Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Applied Mathematical Modelling 30 (2006) 458–465
APPLIED MATHEMATICAL MODELLING
www.elsevier.com/locate/apm

Finding K shortest looping paths with waiting time in a time–window network

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

[a] *Institute of Production System Engineering and Management, National Chin-Yi Institute of Technology, Taiping 411, Taiwan, ROC*

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

Received 1 August 2004; received in revised form 1 May 2005; accepted 25 May 2005
Available online 19 July 2005

Abstract

A time-constrained shortest path problem is a shortest path problem including time constraints that are commonly modeled by the form of time windows. Finding K shortest paths are suitable for the problem associated with constraints that are difficult to define or optimize simultaneously. Depending on the types of constraints, these K paths are generally classified into either simple paths or looping paths. In the presence of time–window constraints, waiting time occurs but is largely ignored. Given a network with such constraints, the contribution of this paper is to develop a polynomial time algorithm that finds the first K shortest looping paths including waiting time. The time complexity of the algorithm is $O(rK^2|V_1|^3)$, where r is the number of different windows of a node and $|V_1|$ is the number of nodes in the original network.

Keywords: Shortest looping path; Time–window

* Corresponding author. Tel.: +886 4 23924505x6015; fax: +886 4 23921742.
E-mail addresses: yanghh@ncit.edu.tw (H.-H. Yang), ylchen@mgt.ncu.edu.tw (Y.-L. Chen).
[1] Tel.: +886 3 4267266; fax: +886 3 4254604.

0307-904X/$ - see front matter
doi:10.1016/j.apm.2005.05.005

1. Introduction

A shortest path problem (SPP) deals with finding a path with minimum time, distance, or cost from a source node to a destination node through a connected network. Including time constraints in a shortest path problem leads to the development of a time-constrained shortest path problem (TCSPP) that can be seen as a generalization of a SPP. These constraints are commonly modeled using time windows where a node can be visited only in a specified time interval [1–3]. Two types of time windows are widely used: *hard* time–window and *soft* time–window. In a hard time–window, the solution is infeasible if time constraints are violated [4,5]. In contrast, in a soft time–window, the solution remains feasible but a cost penalty will be incurred if time–window constraints are violated [6].

The extension from finding one shortest path to K shortest paths appears to be natural because the problem may be associated with some constraints that are difficult to define or to optimize simultaneously. Under the circumstances, a common way is to compute several paths, rank the paths according to some criteria, and then determine the paths that meet the criteria. Depending on the types of constraints, the paths are generally classified into two classes: (1) simple paths (paths without repeated nodes and arcs), and (2) looping paths (paths with repeated nodes and arcs). These two classes of paths have long been studied together because they are not only similar (from one path to K paths) in general but also complementary (simple versus looping) in particular.

Given a network with time windows, the objective of this paper is to find the first K shortest looping paths that include *waiting* time. Waiting time occurs in a TCSPP but is largely ignored in the literature. To know why we consider waiting time, consider a path route (s, A, D, d) shown in Fig. 1 [7], where the number on an arc is the travel time and nodes are associated with sets of windows. Without considering waiting time, the total time of this path route is 10 because we reach and leave a node at the same time. If the waiting time is involved, however, two situations occur. First, we have to wait for the next available window if the arrival time is not within a window. Second, if the arrival time is within a window, we have two choices: (1) leave immediately, or (2) wait. Under normal conditions, we will eventually leave a node after a certain number of windows.

Consider the path $P = (s, A, D, d)$ in Fig. 1 to see how the waiting time may cause a path to be represented in different ways. Let x_t be the departure time of node x in the path at time t. Then, we have two possible ways to represent this path: (s_0, A_4, D_8, d_{10}) and (s_0, A_5, D_8, d_{10}). Because of these different representations, we will refer to *path route* simply as a listing of nodes, whereas *path* as a listing of nodes together with their departure times. That is, a set of paths may correspond to a

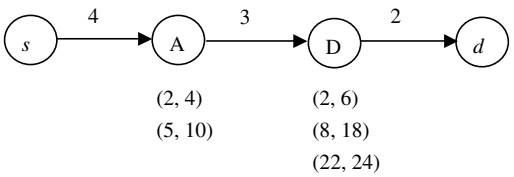

Fig. 1. A path route in a time–window network.

path route. In this case, two paths, i.e., (s_0, A_4, D_8, d_{10}) and (s_0, A_5, D_8, d_{10}), correspond to the path route (s, A, D, d).

With respect to finding the first K simple paths, Yen's algorithm [8] ran in $O(K|V|^3)$ time for a general network, where $|V|$ is the number of nodes. Katoh et al. [9] improved the time to be $O(K(|A| + |V|\log|V|))$, where $|A|$ is the number of arcs. Hadjiconstantinou and Christofides [10] presented an efficient implementation for finding a large number of simple paths based on Katoh et al. [9]. Finding K simple paths for a network with time–schedule or time–window constraints can be found in [11,7]. As for finding K looping paths, Dreyfus' algorithm [12] ran in $O(K|V|\log|V|)$ time given a shortest tree is available. Fox [13] gave an algorithm to run in $O(|V|^2 + K|V|\log|V|)$ time. Using a concept of path deletion, Martins [14] developed an algorithm with the worst time to be $O(K^3|V|)$. Later, this time was improved to be $O(K^2|A|)$ by Azevedo et al. [15] who used an efficient implementation. Martins' path deletion was generalized by Villeneuve and Desaulniers [16] who developed the solution for the shortest path problem with forbidden paths. The algorithm of Azevedo et al. [15] formed the basis of Yang and Chen [17] who developed a polynomial time algorithm for finding K shortest looping paths in a traffic–light network. Eppstein [18] used an implicit representation of paths to run in $O(|A| + |V|\log|V| + K|V|)$ time. On the basis of Eppstein's algorithm, Jiménez and Marzal [19] proposed a modified version that improved computational performance in practice. For a traffic–light network, Yang and Chen [20] developed a polynomial time algorithm to find K shortest "unique-arc walks," which means that the path may include repeated nodes but will exclude repeated arcs. Other recent developments related to K shortest paths can be found in [21,22].

The remainder of this paper is organized as follows. In Section 2, we describe the problem and develop an algorithm to find the first K shortest looping paths in the underlying network. We also provide the time complexity of the algorithm in this section. We present the conclusion in Section 3.

2. The problem and solution

In the following, we define the time–window network in Section 2.1. In Section 2.2 and Section 2.3, we introduce the algorithms of Chen and Yang [7] as well as Yang and Chen [17] that form the basis of our algorithm to solve the problem. We present our algorithm and show its complexity in Section 2.4.

2.1. Problem definition

Let $N = (V_1 \cup V_2, A, WL, t, s, d)$ denote a time–window network, where V_1 is the node set without window constraints, V_2 is the node set with window constraints, A is the arc set without multiple arcs and self-loops, $t(u, v)$ is the travel time of arc $(u, v) \in A$. For each node $u \in V_2$, it is associated with a window-list $WL(u) = (ws_u, w_{u,1}, w_{u,2}, \ldots, w_{u,r})$, 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 $w_{u,i}$. If we represent windows using a repeated sequence and by assuming $w_{u,0} = w_{u,r}$, we have the relationship that $w_{u,(k\times r)+i} = w_{u,i}$ for any non-negative integers k and i, where $i \leqslant r$.

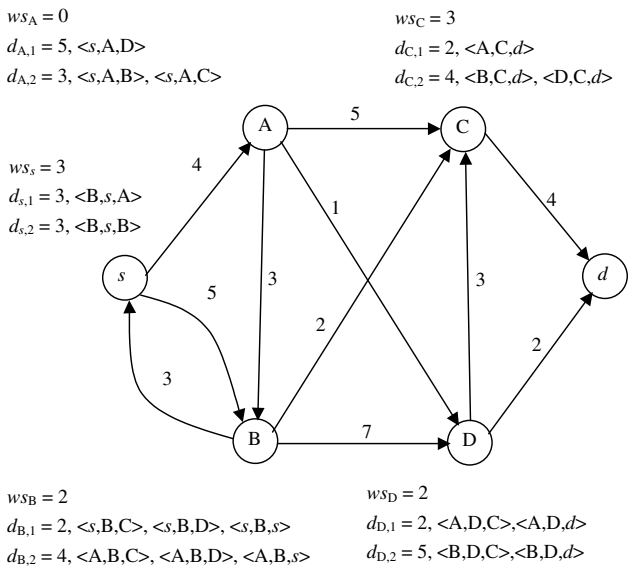

Fig. 2. A time–window network.

Because a node u in V_1 can be regarded as a node in V_2 by associating it with a window of infinite duration and containing all possible node-triplets, we assume that all the nodes are in the set V_2 for ease of presentation. Consider Fig. 2 that illustrates the time–window network, where the number next to each arc is the arc's travel time. We also show each node's, say u's, duration $d_{u,i}$ and its node-triplets $NT_{u,i}$ wherever appropriate. For example, the first window of node C starts at time 3; the duration of window $w_{C,1+2i}$ is 2 and the duration of window $w_{C,2+2i}$ is 4 where i is a nonnegative integer. The triplet $\langle A, C, d\rangle$ is the allowable route in window $w_{C,1+2i}$; $\langle B, C, d\rangle$ and $\langle D, C, d\rangle$ are allowable in window $w_{C,2+2i}$. Therefore, if at node C from node A, we can visit node d only in window $w_{C,1+2i}$, and so on. Chen and Yang [23] developed an algorithm that labels arcs rather than nodes to find the shortest path as (s, A, D, d) with total time 11. Because the algorithm labels arcs, each arc (u, v) in A is associated with a label *arrived* (u, v) to denote the earliest time to arrive at node v through arc (u, v).

2.2. Finding paths corresponding to a path route

Recall that a set of different paths may correspond to a path route because of waiting time involved. Assume that we need to find the first 30 paths. To find these paths, we find the first path route P_1 with total time T_1. Given P_1 and T_1, if we can find 15 paths with the same route but different waiting times, we output these paths and continue to find the second path route, say P_2. Given P_2, assume we find another 19 paths. Because this number has satisfied our requirement, we output only the first 15 paths as the solution. Doing this way, we will successfully find the first 30 paths in the network. We have just roughly described the algorithm in [7] that constructs a graph (called *related-graph*) and then uses this graph to find all corresponding paths (called *related-paths*) given a path route P and its total time T. In addition, the related-graph is used to find the path route P's next total time (called *next-time*) that is later than T.

2.3. Finding shortest looping paths

Once we have found the set of paths corresponding to a path route, we continue to find the next path route that may contain loops. To find this path route, we use the algorithm of Yang and Chen [17] that includes a shortest path algorithm to find the shortest path P and a path deletion algorithm (called *Network Enlarge Algorithm*) to generate a new network N' given a network N. This path deletion algorithm, in fact, results in an enlarged network where all the paths but the deleted one (i.e., P) can be determined. The enlarged network is formed by adding new well-defined nodes and arcs. Take the network in Fig. 3(a) for example. After the shortest path $P = (0, 1, 2, 3, 6, t)$ is found, we can form the enlarged network N' as shown in Fig. 3(b). That is, given N_1 as the initial network, a sequence of networks $\{N_1, N_2, \ldots, N_i, \ldots, N_k\}$ will be generated where the i-th shortest path P_i will be determined from N_i. After enlarging the network, we need to update labels of new nodes and arcs.

2.4. The algorithm

Having described the algorithms that find a set of paths corresponding a path route and find looping paths for a time–window network, we present our framework as follows. The framework iteratively uses the algorithm by [7] to find a path route and its corresponding paths, and the algorithm by [17] to find the next path route that may contain loops.

Because the preceding framework may generate a great number of paths, we apply the heap structure of Fredman and Tarjan [24] to manage the paths. The advantage of using the heap structure is that the times to find and remove the minimum element or to insert a new element are all

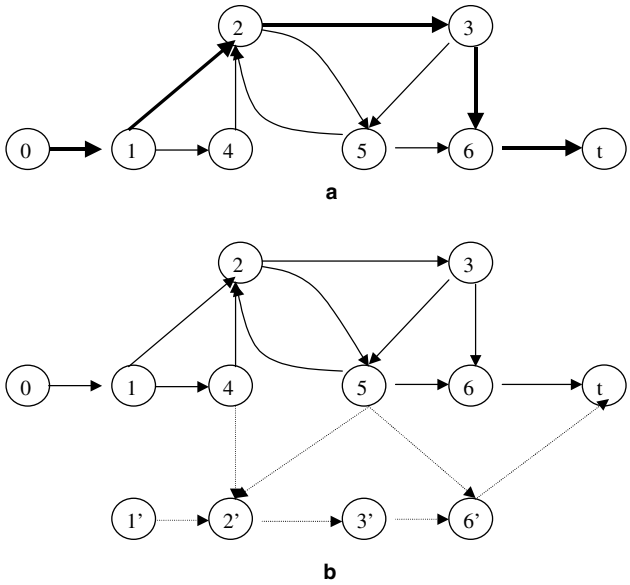

Fig. 3. (a) The network N and the shortest path P. (b) Delete P from N to generate N'.

$O(\log n)$ for a heap with n elements. Let an element in a heap, say Q, represent an enumerated path route and each element is associated with its total time. Recall that a path route is used to generate a set of paths and then to enlarge the network. Moreover, we need to store this path route with its next total time in case it remains to be the next shortest path. Therefore, each path route in Q can be classified into two types: (1) the path route that has been used but with smaller total time, and (2) the path route that has not been used yet. Because selecting the first type of path route will produce a network that has been enlarged, we need to mark those of the first type to prevent the duplication. Therefore, each path in Q is associated with the mark as well as the total time. In the algorithm below, we refer to *f*(*related-paths*) as a function to find the set of paths corresponding to the path route P, *f*(*next-time*) as a function to find the path route P's next total time that is later than T, and use *Network Enlarge Algorithm* to enlarge the network.

K Shortest Looping Algorithm

1. Find P by using the algorithm of Chen and Yang [23]. Store P to Q.
2. For $w = 1$ to K, execute the cycle from step 3 to step 9.
3. Select the shortest path P with total time T from Q, and remove P from Q.
4. Use *f*(*related-paths*) to output the paths as the w-th, …, $(w + x-1)$-th paths.
5. Set $w = w + x-1$. Set $T' = f(\textit{next-time})$ and $P' = P$.
6. Mark P' and store P' with T' into Q.
7. If P has been marked then go to the next cycle.
8. Use *Network Enlarge Algorithm*.
9. Update labels of added nodes and arcs.

To show the complexity of this *K Shortest Looping Algorithm*, we provide some lemmas that are used to show the complexities of intermediate steps. Also recall that r is the number of different windows of a node and $|V_1|$ is the number of nodes of network N_1.

Lemma 1 [23]. *The time complexity to find a path route P in the underlying network is* $O(r|V_1|^3)$.

Lemma 2. *The time complexity of the function f*(*related-paths*) *is* $O(r|V_1|^2)$.

Proof. By Lemma 1 of Chen and Yang [7], the time to construct a relate-graph is $O(r|V_1|^2)$. Having constructed the related-graph, we can execute the function *f*(*related-paths*) by a breadth-first search or a depth-first search. Because the time of a depth-first search or a breadth-first search is simply a linear function of the network size, the total time of the function *f*(*related-paths*) is $O(r|V_1|^2)$. □

Lemma 3 [17]. *The time complexity of executing Network Enlarge Algorithm for the K-th iteration is* $O(rK|V_1|^3)$.

Lemma 4 [17]. *The time complexity to update labels in the K-th iteration is* $O(\log rK|V_1|^3)$. *With preceding lemmas, we use the following lemma to show the complexity of K Shortest Looping Algorithm.*

Lemma 5. *The time complexity of the K Shortest Looping Algorithm is* $O(rK^2|V_1|^3)$.

Proof. Step 1 can be done in $O(r|V_1|^3)$ time by Lemma 1. To analyze a single cycle from step 3 to step 9, we decompose it into three parts: (1) step 3 to step 7, (2) step 8, and (3) step 9. For the first part, the most time-consuming step is step 4 that executes the function *f*(*related-paths*), which can be done in $O(r|V_1|^2)$ time by Lemma 2. The next part, step 8, can be done in $O(rK|V_1|^3)$ time by Lemma 3. Finally, step 9 can be done in $O(\log rK \mid V_1|^3)$ time by Lemma 4. Because $O(rK|V_1|^3)$ is the dominant term among these three parts, the time will be $O(rK|V_1|^3)$ for a single cycle from step 3 to step 9. As a result, we need $O(rK^2|V_1|^3)$ to execute the algorithm K iterations. □

An interesting question about our model is how sensitive are the model results to a given K such as 30. Referring to the problem definition in Section 2.1, we see that the sensitivity of the model is closely related to: (1) the starting time and duration of the window, and (2) the set of node-triplets. For brevity, we illustrate only the former. Reconsider the example in Fig. 1, where we will add some new windows to node A to observe the sensitivity. Given $P = (s, A, D, d)$ and $T = 10$, consider the case that a new window (16, 20) is added to node A. For simplicity, assume P remains to be the next shortest path route. In this case, the next total time T' of P is 24 and the set of paths is $\{(s_0, A_{16}, D_{22}, d_{24}),\ (s_0, A_{17}, D_{22}, d_{24}),\ (s_0, A_{18}, D_{22}, d_{24}),\ (s_0, A_{19}, D_{22}, d_{24})\}$. In comparison, if the new window added is (19, 20), the next total time is 24 and the set of paths is $\{(s_0, A_{19}, D_{22}, d_{24})\}$. In the former situation where the window (16, 20) is added, we may have satisfied the number K because four more paths are generated. In the latter situation, since only one path is generated, we may need to search for the next path route, which implies that the total time will be greater than 24.

3. Conclusions

The main contribution of this paper is that we develop a polynomial time algorithm for enumerating the first K shortest looping paths involving waiting time in a time–window network. Waiting time occurs in the presence of time–window constraints but is largely ignored in the literature. Finding the paths with waiting time may allow us to improve scheduling accuracy or enhance resource utilization. Compared with the previous results by [7,17], it is interesting to observe that the time to find looping paths appears to be the dominant part in the algorithm. This result arises from that finding looping paths relies on enlarging the network after a path is found.

Acknowledgement

The authors are grateful to the anonymous referees for their helpful comments. The first author was supported in part by National Science Foundation Grant No. 92-2416-H-167-002.

References

[1] M. Desrochers, J. Desrosiers, M.M. Solomon, A new optimization algorithm for the vehicle routing problem with time windows, Oper. Res. 40 (1992) 342–354.
[2] Y. Dumas, J. Desrosiers, E. Gelinas, M.M. Solomon, An optimal algorithm for the traveling salesman problem with time windows, Oper. Res. 43 (1995) 367–371.

[3] N. Kohl, O.B.G. Madsen, An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation, Oper. Res. 45 (1997) 395–406.
[4] J. Bramel, D. Simchilevi, Probabilistic analyses and practical algorithms for the vehicle routing problem with time windows, Oper. Res. 44 (1996) 501–509.
[5] R.A. Russell, Hybrid heuristics for the vehicle-routing problem with time windows, Transport Sci. 29 (1996) 156–166.
[6] N. Balakrishnan, Simple heuristics for the vehicle routing problem with soft time windows, J. Oper. Res. Soc. 44 (1993) 279–287.
[7] Y.-L. Chen, H.-H. Yang, Finding the first k shortest paths in a time–window network, Comput. Oper. Res. 31 (2004) 499–513.
[8] J.Y. Yen, Finding the k shortest loopless paths in a network, Manage. Sci. 17 (1971) 712–716.
[9] N. Katoh, T. Ibaraki, H. Mine, An efficient algorithm for k shortest simple paths, Networks 12 (1982) 411–427.
[10] E. Hadjiconstantinou, N. Christofides, An efficient implementation of an algorithm for finding k shortest simple paths, Networks 34 (1999) 88–101.
[11] Y.-L. Chen, D. Rinks, K. Tang, The first k minimum cost paths in a time–schedule network, J. Oper. Res. Soc. 52 (2001) 102–108.
[12] S. Dreyfus, An appraisal of some shortest path algorithms, Oper. Res. 17 (1969) 395–412.
[13] B.L. Fox, Data structures and computer science techniques in operations research, Oper. Res. 26 (1978) 686–717.
[14] E.Q.V. Martins, An algorithm for ranking paths that may contain cycles, Eur. J. Oper. Res. 18 (1984) 123–130.
[15] J.A. Azevedo, M.E.O. Santos Costa, J.J.E.R. Silvestre Madeira, E.Q.V. Martins, An algorithm for the ranking of shortest paths, Eur. J. Oper. Res. 69 (1993) 97–106.
[16] D. Villeneuve, G. Desaulniers, The shortest path problem with forbidden paths, Eur. J. Oper. Res. 165 (2004) 97–107.
[17] H.-H. Yang, Y.-L. Chen, Finding k shortest looping paths in a traffic–light network, Comput. Oper. Res. 32 (2005) 571–581.
[18] D. Eppstein, Finding the k shortest paths, SIAM J. Comput. 28 (1998) 652–673.
[19] V.M. Jiménez, A. Marzal, A lazy version of Eppstein's k shortest paths algorithm, Lect. Notes Comput. Sci. 2647 (2003) 179–191.
[20] H.-H. Yang, Y.-L. Chen, The first k shortest unique-arc walks in a traffic–light network, Math. Comput. Model. 40 (2004) 1453–1464.
[21] E. Ruppert, Finding the k shortest paths in parallel, Algorithmica 28 (2000) 242–254.
[22] N.J. van der Zijpp, S. Fiorenzo Catalano, Path enumeration by finding the constrained k-shortest paths, Transport Res. B 39 (2005) 545–563.
[23] Y.-L. Chen, H.-H. Yang, Shortest paths in traffic–light networks, Transport Res. B 34 (2000) 241–253.
[24] M.L. Fredman, R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM 34 (1987) 596–615.