Journal of Global Optimization **25:** 243–261, 2003.
© 2003 *Kluwer Academic Publishers. Printed in the Netherlands.*

Computation of the Reverse Shortest-Path Problem

JIANZHONG ZHANG[1] and YIXUN LIN[2]
[1]*Department of Mathematics, City University of Hong Kong, Hong Kong;* [2]*Department of Mathematics, Zhengzhou University, Zhengzhou, China*

Abstract. The shortest-path problem in a network is to find shortest paths between some specified sources and terminals when the lengths of edges are given. This paper studies a reverse problem: how to shorten the lengths of edges with as less cost as possible such that the distances between specified sources and terminals are reduced to the required bounds. This can be regarded as a routing speed-up model in transportation networks. In this paper, for the general problem, the NP-completeness is shown, and for the case of trees and the case of single source-terminal, polynomial-time algorithms are presented.

1. Introduction

The shortest-path problem plays an important role in combinatorial optimization because of its fundamental characteristic and wide range of applications. In fact, the model of seeking shortest paths has been proposed from various fields such as transportation, communication, computer networks and manufacturing systems. This problem has been extensively studied in both theoretic and algorithmic aspects (see [1, 8–13]).

Recently, several inverse versions of combinatorial optimization problems, including the inverse shortest path problem, have received much attention, due to practical motivations [2–7, 14–17].

There may be two types of inverse problems as follows:

1. Given a feasible solution x, adjust the parameters with as less cost as possible so that x becomes an optimal solution (with respect to the new parameters).
2. For a given min-problem and a given value v, adjust the parameters with as less cost as possible so that v becomes the optimal value, or an upper bound of the optimal value, for the problem.

The first one has been called “inverse” in the literature [3–6, 14, 15]. To make distinction from the first type, the authors of [16, 17] suggested the term “reverse” for the second type.

With regard to the shortest-path problem, the reverse version has a significant background as follows. In a transportation or communication network, the weight of a link may represent the transit time from one end to the other. So, the weight of a path is the traveling time from the source to the terminal. In order to improve the transmission efficiency of the network, one may want to speed up some routes, i.e., to reduce traveling times between some pairs of sources and terminals by decreasing weights of links. It is natural that reducing traveling time in a link (e.g., improving the conditions of the road to make vehicles run faster) needs some cost. We hope that the total cost is as less as possible to meet the requirement of speed-up. We call this a speed-up model. It is interesting that a slow-down model has been studied in [9] as follows: from an opponent's point of view, try to increase the lengths with a given budget so that the shortest path from a source to a sink is made as long as possible. However, the problem of speed-up is quite different from that of slow-down. Other relevant topics appeared in the literature are those about network improvement. For example, see [2] for the median location problem, see [16, 17] for the center location problem and [7] for the spanning tree problem.

Let us now state the problem formally. We are given a network $N = (V, E, l, \alpha, c)$ where (V, E) is a directed graph with vertex set $V = \{1, 2, \dots, n\}$ and edge set $E \subseteq V \times V$, $|E| = m$, and l, α, c are functions from E to $\mathbb{R}^+$ with the following definitions: for $e = (i, j) \in E$,

$$\begin{aligned} l_{ij} &= l(e) \text{ is the current length of } e,\\ \alpha_{ij} &= \alpha(e) \text{ is the least possible length of } e,\\ c_{ij} &= c(e) \text{ is the cost of reducing the length of } e \text{ by one unit.} \end{aligned}$$

Moreover, k pairs of vertices (s_i, t_i) and upper bounds d_i $(i = 1, 2, \cdots, k)$ are specified with the request that the length of the shortest path from s_i to t_i is not more than d_i. For each pair (s_i, t_i), s_i is called a source and t_i a terminal. We will denote by $d_l(s_i, t_i)$ the distance, i.e., the length of the shortest path, from s_i to t_i under the length function l.

The reverse shortest-path problem (RSP) is to find a new length function $x : E \to \mathbb{R}^+$ so as to

$$\text{minimize} \quad \sum_{(i,j)\in E} c_{ij}(l_{ij} - x_{ij}) \tag{1.1}$$

subject to

$$d_x(s_i, t_i) \leqslant d_i, \quad i = 1, 2, \cdots, k \tag{1.2}$$

$$\alpha_{ij} \leqslant x_{ij} \leqslant l_{ij}, \quad \forall (i, j) \in E. \tag{1.3}$$

In this model, we use a linear objective function (1.1) (i.e., a weighted l_1-norm) to represent the cost for meeting the requests (1.2) and (1.3). However, in some previous work on the inverse shortest-path problem [3–5], quadratic programming models were proposed by considering the l_2-norm of the error:

$$\text{minimize} \quad \sum_{(i,j)\in E} (l_{ij} - x_{ij})^2 \tag{1.4}$$

with a purpose of checking the priori expected lengths $\{l_{ij}\}$ and getting more accurate estimates of the lengths. With the same purpose one may use the l_1-norm to simplify the formulation (see [14, 15]):

$$\text{minimize} \quad \sum_{(i,j)\in E} |l_{ij} - x_{ij}|, \tag{1.5}$$

which is a special case of (1.1). As to the constraint (1.2), there are several cases to consider:
(a) multiple sources and multiple terminals: k pairs (s_i, t_i), $i = 1, \cdots, k$, are given and $k \geqslant 2$;
(b) single source: $s_i = s(i = 1, 2, \cdots, k)$;
(c) single terminal: $t_i = t(i = 1, 2, \cdots, k)$;
(d) single source and single terminal: $k = 1$.

We will show that cases (a)–(c) are strongly NP-hard no matter which objective function of (1.1), (1.4) or (1.5) is taken into account. On the other hand, a polynomial algorithm is proposed for case (d) under the weighted l_1-norm.

So, the main topic of the paper is the computational aspects of the RSP. The paper is organized as follows. In Section 2, we show the NP-completeness of the general model by transforming the three-dimensional matching problem into the model. Section 3 focuses on a dynamic programming algorithm for the case of single source and single terminal. A combinatorial algorithm is presented in Section 4 for the case of trees with a single source. Some concluding remarks are given in Section 5.

2. Results on Computational Complexity

Regarding the computational complexity of an optimization problem, we always discuss its decision version. The decision version of RSP is : for a given budget (threshold) b, is there a function $x : E \to \mathbb{R}^+$ satisfying (1.2), (1.3) and

$$c(x) = \sum_{(i,j)\in E} c_{ij}(l_{ij} - x_{ij}) \leqslant b? \tag{2.1}$$

As usual, to prove that a decision problem A is NP-complete, we need to show two things:

1. A is in the class NP;
2. An NP-complete problem B can be transformed to A in polynomial time.
   And A is said to be strongly NP-complete if we can further show that
3. A is still NP-complete even if the largest number appearing in the instances of A is bounded above by a polynomial (of the size of the instance).

We now proceed to study our RSP problem. To show its NP-completeness, we choose a well-known (strongly) NP-complete problem – three-dimensional matching (3DM), which is regarded as one of the six basic NP-complete problems in

[10]. That is, given three disjoint sets U, V, W with $|U| = |V| = |W| = K$ and a subset Q of $U \times V \times W$, is there a subset M of Q with $|M| = K$ such that whenever (u, v, w) and (u', v', w') are distinct triples in M, $u \neq u'$, $v \neq v'$, and $w \neq w'$? Here, such M is called a perfect matching (if it exists).

THEOREM 1. *The decision version of RSP is strongly NP-complete even if all parameters $l(e), \alpha(e), c(e)$ and d_i are 0 or 1.*

Proof. We will show the three things just mentioned. First, it is well-known that there are polynomial-time algorithms for the original shortest path problem (see, e.g., [11–13]). So, for a given length function x, checking condition (1.2), as well as (1.3) and (2.1), can be completed in a polynomial time. Therefore, the problem is in the class NP.

Second, given an instance of 3DM: $(U, V, W; Q)$, we may construct an instance of RSP, i.e., a network N, as follows:

(a) The vertex set of N is $U \cup V \cup W \cup Q \cup \{s\}$, where the new vertex s is a source and all vertices in U, V, W are terminals.
(b) For every triple $q = (u, v, w) \in Q$, we join edges $(q, u), (q, v), (q, w)$ where $u \in U$, $v \in V$, $w \in W$; also we join (s, q) for all $q \in Q$.
(c) For edges $e = (q, u), (q, v)$ or (q, w), let $l(e) = \alpha(e) = 0$, $c(e) = 1$; for edges $e = (s, q)$, let $l(e) = c(e) = 1, \alpha(e) = 0$. Moreover, for all pairs of sources and terminals $(s, u), (s, v)$ and (s, w), let the requested upper bound for these distances be $d_i = 0$ and let the threshold $b = K$.

An illustration of network N is shown in Figure 1. It is clear that the construction of this network N can be done in polynomial time. We only need to show the following **claim**: The instance of 3DM has a perfect matching M if and only if the instance of RSP (with respect to network N) has a feasible solution x such that $c(x) \leqslant K$.
In fact, if there is a perfect matching $M \subseteq Q$, we may let

$$x(e) = \begin{cases} 1, & \text{if } e = (s, q), q \notin M \\ 0, & \text{otherwise.} \end{cases}$$

Since K vertices of M are adjacent to all $3K$ vertices of $U \cup V \cup W$, it follows that the source s can reach all terminals by paths of length 0. Hence x is a feasible solution of network N with cost $c(x) = |M| = K$.

Conversely, let x be a feasible solution of N with $c(x) \leqslant K$. By condition (1.2) and the request that all $d_i = 0$, we can see that the source s and each terminal in $U \cup V \cup W$ have to be connected by a path of length 0. So, there must be some edges $e = (s, q)$ having $x(e) = 0$. Let $M = \{q \in Q | x(e) = 0 \text{ and } e = (s, q)\}$. Then the vertices of M can reach $3K$ vertices of $U \cup V \cup W$. Thus, $|M| \geqslant K$. Since $c(x) \leqslant K$, we have $|M| = K$ and hence M is a perfect matching.

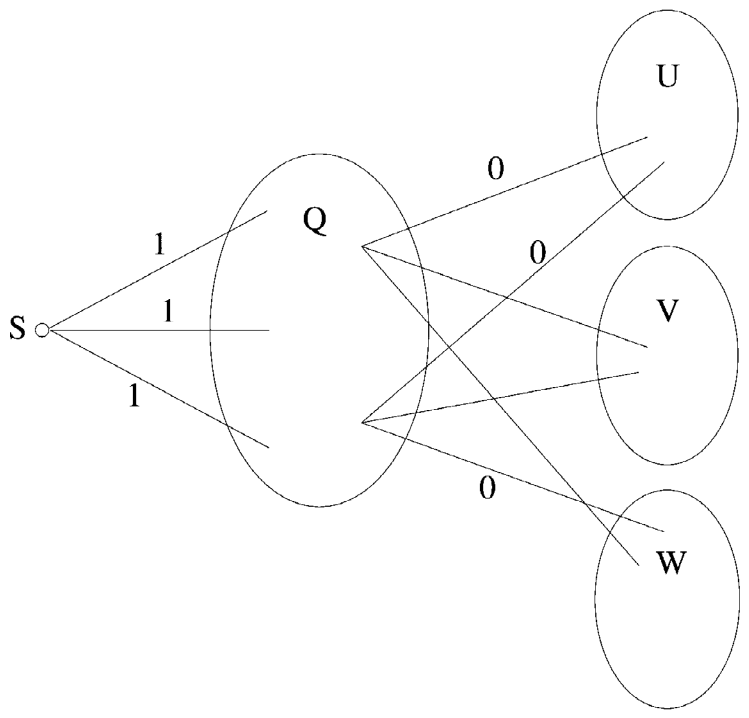

Figure 1.

To sum up, we have shown that the decision version of RSP is NP-complete. Furthermore, since 3DM is strongly NP-complete and in the construction of the instance of RSP all input data are 0 and 1, it follows that RSP is strongly NP-complete. This completes the proof. □

Note that the above proof is also valid for the objective function (1.4) studied in [3–5], as well as (1.5). Hence we obtain a further conclusion as follows.

COROLLARY 1. *The decision versions of RSP with l_2-norm, l_1-norm, and weighted l_1-norm are all strongly NP-complete even for the case of single source.*

For an NP-complete problem, it is unlikely that there would be a polynomial-time algorithm to solve the problem. The following two aspects are worthwhile to further study:
(1) practical (heuristic or approximation) algorithms;
(2) special cases which are polynomially solvable.
For example, we have a result concerning the second aspect.

THEOREM 2. *When the paths from s_i to t_i ($i = 1, \dots , k$) are specified, the RSP can be solved in polynomial time.*

Proof. Let the path from s_i to t_i be denoted by $P_i(i = 1, 2, \cdots, k)$, and let $E = \{e_1, e_2, \cdots, e_m\}$. We may define a $k \times m$ matrix $A = (a_{ij})$, as the path-edge incident matrix, by

$$a_{ij} = \begin{cases} 1, & \text{if edge } e_j \text{ is in path } P_i \\ 0, & \text{otherwise.} \end{cases}$$

Also, we represent the functions l, α, c, x by vectors $l = (l_1, l_2, \cdots, l_m)^T, \alpha = (\alpha_1, \alpha_2, \cdots, \alpha_m)^T, c = (c_1, c_2, \cdots, c_m)^T, x = (x_1, x_2, \cdots, x_m)^T$ where $l_j = l(e_j), \alpha_j = \alpha(e_j), c_j = c(e_j)$, and $x_j = x(e_j)$. Let $d = (d_1, d_2, \cdots, d_k)^T$. Then the RSP is a linear programming :

$$\begin{cases} \min \ c^T(l - x) \\ \text{s.t.} \ \ Ax \leqslant d, \\ \qquad \alpha \leqslant x \leqslant l. \end{cases} \tag{2.2}$$

It is well-known that there are polynomial-time algorithms for solving linear programming problems [12]. So, the theorem follows. □

This special case often appears in practical applications. In the routing speed-up model of transportation networks, the shortened routes are sometimes given in advance. Also, for some networks with special structures, the shortest paths P_i are uniquely determined, for instance, when N is a tree or a one-way network (i.e., there is at most one path between two vertices).

3. Polynomial Solvability for the Single Source and Single Terminal Case

In this section we investigate the RSP (1.1)–(1.3) with a single source and a single terminal, i.e., the case of $k = 1$. Our goal is to show that this case is polynomially solvable. For the vertex set $V = \{1, 2, \ldots, n\}$ assume that the source $s_1 = 1$, the terminal $t_1 = n$ and $d_1 = d$. For any non-negative scalar x, let

$$\begin{aligned} a_{ij}(x) &= \text{the cost for shortening edge } (i, j) \text{ from } l_{ij} \text{ down to } x \ (x \in R^1) \\ &= \begin{cases} 0, & \text{if } \ x \geqslant l_{ij}, \\ c_{ij}(l_{ij} - x), & \text{if } \ \alpha_{ij} \leqslant x \leqslant l_{ij}, \\ \infty, & \text{if } \ x < \alpha_{ij}. \end{cases} \end{aligned}$$

This is a piecewise linear and non-increasing function. We make a convention that $a_{ij}(x) = l_{ij} = \infty$ whenever $(i, j) \notin E$.

In order to establish a dynamic programming algorithm, we define the optimality function as $f_j(y) =$ the minimum shortening cost such that the distance from 1 to j is at most y. Then, $f_n(d)$ is the optimal value of our problem. By the Principle of Optimality, we obtain the DP recursive equation:

$$f_j(y) = \min_{k \neq j} \min_{0 \leqslant x \leqslant y} \{f_k(y - x) + a_{kj}(x)\} \tag{3.1}$$

with the initial condition

$$f_1(y) = 0 \quad \text{for all } y \geqslant 0. \tag{3.2}$$

Our task is to find the solution of this equation. To avoid complicated notations and to concentrate our attention on the structure of solutions, we start with the case that network N is acyclic. Recall that a directed graph is acyclic if and only if there exists a numbering of its vertices such that $(i, j) \in E \Rightarrow i < j$ [11, 12]. So, we may assume that the vertex set $V = \{1, 2, \ldots, n\}$ has been arranged according to this numbering. For a vertex j, $j \geqslant 2$, denote its precedence set by

$$N^-(j) = \{k \in V | (k, j) \in E\}.$$

Then, (3.1) can be written as

$$f_j(y) = \min_{k \in N^-(j)} \min_{0 \leqslant x \leqslant y} \{f_k(y - x) + a_{kj}(x)\}. \tag{3.1$'$}$$

We can calculate the functions $f_1(y), f_2(y), \ldots, f_n(y)$ one by one by this recursive equation. As we know, in the algorithms of the ordinary shortest path problem there is a label, say L_j, associated with each vertex j. And now, for the reverse problem we have to compute a function $f_j(y)$ for each vertex j, which makes the algorithm much more complicated. However, the following property can help us ease the difficulty.

LEMMA 1. *The functions $f_j(y)$, given by* (3.1)$'$, *are piecewise linear and non-increasing.*

Proof. We prove this lemma by induction on j. It is obvious for $j = 1$ by noting (3.2). Assume that the result holds for $1, 2, \ldots, j - 1$ and consider $f_j(y)$. For a given k, we observe the optimization problem

$$s_k(y) = \min_{0 \leqslant x \leqslant y} \{f_k(y - x) + a_{kj}(x)\}, \tag{3.3}$$

where, by the inductive hypothesis, $f_k(y - x)$ and $a_{kj}(x)$ $(k < j)$ are piecewise linear and non-increasing. Without loss of generality, suppose that they are (as shown in Figure 2):

$$a_{kj}(x) = \begin{cases} 0, & \text{if } \; x \geqslant l_0, \\ c_0(l_0 - x), & \text{if } \; \alpha_0 \leqslant x \leqslant l_0, \\ \infty, & \text{if } \; x < \alpha_0, \end{cases}$$

$$f_k(y) = \begin{cases} 0, & \text{if } \; y \geqslant l_1, \\ c_1(l_1 - y), & \text{if } \; l_2 \leqslant y \leqslant l_1, \\ c_1(l_1 - l_2) + c_2(l_2 - y), & \text{if } \; l_3 \leqslant y \leqslant l_2, \\ \cdots & \\ \sum_{i=1}^{p-1} c_i(l_i - l_{i+1}) + c_p(l_p - y), & \text{if } \; l_{p+1} \leqslant y \leqslant l_p, \\ \infty, & \text{if } \; y < l_{p+1}. \end{cases}$$

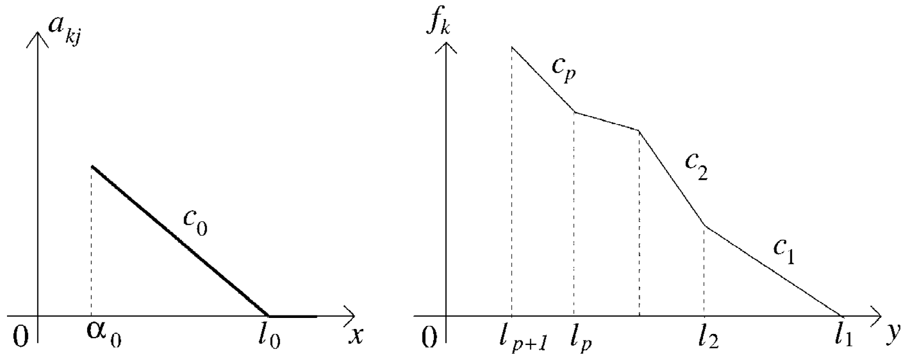

Figure 2.

A solution x of problem (3.3) is said to be *basic* if $x = l_0$ or α_0, or $y - x = l_i (1 \leqslant i \leqslant p+1)$. We have the following

Claim: For any given y, there is an optimal solution of (3.3) which is basic.

In fact, let x^* be an optimal solution of (3.3) satisfying

$$\alpha_0 < x^* < l_0, \; l_{i+1} < y - x^* < l_i.$$

Then

$$f_k(y - x^*) + a_{kj}(x^*) = \sum_{t=1}^{i-1} c_t(l_t - l_{t+1}) + c_i(l_i - y + x^*) + c_0(l_0 - x^*).$$

Assume that $c_0 \geqslant c_i$ (the case of $c_0 \leqslant c_i$ is symmetric). Let

$$\triangle x = \min\{l_0 - x^*, y - x^* - l_{i+1}\}.$$

Then

$$f_k(y - x^* - \triangle x) + a_{kj}(x^* + \triangle x) - f_k(y - x^*) - a_{kj}(x^*)$$
$$= c_i \triangle x - c_0 \triangle x \leqslant 0.$$

So, $x^* + \triangle x$ is also optimal and it is a basic solution because of the choice of $\triangle x$. Thus we confirmed the claim.

By virtue of the claim, we can deduce that the optimal value of (3.3) is

$$s_k(y) = \min\{f_k(y - \alpha_0) + a_{kj}(\alpha_0), f_k(y - l_0) + a_{kj}(l_0), f_k(l_1) + a_{kj}(y - l_1),$$
$$f_k(l_2) + a_{kj}(y - l_2), \ldots, f_k(l_{p+1}) + a_{kj}(y - l_{p+1})\}.$$

Here, all functions on the right-hand side are piecewise linear and non-increasing, and so is the minimum function $s_k(y)$. By the same reason, the minimum function

$$f_j(y) = \min_{k \in N^-(j)} s_k(y), \quad (j \geqslant 2)$$

is also piecewise linear and non-increasing. This completes the proof. □

Now that each function $f_j(y)$ is piecewise linear, and hence it can be represented by a finite sequence whose terms indicate the intervals and coefficients of the linear pieces of $f_j(y)$. For example, the function $f_k(y)$ in the proof of Lemma 1 can be represented by a sequence $(0, l_1; c_1, l_2; c_2, l_3; \cdots; c_p, l_{p+1})$. In the implementation of our algorithm, a compact data structure for the storage of these piecewise linear functions is needed. Otherwise the algorithm may not complete computation in polynomial time.

The main subroutine in our algorithm is to find the minimum function of several piecewise linear functions. This can be reduced to finding the minimum of several linear functions on each of the subintervals. For doing this, we present a naive method here. For instance, one wants to find

$$\varphi(y) = \min\{b_i - c_i y \mid i = 1, 2, \ldots, q\}, \quad (0 \leqslant y \leqslant r).$$

Assume that

$$b_1 \leqslant b_2 \leqslant \cdots \leqslant b_q \quad \text{and} \quad c_i > c_{i+1} \quad \text{if} \quad b_i = b_{i+1},$$

and $c_i \geqslant 0$.

Procedure ML:

Step 0 Let $i_1 = 1, t = 1$.

Step 1 Take

$$r_t = \min\left\{\frac{b_j - b_{i_t}}{c_j - c_{i_t}} \mid j > i_t, c_j > c_{i_t}\right\}$$

(note that $r_t = \infty$ if the set in the above expression is ∅). If $r_t \geqslant r$, go to Step 3; otherwise denote by i_{t+1} the index j that attains the minimum r_t (when there is a tie, take the largest j).

Step 2 Let $t := t + 1$, return to Step 1.

Step 3 Write down the solution

$$\varphi(y) = \begin{cases} b_{i_1} - c_{i_1} y, & \text{if} \quad 0 \leqslant y \leqslant r_1, \\ b_{i_2} - c_{i_2} y, & \text{if} \quad r_1 \leqslant y \leqslant r_2, \\ \cdots \\ b_{i_t} - c_{i_t} y, & \text{if} \quad r_{t-1} \leqslant y \leqslant r_t. \end{cases}$$

The complexity of this procedure is $O(q \log q)$. In fact, it runs at most $O(q)$ steps. However, the complexity for sorting all coefficients b_i is $O(q \log q)$.

The computation of $(3.1)'$ includes two procedures as follows.

Procedure I: Computation of $s_k(y)$ in (3.3).

Suppose that $f_k(y)$ and $a_{kj}(x)$ are the two functions given in the proof of Lemma 1. Due to the claim there, the optimal value $s_k(y)$ can be attained at some basic solutions x. When the optimal basic solution is

$$x = \begin{cases} l_0, & \text{if} \quad y \geqslant l_0 + l_i, \\ y - l_i, & \text{if} \quad \alpha_0 + l_i \leqslant y \leqslant l_0 + l_i, \\ \alpha_0, & \text{if} \quad \alpha_0 + l_{p+1} \leqslant y \leqslant \alpha_0 + l_i, \end{cases}$$

the optimal value of (3.3) will be

$$h_i(y) = \begin{cases} f_k(y - l_0), & \text{if} \quad y \geqslant l_0 + l_i, \\ f_k(l_i) + c_0(l_0 + l_i - y), & \text{if} \quad \alpha_0 + l_i \leqslant y \leqslant l_0 + l_i, \\ f_k(y - \alpha_0) + c_0(l_0 - \alpha_0), & \text{if} \quad \alpha_0 + l_{p+1} \leqslant y \leqslant \alpha_0 + l_i, \\ \infty, & \text{if} \quad y < \alpha_0 + l_{p+1}, \end{cases} \tag{3.4}$$

where $1 \leqslant i \leqslant p + 1$. Then

$$s_k(y) = \min_{1 \leqslant i \leqslant p+1} h_i(y). \tag{3.5}$$

This can be calculated by using Procedure ML.

Procedure II: Computation of $f_j(y) = \min_{k \in N^-(j)} s_k(y)$.

We have obtained several piecewise linear functions $s_k(y)$ by Procedure I (for a given vertex j). Now, we proceed to calculate the minimum of them by using Procedure ML.

In order to keep track of the shortest paths and the shortening scheme, we must store the optimal solution of (3.3) in Procedure I by

$x_{kj}(y) =$ the length of edge (k, j) if the distance from 1 to j is at most y,

and store the index k that attains the minimum of $s_k(y)$ in Procedure II by

$p_j(y) =$ the previous vertex of j in the shortest path from 1 to j if its length is at most y.

The algorithm can be summarized as follows.

DP Algorithm

Step 1 Let $f_1(y) = 0$ for all y.

Step 2 For $j = 2, 3, \ldots, n$ carry out the following computations.

(2a) For each $k \in N^-(j)$ compute

$$s_k(y) = \min_{0 \leqslant x \leqslant y} \{f_k(y - x) + a_{kj}(x)\}$$

by Procedure I and record the optimal solution x as $x_{kj}(y)$.

(2b) Compute

$$f_j(y) = \min_{k \in N^-(j)} s_k(y)$$

by Procedure II and save the optimal index k as $p_j(y)$.

Step 3 Search for the shortest path as follows. Suppose that $w = p_n(d)$. Then w is the previous vertex of the terminal n (in a shortest path with length d). And $x^*_{wn} = x_{wn}(d)$ is the required length of edge (w, n). Next, suppose that $v = p_w(d - x^*_{wn})$. Then v is the previous vertex of w with $x^*_{vw} = x_{vw}(d - x^*_{wn})$ as the adjusted length of edge (v, w). Continueing in this way, we will eventually get $1 = p_i(d - x^*_{wn} - \cdots - x^*_{ij})$ and $x^*_{1i} = x_{1i}(d - x^*_{wn} - \cdots - x^*_{ij})$. So, the shortest path from 1 to n is $(1, i, j, \ldots, v, w, n)$ with length $x^*_{1i} + x^*_{ij} + \cdots + x^*_{vw} + x^*_{wn} = d$ and the minimum cost $f_n(d)$.

LEMMA 2. *The above algorithm correctly solves the recursive equation* (3.1)′ *in polynomial time.*

Proof. By Lemma 1, the functions $f_j(y)$ are piecewise linear. Let us estimate the number of linear pieces (sections) of $f_j(y)$. There are two cases to consider:

Case 1 If the shortest path from 1 to j does not change when y varies, then the optimal shortening scheme can be obtained by shortening edges in the path one by one in a greedy way (the edge with the cheapest cost is shortened first). So, $f_j(y)$ is a convex piecewise linear function, and each piece corresponds to an edge of the path. Thus, the number of pieces of $f_j(y)$ is not more than n.

Case 2 When y decreases from $d_l(1, j)$ to the lower bound, the shortest paths from 1 to j are in turn $P_1, P_2, \ldots, P_s$. For each fixed path P_i, let $h^{(i)}(y)$ be the minimum shortening cost such that its length is at most y. By the argument of Case 1, $h^{(i)}(y)$ is convex and has at most n pieces. It is clear by definition that

$$f_j(y) = \min_{1 \leqslant i \leqslant s} h^{(i)}(y).$$

Also, $f_j(y)$ is composed of a part of $h^{(1)}(y)$, and then a part of $h^{(2)}(y)$, until a part of $h^{(s)}(y)$ (as shown in Figure 3). From this structure of $f_j(y)$, it follows that each path $P_i (i > 1)$ has at least a new edge which has fewer shortening cost and does not appear in the previous paths. Therefore $s \leqslant m$. Thus the number of pieces of $f_j(y)$ is not more than mn.

To sum up, $f_j(y)$ has at most mn pieces for all j.

In the computation of (3.1)′, let p be the number of pieces of $f_k(y)$, where $p \leqslant mn$. Then in Procedure I there are at most p basic solutions to be considered, and for each basic solution the number of pieces of function $h_i(y)$ in (3.4) is $O(p)$. So, in solving (3.5) there are $O(p^2)$ subintervals in which Procedure ML is carried out. As stated before, the complexity of Procedure ML is $O(p \log p)$. Hence the complexity of Procedure I is $O(p^3 \log p)$. As to Procedure II, there are at most n functions $s_k(y)$ each of which has $O(p^2)$ pieces. So, there are $O(np^2)$ subintervals for executing Procedure ML (finding the minimum of at most n linear functions). Thus the complexity of Procedure II is $O(n^2 p^2 \log n)$. In summary, the overall complexity of the algorithm is a polynomial of m and n. □

So far we considered the case that network N is acyclic and the recursive equation is (3.1)′. For a general directed network N we should solve equation (3.1)

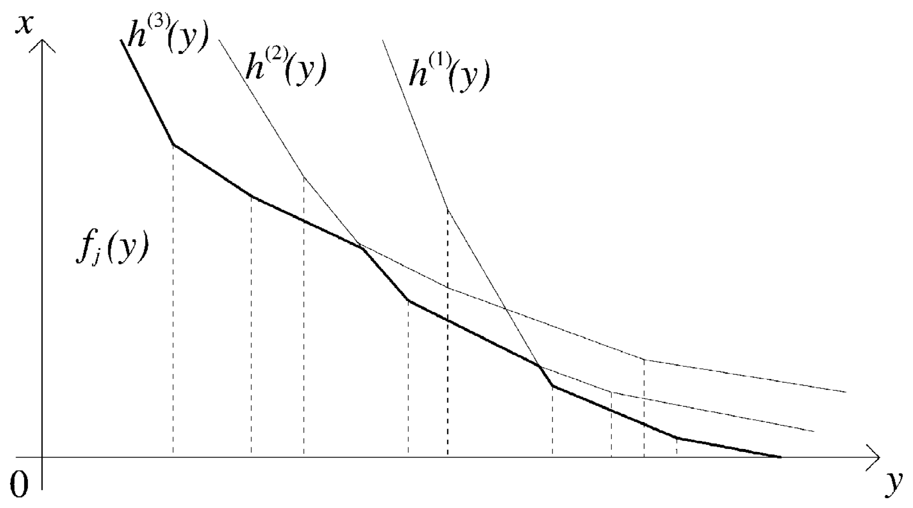

Figure 3.

with initial condition (3.2). Similar to the Bellman-Ford method for the shortest path problem [8, 11], we may apply a successive approximation in terms of the following functions: $f_j^{(r)}(y) =$ the minimum shortening cost such that the distance from 1 to j is at most y, subject to the condition that the shortest path from 1 to j contains no more than r edges.

Then

$$f_j^{(1)}(y) = \begin{cases} 0, & \text{if} \quad j = 1, \\ a_{1j}(y), & \text{if} \quad j \neq 1, \end{cases} \tag{3.6}$$

and

$$f_j^{(r+1)}(y) = \min\{f_j^{(r)}(y), \min_{k \neq j} \min_{0 \leqslant x \leqslant y} \{f_k^{(r)}(y - x) + a_{kj}(x)\}\}. \tag{3.7}$$

Since each of the shortest paths has no more than $n - 1$ edges, the successive process will terminate at the $(n - 1)$th approximation. Namely, $f_j^{(n-1)}(y)$ is the solution $f_j(y)$ of (3.1).

The approximation will start with the functions $f_j^{(1)}(y)$ given by (3.6). Suppose then that $f_j^{(r)}(y)$ have been obtained for $r \geqslant 1$. In order to compute $f_j^{(r+1)}(y)$ in (3.7), we have two procedures as before. In Procedure I, similar to (3.3), we may compute

$$s_k(y) = \min_{0 \leqslant x \leqslant y} \{f_k^{(r)}(y - x) + a_{kj}(x)\};$$

In Procedure II, we get

$$f_j^{(r+1)}(y) = \min\{f_j^{(r)}(y), \min_{k \neq j} s_k(y)\}.$$

These two procedures comprise the approximation of stage r. By comparing the corresponding recursive equations, it is not difficult to see that a stage of approximation is almost the same as the DP algorithm for acyclic networks. Therefore, the complexity of the approximation algorithm for solving (3.1) is n times of the complexity of the algorithm for solving $(3.1)'$. So, we have the following result.

THEOREM 3. *The RSP problem with a single source and a single terminal can be solved in polynomial time.*

4. A Combinatorial Algorithm for Tree Network with Single Source

We have formulated the RSP for tree networks as a linear program (2.2) in Section 2, which can be rewritten as

$$\begin{cases} \min & c^T y \\ \text{s.t.} & Ay \geqslant b, \\ & 0 \leqslant y \leqslant u, \end{cases} \tag{4.1}$$

where $y = l - x = (y_1, y_2, \dots, y_m)^T$ with y_j standing for the part of length l_j being shortened; $b = Al - d = (b_1, b_2, \dots, b_k)^T$ with $b_i = d_l(s_i, t_i) - d_i > 0$ representing the required reduction of the length of P_i; and $u = l - \alpha = (u_1, u_2, \dots, u_m)^T$ with $u_j \geqslant 0$ as the largest possible shortening of edge e_j. Recall that $A = (a_{ij})$ is the $k \times m$ path-edge incident matrix. We denote by a_i^T the i-th row of A. Note that the existing polynomial algorithms for general LP are not polynomial in the strong sense. In this section, based on the structure of tree networks, we present a strongly polynomial algorithm for the case of single source ($s = s_i$).

The following algorithm starts with an infeasible solution $y = 0$, which is nothing but a scheme with no edges being shortened. At stage r, let B^r denote the set of paths not reaching the required reduction. In order to further reduce these paths in B^r, we should choose a set of edges with the cheapest shortening cost to form an improving direction vector z. We then reduce the lengths of edges along this direction z until either the length of an edge reaches its lower bound or a path attains its required reduction. These two cases determine the step-length θ of the iteration. In this way we get a new solution $y + \theta z$. The iteration is carried on until a feasible solution y is found, which is also optimal. This algorithm is rather similar to the steepest descent method.

Combinatorial Algorithm

Step 0 Let $y^0 = 0$ and $r = 0$.

Step 1 If y^r is feasible for (4.1), stop (it is optimal); otherwise let

$$\begin{aligned} B^r &:= \{i \mid b_i - a_i^T y^r > 0\}, \\ c_j &:= \infty, \quad \text{if } y_j^r = u_j. \end{aligned}$$

Step 2 Find a moving direction $z \in R^m$ by solving the subproblem

$$\begin{cases} \min \ c^T z \\ s.t. \ \ a_i^T z \geqslant 1, \ \ \text{for } i \in B^r, \\ \qquad 0 \leqslant z_j \leqslant 1, \ \ \text{for } j = 1, 2, \ldots, m \end{cases} \tag{4.2}$$

(it will be shown that this z can be a $(0, 1)$-vector).

Step 3 Let $y^{r+1} = y^r + \theta z$ where $\theta = \min\{\theta_1, \theta_2\}$ and

$$\theta_1 = \min\{u_j - y_j^r \mid z_j = 1\} > 0,$$
$$\theta_2 = \min\{\frac{b_i - a_i^T y^r}{a_i^T z} \mid i \in B^r\} > 0.$$

Set $r := r + 1$ and return to Step 1.

The core of this iterative algorithm is subproblem (4.2), which has a precise interpretation as follows. Note first that the conditions $z_j \leqslant 1$ in (4.2) for all j are redundant since they must be satisfied by any optimal solution (if some $z_j > 1$, we would get a better solution by setting $z_j = 1$).

LEMMA 3. *Problem (4.2) has integer optimal solutions.*

Proof. It suffices to show that matrix A is totally unimodular, i.e., every subdeterminant of A is either 0 or ± 1 (see [11,12]). Let A' be an $r \times r$ submatrix of A with r rows corresponding to paths $P_{i_1}, P_{i_2}, \cdots, P_{i_r}$ and r columns corresponding to a set E' of edges. We are going to show that $det(A') = 0$ or ± 1 by induction on r. It is obvious for $r = 1$. Suppose it holds for $r < h$ and consider the case of $r = h$. Let e^* be an edge in E' which has the maximal distance from source s. If e^* does not belong to any of $P_{i_1}, P_{i_2}, \cdots, P_{i_r}$, the e^*-column of A' is a zero vector, and thus $det(A') = 0$. If e^* belongs to at least two of them, there will be two rows of A' being the same, thus $det(A') = 0$. So, we only need to consider the case that e^* belongs to a unique path, say P_{i_1} (there is only one 1 in the e^*-column). By using the inductive hypothesis to the matrix A'' obtained by deleting e^*-column and P_{i_1}-row from A', the conclusion follows. □

A subset $W \subseteq E$ is called a cut with respect to B^r if it meets all paths P_i for $i \in B^r$, i.e., $W \bigcap P_i \neq \emptyset$ $(\forall i \in B^r)$. The capacity of cut W is defined by $C(W) = \sum_{e_i \in W} c_i$, where the cost c_i for shortening per unit of length is regarded as the capacity of edge e_i. A minimum cut is a cut with the minimum capacity.

LEMMA 4. *The incident vector z of a minimum cut W is an optimal solution of subproblem (4.2).*

Proof. For a cut W, its incident vector z is defined by

$$z_j = \begin{cases} 1, & \text{if } e_j \in W, \\ 0, & \text{otherwise.} \end{cases}$$

$W \bigcap P_i \neq \emptyset$ $(\forall i \in B^r)$ means that $a_i^T z \geqslant 1$ for all $i \in B^r$. Then z is a feasible solution of (4.2) with the same cost as the capacity of W. On the other hand, for an integer optimal solution z^* of (4.2) (the existence has been claimed in Lemma 4.1), we have $z_j^* \in \{0, 1\}$ $(j = 1, 2, \cdots, m)$. Thus $W^* = \{e_i \in E \mid z_j^* = 1\}$ is a cut with respect to B^r. It is clear that the capacity of W^* is the same as the cost of z^*. So, W^* is a minimum cut. This completes the proof. □

From the above lemma we see that linear program (4.2) is essentially a pure combinatorial problem of finding minimum cuts, which can be solved by the max-flow algorithms. Since network N is a tree with single source s, the max-flow algorithm will be very simple. First, we may consider the network as a rooted tree T in which the root is source s and all leaves are terminals t_i $(i \in B^r)$. In fact, if there are some leaves other than t_i, then they can be deleted from T; if a terminal t_i is not a leaf, then all its descendants can be deleted from T. Doing these has no effect on the max-flow and min-cut. Then, to this rooted tree T with source s, sinks t_i $(i \in B^r)$ and edge capacity c, we produce a recursive procedure for finding max-flow as follows. Suppose that all edges leading from s are $e_1 = (s, v_1), \ldots, e_l = (s, v_l)$. We denote by $T(v_j)$ the subtree of T rooted at v_j (i.e. the subtree constituting the branch out of e_j). Let $v(T)$ represent the value of a maximum flow in T. Then it follows that

$$v(T) = \sum_{j=1}^{l} \min\{c(e_j), v(T(v_j))\}.$$

The procedure can be started at the sinks t_i with initial conditions $v(\{t_i\}) = \infty$, and executed recursively until the source is reached. After a maximum flow is found, a minimum cut can be obtained by taking the first saturated edge in each path P_i $(i \in B^r)$. It is clear that this procedure runs in $O(n)$ time. Therefore, the complexity of Step 2 of our algorithm is $O(n)$.

It remains to prove the correctness of the whole algorithm.

LEMMA 5. *There is an optimal solution y^* of problem (4.1) such that $y^* \geqslant y^1$, where y^1 is the first stage solution of the algorithm* $(r = 1)$.

Proof. Let $b^1 = Ay^1$. Then we can assert that y^1 is an optimal solution of the problem

$$\begin{cases} \min & c^T y \\ \text{s.t.} & Ay \geqslant b^1, \\ & 0 \leqslant y \leqslant u. \end{cases} \tag{4.3}$$

In fact, $y^1 = y^0 + \theta z = \theta z$ where z is an optimal solution of (4.2) for $r = 0$. For any feasible solution y of (4.3), since

$$A\left(\frac{1}{\theta} y\right) \geqslant \frac{1}{\theta} b^1 = \frac{1}{\theta} Ay^1 = Az \geqslant 1,$$

$\frac{1}{\theta}y$ is feasible for (4.2); thus $c^T(\frac{1}{\theta}y) \geqslant c^T z$ and $c^T y \geqslant \theta c^T z = c^T y^1$. Hence y^1 is optimal for (4.3).

Let y^* be an optimal solution of (4.1). If $y^1 \leqslant y^*$, we have done; otherwise we have

Claim: There is a vector $\hat{y} \in R^m$ such that $A\hat{y} = b^1$ and $0 \leqslant \hat{y} \leqslant y^*$.

In fact, this $\hat{y}$ can be constructed as follows. Assume that $y_1^1 > y_1^*$. As $b^1 \leqslant b$, each path P_i containing e_1 must contain another edge e_j with $y_j^1 < y_j^*$. So, we may choose E_1 as the set of edges e_j satisfying:

(a) e_j is contained in a path P_i containing e_1;

(b) $y_j^1 < y_j^*$;

(c) the distance from source s to e_j is minimal.

Then each path containing e_1 contains exactly one edge in E_1. So, we can construct a new vector $\tilde{y}$ by

$$\tilde{y}_j = \begin{cases} y_j^1 - \delta, & \text{if } j = 1, \\ y_j^1 + \delta, & \text{if } e_j \in E_1, \\ y_j^1, & \text{otherwise,} \end{cases}$$

where

$$\delta = \min\left\{ y_1^1 - y_1^*, \ \min_{e_j \in E_1} (y_j^* - y_j^1) \right\} > 0.$$

It is clear that $A\tilde{y} = Ay^1 = b^1$. If $\tilde{y} \leqslant y^*$, the procedure is completed; otherwise the same transformation is made for $\tilde{y}$ (instead of y^1). In this way we eventually get a vector $\hat{y}$ satisfying the claim.

By viewing that $\hat{y}$ is a feasible solution and y^1 is an optimal one for (4.3), we have $c^T\hat{y} \geqslant c^T y^1$. Let $y^{**} = y^* - \hat{y} + y^1$. Then

$$\begin{aligned} Ay^{**} &= Ay^* - A\hat{y} + Ay^1 \geqslant b - b^1 + b^1 = b, \\ c^T y^{**} &= c^T y^* - c^T \hat{y} + c^T y^1 \leqslant c^T y^*. \end{aligned}$$

Thus y^{**} is also an optimal solution of (4.1) and $y^1 \leqslant y^{**} \leqslant u$. This completes the proof. □

By this lemma, we can confine ourselves to finding the optimal solution y^* of (4.1) such that $y^1 \leqslant y^*$. Hence we may first shorten edges according to y^1. This is exactly the same as stage 1 of our algorithm. In stage r of the algorithm, we may update the vector b by $b - b^r$, where

$$b_i^r = \begin{cases} a_i^T y^r, & \text{if } i \in B^{r-1}, \\ b_i, & \text{otherwise,} \end{cases}$$

and update the index set B^r by deleting some i with $b_i = 0$. Then, the procedure of stage r is the same as that of stage 1. By using Lemma 5 again, it follows that for any r there is an optimal solution y^* of (4.1) such that $y^r \leqslant y^*$.

To summarize, we obtain the following conclusion.

THEOREM 4. *The above combinatorial algorithm correctly solves the RSP in a tree with single source in $O(n^2)$ time.*

Proof. At each stage of the algorithm, due to the choice of step length θ, either a variable y_j attains its upper bound u_j, or a path P_i reaches its required length. So, the algorithm terminates (with $b^r = b$) in at most $m + k$ stages. By Lemma 5, $y^r = y^*$ is an optimal solution of problem (4.1). For the time bound, we have seen that the number of stages is $O(n)$. And in each stage, finding a minimum cut for B^r can be carried out in $O(n)$ time, as mentioned before Lemma 5. Therefore, the overall complexity is $O(n^2)$. This completes the proof. □

5. Concluding Remarks

The shortest-path problem is an elementary and well-studied model in combinatorial optimization and dynamic programming. Note that almost all shortest-path algorithms so far are of dynamic programming type. In this paper we discuss a reverse problem: adjusting the lengths of edges optimally so as to reduce travelling times between some specified pairs of vertices to meet given upper bounds. This is in fact a reverse dynamic programming problem: adjusting the parameters in each stage with the least possible cost such that the efficiency of the multi-stage process can be raised to an expected level. The study of these reverse problems is supported by many practical applications.

We have shown that the general form of the problem is strongly NP-hard. So, it is unlikely to get a polynomial-time algorithm for the general case. As usual, our further studies are on polynomially solvable special cases, e.g., the case of trees and the case of single source and single terminal.

It is interesting to compare the following cases:

(i) one source and several terminals;
(ii) single source and single terminal.

In the original shortest-path problem the algorithms for (i) are as easy as those for (ii). As Lawler said in [11], “There seems to be no really good method for finding the length of a shortest path from a specified origin to a specified destination without, in effect, finding the lengths of shortest paths from the origin to all other nodes”. However, in the reverse problem the case (i) is NP-hard whereas the case (ii) is polynomially solvable. This shows that the reverse problem is quite different in nature from the original one.

In the previous section we showed an $O(n^2)$ algorithm for the trees with single-source and multiple-terminal. Symmetrically, the same conclusions holds for the trees with multiple-source and single-terminal. As to the multiple-source and multiple-terminal trees and one-way networks, Theorem 2.3 asserts their polynomial (maybe not strongly polynomial) solvability. In addition, for some special networks in which the shortened routes are internally disjoint, network N could be decomposed into several subnetworks each of which has a single source and a single terminal, and thus the related RSP problem has polynomial algorithms. For example, the multi-path and some series-parellel graphs possess this special structure.

By Theorem 2.3, a heuristic algorithm for general networks can be suggested as follows. When all paths P_i from s_i to t_i have been chosen and kept fixed, the optimal shortening scheme can be determined by solving the linear program (2.2). So, a path system $\{P_i\}$ can be regarded as the search direction of the linearization approximation, and the optimal solution of (2.2) is a locally optimal solution with respect to this direction. This procedure of searching locally optimal solutions can be carried out repeatedly by changing the path system $\{P_i\}$ each time. At last we can choose the best local solution.

Finally, the authors would like to thank the referees for their helpful comments and suggestions.

Acknowledgement

The authors are grateful to the support of Hong Kong University Grant Council under the grants CERG CityU 1081/99P and 1153/01P.

References

1. Bellman, R. (1958), On a routing problem, *Quar. Appl. Math.* 16, 87–90.
2. Berman, O., Ingco, D.I. and Odoni, A.R. (1992), Improving the location of minisum facilities through network modification, *Annals of Oper. Res.* 40, 1–16.
3. Burton, D. and Toint, Ph.L. (1992), On an instance of the inverse shortest paths problem, *Math. Program.* 53, 45–61.
4. Burton, D. and Toint, Ph.L. (1994), On the use of an inverse shortest paths algorithm for recovering linearly correlated costs, *Math. Program.* 63, 1–22.
5. Burton, D., Pulleyblank, W.R. and Toint, Ph.L. (1997), The inverse shortest paths problem with upper bounds on shortest paths costs, *Lecture Notes in Econom. and Math. System* 450, 156–171.
6. Cai, M., Yang, X. and Zhang, J. (1999), The complexity analysis of the inverse center location problem, *J. Global Optimization* 15, 213–218.
7. Drangmeister, K.U., Krumke, S.O., Marathe, M.V., Noltemeier, H. and Ravi, S.S. (1998), Modifying edges of a network to obtain short subgraphs, *Theoretical Computer Science* 203, 91–121.
8. Dreyfus, S.E. and Law, A.M. (1977), *The Art and Theory of Dynamic Programming*, Academic Press, New York.
9. Fulkerson, D.R. and Harding, G.C. (1977), Maximizing the minimum source-sink path subject to a budget constraint, *Math. Program.* 13, 116–118.
10. Garey, M.R. and Johnson, D.S. (1979), *Computers and Intractability: A Guide to the theory of NP-Completeness*, Freeman, San Francisco, CA.
11. Lawler, E.L. (1976), *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York.
12. Papadimitriou, C.H. and Steiglitz, K. (1982), *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.

13. Tarjan, R.E. (1985), Shortest path algorithms, *Graph Theory with Applications to Algorithms and Computer Science* (Kalamazoo, Mich., 1984), Wiley, New York, 753–759.
14. Zhang, J., Ma, Z. and Yang, C. (1995), A column generation method for inverse shortest path problem, *ZOR Math. Methods of Oper. Res.* 41, 347–358.
15. Zhang, J. and Ma, Z. (1996), A network flow method for solving some inverse combinatorial optimization problems, *Optimization* 37, 59–72.
16. Zhang, J., Liu, Z. and Ma, Z. (2000), Some reverse location problem, *European J. Oper. Res.* 124, 77–88.
17. Zhang, J., Yang, X. and Cai, M. (1999), *Reverse center location problem*, Lecture Notes in Computer Science 1741, 279–294.