ISSN 1342-2804

Research Reports on Mathematical and Computing Sciences

DEMiCs: A software package
for computing the mixed volume
via dynamic enumeration of all mixed cells

Tomohiko Mizutani and Akiko Takeda

April 2007, B–441
(Revised August 2007)

Department of
Mathematical and
Computing Sciences
Tokyo Institute of Technology

SERIES B: Operations Research

B-441 DEMiCs: A software package for computing the mixed volume via dynamic enumeration of all mixed cells

Tomohiko Mizutani[†1] and Akiko Takeda[†2]
April 2007, Revised August 2007.

Abstract. DEMiCs is a software package written in C++ for computing the mixed volume of the Newton polytopes of a general semi-mixed polynomial system through dynamic enumeration of all mixed cells. The underlying mixed cells play an essential role for computing all isolated zeros of a polynomial system by polyhedral homotopy continuation method. A notable feature of DEMiCs is in the construction of a dynamic enumeration tree for finding all mixed cells. The dynamic enumeration method, proposed by Mizutani, Takeda and Kojima for fully mixed polynomial systems, is extended to semi-mixed systems and incorporated in the package. Numerical results show that DEMiCs significantly is faster than existing software packages for semi-mixed polynomial systems with many distinct supports. The software package DEMiCs is available at

http://www.is.titech.ac.jp/~mizutan8/DEMiCs/.

Key words.
mixed volume, mixed cell, polyhedral homotopy, polynomial system, semi-mixed structure, dynamic enumeration.

† Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan.
†1:*mizutan8@is.titech.ac.jp*. †2:*takeda@is.titech.ac.jp*.

1 Introduction

The polyhedral homotopy continuation method, proposed by Huber and Sturmfels [15] and Verschelde et al. [24], is a powerful and reliable numerical method [7, 13, 14, 16, 17, 25] for computing all isolated zeros of a polynomial system $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$ in the variables $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{C}^n$. In order to build a family of homotopy functions, this method needs all mixed cells in a fine mixed subdivision of the Newton polytopes of a polynomial system $\boldsymbol{f}(\boldsymbol{x})$ (See [15] and [17]). It uses the mixed cells to construct homotopy functions between start systems, which are polynomial systems whose zeros can be computed easily, and target system $\boldsymbol{f}(\boldsymbol{x})$. The mixed volume, which can be obtained from the volumes of all mixed cells, is also the total number of solutions for the start systems. It is also an upper bound for the total number of isolated zeros in $(\mathbb{C} \setminus \{0\})^n$ of $\boldsymbol{f}(\boldsymbol{x})$, as guaranteed by Bernshtein's theorem [1].

The aim of this paper is to introduce the software package DEMiCs. The package uses dynamic enumeration to compute all mixed cells and the mixed volume of the support of a general semi-mixed polynomial system, including fully mixed and unmixed systems as special cases. The dynamic enumeration method was originally developed for a fully mixed system in [19], and it is significantly faster than other software packages [8, 10, 12, 18, 25, 22] for enumerating all mixed cells for large-scale semi-mixed systems with many distinct support sets. It also opens the door to computing all zeros of larger polynomial systems by polyhedral homotopy.

The mixed cells are enumerated by using a tree on which each node is provided with a linear inequality system. An enumeration tree has the following properties:

(i) A leaf node corresponds to a mixed cell if and only if the linear inequality system attached to the leaf node is feasible.

(ii) Each node shares a common system with the child nodes, so that if the node is infeasible, so are all of its descendant nodes.

There are various ways of constructing such enumeration trees. The static enumeration method [9, 10, 18, 25] fixes the structure of a tree before finding mixed cells. However, it is ideal that most of the child nodes are infeasible and pruned when branching at each node is carried out. To pursue this idea, the paper [19] proposed the dynamic enumeration method for a fully mixed system. This method chooses a suitable child node set among all the candidates generated from a parent node, so that the total number of nodes of a tree is expected to be small. In this situation, it is essential for computational efficiency to utilize information from each node to be generated during the construction of a tree. In this paper, we explain how the dynamic enumeration method is extended to a semi-mixed polynomial system. Recently, we noticed that Emiris and Canny [8] also enumerate all mixed cells dynamically for fully mixed systems, and build the dynamic enumeration tree implicitly for this purpose. Both methods use a linear programming (LP) problem as a means to enumeration mixed cells. The main differences between the two methods are as follows.

(i) Formulation for finding mixed cells: Our formulation is well suited for utilizing rich information which can be obtained by solving the LP problems at upper level of a tree.

(ii) Choice of a child node set at each node: Our method efficiently chooses a suitable child node set in all the candidates generated from a parent node, using information obtained from the LP problems to be solved during the execution of enumeration. Emiris and Canny's method select the best child node set in all the candidates without taking account of such information, and hence, it is more computationally expensive.

In fact, the numerical results in [10, 18, 22] show that Emiris and Canny's dynamic enumeration strategy has a speed disadvantage. In Subsection 3.3, we precisely describe how the two methods construct a dynamic enumeration tree.

There are several related software packages for computing the mixed volume through enumeration of all mixed cells: HOM4PS [11], MixedVol [10, 12], MVLP [8], PHCpack [25], PHoM [22], mvol [18], etc. HOM4PS, PHCpack and PHoM, written in FORTRAN, Ada and C++ respectively, are software packages for computing all isolated zeros of a polynomial system by polyhedral homotopy. These packages each contain a module for enumeration of mixed cells. In particular, PHCpack is the most popular of these software packages. The C package MVLP uses a dynamic enumeration for finding all mixed cells. The C++ package MixedVol, which employs the static enumeration method, specializes in the mixed volume computation. MixedVol performs better than all other software packages in terms of computational efficiency and memory usage, as reported in the papers [10, 12]. A feature of the package is that all mixed cells can be generated efficiently for a semi-mixed system by taking account of the special structure of a semi-mixed support.

Numerical results show that DEMiCs can find all mixed cells much faster than the other software packages [8, 10, 12, 18, 25, 22] not only for fully mixed polynomial systems but also for semi-mixed polynomial systems. Although DEMiCs has a speed advantage over the other packages for large-scale fully mixed systems and semi-mixed systems with many distinct support sets, it does not always excel. Indeed, for unmixed systems and semi-mixed systems with a few distinct supports, DEMiCs is slower than MixedVol [12] because of its computation overhead associated with the dynamic branching rule.

This paper is organized as follows. Section 2 and 3 describe the technical details of our method. Section 2 outlines the dynamic enumeration method for a general semi-mixed polynomial system, and Section 3 explains how to check the feasibility of each node and how to construct an enumeration tree when a polynomial system is a semi-mixed type. Section 4 describes the usage of DEMiCs. Section 5 compares the performance of DEMiCs with those of the other software packages on artificial semi-mixed polynomial systems and well-known large-scale benchmark systems. Section 6 is devoted to concluding remarks.

2 Dynamic enumeration algorithm for a semi-mixed system

2.1 Preliminaries

In this paper, we represent each component polynomial $f_i(\boldsymbol{x})$ in a polynomial system $\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$ in $\boldsymbol{x} \in \mathbb{C}^n$ as

$$f_i(\boldsymbol{x}) = \sum_{\boldsymbol{a} \in \mathcal{A}_i} c_i(\boldsymbol{a})\ \boldsymbol{x}^{\boldsymbol{a}},$$

using a nonempty finite subset $\mathcal{A}_i$ of $\mathbb{Z}^n_+$ and nonzero $c_i(\boldsymbol{a}) \in \mathbb{C}$ for $\boldsymbol{a} \in \mathcal{A}_i$. Here $\mathbb{Z}^n_+$ denotes the set of nonnegative integer vectors in $\mathbb{R}^n$, $\mathbb{R}$ and $\mathbb{C}$ are the sets of real and complex numbers, respectively, and $\boldsymbol{x}^{\boldsymbol{a}} = x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$ for $\boldsymbol{a} = (a_1, a_2 \ldots, a_n) \in \mathbb{Z}^n_+$. We call the set $\mathcal{A}_i$ the support of $f_i(\boldsymbol{x})$, and $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n)$ the support of $\boldsymbol{f}(\boldsymbol{x})$. The convex hull $\mathcal{P}_i := \mathrm{conv}(\mathcal{A}_i)$ is called the *Newton polytope* of f_i. Define $N = \{1, 2, \ldots, n\}$. The following describes the definition of the mixed volume of the n-tuple Newton polytopes $\mathcal{P}_i$ in $\mathbb{R}^n$. For all positive number $\lambda_1, \lambda_2, \ldots, \lambda_n$, it is known that the n-dimensional volume of the Minkowski sum

$$\lambda_1 \mathcal{P}_1 + \lambda_2 \mathcal{P}_2 + \cdots + \lambda_n \mathcal{P}_n = \{\lambda_1 p_1 + \lambda_2 p_2 + \cdots + \lambda_n p_n : p_i \in \mathcal{P}_i,\ i \in N\}$$

is given by a homogeneous polynomial of degree n in λ_i ($i \in N$). The mixed volume for $\mathcal{A}$ is defined to be the coefficient of $\lambda_1 \lambda_2 \cdots \lambda_n$ in the polynomial.

Some support sets in $\mathcal{A} = (\mathcal{A}_i : i \in N)$ of $\boldsymbol{f}(\boldsymbol{x})$ may be equal to each other. We suppose that the polynomial system has exactly m ($\leq n$) distinct support sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ among $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ such that

$$\mathcal{S}_i := \mathcal{A}_{j_1} = \mathcal{A}_{j_2} \text{ for every } j_1, j_2 \in I_i \quad (i \in M), \tag{1}$$

where we define $M = \{1, 2, \ldots, m\}$. Obviously, the subset I_i of N satisfies $\cup_{i \in M} I_i = N$ and $I_{i_1} \cap I_{i_2} = \emptyset$ for every $i_1, i_2 \in M$. A polynomial system with the support $\mathcal{S} = (\mathcal{S}_i : i \in M)$ is called *semi-mixed*. The system is called *unmixed* when $m = 1$, and *fully mixed* when $m = n$, and it is called *semi-mixed of type* $(k_1, k_2, \ldots, k_m)$ if and only if $\mathcal{S}_i$ occurs exactly k_i times in $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$, *i.e.*, $\#I_i = k_i$. In this paper, we treat a semi-mixed polynomial system $\boldsymbol{f}(\boldsymbol{x})$ of type $(k_1, k_2, \ldots, k_m)$ with support $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m)$, and assume that each support set $\mathcal{S}_i$ consists of r_i elements.

For a support $\mathcal{S} = (\mathcal{S}_i : i \in M)$ of a semi-mixed system described above, we use the notation $\mathcal{Q}_i = \mathrm{conv}(\mathcal{S}_i)$ for the Newton polytope of each polynomial. The mixed volume for $\mathcal{S} = (\mathcal{S}_i : i \in M)$ of a semi-mixed type can be computed by finding all *mixed cells* in *a fine mixed subdivision* of the Minkowski sum $\mathcal{Q} = \mathcal{Q}_1 + \mathcal{Q}_2 + \cdots + \mathcal{Q}_m$. These are essentially piece polytopes, called *cells*, in a polyhedral subdivision of $\mathcal{Q}$. The reader may want to refer to the definition of a fine mixed subdivision in [6, 15]. It is known that each cell in a fine mixed subdivision can be represented as the Minkowski sum of simplices $R_j := \mathrm{conv}(C_1^j) + \mathrm{conv}(C_2^j) + \cdots + \mathrm{conv}(C_m^j)$ for $\boldsymbol{C} = (C_1^j, C_2^j, \ldots, C_m^j)$ where each C_i^j is the

subset of $\mathcal{S}_i$ and its convex hull $\mathrm{conv}(C_i^j)$ is a simplex of dimension $\#C_i^j - 1$. In particular, when a polynomial system is a semi-mixed system of type $(k_1, k_2, \ldots, k_m)$, we call a cell R_j that is described as a Minkowski sum of each $\mathrm{conv}(C_i^j)$, a *mixed cell* if $\dim(\mathrm{conv}(C_i^j)) = k_i$ for every $i \in M$. It is shown in [15, Theorem 2.4.] that the mixed volume for $\mathcal{S}$ is obtained from the volumes of all mixed cells in a fine mixed subdivision of $\mathcal{Q}$. This means that only distinct support sets contribute the mixed volume computation. A semi-mixed system certainly can be treated as a fully mixed type, and the mixed volume can be obtained by summing the volumes of all mixed cells in a fine mixed subdivision of $\mathcal{P} = \mathcal{P}_1 + \mathcal{P}_2 + \cdots + \mathcal{P}_n$. However, the numerical results in [10] show that the computational time for finding all mixed cells can be reduced if we focus on the only distinct support sets for a semi-mixed system. Therefore, it is important for computational efficiency to utilize a semi-mixed structure.

The papers [2, 15] describe how to construct a fine mixed subdivision of $\mathcal{Q}$ by using a real-valued function $\omega_i : \mathcal{S}_i \to \mathbb{R}$. The function ω_i lifts $\mathcal{S}_i$ to

$$\hat{\mathcal{S}}_i = \left\{ \begin{pmatrix} \boldsymbol{a} \\ \omega_i(\boldsymbol{a}) \end{pmatrix} : \boldsymbol{a} \in \mathcal{S}_i \right\}.$$

Let $\hat{\mathcal{Q}}$ denote the Minkowski sum $\hat{\mathcal{Q}} = \hat{\mathcal{Q}}_1 + \hat{\mathcal{Q}}_2 + \cdots + \hat{\mathcal{Q}}_m$ for $\hat{\mathcal{Q}}_i = \mathrm{conv}(\hat{\mathcal{S}}_i)$. For the subset $\hat{C}_i$ of $\hat{\mathcal{S}}_i$, we will use the notation $\hat{\boldsymbol{C}} = (\hat{C}_1, \hat{C}_2, \ldots, \hat{C}_m)$. Suppose that the function ω_i gives a random number so that the value $\omega_i(\boldsymbol{a})$ is sufficiently generic for every $i \in M$ and every $\boldsymbol{a} \in \mathcal{S}_i$. Then, the projection $\mathbb{R}^{n+1} \to \mathbb{R}^n$ of the set of lower facets of $\hat{\mathcal{Q}}$ gives a fine mixed subdivision of $\mathcal{Q}$. In this paper, we call such a function ω_i a *generic lifting*.

Li and Li [18] proposed an efficient algorithm for finding lower facets of $\hat{\mathcal{Q}}$ via an enumeration tree. Recently, for a fully mixed polynomial system, the paper [19] improved their algorithm by replacing a static enumeration tree of [9, 10, 18, 22] with a dynamic enumeration tree. In this paper, a dynamic enumeration method is applied to find all mixed cells in a fine mixed subdivision for a semi-mixed system, including a fully mixed and unmixed type.

2.2 Algorithm

We briefly describe the *dynamic enumeration algorithm* of [19] and apply it to a semi-mixed polynomial system. For every $L \subseteq M = \{1, 2, \ldots, m\}$, we define

$$\begin{aligned} \Omega(L) &= \left\{ \boldsymbol{C} = (C_1, C_2, \ldots, C_m) : \begin{array}{l} C_i \subseteq \mathcal{S}_i,\ \#C_i = k_i + 1\ (i \in L) \\ C_j = \emptyset\ (j \notin L) \end{array} \right\} \\ \Omega &= \cup_{L \subseteq M} \Omega(L). \end{aligned}$$

The set Ω represents the collection of all nodes in an enumeration tree. The tree has a root node $\emptyset^m \in \Omega(\emptyset) := \{\emptyset^m\}$ and the leaf nodes $\Omega(M) \subset \Omega$. A node at the ℓth level corresponds to the element in $\cup_{L \subseteq M, \#L = \ell} \Omega(L)$. Let $L(\boldsymbol{C}) = \{i \in M : C_i \neq \emptyset\}$ for any $\boldsymbol{C} \in \Omega(L)$ $(L \subseteq M)$. This definition is used for extracting every index of nonempty sets C_i from $\boldsymbol{C} = (C_1, C_2, \ldots, C_m) \in \Omega(L)$, *i.e.* $L(\boldsymbol{C}) = L$ for $\boldsymbol{C} \in \Omega(L)$ $(L \subseteq M)$. Each node $\boldsymbol{C} = (C_i : i \in L) \in \Omega(L)$ with $L \subseteq M$ has a linear inequality system $\mathcal{I}(\boldsymbol{C})$:

$$\mathcal{I}(\boldsymbol{C}) := \left\{ \begin{array}{ll} \langle \hat{\boldsymbol{a}}_i, \hat{\boldsymbol{\alpha}} \rangle = \langle \hat{\boldsymbol{a}}_i', \hat{\boldsymbol{\alpha}} \rangle, & \forall \hat{\boldsymbol{a}}_i, \hat{\boldsymbol{a}}_i' \in \hat{C}_i \\ \langle \hat{\boldsymbol{a}}_i, \hat{\boldsymbol{\alpha}} \rangle \leq \langle \hat{\boldsymbol{a}}, \hat{\boldsymbol{\alpha}} \rangle, & \forall \hat{\boldsymbol{a}} \in \hat{\mathcal{S}}_i \setminus \hat{C}_i \end{array} \quad (i \in L(\boldsymbol{C})), \right. \tag{2}$$

where

$$\hat{\boldsymbol{\alpha}} = \begin{pmatrix} \boldsymbol{\alpha} \\ 1 \end{pmatrix} \in \mathbb{R}^{n+1}.$$

$\langle \cdot, \cdot \rangle$ stands for the usual inner product in Euclidean space. Note that $\hat{\mathcal{S}}_i$ is obtained by applying a generic lifting ω_i to $\mathcal{S}_i$, and $\hat{C}_i$ is the subset of $\hat{\mathcal{S}}_i$. Li and Li showed in [18] that any mixed cell in a fine mixed subdivision of $\mathcal{S} = (\mathcal{S}_i : i \in M)$ is in one-to-one correspondence to $\boldsymbol{C} = (C_1, C_2, \ldots, C_m)$ with $C_i \subseteq \mathcal{S}_i$ and $\#C_i = k_i + 1$ for each $i \in M$ such that the linear inequality system $\mathcal{I}(\boldsymbol{C})$ is feasible. We say that $\boldsymbol{C} \in \Omega$ is a feasible node when $\mathcal{I}(\boldsymbol{C})$ is feasible. Let

$$\Omega^* = \{\boldsymbol{C} \in \Omega(M) : \boldsymbol{C} \text{ is feasible}\}.$$

Then we can easily see from (2) that Ω^* consists of every mixed cell in a fine mixed subdivision.

For $\boldsymbol{C} \in \Omega$ and $L \subseteq M$, we use the notation $\boldsymbol{C}_L$ for $\boldsymbol{C}_L = (C_i : i \in L)$. Regarding the root node $\emptyset^m \in \Omega$ as a feasible node, we construct an enumeration tree according to Algorithm 2.1 of [19]. Namely, for a node $\boldsymbol{C} \in \Omega(L)$ with the proper subset $L \subsetneq M$ and $t \in M \setminus L(\boldsymbol{C})$ we generate the child node set $W(\boldsymbol{C}, t)$ of $\boldsymbol{C}$

$$W(\boldsymbol{C}, t) = \left\{\bar{\boldsymbol{C}} \in \Omega(L(\boldsymbol{C}) \cup \{t\}) : \bar{\boldsymbol{C}}_{L(\boldsymbol{C})} = \boldsymbol{C}_{L(\boldsymbol{C})}\right\}.$$

Starting from the root node $\emptyset^m \in \Omega$, we choose t from $M \setminus L(\boldsymbol{C})$ at each node $\boldsymbol{C} \in \Omega(L)$ with $L \subsetneq M$ and create child nodes of $\boldsymbol{C}$ until $\#L = m - 1$ based on the algorithm. Thus, Ω^* coincides with the set of the feasible leaf nodes $\boldsymbol{C} \in \Omega(M)$. If we check the feasibility of all leaf nodes, all mixed cells in a fine mixed subdivision can be obtained. Note that this algorithm produces various types of trees depending on a choice of an index $t \in M \setminus L(\boldsymbol{C})$ at each node $\boldsymbol{C} \in \Omega(L)$ with $L \subsetneq M$. *A static enumeration tree* is constructed in the previous works [9, 10, 18, 22], which specify how to choose an index $t \in M \setminus L(\boldsymbol{C})$ at each node $\boldsymbol{C} \in \Omega(L)$ with $L \subsetneq M$ before the building of a tree. In contrast to this, the paper [19] develops a *dynamic enumeration tree* by choosing a suitable index t from $M \setminus L(\boldsymbol{C})$ at each node $\boldsymbol{C} \in \Omega(L)$ with $L \subsetneq M$.

We can enumerate feasible leaf nodes of a tree efficiently if the following property is taken into account. The feasible region of the linear inequality system $\mathcal{I}(\boldsymbol{C})$ attached to a node $\boldsymbol{C}$ contains that of $\mathcal{I}(\bar{\boldsymbol{C}})$, which corresponds to a child node $\bar{\boldsymbol{C}}$ of $\boldsymbol{C}$. That is, we can say that if a parent node $\boldsymbol{C}$ is infeasible, then all child nodes $\bar{\boldsymbol{C}} \in W(\boldsymbol{C}, t)$ $(t \in M \setminus L(\boldsymbol{C}))$ are infeasible. If a node is detected to be infeasible, we can prune a subtree having the node as the root node because there are no mixed cells in the subtree. Therefore, by replacing $W(\boldsymbol{C}, t)$ in Algorithm 2.1 of [19] with

$$W^*(\boldsymbol{C}, t) = \{\bar{\boldsymbol{C}} \in W(\boldsymbol{C}, t) : \bar{\boldsymbol{C}} \text{ is feasible}\} \subseteq W(\boldsymbol{C}, t),$$

every mixed cell can be found as the feasible leaf nodes in the tree. Taking account of this property, we search for all nodes in Ω^* according to the dynamic enumeration algorithm stated below. We will use the notation A_ℓ $(\ell \in \{0\} \cup M)$ for the set of feasible nodes.

Once this algorithm terminates, A_m contains all nodes in Ω^*, *i.e,* every mixed cell can be stored in A_m. We introduce this algorithm, which constructs a tree in breadth-first

Algorithm 1 Dynamic enumeration algorithm

Input: A support $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m)$.

Output: All mixed cells in a fine mixed subdivision.

$A_i \leftarrow \emptyset$ for all $i \in M$.
$A_0 \leftarrow \emptyset^m$ and $\ell \leftarrow 0$.

while $\ell < m$ **do**
  for all $\boldsymbol{C} \in A_\ell$ **do**
    Choose t from $M \setminus L(\boldsymbol{C})$: (A)
    $A_{\ell+1} \leftarrow A_{\ell+1} \cup W^*(\boldsymbol{C}, t)$: (B)
  end for
  $\ell \leftarrow \ell + 1$.
end while

order, for simplicity of description; it is essentially the same as Algorithm 2.2 described in [19], which is in depth-first order. Hence, we can obtain the same result from these two algorithms though there is the difference in the search order for feasible leaf nodes. The software package DEMiCs employs the depth-first order, which is similar to Algorithm 2.2 in [19], to save memory.

The following two issues have a major effect on the computational efficiency of the dynamic enumeration algorithm for semi-mixed systems.

(i) How we choose an index t from $M \setminus L(\boldsymbol{C})$ in (A).

(ii) How we construct $W^*(\boldsymbol{C}, t)$ in (B).

As for (i), in the *static enumeration method* proposed in the previous works [9, 10, 18, 22], we set up a permutation π of M before starting the algorithm, and choose the index t such as $t = \pi(\ell + 1) \in M \setminus L(\boldsymbol{C})$. Hence, a choice of an index t at (A) is determined by a permutation π. Here we employ a refinement of the *dynamic enumeration method* from [19]. Suppose that $\boldsymbol{C}$ is a feasible node in A_ℓ with $\ell < m$. Then, we consider a choice of an index t from $M \setminus L(\boldsymbol{C})$ so that many child nodes of $\boldsymbol{C}$ are expected to be infeasible. Ideally, we would like to choose the index t such that the size of $W^*(\boldsymbol{C}, t)$ is the smallest among $t \in M \setminus L(\boldsymbol{C})$. Although this strategy is employed by Emiris and Canny's method [8], it is costly because we need to construct $W^*(\boldsymbol{C}, t)$ for each $t \in M \setminus L(\boldsymbol{C})$. Therefore, instead of $W^*(\boldsymbol{C}, t)$, we consider another set which can be constructed easily. In Subsection 3.3, we explain how to construct this set.

As for (ii), it may not be an easy task to find all feasible nodes in $W(\boldsymbol{C}, t)$ because the size of $W(\boldsymbol{C}, t)$ is not small. So we embed the construction process for $W^*(\boldsymbol{C}, t)$ in a tree, and prune worthless subtrees in order to reduce computational effort. In Subsection 3.1 we discuss the details of this procedure, and show the formulation of the feasibility check of a node in Subsection 3.2.

3 Feasibility check for a semi-mixed system

3.1 Tree structure for construction of $W^*(\boldsymbol{C}, t)$ in (B)

We now explain a tree structure for finding all elements in $W^*(\boldsymbol{C}, t)$ efficiently. Suppose that an index t is chosen from $M \setminus L(\boldsymbol{C})$ for a feasible node $\boldsymbol{C} \in A_\ell$ with $\ell < m$ in (B) of Algorithm 1. Then, we would like to construct $W^*(\boldsymbol{C}, t) \subseteq W(\boldsymbol{C}, t)$. For a nonnegative integer k, let

$$\Gamma(k; \boldsymbol{C}, t) = \left\{ U = (U_1, U_2, \ldots, U_m) : \begin{array}{l} \boldsymbol{U}_{L(\boldsymbol{C})} = \boldsymbol{C}_{L(\boldsymbol{C})} \\ U_t \subseteq \mathcal{S}_t, \ \#U_t = k \\ U_i = \emptyset \ (i \notin L(\boldsymbol{C}) \cup \{t\}) \end{array} \right\}.$$

Note that $\Gamma(k_t + 1; \boldsymbol{C}, t) = W(\boldsymbol{C}, t)$.

For a feasible node $\boldsymbol{C} \in A_\ell$ with $\ell < m$ in the dynamic enumeration algorithm, we build a tree T for constructing $W^*(\boldsymbol{C}, t)$. The tree structure is outlined as follows. Let $K_t = \{0, 1, \ldots, k_t + 1\}$. The set

$$\Gamma := \cup_{k \in K_t} \Gamma(k; \boldsymbol{C}, t)$$

serves as the collection of all nodes in the tree. The tree has $\boldsymbol{C} \in \Gamma(0; \boldsymbol{C}, t) = \{\boldsymbol{C}\}$ as the root node, and $\boldsymbol{U} \in \Gamma(k; \boldsymbol{C}, t)$ with $\#U_t = k$ as a node at the kth level. Each node $\boldsymbol{U} \in \Gamma(k; \boldsymbol{C}, t)$ has a linear inequality system $\mathcal{I}(\boldsymbol{U})$ in a variable vector $\hat{\boldsymbol{\alpha}} \in \mathbb{R}^{n+1}$. Note that each system $\mathcal{I}(\boldsymbol{U})$ with $\boldsymbol{U} \in \Gamma(k_t + 1; \boldsymbol{C}, t)$ is identical to the linear inequality system $\mathcal{I}(\bar{\boldsymbol{C}})$ at a node $\bar{\boldsymbol{C}} \in W(\boldsymbol{C}, t)$. Therefore, $W^*(\boldsymbol{C}, t)$ can be obtained by checking the feasibility of any node $\boldsymbol{U} \in \Gamma(k_t + 1; \boldsymbol{C}, t)$ which corresponds to each leaf node at the $(k_t + 1)$th level.

We now describe more precisely the tree structure for building $W^*(\boldsymbol{C}, t)$. For $U_t \subseteq \mathcal{S}_t$, we choose a function $m_t : \mathcal{S}_t \to \mathbb{Z}$ which gives the maximum number i among the indices of $\boldsymbol{a}_i \in U_t$. Namely, $m_t(U_t) = \max\{i \in \mathbb{Z} : \boldsymbol{a}_i \in U_t\}$ for $U_t \subseteq \mathcal{S}_t$. Let $T = (V, E)$ be a rooted tree, which describes the relation among elements of a node set Γ. Recall that $\mathcal{S}_t$ has r_t elements. For a node $\boldsymbol{U} \in \Gamma(k; \boldsymbol{C}, t)$, we generate the child node set

$$Z(\boldsymbol{U}; \boldsymbol{C}, t) = \left\{ \bar{\boldsymbol{U}} \in \Gamma(\#U_t + 1; \boldsymbol{C}, t) : \begin{array}{c} \bar{U}_t = U_t \cup \{\boldsymbol{a}_i\}, \\ \text{for every } i, \ m_t(U_t) < i \le r_t \end{array} \right\}$$

and construct $T = (V, E)$ with $V = \bigcup_{k \in K_t} V_k$ and $E = \bigcup_{k \in K_t} E_k$, based on Algorithm 2.

The root node of T, constructed by the algorithm, corresponds to $V_0 = \{\boldsymbol{C}\}$, where $\boldsymbol{C}$ is one of the elements in A_ℓ generated by Algorithm 1. When this algorithm terminates, V_k stores every element in $\Gamma(k; \boldsymbol{C}, t)$ for all $k \in K_t$. Hence, $W(\boldsymbol{C}, t)$, which is generated by a feasible $\boldsymbol{C} \in A_\ell$ and an index $t \in M \setminus L(\boldsymbol{C})$ in Algorithm 1, is equal to V_{k_t+1}, which has each leaf node at the $(k_t + 1)$th level of the tree T.

Example. We consider the distinct support sets $\mathcal{S}_1, \mathcal{S}_2$ ($\mathcal{S}_i \subseteq \mathbb{Z}_+^3$) of the semi-mixed system $\boldsymbol{f}(\boldsymbol{x})$ ($\boldsymbol{x} \in \mathbb{C}^3$) of type $(1, 2)$ such that $|\mathcal{S}_1| = 3$ and $|\mathcal{S}_2| = 4$. We execute Algorithm 1 for the support $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$. Suppose that the algorithm produces a nonempty set A_1 when $t = 1$ is chosen in (A) at the first iteration. Figure 1 shows the shape of the tree

Algorithm 2 Construction of the tree $T = (V, E)$

Input: A feasible node $\boldsymbol{C} \in A_\ell$ and an index $t \in M \setminus L(\boldsymbol{C})$.

Output: $T = (V, E)$ with $V = \bigcup_{k \in K_t} V_k$ and $E = \bigcup_{k \in K_t} E_k$.

$V_0 \leftarrow \boldsymbol{C}$, $E_0 \leftarrow \emptyset$ and $k \leftarrow 0$.
$V_i \leftarrow \emptyset$ and $E_i \leftarrow \emptyset$ for all $i \in K_t \setminus \{0\}$.

while $k < k_t + 1$ **do**
  for all $\boldsymbol{U} \in V_k$: (♮) **do**
    if $m_t(U_t) < r_t$ **then**
      $V_{k+1} \leftarrow V_{k+1} \cup Z(\boldsymbol{U}; \boldsymbol{C}, t)$
      $E_{k+1} \leftarrow E_{k+1} \cup \{(\boldsymbol{U}, \bar{\boldsymbol{U}})) \in V_k \times V_{k+1} : \bar{\boldsymbol{U}} \in Z(\boldsymbol{U}; \boldsymbol{C}, t)\}$
    end if
  end for
  $k \leftarrow k + 1$.
end while

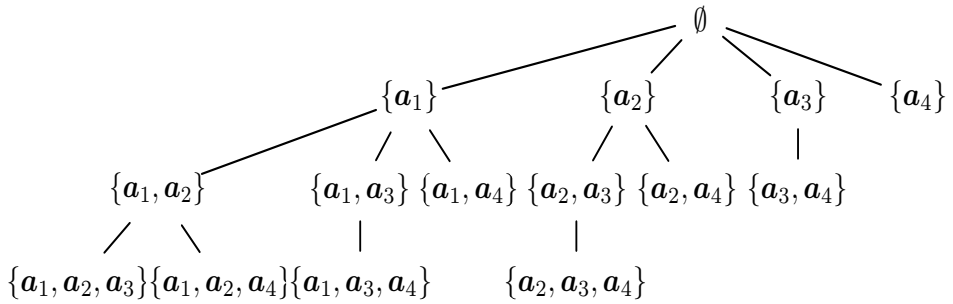

Figure 1: Tree structure generated by Algorithm 2

generated by Algorithm 2 for input data $\boldsymbol{C} \in A_1$. Here, each label of nodes represents $U_2 \subseteq \boldsymbol{\mathcal{S}}_2$ of $\boldsymbol{U} = (U_1, U_2) \in \Gamma(k; \boldsymbol{C}, 2)$ $(0 \leq k \leq 3)$.

For a node $\boldsymbol{U} \in V_k$ and child node $\bar{\boldsymbol{U}} \in V_{k+1}$ of $\boldsymbol{U}$, the feasible region of $\mathcal{I}(\boldsymbol{U})$ contains that of $\mathcal{I}(\bar{\boldsymbol{U}})$. Hence, if the node $\boldsymbol{U} \in V_k$ is infeasible, every child node $\bar{\boldsymbol{U}} \in V_{k+1}$ of $\boldsymbol{U}$ is infeasible. Therefore, the subtree with root node $\boldsymbol{U}$ can be pruned, since it does not contain any feasible leaf nodes at the $(k_t + 1)$th level. Accordingly, even if we replace V_k at (♮) of Algorithm 2 by

$$V_k^* = \{\boldsymbol{U} \in V_k : \boldsymbol{U} \text{ is feasible}\},$$

we can find every feasible leaf node at the (k_t+1)th level in $V_{k_t+1}^*$, and thus obtain $W^*(\boldsymbol{C}, t)$.

After building $T = (V, E)$, we enumerate all feasible nodes at the $(k_t + 1)$th level in T and construct $V_{k_t+1}^*$. First, we initialize V_0^* as $V_0^* = V_0$. Next, we repeat the following

procedure until $k = k_t$. Suppose that V_k^* is constructed. Then, we check the feasibility of every child node $\bar{\boldsymbol{U}}$ of $\boldsymbol{U} \in V_k^*$ and store the feasible nodes in V_{k+1}^*. As a result, $W^*(\boldsymbol{C}, t)$ is obtained as $V_{k_t+1}^*$. Note that here we use Algorithm 2 having a breadth-first order for the simplicity of description and that DEMiCs has a depth-first order to save memory in constructing $W^*(\boldsymbol{C}, t)$.

3.2 Formulation of the feasibility check

Suppose that a tree $T = (V, E)$ with $V = \bigcup_{k \in K_t} V_k$ and $E = \bigcup_{k \in K_t} E_k$ is generated by Algorithm 2 for a feasible node $\boldsymbol{C} \in A_\ell$ and an index $t \in M \setminus L(\boldsymbol{C})$. Then, we need to check the feasibility of each node $\boldsymbol{U} \in V_k$ in order to construct V_k^*. An LP problem can be formulated for the feasibility check of a node $\boldsymbol{U} \in V_k$. Let $\boldsymbol{\gamma}$ denote a specific n-dimensional real vector, and furthermore let $\hat{\boldsymbol{\gamma}}^T = (\boldsymbol{\gamma}^T, 0)$. To determine whether a node $\boldsymbol{U} \in V_k$ is feasible or not, we solve the following problem in a variable vector $\hat{\boldsymbol{\alpha}} \in \mathbb{R}^{n+1}$:

$$\mathrm{P}(\boldsymbol{U}) : \ \big| \ \text{maximize} \ \ \langle \hat{\boldsymbol{\gamma}}, \hat{\boldsymbol{\alpha}} \rangle \ \ \text{subject to} \ \ \mathcal{I}(\boldsymbol{U}).$$

Let $\boldsymbol{a}_i$ be an element in U_i for any $i \in L(\boldsymbol{U})$. The dual problem in a variable vector $\boldsymbol{x} \in \mathbb{R}^d$, where $d = \sum_{i \in L(\boldsymbol{U})} (r_i - 1)$, is written as

$$\mathrm{D}(\boldsymbol{U}) : \left| \begin{array}{lll} \text{minimize} & \Phi(\boldsymbol{x}; \boldsymbol{U}) & \\ \text{subject to} & \Psi(\boldsymbol{x}; \boldsymbol{U}) = \boldsymbol{\gamma}, & \\ & -\infty < x_{\boldsymbol{b}} < +\infty & \boldsymbol{b} \in U_i \setminus \{\boldsymbol{a}_i\} \\ & x_{\boldsymbol{b}'} \geq 0 & \boldsymbol{b}' \in \mathcal{S}_i \setminus U_i \quad (i \in L(\boldsymbol{U})). \end{array} \right.$$

Here, the linear functions $\Phi(\boldsymbol{x}; \boldsymbol{U})$ and $\Psi(\boldsymbol{x}; \boldsymbol{U})$ in $\boldsymbol{x} \in \mathbb{R}^d$ are defined as follows:

$$\Phi(\boldsymbol{x}; \boldsymbol{U}) = \sum_{i \in L(\boldsymbol{U})} \sum_{\boldsymbol{a} \in \mathcal{S}_i \setminus \{\boldsymbol{a}_i\}} (\omega_i(\boldsymbol{a}) - \omega_i(\boldsymbol{a}_i)) \, x_{\boldsymbol{a}}$$
$$\text{and} \quad \Psi(\boldsymbol{x}; \boldsymbol{U}) = \sum_{i \in L(\boldsymbol{U})} \sum_{\boldsymbol{a} \in \mathcal{S}_i \setminus \{\boldsymbol{a}_i\}} (\boldsymbol{a}_i - \boldsymbol{a}) \, x_{\boldsymbol{a}},$$

where

$$\boldsymbol{x} = (x_{\boldsymbol{a}} : \boldsymbol{a} \in \mathcal{S}_i \setminus \{\boldsymbol{a}_i\}, \ i \in L(\boldsymbol{U})) \in \mathbb{R}^d.$$

Any real vector $\boldsymbol{\gamma} \in \mathbb{R}^n$ can be chosen. If $\boldsymbol{\gamma}$ is fixed so that $\mathrm{D}(\boldsymbol{U})$ is feasible, the duality theorem holds for this primal-dual pair. $\mathrm{P}(\boldsymbol{U})$ is feasible if and only if $\mathrm{D}(\boldsymbol{U})$ is bounded below, and $\mathrm{P}(\boldsymbol{U})$ is infeasible if and only if $\mathrm{D}(\boldsymbol{U})$ is unbounded. Hence, the feasibility of $\mathrm{P}(\boldsymbol{U})$ can be revealed if the boundedness of $\mathrm{D}(\boldsymbol{U})$ is detected. The following describes how we set up $\boldsymbol{\gamma}$. Recall that $\boldsymbol{U} \in V_k$ is a node at the kth level in $T = (V, E)$ generated by Algorithm 2, and $\boldsymbol{C} \in V_0$ is the root node of T. We note that the feasible region of $\mathrm{D}(\boldsymbol{C})$ is included in that of $\mathrm{D}(\boldsymbol{U})$ for any $\boldsymbol{U} \in V_k$. Consider a right-hand vector $\boldsymbol{\gamma}$ of $\mathrm{D}(\boldsymbol{U})$ for any $\boldsymbol{U} \in V_k$ as

$$\tilde{\boldsymbol{\gamma}} = \Psi(\tilde{\boldsymbol{x}}; \boldsymbol{C})$$

using an arbitrary nonnegative vector $\tilde{\boldsymbol{x}} \in \mathbb{R}^d$. Then, $\mathrm{D}(\boldsymbol{U})$ becomes feasible for each $\boldsymbol{U} \in V_k$.

Furthermore, we can easily find an initial feasible solution for the dual problem $\mathrm{D}(\boldsymbol{U})$ for any $\boldsymbol{U} \in V_k$ by using an optimal solution of $\mathrm{D}(\boldsymbol{C})$. Recall that the root node $\boldsymbol{C} \in V_0$ was revealed to be feasible. That is, we solved $\mathrm{D}(\boldsymbol{C})$ and obtained an optimal solution $\boldsymbol{x}^* \in \mathbb{R}^d$ of $\mathrm{D}(\boldsymbol{C})$, where $d = \sum_{i \in L(\boldsymbol{C})}(r_i - 1)$. For any $\boldsymbol{U} \in V_k$, the vector

$$\begin{pmatrix} \boldsymbol{x}_* \\ \boldsymbol{0} \end{pmatrix} \in \mathbb{R}^{\bar{d}}, \text{ where } \bar{d} = \sum_{i \in L(\boldsymbol{U})} (r_i - 1). \tag{3}$$

becomes a feasible solution of $\mathrm{D}(\boldsymbol{U})$. In addition, if $\mathrm{D}(\boldsymbol{U})$ is bounded below for $\boldsymbol{U} \in V_k$ with $k < k_t + 1$, an optimal solution of $\mathrm{D}(\boldsymbol{U})$ is a feasible solution of $\mathrm{D}(\bar{\boldsymbol{U}})$ for any child node $\bar{\boldsymbol{U}} \in V_{k+1}$ of $\boldsymbol{U}$ since the feasible region of $\mathrm{D}(\boldsymbol{U})$ is included in that of $\mathrm{D}(\bar{\boldsymbol{U}})$. The simplex method is suitable for solving these dual problems with the common structure. Assume that a node $\boldsymbol{U} \in V_k$ $(k < k_t + 1)$ is feasible and generates the child nodes $\bar{\boldsymbol{U}} \in V_{k+1}$ of $\boldsymbol{U}$. The simplex method usually does not require many iterations for solving $\mathrm{D}(\bar{\boldsymbol{U}})$ when we reuse an optimal solution of $\mathrm{D}(\boldsymbol{U})$ as an initial solution. In terms of problem size, the dual problems are superior to the primal ones as stated in [19]. Therefore, we employ the dual problems to check the feasibility of a node and solve these problems using the simplex method.

3.3 Choice of $t \in M \setminus L(\boldsymbol{C})$ in (A)

In Algorithm 1, we want to detect an index t from $M \setminus L(\boldsymbol{C})$ for a node $\boldsymbol{C} \in A_\ell$ with $\ell < m$, so that a large portion of the child nodes are infeasible. The best way to achieve this is to find an index t among $t \in M \setminus L(\boldsymbol{C})$ such that the size of $W^*(\boldsymbol{C}, t)$ attains the minimum size. Emiris and Canny's method [8] employs this strategy and selects the best support set from all candidates. However, this approach might be unrealistic because it requires a large computation to construct $W^*(\boldsymbol{C}, t)$ for every $t \in M \setminus L(\boldsymbol{C})$. In fact, the numerical results in [10, 18, 22] show that MVLP, in which Emiris and Canny's method is implemented, is slower than MixedVol [10], PHoM [22] and mvol [18]. Remember that each element in $W(\boldsymbol{C}, t)$ is represented as the leaf nodes $\boldsymbol{U} \in V_{k_t}$ of a tree produced by Algorithm 2, and the leaf nodes are infeasible if the ancestor node of these leaf nodes is infeasible. To restrict the number of feasibility checks, we therefore focus on finding feasible nodes in V_1, which is the set of every node at the first level of the tree and is identical to $\Gamma(1; \boldsymbol{C}, t)$. Obviously, the set

$$W_1^*(\boldsymbol{C}, t) = \{\bar{\boldsymbol{C}} \in \Gamma(1; \boldsymbol{C}, t) : \bar{\boldsymbol{C}} \text{ is feasible}\}$$

satisfies

$$W^*(\boldsymbol{C}, t) \subseteq W_1^*(\boldsymbol{C}, t) \subseteq W(\boldsymbol{C}, t).$$

Using a feasible solution $\boldsymbol{x}_{init}$ of $\mathrm{D}(\boldsymbol{U})$ $(\boldsymbol{U} \in \Gamma(1; \boldsymbol{C}, t))$ which can be easily obtained from an optimal solution of $\mathrm{D}(\boldsymbol{C})$, our method estimates the feasibility of each node in $\Gamma(1; \boldsymbol{C}, t)$, and constructs $\hat{W}_1^*(\boldsymbol{C}, t, \boldsymbol{x}_{init})$ satisfying

$$W_1^*(\boldsymbol{C}, t) \subseteq \hat{W}_1^*(\boldsymbol{C}, t, \boldsymbol{x}_{init}) \subseteq W(\boldsymbol{C}, t).$$

The definition of $\hat{W}_1^*(\boldsymbol{C}, t, \boldsymbol{x}_{init})$ is described as follows. To find feasible nodes $\boldsymbol{U}$ in $\Gamma(1; \boldsymbol{C}, t)$, we deal with the dual problem $\mathrm{D}(\boldsymbol{U})$ instead of $\mathrm{P}(\boldsymbol{U})$, and check the boundedness

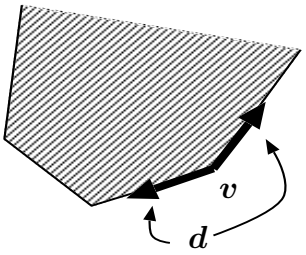

Figure 2: Direction vectors emanating from $\boldsymbol{v}$

of this problem. As stated in the previous subsection, we can obtain a feasible solution $\boldsymbol{x}_{init}$ of $\mathrm{D}(\boldsymbol{U})$ for any $\boldsymbol{U} \in \Gamma(1; \boldsymbol{C}, t)$ by (3) with the use of an optimal solution $\boldsymbol{x}_*$ of $\mathrm{D}(\boldsymbol{C})$. From this initial solution, we start the pivoting process of the simplex method to check the boundedness of $\mathrm{D}(\boldsymbol{U})$. The simplex method easily solves $\mathrm{D}(\boldsymbol{U})$ with $\boldsymbol{x}_{init}$ as the initial solution, because the structure of these problems $\mathrm{D}(\boldsymbol{C})$ and $\mathrm{D}(\boldsymbol{U})$ are similar to each other. Indeed, in numerical experiments, we see that the simplex method requires only a few iterations for $\mathrm{D}(\boldsymbol{U})$ starting from $\boldsymbol{x}_{init}$. Accordingly, we can expect that a feasible solution $\boldsymbol{x}_{init}$ of $\mathrm{D}(\boldsymbol{U})$ has an *unbounded direction* when this problem is unbounded. Thereby, we test whether the feasible solution $\boldsymbol{x}_{init}$ of $\mathrm{D}(\boldsymbol{U})$ has an unbounded direction instead of solving this problem by the simplex method. At (A) of Algorithm 1, we construct

$$\hat{W}_1^*(\boldsymbol{C}, t, \boldsymbol{x}_{init}) = \left\{ \bar{\boldsymbol{C}} \in \Gamma(1; \boldsymbol{C}, t) : \begin{array}{l} \text{there is no unbounded direction} \\ \text{emanating from } \boldsymbol{x}_{init} \text{ in } \mathrm{D}(\bar{\boldsymbol{C}}) \end{array} \right\}$$

We need to explain the phrase "unbounded direction emanating from $\boldsymbol{x}_{init}$ in $\mathrm{D}(\bar{\boldsymbol{C}})$" used in the definition of $\hat{W}_1^*(\boldsymbol{C}, t, \boldsymbol{x}_{init})$. Because a free variable $x_{\boldsymbol{b}}$ satisfying $-\infty < x_{\boldsymbol{b}} < +\infty$ on $\mathrm{D}(\boldsymbol{U})$ can be represented by $x_{\boldsymbol{b}} = x'_{\boldsymbol{b}} - x''_{\boldsymbol{b}}$ using nonnegative variables $x'_{\boldsymbol{b}}, x''_{\boldsymbol{b}}$, for simplicity, we consider the standard LP problem

$$\begin{array}{ll} \text{minimize} & \langle \boldsymbol{c}, \boldsymbol{x} \rangle \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{x} \geq \boldsymbol{0} \end{array}$$

where a coefficient matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, cost vector $\boldsymbol{c} \in \mathbb{R}^n$ and constant vector $\boldsymbol{b} \in \mathbb{R}^m$ are given, and $\boldsymbol{x} \in \mathbb{R}^n$ is a variable vector. Suppose that this problem is feasible, *i.e.* the feasible region, which forms a polytope, is nonempty. Then, for a vertex $\boldsymbol{v}$ of the polytope, the simplex method computes an adjacent vertex $\bar{\boldsymbol{v}} = \boldsymbol{v} + \theta \boldsymbol{d}$ $(\theta \geq 0)$ of $\boldsymbol{v}$ using a direction vector $\boldsymbol{d} \in \mathbb{R}^n$, which emanates from $\boldsymbol{v}$. Fig 2 illustrates the direction vectors incident to $\boldsymbol{v}$. A direction vector $\boldsymbol{d}$ emanating from $\boldsymbol{v}$ is said to be unbounded if we can increase the value of θ up to $+\infty$ while satisfying the following: (i) $\bar{\boldsymbol{v}} = \boldsymbol{v} + \theta \boldsymbol{d}$ satisfies all constraints, and (ii) the cost $\langle \boldsymbol{c}, \bar{\boldsymbol{v}} \rangle$ decreases from $\langle \boldsymbol{c}, \boldsymbol{v} \rangle$. If the LP problem is unbounded, it has an unbounded direction emanating from some vertex $\boldsymbol{v}$. In (A) of Algorithm 1, our dynamic enumeration method chooses an index t such that the size of $\hat{W}_1^*(\boldsymbol{C}, t, \boldsymbol{x}_{init})$ is the smallest among $t \in M \setminus L(\boldsymbol{C})$. Although we need a practical procedure to test whether $\boldsymbol{x}_{init}$ of $\mathrm{D}(\boldsymbol{U})$ has an unbounded direction, a detailed description can be found in [19, Section 3.2].

In (A) of Algorithm 1, our method uses $\hat{W}_1^*(\boldsymbol{C}, t, \boldsymbol{x}_{init})$ whereas Emiris and Canny's method uses $W^*(\boldsymbol{C}, t)$. There may be a difference between the sizes of $\hat{W}_1^*(\boldsymbol{C}, t, \boldsymbol{x}_{init})$ and

$W^*(\boldsymbol{C}, t)$ for some $t \in M \setminus L(\boldsymbol{C})$. However, the numerical results in Section 5 show that our method sufficiently reduces the computational effort for finding all mixed cells in a fine mixed subdivision.

4 Usage of DEMiCs

After unpacking the software package DEMiCs, the user will see SRC and polySys, which include source and sample files, in the main directory. The make file exists in the directory SRC. The executable file "demics" is generated by executing the following command in the directory.

```
make all
```

The input file requires information regarding the support of a polynomial system: the dimension of the system, the number of the distinct support sets, the cardinality, multiplicity and elements of each support set. For example, consider the support sets for a semi-mixed system of type $(2, 1, 1)$ as follows:

$$\begin{aligned}
\mathcal{S}_1 := \quad \mathcal{A}_1 = \mathcal{A}_2 = & \left\{ \begin{array}{l} (1,0,0,0),\ (0,1,0,0),\ (0,0,1,0),\ (0,0,0,1), \\ (0,0,0,0),\ (1,1,1,1) \end{array} \right\} \\
\mathcal{S}_2 := \qquad\quad \mathcal{A}_3 = & \left\{ \begin{array}{l} (2,0,0,0),\ (0,2,0,0),\ (0,0,2,0),\ (0,0,0,2), \\ (0,0,0,0),\ (2,2,2,2) \end{array} \right\} \\
\mathcal{S}_3 := \qquad\quad \mathcal{A}_4 = & \left\{ \begin{array}{l} (3,0,0,0),\ (0,3,0,0),\ (0,0,3,0),\ (0,0,0,3), \\ (0,0,0,0),\ (3,3,3,3) \end{array} \right\}.
\end{aligned}$$

The input file for $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ is written in the following format:

```
# The dimension or the number of variables
Dim = 4

# The number of the distinct support sets
Support = 3

# The number of elements in each support set
Elem = 6 6 6

# The multiplicity of each support set
Type = 2 1 1


# The elements of the 1st support set
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

```
0 0 0 0
1 1 1 1

# The elements of the 2nd support set
2 0 0 0
0 2 0 0
0 0 2 0
0 0 0 2
0 0 0 0
2 2 2 2

# The elements of the 3rd support set
3 0 0 0
0 3 0 0
0 0 3 0
0 0 0 3
0 0 0 0
3 3 3 3
```

The directory polySys also contains some sample files describing the support sets of several benchmark polynomial systems.

The above input file is placed in SRC as "poly.dat". To compute the mixed volume via a fine mixed subdivision, we simply execute

```
demics  poly.dat
```

in SRC, in which the executable file "demics" and the input file "poly.dat" exist. The software package then displays the total number of mixed cells, the value of the mixed volume and cpu time on the screen.

```
# Mixed Cells: 4
Mixed Volume: 24

CPU time: 0 s
```

Furthermore, we can select three options "-s", "-c" and "-cs" when running the program. Because the software package needs a seed number to generate a random number for each lifting value $\omega_i(\boldsymbol{a})$ ($\boldsymbol{a} \in \mathcal{S}_i$), the option "-s" is useful in the case where we change the seed number to generate different lifting values for each execution. If no option is selected as stated in the above, the seed number is automatically set to "1". As an example, when "6" is chosen as the seed number for the input data "poly.dat", we type the command

```
demics -s poly.dat 6
```

The option "-c" offers information about each mixed cell $\boldsymbol{C} = (C_i : i \in M) \in \Omega(M)$. After executing the command

```
demics -c poly.dat
```

the following information is displayed on the screen

```
---------------------------------
* Seed number = 1
* Lifting values for elements in each support set

S1   : 8.40188    3.94383    7.83099    7.9844     9.11647    1.97551
S2   : 3.35223    7.6823     2.77775    5.5397     4.77397    6.28871
S3   : 3.64784    5.13401    9.5223     9.16195    6.35712    7.17297
---------------------------------


# 1 : 1 : ( 2 3 4 ) 2 : ( 3 5 ) 3 : ( 5 1 )
Volume: 6

# 2 : 1 : ( 2 4 6 ) 3 : ( 5 1 ) 2 : ( 5 3 )
Volume: 6

# 3 : 1 : ( 2 4 6 ) 3 : ( 5 1 ) 2 : ( 5 4 )
Volume: 6

# 4 : 1 : ( 2 4 6 ) 3 : ( 4 5 ) 2 : ( 3 4 )
Volume: 6

# Mixed Cells: 4
Mixed Volume: 24

CPU time: 0s
```

The first line "Seed number = 1" indicates the value of the seed number. The lifting values, generated by the seed number, for elements in each support set are shown in the third to fifth lines. On the line with "# 1", "1 : (2 3 4)" means the subset $C_1 = \{\boldsymbol{a}_{12}, \boldsymbol{a}_{13}, \boldsymbol{a}_{14}\}$ of $\mathcal{S}_1$. That is, the number in front of a colon corresponds to the index of the support set. We thus know that one of mixed cells $\boldsymbol{C} = (C_1, C_2, C_3) \in \mathcal{S}_1 \times \mathcal{S}_2 \times \mathcal{S}_3$ consists of

$$C_1 = \{\boldsymbol{a}_{12}, \boldsymbol{a}_{13}, \boldsymbol{a}_{14}\}, \quad C_2 = \{\boldsymbol{a}_{23}, \boldsymbol{a}_{25}\} \quad \text{and} \quad C_3 = \{\boldsymbol{a}_{35}, \boldsymbol{a}_{31}\}$$

where $\boldsymbol{a}_{ij}$ is an element of $\mathcal{S}_i$. "Volume" on the next line represents the volume of the mixed cell $\boldsymbol{C} = (C_1, C_2, C_3)$. Note that the mixed volume is obtained from the summation of volumes of four mixed cells for the specific lifting values $\omega_i(\boldsymbol{a})$ $(\boldsymbol{a} \in \mathcal{S}_i)$. On a line with "#", the sequence of indices i for the subset C_i of each support set $\mathcal{S}_i$ indicates the order of an index t chosen from $M \setminus L(\boldsymbol{C})$ in Algorithm 1. For example, the line with "# 2" shows that the support sets $\mathcal{S}_1, \mathcal{S}_3$ and $\mathcal{S}_2$ are chosen in this order. Given the seed number "2", the "-cs" option is used as follows to get detailed information about mixed cells.

```
demics -cs poly.dat 2
```

5 Numerical results

This software package has been tested on a large variety of polynomial systems including unmixed, semi-mixed and fully mixed types. The papers [10, 12] report the superiority of MixedVol in computational time for these three types of the systems over the existing software packages: HOM4PS [11], MVLP [8], PHCpack [25], PHoM [22] and mvol [18]. Therefore, we compare DEMiCs with MixedVol for each type of polynomial systems in terms of the computational time. Note that MixedVol employs the static enumeration method, while DEMiCs adopts the dynamic enumeration method. All numerical experiments were executed on a 2.4GHz Opteron 850 with 8GB memory, running Linux.

First, we observe how the computational time of DEMiCs and MixedVol varies depending on m, which is the number of distinct support sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ of the semi-mixed polynomial systems $\boldsymbol{f}(\boldsymbol{x})$ in $\boldsymbol{x} \in \mathbb{R}^n$. In numerical experiments, we deal with artificial semi-mixed systems, which are created to investigate the feature of DEMiCs. Each support set $\mathcal{S}_i$ of the semi-mixed systems is given as follows. We choose the subset $\mathcal{T}$ of $\mathbb{Z}^n_+$ with $\#\mathcal{T} = 2n$ such as

$$\mathcal{T} = \left\{ \begin{array}{l} (1,0,\ldots,0,0),\ (0,1,\ldots,0,0),\cdots,(0,0,\ldots,1,0),(0,0,\ldots,0,0) \\ (1,0,\ldots,0,1),\ (0,1,\ldots,0,1),\cdots,(0,0,\ldots,1,1),(0,0,\ldots,0,1) \end{array} \right\}.$$

Note that the convex hull of $\mathcal{T}$ is the n-dimensional prism with a simplex basis, and the n-dimensional volume is $\frac{1}{(n-1)!}$. Let $\boldsymbol{e}_i$ denote the n-dimensional ith unit vector. For $\mathcal{T} \subseteq \mathbb{Z}^n_+$, we consider the transition of $\mathcal{T}$ by the direction vector $\boldsymbol{e}_i$ as $\mathcal{S}_i$. Namely,

$$\mathcal{S}_i := \boldsymbol{e}_i + \mathcal{T} = \{\boldsymbol{a} + \boldsymbol{e}_i : \boldsymbol{a} \in \mathcal{T}\} \quad \text{for each } i \in M.$$

Assume that each support set $\mathcal{S}_i$ has the multiplicity $n/m \in \mathbb{Z}$ for the dimension n and the number of distinct support sets m. That is, we deal with semi-mixed polynomial systems $\boldsymbol{f}(\boldsymbol{x})$ of type $(n/m, n/m, \ldots, n/m)$. Here, the mixed volume for the support $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m)$ of $\boldsymbol{f}(\boldsymbol{x})$ is calculated as $n! \times \frac{1}{(n-1)!} = n$ because the n-dimensional volume of $\mathrm{conv}(\mathcal{S}_i)$ and $\mathrm{conv}(\mathcal{T})$ is equal to each other.

To demonstrate the performance of DEMiCs and MixedVol, we choose two different values $n = 18, 24$ for the dimension of elements in $\mathcal{S}_i \subseteq \mathbb{Z}^n_+$, and change the number of distinct supports sets m in response to each dimension n. Table 1 and 2, which are in the case of $n = 18, 24$ respectively, summarize the cpu time of DEMiCs and MixedVol for each system. We performed 10 times numerical experiments for each system by choosing the different lifting values in DEMiCs and MixedVol. The cpu time listed in Table 1 and 2 is the average for each trial. The column "#Supp." means the number of distinct support sets m, and "Ratio" indicates the ratio between the cpu time of DEMiCs and MixedVol. The symbol "-" means that the software package has not been applied to the corresponding system. From these tables, we see that DEMiCs is superior to MixedVol in the computational time if the number of the distinct support sets is large. To the contrary, if the number of the distinct support sets is small, we may not expect the advantage of a dynamic enumeration method. One of the main reasons is that there is not great difference between the structure of the dynamic and static enumeration trees. Moreover, DEMiCs needs more computational tasks involved in choosing an index t from $M \setminus L(\boldsymbol{C})$ at Algorithm 1. Therefore, DEMiCs

Table 1: $n = 18$ (The mixed volume of all systems is 18)

# Supp. (m)	DEMiCs	MixedVol	Ratio
$m = 1$	0.052s	0.045s	0.87
$m = 3$	8.893s	4.969s	0.56
$m = 6$	8.715s	52.482s	6.02
$m = 9$	15.453s	3m40.422s	14.26
$m = 18$	1m7.927s	1h13m36.590s	65.02

Table 2: $n = 24$ (The mixed volume of all systems is 24)

# Supp. (m)	DEMiCs	MixedVol	Ratio
$m = 1$	0.599s	0.341s	0.57
$m = 3$	11m42.896s	5m32.361s	0.47
$m = 6$	4m20.044s	1h21m13.110s	18.74
$m = 8$	4m31.044s	4h3m41.700s	53.95
$m = 12$	9m9.827s	23h43m40.700s	155.36
$m = 24$	1h40m11.080s	-	

takes more computational time than MixedVol for semi-mixed systems with a few distinct supports.

Second, we consider the following benchmark polynomial systems, including unmixed, semi-mixed and fully mixed systems. The PRS-10 and RRS-12 systems, which are arising from kinematic problems in [21], have 12 polynomials with 12 variables and 11 polynomials with 11 variables, respectively. The PRS-10 system is a semi-mixed system of type $(1, 1, 1, 9)$, and the first three support sets have 4 elements, the last 100 elements. Also, the RRS-12 system is an unmixed system of type (11), and the support set has 224 elements. The cyclic-n[3], chandra-n[5] and katsura-n[4] systems are fully mixed systems, and size-expandable systems by the number n. The katsura-n systems consist of $(n+1)$ polynomials with $(n+1)$ variables, and the others n polynomials with n variables. The detailed description of the systems can be found in the web site [23]. We changed the lifting values 10 times for each system, and executed numerical experiments. In Table 3, we list the comparison of the average cpu time of DEMiCs and MixedVol. The column "$\mathcal{MV}$" presents the mixed volume for the support of corresponding systems, and "Ratio" is the ratio between the cpu time of these software packages. The numerical results for the cyclic-n, chandra-n and katsura-n systems in Table 3 show that DEMiCs improves the cpu time for finding all mixed cells dramatically when we address the polynomial systems with many distinct support sets. However, the numerical results on the PRS-10 and RRS-12 systems imply that it may be difficult for DEMiCs to deal with the unmixed and semi-mixed system which has only a few distinct support sets, compared with MixedVol.

Table 3: CPU time for the benchmark systems

System	Size (n)	$\mathcal{MV}$	DEMiCs	MixedVol	Ratio
PRS-10		142,814	14.3s	4.0s	0.28
RRS-12		226,512	29.9s	0.7s	0.02
Cyclic-n	$n = 12$	500,352	1m11.6s	4m43.0s	3.95
	$n = 13$	2,704,156	11m0.3s	49m57.4s	4.54
	$n = 14$	8,795,976	1h27m27.3s	7h14m24.1s	4.97
Chandra-n	$n = 17$	65,536	1m9.4s	33m13.4s	28.70
	$n = 18$	131,072	3m10.5s	2h14m15.3s	42.29
	$n = 19$	262,144	9m21.1s	8h19m6.3s	53.38
Katsura-n	$n = 12$	4,096	46.9s	14m3.5s	17.98
	$n = 13$	8,192	5m8.1s	1h21m19.4s	15.84
	$n = 14$	16,384	24m17.2s	7h54m29.4s	19.54

Table 4: A large size of cyclic-n, chandra-n and katsura-n systems

System	Size (n)	$\mathcal{MV}$	CPU time
Cyclic-n	$n = 15$	35,243,520	13h33m53.8s
	$n = 16$	135,555,072	110h21m40.5s
Chandra-n	$n = 20$	524,288	29m50.8s
	$n = 21$	1,048,576	1h15m19.7s
	$n = 22$	2,097,152	3h5m48.2s
	$n = 23$	4,194,304	11h24m4.6s
	$n = 24$	8,388,608	30h1m33.6s
Katsura-n	$n = 15$	32,768	1h30m54.7s
	$n = 16$	65,536	9h49m20.8s
	$n = 17$	131,072	46h37m35.8s

Finally, we consider the large-scale cyclic-n, chandra-n and katsura-n polynomial systems. Numerical experiments were carried out 5 times for each system associated with the different lifting values. Table 4 exhibits the average cpu time of DEMiCs for each system. As compared with numerical results in Table 2 of [19], the computational time of DEMiCs is less than that of the program developed in [19] as the size of the systems becomes larger. It could be due to improvement on how to use memory space in DEMiCs.

6 Concluding remarks

In this paper, we introduced the software package DEMiCs. Using the dynamic enumeration method, this package computes the mixed volume, which can be obtained from the volumes of all mixed cells, for the support of a semi-mixed polynomial system. The dynamic enumeration method, which was developed for a fully-mixed type in [19], can be extended to a semi-mixed type naturally. Numerical results show that this package significantly is faster than existing ones for large-scale semi-mixed systems with many distinct support sets. We confirm that it is important to take account of how we construct an enumeration tree for finding mixed cells. From numerical results, we recognize that DEMiCs needs more computational time than MixedVol for an unmixed system and a semi-mixed system with a few distinct support sets. It appears that the dynamic enumeration method does not have a beneficial effect on such systems, because the structure of the dynamic tree is almost the same as that of the static tree. Although we proposed one strategy for constructing an enumeration tree dynamically, there may exist other strategies, which can improve our dynamic enumeration. This is our future work. Finding mixed cells plays a crucial role in the polyhedral homotopy method. We expect that DEMiCs opens the way for computing all isolated zeros of large-scale polynomial systems by polyhedral homotopy method.

Acknowledgments

The author is grateful to the anonymous referees for their valuable comments.

References

[1] D. N. Bernshtein, *The number of roots of a system of equations*, Funct. Anal. Appl. **9**, (1975), pp. 183–185.

[2] U. Betke, *Mixed volumes of polytopes*, Archiv der Mathematik **58**, (1992), pp. 388–391.

[3] G. Björk and R.Fröberg, *A faster way to count the solutions of inhomogeneous systems of algebraic equations*, J. Symbolic Comput. **12**(3), (1991), pp. 329–336.

[4] W. Boege, R. Gebauer, and H. Kredel, *Some examples for solving systems of algebraic equations by calculating Groebner bases*, J. Symbolic Comput. **2**(1), (1986), pp. 83–98.

[5] S. Chandrasekhar, *Radiative Transfer*, Dover, NY, 1960.

[6] D. A. Cox, J. Little and D. O'Shea, *Using Algebraic Geometry*, Springer-Verlag, New York, 2nd edition, 2004.

[7] Y. Dai, S. Kim and M. Kojima, *Computing all nonsingular solutions of cyclic-n polynomial using polyhedral homotopy continuation methods*, J. Comput. Appl. Math. **152**, (2003), pp. 83–97.

[8] I. Z. Emiris and J. F. Canny, *Efficient incremental algorithms for the sparse resultant and the mixed volume*, J.Symbolic Comput. **20**(2), (1995), pp. 117–149. Software available at http://cgi.di.uoa.gr/~emiris/index-eng.html.

[9] T. Gao and T. Y. Li, *Mixed volume computation via linear programming*, Taiwanese J. Math. **4**, (2000), pp. 599–619.

[10] T. Gao and T. Y. Li, *Mixed volume computation for semi-mixed systems*, Discrete and Comput. Geom. **29**(2), (2003), pp. 257–277.

[11] T. Gao and T. Y. Li, The software package HOM4PS is available at http://www.mth.msu.edu/ li/.

[12] T. Gao, T. Y. Li and M. Wu, *Algorithm 846: MixedVol: A Software Package for Mixed Volume Computation*, ACM Trans. Math. Software **31**(4), (2005), pp. 555 – 560. Software available at http://www.csulb.edu/~tgao/.

[13] T. Gunji, S. Kim, M. Kojima, A. Takeda, K. Fujisawa and T. Mizutani, *PHoM – a Polyhedral Homotopy Continuation Method*. Computing **73**(1), (2004), pp. 57–77.

[14] T. Gunji, S. Kim, K. Fujisawa and M. Kojima, *PHoMpara – Parallel implementation of the polyhedral homotopy continuation method*, Computing **77**(4), (2006), pp. 387–411.

[15] B. Huber and B. Sturmfels, *A Polyhedral method for solving sparse polynomial systems*, Math. Comp. **64**, (1995), pp. 1541–1555.

[16] S. Kim and M. Kojima, *Numerical Stability of Path Tracing in Polyhedral Homotopy Continuation Methods*, Computing **73**, (2004), pp. 329–348.

[17] T. Y. Li, *Solving polynomial systems by polyhedral homotopies*, Taiwanese J. Math. **3**, (1999), pp. 251–279.

[18] T. Y. Li and X. Li, *Finding Mixed Cells in the Mixed Volume Computation*, Found. Comput. Math. **1**, (2001), pp. 161–181. Software available at http://www.math.msu.edu/~li/.

[19] T. Mizutani, A. Takeda and M. Kojima, *Dynamic Enumeration of All Mixed Cells*, Discrete Comput. Geom. **37**(3), (2007), pp. 351–367.

[20] A. Morgan, Solving polynomial systems using continuation for engineering and scientific problems, Pentice-Hall, New Jersey, 1987.

[21] H. -J. Su, J. M. McCarthy and L. T. Watson, *Generalized Linear Product Homotopy Algorithms and the Computation of Reachable Surfaces*, ASME J. Comput. Inf. Sci. Eng. **4**(3), (2004), pp. 226–234.

[22] A. Takeda, M. Kojima, and K. Fujisawa, *Enumeration of all solutions of a combinatorial linear inequality system arising from the polyhedral homotopy continuation method*, J. Oper. Soc. Japan **45**, (2002), pp. 64–82. Software available at http://www.is.titech.ac.jp/~kojima/index.html.

[23] J. Verschelde, The database of polynomial systems is in his web site: http://www.math.uic.edu/~jan/.

[24] J. Verschelde, P. Verlinden and R. Cools, *Homotopies exploiting Newton polytopes for solving sparse polynomial systems*, SIAM J. Numer. Anal. **31**, (1994), 915–930.

[25] J. Verschelde, *Algorithm 795: PHCPACK: A general-purpose solver for polynomial systems by homotopy continuation*, ACM Trans. Math. Softw. **25**, (1999), pp. 251–276. Software available at http://www.math.uic.edu/~jan/.