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

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

Abstract. DEMiCs is a software package written in C++ for computing the mixed volume, which is given as the total volume of all mixed cells, of the support of a general semi-mixed polynomial system via the dynamic enumeration method. The mixed cells play an essential role for computing all isolated zeros of a polynomial system by the 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, Kojima and Takeda for a fully-mixed polynomial system, is extended for a semi-mixed polynomial system, and is incorporated in the package. Numerical results exhibit that DEMiCs surpasses existing software packages in computational time especially 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

Recently, the polyhedral homotopy continuation method, proposed by Huber and Stumfels [13], has been established as a powerful and reliable numerical method [5, 11, 12, 14, 15, 23] 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 a variable vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{C}^n$. In this method, we require to find all mixed cells of the support of the polynomial system $\boldsymbol{f}(\boldsymbol{x})$ for construction of a family of homotopy functions. Using the mixed cells, we 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 is given as the total volume of all mixed cells, equals to the total number of solutions for the start systems. Also, it is an upper bound for the total number of isolated zeros in $(\mathbb{C} \setminus \{0\})^n$ of $\boldsymbol{f}(\boldsymbol{x})$, guaranteed by the Bernshtein's theorem [1].

The aim of this paper is to introduce the software package DEMiCs. The package computes all mixed cells and the mixed volume of the support of a general semi-mixed polynomial system, including a fully-mixed and full-unmixed as special cases, via the dynamic enumeration method which was developed for a fully-mixed system in [17]. Due to the method, DEMiCs improves the computational time considerably in enumeration of all mixed cells for large-scale semi-mixed systems with many distinct support sets over the existing software packages [6, 8, 9, 16, 23, 20], and opens the door to compute all zeros of such a large polynomial system by the polyhedral homotopy method.

Mixed cells are founded via an enumeration tree where each node is provided with a linear inequality system. The important properties of the enumeration tree are (i) a leaf node corresponds to a mixed cell if and only if the linear inequality system attached to the leaf node is feasible, and (ii) each node shares a common system with the child nodes, so that if the node is infeasible then so are all of its descendant nodes. The paper [17] refers to the importance of how we construct an enumeration tree, and presents the dynamic enumeration method for a fully-mixed system. In the existing static enumeration method [7, 8, 16, 20], the structure of an enumeration tree is fixed before construction of an enumeration tree, while in the dynamic enumeration method, we construct an enumeration tree dynamically so that many child nodes are infeasible and pruned when a node branches into child nodes. This paper explains how we extend the dynamic enumeration method, developed for a fully-mixed system in [17], to a semi-mixed polynomial system.

There are several related software packages for computing the mixed volume via enumeration of all mixed cells: HOM4PS [10], MixedVol [8, 9], MVLP [6], PHCpack [23], PHoM [20] and mvol [16]. HOM4PS, PHCpack and PHoM, written in FORTRAN, Ada and C++ respectively, are software packages for computing all isolated zeros of a polynomial system by the polyhedral homotopy method. These packages contain the module for finding mixed cells. In particular, PHCpack is the most popular software package among these software packages. The C++ package MixedVol, which employs the static enumeration method, specializes in the mixed volume computation via enumeration of all mixed cells. MixedVol surpasses all other software packages in terms of computational efficiency and a memory requirement, as reported in the papers [8, 9]. A feature of the package is that all mixed cells can be generated efficiently for semi-mixed polynomial systems by taking account of the special structure of a semi-mixed type.

Numerical results show that DEMiCs can drastically reduce the computational time for finding all mixed cells compared to the existing software packages [6, 8, 9, 16, 23, 20] not only for fully-mixed polynomial systems but also for the semi-mixed polynomial systems. Although we confirmed that DEMiCs surpasses the existing packages for a large-scale fully-mixed system and semi-mixed system with many distinct support sets, DEMiCs does not always excel. Indeed, for fully-unmixed systems and semi-mixed systems with a few distinct supports, DEMiCs needs more computational time than MixedVol [9]. It is due to overhead of computation associated with the dynamic branching rule.

This paper is organized as follows. In Section 2 and 3, technical details of our method are described. We outline the dynamic enumeration method for a general semi-mixed polynomial system in Section 2. Section 3 explains how we check the feasibility of each node and how we construct an enumeration tree when a polynomial system is a semi-mixed type. In Section 4, the usage of DEMiCs is described. We report the performance of DEMiCs in comparison with the existing software packages using artificial semi-mixed polynomial systems, and well-known large-scale benchmark systems in Section 5. 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 $\boldsymbol{a} = (a_1, a_2 \ldots, a_n) \in \mathbb{Z}^n_+$. The support of $f_i(\boldsymbol{x})$ indicates $\mathcal{A}_i$, and the support of $\boldsymbol{f}(\boldsymbol{x})$ does $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n)$. Let $N := \{1, 2, \ldots, n\}$. For the support $\mathcal{A} = (\mathcal{A}_i : i \in N)$ of a polynomial system $\boldsymbol{f}(\boldsymbol{x})$, some support sets may be equal to each other. Suppose that the polynomial system has m ($\leq n$) distinct support sets $\mathcal{S}_i$ among $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$. Define $M = \{1, 2, \ldots, m\}$. Then, we use the notation $\mathcal{S}_i$ ($i \in M$) for each distinct support 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 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 $i_1, i_2 \in M$. The polynomial system with the support $\mathcal{S} = (\mathcal{S}_i : i \in M)$ is called a *semi-mixed* system. Especially, the system is called a *fully-unmixed* type when $m = 1$, and a *fully-mixed* when $m = n$. Also, we call the polynomial system with the support $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m)$ a *semi-mixed system of type* $(k_1, k_2, \ldots, k_m)$ when $\#I_i = k_i$ for each $i \in M$. In this paper, we

deal with a semi-mixed polynomial system $\boldsymbol{f}(\boldsymbol{x})$ of type $(k_1, k_2, \ldots, k_m)$ with the 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.

We review the definition of the mixed volume of the support $\mathcal{S} = (\mathcal{S}_i : i \in M)$ of a polynomial system, and how to compute it via enumeration of mixed cells. Let $Q_i = \mathrm{conv}(\mathcal{S}_i)$ denote the convex hull of the support set $\mathcal{S}_i$. For all positive number $\lambda_1, \lambda_2, \ldots, \lambda_m$, we consider the n-dimensional volume of the Minkowski sum

$$\lambda_1 Q_1 + \lambda_2 Q_2 + \cdots + \lambda_m Q_m = \{\lambda_1 q_1 + \lambda_2 q_2 + \cdots + \lambda_m q_m : q_i \in Q_i,\ i \in M\}.$$

This volume is given by a homogeneous polynomial of degree n in λ_i ($i \in M$). The mixed volume of $\mathcal{S}$ is defined by the coefficient of $\lambda_1 \lambda_2 \cdots \lambda_m$ in the polynomial.

Huber and Sturmfels [13] introduced a *fine mixed subdivision* of the Minkowski sum $Q = Q_1 + Q_2 + \cdots + Q_m$ to compute the mixed volume of $\mathcal{S}$, and presented how to construct the subdivision. Each piece polytope in the subdivision contributes construction of a family of homotopy functions of the polyhedral homotopy method. See more details of a fine mixed subdivision in [13]. We call a piece polytope R_j a *cell* in a fine mixed subdivision of Q. It is known that each cell R_j in a fine mixed subdivision is represented as the Minkowski sum $\mathrm{conv}(C_1^j) + \mathrm{conv}(C_2^j) + \cdots + \mathrm{conv}(C_m^j)$ for $\boldsymbol{C} = (C_1^j, C_2^j, \ldots, C_m^j)$ with $C_i^j \subseteq \mathcal{S}_i$. Especially, when a polynomial system $\boldsymbol{f}(\boldsymbol{x})$ is a semi-mixed system of type $(k_1, k_2, \ldots, k_m)$, we call a cell R_j, which is described as the 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 [13] that the mixed volume of the support $\mathcal{S}$ of the system $\boldsymbol{f}(\boldsymbol{x})$ is given by the summation of the volume of all mixed cells in a fine mixed subdivision. A fine mixed subdivision of Q is constructed by applying a lifting function $\omega_i : \mathcal{S}_i \to \mathbb{R}$ whose image value is a real number chosen from $\mathbb{R}$ generically. 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 $\hat{Q}$ denote the Minkowski sum $\hat{Q} = \hat{Q}_1 + \hat{Q}_2 + \cdots + \hat{Q}_m$ for $\hat{Q}_i = \mathrm{conv}(\hat{\mathcal{S}}_i)$. Also we will use the notation $\hat{\boldsymbol{C}} = (\hat{C}_1, \hat{C}_2, \ldots, \hat{C}_m)$ for the subset $\hat{C}_i$ of $\hat{\mathcal{S}}_i$. The projection $\mathbb{R}^{n+1} \to \mathbb{R}^n$ of the set of lower facets of $\hat{Q}$ gives a fine mixed subdivision of Q.

Li and Li [16] proposed an efficient algorithm for finding lower facets of $\hat{Q}$ via an enumeration tree. Recently, for a fully-mixed polynomial system, the paper [17] improved their algorithm by replacing a static enumeration tree of [16] 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 fully-unmixed type.

2.2 Algorithm

We overview a *dynamic enumeration algorithm* in [17] and apply this algorithm to a semi-mixed polynomial system. Let $L \subseteq M$. For any semi-mixed system of type $(k_1, k_2, \ldots, k_m)$, 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 is equipped with the root node $\emptyset^m \in \Omega(\emptyset) := \{\emptyset^m\}$ and the leaf nodes $\Omega(M) \subset \Omega$. A node at the ℓth level is corresponding 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 $\boldsymbol{C} \in \Omega(L)$ $(L \subseteq M)$. A node $\boldsymbol{C} = (C_i : i \in L) \in \Omega(L)$ with $L \subseteq M$ is provided with the linear inequality system $\mathcal{I}(\boldsymbol{C})$:

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

where

$$\hat{\boldsymbol{\alpha}} = \left(\begin{array}{c} \boldsymbol{\alpha} \\ 1 \end{array} \right) \in \mathbb{R}^{n+1}.$$

Here, $\langle \cdot, \cdot \rangle$ stands for the usual inner product in the Euclidean space. Li and Li showed in [16] 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 [17]. 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. Accordingly, 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 [7, 8, 16, 20], 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 [17] 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 $L \subsetneq M$ in order to reduce the cost for finding all feasible leaf nodes.

The purpose of the tree representation is to reduce a computational task at the feasibility check for all leaf nodes. The feasible region of the linear inequality system $\mathcal{I}(\boldsymbol{C})$ attached to a node $\boldsymbol{C}$ contains that of $\mathcal{I}(\bar{\boldsymbol{C}})$ 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 is no mixed cells in the subtree. Therefore, by replacing $W(\boldsymbol{C}, t)$ at Algorithm 2.1 of [17] 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 as stated in below. We will use the notation A_ℓ ($\ell \in \{0\} \cup M$) for the set of feasible nodes.

Algorithm 2.1. (The 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

When this algorithm is terminated, A_m contains all nodes in Ω^*, *i.e,* every mixed cell can be stored in A_m. We introduce this algorithm, which employs the breadth-first order, for simplicity of the description, though it is essentially the same as Algorithm 2.2 described in [17], which utilizes the 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 [17], to save a memory requirement.

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

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

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

As for (a), in the *static enumeration method* proposed in the previous works [7, 8, 16, 20], 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. In this paper, we employ the *dynamic enumeration method*, which is developed for a fully-mixed system in the paper [17]. Naturally, this method can be applied to a semi-mixed system. 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 $t \in M \setminus L(\boldsymbol{C})$. However, this cost is too expensive. Therefore, instead of $W^*(\boldsymbol{C}, t)$, we consider another set $\hat{W}(\boldsymbol{C}, t, \boldsymbol{x}_{init})$ which can be constructed easily by the use of $\boldsymbol{x}_{init}$ generated from a feasible solution of $\mathcal{I}(\boldsymbol{C})$. The dynamic enumeration method computes $\hat{W}(\boldsymbol{C}, t, \boldsymbol{x}_{init})$ satisfying

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

and chooses the index t which attains the minimum size of $\hat{W}(\boldsymbol{C}, t, \boldsymbol{x}_{init})$ for all $t \in M \setminus L(\boldsymbol{C})$. In Subsection 3.3, we refer to how to construct this set. The *relation table* proposed in [8] is useful to find infeasible nodes in $W(\boldsymbol{C}, t)$. The software package DEMiCs removes infeasible nodes from $W(\boldsymbol{C}, t)$ by the use of the relation table before construction of $\hat{W}(\boldsymbol{C}, t, \boldsymbol{x}_{init})$.

As for (b), 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 the computational task. 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. In this subsection, 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 2.1. Then, we would like to construct $W^*(\boldsymbol{C}, t) \subseteq W(\boldsymbol{C}, t)$. For a nonnegative integer k, let

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

This set $\Gamma(k_t + 1; \boldsymbol{C}, t)$ coincides with $W(\boldsymbol{C}, t)$ clearly.

For a feasible node $\boldsymbol{C} \in A_\ell$ with $\ell < m$ in the dynamic enumeration algorithm, we build a tree T for construction of $W^*(\boldsymbol{C}, t)$. 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 a 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 $\boldsymbol{U} \in \Gamma(k; \boldsymbol{C}, t)$ is equipped with the 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, if we check the feasibility of any node $\boldsymbol{U} \in \Gamma(k_t + 1; \boldsymbol{C}, t)$ which corresponds to each leaf node at the $(k_t + 1)$the level, $W^*(\boldsymbol{C}, t)$ can be obtained.

We describe a tree structure for construction of $W^*(\boldsymbol{C}, t)$ more precisely. For $U_t \subseteq \mathcal{S}_t$, we choose a function $m_t : \mathcal{S}_t \to \mathbb{Z}$ which answers 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 $T = (V, E)$ denote 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

$$Z(\boldsymbol{U}; \boldsymbol{C}, t) = \left\{ \bar{\boldsymbol{U}} \in \Gamma(\#U_t + 1; \boldsymbol{C}, t) : \ \bar{U}_t = U_t \cup \{\boldsymbol{a}_i\}, \quad m_t(U_t) < i \leq r_t \ \right\}$$

and construct a tree $T = (V, E)$ such as $V = \bigcup_{k \in K_t} V_k$ and $E = \bigcup_{k \in K_t} E_k$, based on the following algorithm:

Algorithm 3.1. (Construction of a tree $T = (V, E)$)

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

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

$V_0 \leftarrow \boldsymbol{C}$, $E_0 \leftarrow \emptyset$ 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 $\boldsymbol{U} \in V_k$: ($\natural$) **do**
    if $m_t(U_t) \leq k_t + 1$ **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

The root node of the tree T, constructed by the algorithm, corresponds to $V_0 = \{\boldsymbol{C}\}$, which is identical to A_ℓ generated by Algorithm 2.1. When this algorithm is terminated, V_k stores every element in $\Gamma(k; \boldsymbol{C}, t)$ for any $k \in K_t$. Hence, $W(\boldsymbol{C}, t)$, generated by a feasible $\boldsymbol{C} \in A_\ell$ and an index $t \in M \setminus L(\boldsymbol{C})$ at Algorithm 2.1, is equal to V_{k_t+1}, which has each leaf node at the $(k_t + 1)$th level of the tree T.

For a node $\boldsymbol{U} \in V_k$ and the 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 a 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 ($\natural$) of Algorithm 3.1 by V_k^* such that

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

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 the building of $T = (V, E)$, according to the following search procedure, 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, $W^*(\boldsymbol{C}, t)$ can be obtained as $V_{k_t+1}^*$. We introduce Algorithm 3.1 using the breadth-first order for the simplicity of the description. However, the software package DEMiCs adopts the depth-first order for construction of $W^*(\boldsymbol{C}, t)$ to save a memory requirement.

3.2 Formulation for 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 3.1 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^*. For the feasibility check of a node $\boldsymbol{U} \in V_k$, a linear programming (LP) problem can be formulated. Let $\boldsymbol{\gamma}$ denote a specific n-dimensional real vector. Using this $\boldsymbol{\gamma} \in \mathbb{R}^n$, we choose $\hat{\boldsymbol{\gamma}} \in \mathbb{R}^{n+1}$ such as $\hat{\boldsymbol{\gamma}}^T = (\boldsymbol{\gamma}^T, 0)$. To determine whether a node $\boldsymbol{U} \in V_k$ is feasible or not, we solve the following problem in a variable vector $\hat{\boldsymbol{\alpha}} \in \mathbb{R}^{n+1}$:

$$\mathrm{P}(\boldsymbol{U}) : \ \Big| \ \text{max.} \quad \langle \hat{\boldsymbol{\gamma}}, \hat{\boldsymbol{\alpha}} \rangle \quad \text{s. t.} \quad \mathcal{I}(\boldsymbol{U}).$$

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

$$\mathrm{D}(\boldsymbol{U}) : \left| \begin{array}{lll} \text{min.} & \Phi(\boldsymbol{x}; \boldsymbol{U}) & \\ \text{s. t.} & \Psi(\boldsymbol{x}; \boldsymbol{U}) = \boldsymbol{\gamma}, & \\ & -\infty < x_{\boldsymbol{b}} < +\infty & \boldsymbol{b} \in U_i \setminus \{\boldsymbol{a}_i\} \\ & x_{\boldsymbol{b}'} \geq 0 & \boldsymbol{b}' \in \mathcal{S}_i \setminus U_i \quad (i \in L(\boldsymbol{U})). \end{array} \right.$$

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 mentions 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 3.1, and $\boldsymbol{C} \in V_0$ serves as 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 $\boldsymbol{U} \in V_k$. We 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 of a dual problem $\mathrm{D}(\boldsymbol{U})$ for any $\boldsymbol{U} \in V_k$ by the use of 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 have 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 $\boldsymbol{U}$. Then, 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 the size of problems, the dual problems are superior to the primal ones as stated in [17]. Therefore, we employ the dual problems to check the feasibility of a nod and solve these problems using the simplex method.

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

In Algorithm 2.1, 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. Using an optimal solution of $\mathrm{D}(\boldsymbol{C})$ for a feasible node $\boldsymbol{C} \in A_\ell$, we estimate the number of feasible child nodes in $W(\boldsymbol{C}, t)$ associated with a choice of an index $t \in M \setminus L(\boldsymbol{C})$.

Suppose that $\boldsymbol{C}$ is a feasible node in A_ℓ with $\ell < m$ and an index t is chosen from $M \setminus L(\boldsymbol{C})$. As stated in the previous subsection, we can easily obtain a feasible solution of $\mathrm{D}(\boldsymbol{U})$ for any $\boldsymbol{U} \in \Gamma(1; \boldsymbol{C}, t)$ using an optimal solution of $\mathrm{D}(\boldsymbol{C})$. That is, if $\boldsymbol{x}_{init}$ is set up as (3) using an optimal solution $\boldsymbol{x}_*$ of $\mathrm{D}(\boldsymbol{C})$, $\boldsymbol{x}_{init}$ is a feasible solution of $\mathrm{D}(\boldsymbol{U})$ for any $\boldsymbol{U} \in \Gamma(1; \boldsymbol{C}, t)$. Note that in each iteration, the simplex method updates a feasible solution $\boldsymbol{x}$ as $\bar{\boldsymbol{x}} = \boldsymbol{x} + \theta \boldsymbol{d}$ using a nonnegative scalar θ and a direction vector $\boldsymbol{d}$ to which $\boldsymbol{x}$ is incident on the polytope of a feasible region. Especially, $\boldsymbol{d}$ is called an *unbounded direction* if we can choose any nonnegative θ without violating the feasibility of a problem. The simplex methods usually require only a few iterations for solving $\mathrm{D}(\boldsymbol{U})$ with $\boldsymbol{U} \in \Gamma(1; \boldsymbol{C}, t)$ if using $\boldsymbol{x}_{init}$ as the initial feasible solution, because the structure of these problems $\mathrm{D}(\boldsymbol{C})$ and $\mathrm{D}(\boldsymbol{U})$ are similar to each other. From this observation, we can expect that a feasible solution $\boldsymbol{x}_{init}$ of $\mathrm{D}(\boldsymbol{U})$ is incident to 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 2.1, we construct

$$\hat{W}(\boldsymbol{C}, t, \boldsymbol{x}_{init}) = \{\bar{\boldsymbol{C}} \in \Omega(L(\boldsymbol{C}) \cup \{t\}) : \bar{\boldsymbol{C}}_{L(\boldsymbol{C})} = \boldsymbol{C}_{L(\boldsymbol{C})} \text{ and } \bar{C}_t \subseteq \hat{\mathcal{A}}(\boldsymbol{C}, t, \boldsymbol{x}_{init})\},$$

where

$$\hat{\mathcal{A}}(\boldsymbol{C}, t, \boldsymbol{x}_{init}) = \left\{ \boldsymbol{a} \in \mathcal{S}_t : \begin{array}{l} \text{there is no unbounded direction emanating from} \\ \boldsymbol{x}_{init} \text{ of } \mathrm{D}(\boldsymbol{U}) \text{ where } \boldsymbol{U} \in \Gamma(1; \boldsymbol{C}, t) \text{ and } U_t = \{\boldsymbol{a}\} \end{array} \right\}.$$

Then, we choose an index t such that the size of $\hat{W}(\boldsymbol{C}, t, \boldsymbol{x}_{init})$ is the smallest among $t \in M \setminus L(\boldsymbol{C})$, using $\boldsymbol{x}_{init}$ generated from an optimal solution of $\mathrm{D}(\boldsymbol{C})$ with $\boldsymbol{C} \in A_\ell$. To check whether $\boldsymbol{x}_{init}$ of $\mathrm{D}(\boldsymbol{U})$ has an unbounded direction, we need to compute the direction vectors and reduced costs with respect to $\boldsymbol{x}_{init}$. In Section 3.2 of [17], we can see the detailed description of how to get the direction vectors and reduced costs. There may be a gap between the size of $\hat{W}(\boldsymbol{C}, t, \boldsymbol{x}_{init})$ and $W^*(\boldsymbol{C}, t)$ for any $t \in M \setminus L(\boldsymbol{C})$. However, the numerical results in Section 5 show that this evaluation reduces the computational task sufficiently for finding all mixed cells in a fine mixed subdivision.

4 Usage of DEMiCs

After unpacking the software package DEMiCs, we can see SRC and polySys, which include source and sample files, respectively, in the main directory. A make file is found in the directory SRC. When we carry out the command in the directory

```
make all
```

the executable file "demics" is generated.

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, we consider the support sets for a semi-mixed system of type $(2, 1, 1)$ as follows:

$$\begin{array}{rrcl} \mathcal{S}_1 := & \mathcal{A}_1 = \mathcal{A}_2 & = & \{(1,0,0,0),\ (0,1,0,0),\ (0,0,1,0),\ (0,0,0,1),\ (0,0,0,0),\ (1,1,1,1)\} \\ \mathcal{S}_2 := & \mathcal{A}_3 & = & \{(2,0,0,0),\ (0,2,0,0),\ (0,0,2,0),\ (0,0,0,2),\ (0,0,0,0),\ (2,2,2,2)\} \\ \mathcal{S}_3 := & \mathcal{A}_4 & = & \{(3,0,0,0),\ (0,3,0,0),\ (0,0,3,0),\ (0,0,0,3),\ (0,0,0,0),\ (3,3,3,3)\}. \end{array}$$

Then, 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
a1.1 = 1 0 0 0
a1.2 = 0 1 0 0
a1.3 = 0 0 1 0
a1.4 = 0 0 0 1
a1.5 = 0 0 0 0
a1.6 = 1 1 1 1

# The elements of the 2nd support set
a2.1 = 2 0 0 0
a2.2 = 0 2 0 0
a2.3 = 0 0 2 0
a2.4 = 0 0 0 2
a2.5 = 0 0 0 0
a2.6 = 2 2 2 2

# The elements of the 3rd support set
a3.1 = 3 0 0 0
a3.2 = 0 3 0 0
a3.3 = 0 0 3 0
a3.4 = 0 0 0 3
a3.5 = 0 0 0 0
a3.6 = 3 3 3 3
```

Also, in the directory polySys, you can find some sample files in which the support sets of several benchmark polynomial systems are described.

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

```
demics  poly.dat
```

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

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

CPU time: 0 s
```

Furthermore, we can select three options "-c", "-s" and "-cs" when running the program. The option "-c" offers information about each mixed cell $\boldsymbol{C} = (C_i : i \in M) \in \Omega(M)$. After the execution of the command

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

the following information is displayed on a screen

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

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

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

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

# Mixed Cells: 4
Mixed Volume: 24

CPU time: 0.01 s
```

Therefore, we can understand that the mixed volume is given by the summation of volumes of four mixed cells for the specific lifting values $\omega_i(\boldsymbol{a})$ $(\boldsymbol{a} \in \mathcal{S}_i)$. On the first line with "# 1", "1 : (1 2 6)" means the subset $C_1 = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_6\}$ of $\mathcal{S}_1$. That is, the number in front of a colon is corresponding to the index of a support set. We thus know that one of mixed cells $\boldsymbol{C} = (C_1, C_2, C_3)$ consists of

$$C_1 = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_6\} \subseteq \mathcal{S}_1, \quad C_2 = \{\boldsymbol{a}_1, \boldsymbol{a}_5\} \subseteq \mathcal{S}_2 \quad \text{and } C_3 = \{\boldsymbol{a}_5, \boldsymbol{a}_3\} \subseteq \mathcal{S}_3.$$

"Volume" on the next line represents the volume of the mixed cell $\boldsymbol{C} = (C_1, C_2, C_3)$. 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 $M \setminus L(\boldsymbol{C})$ at Algorithm 2.1. For example, the line with "# 3" shows that support sets $\mathcal{S}_1, \mathcal{S}_3$ and $\mathcal{S}_2$ are chosen in this order at the algorithm.

This software package needs the seed number to generate a random number for each lifting value $\omega_i(\boldsymbol{a})$ $(\boldsymbol{a} \in \mathcal{S}_i)$. If no option is selected as stated in the above, the seed number is set to "1" automatically. The option "-s" is useful in the case where we change the seed number to generate different lifting values for each execution. 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
```

Also, when we get detailed information about mixed cells, and take the seed number as "2", the option "-cs" is used as follows:

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

5 Numerical results

This software package has been tested on a large variety of polynomial systems including fully-unmixed, semi-mixed and fully-mixed types. The papers [8, 9] report the superiority of MixedVol in computational time for these three types of the systems over the existing software packages: HOM4PS [10], MVLP [6], PHCpack [23], PHoM [20] and mvol [16]. 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

$$\mathcal{T} = \left\{ \begin{array}{l} (1,0,0,\ldots,0,0),\ (0,1,0,\ldots,0,0),\cdots,(0,0,0,\ldots,1,0),\ (0,0,0,\ldots,0,0) \\ (1,0,0,\ldots,0,1),\ (0,1,0,\ldots,0,1),\cdots,(0,0,0,\ldots,1,1),\ (0,0,0,\ldots,0,1) \end{array} \right\} \subseteq \mathbb{Z}^n_+.$$

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 regard 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 of 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 "Speed-up 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 2.1.

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

# Supp. (m)	DEMiCs	MixedVol	Speed-up 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	Speed-up 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	-	

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

Second, we consider the following benchmark polynomial systems, including fully-unmixed, semi-mixed and fully-mixed systems. The PRS-10 and RRS-12 systems, which are arising from kinematic problems in [19], 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 a fully-unmixed system of type (11), and the support set has 224 elements. The cyclic-n[2], chandra-n[4] and katsura-n[3] 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 cite [21]. 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 "Mixed volume" presents the mixed volume of the support of corresponding systems, and "Speed-up 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 fully-unmixed and

semi-mixed system which has only a few distinct support sets, compared with MixedVol.

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 [17], the computational time of DEMiCs is less than that of the program developed in [17] 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 is given as the total volume of all mixed cells, of the support of a semi-mixed polynomial system. The dynamic enumeration method, which was developed for a fully-mixed type in [17], can be extended to a semi-mixed type naturally. Numerical results show that this package improves the computational time sufficiently over the existing software packages, and can generate all mixed cells for a large-scale semi-mixed system with many distinct support sets. We confirm that it is important to take account of how we construct an enumeration tree for enumeration of mixed cells. From numerical results, we recognize that DEMiCs needs more computational time than MixedVol for a fully-unmixed system and a semi-mixed system with a few distinct support sets. It appears that dynamic enumeration method does not have a beneficial effect on such a system, because the structure of the dynamic tree is almost the same as that of the static tree. 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 the polyhedral homotopy method.

References

[1] D. N. Bernshtein, "The number of roots of a system of equations," *Functional Analysis and Appl.* **9**, 183–185 (1975).

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

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

[4] S. Chandrasekhar, "Radiative Transfer," Dover, NY, 1960.

[5] Y. Dai, S. Kim and M. Kojima, "Computing all nonsingular solutions of cyclic-n polynomial using polyhedral homotopy continuation methods", *J. Computational and Applied Mathematics* **152**, 83-97 (2003).

Table 3: CPU time for the benchmark systems

System	Size (n)	Mixed volume	DEMiCs	MixedVol	Speed-up 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)	Mixed volume	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

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

[7] T. Gao and T. Y. Li, "Mixed volume computation via linear programming," *Taiwan Journal of Mathematics* **4**, 599–619 (2000).

[8] T. Gao and T. Y. Li, "Mixed volume computation for semi-mixed systems," *Discrete and Computational Geometry* **29**(2), 257–277 (2003).

[9] T. Gao, T. Y. Li and M. Wu, "MixedVol: A Software Package for Mixed Volume Computation," *ACM Transactions on Math. Software* **31**(4), (2005). Software available at http://www.csulb.edu/~tgao/.

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

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

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

[13] B. Huber and B. Sturmfels, "A Polyhedral method for solving sparse polynomial systems," *Mathematics of Computation* **64**, 1541–1555 (1995).

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

[15] T. Y. Li, "Solving polynomial systems by polyhedral homotopies", *Taiwan Journal of Mathematics* **3**, 251–279 (1999).

[16] T. Y. Li and X. Li, "Finding Mixed Cells in the Mixed Volume Computation," *Foundation of Computational Mathematics* **1**, 161–181 (2001). Software available at http://www.math.msu.edu/~li/.

[17] T. Mizutani, A. Takeda and M. Kojima "Dynamic Enumeration of All Mixed Cells," *Discrete and Computational Geometry* **37**(3), 351–367 (2007).

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

[19] H. -J. Su, J. M. McCarthy and L. T. Watson, "Generalized Linear Product Polynomial Continuation and the Computation of Reachable Surfaces," *ASME Journal of Information and Computer Sciences in Engineering* **4**(3), 226–234 (2004).

[20] 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. of Operations Society of Japan* **45**, 64–82 (2002). Software available at http://www.is.titech.ac.jp/~kojima/index.html.

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

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

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