A Polynomial-Time-Delay and Polynomial-Space Algorithm for Enumeration Problems in Multi-Criteria Optimization

Yoshio Okamoto[1] and Takeaki Uno[2]

[1] Department of Information and Computer Sciences, Toyohashi University of Technlology, Hibarigaoka 1–1, Tempaku, Toyohashi, Aichi, 441-8580, Japan. E-mail: okamotoy@ics.tut.ac.jp

[2] National Institute of Informatics, Hitotsubashi 2–1–2, Chiyoda-ku, Tokyo, 101-8430, Japan. E-mail: uno@nii.jp

Abstract. We propose a polynomial-time-delay polynomial-space algorithm to enumerate all efficient extreme solutions of a multi-criteria minimum-cost spanning tree problem, while only the bi-criteria case was studied in the literature. The algorithm is based on the reverse search framework due to Avis & Fukuda. We also show that the same technique can be applied to the multi-criteria version of the minimum-cost basis problem in a (possibly degenerated) submodular system. As an ultimate generalization, we propose an algorithm to enumerate all efficient extreme solutions of a multi-criteria linear program. When the given linear program has no degeneracy, the algorithm runs in polynomial-time delay and polynomial space. To best of our knowledge, they are the first polynomial-time delay and polynomial-space algorithms for the problems.

1 Introduction

The multi-criteria optimization is a vast field in optimization theory, operations research, and decision science. In a multi-criteria optimization problem, we usually need to enumerate the solutions which have a certain specified property, for example, the Pareto optimality or the efficiency.[3] See Ehrgott [3] for detail.

There have been two main streams in algorithm design for the multi-criteria optimization: exact approach and approximate approach. In the exact approach, the enumeration has to be exact, namely, all the solutions have to be output (without any duplication). For example, in the multi-criteria linear programming many exact algorithms have been proposed which enumerate all efficient extreme solutions or enumerate all efficient faces (see Ehrgott [3] and references therein). For bi-criteria combinatorial optimization problems, Ulungu & Teghem [8] proposed the so-called two-phase method which first determines the

[3] The word "efficient" is used differently in multi-criteria optimization and in algorithm theory. In multi-criteria optimization (or economics) efficiency is just another name for Pareto optimality. We hope that the reader is never confused.

extreme efficient solutions then enumerate the rest of the efficient solutions. On the other hand, in the approximate approach the enumeration is partial. See Zitzler, Laumanns & Bleuler [11] for example. A bit different approximate approach was done by Papadimitriou & Yannakakis [7], which has a certain approximation guarantee. See a recent short survey by Zaroliagis [10].

This work concentrates on the exact approach and we will exploit techniques from enumeration algorithmics. Despite many algorithm have been reported for multi-criteria optimization from the exact approach viewpoint, few of them have a certain theoretical guarantee of complexity. Observe that enumeration of the Pareto-optimal extreme solutions of a single-criteria linear program is equivalent to enumeration of the vertices of a convex polyhedron, and a recent result by Khachiyan, Boros, Borys, Elbassioni & Gurvich [5] shows that this problem admits no polynomial total time algorithm unless P = NP. This looks one of the obstructions for a theoretical investigation. So, we concentrate on a simpler problem to reveal the difficulty for the development of an algorithmic theory of multi-criteria enumeration problems.

As a sample problem, we study the multi-criteria minimum-cost spanning tree problem: given a connected undirected graph and several edge-cost functions, we have to find all spanning trees which minimize some convex combinations of the cost functions. In the multi-criteria optimization terminology, the outputs are exactly the solutions for all possible weighted sum scalarizations, and they correspond to the extreme efficient solutions. The determination of the extreme efficient solutions is a first step for complete enumeration of the efficient solutions, for example in the two-phase method [8].

We will compare two main methods in enumeration algorithmics. One is the binary partition method, and the other is the reverse search method. In the binary partition method, we recursively divide the solution space until we get trivial instances. In the reverse search method proposed by Avis & Fukuda [1], we implicitly define a rooted tree on the solutions to be enumerated, and traverse it.

We try to apply the two enumeration methods above to the multi-criteria minimum-cost spanning tree problem. For the binary partition method, we prove that a subproblem arising from a natural binary partition approach is NP-complete. This implies that an approach by the binary partition method seems difficult. On the other hand, with the reverse search method we design an algorithm which runs with polynomial-time delay and in polynomial space. This is the first algorithm for this problem with such a complexity guarantee.

Our reverse-search algorithm can be extended to the multi-criteria version of the minimum-cost base problem in matroids and submodular systems. Furthermore, a similar algorithm turns out to work for the multi-criteria linear programming. Although there have been many algorithms proposed for the multi-criteria linear programming, none of them has a performance guarantee as running with polynomial-time delay and in polynomial space (see Ehrgott [3] and references therein). Indeed, these algorithms store all the outputs as a list in the working memory to get rid of duplication, which looks a bottleneck for the efficiency. We may accomplish the polynomial-time delay by a small modification (for example, using a balanced binary search tree instead of a list). However, it appears difficult for these algorithms to achieve the polynomial space by a small modification; namely, the essential improvement for memory usage is by far hard. On the other hand, our reverse-search algorithm can achieve both of the goals. This exhibits the power of the reverse search, and we hope that this work initiates a more fruitful connection of the multi-criteria optimization with the algorithms community.

The paper is organized as follows. In the next section, we give an introduction to enumeration algorithmics terminology and a concise description of the multi-criteria minimum-cost spanning tree problem. Section 3 discusses some existing methods to enumerate the spanning trees and observe how natural extensions of these methods fail. This includes the NP-completeness result of a natural subproblem arising from a binary partition method. Then in Section 4, we consider how we can overcome this issue, and design an algorithm running in polynomial-time delay and polynomial space with the reverse search method. Section 5 discusses a possible generalization of our reverse search algorithm to the multi-criteria linear programming. The final section concludes the paper with some open questions.

2 Preliminaries

An *enumeration problem* asks to output all objects, called *solutions*, which satisfy a given condition. To measure the efficiency of enumeration algorithms, we have to take into account the size of output (i.e., the number of solutions) explicitly since it could be exponentially large in terms of the size of input. An enumeration algorithm runs in *polynomial-time delay* if for any output object the next output object can be obtained in polynomial time in the size of input; it runs in *polynomial-space* if the working space it uses is bounded by a polynomial of the size of input. Note that we only count the working space, excluding the space for outputs. Intuitively speaking, the working space is a read/write memory and the output space is a write-only disk.

A *convex combination* of k functions $c_1, c_2, \ldots, c_k$ is a function $\sum_{i=1}^{k} \lambda_i c_i$ for some non-negative real numbers $\lambda_1, \lambda_2, \ldots, \lambda_k$ summing up to one. We call the vector $(\lambda_1, \ldots, \lambda_k)^\top \in \mathbb{R}^k$ of coefficients the *barycentric coordinate* of the combination.

Given a connected undirected graph $G = (V, E)$, a *spanning tree* of G is an edge subset $T \subseteq E$ of size $|V| - 1$ which embraces no cycle. For a non-negative edge-cost function $c\colon E \to \mathbb{R}_+$, a *minimum-cost spanning tree* of G with respect to c is a spanning tree T of G which minimizes the total cost $c(T) = \sum_{e \in T} c(e)$. We study the following problem.

Problem: MC-MCST
Input: a connected undirected graph $G = (V, E)$ and k distinct non-negative edge-cost functions $c_1, \ldots, c_k\colon E \to \mathbb{R}_+$ **Enumerate**: the spanning trees of G each of which is minimum-cost with respect to some convex combination of $c_1, \ldots, c_k$.

We call a spanning tree of G *feasible* if it is minimum-cost with respect to some convex combination of $c_1, \dots, c_k$ (i.e., if it is to be output in MC-MCST).

3 Failed Attempts for Generalization by Straightforward Approaches

In this section, we first describe two existing methods for enumeration of spanning trees in a given connected graph, and observe why the straightforward generalizations of them to MC-MCST do not give efficient algorithms.

3.1 Binary Partition Method

Let us first look at a simple binary partition approach to enumerate all spanning trees in a given connected undirected graph $G = (V, E)$. First of all, we choose an arbitrary edge $e_1 \in E$ and classify the spanning trees of G into two groups: those containing e_1 and those not containing e_1. Then, we choose another arbitrary edge $e_2 \in E \setminus \{e_1\}$, and divide the groups similarly. This will give a recursion tree, and we stop the recursive call when the obtained group is ensured to contain no spanning tree. In this way, we can reduce redundant computation. The problem to decide whether a group contains a spanning tree can be formulated as "for disjoint subsets $E_1, E_2 \subseteq E$, does there exist a spanning tree of G which contains the edges in E_1 but does not contain any edges in E_2?" This can be solved in linear time.

To solve MC-MCST in the same way, we have to solve the following problem.

Problem: BinaryPartition

Input: a connected undirected graph $G = (V, E)$, two disjoint subsets $E_1, E_2 \subseteq E$ and k distinct non-negative edge-cost functions $c_1, \dots, c_k \colon E \to \mathbb{R}_+$

Question: Does there exist a spanning trees of G which contains the edges in E_1 but does not contain any edges in E_2 and is minimum-cost with respect to some convex combination of $c_1, \dots, c_k$.

If the problem BinaryPartition can be solved in polynomial time, then we can use the same binary partition strategy as above to obtain an algorithm to solve MC-MCST in polynomial-time delay and polynomial space. However, the following theorem shows that it is quite unlikely for us to achieve this goal.

Theorem 1. *The problem BinaryPartition is NP-complete.*

Proof. We can easily see the membership of the problem in NP. We show NP-hardness. To this end, we reduce the satisfiability problem (SAT) to BinaryPartition. An instance of SAT is given as a set of boolean variables $x_1, \dots, x_n$ and a set of clauses $C_1, \dots, C_m$ each of which consists of (possibly several) literals. Each literal is either a variable or its negation.

From the given instance of SAT, we construct a connected graph $G = (V, E)$. For each variable x_i we set up three vertices v_i^r, v_i^t, v_i^f. For each clause C_j we set

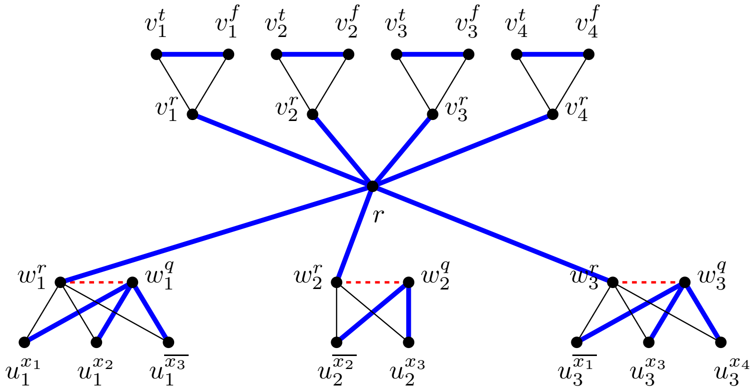

Fig. 1. Reduction in the proof of Theorem 1. This is an example for the formula $C_1 \wedge C_2 \wedge C_3$, where $C_1 = x_1 \vee x_2 \vee \overline{x_3}$, $C_2 = \overline{x_2} \vee x_3$. $C_3 = \overline{x_1} \vee x_3 \vee x_4$. A black thin edge belongs to $E \setminus (E_1 \cup E_2)$; a blue thick edge belongs to E_1; a red broken edge belongs to E_2.

Table 1. Summary of the costs.

	$\{v_i^r, v_i^t\}$	$\{v_i^r, v_i^f\}$	$\{v_i^t, v_i^f\}$	$\{v_i^r, r\}$	$\{v_{i'}^r, v_{i'}^t\}$	$\{v_{i'}^r, v_{i'}^f\}$	$\{v_{i'}^t, v_{i'}^f\}$	$\{v_{i'}^r, r\}$
c_i	0	1	$1/n$	1	0	0	$1/n$	1
$c_{\overline{i}}$	1	0	$1/n$	1	0	0	$1/n$	1

	$\{w_j^r, w_j^q\}$	$\{w_j^r, r\}$	$\{w_j^q, u_j^\ell\}$	$\{w_j^r, u_j^\ell\}$
c_i	$2 - 1/(2n)$	1	1	1 if $\ell = x_i$, 2 otherwise
$c_{\overline{i}}$	$2 - 1/(2n)$	1	1	1 if $\ell = \overline{x_i}$, 2 otherwise

up two vertices w_j^r, w_j^q, and for each literal ℓ of C_j we set up one vertex u_j^ℓ. We also use an extra vertex r. They are the vertices of G.

Next, we draw the edges of G. For each variable x_i, we draw edges $\{v_i^t, v_i^f\} \in E_1$, $\{v_i^r, v_i^t\} \in E \setminus (E_1 \cup E_2)$, $\{v_i^r, v_i^f\} \in E \setminus (E_1 \cup E_2)$, and $\{v_i^r, r\} \in E_1$. For each clause C_j we draw an edge $\{w_j^r, w_j^q\} \in E_2$, $\{w_j^r, r\} \in E_1$, and for each literal ℓ of C_j we draw edges $\{w_j^r, u_j^\ell\} \in E \setminus (E_1 \cup E_2)$, $\{w_j^q, u_j^\ell\} \in E_1$. This completes the description of G. Fig. 1 shows an example.

Now, we set up $2n$ cost functions, each of which is identified with a variable or its negation (i.e., a literal). Namely, for each positive literal x_i, we define the cost function c_i and, Similarly, for a negative literal $\overline{x_i}$, we define the cost function $c_{\overline{i}}$. The definition is as follows. They are summarized in Table 1: $c_i(\{v_i^r, v_i^t\}) = 0$, $c_i(\{v_i^r, v_i^f\}) = 1$, $c_i(\{v_i^t, v_i^f\}) = 1/n$, $c_i(\{v_i^r, r\}) = 1$; for every $i' \in \{1, \ldots, n\} \setminus \{i\}$, $c_i(\{v_{i'}^r, v_{i'}^t\}) = 0$, $c_i(\{v_{i'}^r, v_{i'}^f\}) = 0$, $c_i(\{v_{i'}^t, v_{i'}^f\}) = 1/n$, $c_i(\{v_{i'}^r, r\}) = 1$; for every $j \in \{1, \ldots, m\}$ and every literal ℓ of the clause C_j, $c_i(\{w_j^r, w_j^q\}) = 2 - 1/(2n)$, $c_i(\{w_j^r, r\}) = 1$, $c_i(\{w_j^q, u_j^\ell\}) = 1$, and

$$c_i(\{w_j^r, u_j^\ell\}) = \begin{cases} 1 & \text{if } \ell = x_i, \\ 2 & \text{otherwise.} \end{cases}$$

Similarly, $c_{\bar{i}}(\{v_i^r, v_i^t\}) = 1$, $c_{\bar{i}}(\{v_i^r, v_i^f\}) = 0$, $c_{\bar{i}}(\{v_i^t, v_i^f\}) = 1/n$, $c_{\bar{i}}(\{v_i^r, r\}) = 1$; for every $i' \in \{1, \ldots, n\} \setminus \{i\}$, $c_{\bar{i}}(\{v_{i'}^r, v_{i'}^t\}) = 0$, $c_{\bar{i}}(\{v_{i'}^r, v_{i'}^f\}) = 0$, $c_{\bar{i}}(\{v_{i'}^t, v_{i'}^f\}) = 1/n$, $c_{\bar{i}}(\{v_{i'}^r, r\}) = 1$; for every $j \in \{1, \ldots, m\}$ and every literal ℓ of the clause C_j, $c_{\bar{i}}(\{w_j^r, w_j^q\}) = 2 - 1/(2n)$, $c_{\bar{i}}(\{w_j^r, r\}) = 1$, $c_{\bar{i}}(\{w_j^q, u_j^\ell\}) = 1$, and

$$c_{\bar{i}}(\{w_j^r, u_j^\ell\}) = \begin{cases} 1 & \text{if } \ell = \overline{x_i}, \\ 2 & \text{otherwise.} \end{cases}$$

Thus, we complete the construction of an instance of MC-MCST.

We may prove that there exists a spanning tree of G which is minimum-cost with respect to some convex combination of the c_i and the $c_{\bar{i}}$, $i \in \{1, \ldots, n\}$ if and only if the given SAT instance is satisfiable. We omit the detail here due to the page limitation in this proceedings version. □

Hence, we give up adapting the binary partition method, and try another method.

3.2 Reverse Search Method

The reverse search method, proposed by Avis & Fukuda [1], is one of the most powerful techniques in enumeration algorithmics. Let $G = (V, E)$ be a given undirected connected graph, and we want to enumerate the spanning trees in G. To do this, we set up a rooted tree $\mathcal{R}$ on the spanning trees of G, namely, each node of $\mathcal{R}$ is a spanning tree of G. The enumeration will be done by traversing $\mathcal{R}$ in a depth-first-search manner, but we do not store the entire rooted tree itself; we just specify a parent-child relation which implicitly defines $\mathcal{R}$. In enumeration, we recursively move to children by the depth first search. Therefore, to design an efficient reverse-search algorithm it is enough for us to provide a parent-child relation so that we can find a parent/child efficiently. Since we do not need to store the entire family of spanning trees, but only a spanning tree under current investigation, this enables us to obtain an algorithm which runs in polynomial time delay and polynomial space. See Avis & Fukuda [1] and Nakano & Uno [6] for detail.

First of all, we define an adjacency relation on the family of spanning trees of G. Two distinct spanning trees T and T' of G are *adjacent* if the symmetric difference of T and T' is of size two. Through this adjacency relation, we naturally define the undirected graph $\mathcal{G}(G)$ which has the spanning trees of G as the node set. We can easily see that the number of nodes adjacent to one node is $\mathrm{O}(|V||E|)$, and it is well-known [9, Exercise 2.1.62] that $\mathcal{G}(G)$ is connected.

On $\mathcal{G}(G)$ we define a rooted tree $\mathcal{R}$. For this purpose, we assume that the edges of G are labeled according to some fixed total order $\prec$ as $e_1 \prec e_2 \prec \cdots \prec e_m$. Then, the root of $\mathcal{R}$ is defined as a (unique) lexicographically maximum spanning tree with respect to $\prec$, and a parent of a spanning tree T of G in the rooted tree $\mathcal{R}$ is a (unique) lexicographically maximum neighbor of T in $\mathcal{G}(G)$. This parent-child relation gives a well-defined rooted tree, and we can find a root, a parent of a non-root spanning tree, and the children of a non-leaf spanning tree in polynomial time. Therefore, this leads to an algorithm running in polynomial-time delay and polynomial space for enumerating the spanning trees in an undirected connected graph.

Let us try to generalize this approach to MC-MCST. We are given an undirected connected graph $G = (V, E)$ and k edge-cost functions $c_1, \ldots, c_k$. In this case, we consider the subgraph of $\mathcal{G}(G)$ induced by the feasible spanning trees (i.e., to be enumerated in MC-MCST). Denote this induced subgraph by $\mathcal{G}_M(G)$. Although $\mathcal{G}_M(G)$ depends on the edge-cost functions, we think them fixed thus do not include in the notation for convenience. Ehrgott [2] showed that the graph $\mathcal{G}_M(G)$ is always connected. Therefore, we can define a rooted tree $\mathcal{R}$ on $\mathcal{G}_M(G)$. The most natural way is to use the same strategy as in enumeration of the spanning trees of a connected graph. Namely, we assume that the edges of G are labeled according to some fixed total order $\prec$ as $e_1 \prec e_2 \prec \cdots \prec e_m$. Then, the root of $\mathcal{R}$ is defined as a (unique) lexicographically maximum spanning tree with respect to $\prec$, and a parent of a spanning tree T of G in the rooted tree $\mathcal{R}$ is a (unique) lexicographically maximum neighbor of T in $\mathcal{G}_M(G)$.

However, as opposed to the spanning trees enumeration case, for MC-MCST we have (at least) two troubles here. The first problem is that we do not know how to find a root in polynomial time. Actually, a greedy method fails since there can be many $\prec$-maximal feasible spanning trees in $\mathcal{G}_M(G)$. The second problem is even worse: the graph $\mathcal{R}$ may not be connected. Therefore, the rooted tree is not well-defined in general.

Hence, we need to devise another way to specify a rooted tree on $\mathcal{G}_M(G)$ if we wish to solve MC-MCST via reverse search.

4 The Proposed Algorithm

In our reverse-search algorithm for MC-MCST, we use $\mathcal{G}_M(G)$ defined in the previous section. Then, we have to define a promised rooted tree $\mathcal{R}$. For this purpose, we associate the following type of sequence to each feasible spanning tree. We assume that the edges of G are labeled according to some fixed total order $\prec$ as $e_1 \prec e_2 \prec \cdots \prec e_m$. This order $\prec$ will be used to break a possible tie. For a feasible spanning tree T of G, let $\boldsymbol{\lambda}_T \in \mathbb{R}^k$ be a lexicographically maximum barycentric coordinate of a convex combination of $c_1, \ldots, c_k$ which T minimizes. The following lemma shows that $\boldsymbol{\lambda}_T$ can be computed in polynomial time.

Lemma 2. *For every feasible spanning tree T of G, the vector $\boldsymbol{\lambda}_T$ can be found in polynomial time.*

Proof. We can phrase the problem in the following form.

$$\begin{array}{lll}
\text{lex-max.} & \boldsymbol{\lambda} & \\
\text{subj. to} & \displaystyle\sum_{e \in T} \sum_{i=1}^{k} \lambda_i c_i(e) \le \sum_{e \in T'} \sum_{i=1}^{k} \lambda_i c_i(e) & \text{for all feasible } T' \text{ adjacent to } T, \\
& \displaystyle\sum_{i=1}^{k} \lambda_i = 1, & \\
& \lambda_i \ge 0 & \text{for all } i \in \{1, \dots, k\}.
\end{array}$$

Note that in the first constraint we do not need to take into account all of the spanning trees of G, but we only need the spanning trees adjacent to T. This is due to the convexity (or matroid property) of the minimum-cost spanning tree problem (we omit the detail). Since the number of spanning trees adjacent to T is $\mathrm{O}(|V||E|)$, the number of constraints is polynomial. This lexicographic maximization problem can be solved by maximizing λ_i one by one in increasing order of $i \in \{1, \dots, k\}$, and each maximization is reduced to a linear program. Thus, using any polynomial-time algorithm for linear programming, we can solve the problem in polynomial time. □

The root of $\mathcal{R}$ is chosen as a feasible spanning tree R of G which has a lexicographically maximum $\boldsymbol{\lambda}_T$ among all feasible spanning tree T. Namely, such a barycentric coordinate $\boldsymbol{\lambda}_R$ should satisfy $(\lambda_R)_1 = 1$ and $(\lambda_R)_i = 0$ for all $i \in \{2, \dots, k\}$. Thus, R is a minimum-cost spanning tree with respect to c_1. If there are several minimum-cost spanning trees with respect to c_1, then we choose a $\prec$-maximum one as a root. Such a tree R is unique, and can be found in polynomial time by any polynomial-time minimum-cost spanning tree algorithm.

To specify the parent of a non-root feasible spanning tree T of G, we distinguish two cases. In the first case, we assume that $\boldsymbol{\lambda}_T = (1, 0, 0, \dots, 0)^\top$. Then, T and R both minimize c_1. Therefore, as the following lemma certifies, we can obtain another minimum-spanning tree with respect to c_1 from T by deleting one edge from T and adding one edge from R.

Lemma 3. *Let $G = (V, E)$ be a connected undirected graph, $c \colon E \to \mathbb{R}_+$ be a non-negative edge-cost function, and $T_1, T_2 \subseteq E$ be minimum-cost spanning trees of G with respect to c. Then, there exist two edges $e_1 \in T_1 \setminus T_2$ and $e_2 \in T_2 \setminus T_1$ such that $(T_2 \cup \{e_1\}) \setminus \{e_2\}$ is also a minimum-cost spanning tree of G with respect to c.*

Although this is a well-known fact as, for example, in [9, Exercise 2.3.13], we give a proof here since we actually use the argument in the constructive proof below for the construction of our rooted tree.

Proof. Let us choose a minimum-cost edge $e_1 \in T_1 \setminus T_2$, namely $c(e_1) \le c(e)$ for every $e \in T_1 \setminus T_2$. Then, we can see that $T_2 \cup \{e_1\}$ embraces a unique cycle, say C. Note that C contains e_1. Now, we choose a maximum-cost edge

$e_2 \in C \setminus \{e_1\} \subseteq T \setminus \{e_1\}$, namely, $c(e_2) \geq c(e)$ for every edge $e \in C \setminus \{e_1\}$. Then, $T = (T_2 \cup \{e_1\}) \setminus \{e_2\}$ is a spanning tree of G.

Now we look at the cost. If $c(e_1) < c(e_2)$, then it follows that $c(T) = c(T_2) + c(e_1) - c(e_2) < c(T_2)$. Hence it contradicts the minimality of T_2. On the other hand, suppose that $c(e_1) > c(e_2)$. Then by the choice of e_1 it follows that $c(e) > c(e_2)$ for all $e \in T_1 \setminus T_2$. We consider a (unique) cycle C' in $T_1 \cup \{e_2\}$ and pick an arbitrary edge from $e' \in C' \setminus \{e_2\}$. Then, $T' = (T_1 \cup \{e_2\}) \setminus \{e'\}$ is a spanning tree of G and the cost is $c(T') = c(T_1) + c(e_2) - c(e') < c(T_1)$. Hence it contradicts the minimality of T_1. Thus, it must hold that $c(e_1) = c(e_2)$ and hence T is also a minimum-cost spanning tree of G with respect to c. □

The parent of T is constructively defined as follows. First we choose a minimum-cost edge $e_R \in R \setminus T$ (with respect to c_1), and if there are several choices, we choose a $\prec$-maximum one. This makes the choice of e_R unique. Then, $T \cup \{e_R\}$ contains a cycle C and we choose a maximum-cost edge $e_T \in C \setminus \{e_R\}$ (with respect to c_1), and if there are several choices, we choose a $\prec$-minimum one. From these choices, define the parent of T as $T' = (T \cup \{e_R\}) \setminus \{e_T\}$. From the discussion above, we can see that T' is a feasible spanning tree and $|R \triangle T'| < |R \triangle T|$. Note that T' can be found in polynomial time from T.

In the next case, we assume that $\boldsymbol{\lambda}_T \neq (1, 0, 0, \ldots, 0)^\top$. Then, we consider the corresponding $\boldsymbol{\lambda}_T$. Let $j \in \{2, \ldots, k\}$ be the minimum index such that $(\lambda_T)_j \neq 0$. Then, we take $\boldsymbol{\mu} \in \mathbb{R}^k$ obtained from $\boldsymbol{\lambda}_T$ by increasing the first component by a sufficiently small $\varepsilon > 0$ and decreasing the j-th component by ε. By our assumption for the second case, we can see that such an ε exists which keeps $\boldsymbol{\mu}_T$ to be a barycentric coordinate. Let S be a minimum-cost spanning tree of G with respect to $\sum_{i=1}^{k} \mu_i c_i$. If there are several minimum-cost spanning trees, then we choose a $\prec$-maximum one. By the lexicographic maximality of $\boldsymbol{\lambda}_T$ and the fact that $\boldsymbol{\mu}$ is lexicographically larger than $\boldsymbol{\lambda}_T$, we see that S is different from T. Since ε is sufficiently small, S is also a minimum-cost spanning tree with respect to $c = \sum_{i=1}^{k} (\lambda_T)_i c_i$. Hence, by Lemma 3 similarly to the first case, we choose an edge $e_S \in S \setminus T$ such that $c(e_S) \leq c(e)$ for all $e \in S \setminus T$ (if there are more than one such edges, then we choose the $\prec$-maximal one), and for a (unique) cycle C of $T \cup \{e_S\}$ we choose an edge $e_T \in C \setminus \{e_S\}$ such that $c(e_T) \leq c(e)$ for all $e \in C \setminus \{e_S\}$ (if there are more than one such edges, then we choose the $\prec$-minimal one). Then, we can see (from the proof of Lemma 3) that $T' = (T \cup \{e_S\}) \setminus \{e_T\}$ is a minimum-cost spanning tree with respect to c, and $|S \triangle T'| < |S \triangle T|$ holds. We define the parent of T as T', and we can find T' in polynomial time from T. In this way, the definition of a parent is completed. By the construction, the parent of T is adjacent to T in $\mathcal{G}_M(G)$, and it is unique. Furthermore, the next lemma is important.

Lemma 4. *Let $G = (V, E)$ be a connected undirected graph and $c_1, \ldots, c_k \colon E \to \mathbb{R}_+$ be non-negative edge-cost functions. Then, the parent-child relation defined above is well-defined. Namely, from a non-root feasible spanning tree $T \subseteq E$, by moving to the parent step by step we can arrive at the root R.*

Proof. Let T be a non-root feasible spanning tree and T' its parent. The investigation is divided into two parts according to the case distinctions above. Let us first consider when the first case is applied. In this case it holds that $\boldsymbol{\lambda}_{T'} = \boldsymbol{\lambda}_T = (1,0,0,\ldots,0)^\top$ and $|R \triangle T'| < |R \triangle T|$. Therefore, we can arrive R at some point.

Next let us consider when the second case is applied. Let $T = T_0, T' = T_1$, and in general denote the parent of T_j by T_{j+1}. This construction can continue unless $\boldsymbol{\lambda}_{T_j} = (1,0,0,\ldots,0)^\top$. Hence, it suffices to show that for every j there exists some $j' > j$ such that $\boldsymbol{\lambda}_{T_{j'}}$ is lexicographically larger than $\boldsymbol{\lambda}_{T_j}$. If this is true, then at some point (when the index is j, say) it must hold that $\boldsymbol{\lambda}_{T_j} = (1,0,0,\ldots,0)^\top$ and the case is reduced to the first one.

Fix an arbitrary j. We are done if $\boldsymbol{\lambda}_{T_{j+1}}$ is lexicographically larger than $\boldsymbol{\lambda}_{T_j}$. Therefore, we assume $\boldsymbol{\lambda}_{T_{j+1}} = \boldsymbol{\lambda}_{T_j}$. Let S_j be a spanning tree used to obtain T_j as S was used to obtain T in the text above. Since S_j and S_{j+1} are dependent only on $\boldsymbol{\lambda}_{T_j}$ and $\boldsymbol{\lambda}_{T_{j+1}}$ respectively, it holds that $S_j = S_{j+1}$. However, for any i it holds that $|S_i \triangle T_{i+1}| < |S_i \triangle T_i|$. Therefore, there cannot be an infinitely long sequence $S_i = S_{i+1} = S_{i+2} = \cdots$ of identical spanning trees. Thus, there must exist some $j' > j$ such that $\boldsymbol{\lambda}_{T_{j'}}$ is lexicographically larger than $\boldsymbol{\lambda}_{T_j}$. □

From the discussion above, we finally obtain the following theorem.

Theorem 5. *By the reverse search algorithm described above, we can solve MC-MCST in polynomial-time delay and polynomial space.*

5 Generalization

The reverse search algorithm in the previous section can be generalized in several ways. A close inspection of the discussion shows that we only used the matroid property of the minimum-cost spanning tree problem in the algorithm. Therefore, we can conclude that the multi-criteria minimum-cost base problem in matroids can be solved in polynomial-time delay and polynomial space, when a matroid is given as the independent set oracle. More generally, we can solve the multi-criteria minimum-cost base problem in submodular systems in polynomial-time delay and polynomial space when a submodular function is given as a value-giving oracle. To this end, we need to identify the adjacent bases of a given base in a submodular system. This task is an instance of the submodular function minimization problem, which can be solved in polynomial time [4].

As an extreme generalization, we can consider the multi-criteria linear programming. In a *linear program*, we are given a system of inequalities $A\boldsymbol{x} \geq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$ where $A \in \mathbb{R}^{m \times n}$ is a matrix, and $\boldsymbol{b} \in \mathbb{R}^m$ is a vector. Then we want to find, for a given $\boldsymbol{c} \in \mathbb{R}^n$, a solution $\boldsymbol{x}$ to the inequality system which minimizes $\boldsymbol{c}^\top \boldsymbol{x}$.

The inequality system above defines a convex polyhedron, called the *feasible region* of the problem. Here we assume (without loss of generality) that it is bounded and non-empty. With this assumption, a feasible region has at least one extreme point, and furthermore there exists an optimal solution which is an extreme point of the polyhedron. We call such a solution an *extreme* optimal solution. In a *multi-criteria linear program*, we are given a system of linear inequalities $A\boldsymbol{x} \geq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$, and we want to enumerate the extreme optimal solutions which minimize some convex combination of given k cost vectors $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^k \in \mathbb{R}^n$.

Problem: MC-LP
Input: a matrix $A \in \mathbb{R}^{m\times n}$, two vectors $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^k \in \mathbb{R}^n$ **Enumerate**: the extreme solutions $\boldsymbol{x}$ to the inequality system $A\boldsymbol{x} \geq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$ which minimize some convex combination of $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^k$.

We call an instance of MC-LP *non-degenerated* if every extreme point of the polyhedron determined by the given inequality system lies on n facets.

Theorem 6. *The non-degenerated MC-LP can be solved in polynomial-time delay and polynomial space.*

Proof (sketch). In the feasible region every extreme solution is adjacent to other extreme solutions through edges. This adjacency naturally defines an undirected graph, and in the same way as we did for MC-MCST we can implicitly specify a rooted tree in this graph. For a non-degenerated linear program, every extreme solution is adjacent to at most n other extreme solutions, and the adjacent extreme solutions can be found by pivot operations in polynomial time. The connectedness of the analogue of $\mathcal{G}_M(G)$ is known [3]. Furthermore, we can obtain propositions similar to Lemmas 2, 3 and 4 (the proofs are similar), and thus Theorem 6 is proven. □

Note that MC-LP with possible degeneracy seems very difficult to tackle. It is known that the vertex enumeration of a degenerated convex polyhedron, which corresponds to the enumeration of the extreme solutions to a single-criterion linear program, cannot be performed in polynomial total time (hence not in polynomial-time delay and polynomial space) unless P = NP [5].

6 Concluding Remark

We have looked at some multi-criteria optimization problems from the viewpoint of enumerative algorithmics. There seem many problems in multi-criteria optimization to which the algorithm theory can potentially contribute.

A key fact in our reverse search algorithm for MC-MCST is that there are at most polynomially many spanning trees adjacent to one spanning tree. This is no longer the case if we consider the bipartite matching problem. So far, we do not know how to obtain a polynomial-time delay and polynomial-space algorithm for the multi-criteria assignment problem (i.e., maximum bipartite matching problem). We can show that a natural binary partition approach does not work in the same way as we did in Section 3. We leave this issue as an open problem.

Another problem is concerned with Lemma 2, where we saw that $\boldsymbol{\lambda}_T$ can be obtained in polynomial time. However, it uses a polynomial-time linear programming algorithm, hence not a strongly polynomial-time algorithm. We do not know whether it can be computed in strongly polynomial time.

Acknowledgments This work started when the authors visited Institute of Theoretical Computer Science, ETH Zurich in 2005. The authors are grateful to Emo Welzl and his group members for hospitality. A part of this research is supported by Grant-in-Aid for Scientific Research from Ministry of Education, Science and Culture, Japan, and Japan Society for the Promotion of Science.

References

1. D. Avis and K. Fukuda, Reverse search for enumeration. Discrete Applied Mathematics **65** (1996) 21–46.
2. M. Ehrgott, On matroids with multiple objectives. Optimization **38** (1996) 73–84.
3. M. Ehrgott, Multicriteria Optimization (Second Edition). Springer, Berlin Heidelberg, 2005.
4. M. Grötschel, L. Lovász and A. Schrijver, Geometric Algorithms and Combinatorial Optimization (2nd Corrected Edition). Springer-Verlag, Berlin New York, 1993.
5. L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, and V. Gurvich, Generating all vertices of a polyhedron is hard. Proc. 17th SODA (2006) 758–765. Full version to appear in Discrete & Computational Geometry.
6. S.-I. Nakano and T. Uno, Constant time generation of trees with specified diameter. Proc. 30th WG (2004) 33–45.
7. C. Papadimitriou and M. Yannakakis, On the approximability of trade-offs and optimal access of web sources. Proc. 41st FOCS (2000) 86–92.
8. E.L. Ulungu and J. Teghem, The two phase method: an efficient procedure to solve bi-objective combinatorial optimization problems. Foundations of Computing and Decision Sciences **20** (1995) 149–165.
9. D.B. West, Introduction to Graph Theory (Second Edition). Prentice Hall, Upper Saddle River, 2001.
10. C. Zaroliagis, Recent advances in multiobjective optimization. Proc. 3rd SAGA (2005) 45–47.
11. E. Zitzler, M. Laumanns, and S. Bleuler, A tutorial on evolutionary multiobjective optimization. In: X. Gandibleux, M. Sevaux, K. Sörensen, V. T'kindt, eds., Metaheuristics for Multiobjective Optimisation, Lecture Notes in Economics and Mathematical Systems **535** (2004) pp. 3–38.