

# **Finding All Mixed Cells for Polyhedral Homotopies**

Tomohiko MIZUTANI

Submitted in partial fulfillments of  
the requirement for the degree of  
DOCTOR OF SCIENCE

Department of Mathematical and Computing Sciences  
Tokyo Institute of Technology

June 2008

# Acknowledgment

I would like to express my sincerely gratitude to my adviser Prof. Masakazu Kojima. He introduced me to homotopy methods for solving a polynomial system and in addition, provided a great research environment. A lot of his advice encouraged me to continue to study. I am very thankful for his warm encouragement.

I am largely indebted to Dr. Akiko Takeda for valuable suggestions and comments on my research. Without her help, my dissertation would not attain completion. Many thanks to Prof. Antoine Deza of McMaster University from whom I have learned combinatorics of polytopes during my undergraduate and graduate course. Special thanks to Dr. Makoto Yamashita of Kanagawa University for giving me various knowledge and skills in parallel computing. His support was absolutely necessary when I utilize a parallel computing system. I wish to thank Dr. Mitsuhiro Fukuda for his support in my research and Prof. Sunyoung Kim of Ewha Women's University for her kindness. Through discussions with her for homotopy methods, my English communication skill has been enhanced.

I would like to express my deep thanks to Prof. Tien-Yien Li of Michigan State University for providing the opportunity to study under his supervision. His advice was very beneficial so that it deepens my understanding of solving a polynomial system by polyhedral homotopies. Also, his lecture about numerical polynomial algebra stimulated my research interests. The six month stay at his place was wonderful experience. During the stay, I have been supported by many friends. In particular, I am very grateful to Mr. Chih-Hsiung Tsai who will soon take a Ph.D degree in this May and Dr. Tsung-Lin Lee. I have discussed many things about polyhedral homotopy methods with them.

Finally, I thank all members of operation research group, Kojima lab., Takahashi lab., Mase lab., Miyoshi lab., and Shimodaira lab., for their friendship. I have had a very good time with them.

Tomohiko Mizutani  
February 2008

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Mixed Volume, Mixed Cells and Polyhedral Homotopy Methods</b>	<b>7</b>
2.1	Introduction . . . . .	7
2.2	Polynomial System and Homotopy Methods . . . . .	8
2.3	Mixed Volume . . . . .	10
2.4	Mixed Cells . . . . .	12
2.4.1	Relation between Mixed Volume and Mixed Cells . . . . .	13
2.4.2	Construction of Fine Mixed Subdivision . . . . .	15
2.5	Solving Polynomial Systems through Polyhedral Homotopies . . . . .	17
2.5.1	Polyhedral Homotopies . . . . .	17
2.5.2	Polyhedral-Linear Homotopies . . . . .	20
<b>3</b>	<b>Finding All Mixed Cells in a Fine Mixed Subdivision</b>	<b>24</b>
3.1	Introduction . . . . .	24
3.2	Issue and Related Work . . . . .	26
3.2.1	Discussion on Some Issues in Polyhedral Homotopy Methods . . . . .	26
3.2.2	Related Work and Software . . . . .	30
3.3	Enumeration Tree . . . . .	32
3.3.1	Tree Construction . . . . .	32
3.3.2	Effect of Tree Construction Order . . . . .	34
3.4	Dynamic Enumeration . . . . .	35
3.4.1	Fully Mixed Type . . . . .	36
3.4.2	Semi-Mixed Type . . . . .	38
3.5	Feasibility Check . . . . .	41

<b>4</b>	<b>Efficient Implementation of Dynamic Enumeration</b>	<b>44</b>
4.1	Introduction . . . . .	44
4.2	Implementation Issues . . . . .	45
4.2.1	Lift-Prune Method by Emiris and Canny . . . . .	45
4.2.2	Our Strategy . . . . .	48
4.2.3	Comparison of the Size of Static and Dynamic Trees . . . . .	52
4.3	Numerical Results on Single Processor . . . . .	54
4.3.1	Fully Mixed Type . . . . .	55
4.3.2	Semi-Mixed Type . . . . .	58
<b>5</b>	<b>Parallel Implementation of Dynamic Enumeration</b>	<b>61</b>
5.1	Introduction . . . . .	61
5.2	Parallel Computation for Finding All Mixed Cells . . . . .	61
5.2.1	Background of Parallel Computation . . . . .	62
5.2.2	Parallelization of Dynamic Enumeration . . . . .	63
5.3	Numerical Results on Multiple Processors . . . . .	67
5.3.1	Large Scale Polynomial Systems . . . . .	67
5.3.2	Scalability . . . . .	69
<b>6</b>	<b>Conclusions and Future Directions</b>	<b>75</b>
	<b>Bibliography</b>	<b>80</b>

# Chapter 1

## Introduction

Solving a system of polynomial equations is a typical mathematical problem and occurs in various fields of science and engineering; for example, computation of equilibrium states in economics [54, 14], motion planning and inverse kinematics of robot manipulators [65, 55, 52]. The symbolic methods, in most cases, based on the Buchberger algorithm for constructing a Gröbner base, have been known as one of the standard approaches to the problem. However, the methods seem to be restricted to relatively small polynomial systems because the symbolic manipulation makes these inefficient. On the other hand, the homotopy (continuation) methods are a numerical method for computing all isolated zeros of the system of  $n$  polynomials with  $n$  unknowns. In the European Community Project FRISCO [19] for three years from March 1996, the investigation of the ability of the algorithms for polynomial system solving and development of polynomial system solvers were carried out in order to help industrial users to settle practical issues as stated above. To quote the report published by the project, the textbook [53] by Stetter, who introduced the so-called eigenvalue method for the purpose, suggests that solving large-scale polynomial systems is still hard problem.

The polyhedral homotopy, which is one of the varieties of homotopy methods, appeared in 1995 as a promising method to overcome the hardness. It has been enhanced by many researchers over the past decade and now is known as a powerful and reliable numerical method for computing all isolated zeros of a system of  $n$  polynomials

$$\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_n(\mathbf{x}))$$

in an  $n$ -dimensional complex variables  $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{C}^n$ . The main idea of the homotopy methods is to define a homotopy as a function  $\mathbf{h}(\mathbf{x}, t) \in \mathbb{C}^n \times [0, 1]$  such that the start polynomial system  $\mathbf{h}(\mathbf{x}, 0)$  is easily computable, and the target polynomial system  $\mathbf{h}(\mathbf{x}, 1)$  coincides with the original system  $\mathbf{f}(\mathbf{x})$ . For all fixed  $t \in [0, 1)$ , the system  $\mathbf{h}(\mathbf{x}, t) = \mathbf{0}$  has only nonsingular solutions, and hence, each connected component of the solutions of  $\mathbf{h}(\mathbf{x}, t) = \mathbf{0}$  forms a smooth solution curve, called a homotopy path. Every isolated zero of the system  $\mathbf{f}$  can be found by following the homotopy path starting from each point provided as the solution of the start polynomial system  $\mathbf{h}(\mathbf{x}, 0) = \mathbf{0}$ . Thus, the homotopy

methods are regarded to consist of two stages: the construction of a homotopy function and the path following. While all zeros of  $\mathbf{f}$  are reachable from the starting points  $\mathbf{h}(\mathbf{x}, 0) = \mathbf{0}$  through the homotopy paths, some paths may not converge to the zeros of the system  $\mathbf{f}$  but diverge to infinity. In general, the total number of the homotopy paths defined by a homotopy function  $\mathbf{h}$  is larger than that of the isolated zeros of a polynomial system  $\mathbf{f}$ . It means that among all homotopy paths, there exist some redundant ones.

The beginning of quest for the homotopy methods for polynomial system solving is the works by Garcia and Zangwill [24] and Drexler [15] in the late of 1970's. They made a suggestion that a homotopy can be used for this purpose. The number of paths, generated from their classical homotopies based on the Bézout theorem, is bounded by the total degree of a polynomial system  $\mathbf{f} = (f_1, f_2, \dots, f_n)$ , which is obtained by the multiplication of each degree of polynomials  $f_1, f_2, \dots, f_n$ . Usually, it is large for the number of the zeros of a polynomial system  $\mathbf{f}$ , and in particular it becomes marked as the size of a polynomial system grows. In consequence, for relatively large-scale polynomial systems, the classical homotopy methods force us to follow many redundant paths, which never reach to the zeros of  $\mathbf{f}$ , and spend a lot of wasted time. Therefore, the classical methods have a limitation in solving such large-scale polynomial systems. The paper [29] presents an easy example to recognize it. Consider the eigenvalue problem  $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$  for a generic matrix  $\mathbf{A} \in \mathbb{C}^{n \times n}$ . This problem is considered as a system of  $n + 1$  quadratic polynomial equations with  $n + 1$  variables;  $\sum_{j=1}^n a_{ij}x_j - \lambda x_j = 0$  ( $i = 1, 2, \dots, n$ ) and  $\sum_{j=1}^n x_j^2 = 1$ . Thus, the total degree of the polynomial system is  $2^{n+1}$ , whereas this problem has  $2n$  solutions because each eigenspace intersects the unit ball in two points.

In the middle of 1990's, Huber and Sturmfels [29] and Verschelde et al. [59] made a computational breakthrough in a homotopy method by virtue of Bernshtein's theorem [3]. This theorem guarantees that the mixed volume for  $n$  Newton polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$  of polynomials  $f_1, f_2, \dots, f_n$  of  $\mathbf{f}$  provides an upper bound for the number of isolated zeros of the polynomial system  $\mathbf{f} = (f_1, f_2, \dots, f_n)$  in  $(\mathbb{C} \setminus \{0\})^n$ , and the bound is tight if the polynomial system  $\mathbf{f}$  is in general position. We may need an explanation for some terminologies used in the above theorem. For a polynomial system  $\mathbf{f} = (f_1, f_2, \dots, f_n)$ , the set  $\mathcal{A}_i$  of exponents of all monomials in  $f_i$  is called the support of  $f_i$ . The Newton polytope  $\mathcal{P}_i$  of  $f_i$  is defined as the convex hull of  $\mathcal{A}_i$ . The mixed volume is defined for  $n$  polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$ , and is a generalization of the volume of a polytope. Huber and Sturmfels [29] proposed a family of polyhedral homotopy functions founded on Bernshtein's theorem so that the total number of the paths defined by polyhedral homotopies coincides with the mixed volume for  $n$  Newton polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$  of polynomials  $f_i$ . It is said that the polyhedral homotopies produce much fewer paths than the classical ones based on the Bézout theorem. Indeed, for the above eigenvalue problem, the mixed volume for the quadratic polynomials is  $2n$ .

Although it raises hopes for improvements in computational costs for solving polynomial systems, we need to solve an enumeration problem, which will be mentioned in the below, to construct a family of polyhedral homotopy functions. As the size of polynomial systems

becomes larger, the required computational cost for solving it increases dramatically, and the problem becomes intractable. Namely, it is the main computational bottleneck in the polyhedral homotopy under such a situation.

The final goal of our research is to develop a method for solving large-scale polynomial systems and to provide a useful tool for implementing it. In this thesis, we attempt to answer the question;

*“How do we construct polyhedral homotopies for large-scale polynomial systems?”*

because the answers to the question could be expected to become a major step toward the goal. In Chapter 2, the question is posed, and we quest for the answers in Chapter 3, 4 and 5. Finally, we summarize the achievements to answer the question, and discuss the future directions of the research in Chapter 6.

The most of Chapter 2 is devoted to the explanation of the construction of a family of the polyhedral homotopy functions and the method to solve a polynomial system using it. The construction of polyhedral homotopies utilizes the geometric property of the mixed volume for  $n$  Newton polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$  of polynomials  $f_i$  of  $\mathbf{f}$  such that it is obtained from the volumes of certain piece polytopes in a fine mixed subdivision, which is a polyhedral subdivision satisfying more strict conditions, of the Minkowski sum  $\mathcal{P}$  of  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$ . We call such a piece polytope a mixed cell for  $n$  Newton polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$ . The precise definition of a mixed cell and a fine mixed subdivision will be shown in Subsection 2.4.1. Each polyhedral homotopy function  $\mathbf{h}$  is one-to-one corresponding to the mixed cell in a fine mixed subdivision of the Minkowski sum  $\mathcal{P} = \mathcal{P}_1 + \mathcal{P}_2 + \dots + \mathcal{P}_n$ . The feature of polyhedral homotopies  $\mathbf{h}(\mathbf{x}, t)$  is that the number of all solutions of a start system  $\mathbf{h}(\mathbf{x}, 0) = \mathbf{0}$  is provided as the volume of the corresponding mixed cell. Thus, in the polyhedral homotopy method, the number of the paths coincides with the mixed volume for  $n$  Newton polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$ .

A fine mixed subdivision of  $\mathcal{P}$ , which is the Minkowski sum of  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$ , can be obtained by lifting  $\mathcal{P}_i$  into higher dimensional space. For  $n$  polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$  in an  $n$ -dimensional real space, we lift a polytope  $\mathcal{P}_i$  to  $\hat{\mathcal{P}}_i$  in an  $(n+1)$ -dimensional real space using a generic lifting function, and consider its Minkowski sum  $\hat{\mathcal{P}} = \hat{\mathcal{P}}_1 + \hat{\mathcal{P}}_2 + \dots + \hat{\mathcal{P}}_n \subseteq \mathbb{R}^{n+1}$ , called a lifted polytope for convenience. Then, the projection of the lower envelope of  $\hat{\mathcal{P}}$  in  $\mathbb{R}^{n+1}$  on the original space  $\mathbb{R}^n$  forms a fine mixed subdivision of  $\mathcal{P} = \mathcal{P}_1 + \mathcal{P}_2 + \dots + \mathcal{P}_n \subseteq \mathbb{R}^n$ . Accordingly, through the enumeration of the corresponding lower facets of the lifted polytope  $\hat{\mathcal{P}}$ , we can find all mixed cells for  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$  (that is, a family of polyhedral homotopy functions). This is a typical problem related to a face enumeration in a polytope. By the use of geometric terminologies, the motivated question in this thesis can be restated as follows; how do we enumerate every lower facet, which corresponds to a mixed cell, of a lifted polytope  $\hat{\mathcal{P}}$  induced from  $n$  Newton polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$  of polynomials  $f_i$  of  $\mathbf{f}$  in a short time? In the first section of Chapter 3, the question will be clarified by showing some numerical results obtained from the existing works for the polyhedral homotopy method.

There exist several formulations [17, 61, 38, 57, 22] for finding all mixed cells through the lower facets of a lifted polytope. Among the existing ones, it is considered that a formulation by Li and Li [38] in 2001 is more practical from the perspective of computational efficiency and memory requirement. It will be shown in Chapter 2. In the formulation, an enumeration tree is constructed among a family of systems of linear inequalities, which are induced from the system of linear inequalities with a combinatorial condition describing all mixed cells. The main properties of the enumeration tree are summarized as follows.

- A leaf node corresponds to a mixed cell if and only if the linear inequality system attached to the leaf node is feasible.
- Each node shares a common linear inequality system with the child nodes, so that if it is infeasible then so are all of its descendant nodes.

We then apply an enumeration method for finding all feasible leaf nodes; if a node is determined to be infeasible then so are their descendant nodes; hence the subtree having the node as a root can be pruned because it does not contain any mixed cell.

The crucial issue in the efficient implementation of enumerating mixed cells is how we construct enumeration trees. It is the main concern in Chapter 3 through the first half of Chapter 4. Enumeration trees need to satisfy the above two properties. Specifically, the first property determines the collection of leaf nodes. Here, one has a lot of freedom in choosing and allocating systems of linear inequalities, which are induced from the system of linear inequalities with a combinatorial condition describing the mixed cells, for intermediate level nodes. In the existing works [20, 22, 38, 57], the structure of enumeration trees is determined and fixed before enumerating mixed cells. In such a static construction of an enumeration tree, any information obtained at a node during the execution of the enumeration is not utilized at all to branch the node into its child nodes because a branching rule is fixed in advance. For numerical efficiency, however, it is ideal to branch the node into its child nodes so that a larger portion of its child nodes are infeasible and are pruned. To pursue this idea, in contrast to the existing static enumeration methods, we propose dynamic enumeration where branching at a node is carried out with the effective use of information which is generated from “child LP problems” at the node. This is the first proposal to answer the question. When we utilize information obtained at a node for its child nodes, the dual simplex method has an advantage; hence the dual simplex method plays an essential role in our dynamic enumeration.

The second proposal to answer the question is the extension of our dynamic enumeration so that it can utilize the structure of the support of a polynomial system in order to increase computational efficiency. We sometimes encounter the situation that a polynomial system  $\mathbf{f} = (f_1, f_2, \dots, f_n)$  arising from real-world problems, for instance, mechanism design [65, 55], has a special structure in the supports  $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$  of polynomials  $f_1, f_2, \dots, f_n$  such that some supports  $\mathcal{A}_i$  among these ones is equal to each other. Such a polynomial system is called semi-mixed, and in particular unmixed if all the supports are equal, whereas it is called

fully mixed if all the supports are distinct. In the paper [29], they proved that only distinct supports contribute the mixed volume computation. It implies that we can make the size of an enumeration tree much smaller because the height of tree is determined by the number of the distinct supports  $\mathcal{A}_i$  of a polynomial system  $\mathbf{f}$ . So it is important for computational efficiency in the enumeration of all mixed cells to take account of the support structure for semi-mixed polynomial systems. In Chapter 3, we develop the dynamic enumeration method so that it achieves the sufficient speed up by utilizing the structure of the supports if a polynomial system is a semi-mixed type.

The third proposal to answer the question is the parallel implementation of our dynamic enumeration. Homotopy methods are well-adapted to a parallel computation. This is a remarkable advantage in solving large-scale polynomial systems. In the path following stage of the methods, it is easy to conceive the idea that tracking one homotopy path has nothing to do with tracking others, and thus, the path following is carried out in parallel. Similarly, the construction stage of homotopies in the polyhedral homotopy method can be parallelized. Takeda et al. [57] proposed a parallel method for implementing the static enumeration, and the numerical results in the paper showed that it attains high performance in terms of scalability. The method makes use of the tree property that it is partitioned into a family of the subtrees, which do not share any node with each other. In the first half of Chapter 5, by taking into account the property, our dynamic enumeration is implemented in parallel, so that we could expect that the parallel implementation finds all mixed cells for the polynomial systems never before possible.

In the second half of Chapter 4 and 5, by the numerical results, we show the performance of the serial and parallel implementation of our dynamic enumeration, respectively. Compared with the software package MixedVol [23], which is the implementation of the static enumeration method by Gao and Li [22] and known as the fastest software among the other existing one [17, 61, 38, 57, 22], our dynamic method implemented on a single processor has been tested for the performance evaluation through wide range problems from the theoretical objects such as Fourier analysis [7] in mathematics and Ising model [8, 31] in physics to the practical applications such as design of four-bar linkages for passing nine points on the coupler curve [65, 52] in kinematics and analysis of neural network [47] in artificial intelligence. These test problems contain all types of a polynomial system: fully mixed, semi-mixed and unmixed. Some of them can be found in the test suite of FRISCO [19].

For fully mixed polynomial systems, for example, noon- $n$  problems [47] whose number  $n$  indicates the number of polynomials (or variables), our dynamic method generated all mixed cells for the noon-18 problem and computed the mixed volume in 6 minutes, while MixedVol solves the same problem in 8 hours 23 minutes. Amazingly, the speed-up ratio reaches to near 75. Furthermore, our method could solve the problems up to  $n = 22$  within 24 hours. Among these test polynomial systems, the cyclic- $n$  problem is considered to be notoriously difficult one. For the cyclic-14 problem which had been the largest one in the cyclic- $n$  problems to be solved by the existing methods, our method took 1 hours

27 minutes for the generation of all mixed cells and computation of the mixed volume by contrast to 7 hours 14 minutes of MixedVol. To conduct the investigation for semi-mixed and unmixed polynomial systems, we prepared artificial semi-mixed systems. The numerical results impose us to recognize that although our dynamic method has a speed advantage over MixedVol for semi-mixed systems with many distinct supports, it does not always excel. Indeed, when a semi-mixed polynomial system has only a few distinct supports, our method is slower than MixedVol because there is not major difference between the tree structure built by the dynamic and static methods.

As well as the serial implementation, the parallel implementation of our dynamic method shows good performance in terms of scalability and encourages us to challenge large-scale polynomial systems. For example, the parallel method implemented on 64 processors made it possible to compute all mixed cells and the mixed volume for the cyclic-15 problem 49 times faster than the serial method, and also succeeded to solve the cyclic-16 and cyclic-17 problems in 3 hours 42 minutes and 32 hours 31 minutes, respectively. Certainly, the obtained value of the mixed volume for the cyclic-17 problem, 601,080,390, satisfies the theoretical result proven by Haagerup, as mentioned in [50], that if  $n$  is prime, then the number of solutions of the cyclic- $n$  system is finite, and it equals to  $\frac{(2n-2)!}{(n-1)!^2}$ .

Our answers for the motivated question are summarized as follows:

- In contrast to the existing static enumeration methods, we proposed the dynamic enumeration for finding all mixed cells, and showed by the numerical results that our dynamic enumeration improves the computational time dramatically over the existing methods, and makes it possible to find all mixed cells and compute the mixed volume for the Newton polytopes of some polynomial systems, which had not been solved.
- We extended the dynamic enumeration method so that it can take advantage of the special structure of a support if it is a semi-mixed type. For semi-mixed polynomial systems with many distinct supports, our dynamic method performed better than the existing methods in computational time.
- The software package DEMiCs was developed for the serial implementation of our dynamic enumeration and published at the website [42]. It could be expected to become a useful tool to help researches to carry out the investigation further.
- Our dynamic enumeration method fits a parallel computation very nicely. The numerical results convinced us that the parallel implementation is so powerful and attains good scalability. Indeed, it brought new results for the computation of the mixed cells and mixed volume for the Newton polytopes of some polynomial systems.

# Chapter 2

## Mixed Volume, Mixed Cells and Polyhedral Homotopy Methods

### 2.1 Introduction

In the first four sections of this chapter, we explain the method to compute all isolated zeros of a system of  $n$  polynomials  $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_n(\mathbf{x}))$  in  $\mathbf{x} \in \mathbb{C}^n$  by polyhedral homotopies, and discuss the performance in the last section. In Section 2.2, after the introduction of some symbols on polynomials and their supports which are used in this thesis, we review classical methods whose homotopy is designed on the Bézout theorem. One of the difficulties for solving the large-scale polynomial systems by the classical homotopy methods is the existence of many redundant homotopy paths because it makes us spend a lot of wasted time.

Taking account of Bernshtein's theorem, the polyhedral homotopy method can be expected to conquer it. The theorem tells us that the mixed volumes for  $n$  Newton polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$  of polynomials  $f_1, f_2, \dots, f_n$  of  $\mathbf{f}$  provides an upper bound for the total number of zeros of  $\mathbf{f} = (f_1, f_2, \dots, f_n)$  in  $(\mathbb{C} \setminus \{0\})^n$ . Generally, the bound is much tighter than the classical one given by the Bézout theorem. In the polyhedral homotopy method, the construction of a family of homotopy functions is founded on Bernshtein's theorem so that the number of all homotopy paths defined by polyhedral homotopies coincides with the mixed volume for  $n$  Newton polytopes  $\mathcal{P}_i$  of polynomials  $f_i$  of  $\mathbf{f}$ . In Section 2.3, we define the mixed volume for Newton polytopes, and then summarize some properties of it. The precise description of Bernshtein's theorem is shown at the end of the section.

A family of polyhedral homotopy functions is constructed by utilizing the geometric property such that the mixed volume for  $n$  Newton polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$  of polynomials  $f_1, f_2, \dots, f_n$  of  $\mathbf{f}$  is obtained by summing up the volumes of certain piece polytopes in a fine mixed subdivision of the Minkowski sum  $\mathcal{P}$  of  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$ . We call such a piece polytope a mixed cell. Section 2.4 is composed of two subsections. In the first subsection, we introduce the method for subdivision of  $\mathcal{P} = \mathcal{P}_1 + \mathcal{P}_2 + \dots + \mathcal{P}_n$  for computing the mixed

volume for  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$ , and refer to a reason why the mixed volume can be obtained from the total volume of all mixed cells in the subdivision of  $\mathcal{P}$ . In the second subsection, we see that finding all mixed cells in the subdivision of  $\mathcal{P}$  can be reduced to a problem related to a face enumeration in a polytope. An overall figure that illustrates solving a polynomial system through polyhedral homotopies is presented in Section 2.5.

## 2.2 Polynomial System and Homotopy Methods

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

$$f_i(\mathbf{x}) = \sum_{\mathbf{a} \in \mathcal{A}_i} c_i(\mathbf{a}) \mathbf{x}^{\mathbf{a}} \quad \text{for each } i \in N \quad (2.1)$$

using a nonempty finite subset  $\mathcal{A}_i$  of  $\mathbb{Z}_+^n$  and nonzero  $c_i(\mathbf{a}) \in \mathbb{C}$  for  $\mathbf{a} \in \mathcal{A}_i$ . Here, we define  $N = \{1, 2, \dots, n\}$ . Also,  $\mathbb{Z}_+^n$  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  $\mathbf{x}^{\mathbf{a}} = x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$  for  $\mathbf{a} = (a_1, a_2, \dots, a_n) \in \mathbb{Z}_+^n$ . We call the set  $\mathcal{A}_i$  the *support of  $f_i$* , and  $n$ -tuple supports  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n)$  the *support of  $\mathbf{f}$* . The support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n)$  of  $\mathbf{f}$  is sometimes written as  $\mathcal{A} = (\mathcal{A}_i : i \in N)$  for simplicity of notation. Let  $r_i$  denote the cardinality of each support  $\mathcal{A}_i$ . Some supports  $\mathcal{A}_i$  of  $f_i$  may be equal to each other. We suppose that the polynomial system has exactly  $m$  ( $\leq n$ ) distinct support sets among  $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$  such that

$$\mathcal{A}_j := \mathcal{A}_{j_1} = \mathcal{A}_{j_2} = \cdots = \mathcal{A}_{j_{k_j}} \quad \text{for each } j \in M, \quad (2.2)$$

where we define  $M = \{1, 2, \dots, m\}$ . Obviously,  $\sum_{j \in M} k_j = n$ . A polynomial system with the support  $\mathcal{A} = (\mathcal{A}_j : j \in M)$  is called *semi-mixed*. In particular, it is called *unmixed* when  $m = 1$ , and *fully mixed* when  $m = n$ . By a  $(k_1, k_2, \dots, k_m)$ -support, we mean the support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  of a semi-mixed system if and only if  $\mathcal{A}_j$  occurs exactly  $k_j$  times in  $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$ .

A homotopy is constructed between a start polynomial system  $\mathbf{g}(\mathbf{x})$  and a target polynomial system  $\mathbf{f}(\mathbf{x})$  in order to compute all isolated zeros of  $\mathbf{f}(\mathbf{x})$  in  $\mathbf{x} \in \mathbb{C}^n$ . Li suggests in [37, 39] that a homotopy should be defined as a function  $\mathbf{h}(\mathbf{x}, t) : \mathbb{C}^n \times [0, 1] \rightarrow \mathbb{C}^n$  such that

- (a) The solutions of the start system  $\mathbf{g}(\mathbf{x}) := \mathbf{h}(\mathbf{x}, 0) = \mathbf{0}$  are easily computable.
- (b) The connected component of the solution curves  $\{(\mathbf{x}, t) \in \mathbb{C}^n \times [0, 1] : \mathbf{h}(\mathbf{x}, t) = \mathbf{0}\}$  forms a smooth path in the interval  $t \in [0, 1)$ , called a *homotopy path*.
- (c) Every isolated solution of  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$  can be reached by some homotopy path starting from a solution of the start system  $\mathbf{g}(\mathbf{x}) = \mathbf{h}(\mathbf{x}, 0) = \mathbf{0}$ .

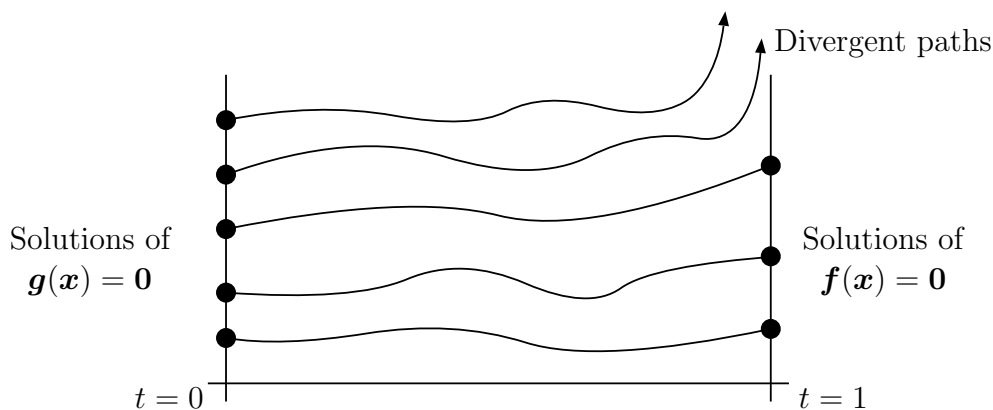


Figure 2.1: Illustration of homotopy paths

The important property of a homotopy, satisfying the above three conditions, is that some homotopy paths may diverge to infinity as the parameter  $t$  approaches to 1 although every solution of the original system  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$  can be reachable starting from solutions of  $\mathbf{g}(\mathbf{x}) = \mathbf{0}$  through the homotopy paths. One never predict whether a path is convergent to the solution of the original system or divergent to infinity as  $t \rightarrow 1$ . In Figure 2.1, the illustration shows that three paths connect to the solutions of  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ , whereas two paths diverge to infinity.

In the early developmental stage of homotopy methods, we choose a trivial polynomial system

$$\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}), \dots, g_n(\mathbf{x})) \quad \text{in } \mathbf{x} \in \mathbb{C}^n$$

as the start system, and a homotopy is defined for a polynomial system  $\mathbf{f} = (f_1, f_2, \dots, f_n)$  as a function  $\mathbf{h}$  of the form

$$\mathbf{h}(\mathbf{x}, t) = (1 - t)\mathbf{g}(\mathbf{x}) + t\mathbf{f}(\mathbf{x}) \quad \text{for } (\mathbf{x}, t) \in \mathbb{C}^n \times [0, 1].$$

Let  $d_i$  be the degree of  $f_i$ . As the typical choice of the system  $\mathbf{g} = (g_1, g_2, \dots, g_n)$  which satisfies the above three conditions, the classical homotopy methods based on the Bézout theorem employ

$$g_i(\mathbf{x}) = a_i x_i^{d_i} - b_i \quad \text{for each } i \in N,$$

where  $a_i, b_i$  are chosen at random in  $\mathbb{C}$ . For the degrees  $d_i$  of polynomials  $f_i$ , we write the multiplication  $d_1 d_2 \cdots d_n$  by  $\mathcal{D}$ , and it is called a *total degree* of the polynomial system  $\mathbf{f}$ . In this choice, the system  $\mathbf{g}(\mathbf{x}) = \mathbf{0}$  has  $\mathcal{D}$  trivial solutions, and all solutions of the original system  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$  can be obtained as a result of tracking  $\mathcal{D}$  homotopy paths. Accordingly, we know that the total degree  $\mathcal{D}$  of a polynomial system  $\mathbf{f}$  serves as the upper bound for the total number of the solutions of  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ . Hence, it is sometime called a *Bézout bound*.

However, we frequently meet the situation that the polynomial systems  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$  arising from practical applications have fewer than  $\mathcal{D}$  solutions. Unfortunately, it means that the classical homotopy methods generates many redundant paths for such polynomial systems. Therefore, the classical methods waste a lot of time in tracking paths, which never reach

the solutions of  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ . It is one of the biggest issues in solving large-scale polynomial systems by the classical methods. The polyhedral homotopy method improves it by utilizing Bernshtein's theorem, which provides much tighter bound for the number of isolated solutions of  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ . It implies that the method produces much fewer paths, compared to the classical ones, and as a result, we can expect that it solves the large-scale polynomial systems.

The fundamental technique for following homotopy paths is predictor-corrector procedures. Although there are some variations in the procedures, the basic idea is that the next point is predicted by easy but inaccurate means, and then, the inaccurate point is corrected so as to return to the homotopy path, using the Newton method. The detail description is found in, for example, [39, 51, 52].

## 2.3 Mixed Volume

We start with the definition of the mixed volume, and next summarize some properties of it. Let  $\mathcal{P}_i$  be the convex hull of the support  $\mathcal{A}_i$  of  $f_i$ .  $\mathcal{P}_i$  is a polytope in  $\mathbb{R}^n$ , and in particular called the *Newton polytope* of  $f_i$ . The mixed volume is a generalization of the volume of a polytope. For the definition of the mixed volume, we define the ‘‘addition’’ for polytopes, which is called the *Minkowski sum* of polytopes, and the ‘‘scalar multiplication’’ for a polytope. The addition (Minkowski sum) for each polytope  $\mathcal{P}$  and  $\mathcal{Q}$  is defined by

$$\mathcal{P} + \mathcal{Q} = \{p + q : p \in \mathcal{P} \text{ and } q \in \mathcal{Q}\},$$

and scalar multiplication of a polytope  $\mathcal{P}$  by a nonnegative number  $\lambda$  is

$$\lambda\mathcal{P} = \{\lambda p : p \in \mathcal{P}\}.$$

Furthermore, it is shown that  $\lambda\mathcal{P} + \mu\mathcal{Q}$  is a polytope in  $\mathbb{R}^n$  for each nonnegative number  $\lambda, \mu$ . For a polytope  $\mathcal{P}$  in  $\mathbb{R}^n$ , we denote the  $n$ -dimensional volume  $\text{Vol}_n(\mathcal{P})$  for which we have

$$\text{Vol}_n(\mathcal{P}) = \int_{\mathcal{P}} 1 \cdot dx_1 dx_2 \cdots dx_n.$$

The following is the definition of the mixed volume for  $n$  Newton polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$  in  $\mathbb{R}^n$  arisen from the support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n)$  of  $\mathbf{f}(\mathbf{x})$  in  $\mathbf{x} \in \mathbb{C}^n$ . As is well known (see, for example, [12, Proposion 4.9]), for any nonnegative number  $\lambda_1, \lambda_2, \dots, \lambda_n$ , the  $n$ -dimensional volume of  $\lambda_1\mathcal{P}_1 + \lambda_2\mathcal{P}_2 + \cdots + \lambda_n\mathcal{P}_n$  is given by a homogeneous polynomial of degree  $n$  in  $\lambda_i$  ( $i \in N$ ). The mixed volume for  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$  is defined to be the coefficient of  $\lambda_1\lambda_2 \cdots \lambda_n$  in the polynomial.

**Definition 2.3.1.** The mixed volume for  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$ , denoted  $\text{MV}_n(\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n)$ , is a coefficient of the monomial  $\lambda_1\lambda_2 \cdots \lambda_n$  in a homogeneous polynomial  $\text{Vol}_n(\lambda_1\mathcal{P}_1 + \lambda_2\mathcal{P}_2 + \cdots + \lambda_n\mathcal{P}_n)$  of degree  $n$ .

Originally, the mixed volume is defined for more general convex sets. However, we use a restricted description for the definition because the property of the mixed volume is utilized in the context of the polyhedral homotopy method.

We introduce some properties of the mixed volume for  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$ , which will be needed in the successive discussions. One may want to refer to the other properties; see for instance [12]. The following proposition points out the relation between the mixed volume and the volume of polytopes.

**Proposition 2.3.2.** ([12]) *Let  $\mathcal{P}_i$  ( $i \in N$ ) be a polytope in  $\mathbb{R}^n$ .*

$$(a) \quad MV_n(\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n) = \sum_{i=1}^n (-1)^{n-k} \sum_{\substack{I \subseteq N \\ |I|=k}} Vol_n(\sum_{i \in I} \mathcal{P}_i).$$

$$(b) \quad \text{If } \mathcal{P}_i = \mathcal{P} \text{ for every } i \in N, \text{ then } MV_n(\mathcal{P}, \mathcal{P}, \dots, \mathcal{P}) = n! Vol_n(\mathcal{P}).$$

This proposition can be shown by the fact that the  $n$ -dimensional volume of  $\sum_{i \in N} \lambda_i \mathcal{P}_i$  for nonnegative numbers  $\lambda_i$  ( $i \in N$ ) is a homogeneous polynomial of degree  $n$  in  $\lambda_i$ . We can confirm easily that (a) and (b) holds when  $n = 2$ . Let  $\mathcal{P}_1, \mathcal{P}_2$  be polytopes in  $\mathbb{R}^2$ . From the above description, we know that  $Vol_2(\lambda_1 \mathcal{P}_1 + \lambda_2 \mathcal{P}_2)$  ( $\lambda_1, \lambda_2 \geq 0$ ) is represented as  $a\lambda_1^2 + b\lambda_2^2 + c\lambda_1\lambda_2$  for some  $a, b, c \in \mathbb{R}$ . To substitute  $(1, 0)$ ,  $(0, 1)$  and  $(1, 1)$  for a variable  $(\lambda_1, \lambda_2)$  of the homogeneous polynomial, the linear equation system with respect to variables  $a, b$  and  $c$  can be constructed. To solve the equation, we get  $MV_2(\mathcal{P}_1, \mathcal{P}_2) = Vol_2(\mathcal{P}_1 + \mathcal{P}_2) - Vol_2(\mathcal{P}_1) - Vol_2(\mathcal{P}_2)$ . Furthermore, if  $\mathcal{P} := \mathcal{P}_1 = \mathcal{P}_2$ , then  $MV_2(\mathcal{P}, \mathcal{P}) = 2! Vol_2(\mathcal{P})$  because  $Vol_2(2\mathcal{P}) = 4Vol_2(\mathcal{P})$ . From the proposition, we recognize that the mixed volume is a generalization of the volume of a polytope, so that computing mixed volume is classified as a  $\#\mathcal{P}$ -complete problem from perspective of computational complexity theory. We can find the detail description in [16, 28]. The following proposition implies that the mixed volume remains invariant under some specific condition.

**Proposition 2.3.3.** ([39]) *Let  $\mathcal{P}_i$  ( $i \in N$ ) be a polytope in  $\mathbb{R}^n$ .*

$$(a) \quad MV_n(\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n) \text{ is invariant if } \mathcal{P}_i \text{ and } \mathcal{P}_j \text{ for } i \neq j \text{ exchange each other.}$$

$$(b) \quad MV_n(\mathcal{P}_1, \dots, \mathcal{P}_i, \dots, \mathcal{P}_n) = MV_n(\mathcal{P}_1, \dots, \mathcal{P}_i + \mathbf{a}, \dots, \mathcal{P}_n) \text{ for } \mathbf{a} \in \mathbb{Z}^n.$$

(a) is trivial from the definition of the mixed volume. By use of the property, we will develop the dynamic enumeration algorithm in Section 3.4. Also, (b) will be utilized when we generate artificial semi-mixed polynomial systems in Subsection 4.3.2.

In the rest of this section, we precisely describe Bernshtein's theorem [3], which is fundamental to polyhedral homotopy method, and show its modification theorem by Li and Wang [35]. In general, Bernshtein's theorem provides a much tighter upper bound for the number of isolated zeros of a polynomial system, compared with the classical one known as

the Bézout bound in [49]. In 1995, Huber and Sturmfels [29] gave the proof of the theorem in different ways from the original one by Bernshtein, and, as a consequence, proposed polyhedral homotopy functions. Also, we see another approach for constructing homotopy functions based on the theorem in [59] written by Verschelde, Verlinden and Cools. They follow the lines of the proof of the theorem. Let  $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_n(\mathbf{x}))$  be a polynomial system in  $\mathbf{x} \in \mathbb{C}^n$  with the support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n)$ , and  $\mathcal{P}_i = \text{conv}(\mathcal{A}_i)$  the Newton polytope of  $f_i$ . The following theorem gives the upper bound for the number of isolated zeros of  $\mathbf{f}(\mathbf{x})$  in  $(\mathbb{C}^*)^n$ , where  $\mathbb{C}^* := \mathbb{C} \setminus \{0\}$ .

**Theorem 2.3.4.** (*[3, Theorem A]*) *The number of isolated zeros, counting multiplicities, in  $(\mathbb{C}^*)^n$  of a polynomial system  $\mathbf{f}(\mathbf{x})$  is bounded by the mixed volume  $MV_n(\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n)$ . In particular, when  $\mathbf{f}(\mathbf{x})$  is in general position, it is exactly equal to the mixed volume  $MV_n(\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n)$ .*

We may need to explain the phrase “ $\mathbf{f}(\mathbf{x})$  is in general position”. The polynomials  $f_i$  of the system  $\mathbf{f} = (f_1, f_2, \dots, f_n)$  are said to be in *general position* if each coefficient of monomials of  $f_i$  is chosen as a random complex number. See, for example, in [37, 39] if one want to refer to the details. The mixed volume bound over the number of solutions is sometimes called the *BKK bound* because there are some related papers by Kushnirenko (1976) and Khovanskii (1978) for Bernshtein’s original paper (1975). A limitation of the theorem is that it only counts the number of isolated zeros of a polynomial system in  $(\mathbb{C}^*)^n$  even though we are interested in all isolated zeros in  $\mathbb{C}^n$ . In 1997, Li and Wang modified the problem as the following theorem.

**Theorem 2.3.5.** (*[35]*) *Let  $\mathcal{A}'_i = \mathcal{A}_i \cup \{0\}$  for every  $i \in N$  and  $\mathcal{P}'_i = \text{conv}(\mathcal{A}'_i)$ . The mixed volume  $MV_n(\mathcal{P}'_1, \mathcal{P}'_2, \dots, \mathcal{P}'_n)$  is an upper bound for the number of isolated zeros, counting multiplicities, of  $\mathbf{f}(\mathbf{x})$  in  $\mathbb{C}^n$ .*

In light of this theorem, we can compute all isolated zeros of a polynomial system  $\mathbf{f}(\mathbf{x})$  in  $\mathbf{x} \in \mathbb{C}^n$  through polyhedral-linear homotopy functions, which will be described in Subsection 2.5.2.

## 2.4 Mixed Cells

We introduce the combinatorial tool for computing the mixed volume for  $n$ -tuple Newton polytopes  $(\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n)$ . The main idea is to subdivide a polytope  $\mathcal{P}$ , which is the Minkowski sum of  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$ , into some piece polytopes  $\mathcal{R}_j$ , which are components in a polyhedral subdivision of  $\mathcal{P}$ , in a special way. According to [12], this idea has been known in the field of combinatorial geometry; it was first written by Betke [4] in 1992, and discovered independently by Huber and Sturmfels [29] in 1995. In the following subsection, we will see that only distinct Newton polytopes among  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$  contribute the mixed volume computation for  $n$ -tuple Newton polytopes  $(\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n)$ . This fact, presented by [29], enables us to reduce the computational task.

### 2.4.1 Relation between Mixed Volume and Mixed Cells

We begin with the definition of a *polyhedral subdivision* of a polytope. Let  $\mathcal{P}$  be a  $n$ -dimensional polytope in  $\mathbb{R}^n$ , i.e.,  $\dim(\mathcal{P}) = n$ . A polyhedral subdivision of  $\mathcal{P}$  is a collection  $\mathcal{R}$  of finitely many polytopes  $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_s$  in  $\mathbb{R}^n$  such that

- (a)  $\dim(\mathcal{R}_j) = n$  for every  $j \in \{1, 2, \dots, s\}$ ,
- (b)  $\mathcal{P} = \bigcup_{j=1}^s \mathcal{R}_j$ ,
- (c)  $\mathcal{R}_{j_1} \cap \mathcal{R}_{j_2}$  is a face of both  $\mathcal{R}_{j_1}$  and  $\mathcal{R}_{j_2}$  for  $j_1 \neq j_2$ .

A piece polytope  $\mathcal{R}_j \in \mathcal{R}$  in a polyhedral subdivision is called *cell*.

Next, we define a fine mixed subdivision, which is a specialization of a polyhedral subdivision, for the Minkowski sum of Newton polytopes. For  $m \leq n$ , let  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  ( $\mathcal{A}_i \subseteq \mathbb{Z}_+^n$ ) be a semi-mixed  $(k_1, k_2, \dots, k_m)$ -support of  $\mathbf{f}(\mathbf{x})$  in  $\mathbf{x} \in \mathbb{C}^n$ , where  $\sum_{i \in M} k_i = n$ . For each polytope  $\mathcal{P}_i = \text{conv}(\mathcal{A}_i)$ , we consider the Minkowski sum  $\mathcal{P} = \mathcal{P}_1 + \mathcal{P}_2 + \dots + \mathcal{P}_m$ . Suppose that  $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_s$  are cells in a polyhedral subdivision of  $\mathcal{P}$ , satisfying (a), (b) and (c). The collection  $\mathcal{R}$  of  $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_s$  is said to be a *mixed subdivision* of  $\mathcal{P}$  if each cell  $\mathcal{R}_j$  is represented as the Minkowski sum  $\text{conv}(C_1^j) + \text{conv}(C_2^j) + \dots + \text{conv}(C_m^j)$  for some subset  $\mathbf{C} = (C_1^j, C_2^j, \dots, C_m^j)$  of  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$ , such that

- (d)  $\sum_{i=1}^m \dim(\text{conv}(C_i^j)) = n$ ,

and if these satisfy an additional condition

- (e) for each  $j \in \{1, 2, \dots, s\}$ ,  $\text{conv}(C_i^j)$  ( $i \in M$ ) is a simplex of dimension  $\#C_i^j - 1$ ,

it is said to be a *fine mixed subdivision*.

For a semi-mixed  $(k_1, k_2, \dots, k_m)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$ , satisfying  $\sum_{i \in M} k_i = n$ , let  $((\mathcal{P}_i, k_i) : i \in M)$  denote  $n$ -tuple Newton polytopes

$$\left( \overbrace{(\mathcal{P}_1, \dots, \mathcal{P}_1)}^{k_1}, \overbrace{(\mathcal{P}_2, \dots, \mathcal{P}_2)}^{k_2}, \dots, \overbrace{(\mathcal{P}_m, \dots, \mathcal{P}_m)}^{k_m} \right).$$

The mixed volume for  $n$ -tuple Newton polytopes  $((\mathcal{P}_i, k_i) : i \in M)$  can be computed easily from the volumes of some specific cells  $\mathcal{R}_j \in \mathcal{R}$  if we construct a (fine) mixed subdivision  $\mathcal{R}$  of the Minkowski sum  $\mathcal{P}$  of  $m$  polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_m$ . By  $\text{MV}_n((\mathcal{P}_i, k_i) : i \in M)$ , We abbreviate the notation for the mixed volume for  $n$ -tuple Newton polytopes  $((\mathcal{P}_i, k_i) : i \in M)$  such as  $\text{MV}_n((\mathcal{P}_i, k_i) : i \in M)$ . The mixed volume for  $n$ -tuple Newton polytopes  $((\mathcal{P}_i, k_i) : i \in M)$  can be obtained by constructing a mixed subdivision of the Minkowski sum  $\mathcal{P}$  of  $m$  Newton polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_m$ . For  $\mathcal{R}_j = \text{conv}(C_1^j) + \text{conv}(C_2^j) + \dots + \text{conv}(C_m^j)$ , we define type  $(\mathcal{R}_j) = (\dim(\text{conv}(C_1^j)), \dim(\text{conv}(C_2^j)), \dots, \dim(\text{conv}(C_m^j)))$ .

**Theorem 2.4.1.** ([29, 12]) Let  $\mathcal{P}$  be the Minkowski sum  $\mathcal{P} = \mathcal{P}_1 + \mathcal{P}_2 + \cdots + \mathcal{P}_m$ . Suppose that  $\mathcal{R}$  is a collection of  $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_s$  in a mixed subdivision of  $\mathcal{P}$ . Then,

$$MV_n((\mathcal{P}_i, k_i) : i \in M) = \sum_{\substack{\mathcal{R}_j \in \mathcal{R} \\ \text{type } (\mathcal{R}_j) = (k_1, k_2, \dots, k_m)}} k_1! k_2! \cdots k_m! \cdot \text{Vol}_n(\mathcal{R}_j). \quad (2.3)$$

To quote the phrase from [12], the key idea of this proof is that a mixed subdivision behave well under scaling. This theorem implies that a mixed subdivision of  $\mathcal{P}$ , which is a weaker condition than a fine mixed subdivision, provides the mixed volume for  $((\mathcal{P}_i, k_i) : i \in M)$ . If a fine mixed subdivision  $\mathcal{R}$  is constructed, we can compute the volume of each cell  $\mathcal{R}_j \in \mathcal{R}$  as the determinant of a matrix. Remember that  $\mathcal{R}_j = \text{conv}(C_1^j) + \text{conv}(C_2^j) + \cdots + \text{conv}(C_m^j)$  and  $\mathbf{C}^j = (C_1^j, C_2^j, \dots, C_m^j)$ . Suppose that  $\text{type}(\mathcal{R}_j) = (k_1, k_2, \dots, k_m)$ . Then, the subset  $C_i^j$  of  $\mathcal{A}_i$  has  $k_i + 1$  elements, denoted  $\mathbf{a}_{i0}, \mathbf{a}_{i1}, \dots, \mathbf{a}_{ik_i}$ . For the cell  $\mathcal{R}_j = \sum_{i \in M} \text{conv}(C_i^j)$  with  $C_i^j = \{\mathbf{a}_{i0}, \mathbf{a}_{i1}, \dots, \mathbf{a}_{ik_i}\}$ , we construct the matrix

$$V(\mathbf{C}^j) = \left( \overbrace{\mathbf{a}_{11} - \mathbf{a}_{10}, \dots, \mathbf{a}_{1k_1} - \mathbf{a}_{10}}^{k_1}, \dots, \overbrace{\mathbf{a}_{m1} - \mathbf{a}_{m0}, \dots, \mathbf{a}_{mk_m} - \mathbf{a}_{m0}}^{k_m} \right) \in \mathbb{Z}^{n \times n}.$$

Then, the determinant of the matrix  $V(\mathbf{C}^j)$  coincides with  $k_1 k_2 \cdots k_m$  times the  $n$ -dimensional volume of  $\mathcal{R}_j$ . Accordingly, we get the more simple formula for the mixed volume computation through a fine mixed subdivision.

**Proposition 2.4.2.** ([29]) Suppose that  $\mathcal{R}$  is a collection of  $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_s$  in a fine mixed subdivision of  $\mathcal{P} = \mathcal{P}_1 + \mathcal{P}_2 + \cdots + \mathcal{P}_m$ . Then,

$$MV_n((\mathcal{P}_i, k_i) : i \in M) = \sum_{\substack{\mathcal{R}_j \in \mathcal{R} \\ \text{type } (\mathcal{R}_j) = (k_1, k_2, \dots, k_m)}} |\det(V(\mathbf{C}^j))| \quad (2.4)$$

For a given  $(k_1, k_2, \dots, k_m)$ -support, by a *mixed cell*, we mean a cell  $\mathcal{R}_j$  in a (fine) mixed subdivision for the Newton polytopes if  $\text{type}(\mathcal{R}_j) = (k_1, k_2, \dots, k_m)$ .

From the above theorem or proposition, we see that only distinct support sets contribute the mixed volume computation. A semi-mixed support certainly can be treated as a fully mixed type. Let  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  be a semi-mixed  $(k_1, k_2, \dots, k_m)$ -support. The mixed volume for  $n$ -tuple Newton polytopes  $((\mathcal{P}_i, k_i) : i \in M)$ , where  $\mathcal{P}_i = \text{conv}(\mathcal{A}_i)$ , can be obtained by summing up the volumes of all mixed cells  $\mathcal{R}_j$ , satisfying  $\text{type}(\mathcal{R}_j) = (1, 1, \dots, 1)$ , in a (fine) mixed subdivision of

$$\mathcal{P} = \overbrace{\mathcal{P}_1 + \cdots + \mathcal{P}_1}^{k_1} + \overbrace{\mathcal{P}_2 + \cdots + \mathcal{P}_2}^{k_2} + \cdots + \overbrace{\mathcal{P}_m + \cdots + \mathcal{P}_m}^{k_m}.$$

However, the numerical results in [22] show that the computational time for finding all mixed cells can be reduced considerably if we focus on the only distinct supports for a semi-mixed system. Therefore, it is important for computational efficiency to utilize a semi-mixed structure. In Subsection 4.3.2, we will again recognize the importance through the numerical results.

## 2.4.2 Construction of Fine Mixed Subdivision

Huber and Sturmfels [29] and also Betke [4] presents how to construct a fine mixed subdivision of a polytope. Let  $\mathcal{P} \subseteq \mathbb{R}^n$  be the Minkowski sum of  $m$  Newton polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_m$ , where  $\mathcal{P}_i = \text{conv}(\mathcal{A}_i)$ , for a semi-mixed  $(k_1, k_2, \dots, k_m)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  ( $\mathcal{A}_i \subseteq \mathbb{R}^n$ ). We consider a real-valued function  $\omega_i : \mathcal{A}_i \rightarrow \mathbb{R}$ , and call it a *lifting function* because the function  $\omega_i$  is used in order to lift  $\mathcal{A}_i$  to

$$\hat{\mathcal{A}}_i = \left\{ \hat{\mathbf{a}} = \begin{pmatrix} \mathbf{a} \\ \omega_i(\mathbf{a}) \end{pmatrix} : \mathbf{a} \in \mathcal{A}_i \right\}. \quad (2.5)$$

Let  $\hat{\mathcal{P}}_i \subseteq \mathbb{R}^{n+1}$  denote the convex hull of  $\hat{\mathcal{A}}_i$ . We consider the Minkowski sum  $\hat{\mathcal{P}} \subseteq \mathbb{R}^{n+1}$ , say a *lifted polytope*, of  $\hat{\mathcal{P}}_1, \hat{\mathcal{P}}_2, \dots, \hat{\mathcal{P}}_m$ . Suppose that the image of each function  $\omega_i$  is a generic number, so that the value  $\omega_i(\mathbf{a})$  is sufficiently random for every  $i \in M$  and every  $\mathbf{a} \in \mathcal{A}_i$ . Then, the projection  $\mathbb{R}^{n+1} \rightarrow \mathbb{R}^n$  of the set of lower facets of the lifted polytope  $\hat{\mathcal{P}} \subseteq \mathbb{R}^{n+1}$  gives a fine mixed subdivision of  $\mathcal{P} \subseteq \mathbb{R}^n$ . Here, a facet of  $\hat{\mathcal{P}}$  is said to be a *lower facet* if the inner normal has a positive last coordinate, and the set of lower facets a *lower envelope* of  $\hat{\mathcal{P}}$ . In this thesis, we call such a function  $\omega_i$  a *generic lifting*. If we have only interest for the construction of a mixed subdivision, it is enough to employ a generic linear lifting. We call a lifting function  $\omega_i$  a *generic linear lifting* if there exists an  $n$ -dimensional real vector  $\boldsymbol{\xi}_i$  chosen from  $\mathbb{R}^n$  randomly such that  $\omega_i(\mathbf{a}) = \langle \boldsymbol{\xi}_i, \mathbf{a} \rangle$  for every  $\mathbf{a} \in \mathcal{A}_i$ . The property of the generic linear lifting  $\omega_i$  is that all elements in  $\hat{\mathcal{A}}_i$ , which is obtained by the application of the lifting  $\omega_i$  to  $\mathcal{A}_i$ , lies on the same hyperplane. The above discussion suggests the enumeration procedure for all mixed cells in a fine mixed subdivision of the Minkowski sum  $\mathcal{P} \subseteq \mathbb{R}^n$  of  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_m$  through a lower envelope of a lifted polytope  $\hat{\mathcal{P}} \subseteq \mathbb{R}^{n+1}$ , as summarized below.

Step 1: Lift  $\mathcal{A}_i$  to  $\hat{\mathcal{A}}_i$ , followed by (2.5), using a generic lifting  $\omega_i$ , and consider the Minkowski sum  $\hat{\mathcal{P}}$  of  $m$  Newton polytopes  $\hat{\mathcal{P}}_1, \hat{\mathcal{P}}_2, \dots, \hat{\mathcal{P}}_m$ ,

Step 2: Find all lower facets, which are corresponding to mixed cells, of a lifted polytope  $\hat{\mathcal{P}}$ .

Finding lower facets of  $\hat{\mathcal{P}}$ , which is a key step of the above procedure, can be conducted by the use of LP problem. There are some formulations for finding mixed cells through a lower envelope of  $\hat{\mathcal{P}}$ . Emiris and Canny developed a geometric formulation in [17]. The similar idea can be found in the Ph. D. thesis [61] written by Verschelde. He also proposed several methods by modifying the lifting function in [60, 62]. Among these formulations, Li and Li's one [38], which also has geometric interpretation, is more practical because it gives compact description for finding mixed cells. Furthermore, it can deal with a semi-mixed support set naturally, whereas Emiris and Canny's one specializes in a fully mixed support. Our method is founded on Li and Li's method. In Subsection 3.2.2, we will refer to the existing methods for finding mixed cells based on the above enumeration procedure in more detail.

In the rest of this subsection, we show the formulation, presented by Li and Li in [38]. Let  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  ( $\mathcal{A}_i \subseteq \mathbb{Z}_+^n$ ) be a semi-mixed  $(k_1, k_2, \dots, k_m)$ -support with  $m \leq n$  and  $\sum_{i \in M} k_i = n$ . We consider a generic lifting  $\omega_i$ , and then lift  $\mathcal{A}_i$  to  $\hat{\mathcal{A}}_i$ , followed by (2.5). Also, for each element  $\mathbf{a}_i \in \mathbb{R}^n$  in a subset  $C_i$  of  $\mathcal{A}_i$ , let  $\hat{C}_i$  denote the set of the lifted elements  $\hat{\mathbf{a}}_i = (\mathbf{a}_i, \omega_i(\mathbf{a}_i)) \in \mathbb{R}^{n+1}$ . Let  $\mathcal{P} = \sum_{i \in M} \mathcal{P}_i$  for  $\mathcal{P}_i = \text{conv}(\mathcal{A}_i)$ , and  $\hat{\mathcal{P}} = \sum_{i \in M} \hat{\mathcal{P}}_i$  for  $\hat{\mathcal{P}}_i = \text{conv}(\hat{\mathcal{A}}_i)$ . Remember that each mixed cell  $\mathcal{R}_j \in \mathcal{R}$  with type  $(\mathcal{R}_j) = (k_1, k_2, \dots, k_m)$  in a fine mixed subdivision  $\mathcal{R}$  of  $\mathcal{P}$  is described as the Minkowski sum  $\text{conv}(C_1) + \text{conv}(C_2) + \dots + \text{conv}(C_m)$  using the subset  $C_i$  of  $\mathcal{A}_i$  with  $\#C_i = k_i + 1$ . The following problem provides every  $\mathbf{C} = (C_1, C_2, \dots, C_m) \in \mathcal{A}_1 \times \mathcal{A}_2 \times \dots \times \mathcal{A}_m$  as the answer, which gives a mixed cell  $\mathcal{R}_j \in \mathcal{R}$  as the Minkowski sum of the convex hull of each  $C_i$ .

**Problem 2.4.3.** Find all  $\mathbf{C} = (C_1, C_2, \dots, C_m) \in \prod_{i=1}^m \mathcal{A}_i$  with  $\#C_i = k_i + 1$  such that the linear inequality system in a variable vector  $(\boldsymbol{\alpha}, \boldsymbol{\beta}) = (\alpha_1, \dots, \alpha_n, \beta_1, \dots, \beta_m) \in \mathbb{R}^{n+m}$

$$\mathcal{E}(\mathbf{C}) := \begin{cases} \beta_i = \langle \hat{\mathbf{a}}, \hat{\boldsymbol{\alpha}} \rangle & \forall \hat{\mathbf{a}} \in \hat{C}_i, \\ \beta_i \leq \langle \hat{\mathbf{a}}', \hat{\boldsymbol{\alpha}} \rangle & \forall \hat{\mathbf{a}}' \in \hat{\mathcal{A}}_i \setminus \hat{C}_i \end{cases} \quad (i \in M) \quad (2.6)$$

where

$$\hat{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}, 1) \in \mathbb{R}^{n+1}$$

is feasible.

The feasibility check of the linear inequality system (2.6) can be carried out by the use of an LP formulation. The problem asks whether there exist the hyperplanes with the normal vector  $(\boldsymbol{\alpha}, \boldsymbol{\beta})$  where each hyperplane supports  $\hat{\mathcal{P}}_i$  at exactly  $k_i + 1$  points in  $\hat{C}_i \subseteq \hat{\mathcal{A}}_i$  for  $i \in M$ . The existence of such hyperplanes for some  $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_m$  means that one of lower facets of  $\hat{\mathcal{P}}$  is represented as the Minkowski sum  $\text{conv}(\hat{C}_1) + \text{conv}(\hat{C}_2) + \dots + \text{conv}(\hat{C}_m)$ .

The Problem 2.4.3 has another geometric interpretation. Define the polytope

$$\mathcal{V} = \{(\boldsymbol{\alpha}, \boldsymbol{\beta}) \in \mathbb{R}^{n+m} : \langle \mathbf{a}, \boldsymbol{\alpha} \rangle + \omega_i(\mathbf{a}) - \beta_i \geq 0 \text{ for every } \mathbf{a} \in \mathcal{A}_i \text{ and every } i \in M\}.$$

using a generic lifting  $\omega_i$ . Because the lifting function  $\omega_i$  provides a sufficient random number as the image, any vertex of the polytope  $\mathcal{V}$  is nondegenerate in the sense that each vertex is given as the intersection of exactly  $n + m$  facets of  $\mathcal{V}$ . Clearly, each solution of (2.6) is corresponding to a nondegenerate vertex of  $\mathcal{V}$ . Therefore,

**Condition 2.4.4.** Let  $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) \in \mathbb{R}^{n+m}$  be the solution of the feasible linear inequality system  $\mathcal{E}(\mathbf{C})$  for some  $\mathbf{C} = (C_1, C_2, \dots, C_m)$  with  $C_i \subseteq \mathcal{A}_i$  and  $\#C_i = k_i + 1$ . Then, for the solution  $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ ,  $n + m$  constraints in the system (2.6) satisfy the equalities, and the others exactly the inequalities.

## 2.5 Solving Polynomial Systems through Polyhedral Homotopies

In this section, we introduce a family of polyhedral-linear homotopy functions  $\mathbf{h}(\mathbf{x}, t) : \mathbb{C}^n \times [0, 1] \rightarrow \mathbb{C}^n$  for computing all isolated zeros of a polynomial system  $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_n(\mathbf{x}))$  in  $\mathbf{x} \in \mathbb{C}^n$ . In Subsection 2.5.1, the polyhedral homotopies, presented by Huber and Sturmfels [29], are defined for  $\tilde{\mathbf{f}}(\mathbf{x})$  which is in general position, that is, has generic (random) coefficients but the same support as the original system  $\mathbf{f}(\mathbf{x})$ . The construction is based on the mixed cells for  $n$  Newton polytopes of the polynomials  $f_i$  of  $\mathbf{f}$ . The polyhedral homotopies serve as a homotopy from the start system  $\mathbf{g}(\mathbf{x})$  to  $\tilde{\mathbf{f}}(\mathbf{x})$ . In Subsection 2.5.2, each polyhedral-linear homotopy is constructed as a combination of a polyhedral homotopy and a linear homotopy, which is shown in the paper [37, 39] by Li and he called the combined homotopy *cheater's homotopy* [34]. The linear homotopy is used as a homotopy between  $\tilde{\mathbf{f}}(\mathbf{x})$  and  $\mathbf{f}(\mathbf{x})$ . Figure 2.2 shows the relation among polyhedral, linear and polyhedral-linear homotopies.

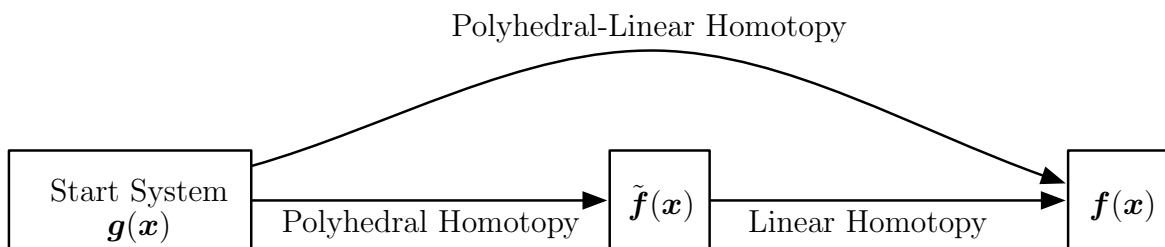


Figure 2.2: Relationship diagram for polyhedral-linear strategy

### 2.5.1 Polyhedral Homotopies

Remember that we write  $f_i(\mathbf{x})$  of  $\mathbf{f}(\mathbf{x})$  in (2.1) as

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

for support  $\mathcal{A}_i \subseteq \mathbb{Z}_+^n$  and some  $c_i(\mathbf{a}) \in \mathbb{C}$  ( $\mathbf{a} \in \mathcal{A}_i$ ). Taking account of Theorem 2.3.5 by Li and Wang, to find all isolated zeros of a polynomial system  $\mathbf{f}(\mathbf{x})$  in  $\mathbb{C}^n$ , we first add a constant term  $\mathbf{x}^{\mathbf{0}} = 1$  to polynomials  $f_i$ , which do not have it, and denote the new supports  $\mathcal{A}_i \cup \{\mathbf{0}\}$  as  $\tilde{\mathcal{A}}_i$ . Next, by choosing a number generically from  $\mathbb{C}$ , an auxiliary polynomial system  $\tilde{\mathbf{f}}(\mathbf{x})$  is obtained. More precisely, for every  $\mathbf{a} \in \tilde{\mathcal{A}}_i$  and every  $i \in N$ , let  $\tilde{c}_i(\mathbf{a}) \in \mathbb{C}$  be complex numbers chosen generically from  $\mathbb{C}$ . Consider an auxiliary polynomial system  $\tilde{\mathbf{f}}(\mathbf{x})$  such that the  $i$ th component is given by

$$\tilde{f}_i(\mathbf{x}) = \sum_{\mathbf{a} \in \tilde{\mathcal{A}}_i} \tilde{c}_i(\mathbf{a}) \mathbf{x}^{\mathbf{a}} \quad (2.7)$$

which is in general position. In the succeeding discussions, we first introduce polyhedral homotopy for  $\tilde{\mathbf{f}}(\mathbf{x})$  and then linear homotopy from  $\tilde{\mathbf{f}}(\mathbf{x})$  to the original polynomial system  $\mathbf{f}(\mathbf{x})$  whose zeros are to be found. Finally, we combine these two homotopies to have a polyhedral-linear homotopy for  $\mathbf{f}(\mathbf{x})$ .

Let  $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_n(\mathbf{x}))$  be a system of  $n$  polynomials  $f_i(\mathbf{x})$  in an  $n$ -dimensional complex variable vector  $\mathbf{x} \in \mathbb{C}^n$ . Suppose that the obtained auxiliary polynomial system  $\tilde{\mathbf{f}}$  from  $\mathbf{f}$

$$\tilde{\mathbf{f}}(\mathbf{x}) = (\overbrace{f_{11}(\mathbf{x}), \dots, f_{1k_1}(\mathbf{x})}^{k_1}, \dots, \overbrace{f_{m1}(\mathbf{x}), \dots, f_{mk_m}(\mathbf{x})}^{k_m}) : \mathbb{C}^n \rightarrow \mathbb{C}^n, \text{ where } \sum_{i \in M} k_i = n,$$

has a semi-mixed  $(k_1, k_2, \dots, k_m)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  such that each support  $\mathcal{A}_{ij}$  of  $f_{ij}$  satisfies

$$\mathcal{A}_i := \mathcal{A}_{i1} = \mathcal{A}_{i2} = \dots = \mathcal{A}_{ik_i} \quad (i \in M).$$

Here, the component  $\tilde{f}_{ij}$  of  $\tilde{\mathbf{f}}$  is written as

$$\tilde{f}_{ij}(\mathbf{x}) = \sum_{\mathbf{a} \in \mathcal{A}_i} \tilde{c}_{ij}(\mathbf{a}) \mathbf{x}^{\mathbf{a}}.$$

Let  $\tilde{\mathcal{H}}$  denote a finite family of polyhedral homotopy functions  $\tilde{\mathbf{h}}$ . For the semi-mixed system  $\tilde{\mathbf{f}}$ , we construct polyhedral homotopy functions  $\tilde{\mathbf{h}}(\mathbf{x}, t) \in \tilde{\mathcal{H}} : \mathbb{C}^n \times [0, 1] \rightarrow \mathbb{C}^n$  such that

$$\tilde{\mathbf{h}}(\mathbf{x}, t) = (\overbrace{\tilde{h}_{11}(\mathbf{x}, t), \dots, \tilde{h}_{1k_1}(\mathbf{x}, t)}^{k_1}, \dots, \overbrace{\tilde{h}_{m1}(\mathbf{x}, t), \dots, \tilde{h}_{mk_m}(\mathbf{x}, t)}^{k_m}) : \mathbb{C}^n \times [0, 1] \rightarrow \mathbb{C}^n.$$

Let  $K_i = \{1, 2, \dots, k_i\}$ . For each polyhedral homotopy function  $\tilde{\mathbf{h}} \in \tilde{\mathcal{H}}$ , the component  $\tilde{h}_{ij}$  of  $\tilde{\mathbf{h}}$  is of the form

$$\tilde{h}_{ij}(\mathbf{x}, t) = \sum_{\mathbf{a} \in \mathcal{A}_i} \tilde{c}_{ij}(\mathbf{a}) \mathbf{x}^{\mathbf{a}} t^{\rho_{ij}(\mathbf{a})} \quad \text{for every } i \in M \text{ and every } j \in K_i. \quad (2.8)$$

Note that  $\tilde{h}_{ij}$  has the same form as  $\tilde{f}_{ij}$  except the parameter  $t^{\rho_{ij}(\mathbf{a})}$ . The nonnegative numbers  $\rho_{ij}(\mathbf{a})$  are presented as byproduct of the answer of Problem 2.4.3, defined in Subsection 2.4.2. Let us consider the problem. Here, if the linear inequality system  $\mathcal{E}(\mathbf{C})$  for some  $\mathbf{C} = (C_1, C_2, \dots, C_m) \in \prod_{i=1}^m \mathcal{A}_i$  with  $\#C_i = k_i + 1$  is feasible, we also call such  $\mathbf{C} = (C_1, C_2, \dots, C_m)$  a mixed cell for convenient although a terminology ‘‘mixed cell’’ was defined in order to represent some specific piece polytope in a fine mixed subdivision. The Problem 2.4.3 requires us to find all mixed cells  $\mathbf{C} = (C_1, C_2, \dots, C_m) \in \prod_{i=1}^m \mathcal{A}_i$  with  $\#C_i = k_i + 1$  such that the linear inequality system  $\mathcal{E}(\mathbf{C})$  is feasible. Let  $\Omega^*$  denote a collection of all the mixed cells  $\mathbf{C}$ , produced as the answer of the problem. As we mention in Subsection 2.4.2, for each  $\mathbf{C} \in \Omega^*$ ,  $\mathcal{E}(\mathbf{C})$  has the only one solution  $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = (\alpha_1^*, \dots, \alpha_n^*, \beta_1^*, \dots, \beta_m^*) \in \mathbb{R}^{n+m}$ . Using the solution  $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) \in \mathbb{R}^{n+m}$ , for every  $i \in M$  and every  $j \in K_i$ , let

$$\rho_{ij}(\mathbf{a}) = \langle \mathbf{a}, \boldsymbol{\alpha}^* \rangle + \omega_i(\mathbf{a}) - \beta_i^* \quad (\mathbf{a} \in \mathcal{A}_i) \quad (2.9)$$

where  $\omega_i : \mathcal{A}_i \rightarrow \mathbb{R}$  is a generic lifting function. It means that a finite family of polyhedral homotopy functions  $\tilde{\mathcal{H}}$  is provided by a set of mixed cells  $\Omega^*$ , and each polyhedral homotopy  $\tilde{\mathbf{h}} \in \tilde{\mathcal{H}}$  is one-to-one corresponding to the mixed cell  $\mathbf{C} \in \Omega^*$ . From Condition 2.4.4, for some mixed cell  $\mathbf{C} = (C_1, C_2, \dots, C_m) \in \Omega^*$ , the powers  $\rho_{ij}(\mathbf{a})$  of  $t$  in  $\tilde{h}_{ij}$  satisfy

$$\begin{aligned} \rho_{ij}(\mathbf{a}) &= 0, & \mathbf{a} \in C_i \\ \rho_{ij}(\mathbf{a}') &> 0, & \mathbf{a}' \in \mathcal{A}_i \setminus C_i \end{aligned} \quad \text{for each } i \in M \text{ and each } j \in K_i.$$

Now, we are ready to define the component  $\tilde{h}_{ij}$  of  $\tilde{\mathbf{h}} \in \tilde{\mathcal{H}}$ . For each  $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) \in \mathbb{R}^{n+m}$  obtained from mixed cells  $\mathbf{C} \in \Omega^*$ ,

$$\begin{aligned} \tilde{h}_{ij}(\mathbf{x}, t) &= \sum_{\mathbf{a} \in \mathcal{A}_i} \tilde{c}_{ij}(\mathbf{a}) \mathbf{x}^{\mathbf{a}} t^{(\mathbf{a}, \boldsymbol{\alpha}^*) + \omega_i(\mathbf{a}) - \beta_i^*} \\ &= \sum_{\mathbf{a} \in C_i} \tilde{c}_{ij}(\mathbf{a}) \mathbf{x}^{\mathbf{a}} + \sum_{\mathbf{a} \in \mathcal{A}_i \setminus C_i} \tilde{c}_{ij}(\mathbf{a}) \mathbf{x}^{\mathbf{a}} t^{(\mathbf{a}, \boldsymbol{\alpha}^*) + \omega_i(\mathbf{a}) - \beta_i^*}. \end{aligned}$$

We recognize that for an auxiliary polynomial system  $\tilde{\mathbf{f}}$ , polyhedral homotopies  $\tilde{\mathbf{h}}$  are constructed using mixed cells  $\mathbf{C} \in \Omega^*$  for the support  $\mathcal{A}$  of  $\tilde{\mathbf{f}}$ , and these form a finite family of functions  $\tilde{\mathcal{H}}$  whose each element  $\tilde{\mathbf{h}}$  is one-to-one corresponding to  $\mathbf{C} \in \Omega^*$ .

For polyhedral homotopies  $\tilde{\mathbf{h}} \in \tilde{\mathcal{H}} : \mathbb{C}^n \times [0, 1] \rightarrow \mathbb{C}^n$ , we have  $\tilde{\mathbf{h}}(\mathbf{x}, 1) = \tilde{\mathbf{f}}(\mathbf{x})$  for every  $\mathbf{x} \in \mathbb{C}^n$ . Although the start system  $\mathbf{g}(\mathbf{x}) := \tilde{\mathbf{h}}(\mathbf{x}, 0)$  becomes a system of  $n$  polynomials with  $n$  unknowns such that

$$\mathbf{g}(\mathbf{x}) = \begin{cases} \mathbf{g}_1(\mathbf{x}) \\ \vdots \\ \mathbf{g}_m(\mathbf{x}) \end{cases} \quad \text{where } \mathbf{g}_i(\mathbf{x}) = \begin{cases} g_{i1}(\mathbf{x}) = \sum_{\mathbf{a} \in C_i} \tilde{c}_{i1}(\mathbf{a}) \mathbf{x}^{\mathbf{a}} \\ \vdots \\ g_{ik_i}(\mathbf{x}) = \sum_{\mathbf{a} \in C_i} \tilde{c}_{ik_i}(\mathbf{a}) \mathbf{x}^{\mathbf{a}}, \end{cases}$$

all the solutions of  $\mathbf{g}(\mathbf{x}) = \mathbf{0}$  are easily computable because  $\mathbf{g}$  can be transformed into a binomial system  $\bar{\mathbf{g}}$ , that is, a system of polynomials consisting of two terms. This is because  $\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_m$  has the special structure. The polynomial system  $\mathbf{g}_i$  consists of  $k_i$  polynomials  $g_{i1}, g_{i2}, \dots, g_{ik_i}$ , and all the polynomials  $g_{ij}$  share  $k_i + 1$  monomials  $\mathbf{x}^{\mathbf{a}}$  ( $\mathbf{a} \in C_i$ ) with each other. Even if we carry out the following operation

$$\begin{aligned} \mathbf{g}_i &= (g_{i1}, \dots, g_{ij_1}, \dots, g_{ij_2}, \dots, g_{ik_i}) \\ &\quad \downarrow \\ \bar{\mathbf{g}}_i &= (g_{i1}, \dots, g_{ij_1}, \dots, \alpha \cdot g_{ij_1} + \beta \cdot g_{ij_2}, \dots, g_{ik_i}), \quad \text{where } \alpha, \beta \in \mathbb{C}, \end{aligned} \tag{2.10}$$

the obtained polynomial system  $\bar{\mathbf{g}}_i(\mathbf{x})$  has the same solution set as  $\mathbf{g}_i(\mathbf{x}) = \mathbf{0}$ . For the polynomial system  $\mathbf{g}_i$  which has the structure as stated above, the operation (2.10) is corresponding the elementary row operation for the matrix

$$\begin{pmatrix} \tilde{c}_{i1}(\mathbf{a}_{i0}) & \tilde{c}_{i1}(\mathbf{a}_{i1}) & \cdots & \tilde{c}_{i1}(\mathbf{a}_{ik_i}) \\ \tilde{c}_{i2}(\mathbf{a}_{i0}) & \tilde{c}_{i2}(\mathbf{a}_{i1}) & \cdots & \tilde{c}_{i2}(\mathbf{a}_{ik_i}) \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{c}_{ik_i}(\mathbf{a}_{i0}) & \tilde{c}_{ik_i}(\mathbf{a}_{i1}) & \cdots & \tilde{c}_{ik_i}(\mathbf{a}_{ik_i}) \end{pmatrix} \in \mathbb{C}^{k_i \times (k_i+1)}$$

where let  $\mathbf{a}_{i0}, \mathbf{a}_{i1}, \dots, \mathbf{a}_{ik_i}$  denote  $k_i + 1$  elements in  $C_i$ . Because all elements in the matrix are the random number, by applying the Gaussian elimination, the matrix is converted into

$$\begin{pmatrix} \tilde{c}_{i1}(\mathbf{a}_{i0}) & \tilde{c}_{i1}(\mathbf{a}_{i1}) & 0 & 0 & \cdots & 0 \\ \tilde{c}_{i2}(\mathbf{a}_{i0}) & 0 & \tilde{c}_{i2}(\mathbf{a}_{i2}) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \tilde{c}_{ik_i}(\mathbf{a}_{i0}) & 0 & 0 & 0 & \cdots & \tilde{c}_{ik_i}(\mathbf{a}_{ik_i}) \end{pmatrix} \in \mathbb{C}^{k_i \times (k_i+1)},$$

where  $\tilde{c}_{ij}$  is a nonzero complex number. Therefore, the polynomial system  $\mathbf{g} = \tilde{\mathbf{h}}(\mathbf{x}, 0) \in \tilde{\mathcal{H}}$  can be transformed into a binomial system

$$\mathbf{g}'(\mathbf{x}) = \begin{cases} \mathbf{g}'_1(\mathbf{x}) \\ \vdots \\ \mathbf{g}'_m(\mathbf{x}) \end{cases} \quad \text{where } \mathbf{g}'_i(\mathbf{x}) = \begin{cases} \tilde{c}_{i1}(\mathbf{a}_{i0})\mathbf{x}^{\mathbf{a}_{i0}} + \tilde{c}_{i1}(\mathbf{a}_{i1})\mathbf{x}^{\mathbf{a}_{i1}} \\ \vdots \\ \tilde{c}_{ik_i}(\mathbf{a}_{i0})\mathbf{x}^{\mathbf{a}_{i0}} + \tilde{c}_{ik_i}(\mathbf{a}_{ik_i})\mathbf{x}^{\mathbf{a}_{ik_i}}. \end{cases} \quad (2.11)$$

Also, as we will see in the next subsection, the binomial system (2.11) can be solved easily, and it has  $|\det V(\mathbf{C})|$ , where

$$V(\mathbf{C}) = \left( \overbrace{\mathbf{a}_{11} - \mathbf{a}_{10}, \dots, \mathbf{a}_{1k_1} - \mathbf{a}_{10}}^{k_1}, \dots, \overbrace{\mathbf{a}_{m1} - \mathbf{a}_{m0}, \dots, \mathbf{a}_{mk_m} - \mathbf{a}_{m0}}^{k_m} \right) \in \mathbb{C}^{n \times n}, \quad (2.12)$$

nonsingular isolated zeros in  $(\mathbb{C}^*)^n$ . Therefore, it follows from Proposition 2.4.2 that the total number of the zeros of every start system  $\mathbf{g}(\mathbf{x}) = \tilde{\mathbf{h}}(\mathbf{x}, 0) \in \tilde{\mathcal{H}}$  coincides with the mixed volume for  $n$  Newton polytopes  $\text{conv}(\mathcal{A}_i)$  of auxiliary polynomials  $\tilde{f}_i$ .

## 2.5.2 Polyhedral-Linear Homotopies

Now, we combine a linear homotopy of the form

$$(1-t)\tilde{\mathbf{f}}(\mathbf{x}) + t\mathbf{f}(\mathbf{x}) \text{ for every } (\mathbf{x}, t) \in \mathbb{C}^n \times [0, 1]$$

from the auxiliary polynomial system  $\tilde{\mathbf{f}}(\mathbf{x})$  to the target polynomial system  $\mathbf{f}(\mathbf{x})$  with each polyhedral homotopy function  $\tilde{\mathbf{h}} \in \tilde{\mathcal{H}} : \mathbb{C}^n \times [0, 1] \rightarrow \mathbb{C}^n$ . Let  $\mathcal{H}$  denote a finite family of polyhedral-linear functions. Using each polyhedral homotopies  $\tilde{\mathbf{h}} \in \tilde{\mathcal{H}}$ , define a family of polyhedral-linear functions

$$\mathbf{h}(\mathbf{x}, t) = \left( \overbrace{h_{11}(\mathbf{x}, t), \dots, h_{1k_1}(\mathbf{x}, t)}^{k_1}, \dots, \overbrace{h_{m1}(\mathbf{x}, t), \dots, h_{mk_m}(\mathbf{x}, t)}^{k_m} \right) \in \mathcal{H} : \mathbb{C}^n \times [0, 1] \rightarrow \mathbb{C}^n$$

by

$$h_{ij}(\mathbf{x}, t) := \sum_{\mathbf{a} \in \mathcal{A}_i} ((1-t)\tilde{c}_{ij}(\mathbf{a}) + tc_{ij}(\mathbf{a})) \mathbf{x}^{\mathbf{a}} t^{\rho_{ij}(\mathbf{a})}.$$

Then,

- $\mathbf{h}(\mathbf{x}, 0) = \tilde{\mathbf{h}}(\mathbf{x}, 0)$  for every  $\mathbf{x} \in \mathbb{C}^n$ .
- $\mathbf{h}(\mathbf{x}, 1) = \mathbf{f}(\mathbf{x})$  for every  $\mathbf{x} \in \mathbb{C}^n$ .

Note that the above definition of the polyhedral-linear homotopy functions  $\mathbf{h} \in \mathcal{H}$  satisfies the properties (a)-(c), which are listed in Section 2.2 as a requirement to become a homotopy because

- For each  $\mathbf{h} \in \mathcal{H}$ ,  $\mathbf{g}(\mathbf{x}) = \mathbf{h}(\mathbf{x}, 0)$  can be transformed into a binomial system  $\mathbf{g}'(\mathbf{x})$ , i.e., a system of the polynomials consisting of exact two terms in variable vector  $\mathbf{x} \in \mathbb{C}^n$ , and hence, all solutions of the starting polynomial system  $\mathbf{h}(\mathbf{x}, 0) = \mathbf{0}$  can be computed easily.
- For every fixed  $t \in [0, 1)$ , the polynomial system  $\mathbf{h}(\mathbf{x}, t) = \mathbf{0}$  has only nonsingular solutions because we choose complex coefficient numbers  $\tilde{c}_{ij}(\mathbf{a})$  randomly; hence each connected component of  $\{(\mathbf{x}, t) \in \mathbb{C}^n \times [0, 1) : \mathbf{h}(\mathbf{x}, t) = \mathbf{0}\}$  forms a smooth path such that  $\{(\mathbf{x}(t), t) : t \in [0, 1)\}$ .
- For each isolated solutions  $\mathbf{x}^*$  of  $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ , there exists a solution  $\mathbf{x}^s$  of the starting polynomial system  $\mathbf{h}(\mathbf{x}, 0) = \mathbf{0}$  such that  $(\mathbf{x}^s, 0)$  is connected to  $(\mathbf{x}^*, 1)$  through a homotopy path of  $\mathbf{h}(\mathbf{x}, t) = \mathbf{0}$ .

Therefore, we confirm that each polyhedral-linear homotopy function  $\mathbf{h}(\mathbf{x}, t) \in \mathcal{H}$  serves as a homotopy in order to compute all isolated zeros of  $\mathbf{f}(\mathbf{x})$ .

In the previous subsection, we understood that the start system  $\mathbf{g}(\mathbf{x}) = \mathbf{h}(\mathbf{x}, 0)$  for each polyhedral-linear homotopy  $\mathbf{h} \in \mathcal{H}$  can be converted into the binomial system  $\mathbf{g}'(\mathbf{x})$ . At the rest of this subsection, we explain how to compute all solutions of a binomial system  $\mathbf{g}'(\mathbf{x}) = \mathbf{0}$ . These solutions are corresponding to the starting points of the homotopy paths defined by the polyhedral-linear homotopy  $\mathbf{h}(\mathbf{x}, t)$ . The binomial system  $\mathbf{g}'$  of (2.11) consists of  $m$  polynomial systems  $\mathbf{g}'_1, \mathbf{g}'_2, \dots, \mathbf{g}'_m$ , and each polynomial equation  $\mathbf{g}'_i(\mathbf{x}) = \mathbf{0}$  is transformed such that

$$\left\{ \begin{array}{l} \tilde{c}_{i1}(\mathbf{a}_{i0})\mathbf{x}^{\mathbf{a}_{i0}} + \tilde{c}_{i1}(\mathbf{a}_{i1})\mathbf{x}^{\mathbf{a}_{i1}} = 0 \\ \vdots \\ \tilde{c}_{ik_i}(\mathbf{a}_{i0})\mathbf{x}^{\mathbf{a}_{i0}} + \tilde{c}_{ik_i}(\mathbf{a}_{ik_i})\mathbf{x}^{\mathbf{a}_{ik_i}} = 0 \end{array} \right. \iff \left\{ \begin{array}{l} \mathbf{x}^{\mathbf{a}_{i1}-\mathbf{a}_{i0}} = -\frac{\tilde{c}_{i1}(\mathbf{a}_{i0})}{\tilde{c}_{i1}(\mathbf{a}_{i1})} \\ \vdots \\ \mathbf{x}^{\mathbf{a}_{ik_i}-\mathbf{a}_{i0}} = -\frac{\tilde{c}_{ik_i}(\mathbf{a}_{i0})}{\tilde{c}_{ik_i}(\mathbf{a}_{ik_i})} \end{array} \right. \quad \text{for each } i \in M.$$

Let  $\mathbf{v}_{ij} = \mathbf{a}_{ij} - \mathbf{a}_{i0}$  and  $b_{ij} = -\frac{\tilde{c}_{ij}(\mathbf{a}_{i0})}{\tilde{c}_{ij}(\mathbf{a}_{ij})}$  for each  $j \in M$  in the right-hand formula. Then  $\mathbf{g}'(\mathbf{x}) = \mathbf{0}$  is written as

$$\mathbf{g}'(\mathbf{x}) = \mathbf{0} \iff \begin{cases} \mathbf{x}^{v_{11}} = b_{11} \\ \vdots \\ \mathbf{x}^{v_{1k_1}} = b_{1k_1} \\ \vdots \\ \mathbf{x}^{v_{m1}} = b_{m1} \\ \vdots \\ \mathbf{x}^{v_{mk_m}} = b_{mk_m}. \end{cases} \quad (2.13)$$

Let

$$\mathbf{V} = (\overbrace{\mathbf{v}_{11}, \dots, \mathbf{v}_{1k_1}}^{k_1}, \dots, \overbrace{\mathbf{v}_{m1}, \dots, \mathbf{v}_{mk_m}}^{k_m}) \in \mathbb{C}^{n \times n} \quad \text{and}$$

$$\mathbf{b} = (\overbrace{b_{11}, \dots, b_{1k_1}}^{k_1}, \dots, \overbrace{b_{m1}, \dots, b_{mk_m}}^{k_m}) \in \mathbb{C}^n.$$

and by  $\mathbf{x}^{\mathbf{V}}$ , we mean  $(\mathbf{x}^{v_{11}}, \dots, \mathbf{x}^{v_{1k_1}}, \dots, \mathbf{x}^{v_{m1}}, \dots, \mathbf{x}^{v_{mk_m}})$  for convenient. Then, (2.13) is rewritten as

$$\mathbf{x}^{\mathbf{V}} = \mathbf{b}.$$

Let  $\mathbf{x} = \mathbf{y}^{\mathbf{U}}$ , where  $\mathbf{U}$  is a unimodular matrix (i.e., all the elements are integers and its determinant is  $\pm 1$ ) such that  $\mathbf{UV}$  is an upper triangular matrix. Then, we have  $\mathbf{x}^{\mathbf{V}} = \mathbf{y}^{\mathbf{UV}} = \mathbf{b}$ , and the solution  $\mathbf{y}$  can be obtained by forward substitutions. Finally, we obtain the solution of  $\mathbf{g}'(\mathbf{x}) = \mathbf{0}$  from  $\mathbf{x} = \mathbf{y}^{\mathbf{U}}$ . Here, the unimodular matrix  $\mathbf{U}$  so that  $\mathbf{UV}$  is an upper triangular matrix for a given integer matrix  $\mathbf{V}$  can be obtained by a series of elementary (unimodular) column operations in order to construct the Hermit normal form for the matrix  $\mathbf{V}$ . See Chapter 4 and 5 of [48], for example.

From the above discussion, we know that the start system  $\mathbf{g}(\mathbf{x}) = \mathbf{h}(\mathbf{x}, 0) = \mathbf{0}$  has  $|\det(\mathbf{V})|$  solutions. The reason is as follows. The system  $\mathbf{y}^{\mathbf{UV}} = \mathbf{b}$  has  $|\det(\mathbf{UV})|$  solutions because  $\mathbf{UV}$  is an upper triangular matrix. Also,  $\mathbf{U}$  is a unimodular matrix, and hence,  $|\det(\mathbf{UV})| = |\det(\mathbf{U}) \det(\mathbf{V})| = |\det(\mathbf{V})|$ . Therefore,  $\mathbf{g}'(\mathbf{x}) = \mathbf{0}$  has  $|\det(\mathbf{V})|$  solutions. Because  $\mathbf{g}'(\mathbf{x}) = \mathbf{0}$  has the same solution set as  $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ , we are convinced that the above statement is correct. Because  $\mathbf{V}$  is identical to the matrix of (2.12), as we have mentioned in the previous subsection, the total number of the solutions of the start system  $\mathbf{g}(\mathbf{x}) = \mathbf{h}(\mathbf{x}, 0) = \mathbf{0}$  for every  $\mathbf{h} \in \mathcal{H}$  coincides with the mixed volume for  $n$  Newton polytopes  $\text{conv}(\mathcal{A}_i)$  of auxiliary polynomials  $\tilde{f}_i$ .

Summing up, the method for computing all isolated zeros of a system of  $n$  polynomials  $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_n(\mathbf{x}))$  in  $\mathbf{x} \in \mathbb{C}^n$  is considered to consist of two stages. Here, we assume that a polynomial system  $\mathbf{f}$  is a semi-mixed type, and has a  $(k_1, k_2, \dots, k_m)$ -support.

1. Construction stage of a family of polyhedral homotopies:

- Find all mixed cells for  $n$  Newton polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$  of the polynomials  $f_1, f_2, \dots, f_n$  through the lower facets of a lifted polytope  $\hat{\mathcal{P}} = \hat{\mathcal{P}}_1 + \hat{\mathcal{P}}_2 + \dots + \hat{\mathcal{P}}_m$ .

- Construct a family of polyhedral-linear homotopies  $\mathbf{h} \in \mathcal{H}$  using the obtained mixed cells, and compute all solutions of start polynomial systems  $\mathbf{h}(\mathbf{x}, 0) = 0$ .

2. Path following stage:

- Follow all the homotopy paths defined by  $\mathbf{h} \in \mathcal{H}$  by the predictor-corrector methods.

# Chapter 3

## Finding All Mixed Cells in a Fine Mixed Subdivision

### 3.1 Introduction

We begin with the discussion of issues in the polyhedral homotopy method. Remember that the homotopy methods consist of two stages: the construction of a homotopy function and the path following. As we have seen in Chapter 2, the polyhedral homotopy method forces us to enumerate all mixed cells for the Newton polytopes of a polynomial system to construct a family of polyhedral homotopy functions. This enumeration requires much computational cost in general. On the other hand, a homotopy of the classical methods is obtained from the convex combination between a trivial polynomial system and a original polynomial system which we want to solve, and thus the computational cost is almost zero. Although evidently the polyhedral homotopy method supported by Bernshtein's theorem produces much fewer paths than classical ones, it is tough to construct polyhedral homotopies and in particular it becomes pronounced as the size of a polynomial system gets larger. In Subsection 3.2.1, some issues in the polyhedral homotopy method are discussed, compared with the other methods to solve a polynomial system, and it could be clarified that a construction stage of polyhedral homotopies, that is, finding all mixed cells, in the method is the main computational bottleneck. In Subsection 3.2.2, we mention the existing works and software packages sharpening our focus on the construction of a family of homotopies in the polyhedral method.

Our dynamic enumeration method is proposed to conquer the main computational issue in the polyhedral homotopy method. It is based on Li and Li's method presented in Subsection 2.4.2; the existence of each mixed cell is described as the feasibility of the corresponding linear inequality system. The main part of Section 3.3 and 3.4 is dedicated to showing the outline of our dynamic enumeration. In Section 3.3, we see that an enumeration tree is constructed among a family of linear inequality systems induced from Li and Li's method describing mixed cells. The enumeration tree has the following properties.

- (a) A leaf node describes a mixed cell if and only if it is feasible (or more precisely the system of linear inequalities attached to the leaf node is feasible).
- (b) Each mixed cell is corresponding to a unique feasible leaf node.
- (c) Each node different from leaf nodes is a common subsystem of its child nodes, so that if it is infeasible then so are all of its descendant nodes.
- (d) The root node is an empty system, which is always feasible.

We apply an enumeration method to the rooted tree for finding all feasible leaf nodes. If a node is infeasible then so is its descendant nodes. Thus, we can prune the subtree having the node as a root because it does not contain any mixed cells.

There are two important issues in the efficient implementation of an enumeration of all mixed cells which correspond to feasible leaf nodes of such an enumeration tree described above. One is how we construct an enumeration tree, and the other is how we formulate the feasibility of each node. In Section 3.4, we outline the construction procedure of a dynamic enumeration tree, and the formulation for checking the feasibility of each node of the tree is given in Section 3.5. There are various ways of the built of the enumeration trees satisfying the above properties (a)-(d). In the existing methods [61, 38, 57, 22] except for Emiris and Canny's one [17], the structure of an enumeration tree is already determined before starting the enumeration of mixed cells. In such static enumeration tree, we do not utilize any information generated from each node on the execution of the enumeration when a parent node branches into child nodes. This is because the branching rule is fixed in the first place. Our method constructs a dynamic enumeration tree where branching at a node is carried out with the effective use of information generated from nodes at upper level, so that a large portion of its child nodes are infeasible and pruned. In Subsection 3.4.1, we develop the dynamic enumeration for fully mixed supports. As we have stated in Subsection 2.4.1, it is important for the computational efficiency in the implementation of the enumeration procedure by Huber and Stumfels to take account of the special structure of the support if it is a semi-mixed type. In Subsection 3.4.2, we thus extend the method proposed for a fully mixed type in the previous subsection so that it can be carried out efficiently for general semi-mixed supports. We will mention the differences with our dynamic enumeration over Emiris and Canny's one in Subsection 4.2.1, and assess the performance by comparison with the static enumeration method.

The other important issue is the formulation for the feasibility check of each node. The papers [38, 20, 22] formulate the feasibility check of each node as an LP problem which has the linear inequality system, attached to a node, as the constraint, while the paper [57] employs the dual problem. When utilizing information generated on the execution of the enumeration of mixed cells, the dual problem has an advantage. Specifically, we can easily choose a feasible solution of the dual of the "child LP" from an optimal solution of the dual of the "parent LP". The employment of a dual formulation is a key point for developing our dynamic enumeration strategy.

## 3.2 Issue and Related Work

This section is composed of two subsections. In the first subsection, we demonstrate the polyhedral homotopy method and other approaches to polynomial system solving for the well-known benchmark problems. Through the numerical results, we will realize the part of the computational bottleneck in the polyhedral homotopy method. In the second subsection, we summarize the related work and software for the construction of homotopies in the polyhedral method.

### 3.2.1 Discussion on Some Issues in Polyhedral Homotopy Methods

In order to highlight the issues in the polyhedral homotopy, we evaluate the performance of it by comparison with the other approaches to solving a polynomial system. For the purpose of the evaluation, we consider the economic- $n$  problems [44], which is known as the typical benchmark polynomial systems. The problem is a size-expandable system by the number  $n$  and the form is as follows;

$$\begin{aligned}
 x_1 + x_2 + \cdots + x_{n-1} + 1 &= 0, \\
 x_{n-1}x_n - (n-1) &= 0, \\
 (x_{n-2} + x_1x_{n-1})x_n - (n-2) &= 0, \\
 &\vdots \\
 (x_2 + x_1x_3 + \cdots + x_{n-3}x_{n-1})x_n - 2 &= 0, \\
 (x_1 + x_1x_2 + x_2x_3 + \cdots + x_{n-2}x_{n-1})x_n - 1 &= 0.
 \end{aligned} \tag{3.1}$$

First, let us solve the systems by the symbolic methods implemented in the commercial software package Maple. It is a popular computer algebra system, and provides the command “solve” to solve a system of polynomial equations by the symbolic methods. In the majority of cases, a Gröbner base plays an essential role in the symbolic methods and it is necessary for the methods to construct it. The main tool for the construction is the Buchberger algorithm. One may refer to the detail description of solving a polynomial system symbolically; see, for example, in [11, 12, 53]. Table 3.1 shows the CPU time required by Maple on a 2.4 GHz Opteron 850 with 8 GB memory, running Linux. From the table,

Table 3.1: Performance of Maple for the economic- $n$  problems from  $n = 5$  to  $n = 9$  on a 2.4 GHz Opteron 850 with 8 GB memory

size ( $n$ )	economic- $n$				
	$n = 5$	$n = 6$	$n = 7$	$n = 8$	$n = 9$
CPU time	0.14s	0.19s	0.87s	14h10m21.65s	-

we see that the computational time increases sharply at  $n = 8$ . Similarly, in terms of the

memory space, whereas the size of up to  $n = 7$  is less than 10 MB, it reaches about 5.5 GB at  $n = 8$ . At  $n = 9$ , Maple does not work because the amount of memory consumed by the computation is beyond the capacity of computer system. The reason is considered that as the size of a polynomial grows, the magnitude of some coefficients of the polynomials expands drastically in the process of the application of the symbolic operations, and as a result, it requires very large computational cost and memory space.

Next, we solve the same polynomial systems using PHoM, which is the software package, developed by our research group [26], for the implementation of the polyhedral homotopy method. It is coded in C++ language. The numerical experiments were carried out on a 2.4 GHz Pentium IV with 512 MB, running Linux. Note that this computer system is the same or inferior to the previous one. The CPU time for each system is summarized in Table 3.2.

Table 3.2: Performance of PHoM for the economic- $n$  problems from  $n = 5$  to  $n = 9$  on a 2.4 GHz Pentium IV with 512 MB memory

size ( $n$ )	economic- $n$				
	$n = 5$	$n = 6$	$n = 7$	$n = 8$	$n = 9$
CPU time	0.44s	1.92s	5.93s	20.18s	1m6.14s

The table tells us that the computational time increases moderately by comparison with Maple, and PHoM can solve the problems at  $n = 8$  and  $n = 9$  within about 1 minutes. For all problems in the experiments, the memory space required by PHoM is less than 1 MB. The numerical results indicate that the computational costs and memory requirements for the execution of the symbolic methods to utilize Gröbner bases provided by the Buchberger algorithm blow up in contrast to the polyhedral homotopy, as the size of polynomial system gets large. The textbook [11] mentions the reasons for it; (a) some coefficients of polynomial systems, which are elements in a Gröbner basis, can be complicated rational integers, and (b) the degree of intermediate polynomials to be generated as the algorithm proceeds can be huge. Indeed, there exists an example such that the construction of Gröbner basis for polynomials of degree less than or equal to  $d$  produces a polynomial with degree  $2^{2^d}$  as an intermediate in the process of algorithm.

We mention the comparison between symbolic methods and homotopy methods from the two different perspectives; the applicable scope and the quality of obtained solutions. Whereas symbolic methods can be applied to any polynomial systems, homotopy methods has a limitation that a polynomial system consists of  $n$  polynomials with  $n$  unknowns and all the solutions are isolated. Recently, Sommese, Verschelde and Wampler [50, 51, 52] introduced numerical algebraic geometry in which homotopy methods are utilized to describe solution components of polynomial systems. They made a suggestion that homotopy methods can find all positive dimensional solutions of polynomial systems where the number of polynomials do not necessarily coincide with that of variables, and decompose these into irreducible components.

The symbolic methods provide a precise description for a solution set of a system of polynomial equations. Furthermore, we can have rich information related to the solution set as the byproducts produced from a Gröbner basis, which is constructed in the symbolic methods. This is a big difference for the homotopy methods because the methods compute the solutions of a polynomial system under finite digit arithmetic, and thus, the obtained solutions represented by a finite floating-point number are only the approximate values. However, at least, in the situation where one solve a polynomial system arising from the real-world problems, the homotopy methods are considered to be a useful tool. As the textbook [53] points out, the numerical data coming from experiments usually contain some errors, and then, the coefficients of polynomials formulated by use of such empirical data have only a limited accuracy. Then, it may be enough to compute the approximate solutions of these polynomial equations by the homotopy methods.

Finally, we consider the comparison with the classical homotopy methods based on the Bézout theorem. Remember that the total number of the homotopy paths defined by the classical methods is given as the total degree of a polynomial system. Table 3.3 presents the total degree, the mixed volume and the number of all isolated solutions of economic- $n$  polynomial systems. We learn from the table that the mixed volume gives a tight bound

Table 3.3: Total degree, mixed volume and the total number of isolated solutions for the economic- $n$  problems

size ( $n$ )	economic- $n$				
	$n = 5$	$n = 6$	$n = 7$	$n = 8$	$n = 9$
total degree	54	162	486	1,458	4,374
mixed volume	8	16	32	64	128
# isolated solutions	8	16	32	64	128

for the isolated solution count of the economic- $n$  systems, whereas the total degree takes a larger value, compared to it. Of course, there is no guarantee that the mixed volume for a polynomial system coincides with the solution count. For instance, although the reimer-4 polynomial system from a test suite in FRISCO [19] has 36 isolated solutions, the mixed volume is 120 which is the same as the total degree. However, in most cases, the mixed volume provides a sharp upper bound for the solution count of a polynomial system. Therefore, we could expect that the polyhedral homotopy method has much less computational cost in the path following stage than the classical methods.

The tables are turned when we look from a path following stage to a construction stage of a homotopy. In the classical method, we need not spend time for constructing a homotopy function because it is given by a convex combination between a trivial polynomial system and the original polynomial system, as we have seen in Section 2.2. By contrast, the polyhedral homotopy method forces us to solve an enumeration problem for attaining the purpose. The Problem 2.4.3 in Subsection 2.4.2 is one of the formulations for it. As the method for

the construction of a family of polyhedral homotopies, PHoM employs the dual approach for an LP problem forming from the feasibility check of the linear inequality system (2.6). In Table 3.4, we summarize the computational time of PHoM for the economic- $n$  problems from  $n = 10$  to  $n = 13$  on a 2.4 GHz Pentium IV with 512 MB. The first and second

Table 3.4: CPU time of PHoM for the economic- $n$  problems from  $n = 10$  to  $n = 13$  on a 2.4 GHz Pentium IV with 512 MB memory

size ( $n$ )	economic- $n$			
	$n = 10$	$n = 11$	$n = 12$	$n = 13$
mixed volume	256	512	1,024	2,048
# isolated solutions	256	512	1,024	2,048
phase 1	11.2s	1m8.0s	6m58.8s	41m2.9s
phase 2	3m4.5s	9m5.1s	25m18.6s	1h9m28.2s
total	3m15.7s	10m13.1s	32m17.4s	1h50m31.1s
phase 1 / total	5.7%	11.1%	21.6%	37.1%

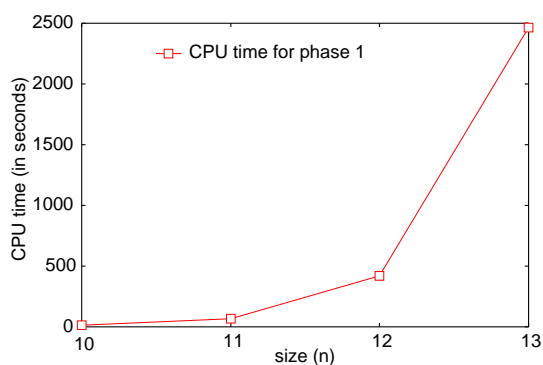


Figure 3.1: CPU time

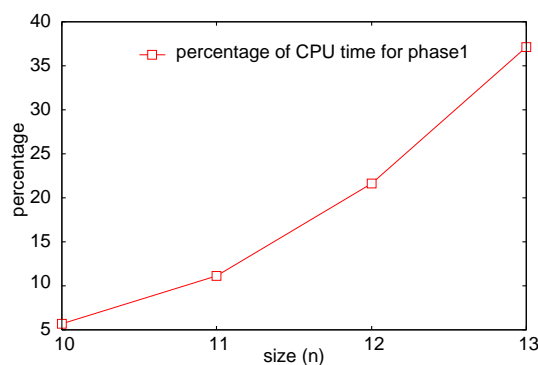


Figure 3.2: Percentage of CPU time

row present the value of mixed volume and the number of all isolated solutions for the corresponding problems. The third row “phase 1” shows the CPU time for constructing polyhedral homotopies and solving the start polynomial systems to obtain the starting points of homotopy paths. Also, the fourth row “phase 2” is the CPU time for following all homotopy paths. The total time for phase 1 and phase 2 is written in the fifth row “total”. The last row “phase 1 / total” indicates the percentage of the CPU time for phase 1 to total. Figure 3.1 and 3.2 are a line graph for CPU time for phase 1 in seconds and percentage of it, respectively. The numerical results bring the fact that as the size of a polynomial system grows, the computational time for solving an enumeration problem to construct a family of polyhedral homotopies increases dramatically, and also the proportion to total denotes the same tendency. Indeed, the construction stage of a homotopy becomes the main computational bottleneck in the polyhedral homotopy method when it challenges

large-scale polynomial system solving. Therefore, in the successive discussion, we propose the methods for solving an enumeration problem efficiently over the existing methods to construct a family of polyhedral homotopies.

### 3.2.2 Related Work and Software

We overview the existing methods and software packages for finding all mixed cells, based on the enumeration procedure by Huber and Sturmfels. In 1995, Emiris and Canny [17] utilized the procedure in order for the computation of sparse resultants. Their method adopts a dynamic enumeration strategy for the efficient implementation of the procedure, and they call it *Lift-Prune method*. The C package MVLP implements their dynamic enumeration method. In Subsection 4.2.1, we will show their method, and compare it with our dynamic enumeration method.

Verschelde proposed the lifting methods, which are summarized in his Ph. D. thesis [61] published in 1996, to increase computational efficiency of the procedure. In the thesis, Huber and Sturmfels's enumeration procedure is called a *static lifting method* because he develops several types of lifting functions; the *dynamic lifting* [62] and *symmetric lifting* [60] methods. However, there may be room for improving these lifting methods. The dynamic lifting method requires the connectivity of mixed cells to achieve the sufficient speed up. According to the paper [36], in case of 2 polytopes, it was proven by Pedersen. So, the dynamic lifting works very well for unmixed polynomial systems. However, it is not necessarily true in general case because Li and Wang [36] encountered the examples whose mixed cells are not connected each other in case of 3 polytopes. The symmetric lifting method needs to choose some specific lifting function in order for preserving the symmetric structure of the supports. But, in general, the lifting function can not guarantee the generality of the lifting value mentioned in Subsection 2.4.2. Hence, the symmetric lifting frequently fails to make a fine mixed subdivision for the support of a general polynomial system. PHCPack [63] is a software package, written by Verschelde using Ada<sup>†</sup> in 1999, for computing all isolated zeros of a system of polynomial equations by the homotopy methods. It contains a module for performing the polyhedral homotopy method. The feature of the software is that it provides several lifting methods for finding all mixed cells, including the static, dynamic and symmetric lifting. Currently, it appears that PHCPack is the most popular software for solving a polynomial system by the polyhedral homotopy method.

In 2001, Li and Li [38] presented a practical formulation, shown in Subsection 2.4.2, through an LP problem. Li and Li's method constructs an enumeration tree implicitly where each leaf node of the tree is corresponding to the linear inequality system of (2.6). Among the leaf nodes of the tree, they enumerate all feasible ones utilizing a bounding (pruning) technique, *one point test*. It is very efficient for narrowing down the candidates for feasible leaf nodes. Note that the tree structure is already fixed before starting the enumeration, called a static tree. As we have seen in Subsection 2.4.1, although it is important for the

---

<sup>†</sup>Gnu Ada compiler (<http://www.gnu.org/software/gnat/gnat.html>)

computational efficiency to take account of the special structure of a support if it is a semi-mixed type, they do not utilize the structure of a semi-mixed support but deal with it as a fully mixed type in the method. The C package `mvol` performs their method.

Takeda, Fujisawa and Kojima [57] employed the dual problem for an LP formulation arisen from the feasibility check of the linear inequality system (2.6) by Li and Li [38] in 2002. Their enumeration has the same spirit as Li and Li's method, and thus, they also build a static enumeration tree. In addition, they proposed a parallel enumeration algorithm (i.e., an implementation of the enumeration in a parallel computer), and showed the high performance in the paper. Thus, it suggests to us that Li and Li's framework for finding all mixed cells fits a parallel computation quite nicely. Again, we will refer to the parallel method in Subsection 5.2.1. The serial enumeration algorithm in [57] was incorporated in the C++ package PHoM, which was published in 2004. The software is the implementation of the polyhedral homotopy method, and in most cases, it performs better than PHCpack in terms of computational time for solving a polynomial system.

Although Li and Li's method is able to treat only fully mixed supports, Gao and Li [22] extended the method to semi-mixed supports in 2003. The framework for the enumeration is almost the same as Li and Li's method [38], i.e., their tree is static. They also developed a powerful tool, *relation table*, for finding many infeasible nodes of an enumeration tree with less effort. The C++ package `MixedVol` [23] was developed for the implementation. By the numerical results in the paper, Gao and Li showed that their method is the most efficient among the other existing methods [17, 60, 61, 62, 38, 57]. Also, the method by Gao and Li was implemented in the FORTRAN package HOM4PS [21] for performing the polyhedral homotopy method. It seems that HOM4PS is faster than PHCpack [63] and PHoM [26] although it becomes unstable, which means that it sometimes returns distinct solution sets for the same polynomial system in each trial, as the size of polynomial systems becomes large. On the other hand, PHoM can solve a polynomial system in relatively stable even if the size grows, whereas it is slower than HOM4PS. One of the reasons is the conservative implementation of the algorithm for tracking homotopy paths. Table 3.5 and 3.6 summarize the existing works and software packages, respectively, in the implementation of Huber and Sturfels's enumeration procedure.

Table 3.5: Existing works

Year	Authors	Methods
1995	Emiris and Canny [17]	dynamic enumeration
1995	Verschelde [60]	symmetric lifting
1996	Verschelde [61, 62]	static and dynamic lifting
2001	Li and Li [38]	static enumeration for a fully mixed type
2002	Takeda, Fujisawa and Kojima [57]	dual approach for Li and Li's static method
2003	Gao and Li [22]	static enumeration for a semi-mixed type

Table 3.6: Existing software packages

Year	Authors	Software	Programming language
1995	Emiris and Canny [17]	MVLP	C
1999	Vershelde [63]	PHCpack	Ada
2001	Li and Li [38]	mvol	C
2002	Gao, Li and Li [21]	HOM4PS	FORTRAN
2003	Gao and Li [23]	MixedVol	C++
2004	Gunji et al. [26]	PHoM	C++

### 3.3 Enumeration Tree

This section is composed of two subsections. In the first subsection, Li and Li's method, described in Subsection 2.4.2, is restated using a tree representation. In the second subsection, the numerical results are shown in order to give the fact that the order for constructing an enumeration tree affects computational time required by finding all mixed cells considerably.

#### 3.3.1 Tree Construction

We give a description of Li and Li's method in terms of a family of linear inequality systems. That is, we define a rooted tree  $T = (V, E)$ , satisfying the properties (a)-(d) in Section 3.1, for describing mixed cells based on their method, and call it an *enumeration tree* for finding all mixed cells. Let  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  ( $\mathcal{A}_i \subseteq \mathbb{Z}_+^n$ ) be a semi-mixed  $(k_1, k_2, \dots, k_m)$ -support of a polynomial system  $\mathbf{f}$ , satisfying  $\sum_{i \in M} k_i = n$ . Note that  $M = \{1, 2, \dots, m\}$ . For every  $L \subseteq M$ , define

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

The set  $\Omega$  serves as candidates of nodes of enumeration trees. Specifically,  $\emptyset^m \in \Omega(\emptyset) = \{\emptyset^m\}$  is the root node, and  $\Omega(M) \subset \Omega$  the leaf nodes. In general, the  $\ell$ th level nodes of an enumeration tree are chosen from  $\cup_{L \subseteq M, \#L=\ell} \Omega(L)$ . For every  $\mathbf{C} \in \Omega$ , let  $L(\mathbf{C}) = \{i \in M : C_i \neq \emptyset\}$  for any  $\mathbf{C} \in \Omega(L)$  ( $L \subseteq M$ ). This definition is used for extracting every index of nonempty sets  $C_i$  from  $\mathbf{C} = (C_1, C_2, \dots, C_m) \in \Omega(L)$ , i.e.,  $L(\mathbf{C}) = L$  for  $\mathbf{C} \in \Omega(L)$  ( $L \subseteq M$ ).

We choose a generic lifting function  $\omega_i : \mathcal{A}_i \rightarrow \mathbb{R}$  to construct a fine mixed subdivision for the semi-mixed support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$ . To achieve the generality of the lifting value  $\omega_i(\mathbf{a})$  ( $\mathbf{a} \in \mathcal{A}_i$ ), for every  $i \in M$  and every  $\mathbf{a} \in \mathcal{A}_i$ , let  $\omega_i(\mathbf{a})$  be a random number chosen from some bounded interval of  $\mathbb{R}$ . For every  $\mathbf{C} = (C_i : i \in L) \in \Omega(L)$  with  $L \subseteq M$ ,

we consider a linear inequality system in a variable vector  $\boldsymbol{\alpha} \in \mathbb{R}^n$

$$\mathcal{I}(\mathbf{C}) := \begin{cases} \langle \mathbf{a}_p - \mathbf{a}_q, \boldsymbol{\alpha} \rangle = \omega_i(\mathbf{a}_q) - \omega_i(\mathbf{a}_p), & \forall \mathbf{a}_p, \mathbf{a}_q \in C_i \\ \langle \mathbf{a}_p - \mathbf{a}, \boldsymbol{\alpha} \rangle \leq \omega_i(\mathbf{a}) - \omega_i(\mathbf{a}_p), & \forall \mathbf{a} \in \mathcal{A}_i \setminus C_i \end{cases} \quad (i \in L(\mathbf{C})).$$

For each  $\mathbf{C} \in \Omega(M)$ ,  $\mathcal{I}(\mathbf{C})$  is obtained from the linear inequality system  $\mathcal{E}(\mathbf{C})$  in the Problem 2.4.3 by eliminating variables  $\beta_i$  ( $i \in M$ ). We say that  $\mathbf{C} \in \Omega$  is feasible when  $\mathcal{I}(\mathbf{C})$  is feasible. Let

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

Then, we know from the Problem 2.4.3 that  $\Omega^*$  forms the set of all mixed cells. Note that a leaf node  $\mathbf{C} \in \Omega(M)$  is a mixed cell if and only if  $\mathbf{C}$  is feasible. Also, the root node  $\emptyset^m \in \Omega(\emptyset)$  is defined to be a feasible node. Thus, properties (a), (b) and (d) which are described in Section 3.1, are satisfied.

For  $L \subseteq M$ , by  $\mathbf{C}_L = (C_i : i \in L)$ , we denote the vector consisting of  $C_i$  ( $i \in L$ ). For every  $\mathbf{C} \in \Omega$  with a proper subset  $L(\mathbf{C})$  of  $M$  and every  $t \in M \setminus L(\mathbf{C})$ , define a set of child nodes of  $\mathbf{C}$  by

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

We can build an enumeration tree if we successively choose  $t \in M \setminus L(\mathbf{C})$  at each node  $\mathbf{C}$  of the tree starting from the root node  $\mathbf{C} = \emptyset^m$  with  $L(\mathbf{C}) = \emptyset$  (see Algorithm 3.3.1 for more details).

In the *static enumeration method* employed in the papers [20, 22, 38, 57], we first choose a permutation of  $M$  or a one-to-one mapping  $\pi : M \rightarrow M$ , and restrict nodes of enumeration trees to  $\mathbf{C} \in \Omega(\{\pi(1), \pi(2), \dots, \pi(m)\})$ . Note that the root node is a feasible node  $\emptyset^m \in \Omega(\emptyset)$ . First, the static enumeration method constructs the set  $W(\emptyset^m, \pi(1))$  as child nodes of the root node  $\emptyset^m \in \Omega(\emptyset)$ . Next, if a node  $\mathbf{C} \in \Omega(\{\pi(1), \pi(2), \dots, \pi(\ell)\})$  for some  $1 \leq \ell < m$  has been found to be feasible, the static enumeration generates  $W(\mathbf{C}, \pi(\ell+1))$  as the set of child nodes of  $\mathbf{C}$ . Thus the structure of the static enumeration tree is completely determined by a permutation  $\pi : M \rightarrow M$ .

In the enumeration method which we propose, an enumeration tree is constructed dynamically as the enumeration of nodes proceeds. To explain a procedure for the construction of such a tree, we define some notation. Let  $T = (V, E)$  be a rooted tree such that the vertex set  $V$  and edge set  $E$  are written as  $V = \bigcup_{\ell=0}^m V_\ell$  and  $E = \bigcup_{\ell=0}^m E_\ell$ .  $V_0$  corresponds to the root node  $\emptyset^m$ , and we define  $E_0$  as an empty set for consistency with discussions below. The procedure for construction of a tree  $T$  with allocating nodes dynamically is written as follows. Here, we call  $m$ -tuple  $(k_1, k_2, \dots, k_m)$  a *multiplicity of the support*  $\mathcal{A}$  for a semi-mixed  $k := (k_1, k_2, \dots, k_m)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$ .

**Algorithm 3.3.1.** (Construction of a tree  $T = (V, E)$  for finding all mixed cells)

*Input:* A support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  and its multiplicity  $k = (k_1, k_2, \dots, k_m)$ .

*Output:* A tree  $T = (V = \bigcup_{\ell=0}^m V_\ell, E = \bigcup_{\ell=0}^m E_\ell)$ .

$V_\ell \leftarrow \emptyset^m$  ( $\ell = 0, \dots, m$ ),  $E_\ell \leftarrow \emptyset$  ( $\ell = 0, \dots, m$ ) and  $\ell \leftarrow 0$ .

**while**  $\ell < m$  **do**

**for all**  $\mathbf{C} \in V_\ell$  **do**

    Choose  $t$  from  $M \setminus L(\mathbf{C})$ .

$V_{\ell+1} \leftarrow V_{\ell+1} \cup W(\mathbf{C}, t)$ , and

$E_{\ell+1} \leftarrow E_{\ell+1} \cup \{(\mathbf{C}, \bar{\mathbf{C}}) \in V_\ell \times V_{\ell+1} : \bar{\mathbf{C}} \in W(\mathbf{C}, t)\}$ .

**end for**

$\ell \leftarrow \ell + 1$ .

**end while**

If two nodes  $\mathbf{C} \in V_\ell$  and  $\bar{\mathbf{C}} \in V_{\ell+1}$  of a tree are joined with an edge, we say that  $\bar{\mathbf{C}}$  is a *child node* of the *parent node*  $\mathbf{C}$ . Any node on all paths from  $\mathbf{C}$  to reachable leaf nodes, which are elements in  $V_n$ , is said to be a *descendant node* of  $\mathbf{C}$ . Conversely, any node on all paths from  $\mathbf{C}$  to the root is a *ancestor node* of  $\mathbf{C}$ . Each feasible leaf node is corresponding to mixed cells. Let  $T = (V, E)$  be an enumeration tree generated by Algorithm 3.3.1. If a node  $\mathbf{C} \in \Omega(L)$  with  $L \subsetneq M$  of the tree  $T$  is infeasible, then, all descendant nodes  $\bar{\mathbf{C}}$  of  $\mathbf{C}$  are infeasible because  $\mathcal{I}(\bar{\mathbf{C}})$  have all linear inequality constraints of  $\mathcal{I}(\mathbf{C})$ . Thus, we confirm that the enumeration trees produced by the algorithm satisfy all properties (a)-(d).

For a given semi-mixed  $(k_1, k_2, \dots, k_m)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$ , the algorithm produces various types of enumeration trees  $T$  depending on a choice of an index  $t$  from  $M \setminus L(\mathbf{C})$ . For instance, a static enumeration tree can be built if we set an index  $t$  as  $\pi(\ell+1)$  for each node  $\mathbf{C} \in V_\ell$  using a permutation  $\pi$  of  $M$ . In our dynamic construction of a tree, we find an appropriate index  $t$  among  $M \setminus L(\mathbf{C})$  with a reasonable effort. Suppose that a node  $\mathbf{C}$  with some proper subset  $L(\mathbf{C})$  of  $M$  has been found to be feasible. Then we try to choose  $t \in M \setminus L(\mathbf{C})$  so that only a small portion of its child nodes  $W(\mathbf{C}, t)$  are expected to be feasible. The important issue here is how inexpensively we estimate the number of feasible child nodes in  $W(\mathbf{C}, s)$  for all  $s \in M \setminus L(\mathbf{C})$ . For this purpose, we propose a simple technique of feasibility check in Subsection 4.2.2, which applies a criterion of unboundedness detection in the simplex method.

### 3.3.2 Effect of Tree Construction Order

We observe how the order for constructing an enumeration tree in the static method has an effect on computational time through the economic- $n$  problems. Remember that the form of the polynomial systems has been shown in Subsection 3.2.1. In the form, let us set the

support of the  $i$ th polynomial from the top as  $\mathcal{A}_i$ . Then, the cardinality of each support is as follows:

$$\#\mathcal{A}_1 = n, \quad \#\mathcal{A}_2 = 2, \quad \#\mathcal{A}_3 = 3, \quad \cdots \quad \#\mathcal{A}_{n-1} = n - 1 \quad \text{and} \quad \#\mathcal{A}_n = n.$$

We consider two types of input data  $\mathcal{A}$  of Algorithm 3.3.1. As stated below, one is a sorted support in ascending order of the size of support, say type A, and to the contrary, another is in descending order, say type B.

$$\begin{aligned} \text{type A: } \mathcal{A} &= (\mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \dots, \mathcal{A}_n, \mathcal{A}_1) \\ \text{type B: } \mathcal{A} &= (\mathcal{A}_1, \mathcal{A}_n, \mathcal{A}_{n-1}, \dots, \mathcal{A}_3, \mathcal{A}_2). \end{aligned}$$

So, the algorithm with type A data generates an enumeration tree so that fewer nodes are allocated at the upper level compared to type B. We show computational time consumed by the static enumeration for type A and type B in Table 3.7. The detail description of static enumeration method will be given in the next two subsections. The numerical experiment for each polynomial system was carried out 10 times using different lifting values. The column “# mixed cells” presents the average number of mixed cells and “CPU time”

Table 3.7: Comparison of computational time on type A and B

economic- $n$	type A		type B		ratio
	# mixed cells	CPU time	# mixed cells	CPU time	
$n = 12$	350	1.8s	369	1m29.4s	50.0
$n = 13$	639	7.0s	623	8m50.6s	75.3
$n = 14$	1,160	29.6s	1,135	18m55.3s	104.5
$n = 15$	2,141	1m57.7s	2,158	5h3m3.5s	154.4

the average CPU time for finding mixed cells by the static enumeration. From the table, we see that the static enumeration for type A is much faster than type B. The economic- $n$  systems have the feature that each support consists of different number of elements and then, can be sorted in the size. Therefore, the main reason why it causes such a remarkable difference in computational time for the economic- $n$  systems is considered to be the order for the construction of an enumeration tree. Then, the observation raises the following question; to reduce computational cost for enumerating all mixed cells substantially, how do we construct an enumeration tree for general polynomial systems; for example, every support has the same number of elements. In the successive discussion, we will look for the answers to the question.

### 3.4 Dynamic Enumeration

This section is composed of three subsections. The outline of the dynamic enumeration of all mixed cells is presented in the first subsection. To carry out the method efficiently, it needs

bounding techniques for reducing the computational task, which increases dramatically as the size of the input data (i.e., supports) for the method grows. The bounding techniques are described in the following succeeding two subsections; in the first subsection, we give an explanation for the techniques in case where the input data for the method is the fully mixed supports, and the semi-mixed case is considered in the second subsection. As we mention in Subsection 3.2.2, Emiris and Canny [17] proposed another dynamic strategy for the enumeration of all mixed cells. In the third subsection, we look at their method, and discuss the difference between our dynamic enumeration strategy and their one.

### 3.4.1 Fully Mixed Type

We consider a semi-mixed  $(k_1, k_2, \dots, k_m)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  of a semi-mixed polynomial system  $\mathbf{f}(\mathbf{x})$  in  $\mathbf{x} \in \mathbb{C}^n$ , where  $\sum_{i \in M} k_i = n$  for  $M = \{1, 2, \dots, m\}$ . Note that  $\mathcal{A}_i$  is a subset of  $\mathbb{Z}_+^n$ . We call  $m$ -tuple  $k := (k_1, k_2, \dots, k_m)$  a *multiplicity of the support*  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$ . For the given support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  and its multiplicity  $k = (k_1, k_2, \dots, k_m)$ , the Algorithm 3.3.1 generates a tree  $T = (V, E)$ . We search for feasible leaf nodes of the tree  $T$ , followed by the dynamic enumeration algorithm as shown below. To describe the algorithm, we introduce the set  $W^*(\mathbf{C}, t)$ . This is a collection of every feasible node in  $W(\mathbf{C}, t)$  for an element  $\mathbf{C}$  in the node set  $\Omega$  of  $T$  and an index  $t \in M \setminus L(\mathbf{C})$ . That is,

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

The use of the word “list” simplifies the description of the algorithm. The word “list” is used as follows. For a finite set  $A$ , we denote  $\text{list}(A)$  as an ordered sequence of the elements in  $A$ , where the actual order is not relevant in our succeeding discussions but it is fixed. For a pair of  $\text{list}(A)$  and  $\text{list}(B)$ , where  $A$  and  $B$  are finite sets,  $\text{list}(A) + \text{list}(B)$  stands for the list which is generated by connecting  $\text{list}(B)$  with  $\text{list}(A)$  by “stacking”  $\text{list}(B)$  on  $\text{list}(A)$ ; for example, if  $\text{list}(A) = (a, b, c)$  and  $\text{list}(B) = (d, e)$ , then  $\text{list}(A) + \text{list}(B) = (a, b, c, d, e)$ .

The dynamic enumeration algorithm is described as follows. In the algorithm,  $\mathcal{N}$  is used for the set of nodes generated by the algorithm, and it is initialized as a root node  $\emptyset^n$ . We stack the number of child nodes, created by a parent node, on  $\nu$  at each step, and  $\nu^*$  represents the total number of nodes generated by the algorithm when the algorithm terminates. Therefore, the total amount of works to generate all mixed cells by the algorithm is measured by  $\nu^*$ . In Subsection 4.2.3, we will show the performance of each static and dynamic enumeration method by  $\nu^*$ .

**Algorithm 3.4.1.** (Dynamic enumeration algorithm)

*Input:* A support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  and its multiplicity  $k = (k_1, k_2, \dots, k_m)$ .

*Output:* All mixed cells  $\mathbf{C} \in \Omega^*$  and the total number  $\nu^*$  of nodes generated.

Lift  $\mathcal{A}_i$  to  $\hat{\mathcal{A}}_i$  using a generic lifting  $\omega_i$  for every  $i \in M$ .  
 $\text{list}(\mathcal{N}) \leftarrow \emptyset^m$  and  $\nu \leftarrow 1$ .

**while**  $\#\mathcal{N} \neq 0$  **do**

    Take out the last element  $\mathbf{C}$  of  $\text{list}(\mathcal{N})$  and remove  $\mathbf{C}$  from  $\text{list}(\mathcal{N})$ .

**if**  $L(\mathbf{C}) = M$  **then**

**output**  $\mathbf{C}$  as a mixed cell.

**else**

        (A): Choose  $t$  from  $M \setminus L(\mathbf{C})$  and  $\nu \leftarrow \nu + \#W(\mathbf{C}, t)$ .

        (B):  $\text{list}(\mathcal{N}) \leftarrow \text{list}(\mathcal{N}) + \text{list}(W^*(\mathbf{C}, t))$ .

**end if**

**end while**

$\nu^* \leftarrow \nu$ .

**return**  $\nu^*$  as the total number of nodes of the constructed tree.

Ideally, we would like to choose  $t \in M \setminus L(\mathbf{C})$  at (A) so that the size of  $W^*(\mathbf{C}, t)$  is the smallest among the sizes of  $W^*(\mathbf{C}, s)$  ( $s \in N \setminus L(\mathbf{C})$ ). Roughly speaking, as we see in Subsection 4.3.1, this idea is employed by Emiris and Canny in [17] although their formulation for checking the existence of mixed cells is different from Li and Li's one. However, it is costly because every  $W^*(\mathbf{C}, s)$  ( $s \in N \setminus L(\mathbf{C})$ ) need to be constructed. Indeed, the numerical results in [38, 57, 22] show that their strategy has a disadvantage in speed. Also, we will present the comparison with computational time of our dynamic enumeration method in Subsection 4.3.1. Accordingly, instead of  $W^*(\mathbf{C}, t)$ , we consider another set which can be constructed easily. In Subsection 4.2.2, we explain how to construct this set.

For constructing  $W^*(\mathbf{C}, t)$  at (B), we need to check the feasibility for every node in  $W(\mathbf{C}, t)$ . To restrict the number of the feasibility checks, we use a bounding technique, so-called *one point test* proposed by [38, 22]. This test is known to be very effective to increase the computational efficiency of enumeration. In this subsection, for simplicity of explanation of the technique, we restrict the input data of the algorithm to fully mixed supports. For a fully mixed  $(1, 1, \dots, 1)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n)$ , let us carry out the above algorithm. Suppose that  $\mathbf{C} \in \mathcal{N}$  and  $t \in N \setminus L(\mathbf{C})$ . For every  $\mathbf{a} \in \mathcal{A}_t$ , the one point test checks feasibility of the system of linear inequalities in  $\boldsymbol{\alpha} \in \mathbb{R}^n$

$$\mathcal{I}(\mathbf{C}, t, \mathbf{a}) := \begin{cases} \mathcal{I}(\mathbf{C}), \\ \langle \mathbf{a} - \mathbf{b}, \boldsymbol{\alpha} \rangle \leq \omega_t(\mathbf{b}) - \omega_t(\mathbf{a}), \quad \forall \mathbf{b} \in \mathcal{A}_t \setminus \{\mathbf{a}\} \end{cases}$$

If  $\mathcal{I}(\mathbf{C}, t, \mathbf{a})$  is infeasible, we can remove  $\bar{\mathbf{C}}$  with  $\bar{\mathbf{C}}_t = \{\mathbf{a}, \mathbf{a}'\}$  for any  $\mathbf{a}' \in \mathcal{A}_t \setminus \{\mathbf{a}\}$  from

$W(\mathbf{C}, t)$  because  $\mathcal{I}(\bar{\mathbf{C}})$  is also infeasible. Therefore, as feasible node candidates, we only consider

$$W_1^*(\mathbf{C}, t) = \{\bar{\mathbf{C}} \in \Omega(L(\mathbf{C}) \cup \{t\}) : \bar{\mathbf{C}}_L = \mathbf{C}_L \text{ and } \bar{\mathbf{C}}_t \subseteq \mathcal{A}_t(\mathbf{C})\}, \quad (3.2)$$

where

$$\mathcal{A}_t(\mathbf{C}) = \{\mathbf{a} \in \mathcal{A}_t : \mathcal{I}(\mathbf{C}, t, \mathbf{a}) \text{ is feasible}\}.$$

After  $r_t (= \#\mathcal{A}_t)$  feasibility checks of linear inequality systems  $\mathcal{I}(\mathbf{C}, t, \mathbf{a})$ , we have  $W_1^*(\mathbf{C}, t)$  satisfying

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

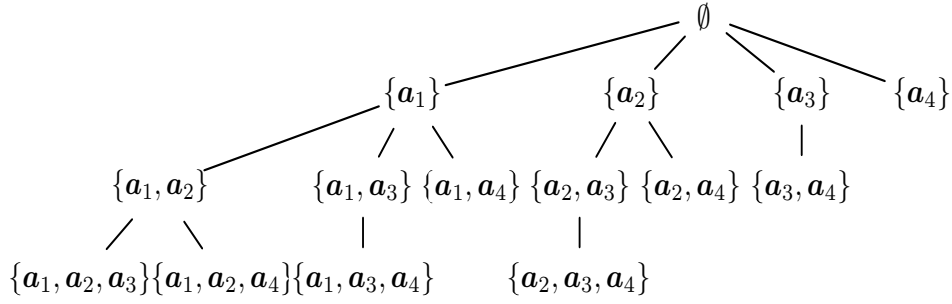
At (B) of Algorithm 3.4.1, we first carry out one point test and construct  $W_1^*(\mathbf{C}, t)$ , and next check the feasibility of each node in the set in order for the construction of  $W^*(\mathbf{C}, t)$ .

### 3.4.2 Semi-Mixed Type

We consider to carry out Algorithm 3.4.1 for semi-mixed supports. In this case, it would waste a lot of time for constructing  $W^*(\mathbf{C}, t)$  for some  $\mathbf{C}$  and some  $t \in M \setminus L(\mathbf{C})$  because the size of  $W(\mathbf{C}, t)$ , which is a set of all candidate nodes for the construction of  $W^*(\mathbf{C}, t)$ , is not small when the semi-mixed supports are given as the input data of the algorithm. In the previous subsection, as a bounding technique for the efficient construction of  $W^*(\mathbf{C}, t)$  in the algorithm, the one point test was introduced for fully mixed supports. In this subsection, to extend it to a semi-mixed type, we construct an enumeration tree  $U = (V, E)$ , which has almost the same properties as  $T$  generated by Algorithm 3.3.1; a candidate node  $\bar{\mathbf{C}} \in W(\mathbf{C}, t)$  is corresponding to the leaf node having the linear inequality system  $\mathcal{I}(\bar{\mathbf{C}})$ , and there is a common linear inequality system between a node different from leaf nodes and its child nodes. Thus, if a node is infeasible, then a subtrees having the node as a root is pruned because the subtree does not have any elements in  $W^*(\mathbf{C}, t)$ . As we prune such worthless subtrees, we search for feasible leaf nodes of  $U$ .

Let us start with a simple example in an easy-to-understand manner for the construction of an enumeration tree  $U$ . Consider the semi-mixed  $(1, 2)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$  ( $\mathcal{A}_i \subseteq \mathbb{Z}_+^3$ ) such that  $\#\mathcal{A}_1 = 3$  and  $\#\mathcal{A}_2 = 4$ . We carry out Algorithm 3.4.1 for the support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ . Suppose that the algorithm takes  $t = 1$  from the index set  $M = \{1, 2\}$  at (A), and produces a nonempty set  $\mathcal{N}$  at (B) in the first iteration. In the next iteration, the remaining index  $t = 2$  is chosen. Then, in order to construct  $W^*(\mathbf{C}, 2)$  for  $\mathbf{C} \in \mathcal{N}$ , we need to check the feasibility of 4 elements in  $W(\mathbf{C}, 2)$ . Figure 3.3 shows the shape of the tree  $U$ , which represents each element in  $W(\mathbf{C}, 2)$  as leaf nodes at the third level. In the figure, for  $\mathcal{A}_2 = \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4\}$ , each node is labeled the corresponding subset of  $\mathcal{A}_2$ . We show the detailed construction procedure of the tree  $U = (V, E)$  as follows.

We consider a semi-mixed  $(k_1, k_2, \dots, k_m)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$ , and apply Algorithm 3.4.1 to the support. The algorithm stores each feasible node  $\mathbf{C} \in \Omega(L)$  for  $L \subseteq M$  in  $\mathcal{N}$ . Suppose that we choose an index  $t \in M \setminus L(\mathbf{C})$  in (A) of the algorithm for


 Figure 3.3: Tree structure for representing every element in  $W(\mathbf{C}, 2)$ 

some element  $\mathbf{C} \in \Omega(L)$  ( $L \subsetneq M$ ) of  $\mathcal{N}$ . To find every feasible node in the child node set  $W(\mathbf{C}, t)$  efficiently, we built a tree  $U = (V, E)$  with a vertex set  $V$  and edge set  $E$ , which represents each element in  $W(\mathbf{C}, t)$  as the leaf node. Let  $K_t = \{0, 1, \dots, k_t + 1\}$ . For a nonnegative integer  $\ell$ , let

$$\Gamma(\ell; \mathbf{C}, t) = \left\{ \mathbf{S} = (S_1, S_2, \dots, S_m) : \begin{array}{l} \mathbf{S}_{L(\mathbf{C})} = \mathbf{C}_{L(\mathbf{C})} \\ S_t \subseteq \mathcal{A}_t, \#S_t = \ell \\ S_i = \emptyset \ (i \notin L(\mathbf{C}) \cup \{t\}) \end{array} \right\}.$$

$$\Gamma = \cup_{\ell \in K_t} \Gamma(\ell; \mathbf{C}, t)$$

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

To describe the tree structure more precisely, we define a function  $q_t$  and the child set  $H(\mathbf{S}; \mathbf{C}, t)$  of  $\mathbf{C} \in \Gamma$ . For  $S_t \subseteq \mathcal{A}_t$ , we choose  $q_t : \mathcal{A}_t \rightarrow \mathbb{Z}$  which gives the maximum number  $i$  among the indices of  $\mathbf{a}_i \in S_t$ . Namely,  $q_t(S_t) = \max\{i \in \mathbb{Z} : \mathbf{a}_i \in S_t\}$  for  $S_t \subseteq \mathcal{A}_t$ . Recall that  $\mathcal{A}_t$  has  $r_t$  elements. For a node  $\mathbf{S} \in \Gamma(\ell; \mathbf{C}, t)$  with  $\ell < k_t + 1$ , let

$$H(\mathbf{S}; \mathbf{C}, t) = \left\{ \bar{\mathbf{S}} \in \Gamma(\#S_t + 1; \mathbf{C}, t) : \begin{array}{l} \bar{S}_t = S_t \cup \{\mathbf{a}_i\}, \\ \text{for every } i, q_t(S_t) < i \leq r_t \end{array} \right\}.$$

The following algorithm describes the construction procedure for a tree  $U = (V, E)$  with  $V = \cup_{\ell \in K_t} V_\ell$  and  $E = \cup_{\ell \in K_t} E_\ell$ .

**Algorithm 3.4.2.** (Construction of a tree  $U = (V, E)$  for describing all elements in  $W(\mathbf{C}, t)$ )

*Input:* A feasible node  $\mathbf{C} \in \Omega(L)$  and an index  $t \in M \setminus L(\mathbf{C})$ .

*Output:*  $U = (V, E)$  with  $V = \bigcup_{\ell \in K_t} V_\ell$  and  $E = \bigcup_{\ell \in K_t} E_\ell$ .

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

```

while  $\ell < k_t + 1$  do
  for all  $\mathbf{S} \in V_\ell$  do
    if  $q_t(\mathbf{S}_t) < r_t$  then
       $V_{\ell+1} \leftarrow V_{\ell+1} \cup H(\mathbf{S}; \mathbf{C}, t)$ 
       $E_{\ell+1} \leftarrow E_{\ell+1} \cup \{(\mathbf{S}, \bar{\mathbf{S}}) \in V_\ell \times V_{\ell+1} : \bar{\mathbf{S}} \in H(\mathbf{S}; \mathbf{C}, t)\}$ 
    end if
  end for
   $\ell \leftarrow \ell + 1$ .
end while

```

The root node of  $U = (V, E)$ , constructed by the algorithm, corresponds to  $V_0 = \{\mathbf{C}\}$ , where  $\mathbf{C}$  is one of the elements in  $\mathcal{N}$  generated by Algorithm 3.4.1. When this algorithm terminates,  $V_\ell$  stores every element in  $\Gamma(\ell; \mathbf{C}, t)$ . Hence,  $W(\mathbf{C}, t)$ , which is generated by a feasible node  $\mathbf{C} \in \mathcal{N}$  and an index  $t \in M \setminus L(\mathbf{C})$  in Algorithm 3.4.1, is equal to  $V_{k_t+1}$ . Note that the structure of this enumeration tree  $U = (V, E)$  does not depend on the process of the algorithm and, that is, it is determined when the input data is given. We see that checking the feasibility of each node at the first level is corresponding to the one point test, proposed in the previous subsection.

For the example described at the beginning of this subsection, we confirm that the algorithm generates the same enumeration tree as Figure 3.3 when we applies it to the example. In the figure, each label of the nodes corresponds to  $S_2 \subseteq \mathcal{A}_2$  of  $\mathbf{S} = (S_1, S_2) \in \Gamma = \bigcup_{\ell \in \{1,2,3\}} \Gamma(\ell, \mathbf{C}, 2)$  for  $\mathbf{C} = (C_1, C_2)$  where the subset  $C_1$  of  $\mathcal{A}_1$  has 2 elements and  $C_2$  is an empty set. Because a node  $\mathbf{S} \in V_\ell$  has a linear inequality system  $\mathcal{I}(\mathbf{S})$ ,  $W^*(\mathbf{C}, t)$  can be obtained if we check the feasibility of each node  $\mathbf{S} \in V_{k_t+1}$ .

Let  $\mathbf{S} \in V_\ell$  with  $\ell < k_t + 1$  be infeasible. Then, all descendant nodes  $\bar{\mathbf{S}} \in \Gamma$  of  $\mathbf{S}$  are infeasible. This is because the feasible region of  $\mathcal{I}(\mathbf{S})$  contains that of  $\mathcal{I}(\bar{\mathbf{S}})$ . Therefore, the subtree having the node  $\mathbf{S}$  as a root can be pruned, since it does not contain any feasible leaf nodes at the  $(k_t + 1)$ th level. Taking account of this property, we enumerate each feasible node in  $V_{k_t+1}$ . To describe the enumeration algorithm, we use the notation  $H^*(\mathbf{S}; \mathbf{C}, t)$  for the set of every feasible node in  $H(\mathbf{S}; \mathbf{C}, t)$ , i.e.,

$$H^*(\mathbf{S}; \mathbf{C}, t) = \{\bar{\mathbf{S}} \in H(\mathbf{S}; \mathbf{C}, t) : \bar{\mathbf{S}} \text{ is feasible}\} \subseteq H(\mathbf{S}; \mathbf{C}, t).$$

The enumeration algorithm is written as follows. Here,  $\mathcal{N}$  is used as a set of nodes, initialized as a feasible node  $\mathbf{C}$ . When the algorithm terminates, all element in  $W^*(\mathbf{C}, t)$  can be obtained. Also, the algorithm is corresponding to the one point test, shown in the previous subsection, when  $k_t = 1$ . Although, in the algorithm, we need to test whether each node in  $H(\mathbf{S}; \mathbf{C}, t)$  is feasible or not, the formulation for checking the feasibility of nodes will be given in the next subsection.

**Algorithm 3.4.3.** (Enumeration algorithm for finding all elements in  $W^*(\mathbf{C}, t)$ )

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

*Output:* All elements in  $W^*(\mathbf{C}, t)$ .

list( $\mathcal{N}$ )  $\leftarrow$   $\mathbf{C}$ .

**while** # $\mathcal{N} \neq 0$  **do**

    Take out the last element  $\mathbf{S}$  of list( $\mathcal{N}$ ) and remove  $\mathbf{S}$  from list( $\mathcal{N}$ ).

**if** # $S_t = k_t + 1$  **then**

**output**  $\mathbf{S}$  as an element in  $W^*(\mathbf{C}, t)$

**else**

**if**  $q_t(S_t) < r_t$  **then**

            list( $\mathcal{N}$ )  $\leftarrow$  list( $\mathcal{N}$ ) + list( $H^*(\mathbf{S}; \mathbf{C}, t)$ ).

**end if**

**end if**

**end while**

## 3.5 Feasibility Check

Let  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  ( $\mathcal{A}_i \subseteq \mathbb{Z}_+^n$ ) be a semi-mixed  $(k_1, k_2, \dots, k_m)$ -support, satisfying  $\sum_{i \in M} k_i = n$ . We apply the dynamic enumeration algorithm (i.e., Algorithm 3.4.1) to the support  $\mathcal{A} = (\mathcal{A}_i : i \in M)$ . Suppose that an index  $t_0$  is chosen among  $M \setminus L(\mathbf{C}_0)$  for some specific element  $\mathbf{C}_0 \in \Omega(L)$  ( $L \subsetneq M$ ) in  $\mathcal{N}$ . Then, we need to construct  $W^*(\mathbf{C}_0, t_0)$  at (B) of the algorithm. To find all elements in the set, an enumeration tree  $U = (V, E)$  is built by Algorithm 3.4.2, and we search for the leaf nodes at the  $(k_t + 1)$ th level as we check the feasibility of each node, based on Algorithm 3.4.3. In this section, we give a formulation for checking the feasibility of each node of  $U$  by Algorithm 3.4.2. We formulate the feasibility check for each node  $\mathbf{C} \in \Gamma = \cup_{\ell \in K_t} \Gamma(\ell, \mathbf{C}_0, t_0)$  of  $U$  through an LP problem. Let  $\boldsymbol{\gamma}$  denote a specific  $n$ -dimensional real vector. To determine whether a node  $\mathbf{C} \in \Gamma$  is feasible or not, we test the feasibility of the following problem in a variable vector  $\boldsymbol{\alpha} \in \mathbb{R}^n$ :

$$P(\mathbf{C}) : \quad \text{maximize } \langle \boldsymbol{\gamma}, \boldsymbol{\alpha} \rangle \quad \text{subject to } \mathcal{I}(\mathbf{C}).$$

Let  $\mathbf{a}_i$  be an element in  $C_i$  for any  $i \in L(\mathbf{C})$ . The dual problem in a variable vector  $\mathbf{x} \in \mathbb{R}^d$ , whose definition will be shown in (3.3), is written as

$$D(\mathbf{C}) : \begin{cases} \text{minimize} & \Phi(\mathbf{x}; \mathbf{C}) \\ \text{subject to} & \Psi(\mathbf{x}; \mathbf{C}) = \gamma, \\ & -\infty < x_{\mathbf{b}} < +\infty \quad \mathbf{b} \in C_i \setminus \{\mathbf{a}_i\} \\ & x_{\mathbf{b}'} \geq 0 \quad \mathbf{b}' \in \mathcal{A}_i \setminus C_i \quad (i \in L(\mathbf{C})). \end{cases}$$

Here, the linear functions  $\Phi(\mathbf{x}; \mathbf{C})$  and  $\Psi(\mathbf{x}; \mathbf{C})$  in  $\mathbf{x} \in \mathbb{R}^d$  are defined by

$$\begin{aligned} \Phi(\mathbf{x}; \mathbf{C}) &= \sum_{i \in L(\mathbf{C})} \sum_{\mathbf{a} \in \mathcal{A}_i \setminus \{\mathbf{a}_i\}} (\omega_i(\mathbf{a}) - \omega_i(\mathbf{a}_i)) x_{\mathbf{a}} \\ \text{and } \Psi(\mathbf{x}; \mathbf{C}) &= \sum_{i \in L(\mathbf{C})} \sum_{\mathbf{a} \in \mathcal{A}_i \setminus \{\mathbf{a}_i\}} (\mathbf{a}_i - \mathbf{a}) x_{\mathbf{a}}, \end{aligned}$$

and the variable vector is given by the column vector

$$\mathbf{x} = (x_{\mathbf{a}} : \mathbf{a} \in \mathcal{A}_i \setminus \{\mathbf{a}_i\}, i \in L(\mathbf{C})) \in \mathbb{R}^d, \quad \text{where } d = \sum_{i \in L(\mathbf{C})} (r_i - 1). \quad (3.3)$$

Any real vector  $\gamma \in \mathbb{R}^n$  can be chosen. If  $\gamma$  is fixed so that  $D(\mathbf{C})$  is feasible, the duality theorem holds for this primal-dual pair.  $P(\mathbf{C})$  is feasible if and only if  $D(\mathbf{C})$  is bounded below, and  $P(\mathbf{C})$  is infeasible if and only if  $D(\mathbf{C})$  is unbounded. Hence, the feasibility of  $P(\mathbf{C})$  can be revealed if the boundedness of  $D(\mathbf{C})$  is detected. The following describes how we set up  $\gamma$ . Recall that  $\mathbf{C}_0$  is a root node of  $U$ . For each node  $\mathbf{C} \in \Gamma$ , the feasible region of  $D(\mathbf{C}_0)$  is included in that of  $D(\mathbf{C})$ . Accordingly, if we set a right-hand vector  $\gamma$  of  $D(\mathbf{C})$  for each  $\mathbf{C} \in \Gamma$  as

$$\tilde{\gamma} = \Psi(\tilde{\mathbf{x}}; \mathbf{C}_0)$$

using an arbitrary nonnegative vector  $\tilde{\mathbf{x}} \in \mathbb{R}^d$ , then  $D(\mathbf{C})$  becomes feasible.

Furthermore, we can easily find an initial feasible solution for the dual problem  $D(\mathbf{C})$  for any  $\mathbf{C} \in \Gamma$  by using an optimal solution of  $D(\mathbf{C}_0)$ . Recall that the root node  $\mathbf{C}_0$  was revealed to be feasible. That is, we solved  $D(\mathbf{C}_0)$  and obtained an optimal solution  $\mathbf{x}^* \in \mathbb{R}^d$  of this problem, where  $d = \sum_{i \in L(\mathbf{C}_0)} (r_i - 1)$ . For any  $\mathbf{C} \in \Gamma$ , the vector

$$\begin{pmatrix} \mathbf{x}^* \\ \mathbf{0} \end{pmatrix} \in \mathbb{R}^{\bar{d}}, \quad \text{where } \bar{d} = \sum_{i \in L(\mathbf{C})} (r_i - 1). \quad (3.4)$$

becomes a feasible solution of  $D(\mathbf{C})$ . In addition, if  $D(\mathbf{C})$  is bounded below for  $\mathbf{C} \in \Gamma$ , an optimal solution of  $D(\mathbf{C})$  is a feasible solution of  $D(\bar{\mathbf{C}})$  for any child node  $\bar{\mathbf{C}} \in H(\mathbf{C}; \mathbf{C}_0, t_0)$  of  $\mathbf{C}$  since the feasible region of  $D(\mathbf{C})$  is included in that of  $D(\bar{\mathbf{C}})$ . The simplex method is suitable for solving these dual problems with the common structure. Assume that a node  $\mathbf{C} \in \Gamma$  is feasible and generates the child nodes  $\bar{\mathbf{C}} \in H(\mathbf{C}; \mathbf{C}_0, t_0)$  of  $\mathbf{C}$ . The simplex method usually does not require many iterations for solving  $D(\bar{\mathbf{C}})$  when we reuse an optimal solution of  $D(\mathbf{C})$  as an initial solution.

In this thesis, we deal with free variables in an LP problem directly without transforming into nonnegative variables. That is, we consider a formulation of an LP as

$$\begin{aligned} & \text{minimize} && \langle \mathbf{c}, \mathbf{x} \rangle \\ & \text{subject to} && \mathbf{G}\mathbf{x} = \mathbf{h} \\ & && x_i \geq 0, && (i \in I), \\ & && -\infty < x_j < +\infty, && (j \in J), \end{aligned} \tag{3.5}$$

where a coefficient matrix  $\mathbf{G} \in \mathbb{R}^{k \times d}$ , cost vector  $\mathbf{c} \in \mathbb{R}^d$  and constant vector  $\mathbf{h} \in \mathbb{R}^k$  are given, and  $\mathbf{x} \in \mathbb{R}^d$  is a variable vector. These index sets  $I$  and  $J$  of the variables satisfy  $I \cap J = \emptyset$  and  $I \cup J = \{1, 2, \dots, d\}$ . Here,  $x_i$  ( $i \in I$ ) and  $x_j$  ( $j \in J$ ) are called as a *nonnegative variable* and a *free variable*, respectively. The primal-dual pair  $P(\mathbf{C})$  and  $D(\mathbf{C})$  can be transformed into (3.5) by introducing slack variables to the inequalities of  $P(\mathbf{C})$  and replacing the cost vector  $\gamma$  of  $P(\mathbf{C})$  by  $-\gamma$ . In consequence of these transformations,  $P(\mathbf{C})$  has  $d(P)$  variables and  $k(P)$  equalities such that

$$d(P) = n + \sum_{i \in L(\mathbf{C})} (r_i - k_i - 1) \quad \text{and} \quad k(P) = \sum_{i \in L(\mathbf{C})} (r_i - 1).$$

On the other hand,  $D(\mathbf{C})$  has  $d(D)$  variables and  $k(D)$  equalities such that

$$d(D) = \sum_{i \in L(\mathbf{C})} (r_i - 1) \quad \text{and} \quad k(D) = n.$$

Here,  $d(D)$  is not greater than  $d(P)$  for any  $L(\mathbf{C}) \subseteq M$ . Also,  $k(D)$  is constant, whereas  $k(P)$  is monotonically increasing with respect to the cardinality of  $L(\mathbf{C})$ . From the definition of a mixed cell,  $\mathcal{A}_i$  ( $i \in M$ ) has at least two elements, i.e.,  $r_i \geq 2$ . Therefore there exists  $L' \subseteq M$  such that  $k(P) \geq k(D)$ . Consequently for any  $L$  such that  $L' \subseteq L \subseteq M$ , the number of constraints and that of variables in  $D(\mathbf{C})$  are not greater than those of  $P(\mathbf{C})$ . Therefore, it is reasonable to check whether  $D(\mathbf{C})$  is bounded for checking feasibility of  $\mathbf{C}$ .

# Chapter 4

## Efficient Implementation of Dynamic Enumeration

### 4.1 Introduction

In the previous chapter, although the framework for dynamic enumeration of all mixed cells was described as Algorithm 3.4.1, it was not given the detail description of how to choose an index  $t$  from  $M \setminus L(\mathbf{C})$  at (A) for a node  $\mathbf{C} \in \Omega(L)$  with  $L \subsetneq M$  of an enumeration tree  $T$ . It is the crucial point for the efficient implementation of the algorithm.

Mainly, the first half of this chapter is dedicated to the explanation of a choice strategy in the dynamic enumeration algorithm. As we have seen in Subsection 3.2.2, the Lift-Prune method proposed by Emiris and Canny [17] constructs a dynamic enumeration tree implicitly for implementing Huber and Sturmfels’s enumeration procedure efficiently for finding all mixed cells. The concept of the strategy is to choose the “best” index  $t \in M \setminus L(\mathbf{C})$  so that the total number of the child nodes of  $\mathbf{C}$  is small. The weak point is that the cost needed in the best choice is expensive. Evidently, an enumeration tree built by the method has much smaller feasible nodes over the static methods [61, 38, 57, 22]. However, the numerical results tell us that it is far behind the static enumeration method by Gao and Li [22] in terms of computational time.

In contrast to the best choice strategy by Emiris and Canny, our strategy chooses an “appropriate” index with much less effort, and aims at reducing the total cost for the implementation of dynamic enumeration. Although in general, our dynamic enumeration tree has more feasible nodes than the tree produced by the Lift-Prune method, we could expect that the node count is much smaller, compared with the static methods. To verify it, therefore, we demonstrate our dynamic enumeration and the static enumeration for several benchmark problems, and show the total number of feasible nodes generated by each method.

In the second half of this chapter, our dynamic enumeration is tested for various polynomial systems arising from theoretical to practical problems in order to reveal the perfor-

mance. As the papers [22, 23] mention, MixedVol, which performs the static enumeration by Gao and Li [22], is considered to be the most efficient software package among the existing ones [17, 60, 61, 62, 38, 57] for now. Therefore, we compare the computational time of our dynamic enumeration with MixedVol for polynomial systems of a fully mixed type and a structured type such as semi-mixed and unmixed systems. Also, the numerical results for large-scale polynomial systems obtained by our method is presented.

## 4.2 Implementation Issues

This section is composed of three subsections. In the first subsection, we give an outline of a choice strategy in the Lift-Prune method by Emiris and Canny. Our choice strategy is proposed in the second subsection. In the last third subsection, the demonstration of our dynamic enumeration and the static enumeration is carried out through well-known benchmark polynomial systems, and then we show the comparison of the total number of feasible nodes produced from each method.

### 4.2.1 Lift-Prune Method by Emiris and Canny

Emiris and Canny [17] proposed a geometric formulation, which is different from Li and Li's one [38] stated in Subsection 2.4.2, for finding all mixed cells based on Huber and Sturmfels's enumeration procedure. Their method specializes in a fully mixed support, ignoring the special structure of the support, and constructs not a fine mixed subdivision but a mixed subdivision, using a generic linear lifting function  $\omega_i$ . In this case, every mixed cell consists of each edge of Newton polytopes. This property is necessary in their method. Let us consider a fully mixed  $(1, 1, \dots, 1)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n)$  ( $\mathcal{A}_i \subseteq \mathbb{Z}_+^n$ ). Then, we want to construct a mixed subdivision of  $\mathcal{P} = \sum_{i \in N} \mathcal{P}_i$  for the Newton polytopes  $\mathcal{P}_i = \text{conv}(\mathcal{A}_i)$ . By applying a generic linear lifting  $\omega_i$  to  $\mathcal{A}_i$ , we obtain a lifted polytope  $\hat{\mathcal{P}}$ , which is given as the Minkowski sum of  $\hat{\mathcal{P}}_1, \hat{\mathcal{P}}_2, \dots, \hat{\mathcal{P}}_n$ . For an edge  $e_i = (\mathbf{a}_{i_1}, \mathbf{a}_{i_2}) \in \mathcal{A}_i \times \mathcal{A}_i$  of  $\mathcal{P}_i$ , we write the corresponding edge of  $\hat{\mathcal{P}}_i$  by  $\hat{e}_i = (\hat{\mathbf{a}}_{i_1}, \hat{\mathbf{a}}_{i_2}) \in \hat{\mathcal{A}}_i \times \hat{\mathcal{A}}_i$ . The idea for the construction of a mixed subdivision of  $\mathcal{P}$  is summarized as follows. First, we compute each edge set  $E_i$  for  $n$  Newton polytopes  $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$ . Next, for every combination of  $n$  edges  $(e_1, e_2, \dots, e_n) \in E_1 \times E_2 \times \dots \times E_n$ , we test whether the Minkowski sum  $\hat{e} = \sum_{i \in N} \hat{e}_i$  lies on the lower envelop of  $\hat{\mathcal{P}}$ . This test can be formulated as an LP problem. Let  $\hat{\mathbf{m}}_i \in \mathbb{R}^{n+1}$  be the center of an edge  $\hat{e}_i = (\hat{\mathbf{a}}_{i_1}, \hat{\mathbf{a}}_{i_2})$ , i.e.,  $\hat{\mathbf{m}}_i = \frac{1}{2}(\hat{\mathbf{a}}_{i_1} + \hat{\mathbf{a}}_{i_2})$ , and  $r_i$  the number of elements in  $\mathcal{A}_i$ . For every combination  $(e_1, e_2, \dots, e_n) \in E_1 \times E_2 \times \dots \times E_n$ , we consider the point  $\hat{\mathbf{m}} = \sum_{i \in N} \hat{\mathbf{m}}_i$  for the center  $\hat{\mathbf{m}}_i$  of each lifted edge  $\hat{e}_i$ , and compute the distance between

$\hat{\mathbf{m}}$  and the lower envelope of  $\hat{\mathcal{P}}$  such that

$$\begin{aligned}
 & \text{maximize} && s \\
 & \text{subject to} && \hat{\mathbf{m}} - s\mathbf{z} = \sum_{i \in N} \sum_{\hat{\mathbf{a}}_i \in \hat{\mathcal{A}}_i} \lambda_{\hat{\mathbf{a}}_i} \hat{\mathbf{a}}_i, \\
 & && \sum_{\hat{\mathbf{a}}_i \in \hat{\mathcal{A}}_i} \lambda_{\hat{\mathbf{a}}_i} = 1 \quad (i \in N) \quad \text{and} \quad \lambda_{\hat{\mathbf{a}}_i} \geq 0 \quad (\hat{\mathbf{a}}_i \in \hat{\mathcal{A}}_i, i \in N),
 \end{aligned} \tag{4.1}$$

where  $s \in \mathbb{R}$  and  $\mathbf{z} = (0, 0, \dots, 0, 1) \in \mathbb{R}^{n+1}$ . For some  $(e_1, e_2, \dots, e_n) \in \Pi_{i=1}^n E_i$ ,  $\hat{\mathbf{m}}$  lies on the lower envelope of  $\hat{\mathcal{P}}$ , which means that the corresponding edges  $e_1, e_2, \dots, e_n$  construct one of mixed cells, if and only if the optimal value is zero. Therefore, it follows from solving the above LP problems for all combination of  $n$  edges that we obtain every mixed cell. The efficient implementation of their method is realized through a tree representation.

Because the construction process of the tree for their method can be described using almost the same idea as Algorithm 3.4.1, we omit the precise definition of the tree structure, and instead illustrate the tree structure through a simple example. Remember that their method is designed on fully mixed supports, using a generic linear lifting  $\omega_i$  in order to construct a mixed subdivision. Consider a fully mixed  $(1, 1, 1)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$  ( $\mathcal{A}_i \subseteq \mathbb{Z}_+^3$ ) such that  $\#\mathcal{A}_1 = 3$ ,  $\#\mathcal{A}_2 = 2$  and  $\#\mathcal{A}_3 = 3$  so that the Newton polytopes  $\mathcal{P}_1, \mathcal{P}_3$  are two distinct triangles in  $\mathbb{R}^3$  and the other one  $\mathcal{P}_2$  is a line in  $\mathbb{R}^3$ , where  $\mathcal{P}_i = \text{conv}(\mathcal{A}_i)$ . Let  $E_i$  be an edge set of the Newton polytope  $\mathcal{P}_i$  for each  $i \in \{1, 2, 3\}$ . We know that the edge sets  $E_1, E_3$  have 3 elements, and the other one  $E_2$  has 1 element. In Emiris and Canny's method, for every edge combination  $(e_1, e_2, e_3) \in E_1 \times E_2 \times E_3$ , we solve an LP problem (4.1), and check whether the optimal value is zero or not. The process is represented by an enumeration tree of Figure 4.1. In the figure, a root node of the tree

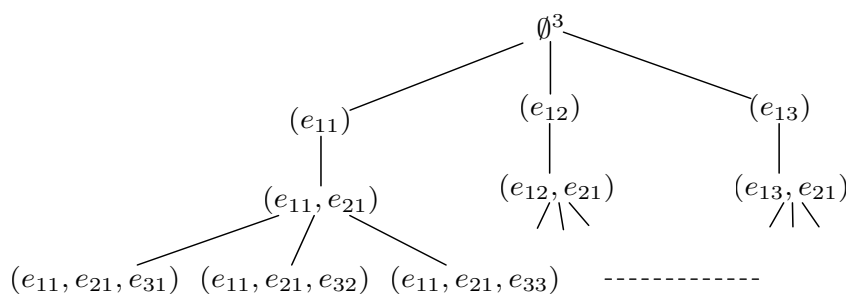


Figure 4.1: Tree structure for Emiris and Canny's method

is defined as  $\emptyset^3$  so that it becomes a rooted tree. Each node  $(e_{1i})$  ( $i \in \{1, 2, 3\}$ ) at the first level is corresponding to the element of  $E_1$ , and the node at the second level is the pair  $(e_{1i}, e_{21})$  consisting of  $e_{1i} \in E_1, e_{21} \in E_2$ , which is the child node of the node  $\{e_{1i}\}$ . The leaf nodes are the triplet  $(e_{1i}, e_{21}, e_{3j})$  ( $i, j \in \{1, 2, 3\}$ ) for  $e_{1i} \in E_1, e_{21} \in E_2$  and  $e_{3j} \in E_3$ . Obviously, we know that each leaf node of the tree is corresponding to the candidate for mixed cells, i.e., every edge combination  $(e_1, e_2, e_3) \in \Pi_{i=1}^3 E_i$ , and a leaf node is feasible if

and only if it is a mixed cell. Thus, if we solve an LP problem (4.1) for every leaf node of the tree, all mixed cells can be found. Although the tree of Figure 4.1 is constructed in the order  $E_1, E_2$  and  $E_3$  from top to bottom, the other trees can be built if we choose the different order of the construction.

To explain Emiris and Canny's dynamic enumeration strategy, we consider the general situation. Let  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n)$  ( $\mathcal{A}_i \subseteq \mathbb{Z}_+^n$ ) be a fully mixed  $(1, 1, \dots, 1)$ -support. After all edges of  $\mathcal{P}_i$  are enumerated into the set  $E_i$  for each Newton polytope  $\mathcal{P}_i = \text{conv}(\mathcal{A}_i)$ , we build an enumeration tree as shown in the above. By the phrase "a problem on  $v$ ", we mean an LP problem (4.1) for  $v = (e_1, e_2, \dots, e_\ell)$ , which corresponds to a node  $v$  at the  $\ell$ th level of the enumeration tree. Note that  $L = \{1, 2, \dots, \ell\}$ . Suppose that the optimal value of the problem on  $v = (e_1, e_2, \dots, e_\ell) \in \prod_{i=1}^\ell E_i$  is zero. Then, their dynamic enumeration method selects the "best" edge set  $E_t$  among  $E_s$  ( $s \in N \setminus L$ ) as the child node set of  $v$ . For every  $e \in E_s$  and every  $s \in N \setminus L$ , their method computes the set

$$W_E^*(v, s) = \{\bar{v} = (e_1, e_2, \dots, e_\ell, e) \in \prod_{i=1}^\ell E_i \times E_s : \begin{array}{l} \text{the optimal value} \\ \text{of the problem on } \bar{v} \text{ is zero} \end{array}\}$$

and select the index  $t$  such that the size of the set attains the minimum among  $s \in N \setminus L$ . Roughly speaking, we may consider this strategy as one of the varieties of Algorithm 3.4.1; at (B) of the algorithm, they choose an index  $t$  for a node  $\mathbf{C}$  so that the size of  $W^*(\mathbf{C}, s)$  is the smallest among  $s \in N \setminus L(\mathbf{C})$ . Here,  $W^*(\mathbf{C}, s)$  is the set of all feasible child nodes generated from a node  $\mathbf{C}$  associated with an index  $s \in N \setminus L(\mathbf{C})$ .

The software package MVLP [17] was developed by Emiris and Canny for implementing their dynamic enumeration method. We compare the performance of MVLP with MixedVol [23], which is the implementation of the static enumeration method with sophisticated bounding techniques proposed by Gao and Li, through numerical experiments. In the numerical experiments, we consider the following benchmark polynomial systems; the cyclic- $n$  [7] and noon- $n$  [47] problems in addition to the economic- $n$  problems of (3.1), which have been used for the evaluation of the ability of the polyhedral homotopy method in Subsection 3.2.1. The form of polynomial systems of the cyclic- $n$  and noon- $n$  problems is as follows.

#### Cyclic- $n$ problems

$$\begin{aligned} x_1 x_2 \cdots x_n - 1 &= 0, \\ x_1 x_2 \cdots x_{n-2} x_{n-1} + x_2 x_3 \cdots x_{n-1} x_n + \cdots + x_n x_1 \cdots x_{n-3} x_{n-2} &= 0, \\ x_1 x_2 \cdots x_{n-2} + x_2 x_3 \cdots x_{n-3} + x_{n-1} x_n x_1 + x_n x_1 \cdots x_{n-3} &= 0, \\ &\vdots \\ x_1 + x_2 + x_3 + \cdots + x_n &= 0, \end{aligned} \tag{4.2}$$

#### Noon- $n$ problems

$$\begin{aligned} x_1 x_2^2 + x_1 x_3^2 + \cdots + x_1 x_n^2 - 1.1x_1 + 1 &= 0, \\ x_2 x_1^2 + x_2 x_3^2 + \cdots + x_2 x_n^2 - 1.1x_2 + 1 &= 0, \\ &\vdots \\ x_n x_1^2 + x_n x_2^2 + \cdots + x_n x_{n-1}^2 - 1.1x_n + 1 &= 0. \end{aligned} \tag{4.3}$$

All numerical experiments were executed on a 2.4 GHz Opteron 850 with 8 GB memory. We summarize the numerical results in Table 4.1. The column “# mixed cells” presents the total number of mixed cells generated by MVLP for the corresponding polynomial system, and “mixed volume” the value of the mixed volume. The last column “ratio” indicates the ration between CPU time of MVLP and MixedVol. We recognize that MVLP is far behind MixedVol in computational time for these problems. The reason is considered as follows. As we have seen in the above, their dynamic enumeration method constructs every  $W_E^*(v, s)$  ( $s \in N \setminus L$ ) on a node  $v$  at the  $\ell$ th level, where  $L = \{1, 2, \dots, \ell\}$ , for the selection of the best index  $t$  among  $t \in N \setminus L$ . Although an enumeration tree built by such a selection has much fewer feasible nodes, the numerical results in Table 4.1 imply that the cost for finding the best index  $t$  in  $t \in N \setminus L$  is too expensive, and the total costs for their dynamic enumeration method increases substantially, compared with the static one by Gao and Li.

Table 4.1: Comparison of computation time between MVLP and MixedVol

system	size ( $n$ )	# mixed cells	mixed volume	MVLP	MixedVol	ratio
cyclic- $n$	10	2,919	35,940	5m50.3s	3.59s	97.86
	11	13,558	184,756	58m44.0s	35.32s	101.03
	12	30,483	500,352	6h28m42.0s	4m43.0s	81.84
economic- $n$	13	770	2,048	18m16.0s	5.75s	190.54
	14	1,354	4,096	1h21m12.3s	22.6s	215.58
	15	2,350	8,192	4h25m1.7s	1m25.8s	150.17
noon- $n$	13	4,750	1,594,297	29m39.0s	37.86s	47.00
	14	9,338	4,782,941	1h44m1.0s	2m37.79s	39.55
	15	18,731	14,348,877	5h44m58.0s	9m35.9s	33.80

## 4.2.2 Our Strategy

Let  $\mathbf{C}_0$  be an element of  $\mathcal{N}$  in the dynamic enumeration algorithm, i.e., Algorithm 3.4.1. Because we want to choose an index  $t \in M \setminus L(\mathbf{C}_0)$  at (A) of the algorithm so that a large portion of the child nodes are infeasible, the best choice is to find an index  $t$  such that the size of  $W^*(\mathbf{C}_0, s)$  attains the minimum among  $s \in M \setminus L(\mathbf{C}_0)$ . Certainly, such choice decreases the total number of the feasible nodes of the tree significantly, and hence, we might expect that it increases the computational efficiency. However, as we have seen in the previous subsection, the task for finding the best index  $t \in M \setminus L(\mathbf{C}_0)$  is too costly in general because all  $W^*(\mathbf{C}_0, s)$  ( $s \in M \setminus L(\mathbf{C}_0)$ ) need to be constructed. Therefore, to reduce the total costs of the dynamic enumeration algorithm, it is necessary to consider the trade-off between finding a good index among all the candidates and the cost required

for it. Our dynamic enumeration is designed so as to choose an appropriate index with a reasonable efforts and improve the computational efficiency eventually over the existing methods [17, 61, 38, 57, 22].

Suppose that we carry out Algorithm 3.4.1 for a semi-mixed  $(k_1, k_2, \dots, k_m)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$ , and take an element  $\mathbf{C}_0$  from  $\mathcal{N}$  in the process of the algorithm. Remember that each element in  $W(\mathbf{C}_0, t_0)$  for any fixed index  $t_0 \in M \setminus L(\mathbf{C}_0)$  is represented as a node  $\mathbf{C} \in \Gamma(k_t + 1; \mathbf{C}_0, t_0)$  at the  $(k_t + 1)$ th level of a tree  $U = (V, E)$  built by Algorithm 3.4.2. If some ancestor node of a node  $\mathbf{C} \in \Gamma(k_t + 1; \mathbf{C}_0, t_0)$  of  $U$  is infeasible, then,  $\mathbf{C}$  is infeasible. Therefore, instead of  $\Gamma(k_t + 1; \mathbf{C}_0, t_0)$ , we focus on the node set  $\Gamma(1; \mathbf{C}_0, t_0)$  at the first level of  $U$ , and find all feasible nodes in the set because it has much less elements than  $\Gamma(k_t + 1; \mathbf{C}_0, t_0)$ . For any  $s \in M \setminus L(\mathbf{C}_0)$ , let

$$W_1(\mathbf{C}_0, s) = \Gamma(1; \mathbf{C}_0, s) \quad \text{and} \quad W_1^*(\mathbf{C}_0, s) = \{\mathbf{C} \in \Gamma(1; \mathbf{C}_0, s) : \mathbf{C} \text{ is feasible}\}.$$

and obviously, we have

$$W^*(\mathbf{C}_0, s) \subseteq W_1^*(\mathbf{C}_0, s) \subseteq W(\mathbf{C}_0, s).$$

Note that  $W_1^*(\mathbf{C}_0, s)$  is exactly same as (3.2) because this set is constructed by the result of one point test, described in Subsection 3.4.1.

For each node  $\mathbf{C} \in W_1(\mathbf{C}_0, t_0)$  with any fixed index  $t_0 \in M \setminus L(\mathbf{C}_0)$ , the feasible solution of  $D(\mathbf{C})$  can be obtained easily from an optimal solution of  $D(\mathbf{C}_0)$ . In Algorithm 3.4.1, to check the feasibility of  $\mathbf{C}_0$ , we solved  $D(\mathbf{C}_0)$ , and then got an optimal solution  $\mathbf{x}_* \in \mathbb{R}^d$  of this problem. As stated in Section 3.5, a feasible solution  $\mathbf{x}_{init} \in \mathbb{R}^{\bar{d}}$  of  $D(\mathbf{C})$  for any  $\mathbf{C} \in W_1(\mathbf{C}_0, t_0)$  can be computed from (3.4) by use of an optimal solution  $\mathbf{x}_*$  of  $D(\mathbf{C}_0)$ . From the initial solution, we start the pivoting process of the simplex method to check the boundedness of  $D(\mathbf{C})$ . In this case, we do not need many iterations to solve  $D(\mathbf{C})$  for each  $\mathbf{C} \in W_1(\mathbf{C}_0, t_0)$  by the simplex method because the structures of  $D(\mathbf{C}_0)$  and  $D(\mathbf{C})$  is similar to each other. Indeed, in our numerical results, we observe that the simplex method requires only 2 or 3 iterations on average to solve  $D(\mathbf{C})$  starting from  $\mathbf{x}_{init}$ . Thus we expect that  $\mathbf{x}_{init}$  is incident to an unbounded direction if  $D(\mathbf{C})$  ( $\mathbf{C} \in W_1(\mathbf{C}_0, t_0)$ ) is unbounded. Accordingly, instead of applying the simplex method to check the feasibility of  $D(\mathbf{C})$  for each  $\mathbf{C} \in W_1(\mathbf{C}_0, t_0)$ , we propose to test whether the feasible solution  $\mathbf{x}_{init}$  of  $D(\mathbf{C})$  has unbounded directions or not. At (B) of Algorithm 3.4.1, we consider for any  $s \in M \setminus L(\mathbf{C}_0)$

$$\hat{W}_1^*(\mathbf{C}_0, s, \mathbf{x}_{init}) = \{\mathbf{C} \in \Gamma(1; \mathbf{C}_0, s) : \mathbf{x}_{init} \text{ of } D(\mathbf{C}) \text{ has no unbounded direction} \}$$

satisfying

$$W^*(\mathbf{C}_0, s) \subseteq W_1^*(\mathbf{C}_0, s) \subseteq \hat{W}_1^*(\mathbf{C}_0, s, \mathbf{x}_{init}) \subseteq W(\mathbf{C}_0, s)$$

and choose an index  $\hat{t} \in M \setminus L(\mathbf{C}_0)$  which attains

$$\min_{s \in M \setminus L(\mathbf{C}_0)} \#\hat{W}_1^*(\mathbf{C}_0, s, \mathbf{x}_{init}).$$

The explanation for the phrase “ $\mathbf{x}_{init}$  of  $D(\mathbf{C})$  has no unbounded direction” will be made in the next paragraph. In general, the index  $\hat{t}$  does not coincide with the index  $t$  which achieves the minimum size of the set  $W^*(\mathbf{C}_0, s)$  among  $s \in M \setminus L(\mathbf{C}_0)$ . However, we will be convinced from the numerical results shown in Section 4.3 that our dynamic enumeration method improves computational time significantly, compared with the existing methods. Also, in the end of this section, we will observe that the evaluation measure  $\nu_*$ , which was defined in Algorithm 3.4.1 to show the performance of the dynamic enumeration algorithm, is much smaller for our dynamic enumeration than for the existing static enumeration.

We explain how to compute elements in  $\hat{W}_1(\mathbf{C}_0, t_0, \mathbf{x}_{init})$  for any fixed  $t_0 \in M \setminus L(\mathbf{C}_0)$  more precisely. In the explanation, instead of  $D(\mathbf{C})$  with  $\mathbf{C} \in \Gamma(1; \mathbf{C}_0, t_0)$ , an LP form of (3.5) will be used because we prefer the simplicity of notation. For an LP problem (3.5), we assume that the number of variables  $d$  is not less than that of constraints  $k$ , and the matrix  $\mathbf{G} = (\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_d)$  has full row rank. If the problem (3.5) is feasible, there exists a vertex  $\mathbf{x} \in \mathbb{R}^d$  on the feasible region which consists of two components: the vector of basic variables  $\mathbf{x}_B = \mathbf{G}_B^{-1} \mathbf{h} \in \mathbb{R}^k$  and nonbasic variables  $\mathbf{x}_N = \mathbf{0} \in \mathbb{R}^{d-k}$  where  $\mathbf{G}_B \in \mathbb{R}^{k \times k}$  is a basic matrix. Note that  $\mathbf{x}_B = (x_{b_1}, x_{b_2}, \dots, x_{b_k})^T$  and  $\mathbf{x}_N = (x_{n_1}, x_{n_2}, \dots, x_{n_{d-k}})^T$ . In particular, the set of basic variables and nonbasic variables are called a *basis* and *nonbasis*. An adjacent vertex  $\tilde{\mathbf{x}}$  of  $\mathbf{x}$  is represented as

$$\tilde{\mathbf{x}} = \mathbf{x} + \theta \mathbf{d}$$

by using a nonnegative scalar  $\theta$  and a direction vector  $\mathbf{d}$ . Let  $D = \{1, 2, \dots, d\}$ , and we denote the set of basic indices  $B = \{b_1, b_2, \dots, b_k\} \subseteq D$ . Note that  $(d - k)$  extreme rays extend from a vertex  $\mathbf{x}$  and one direction  $\mathbf{d}$  is chosen by fixing an index  $j \in D \setminus B$ . The direction  $\mathbf{d}$  is composed of two components vectors  $\mathbf{d}_B$  and  $\mathbf{d}_N$  such that

$$\mathbf{d}_B = -\mathbf{G}_B^{-1} \mathbf{g}_j \in \mathbb{R}^k \text{ and } \mathbf{d}_N = \begin{cases} d_i = 1 & i = j \\ d_i = 0 & i \in D \setminus (B \cup \{j\}) \end{cases} \in \mathbb{R}^{d-k},$$

where  $d_i$  represents a component of  $\mathbf{d}$ . Figure 4.2 illustrates two direction vectors  $\mathbf{d}$  emanating from a vertex  $\mathbf{x}$ , and one of the directions leads an adjacent vertex  $\tilde{\mathbf{x}}$  of  $\mathbf{x}$ . When

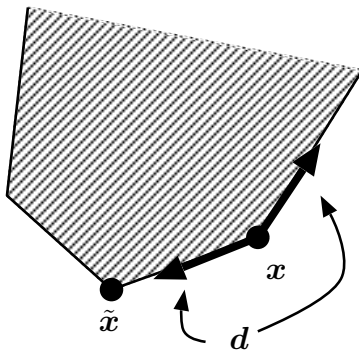


Figure 4.2: Direction vectors emanating from a vertex  $\mathbf{x}$

we move from  $\mathbf{x}$  to  $\tilde{\mathbf{x}}$ , the cost change per unit  $\theta$  is  $\langle \mathbf{c}, \mathbf{d} \rangle$ . Using a component  $\mathbf{c}_B$  of a cost vector  $\mathbf{c}$  which corresponds to a basis  $B$ , this amount can be written as

$$\langle \mathbf{c}, \mathbf{d} \rangle = c_j - \mathbf{c}_B^T \mathbf{G}_B^{-1} \mathbf{g}_j \quad (4.4)$$

and called a *reduced cost* for  $j \in D$ . To obtain an adjacent vertex  $\tilde{\mathbf{x}}$  of  $\mathbf{x}$  so that the value of a cost function decreases from  $\mathbf{x}$ , we search for a direction  $\mathbf{d}$  with  $j \in D$  such that its reduced cost is negative, and determine the step size  $\theta \geq 0$  such that a new vertex  $\tilde{\mathbf{x}} = \mathbf{x} + \theta \mathbf{d}$  satisfies its constraints; if components  $d_i$  of a direction vector  $\mathbf{d}$  are nonnegative for all  $i \in B \cap I$  where  $I$  is the index set of nonnegative variables in an LP problem (3.5), the problem is unbounded and we say that  $\mathbf{x}$  has an *unbounded direction*. Otherwise, we compute the largest  $\theta$  allowed by constraints for variables.

Now, we provide the criteria for detecting that the feasible solution  $\mathbf{x}_{init}$  of  $D(\mathbf{C})$  with  $\mathbf{C} \in \Gamma(1; \mathbf{C}_0, t_0)$  for any fixed index  $t_0 \in M \setminus L(\mathbf{C}_0)$  has an unbounded direction. Notice that the optimal basic matrix  $\mathbf{G}_B \in \mathbb{R}^{n \times n}$  of  $D(\mathbf{C}_0)$  becomes the basic matrix on  $\mathbf{x}_{init}$  of  $D(\mathbf{C})$  for any  $\mathbf{C} \in \Gamma(1; \mathbf{C}_0, t_0)$  because the number of constraints is equal to each other, and a feasible solution  $\mathbf{x}_{init}$  can be represented as (3.4) using an optimal solution  $\mathbf{x}_*$  of  $D(\mathbf{C})$ . Also, note that the vector  $(\mathbf{c}_B^T \mathbf{G}_B^{-1})^T$  in (4.4) is an optimal solution of its dual problem when the duality theorem holds for this primal-dual pair. Let  $\boldsymbol{\alpha}_* \in \mathbb{R}^n$  be an optimal solution of the primal problem  $P(\mathbf{C})$ . For a node  $\mathbf{C} = (C_i : i \in M) \in \Gamma(1; \mathbf{C}_0, t_0)$ , we write one of the elements of the nonempty set  $C_i$  ( $i \in L(\mathbf{C})$ ) by  $\mathbf{a}_i$ . The reduced costs on  $\mathbf{x}_{init}$  of  $D(\mathbf{C})$  for the node  $\mathbf{C} \in \Gamma(1; \mathbf{C}_0, t_0)$  are written as

$$\omega_i(\mathbf{a}) - \omega_i(\mathbf{a}_i) - \langle \mathbf{a}_i - \mathbf{a}, \boldsymbol{\alpha}_* \rangle \quad \text{for every } \mathbf{a} \in \mathcal{A}_i \setminus \{\mathbf{a}_i\} \text{ and every } i \in L(\mathbf{C}).$$

Since  $\boldsymbol{\alpha}_*$  is an optimal solution of  $P(\mathbf{C})$  which has  $\mathcal{I}(\mathbf{C})$  as a constraint, these reduced costs are nonnegative for every  $\mathbf{a} \in \mathcal{A}_i \setminus \{\mathbf{a}_i\}$  and every  $i \in L(\mathbf{C}) \setminus \{t_0\}$ . Consequently, if there are  $\mathbf{b} \in \mathcal{A}_{t_0} \setminus \{\mathbf{a}_{t_0}\}$  such that

- (a)  $\omega_{t_0}(\mathbf{b}) - \omega_{t_0}(\mathbf{a}_{t_0}) - \langle \mathbf{a}_{t_0} - \mathbf{b}, \boldsymbol{\alpha}_* \rangle < 0$
- (b) All components of a vector  $-\mathbf{G}_B^{-1}(\mathbf{a}_{t_0} - \mathbf{b})$ , which corresponds to nonnegative variables in the basis, are nonnegative,

then, the feasible solution  $\mathbf{x}_{init}$  of  $D(\mathbf{C})$  with  $\mathbf{C} \in \Gamma(1; \mathbf{C}_0, t_0)$  has an unbounded direction.

Remember we carried out Algorithm 3.4.1 for a semi-mixed  $(k_1, k_2, \dots, k_m)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$ , and picked up an element  $\mathbf{C}_0$  from  $\mathcal{N}$  in the execution. Let  $\mathbf{C}_0$  be a node satisfying  $L(\mathbf{C}_0) = L$ . Then, the problem  $D(\mathbf{C}_0)$  has  $n_f := \sum_{i \in L} k_i$  free variables, and all the free variables are stayed in the optimal basis if this problem is bounded. Therefore, since the basis on  $\mathbf{x}_{init}$  of  $D(\mathbf{C})$  for any  $\mathbf{C} \in \Gamma(1; \mathbf{C}_0, t_0)$  has  $n_f$  free variables, it is enough to check  $(n - n_f)$  components of a vector  $-\mathbf{G}_B^{-1}(\mathbf{a}_{t_0} - \mathbf{b})$  in (b).

### 4.2.3 Comparison of the Size of Static and Dynamic Trees

We demonstrate the performance of our dynamic enumeration and the existing static enumeration method, using the benchmark polynomial systems; the chandra- $n$  [9] and katsura- $n$  [8, 31] problems as well as the cyclic- $n$  and the noon- $n$  problems in Subsection 4.2.1, and the economic- $n$  problems in Subsection 3.2.1. The form of polynomial systems of the chandra- $n$  and katsura- $n$  problems is described as follows.

#### Chandra- $n$ problems

$$\begin{aligned}
 c_{n,0}x_n(c_{n,1}x_1 + c_{n,2}x_2 + \cdots + c_{n,n-1}x_{n-1} + c_{n,n}) + c_{n,n+1}x_n &= 0, \\
 c_{1,0}x_1(c_{1,1}x_1 + c_{1,2}x_2 + \cdots + c_{1,n-1}x_{n-1} + c_{1,n}) + c_{1,n+1}x_1 &= 0, \\
 c_{2,0}x_2(c_{2,1}x_1 + c_{2,2}x_2 + \cdots + c_{2,n-1}x_{n-1} + c_{2,n}) + c_{2,n+1}x_2 &= 0, \\
 &\vdots \\
 c_{n-1,0}x_{n-1}(c_{n-1,1}x_1 + c_{n-1,2}x_2 + \cdots + c_{n-1,n-1}x_{n-1} + c_{n-1,n}) + c_{n-1,n+1}x_{n-1} &= 0,
 \end{aligned}$$

#### Katsura- $n$ problems

$$\begin{aligned}
 \sum_{i=-n}^n x_i - 1 &= 0, \\
 x_0 + \sum_{i=-n}^n x_i x_{-i} &= 0, \\
 x_0 + \sum_{i=-n}^n x_i x_{1-i} &= 0, & x_{-i} = x_i, \\
 &\vdots & x_i = 0 \text{ if } i > n, \\
 x_0 + \sum_{i=-n}^n x_i x_{n-1-i} &= 0.
 \end{aligned}$$

In the chandra- $n$  problems, each term in the polynomials has a real number coefficient, and we write it by  $c_{i,j}$  because the number is not simple. The precise form can be found in the website of FRISCO [19] and Verschelde [58]. Notice that the katsura- $n$  problem consists of  $n + 1$  polynomials with  $n + 1$  unknowns, and the others  $n$  polynomials with  $n$  unknowns. For these polynomial systems, we denote the support of the  $i$ th polynomial from top as  $\mathcal{A}_i$ . We set the support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n)$  and its multiplicity  $k = (1, 1, \dots, 1)$  as the input data of Algorithm 3.4.1 because all of the polynomial systems are a fully mixed type. When the static enumeration method is employed, we choose a one-to-one mapping  $\pi : N \rightarrow N$  such that  $\pi(i) = i$  for each  $i \in N$ .

Table 4.2 summarizes the total number of nodes  $\nu^*$  generated by the static and dynamic enumeration (abbreviated by “static enum.” and “dynamic enum.”, respectively) for the chandra- $n$ , cyclic- $n$ , economic- $n$ , katsura- $n$  and noon- $n$  problems. The last column “ratio” shows the ratio between  $\nu^*$  given by these two methods. Figure 4.3-4.7 depict each line graph for these five problems; the horizontal line indicates the problem size  $n$ , vertical line

the total number of nodes  $\nu^*$ . The numerical results for these polynomial systems show that the total number of feasible nodes generated by our dynamic enumeration method is much smaller than by the static one. We can see that the ratio increases as the size of each system becomes larger.

Table 4.2: Total number of nodes  $\nu^*$  generated by the static and dynamic enumeration method

system	size ( $n$ )	static enum.	dynamic enum.	ratio
chandra- $n$	$n = 15$	$4.54 \times 10^9$	$8.31 \times 10^7$	54.71
	$n = 16$	$1.73 \times 10^{10}$	$2.14 \times 10^8$	80.63
	$n = 17$	$6.52 \times 10^{10}$	$5.88 \times 10^8$	110.98
	$n = 18$	$2.53 \times 10^{11}$	$1.62 \times 10^9$	156.75
	$n = 19$	$9.06 \times 10^{11}$	$4.43 \times 10^9$	204.70
cyclic- $n$	$n = 10$	$2.46 \times 10^7$	$6.64 \times 10^6$	3.70
	$n = 11$	$2.35 \times 10^8$	$5.31 \times 10^7$	4.42
	$n = 12$	$1.73 \times 10^9$	$3.41 \times 10^8$	5.07
	$n = 13$	$1.76 \times 10^{10}$	$2.84 \times 10^9$	6.21
	$n = 14$	$1.40 \times 10^{11}$	$2.02 \times 10^{10}$	6.95
economic- $n$	$n = 15$	$2.09 \times 10^8$	$6.37 \times 10^7$	3.28
	$n = 16$	$6.89 \times 10^8$	$2.18 \times 10^8$	3.16
	$n = 17$	$2.50 \times 10^9$	$7.64 \times 10^8$	3.28
	$n = 18$	$9.83 \times 10^9$	$2.90 \times 10^9$	3.40
	$n = 19$	$3.43 \times 10^{10}$	$9.72 \times 10^9$	3.53
katsura- $n$	$n = 10$	$3.80 \times 10^7$	$5.08 \times 10^6$	7.50
	$n = 11$	$1.78 \times 10^8$	$2.12 \times 10^7$	8.42
	$n = 12$	$8.50 \times 10^8$	$9.48 \times 10^7$	8.96
	$n = 13$	$4.33 \times 10^9$	$5.13 \times 10^8$	8.44
	$n = 14$	$1.97 \times 10^{10}$	$2.19 \times 10^9$	9.02
noon- $n$	$n = 14$	$2.08 \times 10^9$	$6.11 \times 10^9$	34.02
	$n = 15$	$7.95 \times 10^9$	$1.79 \times 10^8$	44.46
	$n = 16$	$3.17 \times 10^{10}$	$4.80 \times 10^8$	65.99
	$n = 17$	$1.22 \times 10^{11}$	$1.35 \times 10^9$	90.47
	$n = 18$	$4.41 \times 10^{11}$	$3.39 \times 10^9$	130.01

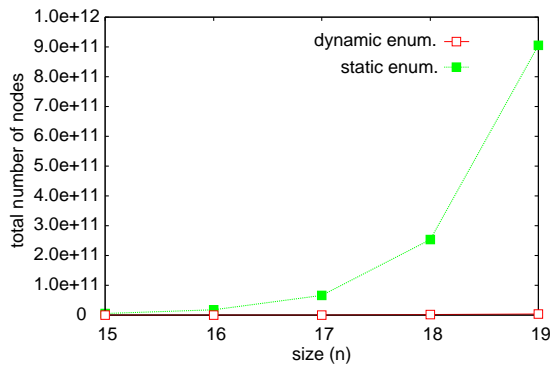


Figure 4.3: Chandra- $n$  problem

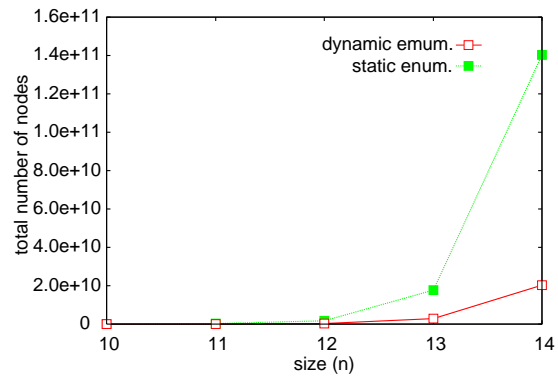


Figure 4.4: Cyclic- $n$  problem

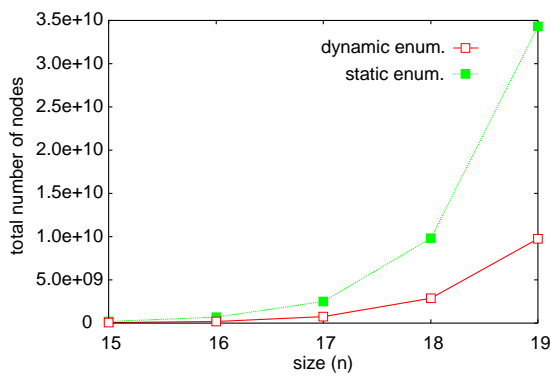


Figure 4.5: Economic- $n$  problem

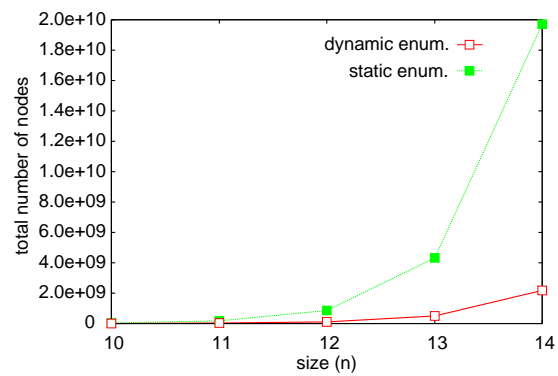


Figure 4.6: Katsura- $n$  problem

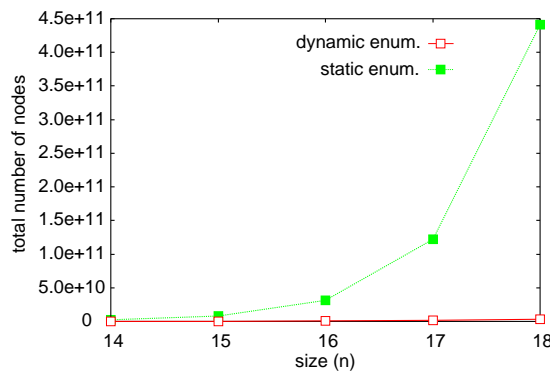


Figure 4.7: Noon- $n$  problem

### 4.3 Numerical Results on Single Processor

The C++ software package DEMiCs (*Dynamic Enumeration of All Mixed Cells*) was developed for implementing our dynamic enumeration method. It is publicly available at the website [42]. DEMiCs has been tested on a large variety of polynomial systems including fully mixed, semi-mixed and unmixed polynomial systems. Because the papers [22, 23]

report the superiority of MixedVol in computational time for these three types of the systems over the existing software packages: MVLP [17], PHCpack [63], mvol [38] HOM4PS [21] and PHCpack [63]. 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 2.4 GHz Opteron 850 with 8 GB memory, running Linux.

### 4.3.1 Fully Mixed Type

As test problems for the comparison of DEMiCs with MixedVol, we choose a fully mixed type of polynomial systems from the benchmark problems as follows; the chandra- $n$  [9], cyclic- $n$  [7], economic- $n$  [44], katsura- $n$  [8, 31] and noon- $n$  [47] problems. The form of these size-expandable systems by the number  $n$  has been given in Subsection 3.2.1, Subsection 4.2.1 and 4.2.3. Note that the katsura- $n$  systems is composed of  $n + 1$  polynomials with  $n + 1$  variables, and the others  $n$  polynomials with  $n$  variables. In particular, the cyclic- $n$  problems [7], arise from Fourier analysis, are known as the most notorious one in polynomial system solving. In the next subsection, we deal with a semi-mixed and unmixed type of polynomial systems for the performance comparison. In the numerical experiments, we

Table 4.3: Performance of DEMiCs and MixedVol on fully mixed polynomial systems

system	size ( $n$ )	# mixed cells	mixed volume	DEMiCs	MixedVol	speed-up ratio
chandra- $n$	$n = 17$	41,415	65,536	1m9.4s	33m13.4s	28.70
	$n = 18$	82,830	131,072	3m10.5s	2h14m15.3s	42.29
	$n = 19$	154,343	262,144	9m21.1s	8h19m6.3s	53.38
cyclic- $n$	$n = 12$	30,147	500,352	1m11.6s	4m43.0s	3.95
	$n = 13$	146,982	2,704,156	11m0.3s	49m57.4s	4.54
	$n = 14$	418,199	8,795,976	1h27m27.3s	7h14m24.1s	4.97
economic- $n$	$n = 18$	12,660	65,536	16m50.2s	1h17m56.0s	4.63
	$n = 19$	23,975	131,072	1h5m53.3s	4h56m4.6s	4.49
	$n = 20$	45,562	262,144	4h13m47.1s	18h41m23.9s	4.42
katsura- $n$	$n = 12$	387	4,096	46.9s	14m3.5s	17.98
	$n = 13$	678	8,192	5m8.1s	1h21m19.4s	15.84
	$n = 14$	1,179	16,384	24m17.2s	7h54m29.4s	19.54
noon- $n$	$n = 16$	47,773	43,046,689	58.4s	33m54.8s	34.85
	$n = 17$	94,945	129,140,129	2m45.1s	2h25m20.8s	52.83
	$n = 18$	179,418	387,420,453	6m46.7s	8h23m19.6s	74.25

changed the lifting values 10 times for each system. The comparison of the average CPU time of DEMiCs and MixedVol is summarized in Table 4.3. The column “# mixed cells” presents the average number of mixed cells generated by DEMiCs for the corresponding systems, and “mixed volume” is the value of mixed volume. The column “speed-up ratio” indicates the ratio between the CPU time of these software packages. The numerical results in the table show that our dynamic enumeration method improves the computational time for finding all mixed cells dramatically, compared with MixedVol, which is the most efficient software package among the existing ones.

Next, let us challenge the large-scale polynomial systems by DEMiCs. All numerical experiments were carried out 5 times for each system associated with the different lifting values. Table 4.4 exhibits the average CPU time of DEMiCs for finding all mixed cells for these polynomial systems. As compared with numerical results in Table 2 of [41], the computational time of DEMiCs is less than that of the program developed in [41] as the size of the systems becomes larger. It could be due to improvement on how to use memory space in DEMiCs.

Table 4.4: Large-scale polynomial systems

system	size ( $n$ )	# mixed cells	mixed volume	CPU time
chandra- $n$	$n = 20$	343,624	524,288	29m50.8s
	$n = 21$	636,145	1,048,576	1h15m19.7s
	$n = 22$	1,257,197	2,097,152	3h5m48.2s
	$n = 23$	2,882,632	4,194,304	11h24m4.6s
	$n = 24$	5,040,423	8,388,608	30h1m33.6s
cyclic- $n$	$n = 15$	1,458,142	35,243,520	13h33m53.8s
	$n = 16$	5,110,494	135,555,072	110h21m40.5s
economic- $n$	$n = 21$	78,327	524,288	14h39m54.1s
	$n = 22$	145,245	1,048,576	57h33m51.1s
katsura- $n$	$n = 15$	2,054	32,768	1h30m54.7s
	$n = 16$	3,594	65,536	9h49m20.8s
	$n = 17$	4,659	131,072	46h37m35.8s
noon- $n$	$n = 19$	389,518	1,162,261,429	24m26.0s
	$n = 20$	660,185	3,486,784,361	54m1.5s
	$n = 21$	1,633,658	10,460,353,161	3h33m16.6s
	$n = 22$	2,535,780	31,381,059,565	8h12m11.2s
	$n = 23$	6,134,331	94,143,178,781	26h19m34.1s

Finally, we apply DEMiCs to a wide variety of problems by picking up the polynomial systems from the test suite of MixedVol. These test problems are contained in the software

Table 4.5: Test problems from MixedVol

problem name	# polynomials (or # variables)	cardinality of supports	# mixed cells	mixed volume
butcher8	8	(5, 6, 7, 7, 9, 9, 9, 8)	10	26
camera1s	6	(4, 16, 14, 16, 16, 16)	11	20
comb3000	10	(5, 3, 7, 3, 2, 2, 2, 2, 2, 2)	10	16
cpdm5	5	(23, 23, 23, 23, 23)	23	242
d1	12	(3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 5, 7)	3	192
des18.3	8	(9, 9, 9, 7, 5, 6, 4, 2)	34	46
des22.24	10	(6, 6, 5, 5, 4, 4, 3, 3, 2, 2)	41	42
geneig	6	(15, 15, 14, 15, 15, 6)	7	10
heart	8	(3, 3, 5, 5, 7, 7, 9, 9)	17	121
il	10	(3, 3, 3, 3, 3, 3, 3, 3, 3, 3)	15	66
ipp	8	(3, 3, 3, 3, 14, 12, 16, 17)	4	64
kin1	12	(3, 3, 3, 3, 3, 3, 7, 5, 4, 4, 4, 4)	3	192
kinema	9	(5, 5, 6, 6, 7, 7, 8, 9, 8)	12	64
ku10	10	(4, 4, 4, 4, 4, 4, 4, 4, 4, 4)	2	2
md6	5	(57, 57, 57, 57, 57)	94	320
planar_4bar	4	(29, 29, 29, 29)	19	80
puma	8	(3, 3, 3, 3, 3, 3, 7, 6)	1	16
quadgrid	5	(5, 8, 11, 14, 17)	4	10
rabmo	9	(5, 6, 6, 6, 6, 3, 3, 4, 4)	17	136
ran_stew	9	(4, 5, 6, 5, 6, 7, 10, 10, 10)	9	64
rbpl	6	(4, 28, 28, 28, 24, 30)	27	160
rbpl24	9	(2, 4, 5, 6, 5, 6, 7, 34, 34)	7	80
rbpl24s	9	(4, 5, 6, 5, 6, 7, 34, 34, 2)	14	80
rose	3	(2, 6, 21)	7	136
s9_1	8	(2, 2, 3, 3, 4, 4, 3, 3)	8	10
speer	4	(23, 23, 23, 23)	44	96
virasoro	8	(13, 13, 13, 13, 11, 11, 11, 8)	101	200

package MixedVol and also DEMiCs. In Table 4.5, we summarize a problem name, size, the cardinality of each support, the total number of mixed cells generated by DEMiCs and the mixed volume. Here, a problem size means the number of polynomials in the problem, i.e., the number of variables, and it is listed in the column “# polynomials (or # variables)”. In the table, we omit the description of the computational time because DEMiCs could solve all problems in less than a few seconds.

### 4.3.2 Semi-Mixed Type

First, we make it clear how the utilization of the special structure inherited in semi-mixed support has an effect on computational time through semi-mixed and unmixed polynomial systems arising from mechanism design. As test problems, the following are considered. The PRS-10 and RRS-12 systems in [55] have 12 polynomials with 12 variables and 11 polynomials with 11 variables, respectively. The support of the PRS-10 system is a semi-mixed  $(1, 1, 1, 9)$ -support, and the first three supports have 4 elements, the last 100 elements. On the other hand, that of the RRS-12 system is an unmixed type, and has 224 elements. The nine-point system in [65] consists of 8 polynomials with 8 variables. Its support is an unmixed type, and has 150 elements. Note that the support of the polynomial systems described in above is obtained after we remove interior points in the Newton polytopes because these points have no effect on the mixed volume computation as the paper [38] mentions. These supports are different from the original ones. The experiments were carried out 10 times with the different lifting values for each trial. Table 4.6 lists the numerical results. Here, the third and fourth row show the average CPU time of DEMiCs in two cases; the support structure of the corresponding semi-mixed system is taken account and conversely, it is regarded as a fully-mixed type. The speed-up ratio is indicated in the last row. The first and second row present the mixed volume and the average number of mixed cells obtained by considering the semi-mixed support structure. The table gives a strong impression that the utilization of semi-mixed support structure leads to improve computational time dramatically.

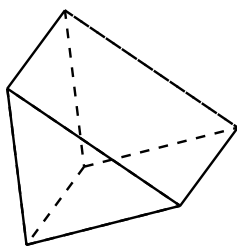
Table 4.6: Utilization of the support structure of semi-mixed polynomial systems from mechanism design

	PRS-10	RRS-10	nine-point
mixed volume	142,815	226,512	79,135
# mixed cells	11,754	25,209	17,338
take account of a semi-mixed support structure	14.3s	29.9s	5.5s
regard the system as a fully-mixed type	8m1.7s	1h6.4s	15m11.9s
speed-up ratio	33.8	120.7	164.4

Next, let us observe how the computational time of DEMiCs and MixedVol varies depending on  $m$ , which is the number of distinct supports  $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m$  of the semi-mixed polynomial systems. In numerical experiments, we deal with artificial semi-mixed systems, which are created to investigate the feature of DEMiCs. Each support  $\mathcal{A}_i$  of the semi-mixed systems is given as follows. We choose the subset  $\mathcal{T}$  of  $\mathbb{Z}_+^n$  with  $\#\mathcal{T} = 2n$  such that

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

Note that the convex hull of  $\mathcal{T}$  is the  $n$ -dimensional prism with a simplex basis, and the  $n$ -dimensional volume is  $\frac{1}{(n-1)!}$ . Fig 4.8 illustrates the convex hull of  $\mathcal{T}$  when  $n = 3$ . Let  $\mathbf{e}_i$

Figure 4.8: The convex hull of  $\mathcal{T}$  ( $n = 3$ )

denote the  $n$ -dimensional  $i$ th unit vector. For  $\mathcal{T} \subseteq \mathbb{Z}_+^n$ , we consider the transition of  $\mathcal{T}$  by the direction vector  $\mathbf{e}_i$  as  $\mathcal{A}_i$ . Namely,

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

Assume that each support set  $\mathcal{A}_i$  has the multiplicity  $n/m \in \mathbb{Z}$  for the dimension  $n$  and the number of distinct support sets  $m$ . That is, we consider a semi-mixed  $(n/m, n/m, \dots, n/m)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  in the numerical experiments. Here, the mixed volume for the support  $\mathcal{A}$  is calculated as  $n! \times \frac{1}{(n-1)!} = n$  from Theorem 2.4.1 because the  $n$ -dimensional volume of  $\text{conv}(\mathcal{A}_i)$  and  $\text{conv}(\mathcal{T})$  is equal to each other by Proposition 2.3.3 (b).

To demonstrate the performance of DEMiCs and MixedVol, we choose two different values  $n = 18, 24$  for the dimension of elements in  $\mathcal{A}_i \subseteq \mathbb{Z}_+^n$ , and change the number of distinct support sets  $m$  in response to each dimension  $n$ . Table 4.7 and 4.8, 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 4.7 and 4.8 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  $M \setminus L(\mathbf{C})$  at Algorithm 3.4.1. Therefore, DEMiCs takes more computational time than MixedVol for semi-mixed systems with a few distinct supports.

Table 4.7:  $n = 18$  (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 4.8:  $n = 24$  (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	-	

# Chapter 5

## Parallel Implementation of Dynamic Enumeration

### 5.1 Introduction

It seems to be natural to get into the idea to borrow enormous power from high-performance parallel computing system when we perform homotopy continuation for computing all isolated zeros of a system of polynomial equations. This is because it is possible to apply a path following algorithm to each homotopy path independently. In fact, the existing works [64, 56, 27] show that the parallelization in path following algorithm can gain significant speed-up and provide new results for polynomial system solving. This feature is important for solving large-scale polynomial systems. As well as path following method, the construction of a family of homotopy functions in the polyhedral method can be parallelized. Although the existing works [18, 57, 27] parallelize the static enumeration, we propose the parallel algorithm for dynamic enumeration in the first half of this chapter after the introduction of background of parallel computing in homotopy methods.

In the second half of this chapter, we present the numerical results obtained from our parallel method by taking the challenge for the generation of all mixed cells and the computation of the mixed volume for larger size polynomial systems, which have not been solved by the serial implementation. Also, we conduct the investigation for how faster our parallel method can compute all mixed cells and the mixed volume for polynomial systems on multiple processors, compared with a single processor.

### 5.2 Parallel Computation for Finding All Mixed Cells

This section is composed of two subsections. In the first subsection, we refer to the related work in parallel computing of homotopy methods, and also have a quick review of history in high-performance computing technology. Our parallelization of dynamic enumeration is proposed in the second subsection.

### 5.2.1 Background of Parallel Computation

Parallel computing is frequently utilized in the implementation of homotopy methods. There seem to be two reasons for it. One reason is the rapid development of high-performance computing technology in the last decade. The TOP 500 Supercomputing sites <sup>†</sup> lists 500 most powerful computer systems in the world. Every time the list of the sites is updated, the ranking changes, so that we glance at the rapid progress in the field of high-performance computing. Indeed, the parallel computer Deep Blue beat the world chess champion in 1997. It made a big impact on people. Also, in the academic field, the notable development encourages us to challenge tough problems which have been considered to be impossible to solve because they require a huge computational cost.

Another reason is that homotopy methods can be parallelized easily and well-suitable for parallel computing as we will see in the successive two subsections. This is a remarkable advantage when we challenge to solve a large-scale polynomial system. In the path following stage, we notice immediately that tracking one homotopy path is independent from tracking others, and hence, a path following technique, based on the predictor-corrector procedures, can be carried out for each homotopy path in parallel. Actually, in 2006, Verschelde and Zhuang [64], and Su et al. [56] parallelized the path following algorithm in PHCpack [63] and HOMPACT [67], respectively. By the parallelization, these two research groups succeeded to compute all isolated zeros of polynomial systems, including the PRS-10 problem from mechanism design [55] which we have used as one of the benchmark problems for the evaluation of our dynamic enumeration method in Subsection 4.3.2. In [64], the authors report that the parallel algorithm achieves high scalability because of a dynamic workload distribution. The meaning of the terminology “scalability”, which is sometimes used in the context of parallel computing, will be explained in the next subsection. The software package developed by Su et al. is published under the name POLSYS\_GLP.

In the polyhedral homotopy method, we need to find all feasible leaf nodes of an enumeration tree  $T$  generated by Algorithm 3.3.1 to construct the homotopy functions. This is a heavy task because it is difficult to avoid the fact that the computational cost increases dramatically as the size of polynomial system grows. However, fortunately, this task also can be parallelized. In 1997, the parallel implementation of the Lift-Prune method [17], shown in Subsection 4.2.1, was carried out by Emiris and Giordano [18]. Their parallel method takes 12 hours in the enumeration of all mixed cells for the cyclic-11 system whose general form has been given in Subsection 4.2.1, using 4 workstations of varying power: two Dec and two Sun-SPARC workstations. In 2002, Takeda et al. [57] implemented the static enumeration method, which is a dual approach to Li and Li’s formulation, in parallel. The idea of parallelization is based on the property that an enumeration tree  $T$  constructed by Algorithm 3.3.1 can be partitioned into the subtrees  $T_s$ , and these subtrees  $T_s$  do not share the common nodes with each other. Thus, a search method for finding feasible leaf nodes can be applied to each subtree  $T_s$  independently. The parallel method made it possible to

---

<sup>†</sup>TOP 500 Supercomputer sites (<http://www.top500.org/>)

compute all mixed cells for the cyclic-14 system in about 5 hours using 128 processors on a PC cluster, where the cluster consists of 64 server machines and each server machine has 2 processors of 824 MHz Pentium III with 640 MB memory. This shows the strength of the parallel implementation because the serial method implemented on one processor in the PC cluster can solve only up to  $n = 12$  for the cyclic- $n$  systems within 4 hours. In 2005, Gunji et al. [27] parallelized PHoM and developed the software package PHoMpara. Remember that the polyhedral homotopy method is composed of two stages: the construction of a family of polyhedral homotopies and the path following. The feature of PHoMpara is that both these stages are parallelized, whereas the parallelization in PHCpack, which employs the polyhedral homotopies as one of the modules, by Verschelde et al. [64] is limited to the path following stage at the moment. PHoMpara utilizes the static enumeration presented by Takeda et al. [57] in the construction stage of polyhedral homotopies. The authors [27] proposed the redistribution technique to improve the scalability for the parallel implementation of static enumeration. Their numerical results imply that PHoMpara can achieve high scalability due to the technique.

## 5.2.2 Parallelization of Dynamic Enumeration

We implement the dynamic enumeration method under a client-server parallel computing platform. In the platform, one client processor communicates with  $r$  server processors  $p_1, p_2, \dots, p_r$  and a communication among the servers is not allowed. Let  $P$  denote the set of all the server processors  $p_1, p_2, \dots, p_r$ . Figure 5.1 illustrates the structure of the platform.

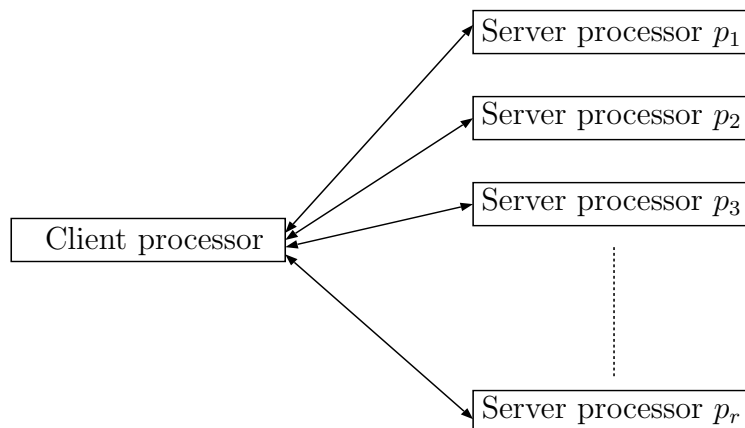


Figure 5.1: A client-server parallel computing platform

Our parallel method utilizes the property that an enumeration tree  $T$  generated by Algorithm 3.3.1 can be partitioned into the subtrees  $T_s$ , and these subtrees  $T_s$  do not share the common nodes with each other. More precisely, let  $\mathbf{C}, \mathbf{C}' \in \Omega(L_s)$  with  $L_s \subsetneq M$  be the two distinct nodes of  $T$ . For these distinct nodes  $\mathbf{C}, \mathbf{C}'$ , as shown in Figure 5.2, there is no

intersection in nodes between the two subtrees  $T_s, T'_s$  whose roots are  $\mathbf{C}$  and  $\mathbf{C}'$ , respectively, so that the dynamic enumeration algorithm (i.e., Algorithm 3.4.1) can be carried out for the subtrees  $T_s, T'_s$  independently. That is, we assign each  $\mathbf{C} \in \Omega(L_s)$  to a server processor  $p_i \in P$ . Then, in the server  $p_i$ , Algorithm 3.4.1 is executed in order to find all feasible leaf nodes of the subtree  $T_s$  whose root is  $\mathbf{C}$ . Here, we call  $L_s$  a *starting index set*, which

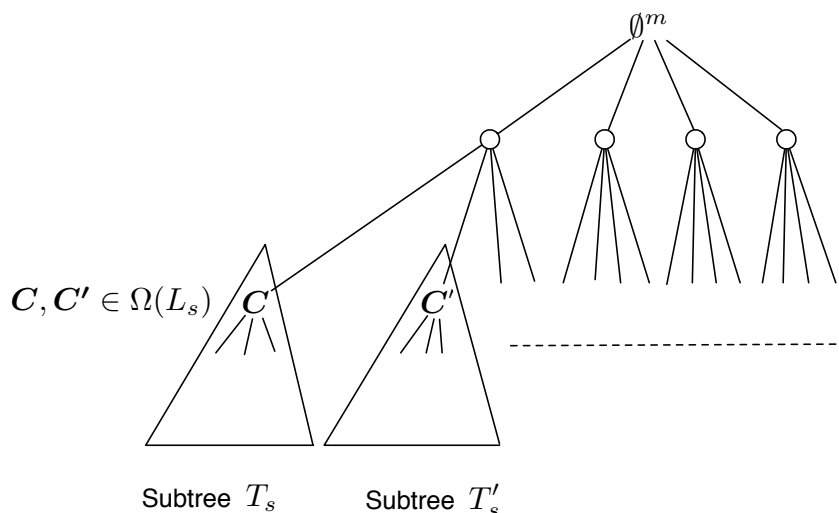


Figure 5.2: The construction of a dynamic enumeration tree in parallel

determines the root set of the subtrees as  $\Omega(L_s)$ . To increase effect of parallelization, we need to avoid the situation that the tasks broadcasted from the client are concentrated on only a few server processors. Therefore, we suppose that a starting index set  $L_s$  is chosen in the parallel method so as to satisfy  $\#\Omega(L_s) \geq r$ . For a semi-mixed  $(k_1, k_2, \dots, k_m)$ -support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$ , the parallel implementation of the dynamic enumeration method is described as Algorithm 5.2.1. Here, we assume that the information about the support  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  and its multiplicity  $k = (k_1, k_2, \dots, k_m)$  is sent to every processor consisting of the client-server parallel computing system before starting the algorithm, and then it can be available during the whole process of the execution of the algorithm.

In the client processor, the algorithm is composed of three parts. The first part is to construct  $\Omega(L_s)$  according to Algorithm 3.3.1. After the construction, all the nodes in  $\Omega(L_s)$  are stored in the set  $\mathcal{N}$ . The second part is to send  $\mathbf{C} \in \mathcal{N}$  to all servers  $p_1, p_2, \dots, p_r$ . In the server processor  $p_i$ , a flag  $F_i$  has two values *on* and *off* to let the client know whether the server  $p_i$  executes Algorithm 3.4.1 or not;  $F_i$  is *on* if  $p_i$  is working for the execution of the algorithm, and otherwise,  $F_i$  is *off*. If the idle server  $p_i$  receives  $\mathbf{C} \in \mathcal{N}$  from the client, it starts Algorithm 3.4.1 for the enumeration of the feasible leaf nodes of a subtree  $T_s$ , whose root node is  $\mathbf{C}$ .  $\mathcal{M}_i$  is constructed as the set of all the feasible leaf nodes of  $T_s$  generated by the algorithm. The server  $p_i$  is in idle state when the algorithm terminates. Meanwhile, the client processor always try to find such idle servers. It is the third task in the client. When the client finds the idle server  $w_i$ , it requires  $w_i$  to send  $\mathcal{M}_i$ , and stores all the elements in

the set  $\mathcal{M}$ . If  $\mathcal{N}$  is not empty, the client sends another node  $\mathbf{C}' \in \mathcal{N}$  to  $w_i$ . This part is repeated until the client receives all answers  $\mathcal{M}_i$ , which were requested by the client, from the servers  $p_i \in P$ .

In the rest of this subsection, we mention how to measure the performance of parallel algorithm in general case. Although there are some measures for the evaluation, one of them is *scalability*. When  $r$  processors are used for the implementation of parallel algorithm, it is reasonable to expect that a problem can be solved  $r$  times faster than a single implementation. Scalability measures how faster a parallel method on  $r$  processors solves a problem compared to a serial method. If the speed-up ratio for computational time between a parallel algorithm on  $r$  processors and a serial algorithm reaches to  $r$ , then the obtained scalability is considered to be ideal. However, as is often the case, it is difficult to achieve such ideal scalability. One reason is the existence of communication time among multiple processors. When a large amount of data is frequently sent and received among processors through network, the communication time has considerable influence on the execution time of a parallel method. In general, the decrease of communication time leads to the improvement of scalability of a parallel method. Another reason is *granularity*. It means the sizes of subproblems into which an original problem is divided in order to assign on multiple processors. To pursue high scalability, it is desirable that each subproblem size is relatively small and there is no outstanding large size one. Such a parallel algorithm is called *fine-grained*. Conversely, if the sizes of subproblems generated from an original problem are relatively large, it is called *coarse-grained*. Note that a fine-grained algorithm communicates small size data frequently among multiple processors, whereas in a coarse-grained one, large size data are transferred infrequently.

**Algorithm 5.2.1.** (Parallel implementation of dynamic enumeration algorithm.)

**The client processor**

*Input:* A starting index set  $L_s$ .

*Output:* All mixed cells  $\mathbf{C} \in \Omega^*$ .

Create a node set  $\Omega(L_s)$  using Algorithm 3.3.1.

Initialize  $\mathcal{N} \leftarrow \Omega(L_s)$  and  $\mathcal{M} \leftarrow \emptyset$ .

**for all**  $p_i \in P$  **do**

    Assign  $\mathbf{C} \in \mathcal{N}$  to the server  $p_i$  and  $\mathcal{N} \leftarrow \mathcal{N} \setminus \{\mathbf{C}\}$ .

**end for**

**while**  $\ell \leq \#\Omega(L_s)$  **do**

**if** the flag  $F_i$  of the server  $p_i \in P$  is *off* **then**

        Request the server  $p_i$  to send  $\mathcal{M}_i$  to the client.

$\ell \leftarrow \ell + 1$  and  $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{M}_i$ .

**if**  $\#\mathcal{N} \neq 0$  **then**

        Assign  $\mathbf{C} \in \mathcal{N}$  to the server  $p_i$  and  $\mathcal{N} \leftarrow \mathcal{N} \setminus \{\mathbf{C}\}$ .

**end if**

**end if**

**end while**

$\Omega^* \leftarrow \mathcal{M}$ .

**return**  $\Omega^*$  as the set of all mixed cells.

**The server processor**  $p_i \in P$

*Input:* A node  $\mathbf{C} \in \Omega(L_s)$ .

*Output:* A set of mixed cells  $\mathcal{M}_i$ .

Initialize  $\mathcal{M}_i \leftarrow \emptyset$ .

Set a flag  $F_i = on$ .

Carry out Algorithm 3.4.1 with an initialization list  $(\mathcal{N}) \leftarrow \mathbf{C}$  instead of  $\emptyset^m$ , using an input data  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_m)$  and  $k = (k_1, k_2, \dots, k_m)$ .

Collect mixed cells obtained from the execution into a set  $\mathcal{M}_i$ .

Set a flag  $F_i = off$ .

### 5.3 Numerical Results on Multiple Processors

The parallel method of our dynamic enumeration, based on Algorithm 5.2.1, was implemented using MPI (Message Passing Interface) for network communication between multiple processors. MPI is a standard library for realizing message-passing on parallel computers. All numerical experiments in the successive two subsections were executed on TSUBAME Grid Cluster at Tokyo Institute of Technology. The cluster is composed of 657 Sun Fire X4600 nodes, where each node has 8 sockets of Dual-Core Opteron, i.e., 16 CPU cores. The overall system has 10512 cores, connected through Infiniband. At the present time, the total peak performance reaches to 56 TFlops, and it is ranked in the top 20 among most high-performance computer systems in the world according to the TOP 500 Supercomputing sites. Among these nodes, we performed the experiments on the nodes equipped with 2.4 GHz Opteron processors with 32 GB memory, running Linux.

#### 5.3.1 Large Scale Polynomial Systems

We test how the parallel implementation of dynamic enumeration is powerful for the large-scale polynomial systems in the benchmark problems utilized for the evaluation of DEMiCs in Subsection 4.3.1; the *chandra-n*, *cyclic-n*, *economic-n*, *katsura-n* and *noon-n* problems. In the numerical experiments, 64 processors are used as the servers  $p_i \in P = \{p_1, p_2, \dots, p_r\}$  in Algorithm 5.2.1. Namely,  $r = 64$ .

The choice of a starting index set  $L_s$  has an effect on efficiency of our parallel method. This is because  $L_s$  specifies  $\Omega(L_s)$  whose size could be the main factor that determines an amount of the task on each server processor  $p_i \in P$ . For the estimation of the average amount of computation on each server, we use the following measure

$$\gamma := \frac{\#\Omega(L_s)}{r}.$$

Suppose that the number of server processors  $r$  be fixed. If a starting index set  $L_s$  is chosen so that  $\gamma$  is small, then our parallel method is considered to be coarse-grained. Conversely, if  $L_s$  is chosen so that  $\gamma$  is large, then it is fine-grained. In the preliminary numerical results, our parallel method showed good performance for these problems in terms of computational time when  $\gamma$  is around from  $10^2$  to  $10^3$ . In the experiments, we assume that the size of the targets of problems is around  $n = 20$ . Therefore, a starting index set  $L_s$  for each benchmark problem is chosen as Table 5.1. Here, we set the support of the  $i$ th polynomial from top

Table 5.1: How to choose  $L_s$  for the large-scale benchmark polynomial systems

system	<i>chandra-n</i>	<i>cyclic-n</i>	<i>economic-n</i>	<i>katsura-n</i>	<i>noon-n</i>
$L_s$	$\{1, 2\}$	$\{1, 2, 3\}$	$\{1, n\}$	$\{1, 2\}$	$\{1, 2\}$

as  $\mathcal{A}_i$  for each of the forms of problem described in Subsection 3.2.1, Subsection 4.2.1 and 4.2.3. For example, in case of the economic- $n$  problems, the nodes at the first and second level of an enumeration tree  $T$  explored by the dynamic enumeration method are composed of the elements of  $\mathcal{A}_1$  and  $\mathcal{A}_n$ , respectively.

Table 5.2: Average computational time for the large-scale polynomial systems for 10 times trial with each different lifting value (64 processors)

system	size ( $n$ )	# mixed cells	mixed volume	ave.time
chandra- $n$	$n = 24$	5,102,875	8,388,608	3h19m49.9s
	$n = 25$	10,615,401	16,777,216	9h52m9.9s
cyclic- $n$	$n = 16$	5,118,988	135,555,072	3h42m49.5s
	$n = 17$	20,227,483	601,080,390	32h31m56.5s
economic- $n$	$n = 22$	142,878	1,048,576	3h42m8.5s
	$n = 23$	260,520	2,097,152	17h24m49.5s
katsura- $n$	$n = 17$	4,895	131,072	2h11m17.1s
	$n = 18$	9,729	262,144	8h21m17.8s
noon- $n$	$n = 23$	5,839,478	94,143,178,781	1h38m51.4s
	$n = 24$	10,860,032	282,429,536,433	4h50m38.5s
	$n = 25$	25,611,522	847,288,609,393	15h39m19.0s

The experiments were carried out by changing the lifting values 10 times for each system. Table 5.2 summarizes the numerical results. The column “# mixed cells” presents the average number of mixed cells generated by each trial and “mixed volume” is the value of mixed volume. The average computational time required by the execution of Algorithm 5.2.1 is exhibited in the column “ave.time”. As the definition of computational time, we use the difference between the time when all processors  $p_i \in P$  start Algorithm 5.2.1 and the time when they all finish it. The numerical results show that the parallel implementation of dynamic enumeration can compute all mixed cells and the mixed volume for larger polynomial systems such as the chandra-25, cyclic-17, economic-23, katsura-18 and noon-24, 25 systems, compared with the serial implementation. The conjecture in [43] is referred to the number of solutions of the cyclic- $n$  problems; if the number of solutions is finite, then it equals to  $\frac{(2n-2)!}{(n-1)!^2}$ . According to the paper [50], it was proven by Haagerup in case where  $n$  is prime number. Certainly, the mixed volume for the cyclic-17 polynomial system, 601,080,390, obtained by our parallel method supports the theoretical result.

To evaluate the performance of parallel algorithm precisely, we need to pay attention to the effect from parallel computing environment. Generally, the computational time of parallel algorithm is affected by the distance among processors; as the length of network cable

between each node with server and client processor becomes longer, the communication time increases. As stated above, 1 master processor and 64 server processors on Tsubame Grid Cluster made up of about 10000 processors were used in the numerical experiment. Here, the processors utilized on the parallel computing system vary according to each execution of experiment because the processors for handling a new job are automatically designated when it is requested by user. Accordingly, even if we have the same input data, for every execution, the computational time may change significantly. Also, there is a possibility that a new job is distributed to processors which are working on our parallel algorithm and then these processors need to handle the new job as well as the job requested by our algorithm. Therefore, the computation time of our algorithm may suffer from the effect of other jobs. To investigate the effect caused by parallel computing environment, we conducted numerical experiments using the same lifting value for each polynomial system in the above experiments. These experiments were performed 5 times for each system. Table 5.3 shows the statistics on the computational time. The column “min.time” and “ave.time” give the minimum computational time among 5 trial and the average one, respectively. From the table, we confirm that the effect from the parallel computing environment on the execution of algorithm is vanishingly small.

Table 5.3: Statistics on computational time for the large-scale polynomial systems with a specific lifting value (64 processors)

system	size ( $n$ )	min.time	ave.time
chandra- $n$	$n = 24$	2h32m49.1s	2h36m24.7s
	$n = 25$	7h21m37.3s	7h44m19.6s
cyclic- $n$	$n = 16$	3h34m31.7s	3h34m45.7s
	$n = 17$	32h39m52.1s	32h44m12.3s
economic- $n$	$n = 22$	3h37m31.1s	3h49m59.6s
	$n = 23$	16h49m9.0s	16h55m47.1s
katsura- $n$	$n = 17$	1h32m13.3s	1h32m34.6s
	$n = 18$	7h59m9.2s	8h0m47.0s
noon- $n$	$n = 23$	1h36m58.7s	1h39m21.2s
	$n = 24$	4h57m14.3s	5h11m44.1s
	$n = 25$	15h25m1.7s	15h34m47.8s

### 5.3.2 Scalability

Next, we investigate the scalability of the parallel implementation of dynamic enumeration for the large size polynomial systems in the chandra- $n$ , cyclic- $n$ , economic- $n$ , katsura- $n$

and noon- $n$  problems to be able to compute all mixed cells and the mixed volume within a reasonable computational time on a single processor. In this subsection, we displays the numerical results for the chandra-23, economic-21 and noon-22 problems because the results in the other problems denoted the same tendency as the above three problems. In the numerical experiments, the starting index set  $L_s$  for each test problem was chosen according to Table 5.1. The experiments were performed 10 times for each problem

Table 5.4: Average computational time and ratio with varying number of server processors

# server processors	chandra-23		economic-21		noon-22	
	ave.time	ratio	ave.time	ratio	ave.time	ratio
$r = 1$	17h42m59.2s	1.00	46h21m43.6s	1.00	12h56m17.3s	1.00
$r = 2$	9h0m26.0s	1.97	23h20m37.9s	1.99	6h34m13.9s	1.97
$r = 4$	4h39m20.1s	3.81	11h38m58.2s	3.98	3h17m59.1s	3.92
$r = 8$	2h35m13.0s	6.85	5h51m21.1s	7.92	1h46m5.9s	7.32
$r = 16$	1h45m56.7s	10.03	2h45m39.0s	16.79	1h1m2.7s	12.72
$r = 32$	1h20m5.2s	13.27	1h41m43.6s	27.34	42m36.6s	18.22
$r = 64$	1h10m36.9s	15.05	1h7m48.8s	41.02	35m58.4s	21.58

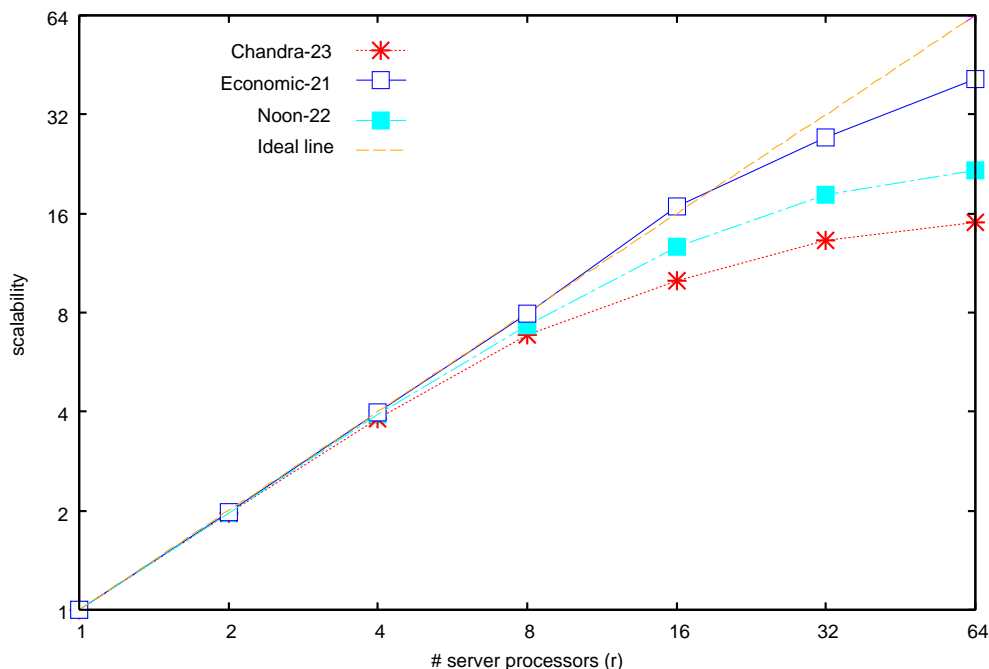


Figure 5.3: Scalability for each test problem

by choosing the different lifting values. In Table 5.4, we list the numerical results. The column “ave.time” presents the average computational time, which is measured by use of the same definition in the numerical experiments of the previous subsection, for the

corresponding problems on  $r$  server processors. The column “ratio” indicates the ratio between the computational time obtained by  $r$  processors and a single processor, and thus each value in the column is regarded as scalability. Figure 5.3 depicts the scalability for each test problem. The horizontal line is the number of processors and the vertical line the scalability, both in the log scale by 2. From the numerical results, we see that our parallel implementation of dynamic enumeration could achieve good scalability for the economic-21 and noon-22 problems, and in particular, for the economic-21 problem it is close to ideal. Table 5.5 summarizes the statistics on computational time, which reveals the effect from parallel computing environment for the chandra-23, economic-21 and noon-22 problems with varying number of server processors. The numerical experiments were conducted 5 times with the same lifting value for each problem. The column “min.time” and “ave.time” provide the minimum computational time among 5 trial and the average one, respectively. We are convinced from the table that there is only little effect from the parallel computing environment.

Table 5.5: Statistics on computational time with varying number of server processors

# server processors	chandra-23		economic-21	
	min.time	ave.time	min.time	ave.time
$r = 1$	16h45m17.7s	17h2m4.7s	45h26m56.6s	45h43m22.984s
$r = 2$	10h38m58.8s	10h58m50.4s	22h47m23.6s	22h57m38.0s
$r = 4$	4h28m52.8s	4h33m56.2s	11h23m17.5s	11h26m7.0s
$r = 8$	2h44m20.8s	2h51m24.2s	5h43m51.6s	5h45m43.5s
$r = 16$	1h43m20.0s	1h44m6.4s	2h50m31.7s	2h50m55.5s
$r = 32$	1h17m22.5s	1h17m55.0s	1h41m8.9s	1h41m24.7s
$r = 64$	1h4m9.4s	1h6m42.3s	1h6m11.2s	1h6m27.2s

# server processors	noon-22	
	min.time	ave.time
$r = 1$	12h21m2.0s	12h35m10.7s
$r = 2$	6h20m39.6s	6h33m33.4s
$r = 4$	3h15m38.0s	3h20m14.5s
$r = 8$	1h43m53.1s	1h45m32.0s
$r = 16$	1h0m49.4s	1h2m31.5s
$r = 32$	41m49.8s	42m5.7s
$r = 64$	34m29.9s	34m45.0s

The scalability for the cyclic-15 problem and katsura-16 problem have a similar tendency to the economic-23 problem and noon-22 problem, respectively, and each speed-up ratio for computation time between a parallel implementation on 64 server processors and a serial implementation is 48.77 and 21.35. However, our parallel method could not obtain satisfying scalability for the chandra-23 problem in contrast to the other problems.

Let us examine the reason. In addition to the numerical results of the chandra-23 problem when 64 server processors are used, by way of comparison, we also consider those of the economic-21 problem, for which our parallel method attains high scalability. The

Table 5.6: Computational time for the chandra-23 and economic-21 problems obtained in one trial of the numerical experiments, using 64 server processors

system	chandra-23	economic-21
comp.time	3750.3s (1h2m30.3s)	3175.6s (52m55.6s)

computational time for these problems obtained in one trial by use of 64 server processors is listed in Table 5.6. Note that the table provides results in one trial using some specific lifting value  $\omega_i(\mathbf{a})$  for each element  $\mathbf{a}$  in a support  $\mathcal{A}_i$ , whereas in Table 5.4 those are the average of the computational time resulting from 10 times trial with the different lifting values  $\omega_i(\mathbf{a})$ .

We first observe the total run time of Algorithm 5.2.1 for these two problems on each server processor  $p_i \in P$ . Here, we give the definition of *total run time*  $T_{p_i}$  on a server processor  $p_i$  as follows. If a server  $p_i$  receives  $n_s$  root nodes  $\mathbf{C}_1, \mathbf{C}_2, \dots, \mathbf{C}_{n_s} \in \Omega(L_s)$  from the client, and each computational time required by the execution of Algorithm 5.2.1 with an input data  $\mathbf{C}_j$  on the server  $p_i$  is  $T_j$ , then we define the total run time on  $p_i$  as  $T_{p_i} = \sum_{j=1}^{n_s} T_j$ . The bar graphs of Figure 5.4 and 5.5 depict the total run time on 64 server processors  $p_1, p_2, \dots, p_{64}$  for the chandra-23 and economic-21 problems, respectively. The horizontal line is each index of server processors  $p_1, p_2, \dots, p_{64}$  and the vertical line the total run time in seconds. Table 5.7 summarizes the statistics. From the numerical results, it is able to understand that for the economic-21 problem, the variation of each total run time on server processors is small but it is large for the chandra-23 problem. In particular, for the chandra-23 problem, the maximum total run time at the server  $p_{39}$  stands out against the others, and it occupies the majority of the computational time.

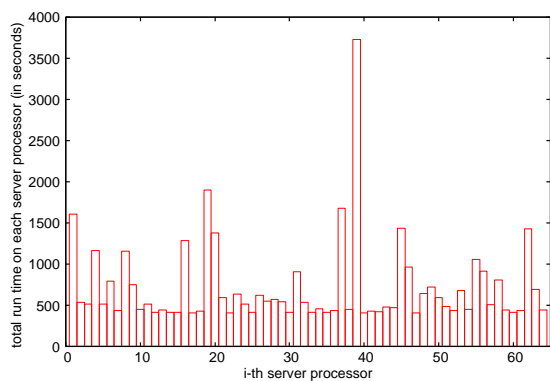


Figure 5.4: Chandra-23 problem

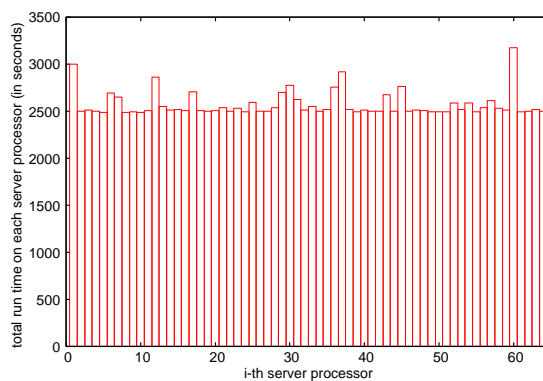


Figure 5.5: Economic-21 problem

Table 5.7: Statistics of total run time (in seconds)

system	chandra-23	economic-21
average	533.0	2571.6
standard deviation	521.3	133.5
maximum total run time	3728.3 (at server $p_{39}$ )	3175.1 (at server $p_{60}$ )

Next, we inspect computational time required by finding mixed cells in equally-partitioned subtrees of  $T$  by the dynamic enumeration. More precisely, by the phrase “*computational time of a subtree  $T_s$* ”, we mean the time from when Algorithm 5.2.1 for a server processor starts to find mixed cells in the subtree  $T_s$  whose root node  $\mathbf{C} \in \Omega(L_s)$  is given from the client as input data till when it finishes. Table 5.8 exhibits the statistics on an amount of the computational time of every subtree for the chandra-23 and economic-21 problems. In

Table 5.8: Statistics of the subtrees

amount of computational time		chandra-23		economic-21	
		# subtrees	percentage	# subtrees	percentage
tiny	(0 min - 1 min)	76,051	99.836%	43,482	98.599%
small	(1 min - 10 min)	111	0.146%	594	1.347%
medium	(10 min - 30 min)	13	0.017%	23	0.052%
large	(30 min - )	1	0.001%	1	0.002%

the row “tiny”, we list the total number of subtrees whose computational time are from greater than 0 minutes to less than or equal to 1 minutes, and the percentage to total. Similarly, the row “small” is from greater than 1 minutes to less than or equal to 10 minutes, “medium” from greater than 10 minutes to less than or equal to 30 minutes, and “large” from greater than 30 minutes. The table tells us that almost every computational time of subtree is less than 1 minutes and however, there exist one large size subtree for each problem. Let  $T_s^*$  and  $T_s^{**}$  denote each large size subtree of the chandra-23 problem and economic-21 problem. The computational time of  $T_s^*$  is 3460.7 seconds, whereas that of  $T_s^{**}$  is 1888.6 seconds. Additionally, for the chandra-23 problem, there exist at least 14 subtrees including  $T_s^*$  which are greater than the average run time 533.0 seconds. Therefore, the existence of such large size subtrees in the chandra-23 problem is considered to be the main reason why the scalability obtained by our parallel method for the problem is poor in contrast to the economic-21 problem. For the chandra-23 problem, the server  $p_{39}$  worked for finding mixed cells in the large size subtree  $T_s^*$ , and the computational time of  $T_s^*$  holds the majority of the total run time 3728.3 seconds at the server. On the other hand, for the economic-21 problem, all subtrees are lower than the average run time 2571.6 seconds. A large size subtree  $T_s^{**}$  was treated at the server  $p_{60}$ , and the computational time of  $T_s^{**}$  is about 60 percent of the total run time 3175.1 seconds at the server. Thus, to improve the

scalability of the chandra-23 problem, we may need to reconsider the method for partitioning a tree  $T$  into the subtrees  $T_s$  in Algorithm 5.2.1. In future directions of Chapter 6, we will discuss the issue and refer to the suggestions for the settlement of it.

# Chapter 6

## Conclusions and Future Directions

Various techniques for solving a system of polynomial equations have been proposed and enhanced by many researchers during the past few decades. Indeed, we can find hundreds of papers related to the topics. This is because it is an important subject in mathematical theory and practical applications. However, it is considered to be still hard problem, and in particular, when the size of a polynomial system is large.

The polyhedral homotopy proposed by Huber and Sturmfels [29] is expected to make it possible to overcome the hardness. Although it is a promising method, the construction stage of polyhedral homotopies is the main computational bottleneck in large size polynomial system solving. Therefore, in this thesis, we explored to seek the answers to the question in the introduction; how do we construct polyhedral homotopies for large-scale polynomial systems? To answer the motivated question, we made three proposals.

- **Design of dynamic enumeration method:**

There exist several formulations [17, 61, 38, 57, 22] for finding all mixed cells through Huber and Sturmfels's enumeration procedure, stated in Subsection 2.4.2. In the formulations, an enumeration tree can be constructed among a family of systems of linear inequalities. Each node has a linear inequality system, and in particular a feasible leaf node is corresponding to a mixed cell. Also, we can say that if a node is infeasible then so are all of its descendant nodes; hence, the subtree having the infeasible node as a root can be pruned because it does not contain any mixed cells. There are variety of ways for building such an enumeration tree, and it is crucial point for the efficient implementation of enumerating mixed cells how we construct enumeration trees. In almost every existing work [61, 38, 57, 22], the structure of a tree is fixed before starting the enumeration of mixed cells. By contrast, we proposed the dynamic construction of a tree so that most of the child nodes become infeasible and pruned when a node branches into its child nodes. At the time, the effective use of information obtained from nodes during carrying out the enumeration plays an essential role in our dynamic enumeration. Under such a situation, the dual simplex methods provide an advantage. The Lift-Prune method presented by Emirs and Canny

[17] also constructs a dynamic enumeration tree. However, the strategy of dynamic construction is different from our method. Their method selects the “best” child node set among all the candidates as a parent node branches, whereas our method chooses an “appropriate” one, and then the computational cost for finding such a child node set is much less. Indeed, in Subsection 4.2.1, we saw the numerical results obtained by MVLP and MixedVol [23], which implement the Lift-Prune method and the static enumeration method by Gao and Li [22]. These make us recognize that the Lift-Prune method is far behind the static enumeration by Gao and Li in view of computational time. This is due to the best selection strategy in the method. Although it certainly makes a size of an enumeration tree much smaller over the other methods [61, 38, 57, 22], the cost for finding the best child node set is too costly, and it leads to the substantial increase in the total cost for finding all mixed cells. In Subsection 4.3.1, through the various test problems, we showed the performance of DEMiCs which is developed for the implementation of our dynamic enumeration, compared with MixedVol which is the most efficient software package among the others [17, 60, 61, 62, 38, 57] at the present time. Table 6.1 summarizes some numerical results for well-known benchmark problems obtained in the subsection. From the table, we see that DEMiCs can compute all the mixed cells and the mixed volume for a larger polynomial system than MixedVol.

Table 6.1: Largest size of the problems to be able to compute all mixed cells and the mixed volume by DEMiCs and MixedVol on a 2.4 GHz Opteron 850 within 24 hours

	chandra- $n$	cyclic- $n$	economic- $n$	katsura- $n$	noon- $n$
DEMiCs	23	15	21	16	22
MixedVol	19	14	20	14	18

- **Utilization of the support structure in dynamic enumeration:**

A polynomial system  $\mathbf{f} = (f_1, f_2, \dots, f_n)$  arising from real-world problem sometimes has a special structure in the supports  $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n$  so that some of the supports are equal to each other. In such a case, it is called semi-mixed, and particularly, unmixed if they are all equal. Conversely, it is called fully mixed if they are all distinct. In [29], Huber and Sturfels point out the fact that only distinct supports contribute the mixed volume computation. It means that the size of enumeration tree built for a semi-mixed type polynomial system is made smaller because the height of the tree is determined by the number of distinct supports. Here, there are concerns over resulting computational cost when we utilize a support structure in the enumeration of mixed cells. As we mentioned in Subsection 3.4.2, although the height of enumeration tree can be shortened by use of the property of semi-mixed support structure, a parent node generates much more child nodes, compared to the case of a fully-mixed type polynomial system, and thus, it may not be expected that the total computational cost

is reduced dramatically. In Subsection 3.4.2, we incorporated the efficient bounding techniques into our dynamic enumeration in order to find and prune infeasible child nodes, which are redundant ones and whose descendant nodes never become mixed cells, generated from a parent node. The basic idea of the bounding techniques is the sequential application of one point tests, proposed by Li and Li [38], to child nodes. So, for semi-mixed polynomial systems, it is expected that our dynamic enumeration utilizing the special structure of their supports can significantly speed-up the enumeration of mixed cells in contrast to the case where these are regarded as a fully mixed type. Indeed, we saw in Subsection 4.3.2 that the computational time required by the generation of all mixed cells and the computation of mixed volume for PRS-10, RRS-12 and Nine-point problems from mechanism design [65, 55] can be improved significantly by utilizing their support structure. In the subsection, we also compared the performance of DEMiCs, which implements our dynamic enumeration taking account of semi-mixed support structure, and MixedVol, which also utilizes semi-mixed structure for increasing computational efficiency. The numerical results revealed that DEMiCs has a big speed advantage over MixedVol for semi-mixed polynomial systems with many distinct supports. However, if the support structure of polynomial systems is as close as an unmixed type, DEMiCs is behind MixedVol in computational time. We will refer to the suggestions for the settlement of the issue in future directions.

- **Parallel implementation of dynamic enumeration:**

Homotopy methods are well-suitable to parallel computing. This is a notable advantage when we challenge large-scale polynomial solving. During the last decade, the field of high-performance computing technology has been developed rapidly. Now, the technology provides enormous computing power and allows us to try tough problems which could not be solved before because of the huge cost requirement. Actually, the paper [56] reports that the parallel implementations of path following stage in homotopy methods made new results for solving large-scale polynomial systems. Similarly, the construction stage of polyhedral homotopies can be parallelized. In Subsection 5.2.2, we presented the parallel algorithm for implementing our dynamic enumeration. The parallel method utilizes the property that an enumeration tree is divided into subtrees whose any nodes are not shared with each other. In Subsection 5.3.1, the numerical results gave the impression that our parallel method is so powerful; it succeeded to compute all mixed cells and the mixed volume for a larger size polynomial systems, compared with the serial implementation. Table 6.2 indicates the largest size of the benchmark problems which the parallel implementation of dynamic enumeration on 64 server processors and the serial implementation were able to solve within 24 hours. In the subsection, we also investigated the scalability of the parallel method through the large size polynomial systems in the chandra- $n$ , cyclic- $n$ , economic- $n$ , katsura- $n$  and noon- $n$  problems. For the majority of the problems, our parallel method could attain good scalability. In fact, for the cyclic-15 and economic-21 problems, the speed-up ratio between computational time on 64 server processors

Table 6.2: Largest size of the problems to be able to compute all mixed cells and the mixed volume by the parallel implementation on 64 processors and the serial implementation within 24 hours

	chandra- $n$	cyclic- $n$	economic- $n$	katsura- $n$	noon- $n$
parallel implementation	25	16	23	18	25
serial implementation	23	15	20	16	22

and a single server processor reaches to about 49 and 41, respectively. However, for the chandra-23 problem, it is about 15. The main reason is the existence of only one large size subtree. Although almost every subtree is very small and it takes no more than 1 minutes for finding mixed cells in the subtree, the large size subtree requires 1 hours for that purpose, so that it occupies the most part of the computational time for finding all mixed cells by our parallel method. Therefore, if an enumeration tree built for the chandra-23 problem was divided into subtrees so that such a large size subtree never appear, the scalability could be improved. The suggestions for increasing scalability under such a situation will be mentioned in future directions.

The numerical results in this thesis show that our proposed methods brought new results for the generation of all mixed cells (that is, a construction of a family of polyhedral homotopy functions) and the computation of mixed volume for large-scale polynomial systems, which have been considered to be difficult. At least, the author believes these proposals to become good answers to the question posed in the introduction. So, we could expect that these methods open the door to solving such large-scale polynomial systems by the polyhedral homotopy method.

Although we succeeded to provide the efficient methods for constructing polyhedral homotopies, our implementation of path following method in [26] needs to be improved because it is far behind the most recent methods incorporated in Bertini [2] and HOM4PS-2.0 [33] in terms of computational efficiency and stability. According to the talk by Sommese at the algebraic geometry and applications seminar of the institute for mathematics and its applications (IMA) in 2006, the most sophisticated path following techniques can track 10 paths per second on a desktop personal computer. In practice, the C software package Bertini [2] developed and published by their research group performs homotopy continuation with efficiency and stability using total-degree start systems or multihomogeneous-degree start systems. Therefore, the computational time for tracking all homotopy paths could be roughly estimated if the path count is known, so that our numerical results for constructing polyhedral homotopies may serve as the purpose for predicting the maximum size of polynomial systems to be able to compute all the isolated zeros within a given length of computation time by the polyhedral homotopy method. In [33], they report excellent numerical results HOM4PS-2.0 written in FORTRAN 90. The software is the updated version of HOM4PS [21] and implements the polyhedral homotopy method. The update are in both stages of the

method: a construction of a family of polyhedral homotopy functions and a path following. In particular, the idea of our dynamic enumeration is employed to increase computational efficiency for constructing polyhedral homotopies. For example, their software HOM4PS-2.0 can compute all isolated zeros of the economic-18 and katsura-20 problems in 1 hours 51 minutes and 8 hours 58 minutes respectively on a 2.2 GHz Pentium IV with 1 GB memory. These are remarkable results because our software PHoM takes 1 hours 50 minutes for solving the economic-13 problems on a 2.4 GHz Pentium IV with 512 MB memory, as stated in Subsection 3.2.1.

The following are future directions of our research.

- **Semi-mixed polynomial systems with a few distinct supports:**

If a semi-mixed polynomial system has only a few distinct supports, it is difficult to expect the advantage of dynamic enumeration because one has no major differences between an enumeration tree built by dynamic and static methods. Specifically, in case of an unmixed polynomial system, our dynamic method is the same as the static enumeration by Gao and Li [22] in principle. Despite this, the numerical results in Subsection 4.3.2 show that DEMiCs by our method is inferior to MixedVol by Gao and Li in computational time for unmixed systems. The reason is considered as follows. Gao and Li make good use of a relation table for finding infeasible nodes in an enumeration tree, and as a result, improve computational time considerably. According to the talk by Li, the use of the table also helps us to reduce the cost in solving LP problems by the simplex method in order to check the feasibility of nodes in an enumeration tree. Although our method constructs the table, it appears that the well skilled use leads to the key to increasing computational efficiency of our method for semi-mixed polynomial systems with a few distinct supports.

- **Further improvement in scalability of our parallel method**

As we have mentioned in Subsection 5.2.1, the redistribution technique is introduced in the development of PHoMpara [27] to increase scalability of the parallel implementation of finding all mixed mixed cells through a static enumeration tree. As well as our parallel implementation of dynamic enumeration, the parallelization of static enumeration in PHoMpara divides an enumeration tree into subtrees, and assigns each subtree to the server processor in order to enumerate feasible leaf nodes in the subtree. The basic idea of the redistribution technique is that a server processor working for long time to seek feasible leaf nodes in a subtree is forced to stop without completing the job, and the nodes whose feasibility is not checked are redistributed to the other idle server processors. The paper [27] shows the effectiveness of the technique through the economic-15, 16 problems and the katsura-14, 15 problems; it gains success to reducing idle time for each server processor. According to the analysis in Subsection 5.3.2, it is considered that the poorness of scalability by our parallel method for the chandra-23 problem is caused by the existence of large size subtrees which require much above average computational time for finding feasible leaf nodes (that is, mixed

cells). So, we could expect that the redistribution technique has potential to improve the scalability of the parallel method under such a situation.

# Bibliography

- [1] E. Allgower and K. Georg, “Numerical continuation methods,” Springer-Verlag (1990).
- [2] D. J. Bates, J. D. Hauenstein, A. J. Sommese and C. W. Wampler, “Bertini: Software for Numerical Algebraic Geometry,” Available at <http://www.nd.edu/~sommese/bertini>.
- [3] D. N. Bernshtein, “The number of roots of a system of equations,” *Funct. Anal. Appl.*, **9**, pp. 183–185 (1975).
- [4] U. Betke, “Mixed volumes of polytopes,” *Archiv der Mathematik*, **58**, pp. 388–391 (1992).
- [5] L. Billera and B. Sturmfels, “Fiber polytopes,” *Ann. of Math.*, **135**, pp. 527–549 (1992).
- [6] W. Boege, R. Gebauer, and H. Kredel, “Some examples for solving systems of algebraic equations by calculating Groebner bases,” *J. Symbolic Comput.*, **2**, pp. 83–98 (1986).
- [7] G. Björk and R. Fröberg, “A faster way to count the solutions of inhomogeneous systems of algebraic equations,” *J. Symbolic Comput.*, **12**(3), pp. 329–336 (1991).
- [8] W. Boege, R. Gebauer, and H. Kredel, “Some examples for solving systems of algebraic equations by calculating Groebner bases,” *J. Symbolic Computation* **2**, pp. 83–98 (1986).
- [9] S. Chandrasekhar, “Radiative Transfer,” Dover, New York (1960).
- [10] V. Chvátal, “LINEAR PROGRAMMING,” W. H. Freeman (1983).
- [11] D. A. Cox, J. Little and D. O’Shea, “Ideals, Varieties, and Algorithms,” Springer-Verlag, New York, 2nd edition (1996).
- [12] D. A. Cox, J. Little and D. O’Shea “Using Algebraic Geometry,” Springer-Verlag, New York, 2nd edition (2004).
- [13] 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**, pp. 83-97 (2003).

- [14] R. S. Datta, “Using computer algebra to find nash equilibria,” In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, pp. 74–79 (2003).
- [15] F. J. Drexler, “Eine methode zur Berechnung sämtlicher Lösungen von Polynomgleichungssystemen,” *Numer. Math.*, **29**, pp. 45–58 (1977).
- [16] M. Dyer, P. Gritzmann and A. Hufnagel, “On the complexity of computing mixed volumes,” *SIAM J. Comput.*, **27**(2), pp. 356–400 (1998).
- [17] I. Z. Emiris and J. F. Canny, “Efficient incremental algorithms for the sparse resultant and the mixed volume,” *J. Symbolic Comput.*, **20**, pp. 117–149 (1995). Software available at <http://cgi.di.uoa.gr/~emiris/index-eng.html>.
- [18] I. Z. Emiris and T. Giordano, “Distributed computation of mixed volume,” *Proc. 13th European Workshop on Computational Geometry* (1997).
- [19] FRISCO - A Framework for Integrated Symbolic/Numeric Computation, 1996–1999. Final report available at <http://www.nag.co.uk/projects/FRISCO.html>.
- [20] T. Gao and T. Y. Li, “Mixed volume computation via linear programming,” *Taiwan J. of Math.*, **4**, pp. 599–619 (2000).
- [21] T. Gao, T. Y. Li and X. Li, The software package HOM4PS is available at <http://www.mth.msu.edu/~li/>.
- [22] T. Gao and T. Y. Li, “Mixed volume computation for semi-mixed systems,” *Discrete Comput. Geom.*, **29**(2), pp. 257–277 (2003).
- [23] 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/>.
- [24] C. B. Garcia and W. I. Zangwill, “Finding all solutions to polynomial systems and other systems of equations,” *Math. Program.*, **16**(1), pp. 159–176 (1979).
- [25] I. M. Gelfand and M. M. Kapranov and A. V. Zelevinsky, “Discriminants, Resultants, and Multidimensional Determinants,” Birkhäuser, Boston (1994).
- [26] T. Gunji, S. Kim, M. Kojima, A. Takeda, K. Fujisawa and T. Mizutani, “PHoM – A polyhedral homotopy continuation method,” *Computing*, **73**(1), pp. 57–77 (2004).
- [27] T. Gunji, S. Kim, K. Fujisawa and M. Kojima, “PHoMpara – Parallel implementation of the polyhedral homotopy continuation method,” *Computing*, **77**(4), pp. 387–411 (2006).

- [28] L. Gurvits and A. Samorodnitsky, “A deterministic algorithm for approximating the mixed discriminant and mixed Volume, and a Combinatorial Corollary,” *Discrete Comput. Geom.*, **27**(4), pp. 531–550 (2002).
- [29] B. Huber and B. Sturmfels, “A polyhedral method for solving sparse polynomial systems,” *Math. Comp.*, **64**, pp. 1541–1555 (1995).
- [30] B. Huber and J. Verschelde, “A polyhedral end games for polynomial continuation,” *Num. Alg.*, **18**(1), pp. 91–108 (1998).
- [31] S. Katsura, “Theory of spin glass by the method of the distribution function of an effective field,” *Progr. Theor. Phys. Suppl.* **87**, pp. 139–154 (1986).
- [32] S. Kim and M. Kojima, “Numerical stability of path tracing in polyhedral homotopy continuation methods,” *Computing*, **73**, pp. 329–348 (2004).
- [33] T. L. Lee, T. Y. Li and C. H. Tsai, “HOM4PS-2.0: A software package for solving polynomial systems by the polyhedral homotopy continuation method,” Preprint.
- [34] T. Y. Li, T. Sauer and J. A. York, “The cheater’s homotopy: an efficient procedure for solving systems of polynomial equations,” *SIAM J. Numer. Anal.*, **26**, pp. 1241–1251 (1989).
- [35] T. Y. Li and X. Wang “The BKK root count in  $\mathbf{C}^n$ ,” *Math. Comp.*, **60**, pp. 669–680 (1993).
- [36] T. Y. Li and X. Wang “Counterexamples to the connectivity of the mixed cells,” *Discrete Comput. Geom.*, **20**, pp. 515–521 (1998).
- [37] T. Y. Li, “Solving polynomial systems by polyhedral homotopies,” *Taiwan J. of Math.*, **3**, pp. 251–279 (1999).
- [38] T. Y. Li and X. Li, “Finding mixed cells in the mixed volume computation,” *Foundation Comput. Math.*, **1**, pp. 161–181 (2001). Software available at <http://www.math.msu.edu/~li/>.
- [39] T. Y. Li, “Numerical solution of polynomial systems by homotopy continuation methods,” In Handbook of Numerical Analysis, Vol. XI, Special Volume: Foundations of Computational Mathematics, edited by F. Cucker, pp. 209–304. North-Holland (2003)
- [40] J. T. Linderoth and M. W. P. Savelsbergh, “A computational study of branch and bound search strategies for mixed integer programming,” *INFORMS J. Comput.*, **11**, pp. 173–187 (1999)
- [41] T. Mizutani, A. Takeda and M. Kojima, “Dynamic Enumeration of All Mixed Cells,” *Discrete Comput. Geom.*, **37**(3), pp. 351–367 (2007).

- [42] T. Mizutani and A. Takeda, “DEMiCs: A software package for computing the mixed volume via dynamic enumeration of all mixed cell,” to appear in *Software for Algebraic Geometry, I.M.A. Volumes in Mathematics and its Applications*, M. Stillman, N. Takayama and J. Verschelde (eds.). Software available at <http://www.is.titech.ac.jp/~mizutan8/DEMiCs/>.
- [43] H. M. Möller, “Gröbner bases and numerical analysis,” In *Gröbner Bases and Applications*, London Mathematical Lecture Note Series, Vol. 251, pp. 159–178, B. Buchberger and F. Winkler (eds.), Cambridge University Press (1998).
- [44] A. Morgan, “Solving polynomial systems using continuation for engineering and scientific problems,” Prentice-Hall (1987).
- [45] A. P. Morgan and A. J. Sommese, “A homotopy for solving general polynomial systems that respect  $m$ -homogeneous structures,” *Appl. Math. Comput.*, **24**(2), pp. 101–113 (1987).
- [46] A. P. Morgan and A. J. Sommese, “Coefficient-parameter polynomial continuation,” *Appl. Math. Comput.*, **29**, pp. 123–160 (1989).
- [47] V. W. Noonburg, “A neural network modeled by an adaptive Lotka-Volterra system,” *SIAM J. Appl. Math.*, **49**, pp. 1779–1792 (1989).
- [48] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley, New York (1986)
- [49] I. R. Shafarevich, *Basic Algebraic Geometry*, Springer-Verlag, New York (1977)
- [50] A. J. Sommese and J. Verschelde, “Numerical homotopies to compute generic points on positive dimensional algebraic sets,” *Journal of Complexity*, **16**(3), pp. 572–602 (2000).
- [51] A. J. Sommese, J. Verschelde and C. W. Wampler, “Introduction to numerical algebraic geometry,” In *Solving Polynomial Equations*, Series: Algorithms and Computation in Mathematics, Vol. 14, edited by A. Dickenstein and I. Z. Emiris, Springer-Verlag (2005).
- [52] A. J. Sommese and C. W. Wampler, “The Numerical Solution of Systems of Polynomials,” World Scientific, Singapore (2005).
- [53] H. J. Stetter, “Numerical Polynomial Algebra,” SIAM, Philadelphia (2004).
- [54] B. Sturmfels, “Solving Systems of Polynomial Equations,” CBMS Regional Conference Series in Mathematics, No. 97. Published for the Conference Board of the Mathematical Sciences, Washington DC (2002).
- [55] 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), pp. 226–234 (2004).

- [56] H. -J. Su, J. M. McCarthy, M. Sosonkina and L. T. Watson, “Algorithm 857 : POLSYS\_GLP – A parallel general linear product homotopy code for solving polynomial systems of equations,” *ACM Trans. Math. Softw.*, **32**(4), pp. 561–579 (2006).
- [57] 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**, pp. 64–82 (2002). Software available at <http://www.is.titech.ac.jp/~kojima/index.html>.
- [58] J. Verschelde, The database of polynomial systems is in his web site: “<http://www.math.uic.edu/~jan/>.”
- [59] J. Verschelde, P. Verlinden and R. Cools, “Homotopies exploiting Newton polytopes for solving sparse polynomial systems,” *SIAM J. Numerical Analysis*, **31**, pp. 915–930 (1994).
- [60] J. Verschelde and K. Gatermann, “Symmetric newton polytopes for solving sparse polynomial systems,” *Adv. Appl. Math.*, **16**(1), pp. 95–127 (1995).
- [61] J. Verschelde, “Homotopy Continuation Methods for Solving Polynomial Systems,” Ph. D. thesis, Katholieke Universiteit Leuven, Belgium.
- [62] J. Verschelde, K. Gatermann, and R. Cools, “Mixed-volume computation by dynamic lifting applied to polynomial system solving,” *Discrete Comput. Geom.*, **16**(1), pp. 69–112 (1996).
- [63] J. Verschelde, “Algorithm 795: PHCPACK: A general-purpose solver for polynomial systems by homotopy continuation,” *ACM Trans. Math. Softw.*, **25**, pp. 251–276 (1999). Software available at <http://www.math.uic.edu/~jan/>.
- [64] J. Verschelde and Y. Zhuang, *Parallel implementation of the polyhedral homotopy method*, 2006 International Conference on Parallel Processing Workshops, Timothy Mark Pinkston and Fusun Ozguner. (eds.), pp. 481-488.
- [65] C. W. Wampler, A. P. Morgan and A. J. Sommese, “Complete solution of the nine-point path synthesis problem for four-bar linkages,” *ASME J. Mech. Design*, **114**, pp.153–159 (1992).
- [66] L. T. Watson, S. C. Billips, and A. P. Morgan, “Algorithm 652: HOMPACk: A suite for codes for globally convergent homotopy algorithms,” *ACM Trans. Math. Softw.*, **13**(3), pp. 281-310 (1987).
- [67] L. T. Watson, M. Sosonkina, R. C. Melville, A. P. Morgan, and H. F. Walker, “HOMPACk90: A suite of Fortran 90 codes for globally homotopy algorithms,” *ACM Trans. Math. Softw.*, **23**(4), pp. 514–549 (1997).