K Constrained Shortest Path Problem

Ning Shi

***Abstract*—Motivated by a real project for a sophisticated automated storage and retrieval system (AS/RS), we study the problem of generating K shortest paths that are required to satisfy a set of constraints. We propose a structural branching procedure that decomposes the problem into at most $K|\mathcal{N}|$ subproblems, where $|\mathcal{N}|$ is the number of nodes in the network. By using a *Network Modification* procedure, each subproblem can be transformed into a constrained shortest path problem (CSP). When these constraints satisfy a so called *separable* property, the subproblem can be further simplified. Based on this branching procedure, we propose a specific algorithm for an application where resource and loopless constraints have to be respected. Numerical results show that our algorithm is very efficient and robust.**

***Note to Practitioners*—Motivated by a real project for an AS/RS, we study the K constrained shortest path problem. It is a fundamental network optimization problem of great practical interest in transportation planning, telecommunications network routing, logistics management, etc. We present an efficient algorithm for solving this problem. One novel feature of this algorithm is its "robustness." Comparing with previous works, its computational performance is much less sensitive to the network size and the value of K. The algorithm has been implemented as a core engine of a routing control system in the AS/RS. Meanwhile, it can serve as a basis for solving more complicated problems such as the multi-criterion constrained shortest path problem.**

***Index Terms*—Logistics, networks, operations research, routing.**

I. Introduction

MUCH has been said on the K shortest path problem. It is a classical network optimization problem of great practical interest, which has to be solved in a wide variety of real world applications. Typical examples are modeling of transportation, communication, and distribution networks, etc. In these applications, alternative paths have to be generated when some nodes are temporarily unreachable through the optimal route. The K shortest path problem is also used as a subproblem in more complicated problems, especially when using the column generation method.

In many applications, these paths are required to satisfy a set of practical constraints. For example, in the telecommunications industry, besides the speed, the service provider has to provide quality service by efficiently utilizing the network resources (e.g., bandwidth, administrative weight, etc.). Such requirements impose multiple quality of service (QoS) constraints. Therefore, a service provider may need to generate K constrained shortest paths [11]. Another example is our motivating application in the automated storage and retrieval system (AS/RS). To maintain a robust and stable routing, we need to generate a group of routes from the point of the origin to the targeted destination. These routes are required to satisfy constraints that are determined by the operator's years of experience. For example, there could be a constraint that the total number of steps of each path cannot exceed a maximum bound; or some specific type of nodes cannot be visited more than twice, etc. We call the problem of finding K constrained shortest paths *K Constrained Shortest Path Problem* (KCSP). Mathematically, consider a network $G = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of arcs. Each arc $(i, j) \in \mathcal{A}$ is associated with a non-negative cost c_{ij}. We consider the problem of finding K paths from the origin O to the destination D. Each path should satisfy a group of constraints. Note that here, the constraints can be of any form. The feasibility is defined by the feasible region P that is the set of all feasible paths. In this paper, the feasible region P is outlined in a very general sense; its concrete definition is application dependent. The objective is to find K distinct shortest paths from the set P, i.e., those paths in P that have the smallest, the second smallest, … and the Kth smallest cost.

The K unconstrained shortest path problem has been studied thoroughly. The algorithm with the best worst-case theoretical order is in [6], and efficient algorithms with computational advantages can also be found in [9], [10], [12], etc. When the paths are required to satisfy the "loopless" constraints, the problem is called *K shortest loopless path problem*. This problem can be solved by a polynomial algorithm in [15]. The idea of this algorithm dates back to [8]. Clarke *et al.* [3] offers a modified and extended version of the method in [8]. The idea of their algorithm is to find the kth shortest path from all paths that "branch" out from the first $k-1$ shortest paths. Yen [15] invented the first polynomial algorithm for the K-shortest loopless path problem. The algorithm has a computational complexity of $O(K|\mathcal{N}|^3)$ where $|\mathcal{N}|$ is the number of nodes in the network. This algorithm is very efficient when K is not large. Researchers have improved new implementations of Yen's algorithm [7], [12], [13]. Among them, the MPS algorithm [12] is one of the most efficient. The MPS algorithm belongs to the class of deviation algorithms, which is characterized by using a set X of candidates paths for the K shortest loopless path. Different from the normal deviation algorithms, the MPS algorithm updates the reduced cost in each arc, and only the arc with the lowest reduced cost is chosen when finding the shortest deviation. Details of the MPS algorithm can be referred to [12].

One common practice to solve the KCSP is to select K feasible paths from a set of candidate paths generated by a path

Manuscript received March 31, 2008; revised August 20, 2008. First published May 08, 2009; current version published January 08, 2010. This paper was recommended for publication by Associate Editor F. Chen and Editor Y. Narahari upon evaluation of the reviewers' comments. This work was supported in part by the China National Science Foundation under Grant 70802063.

The author is with the School of Business, Sun Yat-sen University, Guangzhou 510275, China (e-mail: shiningchina08@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2009.2012434

ranking algorithm [14]. Sometimes this method is referred to "strike out" approach. Researchers also propose approaches that intelligently narrow the searching space [16]. However, the fundamental ideas are the same: finding shortest paths first and then selecting feasible ones by a constraint filter. To obtain K feasible paths, one may need to generate an extremely large set of candidate paths. Therefore, researchers also turn to heuristic approaches [2]. Recently, an adaption of the MPS algorithm has been proposed in [4] to solve the constrained shortest path ranking problem. Instead of inserting all candidate paths into X, some paths can be avoided whenever they are infeasible and cannot originate any feasible solution. Though theoretically this MPS adaptation cannot promise that the number of candidate paths is bounded, numerical experiments show that it is very efficient both in terms of computational times and the number of candidate paths. Details of this adaptation can be referred to [4].

On the other hand, the *Constrained Shortest Path Problem* (CSP) has received attention from many researchers. The most important class of problems is called the *Resource-Constrained Shortest Path Problem*, where the arcs are associated with a given number of resources and a feasible path cannot use resources beyond a given bound. This type of problems can usually be solved very quickly. According to the experiments reported in [5], the algorithm used can search the shortest path in a network with more than 70 000 nodes in less than 10 s. The success of algorithms in solving CSP motivates us to deal with the KCSP by decomposing it into a number of CSP subproblems.

In this paper, we show that the "branching" strategy in [3] can be generalized to make it applicable to the KCSP. We assume that the length of any feasible path is bounded by $|\mathcal{N}|$. This assumption can be easily satisfied in many practical cases. For example, if the paths are required to be loopless, then each subproblem has a loopless solution and thus one can ensure that the length of any feasible path is bounded by $|\mathcal{N}|$. The advantage of this branching strategy is that to obtain K constrained shortest paths, we need to solve, at most, $K|\mathcal{N}|$ subproblems.

The remaining part of this paper is organized as follows. In Section II, we introduce the problem and the branching strategy. In Section III, we apply this procedure to a specific application, a real project for a sophisticated automated storage and retrieval system (AS/RS) where the *loopless* and the *resource constraints* have to be respected. We propose an exact algorithm for this specific application. Numerical experiments are conducted in Section IV to evaluate the efficiency of this algorithm. Section V concludes this paper.

II. Branching Strategy

Let p^j denote the jth shortest feasible path. To find p^k, we assume that all $p^j, j = 1, 2, \ldots, k-1$ have been already known. Let $P[k-1] = \bigcup_{i=1}^{k-1} p^i$ be the set of the first $k-1$ shortest feasible paths. Let

$T(k-1) =$ The network formed by the first $k-1$ shortest feasible paths.

$R(k-1) =$ The set of subpaths in $T(k-1) \backslash D$.

Here, we illustrate these notations by an example. Fig. 1 shows network $T(2)$ which contains the first and the second shortest feasible paths: $p^1 = O \rightarrow 1 \rightarrow 2 \rightarrow D$ and $p^2 = O \rightarrow 3 \rightarrow 2 \rightarrow D$. Then, the set $R(2) = \{(O \rightarrow 1), (O \rightarrow 3), (O \rightarrow 1 \rightarrow 2), (O \rightarrow 3 \rightarrow 2)\}$.

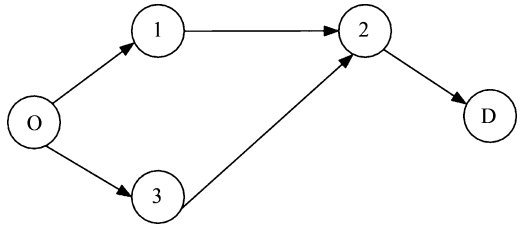

Fig. 1. Network formed by the best and second best paths.

Let $c(p)$ be the cost of path p. The kth shortest feasible path can be found by solving the following problem:

$$p^k = \arg \min_{p \in P \backslash P[k-1]} c(p). \tag{1}$$

The correctness of our algorithm is based on the following theorem.

Theorem 1: Given the first $k-1$ shortest feasible paths, the feasible region for the kth shortest feasible path can be partitioned into at most $(k-1)|\mathcal{N}|$ mutually independent subregions.

Proof: Let r be one path in $R(k-1)$. If any path in $P \setminus P[k-1]$, say p, has a subpath that is equal to r, we call this path one "son" of r. Denote the set of r's sons as $P(r)$. It is possible that there exists one path r' that has a subpath that is equal to r. In that case, $P(r) \supseteq P(r')$. Therefore, we further define a set $S(k, r)$ as the set of r's sons that solely belong to $P(r)$, that is, these sons do not belong to any other subpath in $R(k-1)$. Let $R(k-1, r)$ be the subset of paths of $R(k-1)$ that have a subpath equal to r. Mathematically, we can define $S(k, r)$ as

$$S(k, r) = P(r) \setminus \left(\bigcup_{r' \in R(k-1, r)} P(r') \right). \tag{2}$$

According to the definition, different sets of $S(k, r)$ are mutually excluded, i.e., for any two $S(k, r)$, $S(k, r')$, $r \neq r'$

$$S(k, r) \bigcap S(k, r') = \emptyset. \tag{3}$$

It is easy to verify that

$$\bigcup_{r \in R(k-1)} S(k, r) = P \setminus P[k-1]. \tag{4}$$

Therefore, the problem is equivalent to

$$p^k = \arg \min_{r \in R(k-1)} \left\{ \min_{p \in S(k,r)} c(p) \right\}. \tag{5}$$

Let us define the shortest path in subregion $S(k, r)$ as $p(k, r)$. Finding the shortest path in subregion $S(k, r)$ is defined as the subproblem $SP(k, r)$ to find

$$p(k, r) = \arg \min_{p \in S(k,r)} c(p). \tag{6}$$

Therefore, we can partition the solution region into at most $|R(k-1)|$ mutually excluded subregions defined by $S(k, r)$. The number $|R(k-1)|$ is bounded by $|(k-1)\mathcal{N}|$. Then, by

searching the best solution in each subregion, we can find the global optimal solution.

Observing that $R(k) \supseteq R(k-1)$, it is not necessary to solve all $|R(k-1)|$ subproblems to find the kth shortest feasible path. In fact, we can use a heap to store the solutions of the subproblems that have been solved in the process of finding the first $k-1$ shortest feasible paths. We only need to add in the solutions of the subproblems corresponding to subpaths of the $(k-1)$th shortest feasible path. This results in the following theorem.

Theorem 2: Suppose the subproblems for the paths in $R(k-2)$ have been solved. To find the kth shortest feasible path, we only need to solve the subproblems corresponding to subpaths of the $(k-1)$th shortest feasible path.

Proof: Let the set of the subpaths of the $(k-1)$th path be $\Delta R(k-1)$. By definition, $R(k-1) = \Delta R(k-1) \bigcup R(k-2)$. Let $\Delta R(k-1, r)$ be the subset of $\Delta R(k-1)$ that has a subpath equal to r.

Suppose $r \in R(k-2) \setminus \Delta R(k-1)$. In this case, the subregion corresponding to r, $S(k, r)$, is in fact equal to the set $S(k-1, r)$

$$\begin{aligned} S(k,r) &= P(r) \setminus \left(\bigcup_{r' \in R(k-1,r)} P(r') \right) \\ &= P(r) \setminus \left(\bigcup_{r' \in R(k-2,r)} P(r') \cup \bigcup_{r' \in \Delta R(k-1,r)} P(r') \right). \end{aligned}$$

Since $r \in R(k-1) \setminus \Delta R(k-1)$, $r \bigcap \Delta R(k-1) = \emptyset$. Then by definition, set $\Delta R(k-1, r)$ is null. Therefore,

$$\begin{aligned} S(k,r) &= P(r) \setminus \left(\bigcup_{r' \in R(k-2,r)} P(r') \right. \\ &= S(k-1, r). \end{aligned}$$

So we need not re-solve the subproblems corresponding to any subpath in $R(k-2) \setminus \Delta R(k-1)$. It follows that at most we need to solve $|\Delta R(k-1)|$ new subproblems, where $|\Delta R(k-1)|$ is bounded by $|\mathcal{N}|$.

Finally, we reach Theorem 3.

Theorem 3: The KCSP can be solved by solving at most $K|\mathcal{N}|$ subproblems defined by (6).

This branching strategy can be formally described below.

Step 1) *Step 1: Find the first shortest feasible path.*
Set $k = 0$, find the path p^k by solving a constrained shortest path algorithm. Put p^k in a heap H. The paths in the heap are sorted by their costs.

Step 2) *Step 2: Find the kth shortest feasible path.*
Generate $T(k)$ and $R(k)$. Let p_n^k be one *root path* from the origin to the nth node in p^k. According to the definition of $R(k)$, $p_n^k \in R(k)$. For each p_n^k, $n = 1, \ldots, |p^k| - 1$, solve the subproblem $SP(k, p_n^k)$, obtain the resulting candidate path and put it in heap H.

Step 3) *Step 3: Algorithm terminates.*
Let $k = k + 1$. Derive the path with minimal cost from the heap and let it be p^k. If $k \geq K$ or H is empty, stop. Otherwise return to Step 2.

This branching strategy is similar to that in [3]. However, our proof is based on the set theory and does not rely on the concrete forms of constraints, while [3] considers only the loopless constraint. Moreover, our proof of the correctness of this strategy is much simpler than the proof in [3].

Note that this branching strategy works only if the subproblems can be solved efficiently. In the following subsection, we discuss how to solve the subproblems.

A. Solving the Subproblems

One difficulty of solving subproblems of (6) is to narrow the feasible region into subregion $S(k, r)$. Note that the branching strategy does not require the paths r be loopless. However, if the paths r contain loops, narrowing the feasible region into the subregion $S(k, r)$ is quite application dependent. Accordingly, the corresponding subproblems $SP(k, r)$ need specific approaches to solve.

If the paths r are loopless, then we have a general approach to narrow the feasible region into subregion $S(k, r)$. Instead of working on the original network $G = (\mathcal{N}, \mathcal{A})$, we perform a *Network Modification* procedure and work on the modified network $G(r)$. The *Network Modification* procedure is defined as follows.

1) Network Modification: Let the end node of path r be $e(r)$. Let the jth node in r be r_j. For any $j = 1, \ldots, |r|-1$, in network G, we find the outlet arcs from node r_j. Denote this set as $A[r_j]$. Obviously, the arc (r_j, r_{j+1}) must be in that set. Then remove the arcs in $A[r_j]$ from network $G = (\mathcal{N}, \mathcal{A})$ except (r_j, r_{j+1}). In network $T(k-1)$ (Note: not the network G!), find the set of outlet arcs from node $e(r)$, denoted by $A[e(r)]$. Remove the arcs in $A[e(r)]$ from network $G = (\mathcal{N}, \mathcal{A})$. Denote the modified network as $G(k, r)$.

2) Explanation: By such a *Network Modification* procedure, we can guarantee that: 1) any path in $G(k, r)$ is a son of r; 2) any path r' in $T(k-1)$ "covering" path r is not a path in $G(k, r)$ because any outlet from $e(r)$ has been "cut." Let the feasible paths in $G(k, r)$ be $P(G(k, r))$. Thus, $P(G(k, r))$ satisfies the definition of $S(k, r)$, i.e.,

$$P(G(k, r)) = S(k, r).$$

We solve (6) in modified network $G(k, r)$. The problem of (6) now can be stated as

$$CSP : p(k, r) = \arg \min_{p \in P(G(k,r))} c(p) \tag{7}$$

i.e., the subproblem defined in (6) is equivalent to a constrained shortest path problem in a modified network $G(k, r)$.

Observe that any path p in $P(G(k, r))$ covers subpath r, i.e., $p = r + s$. We use s to denote a subpath from $e(r)$ to D, which is called *spur path* in the literature. In case $c(p) = c(r) + c(s)$ (once the cost is associated with the arc, this property will hold!), the problem CSP is equivalent to

$$CSP2 : s = \arg \min_{(r+s) \in P(G(k,r))} c(s). \tag{8}$$

In the following, we show that when the constraints satisfy a *separable property*, the problem CSP2 is equivalent to

$$CSSP: s = \arg\min_{s \in \bar{P}(r)} c(s) \tag{9}$$

where $\bar{P}(r)$ is the set of "feasible" *spur paths* corresponding to r. The feasibility evaluation is dependent on r.

From (9), instead of finding the complete shortest feasible path from the origin to the destination, we can find the shortest "feasible" *spur path*. In practice, this will significantly reduce the computational burden.

The feasibility of each path can be evaluated by an "evaluating" function E, defined as

$$E(p) = \begin{cases} 1, & \text{if } p \in P, \\ 0, & \text{otherwise.} \end{cases}$$

With this function, the feasible region can be characterized as

$$P = \{p : E(p) = 1\}.$$

Separable property Given the *root path* r, there exists an "evaluating function" M such that the feasibility of path p can be examined by examining the feasibility of path s based on information from r, i.e.,

$$E(p) = M(s, r).$$

In this property, function M itself is dependent on function E and *root path* r. We call any constraint that satisfies this property "separable constraint." Here, we show two examples. Let the ith node of path p be p_i.

3) Example 1: Resource Constraint: Suppose any arc $(p_i, j) \in A[p_i]$ is associated with a weight $w(p_i, p_{i+1})$ and the total weight of one path p cannot exceed the bound W. The "evaluating function" can be specified in the following

$$E(p) = \begin{cases} 1, & \text{if } W - \sum_{i=1}^{|p|-1} w(p_i, p_{i+1}) \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

Suppose that the total weight of *root path* r is $W(r)$

$$M(s, r) = \begin{cases} 1, & \text{if } W - W(r) - \sum_{i=1}^{|s|-1} w(s_i, s_{i+1}) \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

4) Example 2: Loopless Constraint: To examine whether path p is loopless or not, the "evaluating function" for p is

$$E(p) = \begin{cases} 1, & \text{if } p_i \neq p_j,\ \forall i, j = 1, \ldots, |p|,\ i \neq j \\ 0, & \text{otherwise.} \end{cases}$$

Given the *root path* r, we can examine whether r is loopless by evaluation function $E(r)$. Then, if the *surplus path* s is loopless and shares only one node with path r, i.e., the end node of $r(e(r))$, then path p is feasible. Therefore, the "evaluating function" for the *spur path* can be defined as

$$M(s, r) = \begin{cases} 1, & \text{if } E(r) = 1,\ E(s) = 1,\ s \bigcap r = \{e(r)\} \\ 0, & \text{otherwise.} \end{cases}$$

We can see that in this example, the evaluation function for the *spur path* is more complicated.

Theorem 4: With the *Network Modification* procedure and the *Separable Property* of the constraints, each subproblem defined in (6) can be transformed into a classical constrained shortest path problem defined in (9).

Proof: According to the definition, we know that

$$P = \{p : E(p) = 1\} = \{s + r : M(s, r) = 1\}.$$

Let

$$\bar{P}(r) = \{s : M(s, r) = 1\}.$$

We can see that the problem CSP2 is equivalent to the problem CSSP.

III. Application

In the motivating project, we design and implement a robust automated routing table generator for a sophisticated AS/RS system, which is one of the most important facilities in a major air cargo terminal in the world. The AS/RS system has more than 10 000 origin-destination pairs. The cargoes are shipped from an origin to a destination by various types of equipment such as stack crane, conveyers, etc. A route can be defined as a sequence of equipment from an origin to a destination. To prepare for equipment failure contingencies, for each origin-destination pair, a number of routes are required to be generated. Each piece of equipment is associated with a score and the routes with minimal total scores are desired. Meanwhile, these routes are required to satisfy a number of constraints prescribed by the operator on the basis of years of experience. For example, some types of equipment cannot be used more than twice. Therefore, this problem can be modeled as a KCSP. The majority of the constraints in our application can be categorized into two types: *Loopless Constraint* and *Resource Constraint*. As shown in the examples, both constraints are "separable."

The KCSP with *Loopless Constraint* has been well-studied in the literature. The fundamental idea is to use the branching strategy and then transfer any subproblem $CSP(r)$ to $CSSP(r)$ by removing all nodes in r, except $e(r)$, to avoid loops.

There are very few studies on the KCSP with *Resource Constraint*. One important work is in [4], which proposes an adaptation of MPS algorithm for ranking feasible loopless paths that have to satisfy some constraints of resource type.

In our project, each arc $(i, j) \in \mathcal{A}$ is associated with four weights and one cost. Each weight represents one type of resource consumed by covering this arc. The total resource consumed by any path must be no greater than a given value, say, W^U. Let $W(r)$ be the resource consumed by the *root path* r. To transform the problem $CSP(r)$ into the problem $CSSP(r)$, we need to limit the resource to $W^U - W(r)$ in s. The *Resource Constrained Shortest Path Problem* has been studied intensively in the literature. We implement a labeling setting algorithm (LSA) in [5] to solve the constrained shortest path problem. The LSA is similar to Dijkstra's algorithm [1] for the shortest path problem: it maintains a set of labels for

each node and each label on a node represents a different path from the origin to that node. Besides cost, each label also records the weights of the corresponding path. The algorithm uses a concept of "dominance" to eliminate the labels that are "dominated" by some other labels on the same node. Therefore, no labels having the same cost are stored, and for each label on a node, any other label with a lower cost must have at least one greater weight. The label elimination can be strengthened by using some bounds and feasibility checking procedures. Eventually, we can find the shortest feasible path by selecting the label with minimal cost on the end node. Interested readers may refer to [5] for more details. As a summary, implementation of the KCSP for this application is described below.

Implementation of the KCSP

Set $k = 1$; Get the first shortest feasible path, p^1, by LSA:
If (p^1 is not defined) **Then Stop**
/* There are no feasible loopless paths */
Else Insert p^1 into X
End If
While ($k \leq K$ and X is not empty) **Do**
  Retrieve the kth shortest feasible path from the heap X; Let p^k be the kth shortest feasible path;
  Set $n = 1$;
    While ($n \leq |p^k|$), **Do**
      Let p_n^k be the root path from the first node to the nth node of p^k;
      Perform a *Network Modification* step and get $G(k, p_n^k)$;
      Update the remained resources as $W^U - W(p_n^k)$;
      Given the remained resources, in network $G(k, p_n^k)$, obtain the best spur path s from the end node of p_n^k, $e(p_n^k)$, to the destination by LSA;
      If (s is not defined)**Then**
      Let $q = p_n^k + s$ and insert q into the heap X;
      End If
      Set $n = n + 1$;
    End While
  Set $k = k + 1$;
End While

Notice that in this implementation the *Network Modification* step need not be performed from scratch every time because we can obtain modified network $G(k, p_n^k)$ by updating network $G(k, p_{n-1}^k)$. Also, the preprocessing and scaling steps in [5] help speed up the LSA.

IV. Numerical Experiment

To evaluate the efficiency of the KCSP algorithm, we compare the KCSP algorithm with the algorithm in [4]. It is an adaptation of the MPS algorithm. The MPS algorithm belongs to a class of deviation algorithms, which is characterized by use of a set X of candidate paths for the K shortest loopless path. Let us call this algorithm KCSPMPS. To simplify the process of determining the candidate path, the KCSPMPS algorithm begins by computing a shortest path tree. Each arc's cost is replaced by the respective reduced cost, to allow one to choose only the "best" arc, which does not violate the loopless and the resource constraints with the root path, of the lowest reduced cost when finding the shortest deviation. However, in the KCSPMPS, the candidate paths are not required to be feasible. Therefore, each candidate path can be generated by concatenating a root path with a spur path, which can be created by standard shortest path algorithms. The efficiency can be further improved by generating the spur paths from a shortest path tree, which is computed in advance. However, as commented in [13], an upper bound of the number of candidate paths inserted into the set X cannot be established. Implementation of the KCSPMPS is described below.

Implementation of KCSPMPS

$\mathcal{T}(i) \leftarrow$ the tree of the shortest paths from node $i \in \mathcal{N}$ to the destination. Let node 0 be the origin;
$\mathcal{T} \leftarrow \bigcup_{i \in \mathcal{N}} \mathcal{T}(i)$; Generate $\mathcal{T}$ by standard shortest path algorithm;
If ($\mathcal{T}(0)$ is not defined) **Then Stop**;
/* There are no feasible loopless paths */
End If
Update the reduced costs for each arc in the network;
Let X be a heap and insert all paths in $\mathcal{T}(0)$ into X (sorted according to the reduced costs);
Let $k = 1$; *AllFound* $\leftarrow$ *False*;
While (($k \leq K$) and (not *AllFound*))
Feasible $\leftarrow$ *False*;
  While (($X \neq \emptyset$) and (not *Feasible*)), **Do**
  Retrieve the path of the least cost, i.e., p, from the heap X;
  Let $n = 1$. Let r be the root path from the origin to the nth node;
    While (($n < |p|$) and (r is loopless and satisfies resource constraints)) **Do**
    Let $A[e(r)]$ be the set of outlet arcs from the end node $e(r)$ of r;
    Let $\ell = 1$; Let (v_n, j) be the ℓth arc in $A[e(r)]$;
      While (((v_n, j) forms a cycle with r) or (violates the resource constraints))
      $\ell \leftarrow \ell + 1$
      End While
      If ($\ell \leq |A[e(r)]|$),**Then**
      $s \leftarrow (v_n, j) + \mathcal{T}(j)$ and $q = r + s$;
      Insert path q into the heap X;
      End if
    Let $n \leftarrow n + 1$; Let r be the root path from the origin to the nth node;

End While

If ((p is loopless) and (constraints are satisfied))

Then *Feasible* ← *True*, $p^k \leftarrow p$, $k = k + 1$;

End If

End While

If (*not* Feasible), **Then**

/* There are no feasible loopless paths */

AllFound ←*True*.

End If

End While.

Notice that in the implementation, retrieving a path of the minimal total reduced cost is very efficient as X is a heap. Moreover, arcs with the same tail node are ordered according to the reduced costs. This is known as the sorted forward star form (see [12]). Use of the sorted forward star form can decrease the number of performed operations. By checking the feasibility of the generated candidate paths, the nonfeasible paths are not considered and the generation of a new path is avoided whenever we can ensure that it is not feasible and it cannot originate any feasible solution. For more details of this algorithm, one can be referred to [4].

We conduct three sets of experiments. All experiments are conducted on a personal IBM PC with 2.0-GHZ CPU and 1-GB RAM. The first two sets of experiments use the randomly generated cases and the third set uses the real data from our client. From the randomly generated cases, we can make comparisons under various conditions. Parameters for the random problem generator include the number of nodes in the network, the arc density of the network, a parameter $\alpha \in (0, 1)$ that controls how arc resources are generated, and a parameter $\beta \in Z+$ that controls the resource limit. The cost of each arc is set to be a randomly generated integer between 1 and 100. We consider two situations. In the first situation, each arc is associated with one type of resource. In the second situation, each arc is associated with two types of resources. For resource type t, the amount of resource w_a^t associated with each arc a is generated according to the formula in [5]

$$w_a^t = 1 + \left\lfloor \frac{\alpha C}{c_a + 1} \right\rfloor + nrand\left(\left\lfloor \frac{C}{c_a + 1} \frac{1 - \alpha^2}{\alpha} \right\rfloor \right).$$

where $nrand(\ell)$ is the function which randomly generates an integer between 0 and ℓ (here we use Java's random number generator. For a different resource type t, its seed is different), c_a is the cost of the arc a and C is the maximum cost of an arc over all arcs in the network. For each type of resource t, its corresponding resource limit W^t is generated by taking the sum of all arc resources, dividing by β and rounding down. That is

$$W^t = \left\lfloor \frac{\sum_{a \in A} w_a^t}{\beta} \right\rfloor.$$

We found that the difficulty in the problems generated in this manner is quite sensitive to the choice of α and β. This coincides with the findings in [5]. Dumitrescu (2003) explains that the parameter α affects the degree of variability in the arc weights. In general, higher values of α yield shorter feasible paths, and decreasing α for the same value of β increases computational time. In [5], the authors set $\alpha = 0.163$ to make the problems relatively difficult. In our paper, we select α from the range of $[0.113, 0.473]$.

TABLE I
PROBLEM CHARACTERISTICS FOR EXPERIMENT 1

Problem	$\|N\|$	$\|A\|$	α	β	K	$\|T\|$
A1	100	489	0.113	80	50	1
A2	100	489	0.153	80	50	1
A3	100	489	0.193	80	50	1
A4	100	489	0.233	80	50	1
A5	100	489	0.273	80	50	1
A6	100	489	0.313	80	50	1
A7	100	489	0.353	80	50	1
A8	100	489	0.393	80	50	1
A9	100	489	0.433	80	50	1
A10	100	489	0.473	80	50	1
A11	100	489	0.113	80	100	1
A12	100	489	0.153	80	100	1
A13	100	489	0.193	80	100	1
A14	100	489	0.233	80	100	1
A15	100	489	0.273	80	100	1
A16	100	489	0.313	80	100	1
A17	100	489	0.353	80	100	1
A18	100	489	0.393	80	100	1
A19	100	489	0.433	80	100	1
A20	100	489	0.473	80	100	1
A21	100	489	0.113	80	150	1
A22	100	489	0.153	80	150	1
A23	100	489	0.193	80	150	1
A24	100	489	0.233	80	150	1
A25	100	489	0.273	80	150	1
A26	100	489	0.313	80	150	1
A27	100	489	0.353	80	150	1
A28	100	489	0.393	80	150	1
A29	100	489	0.433	80	150	1
A30	100	489	0.473	80	150	1
A31	100	489	0.113	80	200	1
A32	100	489	0.153	80	200	1
A33	100	489	0.193	80	200	1
A34	100	489	0.233	80	200	1
A35	100	489	0.273	80	200	1
A36	100	489	0.313	80	200	1
A37	100	489	0.353	80	200	1
A38	100	489	0.393	80	200	1
A39	100	489	0.433	80	200	1
A40	100	489	0.473	80	200	1

In the first set of experiments, we create a total of 40 cases. We set the number of nodes in the network $|\mathcal{N}| = 100$, the number of arcs in the network $|\mathcal{A}| = 489$, the value of $\beta = 80$, the number of types of resources, denoted as $|T|$, as 1. We change the value of K from 50 to 200 with increments of 50, and α from 0.113 to 0.473 with increments of 0.04. The characteristics of this set of test problems can be found in Table I.

The result of Experiment 1 can be found in Table II. For each case, we record the computational time measured by seconds and the number of candidate paths (# CP) generated in the whole process. We define a ratio γ as the computational time of KCSPMPS over the computational time of KCSP. It shows that the average value of γ is 22.4, implying that the speed of the KCSP algorithm is on average 22.4 times faster than the KCSPMPS algorithm. The reason is that the KCSPMPS algorithm needs to generate more candidate paths than the KCSP algorithm. It

TABLE II
EXPERIMENTAL RESULTS FOR EXPERIMENT 1

Case	KCSP		KCSPMPS		Speed Ratio	Ratio of # of CP
	Time (s)	# CP	Time (s)	# CP		
A1	3.90	176	1.31	15978	0.34	90.78
A2	2.43	170	1.18	15357	0.48	90.34
A3	2.79	171	1.32	15563	0.47	91.01
A4	2.45	171	1.78	21320	0.73	124.68
A5	2.20	161	6.39	47836	2.90	297.12
A6	2.34	166	3.34	31497	1.43	189.74
A7	2.84	169	3.36	32226	1.18	190.69
A8	2.27	165	4.51	39245	1.99	237.85
A9	2.19	156	3.40	33541	1.56	215.01
A10	2.08	153	2.71	27782	1.30	181.58
A11	5.87	338	7.70	53276	1.31	157.62
A12	5.06	336	5.86	46869	1.16	139.49
A13	5.66	345	5.82	46168	1.03	133.82
A14	5.27	337	8.76	57947	1.66	171.95
A15	4.66	333	68.20	129628	14.64	389.27
A16	4.94	329	19.84	85303	4.02	259.28
A17	5.39	332	16.73	78443	3.10	236.27
A18	4.92	327	50.00	113785	10.16	347.97
A19	4.51	321	64.79	119167	14.34	371.24
A20	4.40	309	28.04	92647	6.36	299.83
A21	8.79	513	27.03	92402	3.07	180.12
A22	7.82	513	31.26	92402	3.99	180.12
A23	8.54	516	23.95	84098	2.80	162.98
A24	7.82	497	37.26	100378	4.68	201.97
A25	6.78	480	468.40	245730	69.07	511.94
A26	7.42	485	234.31	184220	31.59	379.94
A27	8.46	497	112.31	140867	13.26	283.43
A28	6.98	470	361.54	220642	51.76	469.45
A29	6.60	477	380.37	227215	57.55	476.34
A30	6.79	457	251.07	190895	36.94	417.71
A31	11.01	674	94.59	134610	8.59	199.72
A32	10.73	677	94.89	135584	8.84	200.27
A33	11.57	681	76.71	126145	6.63	185.23
A34	10.98	667	154.56	160302	14.07	240.33
A35	8.92	636	1205.89	365823	135.16	575.19
A36	10.51	643	697.68	278558	66.35	433.22
A37	10.82	657	403.48	227215	37.26	345.84
A38	12.79	626	1178.23	357836	92.07	571.62
A39	10.39	637	1182.18	365666	113.77	574.04
A40	10.29	607	710.68	290806	69.02	479.09

TABLE III
TEST PROBLEMS CHARACTERISTICS FOR EXPERIMENT 2

Problem	$\|N\|$	$\|A\|$	β	K	$\|T\|$
B1	100	489	40	100	1
B2	100	489	40	200	1
B3	100	489	40	300	1
B4	100	489	40	400	1
B5	100	489	100	100	1
B6	100	489	100	200	1
B7	100	489	100	300	1
B8	100	489	100	400	1
B9	200	787	40	100	1
B10	200	787	40	200	1
B11	200	787	40	300	1
B12	200	787	40	400	1
B13	200	787	80	100	1
B14	200	787	80	200	1
B15	200	787	80	300	1
B16	200	787	80	400	1
B17	200	787	100	100	1
B18	200	787	100	200	1
B19	200	787	100	300	1
B20	200	787	100	400	1
B21	100	489	40	100	2
B22	100	489	40	200	2
B23	100	489	40	300	2
B24	100	489	40	400	2
B25	100	489	100	100	2
B26	100	489	100	200	2
B27	100	489	100	300	2
B28	100	489	100	400	2
B29	200	787	40	100	2
B30	200	787	40	200	2
B31	200	787	40	300	2
B32	200	787	40	400	2
B33	200	787	80	100	2
B34	200	787	80	200	2
B35	200	787	80	300	2
B36	200	787	80	400	2
B37	200	787	100	100	2
B38	200	787	100	200	2
B39	200	787	100	300	2
B40	200	787	100	400	2

shows that on average the KCSPMPS algorithm needs to generate 228.1 times more candidate paths than the KCSP algorithm. Though generating one candidate path by the KCSPMPS algorithm is much simpler than the KCSP algorithm, the KCSPMPS algorithm may not be as efficient as the KCSP algorithm when the number of candidate paths is too large. As Experiment 2 will show later, in some difficult cases the KCSPMPS algorithm cannot even solve the problem because the computer falls short of memory. Experiment 1 also confirms that higher values of α make the problems easier.

In the second set of experiments, we create in total 40 groups of cases. Each group contains ten cases which vary with the value of α, ranging from 0.113 to 0.473 with increments of 0.04. For each group, we record the average computational times and average number of candidate paths. In the first 20 groups, we consider only one type of resource. In the second 20 groups, we consider two types of resources. The problems vary with the network size β, K, and $|T|$. Detailed characteristics of those cases are shown in Table III.

Results of Experiment 2 are reported in Table IV. Note that the symbol "$*$" means that the algorithm fails due to shortage of computer memory. It shows that the values of β, $|T|$ can affect performances of the algorithms significantly. In particular, we have the following observations.

1) The performance of the KCSPMPS algorithm is more sensitive to the value of β. Taking the cases B29, B33, and B37, for instance (see Fig. 2, when β increases from 40 to 100, the KCSPMPS algorithm needs to generate 29 times more candidate paths, using 334 times more computational times, while the KCSP algorithm generates a lesser number of candidate paths with less computational times. This clearly demonstrates one outstanding characteristic of the KCSP algorithm: *Robust*.
2) The performance of the KCSPMPS algorithm is more sensitive to the values of K. Taking cases B33, B34, and B35, for instance (illustrated in Fig. 3). It demonstrates that when K increases from 100 to 300, the KCSPMPS algorithm needs to generate 5.43 times more candidate paths with 81.80 times more computational time; while the KCSP algorithm only needs to generate 2.99 times more candidate paths with 2.86 times more computational time. Note that when $K = 400$, the KCSPMPS fails in cases B8, B20, B24, B36, and B40, as the available computer memory could not store all generated candidate paths.

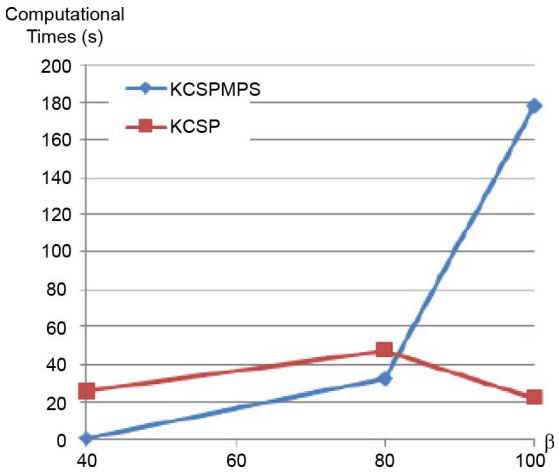

Fig. 2. Performance comparison under different values of β.

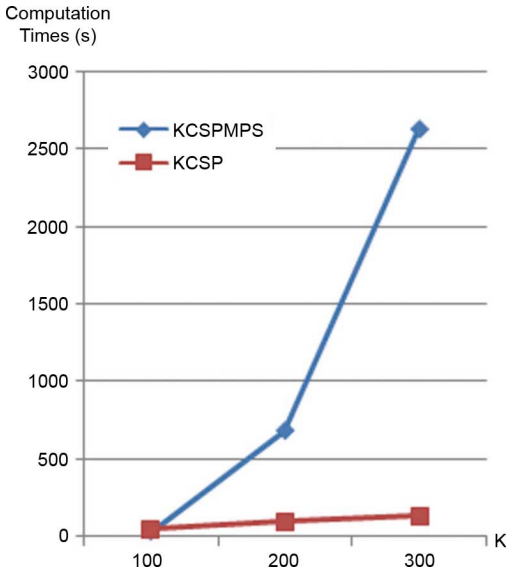

Fig. 3. Performance comparison under different values of K.

TABLE IV
RESULTS FOR EXPERIMENT 2

Case	KCSP		KCSPMPS		Speed Ratio	Ratio of # CP
	Time (s)	# CP	Time (s)	# CP		
B1	18.19	410	0.92	12359	0.05	30.14
B2	34.98	800	2.61	30304	0.07	37.88
B3	50.27	1191	5.27	46372	0.10	38.94
B4	64.45	1591	9.63	63475	0.15	39.90
B5	12.80	338	63.48	100659	4.95	297.81
B6	21.06	674	104.08	308140	49.41	457.18
B7	30.36	979	382.75	550365	126.07	562.17
B8	37.34	1304	*	*	*	*
B9	27.11	399	1.16	4784	0.04	11.98
B10	47.91	782	1.91	11857	0.04	15.16
B11	70.31	1171	3.80	19657	0.05	16.78
B12	88.52	1553	6.64	27874	0.08	17.94
B13	15.25	471	29.69	60971	1.94	129.45
B14	32.30	813	332.94	169679	10.31	208.71
B15	50.83	1230	1443.58	85303	28.40	261.34
B16	73.89	1658	3299.48	473683	44.65	285.69
B17	15.56	457	211.19	145900	13.62	319.26
B18	33.97	896	1456.19	329427	42.87	367.66
B19	56.00	1340	2818.72	473432	50.33	353.30
B20	82.91	1799	*	*	*	*
B21	14.99	299	5.56	26396	0.37	88.28
B22	27.89	594	23.38	64594	0.84	108.74
B23	40.30	875	69.34	106272	1.72	121.45
B24	51.67	1160	202.27	159521	3.91	137.52
B25	26.41	356	185.62	181668	7.03	510.30
B26	59.59	724	1104.80	365823	18.54	505.28
B27	87.41	1088	2789.92	550365	31.92	505.85
B28	103.48	1439	*	*	*	*
B29	25.58	448	1.14	5563	0.04	12.42
B30	54.61	883	2.09	14396	0.04	16.30
B31	77.17	1315	4.20	22945	0.05	17.45
B32	98.06	1782	8.22	34539	0.08	19.38
B33	46.97	418	32.19	91328	0.69	218.49
B34	98.30	827	683.80	270982	6.96	327.67
B35	134.61	1250	2633.11	495961	19.56	396.77
B36	166.09	1635	*	*	*	*
B37	21.88	401	177.94	162727	8.13	405.80
B38	43.58	788	1073.28	329421	24.63	418.05
B39	63.19	1163	3534.72	495961	55.93	426.45
B40	70.31	1545	*	*	*	*

From the above observations, we can see that the KCSP algorithm is more *robust* than the KCSPMPS algorithm, especially for difficult problems. The KCSPMPS algorithm is very efficient when the values of β and K are small. In particular, when $\beta = 40$, the average γ is 0.47. It means that the KCSPMPS algorithm is faster than the KCSP algorithm in those cases.

However, when the values of β and K are large, the KCSPMPS algorithm may not be as efficient as the KCSP algorithm. For example, when $\beta = 100$, the average γ is 37.02. It means that the KCSP algorithm is on average 37 times faster than the KCSPMPS algorithm in these cases. Besides, the KCSPMPS algorithm cannot achieve optimal solutions in some very difficult cases. It can be attributed to the difference between the KCSP algorithm and the KCSPMPS algorithm in terms of algorithmic design. The KCSP algorithm is a decomposition approach. As the number of subproblems is linearly increasing with the values of K, the bound of the total number of candidate paths is also linearly increasing with values of K. Also, the values of β have both positive and negative effect on the performance of the KCSP algorithm. On one hand, higher values of β make it more difficult to generate one single candidate path. On the other hand, higher values of β may narrow the searching space. The KCSMPS algorithm enjoys the efficiency of shortest path algorithms. However, there is no theoretical bound for the number of candidate paths, and higher values of β and K lead to exponentially increasing number of generated candidate paths. In some cases, the number of candidate paths can be too large to be handled, as Experiment 2 shows. As both algorithms have their advantages, the KCSP algorithm and the KCSPMPS algorithm are complementary in some sense.

In the third set of experiments, we use the real data from the client. We do not intend to describe this data set in detail but provide a sense of the problem size. In this data set, there are around 1500 nodes and around 2300 arcs, and there are four types of resources. We consider five different K values from 200 to 1000 with increments of 200. The result is shown in Table V.

The result shows that in the real problems C1–C5, the KCSP again outperforms the KCSPMPS algorithm when the value of K is large. This too coincides with the observations from Experiment 2.

TABLE V
RESULTS FOR EXPERIMENT 3

Problem	KCSP		KCSPMPS		Speed Ratio	Ratio of # CP
	Time (s)	# CP	Time (s)	# CP		
C1	11.52	483	3.45	6512	0.34	36.87
C2	24.83	811	38.24	26860	1.54	158.34
C3	48.72	1694	524.71	51300	10.77	299.75
C4	96.32	3888	2054.51	88065	21.33	514.98
C5	116.19	6001	7031.36	143463	52.34	493.32

V. CONCLUSION

The problem of finding the Kth constrained shortest path, irrespective of the exact forms of the constraints, can be solved by a branching strategy that partitions the whole feasible region into at most $K|\mathcal{N}|$ subregions. The Kth shortest feasible path can be found from the shortest feasible paths of all subregions. The problem of finding the shortest feasible path of one subregion is equivalent to a constrained shortest path problem in a modified network. Therefore, the KCSP can be solved by solving at most $K|\mathcal{N}|$ constrained shortest path problems. We compare the KCSP algorithm with an alternative method that is an adaptation of the MPS algorithm. The numerical experiments show that for problems with *loopless constraint* and *resource constraint*, the KCSP outperforms the alternative method, especially for difficult cases. The success of the KCSP algorithm relies on the efficiency of the constrained shortest path algorithms. Therefore, it is interesting to investigate whether this KCSP algorithm works well for K shortest path problems with more complex constraints. This is for future research.

REFERENCES

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

[2] D. L. Barra, B. Perez, and J. Anez, "Multidimensional path search and assignment," in *Proc. 21st PTRC Summer Annu. Meeting*, Manchester, U.K., 1993, pp. 307–309.

[3] S. Clarke, A. Krikorian, and J. Rausen, "Computing the N best loopless paths in a network," *SIAM J. Appl. Math.*, vol. 11, pp. 1096–1102, 1963.

[4] J. Climaco, J. Craveirinha, and M. Pascoal, "A bicriterion approach for routing problems in multimedia networks," *Networks*, vol. 11, pp. 399–404, 2003.

[5] I. Dumitrescu and N. Boland, "Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path algorithm," *Networks*, vol. 44, pp. 951–963, 2003.

[6] D. Eppstein, "Finding the k shortest paths," *SIAM J. Comput.*, vol. 28, pp. 652–673, 1998.

[7] E. Hadjiconstantinous and N. Christofides, "An efficient implementation of an algorithm for finding k shortest simple paths," *Networks*, vol. 34, pp. 88–101, 1999.

[8] W. Hoffman and R. Pavley, "A method for the solution of the Nth best path problem," *J. ACM*, vol. 6, pp. 506–514, 1959.

[9] F. Guerriero, R. Musmanno, V. Lacagnina, and A. Pecorella, "A class of label-correcting methods for the K shortest paths problem," *Operat. Res.*, vol. 49, pp. 423–429, 2001.

[10] V. Jimenez and A. Marzal, "Computing the K shortest paths: A new algorithm and an experimental comparison," *Lecture Notes in Comput. Sci.*, vol. 1668, pp. 15–29, 1999.

[11] G. Liu and K. G. Ramakrishnan, "A * Prune: An algorithm for finding K shortest paths subject to multiple constraints," in *Proc. IEEE INFOCOM*, 2001, pp. 743–749.

[12] E. Martins, M. Pascoal, and J. Santos, "Deviation algorithms for ranking shortest paths," *Int. J. Foundations Comput. Sci.*, vol. 10, pp. 247–263, 1999.

[13] E. Martins and M. Pascoal, "A new implementation of Yen's ranking loopless paths algorithm," *4OR*, vol. 1, pp. 121–133, 2004.

[14] H. H. Yang and Y. L. Chen, "Finding K shortest looping paths with waiting times," *Appl. Math. Model.*, vol. 30, pp. 458–465, 2006.

[15] J. Y. Yen, "Finding the K shortest loopless paths in a network," *Manag. Sci.*, vol. 17, pp. 712–716, 1971.

[16] N. J. Zijpp and S. F. Catalano, "Path enumeration by finding the constrained K-shortest paths," *Transport. Res. B*, vol. 39, pp. 545–563, 2005.

Ning Shi received the Ph.D. degree from the Hong Kong University of Science and Technology, Hong Kong, in 2007.

After working with IBM China Research Lab, he joined the School of Business, Sun Yat-sen University, Guangzhou, China. His research interests include operations research, supply chain management, and financial engineering.