AIMS Mathematics

http://www.aimspress.com/journal/Math

AIMS Mathematics, 6(4): 3996–4010.
DOI:10.3934/math.2021237
Received: 04 December 2020
Accepted: 24 January 2021
Published: 04 February 2021

Research article

Expanding maximum capacity path under weighted sum-type distances

Javad Tayyebi[1] **and Adrian Deaconu**[2,*]

[1] Department of Industrial Engineering, Birjand University of Technology, Birjand, Iran

[2] Department of Mathematics and Computer Science, Transilvania University, Brasov, 500091, Romania

* **Correspondence:** Email: a.deaconu@unitbv.ro; Tel: +40268413000.

Abstract: This paper investigates an inverse version of maximum capacity path problems. Its goal is how to increase arc capacities under a budget constraint so that the maximum capacity among all paths from an origin to a destination is improved as much as possible. Two distinct cases are considered: fixed costs and linear costs. In the former, an algorithm is designed to solve the problem in polynomial time. In the latter, a polynomial-time approach is developed to contain two phases. The first phase applies the first algorithm as a subroutine to find a small interval containing the optimal value. The second phase converts the reduced problem into a minimum ratio path problem. Then, a Secant-Newton hybrid algorithm is proposed to obtain the exact optimal solution. Some theoretical aspects as well as experimental computations are performed to guarantee the correctness and performance of our proposed algorithm.

Keywords: reverse optimization; maximum capacity path; Secant-Newton algorithm; minimum ratio path
Mathematics Subject Classification: 05C85, 68R10, 90C27

1. Introduction

Several variants of combinatorial optimization problems are defined on the path structure in graphs. One of the most important problems is the maximum capacity path problem. It is to find a path joining two prescribed nodes whose capacity, that is the minimum capacity of its arcs, is maximized. This problem is also known as the bottleneck path problem, the max-min path problem, and the widest path problem. Due to the fact that the problem can be seen as a network flow optimization problem, it admits a wide range of applications. For example, in telecommunication networks, the Multiprotocol Label Switching (MpLS) is a protocol to send a packet along a path whose minimum bandwidth is maximized [11]. As another application, single service units (e.g., fire fighter, police) estimate the

quality of a route by the maximum distance to an available service location [5].

As other instances of combinatorial optimization problems defined on path structure, shortest and longest path problems have completely different behaviours: the shortest path problem is polynomially solvable, whereas the longest path problem is NP-hard in general [3]. However, the maximum capacity path problem and its minimum version, namely the min-max path problem, have the same behaviour. Indeed, one can use any algorithm solving one to obtain an optimal path of the other. A recursive algorithm is the best-known algorithm that is capable of solving maximum capacity path problems in linear time [15].

This paper concentrates on an inverse version of maximum capacity path problems, called the maximum capacity path expansion problem. In general, three types of inverse problems are definable for any optimization problem:

- Given an initial feasible solution, the goal is to change some parameters of the problem as little as possible so that the solution becomes optimal.
- The problem is to change some parameters as little as possible such that the objective value of the optimization problem is bounded by a prescribed value.
- The goal is to modify some parameters under a budget constraint in a way that the optimal value of the optimization problem is improved as much as possible.

It is customary that the first category is called the inverse optimization [9, 10, 20]. To distinguish from the first, the problems belonging to the second category are referred to as the reverse optimization problems [18, 19], and optimization improvement problems [24, 25]. Third category has not any formal name in the literature. However, for the special case that the objective function is bottleneck-type (namely, min-max or max-min), the third category's problems are called the bottleneck capacity expansion problems (BCEPs) [13, 23, 26]. It is obvious that the problems of second and third categories have a close relationship together because if one interchanges the objective function and the budget constraint of a problem in the third category, then its corresponding problem in the second category is obtained.

1.1. Literature review on capacity expansion problems

Suppose that $E = \{e_1, e_2, \ldots, e_m\}$ is a ground set and $F \subseteq 2^E$. Let each $e_i \in E$ be associated to a capacity c_i. A (max-min) bottleneck-type combinatorial optimization problem is to find a $f \in F$ so that the minimum capacity of its elements is maximized, i.e.,

$$\max_{f \in F} \min_{e_i \in f} c_i. \tag{1.1}$$

This problem is an extension of several combinatorial optimization problems, such as maximum capacity path problem [14], max-min spanning tree problem [21].

For every type of problem (1.1), one can define a capacity expansion problem. It is to increase capacities under some budget and bound constraints so that the capacity of the optimal solution of problem (1.1) is increased as much as possible. This problem is the same BCEP which can be formulated as

$$\max_{x_i \in X} \max_{f \in F} \min_{e_i \in f} c_i + x_i, \tag{1.2}$$

in which X is a feasible region to restrict capacity increments.

Yang and Zhang [22] were the first in considering BCEPs defined on networks. They studied the problem of increasing arc capacities under a budget constraint so that the capacities of maximum capacity paths between every pair of nodes are maximized. They proposed a strongly polynomial-time algorithm which requires to solve a minimum spanning tree at every iteration. Subsequently, Chao and Jinglin [8] investigated the capacity expansion problem in spanning trees. They showed that the problem is reducible to a parametric spanning tree problem. Then, they presented a strongly polynomial-time algorithm to solve the problem. Li and Zhu [13] used the dynamic programming technique to present a pseudopolynomial-time algorithm for expanding the capacity of maximum capacity path from a source to a sink. Then, they designed polynomial-time algorithms for two special cases of the problem. Bukard et al. [6] studied the weight reduction problems with the min-max objective function, instead of max-min. They proved that the problem is NP-hard in general. Then, they reduced the problem to a parametric optimization problem which can be used for solving the problem in the special case of spanning trees. Chao and Jianzhong [7] considered three types of BCEPs (path, spanning-tree, and maximum flow) in two arc-expanding and node-expanding cases. They presented some results on their complexity.

BCEPs are also studied in the general case. A modified binary search algorithm [23] and a discrete-type Newton algorithm [26] are presented to solve BCEPs in the general case.

1.2. Our contribution

As a special case of BCEPs, this paper focuses on the maximum capacity path expansion problem (MCPEP) in two distinct cases: (I) fixed modification costs; (II) linear modification costs. To the best of our knowledge, the fixed-cost case is not investigated in the literature. In this case, a simple algorithm based on binary search is presented to solve the problem in strongly polynomial time. In the linear-cost case, a two-phase algorithm is developed to solve the problem in polynomial time. The first phase applies the same fixed-cost algorithm to find an interval containing the optimal value. The second phase converts the resulted problem into a maximum ratio path problem. Then, it uses a proposed hybrid Secant-Newton algorithm to solve exactly the problem. To validate our proposed algorithm, we theoretically and computationally compare it with the algorithms presented in [23] and [26].

Let us now state an application of MCPEP in transforation systems. Depending on the type of vehicle crossing a road, a capacity can be assigned to each road. For example, if only light vehicles are allowed to pass, the capacity can be equal to a small number, and if any type of light and heavy vehicles is allowed to pass, the capacity can be equal a large number. The capacity of a road depends on various factors such as road width, and road infrastructure. It is clear that the capacity of a route is equal to the minimum capacity of its roads. Now suppose we want to increase road capacities under a budget constraint so that heavy vehicles can pass between two specific points. This is a typical application of MCPEPs.

The rest of this paper is organized as follows. Section 2 states formally the problem and gives some initial results. Section 3 concentrates on the problem with fixed costs. Section 4 focuses on the problem with linear costs. It presents a strongly polynomial-time algorithm and compares it with the others. Section 5 presents our concluding remarks.

2. Problem statement

This section recalls the definition of maximum capacity path problems and introduces its expansion version. Then, it presents some initial results used throughout the paper.

Suppose that a connected and directed network $G(V, A, \mathbf{c})$ is given in which $V = \{1, 2, \ldots, n\}$ is the node set; A is the arc set containing m arcs. Each arc (i, j) is associated to a nonnegative capacity c_{ij}. The network contains two specific nodes s and t which are called the origin and the destination, respectively. A sequence $v_0 - v_1 - \ldots - v_k$ of nodes is said to be a walk if $(v_{i-1}, v_i) \in A$ for $i = 1, 2, \ldots, k$. A path is a walk without any repetition of nodes. A cycle is a path $v_0 - v_1 - \ldots - v_k$ together with the arc (v_k, v_0). It is obvious that one can simply obtain a path from a given walk P by removing its all cycles. This paper concentrates on paths from the origin s to the destination t, called st-paths. Any st-path P has a capacity c_P which is the minimum capacity of its arcs, i.e., $c_P = \min_{(i,j)\in P} c_{ij}$.

A classical combinatorial optimization problem, called the maximum capacity path problem (MCPP), is to find an st-path whose capacity is the maximum among all st-paths. Similar to the formulation of shortest path problems [3], one can simply formulate MCCP in the form of a zero-one linear programming problem as follows:

$$\max \quad z \tag{2.1a}$$

$$\text{s.t} \quad z \le c_{ij} + C(1 - x_{ij}) \tag{2.1b}$$

$$\sum_{(i,j)\in A_i^+} x_{ij} - \sum_{(j,i)\in A_i^-} x_{ji} = \begin{cases} 1 & i = s, \\ 0 & i \neq s, t, \\ -1 & i = t, \end{cases} \qquad \imath \in V, \tag{2.1c}$$

$$\sum_{(i,j)\in A_i^+} x_{ij} \le 1 \qquad \forall i \in V, \tag{2.1d}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A. \tag{2.1e}$$

in which $C = \max_{(i,j)\in A}\{c_{ij}\}$. Constraint (2.1b) guarantees that z equals the maximum capacity of the path P determined by constraints (2.1c). x_{ij} is a zero-one variable which takes one if and only if (i, j) belongs to the desired path P. The purpose of A_i^+ (A_i^-) is the set of arcs enumerating from (terminating at) i. Moreover, constraint (2.1d) is added to prevent node repetitions in the obtained path. On the other word, the solution is a path, and not only a walk. If the objective value of problem (2.1) is equal to z^*, then we simply say that the maximum capacity of $G(V, A, \mathbf{c})$ is z^*.

Now let us define the expansion version of problem (2.1). Suppose that we are capable of increasing arc capacities under some budget and bound constraints, and we would like improve the optimal value of problem (2.1) as much as possible. This problem can be formulated as follows:

$$\max \quad z \tag{2.2a}$$

$$\text{s.t} \quad z \le c_{ij} + y_{ij} + C^u(1 - x_{ij}) \tag{2.2b}$$

$$\sum_{(i,j)\in A_i^+} x_{ij} - \sum_{(j,i)\in A_i^-} x_{ji} = \begin{cases} 1 & i = s, \\ 0 & i \neq s, t, \\ -1 & i = t, \end{cases} \qquad \imath \in V, \tag{2.2c}$$

$$\sum_{(i,j)\in A_i^+} x_{ij} \le 1 \qquad \forall i \in V, \tag{2.2d}$$

$$\sum_{(i,j)\in A} b_{ij}(y_{ij}) \le B, \tag{2.2e}$$

$$0 \le y_{ij} \le c^u_{ij} - c_{ij} \qquad \forall (i,j) \in A, \tag{2.2f}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A, \tag{2.2g}$$

in which $C^u = \max_{(i,j)\in A}\{c^u_{ij}\}$, and y_{ij} is a continuous variable to determine the increment amount of capacity in (i, j); c^u_{ij} is an upper bound for increasing the capacity of (i, j); $b_{ij}(.)$ is a nonnegative, real-valued and nondecreasing function to measure the cost of increasing the capacity of (i, j). Moreover it satisfies the property that $b_{ij}(0) = 0$. The other notations are defined the same as problem (2.1).

Without the loss of generality, we may impose the following assumption to problem.

Assumption 1. *Each arc of the network belongs to at least an st-path.*

Assumption 1 is not restrictive because if not, one can perform two arc traversals, one time from s and another time from t in the opposite direction of arcs. The double-traversed arcs satisfy the assumption. So removing the other arcs satisfies Assumption 1. Obviously, this procedure can be performed in $O(m)$ time.

This paper focuses on problem (2.2) for two distinct cost functions:

Fixed costs: In this case, it is assumed that $b_{ij}(y_{ij})$ is equal to zero for $y_{ij} = 0$ and a positive fixed value $\bar{w}_{ij}$ for $y_{ij} > 0$. In the inverse optimization context [12, 17], $b_{ij}(y_{ij}) = \bar{w}_{ij}H(0, y_{ij})$, where $H(0, y_{ij})$ is the Hamming distance between 0 and y_{ij} defined as

$$H(0, y_{ij}) = \begin{cases} 0 & y_{ij} = 0, \\ 1 & y_{ij} \neq 0. \end{cases}$$

Linear costs: This case is to assume that $b_{ij}(y_{ij}) = w_{ij}y_{ij}$ in which $w_{ij} > 0$ is the cost of per unit increment of the capacity. So $b_{ij}(.)$ is a linear function with the slope w_{ij} and the y-intercept 0. In the literature, the sum of these linear cost functions are called the weighted Manhattan distance [4, 21].

A simple observation of problem (2.2) is that $y_{ij} = 0$, $(i, j) \in A$, is a feasible solution whose objective value z^{min} is equal to the maximum capacity of $G(V, A, \mathbf{c})$. So it provides a lower bound for the optimal value z^* of problem (2.2). On the other hand, $y_{ij} = c^u_{ij} - c_{ij}$, $(i, j) \in A$, tights bound constraints (2.2f), but it may not satisfy budget constraint (2.2e). So its objective value, denoted by z^{max}, provides an upper bound for z^*. Notice that z^{max} is obtained directly from bound constraints even without noting the budget constraint. In the case of linear costs, a new upper bound can be obtained as $z^{min} + \frac{B}{w_L}$ in which $w_L = \min_{(i,j)\in A} w_{ij}$ because for $(i, j) \in A$ with $x_{ij} = 1$, we have

$$\begin{aligned} w_{ij}z &\le w_{ij}c_{ij} + w_{ij}y_{ij} \\ &\le w_{ij}c_{ij} + B, \qquad \forall (i,j) \in A, \end{aligned}$$

which the last inequality hold due to the budget constraint. This inequality implies the upper bound $z^{min} + \frac{B}{w_L}$ if we assume that (i, j) is the arc which determines the initial maximum capacity of the network. In the case of fixed costs, a similar proof can be done to achieve this upper bound. Let us state formally these observations.

Lemma 2.1. *The optimal value of problem* (2.2) *belongs to*

$$[\bar{z}^{min}, \bar{z}^{max}] = [z^{min}, \min\{z^{max}, z^{min} + \frac{B}{w_L}\}]$$

where z^{min} and z^{max} are respectively the maximum capacities of $G(V, A, c_{ij})$ and $G(V, A, c^u_{ij})$, and

$$w_L = \begin{cases} \min_{(i,j)\in A} w_{ij} & \text{for linear costs,} \\ \min_{(i,j)\in A} \bar{w}_{ij} & \text{for fixed costs.} \end{cases}$$

Proof. The proof is straightforward based on the preceding paragraph. □

For an *st*-path P as well as a fixed value $z \in [\bar{z}^{min}, \bar{z}^{max}]$, consider a capacity vector $\mathbf{c}^{(z,P)}$ defined as

$$c^{(z,P)}_{ij} = \begin{cases} z & (i,j) \in P \wedge z \in (c_{ij}, c^u_{ij}], \\ c_{ij} & \text{otherwise,} \end{cases} \qquad \forall (i,j) \in A. \tag{2.3}$$

The following lemma shows that we can restrict ourselves to such capacity vectors to find an optimal solution of problem (2.2).

Lemma 2.2. *If $\bar{\mathbf{c}}$ is a feasible capacity vector to problem* (2.2)*, then $\mathbf{c}^{(\bar{z},\bar{P})}$ is also feasible, where $\bar{P}$ is a maximum capacity path with respect to $\bar{\mathbf{c}}$, and $\bar{z}$ is its capacity. Moreover, the objective values of both the vectors are the same.*

Proof. It is easy to see that $c^{(\bar{z},\bar{P})}_{ij} \le \bar{c}_{ij}$ for every $(i,j) \in A$. This simple observation together with the feasibility of $\bar{\mathbf{c}}$ imply that $\mathbf{c}^{(\bar{z},\bar{P})}$ satisfies the bound and budget constraints. On the other hand, by definition, the capacity of $\bar{P}$ with respect to $\mathbf{c}^{(\bar{z},\bar{P})}$ is exactly equal to $\bar{z}$. So the objective value of $\mathbf{c}^{(\bar{z},\bar{P})}$ is the same as that of $\bar{\mathbf{c}}$. □

Notice that the capacity vector $\mathbf{c}^{(z,P)}$ is defined with respect to an *st*-path P as well as a value $z \in [\bar{z}^{min}, \bar{z}^{max}]$. As seen in Lemma 2.2, z depends on P. On the other word, if P is determined, then we can simply compute z because it is the minimum capacity of arcs belonging to P. Let us now ask this question: *Is there a convenient st-path P for a given value z?* To reply this question, consider a cost vector $\mathbf{w}^{(z)}$ defined as follows:

$$w^{(z)}_{ij} = \begin{cases} +\infty & z > c^u_{ij}, \\ b_{ij}(z - c_{ij}) & c_{ij} < z \le c^u_{ij}, \\ 0 & z \le c_{ij}, \end{cases} \qquad \forall (i,j) \in A. \tag{2.4}$$

The following lemma clarifies the relationship between $\mathbf{w}^{(z)}$ and our question.

Lemma 2.3. *For a fixed value z, let $\bar{P}$ be a shortest st-path with respect to $\mathbf{w}^{(z)}$. Then,*

1). *$\mathbf{c}^{(z,\bar{P})}$ is a feasible solution of problem* (2.2) *if the length of $\bar{P}$ is less than or equal to B.*
2). *There is not any st-path P so that $\mathbf{c}^{(z,P)}$ is a feasible solution of problem* (2.2) *if the length of $\bar{P}$ is greater than B.*

Proof. The proof of the first part is trivial. To prove the second part, by contradiction, suppose that $\mathbf{c}^{(z,\tilde{P})}$ is feasible for some *st*-path $\tilde{P}$. So $\mathbf{c}^{(z,\tilde{P})}$ satisfies the budget constraint. This together with the assumption that the length of $\bar{P}$ is greater than B imply that the length of $\tilde{P}$ is less than that of $\bar{P}$ with respect to $\mathbf{w}^{(z)}$. This is a contradiction because $\bar{P}$ is a shortest *st*-path. □

3. Fixed costs

This section studies problem (2.2) with fixed costs. It presents an algorithm to solve the problem in strongly polynomial time.

According to Lemma 2.3, problem (2.2) reduces to finding the greatest value $z \in [\bar{z}^{min}, \bar{z}^{max}]$ so that the length of the shortest *st*-path with respect to $\mathbf{w}^{(z)}$ does not exceed B. This result motivates us to apply a binary search procedure for finding such value z. The following lemma aids us to limit the search interval only to a finite set.

Lemma 3.1. *The optimal value of problem* (2.2) *with fixed costs belongs to the finite set* $S = \bigcup_{(i,j)\in A}\{c_{ij}, c^u_{ij}\} \cap [\bar{z}^{min}, \bar{z}^{max}]$.

Proof. By contradiction, assume that z^* is the optimal value not belonging to S. So there is an *st*-path P^* so that $\mathbf{c}^{(z^*,P^*)}$ is the optimal capacity vector. Sort the distinct elements of S in a nondecreasing order and let $z_1 < z_2 < \ldots < z_k$ be the sorted list. So there is an index $l \in \{2, \ldots, k\}$ so that $z_{l-1} < z^* < z_l$. Now, consider the capacity vector $\mathbf{c}^{(z_l,P^*)}$. It is easy to prove that this vector satisfies the bound and budget constraints, and moreover, its optimal value is equal to z_l. This contradicts the fact that z^* is the optimal value. □

Algorithm 1 presents a binary search in complete details for solving problem (2.2).

Algorithm 1 Solving problem (2.2) with fixed costs.

Input: An instance of problem (2.2) with fixed costs defined on a network $G(V, A, \mathbf{c}, \mathbf{w})$ with two specified nodes s and t in which $\mathbf{c}$ is initial capacity vector and $\mathbf{w}$ is fixed cost vector.
Output: An optimal *st*-path P^*, and the optimal value z^*.
Set z^{max} to the length of the maximum capacity path with respect to capacities c^u_{ij}.
Find a shortest path P with respect to $\mathbf{w}^{(z^{max})}$.
if The length of P less than or equal to B **then**
    Stop because the problem has the optimal path $P^* = P$, and the maximum capacity $z^* = z^{max}$.
end if
Set $\bar{z}^{min}$ to the length of the maximum capacity path with respect to capacities c_{ij}.
Set $\bar{z}^{max} = \min\{z^{max}, \bar{z}^{min} + \frac{B}{w_L}\}$ where $w_L = \min_{(i,j)\in A} w_{ij}$.
Sort the distinct elements of $S = \bigcup_{(i,j)\in A}\{c_{ij}, c^u_{ij}\} \cap [\bar{z}^{min}, \bar{z}^{max}]$. Let $z_1 < z_2 < \ldots < z_k$ be the sorted list.
Set l=1, u=k.
while $l < u$ **do**
    Set $mid = [\frac{l+u}{2}]$.
    Find a shortest path P with respect to $\mathbf{w}^{z_{mid}}$.
    if The length of P less than or equal to B **then**
        Set $l = mid$, $z^* = z_{mid}$, and $P^* = P$.
    else
        Set $u = mid$.
    end if
end while

Theorem 3.2. *Algorithm 1 solves problem* (2.2) *in strongly polynomial time.*

Proof. The correctness of Algorithm 1 follows from the results presented in Lemma 3.1 and the preceding section. So let us argue only about its complexity. The bottleneck operation of Algorithm 1 is to solve a shortest path problem in the While loop. Since the While loop runs $O(\log(2m)) = O(\log(n))$ times, and the Fibonacci heap implementation of Dijkstra's algorithm requires $O(m + n\log(n))$ computations, it follows that Algorithm 1 solves problem (2.2) in $O(m\log(n) + n\log^2(n))$ time. □

4. Linear costs

This section concentrates on problem (2.2) whenever the cost of (i, j) is a linear function as $b_{ij}(y_{ij}) = w_{ij}y_{ij}$. Although Lemma 2.3 remains valid in this case, the issue is not as simple as the fixed-cost case because the search space is a continuous interval, instead of a finite set.

To develop an algorithm for solving problem (2.2) with linear costs, consider the function $b(z)$ which is the length of the shortest st-path with respect to the weight vector $\mathbf{w}^{(z)}$, i.e.,

$$b(z) = \min_{P\in\mathcal{P}^{(z)}} \sum_{(i,j)\in P} \max\{0, w_{ij}(z - c_{ij})\}. \tag{4.1}$$

Here, $\mathcal{P}^{(z)}$ denotes the set of all st-paths which have not any intersection with the set $\{(i, j) : w_{ij}^{(z)} = +\infty\} = \{(i, j) : z > c_{ij}^u\}$. In the special case that there is not any st-path for some value z, we set $b(z) = +\infty$. Since $b(z)$ is a parametric shortest path function in terms of z, it follows that it is non-decreasing and piecewise-linear, but unfortunately, it does not enjoy any convexity and concavity property in general. However, we show that $b(z)$ is concave in a small subinterval of $[\bar{z}^{min}, \bar{z}^{max}]$ containing the optimal value.

Before paying attention to these details, let us state a general description of our proposed algorithm which contains two phases. In the first phase, the binary search is applied to find a subinterval of $[\bar{z}^{min}, \bar{z}^{max}]$ in which $b(z)$ is concave. In the second phase, the exact optimal value is found by a new hybrid Secant-Newton algorithm.

Let us now return to analysing the behaviour of $b(z)$. It is obvious that $b(z)$ is increasing. Whenever z increases continuously, the breakpoints of $b(z)$ are created in one of three following situations.

- The shortest path is changed. So the slope of the corresponding line in $b(z)$ is either fixed or decreased.
- The shortest path is not changed. However, the weight of its one arc is increased from zero to a positive value. In this case, the slope of the corresponding line in $b(z)$ is increased.
- For an arc (i, j) of the shortest path P, we have $z > c_{ij}^u$. In this situation, the length of P grows unexpectedly to $+\infty$ because $w_{ij}^{(z)} = +\infty$. So the second shortest path is replaced to P. Consequently, the slope of the corresponding line in $b(z)$ is either unchanged or increased.

If the search interval $[\bar{z}^{min}, \bar{z}^{max}]$ is reduced so that the second and third cases never occur, then $b(z)$ is concave based on the first case. For this purpose, we reconsider the sorted list $z_1 < z_2 < \ldots < z_k$ of the set S. Then, we find an index $l \in \{1, 2, \ldots, k-1\}$ so that the optimal value belongs to $[z_l, z_{l+1})$. Obviously, the second and third cases does not occur in this interval. This procedure can be done

perfectly by Algorithm 1 under the assumption that $b_{ij}(z_l) = w_{ij}(z_l - c_{ij})$, instead of $b_{ij}(z) = \bar{w}_{ij}$. Hence, the first phase applies Algorithm 1 as a subroutine to find an interval $[z_l, z_{l+1})$ containing the optimal value. The following result states the concept used in the second phase.

Lemma 4.1. *If the optimal value z^* is less than $\bar{z}^{max}$, then $b(z^*) = B$.*

Proof. By definition, it is obvious that the function $b(z)$ is strictly increasing in $[\bar{z}^{min}, \bar{z}^{max})$, and it is constant elsewhere. Since $z^* < \bar{z}^{max}$, it follows that there is an index l so that $z^* \in [z_l, z_{l+1})$. Based on Lemma 2.2, we know that there is an *st*-path P^* so that $c^{(z^*,P^*)}$ is the optimal capacity vector.

Now assume that $b(z^*)$ is less than B by contradiction. So we can find a value $z' \in (z^*, z_{l+1})$ so that P^* is the shortest path with respect to $\mathbf{w}^{(z')}$ and $c^{(z',P^*)}$ satisfies the budget constraint. Since $c^{(z^*,P^*)}$ satisfies the bound constraints and both the values z^*, z' belong to $[z_l, z_{l+1})$, we imply that $c^{(z',P^*)}$ also satisfies the bound constraints. Hence, $c^{(z',P^*)}$ is a feasible capacity vector with the objective value greater than z^* which contradicts the optimality of $c^{(z^*,P^*)}$. □

Based on Lemma 4.1, problem (2.2) is reduced to finding a root of the equation $b(z) = B$ in the interval $[z_l, z_{l+1})$. Since $b(z)$ is strictly increasing in this interval, it follows that the root is unique. The equation $b(z) = B$ can be rewritten as

$$\min_{P\in\mathcal{P}^{(z)}} \sum_{(i,j)\in P: c_{ij}\le z} w_{ij}(z - c_{ij}) = B. \tag{4.2}$$

in which $\mathcal{P}^{(z)}$ is defined the same as 4.1. So it is dependent on z. Using the fact that the search interval is restricted to $[z_l, z_{l+1})$, we can write the equation in the following fashion:

$$\min_{P\in\mathcal{P}^{(z_l)}} \sum_{(i,j)\in P: c_{ij}\le z_l} w_{ij}(z - c_{ij}) = B. \tag{4.3}$$

Now suppose that (z^*, P^*) is an optimal solution of problem (4.3). Trivially, we have

$$\sum_{(i,j)\in P^*: c_{ij} > z_l} w_{ij}(z^* - c_{ij}) = B, \tag{4.4}$$

$$\sum_{(i,j)\in P: c_{ij} > z_l} w_{ij}(z^* - c_{ij}) \ge B, \qquad \forall P \in \mathcal{P}^{(z_l)}. \tag{4.5}$$

These relations imply that

$$\frac{B + \sum_{(i,j)\in P^*: c_{ij}\le z_l} w_{ij}c_{ij}}{\sum_{(i,j)\in P^*: c_{ij}\le z_l} w_{ij}} = z^* \le \frac{B + \sum_{(i,j)\in P: c_{ij}\le z_l} w_{ij}c_{ij}}{\sum_{(i,j)\in P: c_{ij}\le z_l} w_{ij}} \qquad \forall P \in \mathcal{P}^{(z_l)}. \tag{4.6}$$

So the following result is immediate.

Theorem 4.2. *If the optimal value of Problem (2.2) belongs to $[z_l, z_{l+1})$, then the problem is reduced to a minimum ratio path problem as follows:*

$$\min_{P\in\mathcal{P}^{(z_l)}} z = \frac{B + \sum_{(i,j)\in P: c_{ij}\le z_l} w_{ij}c_{ij}}{\sum_{(i,j)\in P: c_{ij}\le z_l} w_{ij}}. \tag{4.7}$$

Problem (4.7) is a minimum cost-reliability ratio path problem which is NP-hard in general because it is possible that there are some negative cycles in the network with respect to the cost vector $\mathbf{w}^{(z)}$ [1, 2]. Fortunately, since we have assumed that the costs $w_{ij}^{(z)}$ are nonnegative, the problem can be solved efficiently by Newton discrete-type method [16]. This is the same concept used by Zhang and Liu [26] to solve BCEPs. Let us recall their proposed method. It generates an increasing sequence $\{z^{(k)}\}_{k=0,1,\ldots}$ starting $z^{(0)} = z_l$. Suppose that the algorithm has computed the value $z^{(k)}$. To calculate the next element of the sequence, the algorithm solves a shortest path problem with respect to the nonnegative length vector

$$v = w^{(z^{(k)})}. \tag{4.8}$$

Assume that path P^* is found. Now, we may encounter two distinct situations:

- The length of P^* is less than B. In this case, we have

$$B > \sum_{(i,j)\in P^*} v_{ij} = \sum_{(i,j)\in P^*:c_{ij}\le z^{(k)}} w_{ij}(z^{(k)} - c_{ij}) \Rightarrow$$
$$z^{(k)} < \frac{B + \sum_{(i,j)\in P^*:c_{ij}\le z_l} w_{ij}c_{ij}}{\sum_{(i,j)\in P^*:c_{ij}\le z_l} w_{ij}},$$

  So we can set $z^{(k+1)}$ equal to the right-hand term of this inequality.
- The length of P^* is exactly equal to B. In this case, P is an optimal solution of problem (4.7) because P^* satisfies (4.4) and all st-paths satisfy (4.5) for $z^* = z^{(k)}$. So inequalities (4.6) hold for $z^* = z^{(k)}$.

It is remarkable that the case that the length of P^* is greater than B never occurs [26].

This algorithm generates an increasing sequence converging the optimal value. However, due to the special structure of $b(z)$, it always approaches to the optimal value only from one side, and it does not use points of the other side. For example, the sequence generated by algorithm present in [26] is as

$$z_l = z^{(0)} < z^{(1)} < z^{(2)} < \ldots < z^* < z_{l+1}.$$

So the algorithm does not use any point in $[z^*, z_{l+1}]$ to find z^*. Specially, if z^* is very close to z_{l+1}, then the algorithm has to approximately traverse all length of $[z_l, z_{l+1})$ to look for the optimal value. Here, we present a hybrid Secant-Newton algorithm to approach both the side of the optimal value. This algorithm generates two sequence $\{z^{(k)}\}$ and $\{y^{(k)}\}$ starting $z^{(0)} = z_l$ and $y^{(0)} = z_{l+1}$. Assume that we have computed the kth elements of both the sequences. Their next elements are calculated as follows:

$$y^{(k+1)} = \frac{y^{(k)}b(z^{(k)}) - z^{(k)}b(y^{(k)})}{y^{(k)} - z^{(k)}} - B, \tag{4.9}$$
$$z^{(k+1)} = \max\{\hat{z}^{(k)}, \hat{y}^{(k)}\}, \tag{4.10}$$

in which

$$\hat{y}^{(k)} = \frac{y^{(k)}b(y^{(k+1)}) - y^{(k+1)}b(y^{(k)})}{y^{(k)} - y^{(k+1)}} - B, \tag{4.11}$$
$$\hat{z}^{(k)} = \frac{B + \sum_{(i,j)\in P^*:c_{ij}\le z_l} w_{ij}c_{ij}}{\sum_{(i,j)\in P^*:c_{ij}\le z_l} w_{ij}}. \tag{4.12}$$

$y^{(k+1)}$ is obtained from solving the equation $L(z) = B$ where $L(z)$ is the line joining two points $(y^{(k)}, b(y^{(k)}))$ and $(z^{(k)}, b(z^{(k)}))$. This is motivated to use the well-known Secant method in computing the root of an equation. $\hat{z}^{(k)}$ is the same approximation obtained from the Newton method and $\hat{y}^{(k)}$ is the approximation obtained from the Secant method using two consecutive values $y^{(k)}$ and $y^{(k+1)}$. Finally, $z^{(k+1)}$ is the maximum value of two calculated approximation. Since $z^{(k+1)} \geq \hat{z}^{(k)}$, it follows that the number of iterations at the hybrid method is at most equal to that of the Newton method. So $\{z^{(k)}\}$ converges the optimal value at the end. Figure 1 depicts a visual description of the hybrid Secant-Newton method. It is easy to see that the sequence $\{z^{(k)}\}$ ($\{y^{(k)}\}$) is increasing (decreasing) and approaches to the optimal value from the left (right) side. The running time of an iteration of the hybrid algorithm is at most three times of the Newton method's because it calls the function $b(z)$ three times. Hence its worst-case complexity is the same as the Newton method. This proves the following result.

Theorem 4.3. *Algorithm 2 solves problem* (2.2) *in strongly polynomial time.*

Proof. The proof is immediate from the above discussion and the fact that the Newton method solves the problem in strongly polynomial time [26]. □

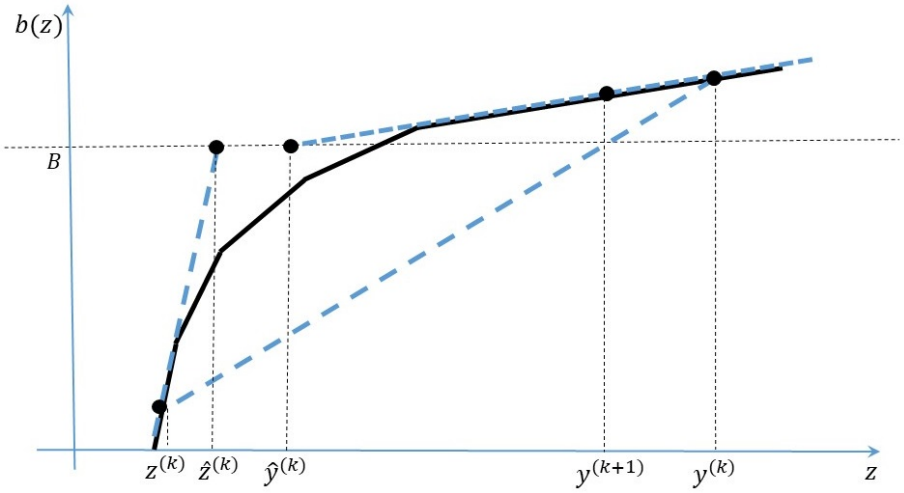

Figure 1. A visual description of the hybrid Secant-Newton method.

4.1. Computational and theoretical comparison

We now compare Algorithm 2 with the available algorithms in the literature. We recall that a modified binary search algorithm and a Newton discrete-type algorithm are presented respectively in [23] and [26] to solve BCEPs. Since MCPEP is a special case of BCEPs, we can apply them to solve MCPEP. Based on our preceding discussion, Algorithm 2 has the same worst-case complexity of the algorithm proposed in [26] which runs in strongly polynomial time. Moreover, the algorithm presented in [23] is only a (weak) polynomial time algorithm. This implies that Algorithm 2 as well as one proposed in [26] are theoretically better than one provided in [23].

Let us now compare them in practice. We recall that $b(z)$ is a linear-piecewise function. In instances of MCPEP for which $b(z)$ has a many number of breakpoints in the search interval $[z_l, z_{l+1}]$ of the

second phase, Algorithm 2 has a better performance than the others. This occurs specially whenever the upper bounds are infinity and the amount of budget is very large because in this case,

- The initial search interval $[z^{\min}, z^{\max}] = [z_0, z_k]$, specially subinterval $[z_{k-1}, z_k]$, is very wide;
- The search interval in the second phase is usually the same interval $[z_{k-1}, z_k]$ containing many numbers of $b(z)$'s breakpoints.

Let us state an example in this case. Consider an instance of MCPEP shown in Figure 2. Here, it is assumed that $B = 39$, $s = 0$ and $t = 37$. Moreover, all upper bounds are $+\infty$ and $w_{ij} = 0.01$ for every arc $(i, j) \in A$. In this example, we have $[z^{\min}, \min\{z^{\min} + \frac{B}{w_L}, z^{\max}\}] = [3200, 7100]$ at which the function $b(z)$ contains 8 breakpoints (see Figure 3). This interval is the same interval $[z_l, z_{l+1}]$ containing the optimal value. For this example, the algorithms were implemented on a laptop with an Intel Core $i5 - 2520M$ CPU (2.50GHz) and 4 GB of RAM. To code the algorithms, Python 2.7.5 together with the library NetworkX 1.8.1 were used. Table 1 compares the performance of the algorithms. It is obvious that Algorithm 2 has solved the problem in a less number of iterations.

Algorithm 2 A hybrid Secant-Newton method to solve problem (2.2) with linear costs

Input: A network $G(V, A, \mathbf{c}, \mathbf{w})$ with two specified nodes s and t
Output: An optimal st-path P^*, and the optimal value z^*.
Phase I:
Apply Algorithm 1 as a subroutine for arc costs $b_{ij}(z_l) = w_{ij}(z_l - c_{ij})$ to either detect the optimal value is z_{max} or find an interval $[z_l, z_{l+1})$ containing the optimal value. In the former, stop because the optimal value is found. In the latter, go to Phase II.
Phase II:
Set $z = z_l$.
Set $y = z_{l+1}$.
while True **do**
  Obtain the length vector $\mathbf{v}$ defined in (4.8) for $z^{(k)} = z$.
  Find shortest path P^* with respect to the length vector $\mathbf{v}$.
  if the length of P^* equals B **then**
    Stop because the optimal solution is found (see Lemma 4.1).
  else
    Set $y' = \frac{y^{(k)}b(z^{(k)}) - z^{(k)}b(y^{(k)})}{y^{(k)} - z^{(k)}} - B$.
    Set $z = \max\{\frac{yb(y') - y'b(y)}{y - y'} - B, \frac{B + \sum_{(i,j)\in P^*: c_{ij} \le z_l} w_{ij}c_{ij}}{\sum_{(i,j)\in P^*: c_{ij} \le z_l} w_{ij}}\}$.
    Set $y = y'$.
  end if
end while

Table 1. Comparing the algorithms in an instance of MCPEP.

	Number of iterations	Running time (sec.)	Values z in all iterations
Alg. in [23]	3	0.037	3900, 3875, 3950
Alg. in [26]	4	0.061	3633.33, 3875, 3930,3950
Alg. 2	1	0.013	3950

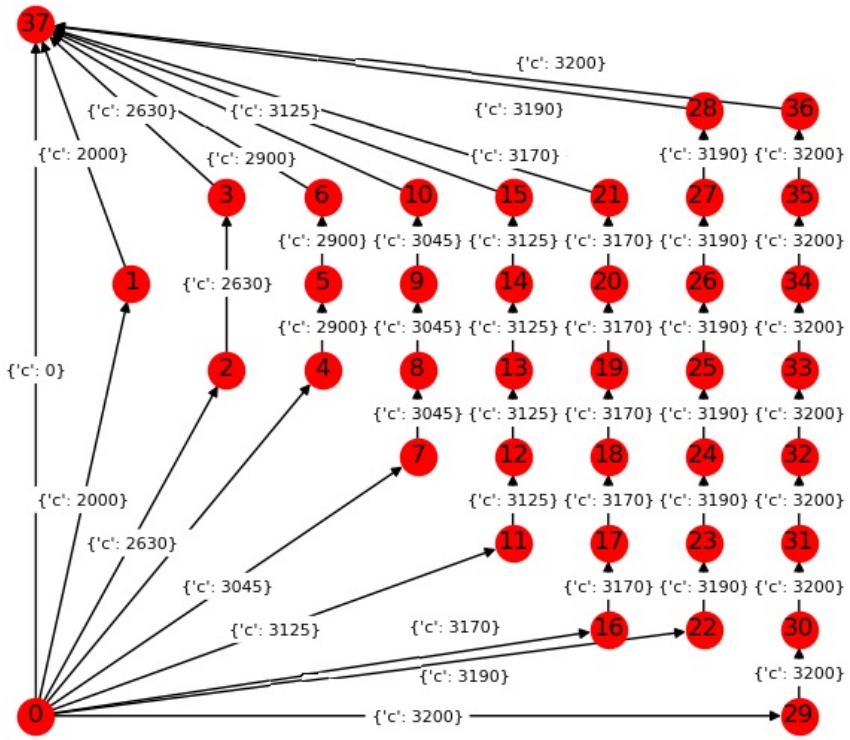

Figure 2. An instance of MCPEP with $s = 0$ and $t = 37$.

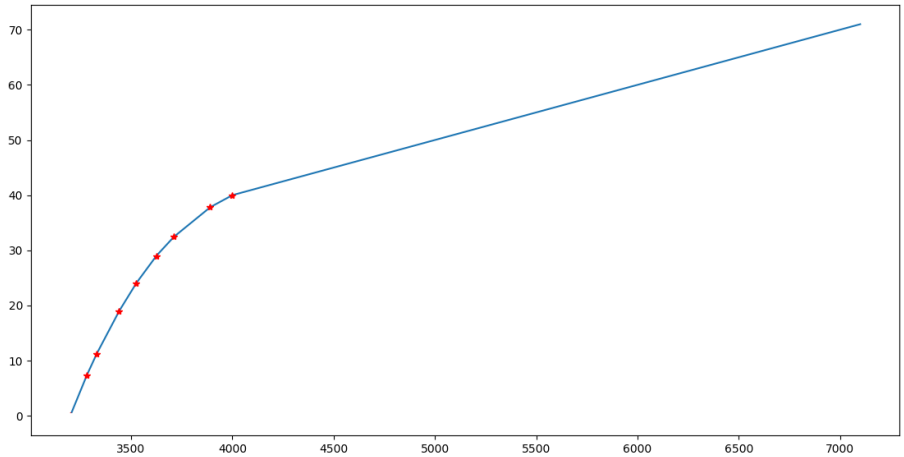

Figure 3. The graph of $b(z)$ at $[3200, 7100]$.

5. Conclusions

This paper studied maximum capacity path expansion problems with two types of modification costs: fixed cost, and linear cost. In the former, it proposed a strongly polynomial-time algorithm based on binary search to solve the problem. In the latter, it developed a two-phases algorithm. The first phase uses the first algorithm to find a small interval containing the optimal value at which the cost function is concave. Then, a hybrid Secant-Newton algorithm is proposed to obtain exactly the optimal value. The hybrid algorithm has a admissible performance rather than the available algorithms in the literature.

Since the hybrid algorithm can be applied to solve general fractional combinatorial optimization problems, it will be meaningful that one compares it with the other customary approaches, such as binary search, discrete-type Newton method, and Meggido's parametric search [16].

Although there is a variety of studies on capacity expansion problems, there are some aspects that are not considered until now (for example, max-min matching expansion problems, and max-min cut expansion problems). It will be worthwhile to focus on the other expansion problems in the future works.

Acknowledgments

We would like to thank the editor and anonymous reviewers for their constructive comments, which helped us to improve the paper.

Conflict of interest

We have no conflict of interest to declare.

References

1. R. K. Ahuja, Minimum cost-reliability ratio path problem, *Comput. Oper. Res.,* **15** (1988), 83–89.
2. R. K. Ahuja, J. L. Batra, S. K. Gupta, Combinatorial optimization with rational objective functions: A communication, *Math. Oper. Res.,* **8** (1983), 314–314.
3. R. K. Ahuja, T. L. Magnanti, J. B. Orlin, *Network flows: theory, algorithms, and applications*, USA: Prentice Hall, 1993.
4. B. Alizadeh, E. Afrashteh, F. Baroughi, Inverse obnoxious p-median location problems on trees with edge length modifications under different norms, *Theor. Comput. Sci.,* **772** (2019), 73–87.
5. O. Berman, G. Y. Handler, Optimal minimax path of a single service unit on a network to nonservice destinations, *Transport. Sci.,* **21** (1987), 115–122.
6. R. E. Burkard, Y. X. Lin, J. Z. Zhang, Weight reduction problems with certain bottleneck objectives, *Eur. J. Oper. Res.,* **153** (2004), 191–199.
7. Y. Chao, L. J. lin, A capacity expansion problem with budget constraint and bottleneck limitation, *Acta Math. Sci.,* **21** (2001), 428–432.
8. Y. Chao, Z. J. Zhong, On the bottleneck capacity expansion problems on networks, *Acta Math. Sci.,* **26** (2006), 202–208.

9. A. Ghate, Inverse optimization in semi-infinite linear programs, *Oper. Res. Lett.,* **48** (2020), 278–285.
10. K. Ghobadi, T. Lee, H. Mahmoudzadeh, D. Terekhov, Robust inverse optimization, *Oper. Res. Lett.,* **46** (2018), 339–344.
11. K. Kar, M. Kodialam, T. V. Lakshman, Minimum interference routing of bandwidth guaranteed tunnels with MPLS traffic engineering applications, *IEEE J. Sel. Area. Comm.,* **18** (2000), 2566–2579.
12. X. Y. Li, X. C. Shu, H. J. Huang, J. J. Bai, Capacitated partial inverse maximum spanning tree under the weighted Hamming distance, *J. Comb. Optim.,* **38** (2019), 1005–1018.
13. J. P. Li, J. P. Zhu, The capacity expansion path problem in networks, *J. Appl. Math.,* **2013** (2013), 156901.
14. A. Mohammadi, J. Tayyebi, Maximum capacity path interdiction problem with fixed costs, *Asia-Pac. J. Oper. Res.,* **36** (2019), 1950018.
15. A. P. Punnen, A linear time algorithm for the maximum capacity path problem, *Eur. J. Oper. Res.,* **53** (1991), 402–404.
16. T. Radzik, Fractional combinatorial optimization, In: *Handbook of combinatorial optimization*, Boston, MA: Springer, 1998, 429–478.
17. J. Tayyebi, M. Aman, On inverse linear programming problems under the bottleneck-type weighted Hamming distance, *Discrete Appl. Math.,* **240** (2018), 92–101.
18. J. Tayyebi, A. Mohammadi, S. M. R. Kazemi, Reverse maximum flow problem under the weighted Chebyshev distance, *RAIRO-Ope. Res.,* **52** (2018), 1107–1121.
19. J. Tayyebi, M. Aman, Efficient algorithms for the reverse shortest path problem on trees under the hamming distance, *Yugoslav J. Oper. Res.,* **27** (2016), 46–60.
20. J. Tayyebi, A. Deaconu, Inverse generalized maximum flow problems, *Mathematics,* **7** (2019), 899.
21. A. Tayyebi, A. R. Sepasian, Partial inverse min-max spanning tree problem, *J. Comb. Optim.,* **40** (2020), 1075–1091.
22. C. Yang, J. Z. Zhang, A constrained capacity expansion problem on networks, *Int. J. Comput. Math.,* **70** (1998), 19–33.
23. J. Z. Zhang, C. Yang, Y. X. Lin, A class of bottleneck expansion problems, *Comput. Oper. Res.,* **28** (2001), 505–519.
24. B. W. Zhang, X. C. Guan, P. M. Pardalos, C. Y. He, An algorithm for solving the shortest path improvement problem on rooted trees under unit hamming distance, *J. Optim. Theory Appl.*, **178** (2018), 538–559.
25. B. W. Zhang, J. Z. Zhang, L. Q. Qi, The shortest path improvement problems under Hamming distance, *J. Comb. Optim.,* **12** (2006), 351.
26. J. Z. Zhang, Z. H. Liu, An oracle strongly polynomial algorithm for bottleneck expansion problems, *Optim. Methods Software,* **17** (2002), 61–75.