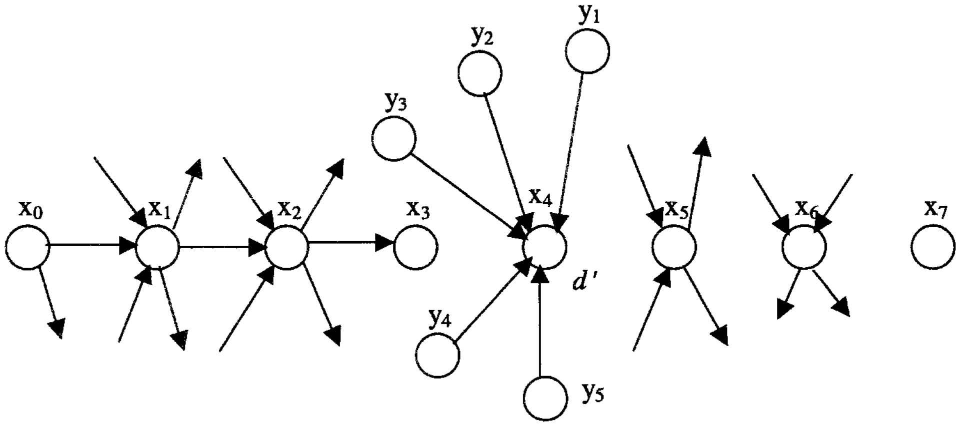
After the partition of walk subset, we need to find the shortest unique-arc walk from s to d in the walk subset that includes a given in-subwalk and excludes a given out-arc set. For ease of presentation, let P^{in} and A^{out} denote the in-subwalk and out-arc set, and we refer to the walk satisfying P^{in} and A^{out} as the constrained walk.

2.3. How to Find the Shortest Constrained Walk

Given N, our algorithm constructs a network $N' = (V', A', s', d')$ so that the shortest walk P^* from $s' = s$ to $d' = u^*$ in N' followed by P^{in} forms a shortest constrained walk from s to d in N.



(a). A shortest constrained walk in N.



(b). Transform N into N'.

Figure 3.

N' is constructed as follows.

(1) Remove all of the arcs in P^{in} from N because P^* does not pass through these arcs.
(2) Remove all of the arcs in A^{out} from N.
(3) Set $s' = s$ and $d' = u^*$.

For example, Figure 3a shows a shortest constrained walk from x_0 to x_7, where $x_0 = s' = s$, $x_4 = d'$, $x_7 = d$, $P^{\text{in}} = (x_4, x_5, x_6, x_7)$ and $A^{\text{out}}\{(x_3, x_4)\}$. By the transformation above, we can construct the network N' similar to Figure 3b, where all arcs in P^{in} and A^{out} are removed and we need to find the shortest unique-arc walk from x_0 to x_4.

At this point, we make the following observations.

(1) Since the constructed network N' is a traffic-light network, the shortest path algorithm such as Dijkstra's algorithm [22] cannot be applied. Instead, the shortest unique-arc walk algorithm of Chen and Yang [11] should be used to find the shortest walk from $s' = s$ to $d' = u^*$.
(2) Note that the shortest walk P^* from s' to d' in N' followed by P^{in} may not necessarily form the shortest constrained walk from s to d. Consider Figure 3 again. Suppose there is one shortest walk $P^* = (x_0, \dots, y_2, x_4)$ with minimum arrival time 30 and the other walk $P^{\&} = (x_0, \dots, y_4, x_4)$ with a larger arrival time 33. Further, suppose that direction $\langle y_2, x_4, x_5 \rangle$ does not allow us to leave for node x_5 from x_4 until time 40, while direction $\langle y_4, x_4, x_5 \rangle$ does at time 35. As a result, the walk $P^{\&}$ can leave for node x_5 earlier than the walk P^*, because the earliest time to leave for node x_5 from node x_4 through arc (y_2, x_4) is 40 while through arc (y_4, x_4) is 35. The example indicates that what really matters in a traffic-light network is the departure time, rather than the arrival time. Compared

to the conventional shortest path problem, where earlier arrival always leads to earlier departure, we focus on finding a walk from s' to d' in N' so that we leave node d' the earliest.

(3) Let pred denote the second-to-the-last node in the walk $P^{\&}$ from s' to d' and suc denote the second node in the walk P^{in}. Then, the earliest time to leave for node suc from Node d' at time t coming from node pred can be denoted as earliest(pred, d', suc, t). To determine this value, we need the following data: the time reaching Node d', the time window list associated with Node d', and the routing direction $\langle$pred, d', suc$\rangle$. Readers are referred to [11, Section 2.1] for the computational procedure.

(4) For all of the nodes pred preceding Node d' in N', we can compute the earliest time to arrive at Node d' through arc (pred, d') by the algorithm of Chen and Yang [11]. Let this value be denoted as arrived(pred, d'). Then, the time to leave for node suc from Node d' with the preceding node pred is determined as earliest(pred, d', suc, arrived(pred, d')). Among all of the nodes preceding Node d', we choose the node through which we can leave Node d' as early as possible. Therefore, the walk $P^{\&}$ from s' to d' in N' whose leaving time for node suc, equal to the following value, is the subwalk that forms the shortest constrained walk from s to d in N.

$$\min_{\text{for all pred}} \text{earliest}(\text{pred}, d', \text{suc}, \text{arrived}(\text{pred}, d')) \tag{1}$$

By the preceding observations, we derive the following conclusion.

THEOREM 1. *The shortest walk from s to d in the walk subset with P^{in} and A^{out} is the combination of the shortest walk $P^{\&}$ from s to d' in N' followed by P^{in} from d' to d, where $P^{\&}$ satisfies relation (1) shown above.*

To find the shortest walk from s to $d$in N subject to P^{in} and A^{out}, we develop the following algorithm.

THE SCW ALGORITHM.

1. Transform N into N'.
2. Find the values arrived(pred, d') for all of the nodespread preceding Node d' in N'.
3. Compute the value of earliest(pred, d', suc, arrived(pred, d')) for all pred preceding d'.
4. Choose the walk $P^{\&}$ from s to d' in N' satisfying the relation (1).
5. Obtain the shortest constrained walk in N subject to P^{in} and A^{out} by appending P^{in} to $P^{\&}$.

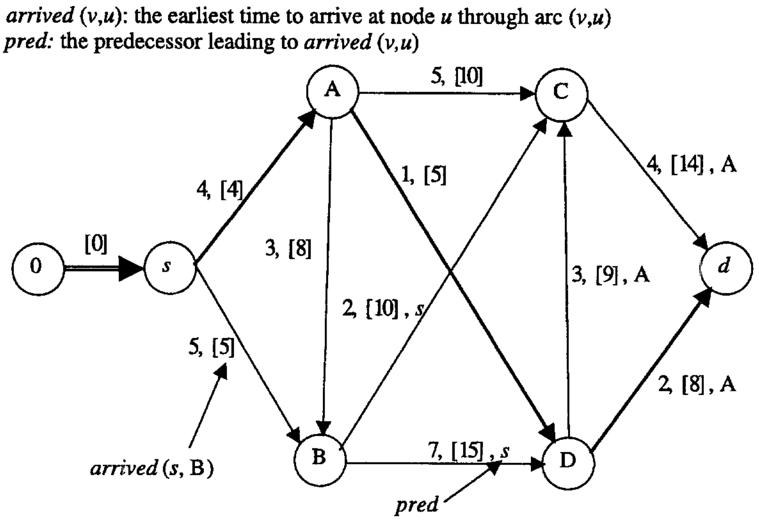


Figure 4. The first shortest unique-arc walk in the network.

To illustrate the SCW algorithm, reconsider Figure 1. After applying the algorithm of Chen and Yang [11], we obtain the network shown in Figure 4, where $P_1 = (s, A, D, d)$ with total time 8, and the number in the square bracket beside each arc is the earliest arrival time through the arc. In addition, we use pred to specify the predecessor that leads to arrived(v, u) if more than one node preceding Node v. Observe that arrived$(C, d)(= 14)$ verifies that even though arrived$(A, C)(= 10)$ arrives at Node C later than arrived(D, C)(=9) does, the direction $\langle A, C, d\rangle$ allows us to leave for Node d earlier. The set of walks $P_c - \{P_1\}$ can be partitioned into three disjoint walk subsets as follows.

$P_c^{(1)}$: The walks with $P^{\text{in}} = (A, D, d)$ and without $A^{\text{out}} = (s, A)$.
$P_c^{(2)}$: The walks with $P^{\text{in}} = (D, d)$ and without $A^{\text{out}} = (A, D)$.
$P_c^{(3)}$: The walks without $A^{\text{out}} = (D, d)$.

To show how to find the shortest constrained walk in a given walk set, consider $P_c^{(2)}$. By removing the arcs in A^{out} and P^{in}, the resulting network is shown in Figure 5. Since the shortest walk in Figure 5 is the walk (s, B, D) with arrived$(B, D) = 15$ and earliest$(B, D, d,$ arrived$(B, D)) =$ earliest$(B, D, d, 15) = 16$, we obtain the shortest constrained walk (s, B, D, d) with total time 18.

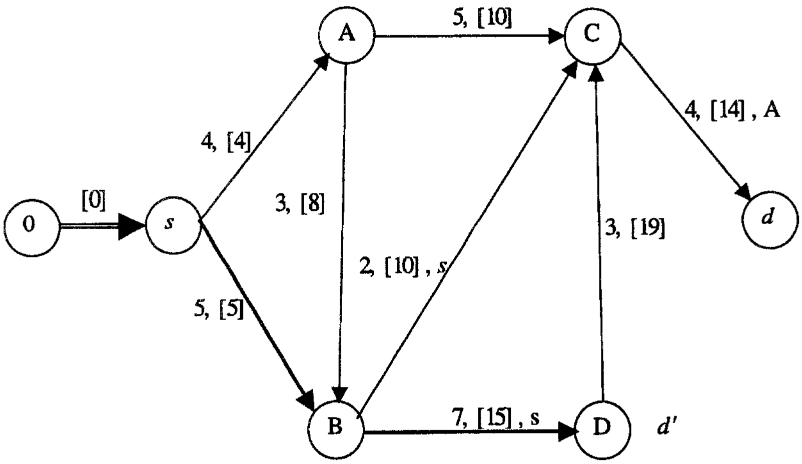


Figure 5. Subset $P_c^{(2)}$ with $A^{\text{out}} = \{(A, D)\}$ and $P^{\text{in}} = (D, d)$.

LEMMA 1. *The time complexity of SCW algorithm is* $O\,(r|V|^3)$, *where* $|V|$ *is the number of nodes of the network and* r *is the number of different windows of a node.*

PROOF. Since every arc and node will be examined and processed at most one time in transforming the network from N into N', the time for Step 1 is $O(|A| + |V|)$. Steps 2 and 3 can be done in time $O(r|V|^3)$ owing to Chen and Yang [11]; the time to perform Steps 4 and 5 is negligible. Therefore, the total time complexity is $O(r|V|^3)$.

2.4. How to Find the First K Shortest Unique-Arc Walks

The procedure shown in Figure 2 may generate a substantial number of walks and their corresponding constraints after a number of iterations. To manage this problem, we use the heap structure [23], where the times to find and remove the minimum element or to insert a new element are all $O(\log n)$ for a heap with n elements. Let each element in the heap represent an enumerated walk and each element is associated with its in-subwalk and out-arc set. We develop the following algorithm to find the first K shortest unique-arc walks, where the set Q is stored by a heap structure.

THE KSCW ALGORITHM.

Find P_1, the first shortest unique-arc walk.

Let in-subwalk$(P_1) = \phi$ and out-arc$(P_1) = \phi$, where ϕ represents an empty walk or set.

Store the element of P_1 into Q.
For $w = 1$ to K
  Select the shortest unique-arc walk P from Q, output it as the w^{th} shortest walk, and remove P from Q.
  Let P' be the subwalk of P satisfying $P = P' \oplus$ in-subwalk(P), where $\oplus$ is the operator to connect two subwalks.
  Let the number of arcs in P' be m, and $P'(i,j)$ denote the subwalk from the i^{th} arc to the j^{th} arc of P'.
  Partition the walk set of P into m disjoint walk subsets. The in-subwalk and out-arc set of the i^{th} walk subset, where $1 \le i \le m$, can be obtained by:
    the out-arc set is $\{P'(i,i)\} \cup$ out-arc(P),
    the in-subwalk is $P'(i+1,m) \oplus$ in-subwalk(P).
  Use SCW algorithm to find the shortest constrained unique-arc walk for each walk subset.
  For each walk subset, if there exists a shortest constrained unique-arc walk, then store it and its associated in-subwalk and out-arc set into Q.

LEMMA 2. *The time complexity of the KSCW algorithm is $O(Kr|V|^3|A|)$, where $|A|$ is the number of arcs of the network.*

PROOF. The most time-consuming part of obtaining the next shortest unique-arc walk is the partition of the subset of walks containing the one that is most recently found into disjoint walk subsets. By definition, there are at most $|A|$arcs in an unique-arc walk; hence, there are at most $|A|$ disjoint subsets in each partition. For each disjoint subset, we find the shortest unique-arc walk by the *SCW* algorithm, which requires $O(r|V|^3)$. Hence, the total time to find the next walk is $O(r|V|^3|A|)$. If K walks are enumerated, the time is $O(Kr|V|^3|A|)$.

EXAMPLE 2. Consider Figure 1 again. In Step 1, we find $P_1 = (s,A,D,d)$ as shown in Figure 4. Walk P_1 with in-subwalk$(P_1) = \phi$ and out-arc$(P_1) = \phi$ are stored into the heap.

In the first cycle of Step 2, where $w = 1$, the walk P removed from the heap Q is P_1. Since in-subwalk$(P_1) = \phi$, $P' = (s,A,D,d)$; we partition the walk set of P into three walk subsets as follows.

The first subset $P_c^{(1)}$ (Figure 6) has $A^{\text{out}} = \{(s,A)\}$ and $P^{\text{in}} = (A,D,d)$.
The second subset $P_c^{(2)}$ (Figure 5) has $A^{\text{out}} = \{(A,D)\}$ and $P^{\text{in}} = (D,d)$.
The third subset $P_c^{(3)}$ (Figure 7) has $A^{\text{out}} = \{(D,d)\}$ and $P^{\text{in}} = \phi$.
For each of these three subsets, we find their shortest constrained unique-arc walks as follows.
There is no such walk in $P_c^{(1)}$.

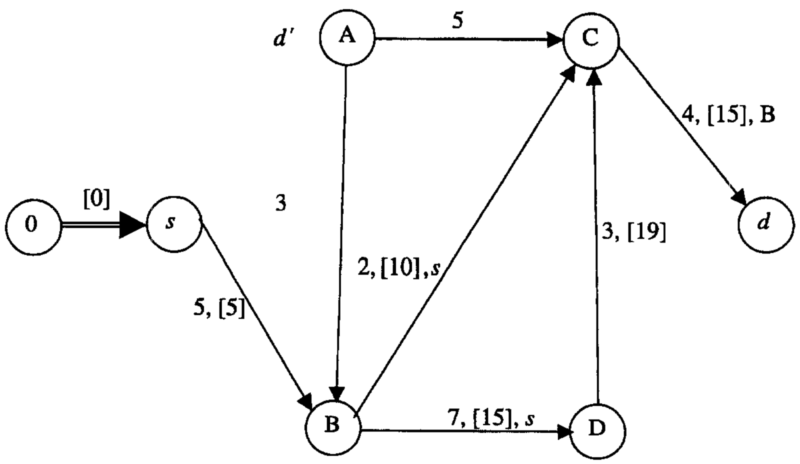


Figure 6. Subset $P_c^{(1)}$ with $A^{\text{out}} = \{(s,A)\}$ and $P^{\text{in}} = (A,D,d)$.

Table 1. Summary of *KSCW* algorithm for Example 2.

w	Shortest Constrained Walk P	$P^{\text{in}}(P)$	$A^{\text{out}}(P)$	Q		
				Walks	P^{in}	A^{out}
1	$(s, \text{A}, \text{D}, d) = 8$	ϕ	ϕ	$(s, \text{B}, \text{D}, d) = 18$ $(s, \text{A}, \text{C}, d) = 14$	(D,d) ϕ	{(A,D)} {(D,d)}
2	$(s, \text{A}, \text{C}, d) = 14$	ϕ	{(D,d)}	$(s, \text{B}, \text{D}, d) = 18$ $(s, \text{A}, \text{D}, \text{C}, d) = 15$	(D,d) (C,d)	{(A,D)} {(A,C),(D,d)}
3	$(s, \text{A}, \text{D}, \text{C}, d) = 15$	(C, d)	{(A,C),(D,d)}	$(s, \text{B}, \text{D}, d) = 18$ $(s, \text{B}, \text{D}, \text{C}, d) = 23$ $(s, \text{B}, \text{C}, d) = 15$	(D, d) (D,C,d) (C,d)	{(A,D)} {(A,D),(A,C),(D,d)} {(D,C),(A,C),(D,d)}
4	$(s, \text{B}, \text{C}, d) = 15$	(C,d)	{(D,C),(A,C),(D,d)}	$(s, \text{B}, \text{D}, d) = 18$ $(s, \text{B}, \text{D}, \text{C}, d) = 23$ $(s, \text{A}, \text{B}, \text{C}, d) = 16$	(D,d) (D,C,d) (B,C,d)	{(A,D)} {(A,D),(A,C),(D,d)} {(s,B),(D,C),(A,C),(D,d)}
5	$(s, \text{A}, \text{B}, \text{C}, d) = 16$	(B,C,d)	{(s,B),(D,C),(A,C),(D,d)}	$(s, \text{B}, \text{D}, d) = 18$ $(s, \text{B}, \text{D}, \text{C}, d) = 23$	(D,d) (D,C,d)	{(A,D)} {(A,D),(D,d)}
6	$(s, \text{B}, \text{D}, d) = 18$	(D,d)	{(A,D)}	$(s, \text{B}, \text{D}, \text{C}, d) = 23$ $(s, \text{A}, \text{B}, \text{D}, d) = 19$	(D,C,d) (B,D,d)	{(A,D),(D,d)} {(s,B),(A,D)}
7	$(s, \text{A}, \text{B}, \text{D}, d) = 19$	(B,D,d)	{(s,B),(A,D)}	$(s, \text{B}, \text{D}, \text{C}, d) = 23$	(D,C,d)	{(A,D),(D,d)}
8	$(s, \text{B}, \text{D}, \text{C}, d) = 23$	(D,C,d)	{(A,D),(D,d)}	$(s, \text{A}, \text{B}, \text{D}, \text{C}, d) = 24$	(B,D,C,d)	{(s,B),(A,D),(D,d)}
9	$(s, \text{A}, \text{B}, \text{D}, \text{C}, d) = 24$	(B,D,C,d)	{(s,B),(A,D),(D,d)}	ϕ		

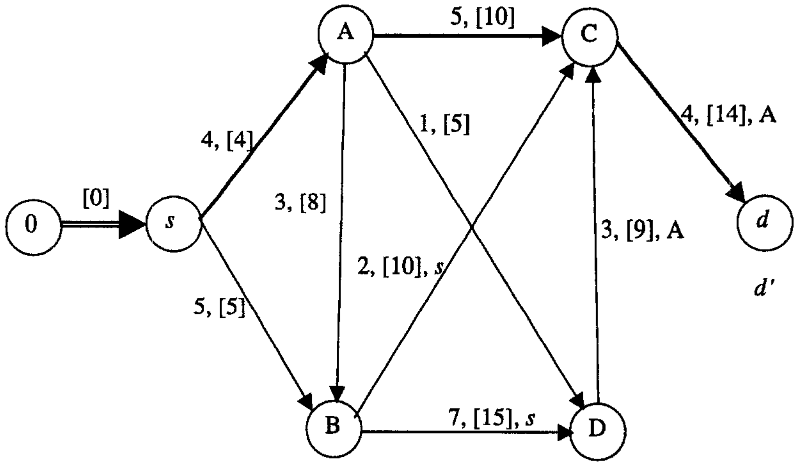


Figure 7. Subset $P_c^{(3)}$ with $A^{\text{out}} = \{(\text{D}, d)\}$ and $P^{\text{in}} = \theta$.

The shortest constrained unique-arc walk in $P_c^{(2)}$ is (s, B, D, d) with time 18.

The shortest constrained unique-arc walk in $P_c^{(3)}$ is (s, A, C, d) with time 14.

Hence, walk subsets $P_c^{(2)}$ and $P_c^{(3)}$ and their related information are stored into the heap Q; the result of the execution of the algorithm is summarized in Table 1. To see how finding K walks may help to choose a route when the other criteria are involved, consider the walks $p_3 =$ (s, A, D, C, d) and $p_4 =$ (s, B, C, d). The travel time of each of these two walks is 15; the stop time of p_3 is 3, while the stop time of p_4 is 4. As we described in Section 1, although two walks reach the destination at the same time, choosing which one walk could depend on the length of stop time.

3. CONCLUSIONS

This paper studies finding the first K shortest unique-arc walks in a traffic-light network that models operations of traffic signals and intersection movements. The name unique-arc walk derives from the fact that the walk found in this paper may include repeated nodes but will exclude repeated arcs. The major contribution of the paper is that we have developed an algorithm of polynomial time to find the first K shortest unique-arc walks in the present network. This paper has several possible extensions. First, we can consider the situation where we choose to stop for some time and leave later. Note that the length of stop time can vary widely depending on the decision scenario. Dealing with this issue not only complicates the enumeration of all possible walks, but also raises the question as to whether the walks remain unique-arc. In addition, the distinction between visiting repeated nodes and stopping at the same node should be clearly made even if the walk is no longer simple. Finally, we may consider criteria jointly, for example, minimization of total travel time subject to total stop time, or minimization of a weighted sum of total travel time and total stop time.

REFERENCES

1. A.J. Swersey and W. Ballard, Scheduling school buses, *Management Science* **30**, 844–853, (1984).
2. Y.-L. Chen and Y.-H. Chin, The quickest path problem, *Computers & Operations Research* **17**, 153–161, (1990).
3. L.D. Bodin, B.L. Golden, A.A. Assad and M.O. Ball, Routing and scheduling of vehicles and crews: The state of the art, *Computers & Operations Research* **10**, 63–211, (1982).
4. N. Deo and C. Pang, Shortest path algorithms: Taxonomy and annotation, *Networks* **14**, 275–323, (1984).
5. B.L. Golden and T.L. Magnanti, Deterministic network optimization: A bibliography, *Networks* **7**, 149–183, (1977).
6. M. Desrochers and F. Soumis, A generalized permanent labelling algorithm for the shortest path problem with time windows, *INFOR* **26** (3), 191–212, (1988).

7. M. Desrochers, J. Desrosiers and M.M. Solomon, A new optimization algorithm for the vehicle routing problem with time windows, *Operations Research* **40**, 342–354, (1992).
8. Y. Dumas, J. Desrosiers, E. Gelinas and M.M. Solomon, An optimal algorithm for the traveling salesman problem with time windows, *Operations Research* **43**, 367–371, (1995).
9. N. Kohl and O.B.G. Madsen, An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation, *Operations Research* **45**, 395–406, (1997).
10. A.K. Ziliaskopoulos and H.S. Mahmassani, A note on least time path computation considering delays and prohibitions for intersection movements, *Transportation Research Part B* **30** (5), 359–367, (1996).
11. Y.-L. Chen, and H.-H. Yang, Shortest paths in traffic-light networks, *Transportation Research Part B* **34**, 241–253, (2000).
12. D. Eppstein, Finding the k shortest paths, *SIAM Journal on Computing* **28**, 652–673, (1998).
13. Institute of Transportation, *2001 Highway Capacity Manual in Taiwan*, Ministry of Transportation of Communications, Taipei, Taiwan, ROC, (2001).
14. Department of Transportation, *2003 Traffic Flow Data*, Taipei City Government, Taipei, Taiwan, ROC, (2003).
15. F. Dion, H. Rakha and Y.-S. Kang, Comparison of delay estimates at under-saturated and over-saturated pre-timed signalized intersections, *Transportation Research Part B* **38** (2), 99–122, (2004).
16. F.B Zhan and C.E. Noon, Shortest path algorithms: An evaluation using real road networks, *Transportation Science* **32** (1), 65–73, (1998).
17. L. Fu and L.R. Rilett, Expected shortest paths in dynamic and stochastic traffic networks, *Transportation Research Part B* **32** (7), 499–516, (1998).
18. J.Y. Yen, Finding the k shortest loopless paths in a network, *Management Science* **17**, 712–716, (1971).
19. N. Katoh, T. Ibaraki and H. Mine, An efficient algorithm for k shortest simple paths, *Networks* **12**, 411–427, (1982).
20. S. Dreyfus, An appraisal of some shortest path algorithms, *Operations Research* **17**, 395–412, (1969).
21. B.L. Fox, Data structures and computer science techniques in operations research, *Operations Research* **26**, 686–717, (1978).
22. E. Dijkstra, A note on two problems in connection with graphs, *Numeriche Mathematics* **1**, 269–271, (1959).
23. M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *Journal of ACM* **34**, 596–615, (1987).