20th International Symposium on Transportation and Traffic Theory

Bicriterion shortest path problem with a general nonadditive cost

Peng (Will) Chen\textsuperscript{a}, Yu (Marco) Nie\textsuperscript{a,*}

\textsuperscript{a}Department of Civil and Environmental Engineering, Northwestern University
2145 Sheridan Road, Evanston, IL 60208, United States

Abstract

A bicriterion shortest path problem with a general nonadditive cost seeks to optimize a combination of two path costs, one of which is evaluated by a nonlinear function. This paper first identifies a number of emerging transportation applications for which such a shortest path problem might be considered a core subproblem. We propose to first approximate the general nonlinear cost function with a piecewise linear counterpart, and then solve each linear subproblem sequentially. A specialized algorithm is developed to solve the subproblems, which makes use of the efficient path set (or the convex hull) to update upper and lower bounds of the original problem. Conditions under which the solution to a subproblem must belong to the efficient path set are specified. Accordingly, we show that the optimal path must be efficient if the nonlinear cost function is concave. If the optimal path to a subproblem is not efficient, partial path enumeration, implemented using a simple K-shortest path ranking procedure, is conducted to close the gap. The proposed algorithm includes strategies aiming to expedite path enumeration by using upper bounds derived from the efficient path set. Numerical experiments are conducted to demonstrate correctness and effectiveness of the proposed algorithm.

1. Introduction

Shortest path problems are among the most studied optimization problems. In transportation, finding shortest paths of various sorts lies at the core of many applications, ranging from in-vehicle route guidance to regional planning activities that involve billions dollars in infrastructure investment. This paper considers a special and important instance of these problems, in which the decision makers have to strike a balance between two attributes of the paths under consideration while valuing one of the attributes nonlinearly. Specifically, the decision makers seek to minimize $P_1^k + h(P_2^k)$ where $P_i^k$ denote ith property of path k and $h$ is a general nonlinear cost function.
function. By *general*, we mean that the only restriction is that \( h \) has to be continuous. In contrast, most studies in the literature require \( h \) to be monotone at minimum. The assumption of monotonicity may be too restrictive in some real world applications, as explained in Section 3.

Variants of the bicriterion shortest path problem concerned in this paper have been studied in the literature for years. Dial (1979) proposes an algorithm to construct a set of *efficient* (or extreme) paths, which are such defined that no other paths provide a better overall cost for any linear combination of the attributes. Dial’s algorithm has since been adopted for solving multiclass traffic assignment problems that consider heterogeneity in users’ valuation of time (see e.g. Leurent, 1993; Dial, 1997; Marcotte et al., 1996). However, the objective function in Dial’s bicriterion problem is linear in both attributes, and hence additive over paths. Henig (1985) provides a comprehensive treatment of a bicriterion shortest path problem with continuous monotone functions. Similar to Dial (1979), his algorithm also starts from finding a set of efficient paths. Henig (1985) shows that the optimal path is always efficient only if the objective function to be minimized is quasiconcave. For quasiconvex functions, a line search procedure is proposed to locate the efficient path that admits the best upper bound and to further close the gap a search for K-shortest paths is recommended. Revisiting Henig’s problem, Mirchandani & Wiecek (1993) show that for any monotone quasiconvex function the optimal path must be a non-dominated path. They also refine Henig’s linear search procedure for this case. Tsaggouris & Zaroliagis (2004) develop an exact algorithm for the nonadditive bicriterion shortest path problem, assuming \( h(\cdot) \) to be convex and non-decreasing. Their algorithm consists of two phases. The first solves a Lagrangian relaxation of the original problem, which is equivalent to computing the best efficient path. Phase I ends with either an optimal solution or a duality gap. If a non-zero gap is found (i.e. the optimal path is not efficient), the second phase of the algorithm closes it using a path enumeration procedure based on branch-and-bound. Gabriel & Bernstein (1999) propose a feasible direction algorithm for solving the problem without linearizing \( h(\cdot) \). Their algorithm moves around extreme points generated by a Frank-Wolfe type linear subproblem (Frank & Wolfe, 1956). Whenever the next extreme point fails to improve the upper bound, a heuristic line search procedure, which itself involves solving a sequence of nontrivial LP, is invoked to find a non-extreme path. It is unclear whether the procedure ensures to find such a path (if one exists), and if it does not how large the gap would be. The bicriterion shortest path problem concerned in this paper is also closely related to the constrained shortest path problem (Ahuja et al., 1993). The reader is referred to Carlyle et al. (2008) for latest developments in that area. In essence, the proposed algorithm converts the original problem into a series of additive bicriterion constrained shortest path problem that is relatively easy to solve.

The bicriterion shortest path problem with a general nonadditive cost finds numerous applications in network optimization. One focus of this paper is to identify and discuss representative applications. Nonadditive path cost may arise from nonlinear valuation of path attributes, such as travel time (the fact that 60 one-minute blocks of time is less valuable than one 60-minute block of time) and emissions (Gabriel & Bernstein, 1997b; Hjorth & Fosgerau, 2012). Accordingly, the bicriterion shortest path problem concerned in this paper is a core subproblem in finding equilibrium solutions to the traffic assignment problem with such nonadditive cost (see e.g. Gabriel & Bernstein, 1997a,b; Lo & Chen, 2000; Ageppa et al., 2007; Chen et al., 2010). Nonlinear congestion pricing (Maruyama & Sumalee, 2007; Lawphongpanich & Yin, 2012), in which the road toll is treated as a general nonlinear function of travel distance, also requires solving such shortest path problems repeatedly. In particular, Lawphongpanich & Yin (2012) adopt a two-part piecewise linear tolling function. In fact, even when the road toll is additive over path, the cost associated with the toll may be not. One example has to do with the design of tradable mobility credit schemes (e.g. Yang & Wang, 2011; Nie, 2012). In this case, travellers would receive certain mobility credits and hence only need to pay a fraction of all the tolls on the path. Accordingly, the cost of toll remains at zero until the required path toll exceeds the value of the acquired mobility credits. Nonlinear path cost may also arise when the schedule cost is imposed on a path along with a time window (Nie et al., 2012). Typically such a schedule cost takes a skewed “V” shape
when both early and late arrivals are penalized (e.g. Vickrey, 1969; Small, 1982).

The other focus of this paper is developing efficient solution techniques for the bicriterion shortest path problem. The proposed algorithm is based on approximating $h$ with a piecewise linear function. Although only two or three segments are typically needed to approximate most functions of practical interest, the methodology can handle as many segments as desired. Once linearized, the bicriterion shortest path problem is solved sequentially for each segment, and the best upper bound is updated accordingly. Thus, the key to computational efficiency is how to quickly solve each subproblem. To this end, a specialized method is developed that relies on the efficient path set (or the convex hull) to provide upper and lower bounds. We specify conditions under which the solution to the subproblem must belong to the efficient path set. Accordingly, a well known result in the literature is verified in the linearized case, that the optimal path must be efficient if $h$ is concave. When the above conditions are not met, partial path enumeration has to be conducted to close the gap. The paper will also discuss strategies aiming to minimize the efforts for such path enumeration by making use of upper bounds available from the efficient path set. The proposed algorithm shares a similar structure with several predecessors in the literature (Henig, 1985; Mirchandani & Wiecek, 1993; Tsaggouris & Zaroliagis, 2004), in that it utilizes efficient paths to guide the search, and relies on partial path enumeration to close the gap. The main differences of this work are (1) the consideration of general nonlinear cost represented with piecewise linear functions, (2) efficient partial path enumeration guided by appropriate upper bounds, and (3) graphical illustration that provides intuitive justification of the proposed method.

For the remainder, Section 2 formally defines the problem and presents a mathematical formulation. Section 3 surveys four applications that require solving the bicriterion shortest path problem considered in this paper as a subproblem. Section 4 defines the efficient path set and briefly reviews Dial-Henig algorithm. Section 5 presents the main results, including both the solution algorithm and its justification. Results of numerical experiments are reported in Section 6 and Section 7 concludes the paper.

2. Problem formulation

Consider a directed and connected network $G(N,A)$, where $N$ represents a set of nodes and $A$ represents a set of links. Let $d_a$, $\tau_a$ and $u_a$ be the length, travel time and monetary cost associated with link $a \in A$ where $d_a$, $\tau_a$ and $u_a$ are assumed to be nonnegative. A path for a given origin-destination (O-D) pair $r-s$ is denoted using $k$, and the length, travel time and monetary cost associated with the path $k$ are given by

$$l_k = \sum_{a \in A} \delta_k^a d_a; \quad t_k = \sum_{a \in A} \delta_k^a \tau_a; \quad c_k = \sum_{a \in A} \delta_k^a u_a$$

where $\delta_k^a = 1$ if link $a$ is on path $k$ and 0 if not. The set of all paths that connect an O-D pair $r-s$ is denoted by $K$. We are interested in finding the path between an O-D pair $r-s$ to minimize $P_k^1 + h(P_k^2)$, subject to: $k \in K$ (1)

where $P_k^i = \sum_a \delta_k^a p_i^a$ are $i$th cost of traversing path $k$ ($i = 1, 2$), and $h(\cdot) : \mathbb{R}^+ \rightarrow \mathbb{R}$ is a general continuous function that transforms $P_k^2$ to facilitate the trade-off between the two costs. The above formulation fits into a variety of applications. For example, if one interprets $P_k^1$ as $c_k$ and $P_k^2$ as $l_k$, the function $h$ can be considered as an evaluation of travel time in the monetary cost. Another possibility is to consider $P_k^1$ as $l_k$ and $P_k^2$ as $c_k$, in which case $h$ can be interpreted as a distance-based toll measured in the unit of time. For narrative convenience, hereafter we shall consider $h$ as a function of $l_k$ and $c_k$ is the other path cost, unless otherwise specified. Problem (1) is known to be NP-complete (Henig, 1985).
As \( h \) is treated as a general continuous function, we propose to approximate it with a piecewise linear function \( H \). Specifically, the feasible range for \( t \) is divided into \( m \) intervals denoted as \([L_j, U_j]\) where \( L_1 = 0, U_m = B \), and \( L_j = U_{j-1} \) for \( j = 2, \ldots, m \). We note that \( B \) may be naturally available for a given application or may be selected at the modeller’s discretion. For each segment \( j \), the slope of the line, denoted as \( \gamma_j \), can be obtained as

\[
\gamma_j = \frac{h(U_j) - h(L_j)}{U_j - L_j}
\]

(2)

The line intersects with the vertical axis (\( t = 0 \)) at

\[
\alpha_j = h(L_j) - \gamma_j L_j
\]

(3)

Therefore, we can write the piecewise linear function \( H(t) \) as follows:

\[
H(t) = \alpha_j + \gamma_j t; \quad \text{if } t \in [L_j, U_j], j = 1, \ldots, m
\]

(4)

It is worth noting that \( m \) is not expected to be a large number in real applications. In most cases, discretizing \( t \) into two or three pieces would suffice to capture the nonlinear behaviour of practical significance.

We emphasize that an important difference of Problem (1) in comparison with those in the literature has to do with the generalized definition of \( h \). To the best of our knowledge, most existing studies require \( h \) to be smooth and monotonically increasing (see e.g. Dial, 1979; Henig, 1985; Mirchandani & Wiecek, 1993; Gabriel & Bernstein, 1999), whereas what is required herein is nothing more than continuity. This generalization lends itself to tackling several transportation applications that arise in the recent literature, to which we now turn.

3. Applications

For narrative convenience, let us first introduce some additional notations. Let \( x_a \) and \( \tau_a(x_a) \) be the traffic flow and travel time on link \( a \in A \), where \( \tau_a(\cdot) \) is a non-negative, strictly increasing and convex function. To be consistent with the typical setting in the shortest path problem, a single O-D pair \( r - s \) is considered in the applications discussed in the following. Extensions to multiple O-D pairs, however, are straightforward. Let the travel demand between the O-D pair be denoted as \( q \). The set of paths that connect the O-D pair is denoted by \( K \), and the flow on path \( k \) \( \forall k \in K \) is represented by \( f_k \).

3.1. Nonadditive traffic equilibrium problem

Gabriel & Bernstein (1997a) considers a general path cost of the following form

\[
g_k = c_k + h(t_k)
\]

(5)

where \( c_k \) is the monetary path cost, and \( h \) is a nonlinear function that converts time to money. The traffic equilibrium problem with the above path cost function is typically formulated as a variational inequality or a nonlinear complementarity problem, and solved by column generation, which iteratively generates paths required to solve the problem. In the column generation, the path with minimum \( g_k \) is found by solving a shortest path problem of form (1).
3.2. Distance-based congestion pricing

The congestion pricing model of Lawphongpanich & Yin (2012) assumes the amount of toll is a nonlinear function of the distance travelled inside tolled areas. The simplest version of the tolled user equilibrium traffic assignment problem may be written as follows:

\[
\begin{align*}
\min_z & \sum_a \int_0^{x_a} \tau_a(w)dw + \sum_{k \in K} T(l_k) f_k \\
\text{subject to:} & \\
\sum_k f_k &= q \\
\sum_k f_k \delta^k_a &= x_a \quad \forall a \in A \\
f_k &\geq 0 \quad \forall k \in K
\end{align*}
\]

where \(T(l_k)\) is a distance-based toll and the function \(T\) takes the following form

\[
T(l_k) = \begin{cases} 
T(l_k) \text{ or } \overline{T}(l_k) & l_k > 0 \\
0 & l_k \leq 0
\end{cases} \quad \overline{T}(l_k) = \max\{\alpha_1 + \beta_1 l, \alpha_2 + \beta_2 l\} \quad \underline{T}(l_k) = \min\{\alpha_1 + \beta_1 l, \alpha_2 + \beta_2 l\}
\]

It is well known that solving (6) typically involves iteratively solving a linearized subproblem (Frank & Wolfe, 1956)

\[
\begin{align*}
\min & \sum_k (t_k + T(l_k)) f_k \\
\text{subject to} & (6b) - (6d),
\end{align*}
\]

which in turn can be solved as a nonadditive shortest path problem of form (1).

3.3. Optimal path problem considering schedule penalty

When on-time delivery is important, decision makers may choose to impose a penalty cost on both late and early arrivals, which leads to a special instance of optimal path problem that takes the following form:

\[
\min a t_k + h(t_k) \text{ subject to: } k \in K
\]

where \(h(\cdot)\) is the schedule cost function, and \(a\) may be interpreted as the value of time. Denoting by \(t_0\) the scheduled travel time, the penalty cost may be estimated using the following piecewise linear function

\[
h(t_k) = \begin{cases} 
\theta(t_0 - t_k) & t_k \leq t_0 \\
\gamma(t_k - t_0) & t_k > t_0
\end{cases}
\]

where \(\theta\) and \(\gamma\) are the unit early and late arrival costs. Note that the above function is convex but not monotone, since the early arrival corresponds to a negative slope \((-\theta)\) and the late arrival corresponds to a positive slope \((\gamma)\). The schedule cost of the above form is widely used in the morning commute analysis (Vickrey, 1969; Henderson, 1974; Hendrickson & Kocur, 1981), in which commuters choose their departure time based on the trade-off between travel delays and schedule costs. Similar schedule cost functions have also been used in vehicle routing problems with time windows (Ando & Taniguchi, 2006) and optimal path problems with second order stochastic dominance constraints (Nie et al., 2012).
3.4. Tradable credit problem considering transaction cost

Nie (2012) studies a congestion pricing scheme coupled with a market that allows travellers to trade mobility credits. In such a system, each traveller initially receives a same amount of mobility credits, denoted using $\phi$, from the toll authority and then trade them with each other through negotiation. For each unit of credit traded, both the buyer and the seller have to pay $p_t$ for the brokerage service. The travellers have the option to buy credit from the authority at a price $p_g$ to fulfil their unmet mobility needs. Since the total number of travellers in the system is $q$, the total credits issued by the authority is $q\phi = \Pi$. Let $y_k$ denote the travellers on path $k$ who purchase credits from the market, $\nu_k = \sum_a \delta^k_a u^a$ denote the required credits to use path $k$, and $\rho_k = \nu_k - \phi$ denote the extra credits one has to purchase, either from the market or the authority, when travelling on path $k$. The traffic and credit-trading equilibrium of this system can be obtained by solving the following model:

$$ \begin{align*} 
\min & \sum_a \int_0^{x_a} \tau_a(w)dw + \sum_k (p_g(f_k - y_k)[\rho_k]^+ + y_k|\rho_k|p_t) \\
\text{subject to:} & \\
\sum_k f_k &= q \\
\sum_k f_k \delta^k_a &= x_a \quad \forall a \in A \\
\sum_k y_k \rho_k &\leq 0 \\
y_k &\leq f_k, f_k \geq 0, \quad y_k \geq 0 \quad \forall k \in K 
\end{align*} $$

(11a)

where $[a]^+ = a$ if $a \geq 0$, and 0 otherwise. As indicated in the second term of the objective function, the travellers who abandon the market would purchase credits from the authority only when his/her initial endowment $\phi$ is not enough to cover all credit charges. The third term of the objective function ensures that both buyers and sellers in the market pay for transactions in proportion to the amount of trading. Constraint (11d) requires that the credits needed by all those who opt to use the market should not exceed the sum of their own initial endowments. In other words, either the market is cleared (i.e. credits sold exactly balance credits purchased), or no trading should take place at all.

Dualizing Constraint (11d) with a multipiler $p$ and linearizing (11) lead to

$$ \begin{align*} 
\min & (t_k + h_1(\rho_k))f_k + h_2(\rho_k)y_k \\
\text{subject to:} & (11b) \\
0 &\leq y_k \leq f_k, \forall k \in K 
\end{align*} $$

(12)

where

$$ h_1(\rho_k) = p_g|\rho_k|; \quad h_2(\rho_k) = -p_g|\rho_k| + |\rho_k|p_t + p\rho_k. $$

If $p_g \gg p + p_t$, $y_k$ would have to take the upper bound ($f_k$) to minimize the total cost. In this case, the objective function becomes $(t_k + h_1(\rho_k) - h_2(\rho_k))f_k = (t_k + |\rho_k|p_t + p\rho_k)f_k$. This problem has the exact structure as (1) (with a cost function associated with $\rho_k$ taking a V shape), which can be tackled with the solution techniques presented herein.

4. Efficient path set

Suppose now that travellers would choose paths based on a general cost with a linear function $h$, defined by

$$ g_k = c_k + h(t_k) = c_k + \gamma t_k, $$

(14)
where $\gamma \in \mathbb{R}$ is a real scalar that converts travel time to an equivalent monetary cost. An efficient path is formally defined as follows in this paper.

**Definition 1 (Efficient path).** A path $k$ is efficient if (1) it is simple, i.e. it does not contain any cycles and (2) for some $\gamma \in \mathbb{R}$, there exists no other simple path $k'$ such that $g_{k'} < g_k$.

Simply speaking, an efficient path $k$ must have minimum cost $g_k$ for some $\gamma$ among all simple paths between the O-D pair. Unlike those in the literature (e.g. Dial, 1979; Mirchandani & Wiecek, 1993), the above definition of efficiency requires explicit acyclicity because $\gamma$ is allowed to take negative values here. As cyclic paths are excluded, the set of efficient paths, denoted as $E_{rs}$, is always finite. Define two special path sets $K^1$ and $K^2$, where

$$K^1 = \text{argmin}\{t_k, k \in K\}; \quad K^2 = \text{argmin}\{c_k, k \in K\}$$

Clearly, $K^1, K^2 \subseteq E_{rs}$. Further, define paths

$$k_1 = \text{argmin}\{c_k, k \in K^1\}; \quad k_2 = \text{argmin}\{t_k, k \in K^2\},$$

and for notational convenience let $\bar{t} \equiv t_{k_1}, \bar{t} \equiv t_{k_2}$. It is trivial to show that $t \leq \bar{t}$. As demonstrated in Dial *et al.* (1979), it is useful to visualize the concept of efficient path set using a two-dimensional plot as shown in Figure 1. Each point in Figure 1 represents a simple path with its $x$ and $y$ coordinates being $t$ and $c$ respectively. The solid line shows the convex hull of all points. The paths corresponding to the points on the convex hull are efficient paths (Mirchandani & Wiecek, 1993). Note that the efficient paths can be distinguished according to whether or not their corresponding travel times are larger than $\bar{t}$. Formally, we define

$$E_{rs}^+ = \{k | t_k \leq \bar{t}, k \in E_{rs}\}; \quad E_{rs}^- = \{k | t_k > \bar{t}, k \in E_{rs}\}$$

Clearly, $E_{rs}^+$ and $E_{rs}^-$ correspond to the left and right sides of the convex hull, respectively. For any $\gamma \in \mathbb{R}$, the general cost $g_k$ may be interpreted as the $c$-intercept of a straight line $g = c + \gamma t$ which passes through the point $(t_k, c_k)$, see Figure 1. Note that the slope of the line equals $-\gamma$. If $\gamma \geq 0$, $g_k$ is minimized if one shifts the line up until it first hits a path in $E_{rs}^+$ (i.e. left side of
the convex hull), see Figure 1. When $\gamma < 0$, the same maneuver would identify a path in $E_{rs}^-$. Yet, this path may not minimize $g_k$ if cycles exist and are allowed at optimality. In this paper, we exclude cyclic paths from consideration, even if they may give lower value of $g_k$ for a given $\gamma$. We will explain how this is accomplished in Section 5 when the algorithm is presented.

If the efficient path set $E_{rs}$ is available, a minimum cost simple path for a given $\gamma$ can be obtained by simply comparing $\gamma$ with the slopes of the line segments on the convex hull. To formalize the presentation of the procedure, let $E_{rs} = \{e_1, e_2, \ldots, e_n\}$ where $t_{e_1} < t_{e_2} \cdots < t_{e_n}$, and $\beta_i$ ($i = 1, 2, \ldots, n - 1$) denote the slope of the line segment connecting path $e_i$ and path $e_{i+1}$. Note that $\beta_1 < \beta_2 < \cdots < \beta_n$.

Algorithm 1: Finding a minimum cost simple path $e^*$ for a given $E_{rs}$

**Inputs:** $\gamma$

**Outputs:** $e^*$

**Step 1** If $\gamma > -\beta_1$, $e^* = e_1$

Else if $\gamma < -\beta_{n-1}$, $e^* = e_n$

Else if $-\beta_{i} < \gamma < -\beta_{i-1}$, $e^* = e_i$

Else ($\gamma = -\beta_i$), $e^* = e_i$ or $e^* = e_{i+1}$.

Certainly, if one is only interested in optimizing a linear $g_k$ of form (14), a standard shortest path algorithm would suffice when $\gamma \geq 0$ or when the network does not contain any cycle $p$ such that $c_p + \gamma t_p < 0$. However, generating and retaining $E_{rs}$ is useful for general cost function $h(\cdot)$.

The question that may be raised now is how $E_{rs}$ can be generated. The algorithm for generating $E_{rs}^+$ is readily available in the literature (see e.g. Dial, 1979; Henig, 1985). For $E_{rs}^-$, no efficient algorithm exists, again because negative $\gamma$ may create negative cycles. One may use a partial path enumeration to generate simple efficient paths when $\gamma < 0$, but the heavy computational overhead makes it an unattractive option. Therefore, we will not explicitly generate these efficient paths before hand. The Dial-Henig algorithm for generating $E_{rs}^+$ is briefly reviewed below for the readers' convenience.

Algorithm 2: Dial-Henig algorithm for finding $E_{rs}^+$

**Step 0** Find $k_1^1 \in K^1$ and $k_2^1 \in K^2$ using the standard shortest path algorithm. Set $E_{rs}^+ = \{k_1^1, k_2^1\}$ and $n = 2$.

**Step 1** Set the number of newly generated efficient paths $n_0 = 0$. For $i = 1$ to $n - 1$, do the following before moving to Step 2.

Consider the pair of paths $e_i$ and $e_{i+1}$. Set

$$\gamma = -\frac{c_{e_{i+1}} - c_{e_i}}{t_{e_{i+1}} - t_{e_i}}$$

and find the path $e_0$ that minimizes $c_k + \gamma t_k$ by the standard shortest path algorithm. If $e_0 \notin E_{rs}^+$, add it in between $e_i$ and $e_{i+1}$, set $n_0 = n_0 + 1$; otherwise, move to next $i$.

**Step 2** If $n_0 > 0$, set $n = n + n_0$ and then return to Step 1; otherwise, determine $k_1$ and $k_2$ from $E_{rs}$, remove all other paths in $K^1$ and $K^2$ from $E_{rs}^+$, and terminate.

The above algorithm is computationally efficient when the number of efficient paths is relatively small, because it only requires solving a number of classical shortest path problems.

---

1When more than one path in $E_{rs}$ has exactly same $t$ and $c$, only one path is retained in the efficient path set to ensure strict inequality. In the rare event that $K^1$ contains more than one path with exactly same travel time, only $k_1$ is retained.
5. Solution algorithm

We are ready now to present an algorithm for solving Problem (1). We shall assume that \( h \) is linearized as in (4) and that the efficient path set associated with all \( \gamma \geq 0 \), i.e. \( E^+_{rs} \), is generated using the algorithm presented in the previous section. Due to discretization, the original problem can be decomposed into a sequence of subproblems as follows:

\[
\min_{k \in K} z_j = c_k + \gamma_j t_k
\]

subject to: \( t_k \in [L_j, U_j] \)  

Then, the optimal solution to Problem (1) can be found by solving the following problem:

\[
z = \min_{j=1,\ldots,m} \alpha_j + z^*_j
\]

where \( z^*_j \) is the optimal solution to the \( j \)th subproblem.

Notice that each subproblem (16) has a similar structure as the shortest path problem considered in Section 4 except that it has a constraint on \( t \). Recognizing this similarity, the algorithm proposed in the following takes advantage of the structure of the efficient paths whenever possible. We note that the algorithm aims to find an “exact” solution, but it will become clear later that this goal is achievable only if full path enumeration is allowed. The description of the algorithm is given below, followed by its justification.

5.1. Description

First, let us define \( E^j_{rs} \) as the \( j \)th subset of the efficient path set \( E_{rs} \), where

\[
E^j_{rs} = \{ k \in E_{rs} | L_j \leq t_k \leq U_j \}
\]

\( E_{rs} \) is the partial efficient path set associated with the \( j \)th subproblem, which might be empty in some cases. Further, define two paths \( k^L_j \) and \( k^U_j \) as

\[
k^L_j = \arg\min_{k} \{ t_k, k \in E_{rs} | t_k \geq L_j \} \quad \text{and} \quad k^U_j = \arg\max_{k} \{ t_k, k \in E_{rs} | t_k \leq U_j \}.
\]

Simply speaking, \( k^L_j \) is the closest efficient path to the right of the vertical line \( t = L_j \) and \( k^U_j \) is the closest efficient path to the left of the vertical line \( t = U_j \). If there is only one efficient path within interval \( j \), then \( k^L_j = k^U_j \). Also worth noting is that \( k^L_j \) and \( k^U_j \) may exist outside of the interval \( j \) if \( E^j_{rs} = \emptyset \). Corresponding to each efficient path \( e \), let \( \beta^-_e \) and \( \beta^+_e \) denote the slopes of the left and right line segments on the convex hull, respectively. For the efficient path \( e \in K^1 \), define \( \beta^-_e = -\infty \). Finally, \( z^U_j, z^L_j \) and \( e \) denote the upper bound, lower bound and the gap of the \( j \)th subproblem, respectively (\( z^U_j = z^L_j + e \)).

Algorithm 3: Solving the linearized non-additive bicriterion shortest path problem.

**Outputs:** \( z^*, e^*, k^* \)

**Step 0** Initialization
Set the best current solution \( z^* = \infty \), the corresponding gap \( e^* = \infty \), and \( j = 1 \);

**Step 1** Solve the \( j \)th subproblem

**Case 0:** \( L_j < U_j < t \)
Do nothing;
Case 1: \( t \leq U_j \leq 7 \)

Case 1.1: If \( E_{\text{in}} \neq \emptyset \),

\[ \rightarrow \text{Case 1.1.1: If } -\beta_{k_j}^+ \leq \gamma_j \leq -\beta_{k_j}^- \]

\[ \rightarrow \text{call Algorithm 1 with the input } (\gamma_j) \text{ to obtain } k_j^* \text{, set } z_j^* = c_{k_j^*} + \gamma_j t_{k_j^*} \]

\[ \rightarrow \text{if } \alpha_j + z_j^* < z^* \text{, set } z^* = \alpha_j + z_j^*, e^* = 0, k^* = k_j^* \]

\[ \rightarrow \text{Case 1.1.2: Else if } \gamma_j > -\beta_{k_j}^- \]

\[ \rightarrow \text{call Algorithm 4 with inputs } (z^*, k_j^L, \text{Left}, \beta_{k_j}^-) \text{ to update } z^*, e^* \text{ and } k^*. \]

\[ \rightarrow \text{Case 1.1.3: Else,} \]

\[ \rightarrow \text{call Algorithm 4 with inputs } (z^*, \emptyset, \text{Center}, \beta_{k_j}^+) \text{ to update } z^*, e^* \text{ and } k^*. \]

Case 1.2: Else,

\[ \rightarrow \text{call Algorithm 4 with inputs } (z^*, \emptyset, \text{Center}, \beta_{k_j}^+) \text{ to update } z^*, e^* \text{ and } k^*. \]

Case 2: \( L_j \leq t < U_j \)

\[ \rightarrow \text{Case 2.1: If } 0 \leq \gamma_j \leq -\beta_{k_j}^- \]

\[ \rightarrow \text{call Algorithm 1 with the input } (\gamma_j) \text{ to obtain } k_j^* \text{, set } z_j^* = c_{k_j^*} + \gamma_j t_{k_j^*} \]

\[ \rightarrow \text{if } \alpha_j + z_j^* < z^* \text{, set } z^* = \alpha_j + z_j^*, e^* = 0, k^* = k_j^* \]

\[ \rightarrow \text{Case 2.2: Else if } \gamma_j > -\beta_{k_j}^- \]

\[ \rightarrow \text{call Algorithm 4 with inputs } (z^*, k_j^L, \text{Left}, \beta_{k_j}^-) \text{ to update } z^*, e^* \text{ and } k^*. \]

\[ \rightarrow \text{Case 2.3: Else,} \]

\[ \rightarrow \text{call Algorithm 4 with inputs } (z^*, k_j^L, \text{Right}, 0) \text{ to update } z^*, e^* \text{ and } k^*. \]

Case 3: \( t < L_j < U_j \)

\[ \rightarrow \text{call Algorithm 4 with inputs } (z^*, \emptyset, \text{Center}, 0) \text{ to update } z^*, e^* \text{ and } k^*. \]

Step 2: If \( j < m \), set \( j = j + 1 \) and return to Step 1; otherwise terminate.

Algorithm 4: Partial path enumeration for given \( k_j^L, k_j^U, L_j, U_j, \) and \( \gamma_j \)

Inputs: \( z^*, k_j, \text{Type}, \beta_j \)

Outputs: \( z^*, e^*, k^* \)

Case 1: Type = Right

Step 1.0 Set \( \epsilon_j = -(\gamma_j + \beta_j)(U_j - t_{k_j}), z_j^U = c_{k_j} + \gamma_j t_{k_j}, z_j^L = z_j^U - \epsilon_j \)

Step 1.1 if \( \alpha_j + z_j^+ < z^* \), enumerate up to \( Y \) simple paths in \( \Omega_j = \{ k \in K | z_j^* \leq c_k + \gamma_j t_k \leq \} \) using a K-shortest path ranking algorithm, go to Step 1.2; otherwise, return.

Step 1.2 set \( z_j^* = \min \{ c_k + \gamma_j t_k, k \in \Omega_j \} \)

Step 1.3 if \( \alpha_j + z_j^+ < z^* \), [set \( z^* = \alpha_j + z_j^+, k^* = k_j^* \). If \( Y > |\Omega_j| \), set \( e^* = 0 \); otherwise, set \( e^* = z^* - (\alpha_j + z_j^+) \)]

Case 2: Type = Left

Step 2.0 set \( \epsilon_j = -(\gamma_j + \beta_j)(t_{k_j} - L_j), z_j^U = c_{k_j} + \gamma_j t_{k_j}, z_j^L = z_j^U - \epsilon_j \)

Step 2.1 if \( \alpha_j + z_j^+ < z^* \), enumerate up to \( Y \) simple paths in \( \Omega_j = \{ k \in K | z_j^* \leq c_k + \gamma_j t_k \leq \} \) using a K-shortest path ranking algorithm, go to Step 2.2; otherwise, return.

Step 2.2 set \( z_j^* = \min \{ c_k + \gamma_j t_k, k \in \Omega_j \} \)

\( k_j^* = \arg \min \{ c_k + \gamma_j t_k, k \in \Omega_j \} \)
Step 2.3 if $\alpha_j + z_j^* < z^*$, set $z^* = \alpha_j + z_j^*, k^* = k_j^*$. If $Y > |\Omega_j|$, set $\epsilon^* = 0$; otherwise, set $\epsilon^* = z^* - (\alpha_j + z_j^*)$.

Case 3: Type = Center

Step 3.0 enumerate the first $Y$ minimum cost paths with $c_k - \beta_j t_k$ being defined as the path cost, using a K-shortest path ranking algorithm.

Step 3.1 Define $Q_j$ as the set containing each path $k$ generated in Step 3.0 such that $L_j \leq t_k \leq U_j$. If $Q_j = \emptyset$, return; else, set $\bar{k}_j = \text{argmin}\{c_k - \beta_j t_k | k \in Q_j\}$, go to Step 3.2.

Step 3.2 If $\gamma_j = -\beta_j$, $\{z_j^* = c_{\bar{k}_j} + \gamma_j t_{\bar{k}_j} \}$.

else set $z_j^L = \min\{c_{\bar{k}_j} + \beta_j (L_j - \bar{t}_k) + \gamma_j \bar{t}_k, C_{\bar{k}_j} + \bar{t}_k U_j, \gamma_j U_j\}$, $z_j^U = c_{\bar{k}_j} + \gamma_j t_{\bar{k}_j}, \epsilon_j = z_j^U - z_j^L$, go to Step 3.3.

Step 3.3 if $\alpha_j + z_j^* < z^*$, enumerate up to $Y$ simple paths in $\Omega_j = \{k \in K | z_j^L \leq c_k + \gamma_j t_k \leq z_j^U, L_j \leq t_k \leq U_j\}$ using a K-shortest path ranking algorithm, go to Step 3.4; otherwise, return;

Step 3.4 set $z_j^* = \min\{c_k + \gamma_j t_k, k \in \Omega_j\}$, $k_j^* = \text{argmin}\{c_k + \gamma_j t_k, k \in \Omega_j\}$.

Step 3.5 if $\alpha_j + z_j^* < z^*$, set $z^* = \alpha_j + z_j^*, k^* = k_j^*$. If $Y > |\Omega_j|$, set $\epsilon^* = 0$; otherwise, set $\epsilon^* = z^* - (\alpha_j + z_j^*)$.

Algorithm 3 sequentially solves each subproblem, while keeping the best current solution (best upper bound) and the corresponding gap. The solution of a subproblem depends on the relative location of $L_j$, $U_j$, $t$ and $\bar{t}$, as well as the slope of the $j$th linear segment of function $H$, i.e. $\gamma_j$. When certain conditions are satisfied (see Cases 1.1.1 and 2.1), the optimal solution can be directly obtained from the efficient path set using Algorithm 1. Otherwise, partial path enumeration may be conducted by calling Algorithm 4. If the feasible range contains at least one efficient path, it can be used to guide the search as an upper bound. If such an upper bound is not readily available from the efficient path set, we will have to perform another partial path enumeration to identify a path, which will then be used as an upper bound for further search. The partial path enumeration algorithm is described below, which requires four inputs: the best upper bound $z^*$, the path that provides the current upper bound ($k_j^*$), the type of path enumeration (explained later), the slope that provides the objective cost function for the K-shortest path algorithm. Besides, the number of enumerated paths for each subproblem is not allowed to exceed a predefined maximum number $Y$ in order to control the computation time.

5.2. Justification

We now show why the above algorithms correctly solve the linearized version of Problem (1). It is clear that if we can solve each subproblem in (16), then solving the linearized (1) is just a matter of comparing the solutions to these subproblems, as shown in (17). We formally state the result as follows.

Proposition 1. Algorithms 3 and 4 solve the the discrete subproblem (16) exactly if Algorithm 4 is allowed to enumerated as many simple paths as needed.

Proof: We will prove that in each of the four cases defined in Algorithm 3, the proposed algorithm will correctly identify the optimal solution to the subproblem (16).

Case 0 is trivial since no feasible path exists when $U_j$ is less than the minimum possible path travel time $t_j$.

For Case 1 (i.e. when $U_j$ lies between $t$ and $\bar{t}$), two possibilities arise that call for different treatments. The first case is when there are efficient paths whose travel times are between $L_j$ and $U_j$, see Figure 2(a), which has three subcases depending on the value of $\gamma_j$. 
1. If \(-\beta_{k_j}^+ \leq \gamma_j \leq -\beta_{k_j}^-\) (Case 1.1.1), the optimal path to the subproblem must be an efficient path between \(L_j\) and \(U_j\) as per the definition of the efficient path set and Algorithm 1. Hence, the optimal path can be identified easily using Algorithm 1. As the solution is optimal, the gap is zero.

2. When \(\gamma_j > -\beta_{k_j}^-\) (Case 1.1.2), we claim that the optimal path to the subproblem must lie in the triangle area \(ABC\). That the path must lie above the convex hull (line \(AB\) in Figure 2(a)) and to the right of \(L_j\) (line \(BC\) in Figure 2(a)) is obvious. If one shifts a line \(z_j = c + \gamma_j t\) upwards (see Figure 2(a)), it will first hit point B (when it enters the feasible region) and finally reach point A when it overlaps with line AC. Moving beyond \(A\) would be meaningless for the sake of cost minimization because the path at point A (\(k_L^J\) in the figure) would surely be a better alternative compared to any path above line AC. In other words, path \(k_L^J\) offers an upper bound to the subproblem in this case. Since our imaginary line first hits point B in its journey, point \(B\) offers a lower bound, which may or may not be attainable. Either way, the length of line \(BC\) offers the maximum gap, i.e. the maximum possible error if we are to use \(k_L^J\) as an approximated solution. In addition to providing a gap, the lower bound can also be used to check if the current subproblem has a chance to improve the overall objective function at all. Specifically, if the subproblem’s lower bound implies an objective function larger than or equal to the best upper bound (\(z^*\)), there is no need to further close the gap for the current subproblem. This proves the correctness of Type Left path enumeration described in Algorithm 4. Finally, to find an exact solution requires enumerating all simple paths in the triangle ABC, which are denoted as \(\Omega_j\) in the algorithm. This can be ensured only if \(Y\) is equal or close to the number of simple paths in the network \(|K|\) in Algorithm 4.

3. The case of \(\gamma_j < -\beta_{k_j}^+\) (Case 1.1.3) is similar to Case 1.1.2 except that the path providing upper bound becomes \(k_U^J\) and that the area to be examined is \(DEF\) (cf. Figure 2(a)). It corresponds to Type Right path enumeration described in Algorithm 4, whose correctness can be similarly proven.

Let us now turn to the other possibility in Case 1 of Algorithm 3, where no efficient path exists with travel time between \(L_j\) and \(U_j\), see Figure 2(b). In this case, either point \(A\) or \(B\) could provide a lower bound depending on the value of \(\gamma_j\). Since we do not have an upper bound to guide path enumeration as before, it is necessary to first find such an upper bound. To do this, a path enumeration can be performed to find the path \(\tilde{k}_j\) that minimizes the cost \(c_k - \hat{\beta}_{k_j}^- t_k\) while
Case 2 can be dealt with similarly as in Case 1 except for two things. First, by definition Case 2 should always have at least one feasible efficient path, i.e. $k_2$, see Figure 3(a). Thus, only Type Left or Type Right path enumeration is needed. Second, the $k_U^i$ is not generally available because $E_{rs}$ is unknown. Instead, we have to always use $k_2$ as a replacement for $k_U^i$. This implies that $\beta_{k_2}^+ = 0$.

Case 3 is similar to Case 1.2 in that no upper bound is available because all efficient paths are on the right side of the axis $t = T_i$, see Figure 3(b). Hence, a Type Center path enumeration, which consists of two path enumerations with the first aiming to find an upper bound, is needed. The only difference is that in the first enumeration, the path cost is always set to $c_k$. Besides, $k_2$ will be treated as $k_U^i$ in this case. □

A few remarks are in order here about path enumeration.

1. in Algorithm 4, path enumeration is performed by generating K-shortest paths. To improve efficiency, two revisions have to be added to a standard K-shortest path procedure. First, it must terminate whenever the cost difference between a generated path and the shortest path exceeds the gap. Second, the cyclic paths have to be excluded. In this study, Eppstein’s algorithm (Eppstein, 1999) is adopted to find K-shortest paths. This algorithm first builds a graph representing all possible deviations from the shortest path tree which is produced by a one-to-all shortest path searching procedure. Once the deviation graph is built, the K-shortest paths can be picked in the order of increasing cost at an attractive time complexity $O(m + nK)$ (where $m$, $n$ and $K$ are respectively the numbers of nodes, links and the required shortest paths). Our implementation is built on the cyclic-free version of Eppstein’s algorithm (Nie & Lee, 2002). For brevity, the description of the K-shortest algorithm is not provided in the current paper. The reader is referred to Nie & Lee (2002) and the references cited therein.

2. The path enumeration procedure implemented in this study will include infeasible paths whose travel times do not range between $L_i$ and $U_i$. These paths will be excluded when $\Omega_j$ is searched for the optimal path. As a possible further improvement, one could revise the enumeration procedure such that these paths are not generated in the first place. Accomplishing such a “smart” enumeration requires deliberated efforts, however, both in terms of algorithm design and implementation.
3. It is worth emphasizing again that the subproblem solution \( z^*_j \) obtained from Algorithm 3 is guaranteed to be optimal only if we can enumerate all paths in \( \Omega_j \). In this case, the gap associated with the best upper bound is zero. In practice, however, it is difficult to generate all paths in \( \Omega_j \). Most likely we would have to limit how many paths could be generated (\( Y \)). If \( \Omega_j \) is only partially known, it is more appropriate to obtain the gap by \( z^*_j - z^*_j \), albeit the estimation often turns out to be quite conservative.

5.3. Some analytical results

The algorithms presented above aim to deal with very general piecewise linear cost functions. As shown, to find exact solutions, expensive path enumerations are often needed. Hence, one has to strike a balance between the solution quality and the computational effort. In most cases, this tradeoff is represented by the choice of parameter \( Y \) in Algorithm 4, which is the number of paths allowed to be generated in each enumeration process. However, for special piecewise linear functions, we can ensure that an optimal path of Problem (1) must be efficient. In these circumstances, path enumeration is unnecessary and should not be conducted. We need the following lemma to present the main result.

**Lemma 1.** For the piecewise linear function \( H \) defined in (4), if \( \gamma_1 \geq \gamma_2 \geq \ldots \geq \gamma_m \), the following inequality always holds for \( t \in [L, U] \)

\[
\alpha_u + \gamma_u t \geq \alpha_v + \gamma_v t
\]

where \( u \neq v \) are indexes of line segments.

**Proof:** Assume that \( u < v \), then

\[
\alpha_v + \gamma_v t = H(U_u) + \gamma_{u+1}(U_{u+1} - L_{u+1}) + \ldots + \gamma_{v-1}(U_{v-1} - L_{v-1}) + \gamma_v(t - L_v)
\]

\[
\leq H(U_u) + \gamma_u(U_{u+1} - L_{u+1}) + \ldots + \gamma_u(U_{v-1} - L_{v-1}) + \gamma_u(t - L_v)
\]

\[
= H(U_u) + \gamma_u(U_{u+1} - U_u + \ldots + U_{v-1} - U_{v-2} + t - U_{v-1})
\]

Since \( H(U_u) + \gamma_u(t - U_u) = \alpha_u + \gamma_u t \), we have

\[
\alpha_u + \gamma_u t \geq \alpha_v + \gamma_v t
\]

Also, we can easily prove in the same manner that the above inequality holds for the case \( u > v \). Hence, the inequality \( \alpha_u + \gamma_u t \geq \alpha_v + \gamma_v t \) always holds for \( t \in [L, U] \) where \( u \neq v \).

**Theorem 1.** Let \( k^* \) be an optimal solution to Problem (1) linearized with a piecewise linear function \( H(\cdot) \). If \( \gamma_1 \geq \gamma_2 \geq \ldots \geq \gamma_m \), then \( k^* \in E_{rs} \).

**Proof:** First note that the optimal path \( k^* \) must be the optimal path to one of the \( m \) subproblems. Let \( u \) be the index of this subproblem, namely

\[
u = \arg\min_{j=1,2,\ldots,m} \alpha_j + z^*_j
\]

We proceed to prove that \( k^* \) also solves the following minimization problem

\[
\min_{k \in K} z_u = c_k + \gamma_u t_k
\]

which is the \( u \)th subproblem without the travel time constraint. According to Definition 1 the optimal path to the above problem (18) must be efficient. Hence, if we can prove that \( k^* \) is the
solution to Problem (18), then \( k^* \) must be efficient. We now prove this by contradiction. Suppose path \( o \), instead of \( k^* \), is the optimal solution to Problem (18), then we have

\[
c_k + \gamma u t_k > c_o + \gamma o t_o. \tag{19}
\]

Further, suppose path \( o \) is a feasible path for the \( v \)th subproblem (i.e. \( L_v \leq t_o \leq U_v \)). Clearly, \( v \neq u \), otherwise Inequality (19) would be invalid. Since path \( k^* \) minimizes Problem (1), it follows that

\[
\alpha_u + c_k + \gamma u t_k \leq \alpha_v + c_o + \gamma o t_o \tag{20}
\]

Combining the equations (19) and (20) yields

\[
\alpha_u + \gamma u t_o - (\alpha_v + \gamma o t_o) \leq c_o + \gamma u t_o - (c_k + \gamma u t_k) < 0 \tag{21}
\]

which is a contradiction as per Lemma 1. Therefore, \( c_k + \gamma u t_k = c_o + \gamma u t_o \) must hold, which in turn implies path \( k^* \) must be an efficient path.

Theorem 1 can be used to bypass path enumeration when \( h \) is concave, as the optimal path can always be found in the efficient path set. If we further restrict \( h \) to be nondecreasing and concave, then the optimal path must belong to \( E^+_{rs} \). This result is formally stated below.

**Corollary 1.** Let \( k^* \) be an optimal solution to Problem (1) linearized with a piecewise linear function \( H(\cdot) \). If \( \gamma_1 \geq \gamma_2 \geq \ldots \geq \gamma_m \geq 0 \), then \( k^* \in E^+_{rs} \).

**Proof:** It follows directly from the proof of Theorem 1. Note that path \( k^* \) is the optimal solution to the problem (18). Since all \( \gamma \) are non-negative, so is \( \gamma_u \). Consequently, \( k^* \) must be an efficient path among the set \( E^+_{rs} \).

We note that the above corollary confirms the result given in Henig (1985) and Mirchandani & Wieck (1993).

6. Numerical experiments

In this section, we present the results of numerical experiments. The algorithm is coded using TNM, a C++ library for network applications (Nie, 2006), and tested on a laptop with Window 7 Home Premium, Intel(R) Core(TM) i7-2630QM CPU@2.00GHz and 8.00 GB memory. Three classes of problems are tested: a small textbook example; ten by ten grid networks, and a large real-world transportation network. For the two latter classes of networks, link properties are randomly generated using Gamma distribution. The mean, variance and minimum value of both travel time and cost are 2.5, 5 and 0.5, respectively.

6.1. A textbook example

The example is taken from Chapter 16 (Ahuja et al., 1993), which represents a network with six nodes and ten arcs, as shown in Figure 4(a). The first and second numbers in parentheses denote the travel time and cost, respectively. The efficient path frontier for a chosen O-D pair (from node 1 to node 6) is shown in Figure 4(b).

Six different two-piece linear \( H \) functions are used to test the algorithm, which include monotone (increasing or decreasing) functions, “V” shape and “Λ” shape functions (see Table 1). Without loss of generality, all the functions are assumed to pass through the origin \((0,0)\), which means the intercept of the first piece on the axis \( c \) is 0. The details of the optimal solutions corresponding to each function are reported in Table 1. The second and third columns in the table specify the boundaries and slopes of each linear segment of \( H \). The fourth column is the best solution given by the algorithm. The fifth column gives the absolute gap of the best solution from the lower bound. If the gap is 0, the optimal solution is actually found. The sixth column reports the number of enumerated simple paths by the algorithm. The seventh column specifies the maximum
number of simple paths allowed in each enumeration. The last column is the optimal solution given by a brute-force path enumeration. As expected, the algorithm identifies the optimal solutions for all the six instances. For the last three instances (i.e. decreasing function, “V” shape and “Λ” shape functions), path enumeration is necessary. In particular, for the “V” shape function (the fifth function), the number of enumerated paths is even higher than the total number of paths in the network. This is because path enumeration is needed in both subproblems.

### Table 1. Numerical results for a small 6-node network

<table>
<thead>
<tr>
<th>Pieces([Lj, Uj])</th>
<th>Slopes(γj)</th>
<th>Best Obj</th>
<th>Gap</th>
<th>Enum. Paths</th>
<th>Y</th>
<th>Optimal Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 [0,12],[12,20]</td>
<td>4;1</td>
<td>55</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>55</td>
</tr>
<tr>
<td>2 [0,12],[12,20]</td>
<td>4;3</td>
<td>55</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>55</td>
</tr>
<tr>
<td>3 [0,12],[12,20]</td>
<td>3;4</td>
<td>45</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>45</td>
</tr>
<tr>
<td>4 [0,12],[12,20]</td>
<td>-4;3</td>
<td>-63</td>
<td>0</td>
<td>9</td>
<td>20</td>
<td>63</td>
</tr>
<tr>
<td>5 [0,12],[12,20]</td>
<td>-4;4</td>
<td>-31</td>
<td>0</td>
<td>12</td>
<td>20</td>
<td>31</td>
</tr>
<tr>
<td>6 [0,12],[12,20]</td>
<td>4;-4</td>
<td>27</td>
<td>0</td>
<td>2</td>
<td>20</td>
<td>27</td>
</tr>
</tbody>
</table>

### 6.2. Grid networks

We now test two larger examples, both are ten by ten grid networks. The first is acyclic while the second contains cycles. The chosen O-D pair for both networks is from the left-bottom node to the right-top node. The acyclic network is included in the test mainly because it allows us to compare the best solution given by the algorithm with the true optimal solution obtained from the brute-force path enumeration.

#### 6.2.1. Acyclic grid network

For the acyclic 10×10 grid network, the efficient path frontier of the chosen O-D pair is shown in Figure 5(a). We first test six two-piece linear functions, corresponding to Case 1.1 and Case 2 in Algorithm 3, and the results are reported in Table 2. For the first three instances, no
enumeration is needed. For the fourth instance with the decreasing piecewise linear function, enumeration is necessary to close the gap. As revealed from the table, the gap becomes smaller as the number of enumerated paths increases. Interestingly, the optimal solution is identified in 4(c) after enumerating 11604 paths, but the reported gap is still not zero. Because the total number of paths in $\Omega_j$ is larger than that of currently enumerated simple paths, the gap has to be set as the difference between the current upper bound and the lower bound. For 4(d), the gap is set to be 0 because all paths in $\Omega_j$ have been enumerated. Similar trends are observed in Instance 5 and 6. These observations indicate that the optimal path can be found well before $\Omega_j$ is fully inspected.

![Efficient path frontier for the acyclic grid network](a)

![Efficient path frontier for the cyclic grid network](b)

Fig. 5. Efficient path frontier for the grid network

Table 3 shows the numerical results of three-piece linear function for the same $10 \times 10$ acyclic grid network. For the first three instances, there are no efficient paths in the second subproblem (cf. Figure 5(a)), corresponding to Case 1.2 in Algorithm 3. According to the algorithm, two path enumerations are needed to find the optimal solution for this subproblem. However, the first instance reports the gap is 0 without any enumeration, because the lower bound of the second subproblem indicates no need to close the gap. The optimal solution of the second instance can also be easily found without much enumeration. The third instance enumerates 9605 paths to find the optimal solution and 51765 paths to guarantee it (i.e. the gap is 0). For the last three instances, the feasible paths of the third subproblem are on the right side of the axis $t = \bar{t}$ (see Figure 5(a)), which corresponds to Case 3 in Algorithm 3. According to Algorithm 3, path enumeration is essential to solve the subproblem in Case 3. However, for both instance 4 and 5, no path enumeration has been reported. This is because the lower bound of that subproblem indicates no need to close the gap. For Instance 6, path enumeration is conducted to close the gap.

The above experiments are designed to cover all possible cases and different types of piecewise linear functions. The results demonstrate that the algorithm can robustly solve the problem in most cases. However, the problem is more difficult to solve if a negative $\gamma_j$ appears on the right side of the axis $t = \bar{t}$ (e.g. instance 4 and 6 in Table 2; instance 3 and 6 in Table 3). In these cases, a considerable amount of path enumeration is needed to achieve small gaps. Fortunately, this type of cost functions rarely arise in the real-world transportation applications.

We proceed to test the use of the piecewise linear function for approximating a general nonlinear function. These experiments solve the problem with the approximated piecewise linear function using our algorithm and then compare the results with the optimal solution associated with the original function obtained from brute-force path enumeration. The following nonlinear
### Table 2. Numerical results of two-piece linear functions for the 10 × 10 acyclic grid network

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 [0,17];[17,45]</td>
<td>4;1</td>
<td>74.6680</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>74.6680</td>
</tr>
<tr>
<td>2 [0,17];[17,45]</td>
<td>4;3</td>
<td>74.6680</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>74.6680</td>
</tr>
<tr>
<td>3 [0,17];[17,45]</td>
<td>3;4</td>
<td>60.7197</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>60.7197</td>
</tr>
<tr>
<td>4(a) [0,17];[17,45]</td>
<td>-4;-3</td>
<td>-115.7760</td>
<td>23.4594</td>
<td>4000</td>
<td>2000</td>
<td>-121.7880</td>
</tr>
<tr>
<td>4(b) [0,17];[17,45]</td>
<td>-4;-3</td>
<td>-118.6350</td>
<td>20.5998</td>
<td>6000</td>
<td>3000</td>
<td>-121.7880</td>
</tr>
<tr>
<td>4(c) [0,17];[17,45]</td>
<td>-4;-3</td>
<td>-121.7880</td>
<td>17.4469</td>
<td>11604</td>
<td>7000</td>
<td>-121.7880</td>
</tr>
<tr>
<td>4(d) [0,17];[17,45]</td>
<td>-4;-3</td>
<td>-121.7880</td>
<td>0</td>
<td>53224</td>
<td>50000</td>
<td>-121.7880</td>
</tr>
<tr>
<td>5(a) [0,17];[17,45]</td>
<td>-4;-4</td>
<td>-52.3360</td>
<td>0.9586</td>
<td>1295</td>
<td>1000</td>
<td>-52.3360</td>
</tr>
<tr>
<td>5(b) [0,17];[17,45]</td>
<td>-4;-4</td>
<td>-52.3360</td>
<td>0</td>
<td>4899</td>
<td>5000</td>
<td>-52.3360</td>
</tr>
<tr>
<td>6(a) [0,17];[17,45]</td>
<td>4;-4</td>
<td>-1.3766</td>
<td>29.8584</td>
<td>2000</td>
<td>1000</td>
<td>-10.1825</td>
</tr>
<tr>
<td>6(b) [0,17];[17,45]</td>
<td>4;-4</td>
<td>-6.2832</td>
<td>24.9519</td>
<td>4000</td>
<td>2000</td>
<td>-10.1825</td>
</tr>
<tr>
<td>6(c) [0,17];[17,45]</td>
<td>4;-4</td>
<td>-10.1825</td>
<td>21.0526</td>
<td>8000</td>
<td>4000</td>
<td>-10.1825</td>
</tr>
<tr>
<td>6(d) [0,17];[17,45]</td>
<td>4;-4</td>
<td>-10.1825</td>
<td>0</td>
<td>48620</td>
<td>50000</td>
<td>-10.1825</td>
</tr>
</tbody>
</table>

### Table 3. Numerical results of three-piece linear functions for the 10 × 10 acyclic grid network

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 [0,15.5];[15.5,18];[18,45]</td>
<td>3;2;1</td>
<td>60.7197</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>60.7197</td>
</tr>
<tr>
<td>2 [0,15.5];[15.5,18];[18,45]</td>
<td>3;2;1</td>
<td>55.0107</td>
<td>0</td>
<td>1000</td>
<td>1000</td>
<td>55.0107</td>
</tr>
<tr>
<td>3(a) [0,15.5];[15.5,18];[18,45]</td>
<td>3;2;-1</td>
<td>37.8553</td>
<td>10.5904</td>
<td>4000</td>
<td>2000</td>
<td>37.4296</td>
</tr>
<tr>
<td>3(b) [0,15.5];[15.5,18];[18,45]</td>
<td>3;2;-1</td>
<td>37.4296</td>
<td>10.1647</td>
<td>9605</td>
<td>6000</td>
<td>37.4296</td>
</tr>
<tr>
<td>3(c) [0,15.5];[15.5,18];[18,45]</td>
<td>3;2;-1</td>
<td>37.4296</td>
<td>0</td>
<td>51765</td>
<td>50000</td>
<td>37.4296</td>
</tr>
<tr>
<td>4 [0,17];[17,30];[30,45]</td>
<td>3;2;1</td>
<td>60.7197</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>60.7197</td>
</tr>
<tr>
<td>5 [0,17];[17,30];[30,45]</td>
<td>3;2;1</td>
<td>60.7197</td>
<td>0</td>
<td>1000</td>
<td>1000</td>
<td>60.7197</td>
</tr>
<tr>
<td>6(a) [0,17];[17,30];[30,45]</td>
<td>3;2;-1</td>
<td>33.3553</td>
<td>10.5904</td>
<td>4000</td>
<td>2000</td>
<td>32.9296</td>
</tr>
<tr>
<td>6(b) [0,17];[17,30];[30,45]</td>
<td>3;2;-1</td>
<td>32.9296</td>
<td>10.1647</td>
<td>12000</td>
<td>6000</td>
<td>32.9296</td>
</tr>
<tr>
<td>6(c) [0,17];[17,30];[30,45]</td>
<td>3;2;-1</td>
<td>32.9296</td>
<td>0</td>
<td>74344</td>
<td>50000</td>
<td>32.9296</td>
</tr>
</tbody>
</table>
functions are tested:

(1) \( a(x - b)^2 + c; \)

(2) \( a e^{bx} + c. \)

where \( x \geq 0. \) The results are reported in Table 4. The second column in the table specifies the nonlinear functions which include increasing, decreasing and non-monotone functions. The third column shows the breaking points of the two-piece linear functions used to approximate the original nonlinear function. The linearized problems are solved to optimality for all the instances in Table 4. The fourth column reports the optimal objective value given by the approximation problem. Note that we first find the optimal path with respect to the linearized objective function via Algorithm 3, and then evaluate the objective value based on the original nonlinear objective function. The seventh column gives the optimal objective value solved by the brute-force path enumeration using the original nonlinear function. If the gap between the two objective function values are zero, it indicates that the approximation method and the brute-force path enumeration method actually found the same optimal path. The results show that the two-piece approximation identifies the correct optimal path in 9 out of 11 instances. Besides, the performance of the algorithm seems satisfactory for these two types of nonlinear functions, since only three instances out of eleven need extensive enumeration efforts.

<table>
<thead>
<tr>
<th>Function</th>
<th>Pieces([( L_j ), ( U_j )])</th>
<th>Approx. Obj.</th>
<th>Enum. Paths</th>
<th>( Y )</th>
<th>Optimal Obj.</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( x^2 )</td>
<td>[0,20], [20,45]</td>
<td>206.8788</td>
<td>0</td>
<td>50000</td>
<td>206.8788</td>
<td>0</td>
</tr>
<tr>
<td>2 ( 0.1x^2 )</td>
<td>[0,20], [20,45]</td>
<td>38.3303</td>
<td>0</td>
<td>50000</td>
<td>38.3303</td>
<td>0</td>
</tr>
<tr>
<td>3 ( 10x^2 )</td>
<td>[0,20], [20,45]</td>
<td>1884.9188</td>
<td>0</td>
<td>50000</td>
<td>1884.9188</td>
<td>0</td>
</tr>
<tr>
<td>4 ( (x - 20)^2 )</td>
<td>[0,20], [20,45]</td>
<td>15.0206</td>
<td>97240</td>
<td>50000</td>
<td>13.2312</td>
<td>1.7894</td>
</tr>
<tr>
<td>5 ( -(x - 20)^2 )</td>
<td>[0,20], [20,45]</td>
<td>-451.0271</td>
<td>48620</td>
<td>50000</td>
<td>-451.8664</td>
<td>0.8393</td>
</tr>
<tr>
<td>6 ( e^{0.1x} )</td>
<td>[0,20], [20,45]</td>
<td>19.3682</td>
<td>0</td>
<td>50000</td>
<td>19.3682</td>
<td>0</td>
</tr>
<tr>
<td>7 ( e^{0.01x} )</td>
<td>[0,20], [20,45]</td>
<td>14.0255</td>
<td>2</td>
<td>50000</td>
<td>14.0255</td>
<td>0</td>
</tr>
<tr>
<td>8 ( -e^{0.1x} )</td>
<td>[0,20], [20,45]</td>
<td>-43.3725</td>
<td>48632</td>
<td>50000</td>
<td>-43.3725</td>
<td>0</td>
</tr>
<tr>
<td>9 ( -e^{0.01x} )</td>
<td>[0,20], [20,45]</td>
<td>11.4818</td>
<td>2</td>
<td>50000</td>
<td>11.4818</td>
<td>0</td>
</tr>
<tr>
<td>10 ( e^{-0.1x} )</td>
<td>[0,20], [20,45]</td>
<td>12.8476</td>
<td>2</td>
<td>50000</td>
<td>12.8476</td>
<td>0</td>
</tr>
<tr>
<td>11 ( e^{-0.01x} )</td>
<td>[0,20], [20,45]</td>
<td>13.5443</td>
<td>4</td>
<td>50000</td>
<td>13.5443</td>
<td>0</td>
</tr>
</tbody>
</table>

6.2.2. Cyclic grid network

We now test the proposed algorithm on a 10 \( \times \) 10 grid network with cycles. In this case, the true optimal solution is unknown because enumerating all simple paths is too expensive. The solution quality has to be evaluated based on the gap. We note that cyclic paths may be generated by our K-shortest path algorithm, although they will be excluded later. The efficient path frontier of this network for the chosen O-D pair is shown in Figure 5(b). The same six functions used for the acyclic network are employed in the cyclic setting. The results are reported in Table 5. Here, \( Y^* \) denotes the maximum number of general paths (i.e. including cyclic paths) allowed in the each enumeration \(^2\). As before, the algorithm found the optimal solutions to the first three instances without path enumeration. For the last three instances, path enumeration is invoked. The gaps for both instances 4 and 6 remain large, even after enumerating 30,000 paths (including

\(^2\)The K-shortest path algorithm used in this study excludes cyclic paths in a postprocess. Consequently, it cannot control how many simple paths (i.e. \( Y \)) it can generate in a given run.
cyclic paths). Yet, the objective value does not improve when the number of enumerated simple paths increase from 5,466 to 7,201 (5,457 to 7,192) in Instance 4 (6). Based on previous observations, this is likely an indication that the solution is close to true optimality. Overall, compared with Table 2, we can see more paths (including cyclic paths) have to be enumerated in the cyclic network to get a stable solution.

### Table 5. Numerical results of two-piece linear functions for the 10 × 10 cyclic grid network

<table>
<thead>
<tr>
<th>Pieces([L_j, U_j])</th>
<th>Slopes(γ_j)</th>
<th>Best Obj.</th>
<th>Gap</th>
<th>Enum. Simple Paths</th>
<th>Y*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 [0,17];[17,45]</td>
<td>4;1</td>
<td>78.8728</td>
<td>0</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>2 [0,17];[17,45]</td>
<td>4;3</td>
<td>78.8728</td>
<td>0</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>3 [0,17];[17,45]</td>
<td>3;4</td>
<td>64.0864</td>
<td>0</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>4(a) [0,17];[17,45]</td>
<td>-4;-3</td>
<td>-107.4153</td>
<td>29.2874</td>
<td>3376</td>
<td>10000</td>
</tr>
<tr>
<td>4(b) [0,17];[17,45]</td>
<td>-4;-3</td>
<td>-114.3709</td>
<td>22.3318</td>
<td>5466</td>
<td>20000</td>
</tr>
<tr>
<td>4(c) [0,17];[17,45]</td>
<td>-4;-3</td>
<td>-114.3709</td>
<td>22.3318</td>
<td>7201</td>
<td>30000</td>
</tr>
<tr>
<td>5(a) [0,17];[17,45]</td>
<td>-4;4</td>
<td>-51.5094</td>
<td>0.543</td>
<td>451</td>
<td>5000</td>
</tr>
<tr>
<td>5(b) [0,17];[17,45]</td>
<td>-4;4</td>
<td>-51.5094</td>
<td>0.543</td>
<td>451</td>
<td>10000</td>
</tr>
<tr>
<td>6(a) [0,17];[17,45]</td>
<td>4;-4</td>
<td>16.628</td>
<td>45.3307</td>
<td>2086</td>
<td>5000</td>
</tr>
<tr>
<td>6(b) [0,17];[17,45]</td>
<td>4;-4</td>
<td>9.2849</td>
<td>37.9875</td>
<td>3367</td>
<td>10000</td>
</tr>
<tr>
<td>6(c) [0,17];[17,45]</td>
<td>4;-4</td>
<td>-0.1066</td>
<td>28.5961</td>
<td>5457</td>
<td>20000</td>
</tr>
<tr>
<td>6(d) [0,17];[17,45]</td>
<td>4;-4</td>
<td>-0.1066</td>
<td>28.5961</td>
<td>7192</td>
<td>30000</td>
</tr>
</tbody>
</table>

#### 6.3. Large scale real network

Finally, a large scale real transportation network, the Chicago Regional network (Bar-Gera et al., 2012), is used to test the computational performance of the proposed algorithm. The network has 12,982 nodes and 39,018 links (see Figure 6(a)). All the link cost and time are randomly generated as described before. The efficient path frontier of this network for a chosen O-D pair is shown in Figure 6(b). The maximum number of enumerated general paths (including cyclic paths) $Y^*$ for each subproblem is limited to 1000. The numerical results are shown in Table 6. For the first three instances, the algorithm performs well as no path enumeration is needed. Also expected are the large gaps for the decreasing functions (Instances 5 and 6) and “Λ” shape function (Instance 8). In these cases, the algorithm cannot take full advantage of the efficient path set. Instead, it has to heavily rely on path enumeration to close the gap. For the “V” shape function, the gap is quite small, although the computational overhead is still high.

#### 7. Conclusions

The bicriterion shortest path problem with a general nonadditive cost has found important applications in transportation models. This paper surveyed several models for which the bicriterion shortest path problem considered herein is a core building block. While variants of this problem have been researched in the literature, the generality of the nonlinear cost function employed in this paper distinguishes it from most existing work. As explained in Section 3, such generality is crucial to two of the four surveyed models. The proposed approach to solving the bicriterion nonadditive shortest path problem involves discretizing the nonlinear function and solving the resulted subproblems sequentially. These subproblems, which are constrained shortest path problems with additive cost, are solved using a specialized algorithm that makes use of the geometric properties of the efficient path set. The justification of the algorithm identifies conditions under which the solution to the subproblem must be an efficient path. Importantly,
Fig. 6. Topology and efficient path frontier for the Chicago Regional network

Table 6. Numerical results of two-piece linear functions for the Chicago regional network

<table>
<thead>
<tr>
<th>Pieces([L_j, U_j])</th>
<th>Slopes(γ_j)</th>
<th>Best Obj.</th>
<th>Gap</th>
<th>Enum. Simple Paths</th>
<th>Y*</th>
<th>CPU Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 [0,60];[60,120]</td>
<td>4;1</td>
<td>273.5131</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>0.0130</td>
</tr>
<tr>
<td>2 [0,60];[60,120]</td>
<td>4;3</td>
<td>273.5131</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>0.0150</td>
</tr>
<tr>
<td>3 [0,60];[60,120]</td>
<td>3;4</td>
<td>219.6575</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>0.0160</td>
</tr>
<tr>
<td>4 [0,60];[60,120]</td>
<td>0;3</td>
<td>53.3869</td>
<td>0.0665</td>
<td>879</td>
<td>1000</td>
<td>32.3070</td>
</tr>
<tr>
<td>5 [0,60];[60,120]</td>
<td>-4;-3</td>
<td>-259.6078</td>
<td>107.6952</td>
<td>1254</td>
<td>1000</td>
<td>49.0000</td>
</tr>
<tr>
<td>6 [0,60];[60,120]</td>
<td>-4;-1</td>
<td>-210.5143</td>
<td>36.7887</td>
<td>1254</td>
<td>1000</td>
<td>48.3290</td>
</tr>
<tr>
<td>7 [0,60];[60,120]</td>
<td>-4;4</td>
<td>-185.7762</td>
<td>0.9034</td>
<td>1341</td>
<td>1000</td>
<td>34.0860</td>
</tr>
<tr>
<td>8 [0,60];[60,120]</td>
<td>4;-4</td>
<td>195.8455</td>
<td>143.1485</td>
<td>790</td>
<td>1000</td>
<td>33.1340</td>
</tr>
</tbody>
</table>
we prove that the optimal path must be efficient when the piecewise linear function is concave regardless of monotonicity.

The proposed algorithm is tested using different types of nonlinear functions on various networks. The main findings can be summarized as follows:

- The performance of the algorithm is satisfactory for increasing and “V” shape functions, which represent an important class of cost functions found in the real applications. Indeed, of the two applications surveyed in Section 3 that require non-monotone functions, both involve “V” shape functions.
- For the decreasing and “A” shape functions, extensive path enumeration is often needed to achieve a stable solution.
- Piecewise linear functions with two or three segments seem to provide good approximation to the nonlinear cost functions tested in our experiments (quadratic and exponential).

In the current implementation, the path enumeration procedure is independent of the main algorithm. Such a structure provides flexibility to those who would like to use any available K-shortest path code. Yet, greater efficiency is promised if the algorithm can be better integrated with the path enumeration procedure. Specifically, many conditions that are applied posteriori now (such as those defined by the triangle areas in Section 5) can be applied during the enumeration process to filter out infeasible paths. As it promises to accelerate path enumeration substantially, this strategy is worth of further investigation. Tailoring the model and algorithm proposed in this paper to the transportation applications discussed in Section 3 is another possible direction for future research.

References


Hjorth, K., & Fosgerau, M. 2012. Using prospect theory to investigate the low marginal value of travel time for small time changes. *Transportation Research Part B, 46*(8), 917–932.


