

國立台灣大學理學院數學系

碩士論文

Department Of Mathematics

College of Science

National Taiwan University

Master Thesis

使用較少記憶體之 F_4 演算法

A time-memory tradeoff in Faugère's algorithm for computing

Gröbner bases

黃筠茹

Yun-Ju Huang

指導教授：陳君明 博士

Advisor : Jiun-Ming Chen, Ph.D.

中華民國 99 年 8 月

August, 2010

序言

研究一直都是一片漫漫無邊際的世界。即便這世界已經有了許多豐碩的果實，但是空白茫然的未知卻更多。而我們的工作便是嘗試要從這茫然中種出一朵花來。這過程是令人焦躁而不安的。雖然現在回頭而顧那裡的確怯生生的長出了一朵不太大的花朵。從決定研究題目，並努力去找到更好的方法去解決，每天每天的苦惱與不確定，現在確實的有了成果。而只要長出了一朵，之後的我們肯定會更有能力以及經驗去栽出更炫目的花。我想這是研究生生涯所要努力教給我們的最重要的事。

首先，我要先感謝我的指導教授陳君明老師。如果不是老師把 F_4 介紹給我，這篇論文就不會產生了，而我也不會有這麼有趣的研究經驗。而不只在學術上提供了我建議，老師也在生活上照顧我許多，更提供我許多新知，使得我的研究生活不至於太過單調與匱乏。

我還要感謝道格與涵教授。這兩位老師給予我的研究極大的幫助，常常在研究上碰到問題都與老師討論並找出解決之道。兩位老師除了研究上，也在生活方面給我很多的照顧與精神支持，使我能夠專心在學術上專研。

真的真的很感謝我所遇到的老師們，包括在數學所教導我其他課程的陳其誠老師、朱樺老師等人。是他們讓不知所措的我，開始學習如何思考，如何踏實的走出一條路，如何讓腦袋裡面一團亂糟糟的想法去歸納整理成爲可描述的具體問題並進而解決。因爲大學時候與研究所讀的科系不同，數學又是一門吸引人但困難的學問，一度自身真的失去了自信，徘徊在學問知識間不得其門而入，無所適從，甚至失去了人生規劃。是老師們帶領我一步一步，讓我在研究裡找到了立足之地，並開始相信自己可以做得到。

另外，還要謝謝研究室裡的好同學們，小花麋鹿、暴力小彤、賢慧小池，與其

他學弟妹們。他們無所不用其極的鞭策與鼓勵，還有帶給我的歡樂，並時時耳提面命提醒我我緊迫的時間，是我每天精神上得支持，陪我一天天的走到現在。研究室的歡樂氣氛，互相的打氣與吐槽，我想是每個研究生最好的安慰。

我也要謝謝我大學的好朋友。雖然大家已經從大學畢業了，各自研究領域也不相同，但是互相的關心常常令我感到安心。尤其大家青眼有加的呵護，常常會有「啊，一直都停留在熱血的青春歲月」這樣的錯覺。當遇到什麼瓶頸，他們都會對我的撒嬌給予溫暖的吐槽，讓我又有動力回到研究裡。

最後，我要感謝我爹娘以及老弟還有其他家人給予我的無條件支持。他們對於我任性的生涯決定從沒有過質疑，而不論我遇到什麼問題，他們永遠張開懷抱接納我的煩惱。在最後這幾個月馬不停蹄根本無暇顧及它事的趕工裡，更是給我生活上最直接及實質的支援。我對於他們的支持非常的感激。如果沒有我的家人，今日這一切都將不在。我會更加的努力，以回報這份愛。



摘要

解多變量多項式方程組，不論是對於破解多變量密碼系統的應用而言或者對於多變量多項式方程組的研究本身，都是個重要的課題。目前大部分有效解多變量多項式系統的演算法，大多是利用多變數環中Gröbner-basis的特性而轉為求Gröbner-basis的演算法，包括XL演算法以及Faugère的 F_4 、 F_5 演算法。

F_4 演算法是一個相當有效率的計算Gröbner-basis的演算法。即便如此，演算法本身的時間跟空間複雜度依然隨變數個數 n 成指數成長。這使得我們在想要解較大的多變量方程組系統情況下，將不可避免的碰到了執行面上得瓶頸。例如，假設現在有一個變數個數 $n = 40$ 且方程式個數 $m = 40$ 的系統，若想要利用 F_4 來解此方程組，現有的絕大部分電腦都會遭遇硬體無法負荷的情形，以致在解出答案前就用光了所有的記憶體。縱然 F_4 本身有機制可以對記憶體用量稍作調整，但是效益很低。

在本篇論文中，我們開始思考下列幾個關於 F_4 的記憶體資源使用的問題：

1. 能不能讓 F_4 演算法，或者其變型，在任何記憶體限制下都能夠執行？
2. 如果不能， F_4 演算法至少需要多少記憶體才能夠執行？
3. 若給予更多記憶體，能不能使修改過的 F_4 演算法執行的更快？

透過回答上述問題，我們觀察到 F_4 的每個步驟的記憶體使用情況，而據此我們針對 F_4 演算法提出一些建議以修改記憶體的使用問題。我們修改過後的 F_4 演算法較原本的演算法使用更少的記憶體。更甚者，修改過後的演算法在相同記憶體之下執行速度比原本的 F_4 演算法快。而透過將大量的運算拆成許多較小的運算子集再分別一一運算，我們的演算法成功對其記憶體消耗量作出控制。當然這實際上是在時間與空間上作權衡，而將時間去換取空間的結果。由於我們修改過後

的 F_4 演算法需要更多的控制檢查，而在記憶體有限的情況之下，很多資訊無法儲存而必須重複計算。透過計算用時間與空間積值，利用以下幾點我們可以說明新的 F_4 演算法在時間與空間的權衡上是有意義的，且提供了相當的彈性。

1. 我們的修改演算法平均上較原本的 F_4 演算法有更小的時間與空間的乘積。
2. 只要記憶體大於計算所需的最小記憶體需求，我們的修改過的 F_4 演算法可以在任意的記憶體限制環境下計算 Gröbner-basis。
3. 若我們的演算法使用越多的記憶體，則執行的越快。

在論文最後，我們實做了原 F_4 演算法以及修改過後的演算法原型，並得到大量組別的實驗數據。多組不同參數的實驗數據驗證我們修改過後的演算法的確達到上述的研究目標。例如：使用者只要多花兩倍的執行時間，就可以用10%的記憶體執行 F_4 演算法。

關鍵字 密碼學, 代數密碼分析, 解方程組, Gröbner 基底, Faugère的 F_4 演算法, 時間與空間的權衡

Abstract

Solving multivariate systems of polynomial equations is an important problem both as a subroutine in algebraic cryptanalysis and in its own right. Currently, the most efficient solvers are the Gröbner-basis solvers, which include the XL algorithm, as well as Faugère's F_4 and F_5 algorithms.

The F_4 algorithm is an advanced algorithm for computing Gröbner basis. However, the algorithm has exponential space complexity. This poses a serious challenge when we want to use it to solve instances of sizes of practical interest. For example, if we are going to solve a multivariate polynomial system of 40 equations in 40 variables, then most of today's computers will run out of memory before the execution of the algorithm finishes. Furthermore, the original F_4 algorithm does not provide much flexibility in terms of controlling memory usage.

In this thesis, we set out to address this shortcoming by starting with the following questions about F_4 's memory consumption.

1. Can F_4 , or any variant of it, be executed under any memory limitation?
2. If not, at least how much memory is necessary for F_4 to be successfully executed?
3. Can we make the modified F_4 algorithm run faster when given more memory?

Throughout the process of answering these questions, we observe the memory usage in each part of the F_4 algorithm, based on which we propose modifications to the algorithm. Our modified F_4 algorithm uses less memory than the original algorithm. More importantly, our modified F_4 algorithm runs faster than the original algorithm using the same amount of memory. Our modified F_4 algorithm controls its memory consumption by dividing the

work into chunks of smaller working sets and executing them one at a time. This in effect trades time for memory because it involves more computation, some of which might even be carried out repeatedly. We will show that such a trade-off makes sense in terms of time-memory product and is extremely flexible by showing the following.

1. Our modification on average yields smaller time-memory products than the original F_4 algorithm.
2. Our modified F_4 algorithm allows the Gröbner basis be computed using an arbitrary amount of memory as long as it is above the minimum amount of memory required to solve the instance.
3. The more memory our modified F_4 algorithm uses, the faster it runs.

We have implemented a prototype of the proposed modified F_4 algorithm and conducted an extensive set of experiments with it. The experiment results demonstrate that the proposed modification does achieve the three goals listed above over a broad set of parameters and problem sizes. As an example showcase, it is possible to solve certain instances using only 10% of the memory in less than twice as much time than the original F_4 algorithm.

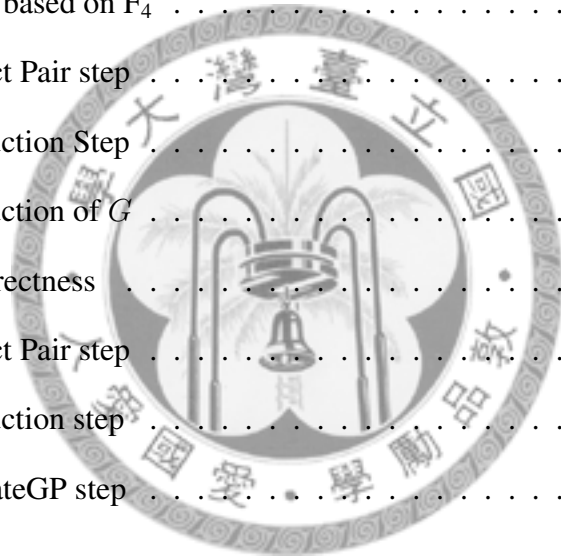
Keywords: *Cryptography, algebraic cryptanalysis, system solver, Gröbner basis, Faugère's F_4 algorithm, time-memory trade-off*

Contents

序言	i
摘要	iii
Abstract	v
Table of Contents	vii
List of Algorithms	ix
List of Tables	x
List of Figures	xi
1 Introduction	1
1.1 Motivation	2
1.2 Challenges	3
1.3 Contributions	5
1.4 Thesis Organization	6
2 Background	8



2.1	Groebner Basis	9
2.1.1	Order	9
2.1.2	Groebner Basis	11
2.2	Buchberger Algorithm	14
2.3	XL Algorithm	18
2.4	Faugère’s algorithm (F_4)	20
3	Modified Algorithm	24
3.1	New scheme based on F_4	25
3.1.1	Select Pair step	26
3.1.2	Reduction Step	28
3.1.3	Reduction of G	30
3.2	Proof of Correctness	32
3.2.1	Select Pair step	32
3.2.2	Reduction step	33
3.2.3	UpdateGP step	34
4	Experiment Results	38
4.1	$m = n + 2$	40
4.2	$m = 1.5n$	42
4.3	$m = 2n$	44
4.4	Analysis	46
5	Conclusion	48
	Bibliography	51

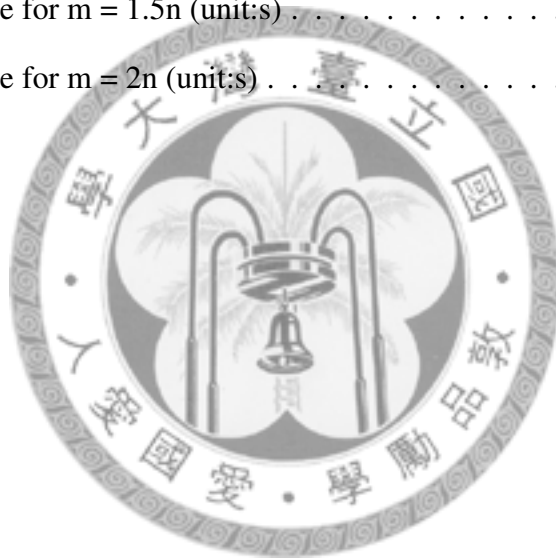


List of Algorithms

2.1	Pseudocode of Buchberger Algorithm	15
2.2	Pseudocode of SelectPair function in Buchberger Algorithm	16
2.3	Pseudocode of UpdateGP function	17
2.4	Pseudocode of XL Algorithm	18
2.5	Pseudocode of F_4 Algorithm	22
2.6	Pseudocode of SelectPairs function in F_4	23
3.1	Pseudocode of modified scheme	36
3.2	Pseudocode of SelectPairs function in modified scheme	37
3.3	Pseudocode of ReduceG function in modified scheme	37

List of Tables

4.1	Running Time for $m = n+2$ (unit:s)	41
4.2	Running Time for $m = 1.5n$ (unit:s)	43
4.3	Running Time for $m = 2n$ (unit:s)	45

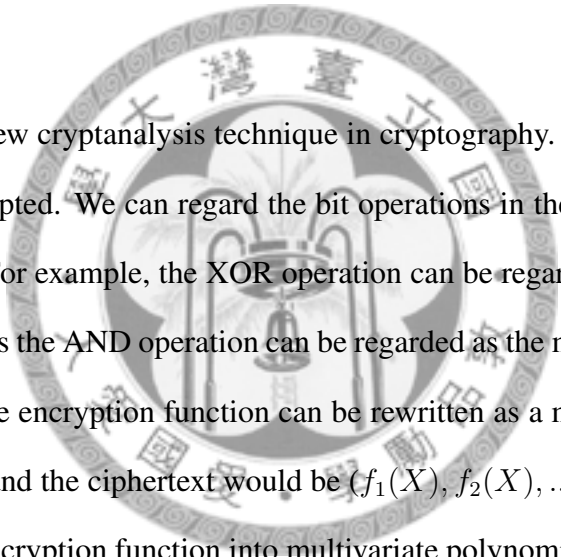


List of Figures

2.1	Simple sketch of Groebner Basis	12
2.2	Transformation between matrix and polynomials	17
4.1	Memory curve for $m = n+2$ (unit:int)	40
4.2	Time curve for $m = n+2$ (unit:s)	41
4.3	Time * Memory curve for $m = n+2$	42
4.4	Time curve for $m = 1.5n$ (unit:s)	43
4.5	Time * Memory curve for $m = 1.5n$	44
4.6	Time curve for $m = 2n$ (unit:s)	45
4.7	Time * Memory curve for $m = 2n$	46
4.8	Memory curve for each case (unit:int)	46
4.9	Time * Memory curve for each case for 1/10 and 1/8 memory	47

Chapter 1

Introduction



Algebraic attack is a new cryptanalysis technique in cryptography. Let X be the plaintext block need to be encrypted. We can regard the bit operations in the encryption process as algebraic operations. For example, the XOR operation can be regarded as the addition operation in GF2, whereas the AND operation can be regarded as the multiplication operation in GF2. As a result, the encryption function can be rewritten as a multivariate polynomial system (f_1, f_2, \dots, f_n) and the ciphertext would be $(f_1(X), f_2(X), \dots, f_n(X))$. Through the transformation from encryption function into multivariate polynomial system, the question of breaking a cryptosystem is equivalent to solving the corresponding multivariate polynomial system.

To solve a multivariate system, currently there are many existing system solvers. However, most of these algorithms have exponential space and time complexities. Solving a large system becomes impractical due to the requirement of large amounts of hardware resource. Not only the computation ability but also the memory requirement would be the bottleneck of these solvers. It is the memory limitation that motivates this research.

In the rest of this chapter, Section 1.1 will show the readers the research motivation of this thesis in detail. Next, Section 1.2 will describe the challenges we might meet in this kind of research, modifying existing algorithms to lower hardware usage. Section 1.3 will show the contributions and their implications of this thesis to the system solvers. Last, Section 1.4 will detail the organization of the rest of this thesis in order for the readers to understand this thesis more easily.

1.1 Motivation

To solve a multivariate cryptosystem, we can reduce the question to solving a multivariate polynomial system by using the idea of algebraic attack. That is, what we need now is a system solver. There are many existing system solvers, for example, XL algorithm and Faugère's F_4 algorithm.

The F_4 algorithm is a new and efficient concept on computing Groebner basis. However, after doing some research on F_4 , we find that the algorithm has some problems. The F_4 algorithm itself does not concern about the hardware resource; that is, the F_4 algorithm not only uses a lot of memory but also does not have any mechanism to run with smaller amount of memory. This makes F_4 fail to work when resource is scarce. What makes things even worse is the exponential space complexity of F_4 , which means the hardware requirement grows very fast. If we are going to solve a multivariate polynomial system with larger n , the number of variables, for example, $n = 40$, most of today's computers will simply run out of memory. This is a serious flaw of the F_4 algorithm to be used in practice.

Most of time, we can trade memory for computing speed. However, when there we run out of memory, maybe we can spend a little more time to save a lot of memory. Thus, we

start to think of some questions about the memory consumption of the F_4 algorithm, such as follows.

1. Can F_4 , or any variant of it, be executed under any memory limitation?
2. If not, at least how much memory is necessary for F_4 to be successfully executed?
3. Can we make the modified F_4 algorithm run faster when given more memory?

These questions motivate the research work described in this thesis. Based on the structure of the F_4 algorithm, we tried to modify it and make it run with smaller amounts of memory. We also tried to make it run with memory as little as possible but at the same time run as fast as possible under the same memory usage. **What we want to do in this thesis is to show the readers a new and better scheme of the F_4 algorithm in which we can trade off time for memory.** Thus, when the memory is not enough, our implementation can still work well with reasonably tolerable efficiency loss.

1.2 Challenges

Here we try to provide a high-level overview of the challenges we face without going into too much detail, which will be available in Section 2.4. The F_4 algorithm indeed has a subroutine called `SelectPair`, which provides a little flexibility on memory consumption. We can select fewer pairs to check to reduce the memory usage. However, it is not a good way to do this. First, due to the the design of the F_4 algorithm, even if the user chooses a few pairs in `SelectPair`, it produces a large matrix in the Reduction step and uses a lot of memory, defeating the benefits brought by selecting fewer pairs. Second, the `SelectPair` subroutine should be designed to choose the “proper” pairs to make the algorithm run faster.

If we select too few pairs here, the running time will be much longer. That is, we can do only a little with the original F_4 design. We have to modify the algorithm to achieve our goals.

In this thesis, what we are going to do is to design a new scheme based on F_4 to control the memory usage exactly without hurting performance. The new scheme should provide a time-memory trade-off mechanism that the users may select their parameters based on how much memory they have and how much time they would like to spend.

Now that we want to make F_4 run with much less memory than it used to use, we need to modify it to use the memory in the most efficient way to achieve our goals. Though it is very hard to save memory without hurting performance at the same time, the following are what we have tried to achieve.

- Minimizing memory usage

This is one of the most important goal of our work. However, to achieve this goal, we need to discard and recompute some auxiliary information when needed. We also need to reduce the sizes of matrices appearing in F_4 , which is the main source of memory usage in the execution of the algorithm. We also need to reduce the size of the basis that serves a bookkeeping purpose and would become the Groebner Basis at the end.

- Flexible in terms of memory usage

The total memory size will be set as a parameter so that the memory used will not exceed this size. In the original F_4 algorithm, it is hard to control the memory usage; in fact, the total amount of used memory is not kept track of at all. Our modified scheme will achieve this goal by precisely controlling the memory used. Thus our

implementation may run under a variety of memory usages based on an input parameter.

- Maximizing the efficiency of pair checking

The F_4 algorithm is based on Buchberger algorithm. Checking many pairs at the same time is the most significant and important advantage of F_4 . If SelectPair subroutine chooses proper pairs, then the efficiency will increase significantly. This is one of the design that makes F_4 much faster than Buchberger algorithm. However, when there is memory limitation, choosing more pairs will result in more memory consumption. As a result, we can only check much less pairs. However, we want to minimize the difference between pairs selection in the modified scheme and in the original F_4 scheme to keep the efficiency of modified scheme as high as possible.

To fulfill our goals above, we need to modify the F_4 algorithm a lot, especially the Reduction step. More detail of the modified scheme will be described in Section 3.1.

1.3 Contributions

What we do in this thesis is to provide a modified scheme based on the F_4 algorithm. Neither the modified scheme nor the original scheme may run under an arbitrary memory limitation. Indeed, there is minimum memory requirement for each instance to store the absolutely necessary information for the computation. Nevertheless, the new scheme may run on platforms having less memory as long as it is greater than this minimum memory requirement. Moreover, the memory consumption in the new scheme is much less than the original F_4 algorithm. For example, for some instances we only use 1/10 memory.

This number may become smaller when n becomes larger. The original F_4 algorithm may not work with this amount of memory, even if we cut the number of pair selection to one. Under the same memory usage, the modified F_4 algorithm is faster than the original one even if this pair selection strategy works.

When the memory budget is tight, a lot of unnecessary information will not be stored, and a lot of computation needs to be redone. These naturally make our modified scheme run slower than the original scheme. However, the modified scheme has a better time-memory product, which means that we can save a lot of memory at the price of having only a little bit longer running time. Users now may make better trade-off between time and memory using our scheme. For example, people may save 10 times memory at the expense of 2 times running time.

We repeatedly emphasize the importance of memory in this thesis. It is not only because memory is usually the more expensive resource but also because we can use other computing platforms that have much stronger computational capabilities but relatively little memory like the FPGA. Moreover, our modified scheme may allow an implementation on modern parallel computers, where it is relatively easy to scale computational capacity but not memory capacity.

1.4 Thesis Organization

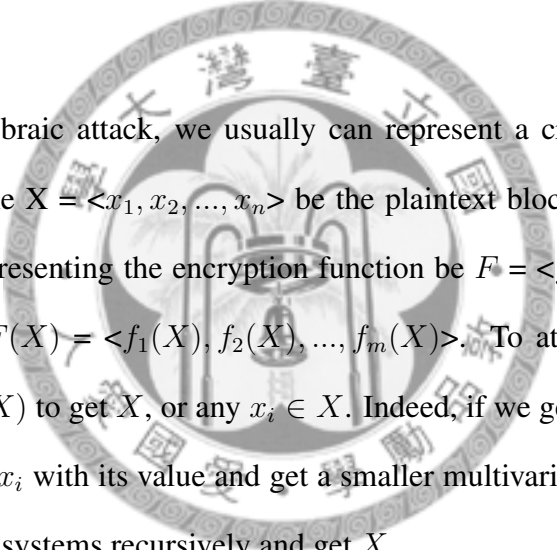
The rest of this thesis is organized as follows. In Chapter 2, we go through the necessary background information and introduce some existing algorithms for computing Groebner Basis. In Chapter 3, we show how we modify Faugère's F_4 algorithm to allow a trade-off between time and memory. We demonstrate how the modified scheme performs under a

variety of inputs and compare the performance with that of the original F_4 algorithm in Chapter 4. Last, we conclude this thesis and point out some future work in Chapter 5.



Chapter 2

Background



Using the idea of algebraic attack, we usually can represent a cryptosystem with a set of polynomials. Let the $X = \langle x_1, x_2, \dots, x_n \rangle$ be the plaintext block, and the multivariate polynomial system representing the encryption function be $F = \langle f_1, f_2, \dots, f_m \rangle$, then the cyphertext would be $F(X) = \langle f_1(X), f_2(X), \dots, f_m(X) \rangle$. To attack a cryptosystem is equivalent to solve $F(X)$ to get X , or any $x_i \in X$. Indeed, if we get the value of some x_i , then we can substitute x_i with its value and get a smaller multivariate polynomial system. Thus, we can solve the systems recursively and get X .

The Faugère's algorithm we are going to work on in this thesis is one of the famous algorithms in solving a multivariate polynomials system. Basically, the Faugère's algorithm is an algorithm calculating Groebner Basis of an Ideal. In the multivariate polynomial system F , consider the ideal I spanned by the polynomials F and its Groebner Basis G of I . Due to the property of Groebner Basis, there is usually a univariate polynomial in G , which is easier to solve than multivariate polynomial equations. Thus, we can solve $F(X)$ and get each x_i by calculating its Groebner Basis repeatedly.

To understand what we are doing in this thesis, we should have an image of the background knowledge. Thus, we will show the reader basic knowledge and existing system solvers in this chapter. First, Section 2.1 is going to give the definitions, theorems and other relevant knowledge of Groebner Basis. The next three sections are going to show the readers three different algorithms calculating Groebner Basis. Specifically, Section 2.2 shows the Buchberger algorithm, which is given by Buchberger in 1976. Section 2.3 shows the XL algorithm, given by A. Shamir, J. Patarin, N. Courtois, and A. Klimov in 2000 and Section 2.4 shows the Faugère's algorithm (F_4), which is given by Faugère in 1999. The last algorithm (F_4) is what we want to work on in this thesis. We will give more detail on it.

2.1 Groebner Basis

First we define some symbols. Let $X = \{x_1, x_2, \dots, x_n\}$, $R = \text{GF}_{p^k}[X]$ be a multivariate polynomial ring, and $\{f_1, f_2, \dots, f_m\}$ be the set of polynomials in R . Ideal $I \subset R = \langle f_1, f_2, \dots, f_m \rangle$ is an ideal finitely generated by $\{f_1, f_2, \dots, f_m\}$.

2.1.1 Order

It is easy to give an order between monomials in a univariate polynomial ring, for example, $x^7 > x^5 > 1$. However, the problem is how to determine the order in a multivariate polynomial ring, for example, $xy > z$ or $xy < z$ for $xy, z \in \text{GF}_2[x, y, z]$.

To give an order in a multivariate polynomial ring, we need to define an order among x_1 to x_n first. Without loss of generality, we define $x_1 > x_2 > \dots > x_n$. After having the order among variables, now we can construct the order among monomials.

An order is a relation satisfying:

- $x = x$
- $x > y$ then $y < x$
- $x > y$ and $y > z$ then $x > z$
- $x > y$ then $c^*x > c^*y$, $c \neq 0$

We now can define order among monomials in R . There are several orders in a multivariate polynomial ring. We show some common orders here. Note that we denote the head term of a polynomial f as $\text{HT}(f)$.

- **Lexical order (lex)**

This order is what we usually do with words in dictionary. For a monomial m_1, m_2 , if the degree of x_1 in m_1 is more than m_2 , then $m_1 > m_2$. If the degree of x_1 in m_1 equal to m_2 , then we compare the degree of x_2 , and so on.

For example, $x_1^2 > x_1x_3^7 > x_2^3x_3^5 > x_2^2x_3^6 > x_2^2x_3^4$

- **Graded Lexical order (grlex)**

This order is a little different with lexical order. First we compare the total degree of m_1 and m_2 . If $\deg(m_1) > \deg(m_2)$, $m_1 > m_2$. If $\deg(m_1) = \deg(m_2)$ then we check the order of m_1 and m_2 using lexical order.

For example, $x_1x_3^7 > x_2^3x_3^5 > x_2^2x_3^6 > x_2^2x_3^4 > x_1^2$

- **Graded Reverse Lexical order (grvlex)**

Like graded lexical order, we compare the total degree of m_1 and m_2 . If $\deg(m_1) > \deg(m_2)$, $m_1 > m_2$. If $\deg(m_1) = \deg(m_2)$ then we check the degree of x_n in m_1 and

m_2 . The higher the degree of x_n is, the less it is. If the degree of x_n in m_1 equal to m_2 , then we compare the degree of x_{n-1} , and so on.

For example, $x_2^3x_3^5 > x_2^2x_3^6 > x_1x_3^7 > x_2^2x_3^4 > x_1^2$

Notice that we do not care about the coefficient of monomials, that is, two terms are the same if they have the same monomials, no matter what the coefficients are. For example, $3x_1^2$ have the same order with x_1^2 .

After defining order in monomial, then we can easily tell the order among polynomials. Let f_1 and f_2 be two polynomials in R . First we make the monomials in f_1 and f_2 in order and compare the head term of these two polynomials. If $HT(f_1) > HT(f_2)$, then $f_1 > f_2$. If $HT(f_1)$ equal to $HT(f_2)$, then we check the next monomials. For example, $x_2^3x_3^5 + 1 > x_2^2x_3^6 + x_3^2 + 1 > x_2^2x_3^6 + x_3 + 1$ in graded reverse lex order.

2.1.2 Groebner Basis

Definition 1 *Groebner Basis*

Let G be a basis of $I = \langle f_1, f_2, \dots, f_m \rangle$, then G is a Groebner Basis if and only if $\forall f \in I, \exists g \in G$, such that $HT(g) \mid HT(f)$

For example, $\langle 8, 6 \rangle = \langle 2 \rangle$ in Z . $\{8, 6\}$ is not a Groebner basis, while $\{2\}$ is. Another example in $\text{GF}_2[x, y]$, $\langle x^2 + y, x + y \rangle$ and $\langle x + y, y^2 + y \rangle$ represent the same ideal, however $\{x^2 + y, x + y\}$ is not a Groebner basis while $\{x + y, y^2 + y\}$ is.

In Figure 2.1, every circle represents a $HT(f)$, $f \in R$. The arrow from circle $HT(f_i)$ to circle $HT(f_j)$ represents $HT(f_i) \mid HT(f_j)$. Let $HT(f_i) <_{HT} HT(f_j)$ iff $HT(f_i) \mid HT(f_j)$. Figure 2.1 shows the partial order $<_{HT}$ defined on the set of head terms in ideal I . We

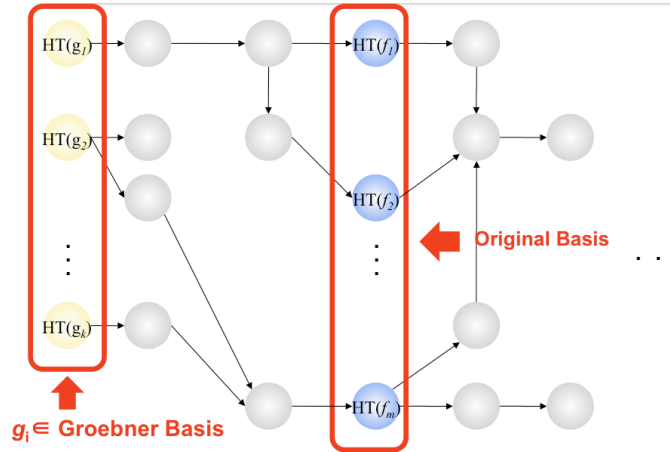


Figure 2.1: Simple sketch of Groebner Basis

can say that, by definition of Groebner Basis, a Groebner Basis G must contains all those minimal polynomials g_i with $HT(g_i)$ being the leftmost node.

Next, to calculate Groebner Basis, we have to calculate SPolynomials first.

Definition 2 *SPolynomial*

Let $f_i, f_j \in I$, $c_i = \frac{\text{lcm}(HT(f_i, f_j))}{HT(f_i)}$, $c_j = \frac{\text{lcm}(HT(f_i, f_j))}{HT(f_j)}$, then $c_i f_i - c_j f_j$ is the SPolynomial of (f_i, f_j) , denoted as $SPoly(f_i, f_j)$. The degree of $SPoly(f_i, f_j)$ is the degree of $\text{lcm}(f_i, f_j)$, denoted as $deg(SPoly(f_i, f_j))$.

Definition 3 *Pair*

Let f_i, f_j, c_i, c_j be the same defintion in the Definition 2, then we said that $\langle \text{lcm}(f_i, f_j), c_i, f_i, c_j, f_j \rangle$ is a pair which gives the information of $SPoly(f_i, f_j)$. Let p be a pair of (f_i, f_j) , the contents of p is denoted in order as $p.lcm, p.f_i \text{mult}, p.f_i, p.f_j \text{mult}, p.f_j$.

Definition 4 *pre-SPolynomial*

Let f_i, f_j, c_i, c_j be the same defintion in the Definition 2, $c_i f_i, c_j f_j$ are the pre-SPolynomials of (f_i, f_j) , denoted as $\text{pre-SPoly}_i(f_i, f_j)$ and $\text{pre-SPoly}_j(f_i, f_j)$ respectively.

By definition of SPolynomial, $SPoly(f_i, f_j)$ eliminates the head term of pre- $SPoly_i$ and pre- $SPoly_j$, which is divisible by head term of f_i and f_j respectively. Thus, we will find out new head terms that may not be divided by the head terms in G .

For example, let $I = \langle x^2y + 1, xy^2 + 1 \rangle$, $SPoly(x^2y + 1, xy^2 + 1) = -x + y$, it is a new head term not divided by x^2y and xy^2 .

The next question is, how we determine whether a basis G is Groebner Basis or not. Buchberger then gave a theorem as followed :

Theorem 1 *Buchberger Theorem*

Let G be a basis of I , then G is a Groebner Basis of I if and only if $\forall f_i, f_j \in I$, $SPoly(f_i, f_j)$ can be reduced to zero by G .

This theorem states that if all the SPolynomials of G can be reduced to zero by G , then G is Groebner Basis, and vice versa. One direction of the proof is straightforward. If there exists a $SPoly(f_i, f_j)$, denoted as $spoly$, that cannot be reduced to zero by G , that is, $spoly$ is reduced to p by G , $p \neq 0$, then $\forall g \in G$, $HT(g) \nmid HT(p)$. This conflicts with the definition of Groebner Basis. In other hand, if G satisfied the statement, then every arithmetic combination of g_i in G will have a head term that is a multiple of g_i (Standard Representation). Thus, it is a Groebner Basis. The detail of proof can be found in [1, 1993].

Now we have a systematic method to examine whether a basis G is a Groebner Basis or not. Buchberger also stated some special cases for $SPoly(f_i, f_j)$ that do not need to be checked.

Criteria 1 *Buchberger First Criteria*

Let $f_i, f_j \in I$, if $\gcd(f_i, f_j) = 1$, then $SPoly(f_i, f_j)$ can be reduced to zero by $\{f_i, f_j\}$.

Criteria 2 Buchberger Second Criteria

Let $f_i, f_j, f_k \in I$, and $\text{lcm}(f_i, f_j) \mid \text{lcm}(f_j, f_k)$, $\text{lcm}(f_i, f_k) \mid \text{lcm}(f_j, f_k)$, then if both $\text{SPoly}(f_i, f_j)$ and $\text{SPoly}(f_i, f_k)$ can be reduced to zero by G , so is $\text{SPoly}(f_j, f_k)$.

The Buchberger First Criteria stated that if we have f_i, f_j with $\text{gcd}(f_i, f_j) = 1$, then we do not need to check $\text{SPoly}(f_i, f_j)$, since it must be reduced to zero by $\{f_i, f_j\} \subseteq G$, thus must be reduced by G . The Buchberger Second Criteria stated that if we have f_i, f_j, f_k satisfying the condition specified, then we only need to check $\text{SPoly}(f_i, f_j)$ and $\text{SPoly}(f_i, f_k)$, since if both the two SPolynomial can be reduced to zero by G , then $\text{SPoly}(f_j, f_k)$ can be reduced to zero by G , too. The detail of proof also can be found in [1, 1993].

The following three section shows three different algorithms finding out Groebner Basis of $\langle F \rangle$, where F is a set of polynomials that we want to solve.

For convenience, when we said about divisibility of polynomials in this thesis, it means the divisibility of their head terms. For example, when we said $f \mid g$, it means $HT(f) \mid HT(g)$.

2.2 Buchberger Algorithm

Buchberger algorithm is based on Buchberger Theorem 1 stated by Bruno Buchberger in 1976. It is the first algorithm to compute a Groebner Basis of an Ideal systematically. Buchberger Theorem really told us how to examine whether a Basis is a Groebner Basis or not. According to the theorem, what the algorithm does is very simple:

1. Set the input F to be a Groebner Basis G
2. Check if G satisfies the condition stated in Buchberger Theorem
3. If G does not satisfy, modify the elements in G and go to step 1

We now introduce more detail as follows.

Algorithm 2.1 Pseudocode of Buchberger Algorithm

Input: F

REM Initialization

2 $F \leftarrow \text{ReducedRowEchelon}(F)$

3 $G, P \leftarrow \text{UpdateGP}(G, P, F)$

REM Main program - check pairs and let G satisfied the condition

5 **while** $P \neq \{\}$ **do**

6 $spoly, P \leftarrow \text{SelectPair}(P)$

7 $r \leftarrow \text{MultivariateDivision}(spoly, G)$

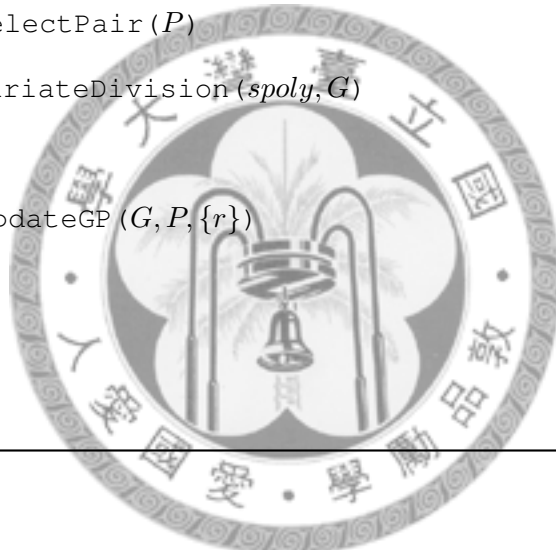
8 **if** $r \neq 0$ **then**

9 $G, P \leftarrow \text{UpdateGP}(G, P, \{r\})$

10 **end**

11 **end**

Output: G



We can see in Algorithm 2.1 the whole flow of Buchberger algorithm. The input of the algorithm is F , the polynomial system we would like to solve, while the output of the algorithm is G , the Groebner Basis of F , which spans the same ideal and contains at least one univariate polynomial.

The algorithm takes the input F and set it as G first (line 3). At the same time, it also maintains a pool of Pairs P to store those pairs that haven't been checked. After the initialization, it starts to examine if G is really a Groebner Basis by checking all the pairs generated by G . The pool P stores all these pairs; those which have been examined are

Algorithm 2.2 Pseudocode of SelectPair function in Buchberger Algorithm

Input: P

- 1 $pair \leftarrow \text{SelectStrategy}(P)$
- 2 $P \leftarrow P \setminus \{pair\}$
- 3 $spoly \leftarrow pair.imult * pair.f_i - pair.jmult * pair.f_j$

Output: $spoly, P$

removed from P .

In the while loop (line 5), we pick and check pairs from P one by one. First, SelectPair function picks a pair from P with a select strategy, as Algorithm 2.2 shows. The select strategy is left open, so the user can select pairs in any way she likes, as long as all the pairs are selected and checked eventually. Nevertheless, it is suggested that the user select the pair with the least degree of $pair.lcm$. After SelectPair function, we get a SPolynomial $spoly$. We check if it can be reduced to zero by G at line 7. The MultivariateDivision function reduces $spoly$ by G and gets r either equal to zero or each monomial mn in r with $g_i \nmid mn, g_i \in G$. By the Buchberger Theorem, if there is one SPolynomial that cannot be reduced to zero by G , then G is not a Groebner Basis. Thus, if $r \neq 0$, then it means we need to modify G to make it satisfy the condition of Buchberger Theorem. The simplest and valid way to do this is to add r into G (line 9). This operation would not change the ideal spanned by G since r is a linear combination of $g_i \in G$. By doing this, $spoly$ is reduced to zero by G since its remainder r is in G now. By keeping checking the pairs and adding new remainders if needed, finally we will arrive at the Groebner Basis.

The UpdateGP function appears in line 3 and line 9 is indeed taking a new set F , adding its elements into Basis G , and putting the new pairs generated into P , as shown

Algorithm 2.3 Pseudocode of UpdateGP function

Input: G, P, F

```
1 forall the  $f \in F$  do  
2    $P \leftarrow P \cup \{\text{getPair}(f, g_i) \mid g_i \in G\}$   
3    $P \leftarrow \text{BuchbergerCriteria}(P)$   
4    $G \leftarrow G \cup \{f\} \setminus \{g_i \mid g_i \in G, f|g_i\}$   
5 end
```

Output: G, P

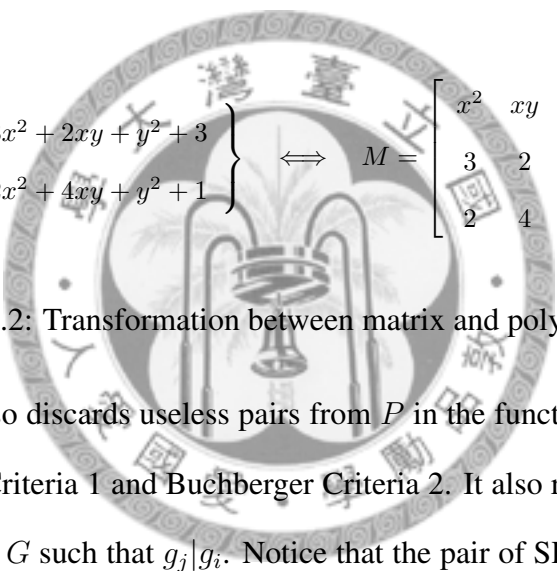

$$F = \left\{ \begin{array}{l} 3x^2 + 2xy + y^2 + 3 \\ 2x^2 + 4xy + y^2 + 1 \end{array} \right\} \iff M = \begin{bmatrix} x^2 & xy & y^2 & 1 \\ 3 & 2 & 1 & 3 \\ 2 & 4 & 1 & 1 \end{bmatrix}$$

Figure 2.2: Transformation between matrix and polynomials

in Algorithm 2.3. It also discards useless pairs from P in the function `BuchbergerCriteria` based on Buchberger Criteria 1 and Buchberger Criteria 2. It also removes useless $g_i \in G$ if there is another $g_j \in G$ such that $g_j|g_i$. Notice that the pair of $\text{SPoly}(g_i, g_j)$ is in P now according to the design of `UpdateGP`. Consider a new remainder r to be added into G ; then g_i, g_j, r satisfy the condition of Buchberger Criteria 2, which means that we just have to check $\text{SPoly}(g_i, g_j)$ and $\text{SPoly}(g_j, r)$, and then $\text{SPoly}(g_i, r)$ would also satisfy. Indeed, that means G always keeps the leftmost Head Term of every line shown in Figure 2.1.

What the `ReducedRowEchelon` function in line 2 does is that it transforms the polynomials F into a matrix M , computes the reduced row echelon form of M , transforms it to polynomials again, and puts them into F . The transformation between matrix and

polynomials is as figure 2.2 shows.

The Buchberger algorithm will terminate deterministically since the basis must be finitely generated based on Hilbert's basis theorem and the fact that we keep finding smaller polynomials in the ideal spanned by F . However, it is too slow to be practical in solving a larger system. If there are m polynomials in F , then we have to check almost $\frac{m*(m-1)}{2}$ pairs, and more as new remainders are added into G . Doing this checking one by one is not very efficient. There are a lot of duplicated work that may be done together to save time. That's how F_4 improved the Buchberger algorithm, which we will show later in Section 2.4. More detail of the Buchberger algorithm can be found in [2, 1976].

2.3 XL Algorithm

Algorithm 2.4 Pseudocode of XL Algorithm

Input: F

REM Initialization

2 $D \leftarrow \text{ChoseD}(m, n, d)$

REM Main program — extend the polynomials in F to deg D and do Reduced Gaussian

Elimination

4 $M \leftarrow \text{Monomials with deg} \leq D - d$

5 $F' \leftarrow \{m*f \mid m \in M, f \in F\}$

6 $G \leftarrow \text{ReducedRowEchelon}(F')$

Output: G

XL algorithm was given by A. Shamir, J. Patarin, N. Courtois, and A. Klimov in 2000.

However, the idea had been proposed earlier but did not considered to be practical due to the large memory requirement. It is constructed in a straightforward manner that is easy to understand. Consider that when we do the calculation of the $\text{SPoly}(f_i, f_j)$, we extend both f_i and f_j first and then subtract them. When we do the division of $g|f$, we also extend g to the head term of f and subtract them. The subtraction of polynomials is indeed the subtraction of rows in a matrix after transformation. Thus, what we want to do is to put enough polynomials into the matrix, then perform reduced Gaussian Elimination once and we will get a Groebner Basis of F .

The set of needed polynomials can be computed in practice, e.g., simply taking all those polynomials appear during execution of Buchberger algorithm. Let the set of polynomials be denoted as $Poly$. Since $Poly$ in Buchberger algorithm is finite, the maximum degree D in $Poly$ exists. Let d be the degree of F and F' be the polynomials set extended from F containing all the polynomials in $\langle F \rangle$ with degree from d to D . Then $Poly \subseteq F'$ obviously. The idea of XL is that when we extend F to a large enough degree D , which might not be equal to the maximum degree appears in Buchberger algorithm, then we will get all of the information needed to calculate the Groebner Basis.

We can see in Algorithm 2.4 that how the XL works. First, we need to initialize the degree D . We choose D based on parameters m, n, d , where m is the number of polynomials in F , n is the number of variables, and d is the degree of F . See [3, 2004] for a good way of choosing degree D . Then what XL does is simple — extending F to F' (line 5), and getting Groebner Basis G from the reduced row echelon form of F' (line 6).

XL algorithm is designed to work in Lex order first. However, now there are many different algorithms based on XL algorithm, for example, FXL and XL2. More detail about XL algorithm can be found in [4, 2000].

2.4 Faugère's algorithm (F_4)

F_4 algorithm was given by Faugère in 1999. It was based on the idea of Buchberger algorithm with some improvement. The improvement makes it amazingly fast. The changes are summarized as follows.

- Check instead of a pair but a set of pairs at one time
- Do the reduction of SPolynomial in matrix form

We can see that how F_4 works in Algorithm 2.5. The initialization of F_4 is the same as Buchberger algorithm. Both Buchberger algorithm and F_4 use the same ReduceRowEchelon and UpdateGP function. However, the Select Pair step is a little different. Unlike Buchberger, SelectPairs here selects a set of pairs to check and return a set of pre-SPolynomials of those pairs stored in $Poly$, as shown in Algorithm 2.6. The SelectStrategy is left open, so the user can select pairs any way she likes. It is recommended that those pairs with lcm of the least degree be selected first. The improvement here is that the F_4 algorithm increases the throughput of the pairs checking. The reason to select a set of pairs is not only to increase the throughput, but also to eliminate some duplicated work. For example, if there is a monomial m in both f_i, f_j that can be divided by g , in Buchberger algorithm, g must to extend to m twice to do the reduction, whereas in F_4 algorithm, the extension only needs to be done once since we check f_i, f_j together.

The Reduction step here is a little bit complicated. What from line 9 to line 15 in Algorithm 2.5 do is to make sure that every monomial $m \in Poly$, if there exists a g in G such that $g|m$, then g will be extended to m and contained in $Poly$. $Monomials(Poly)$ is the set of monomials in $Poly$, and $Headterms(Poly)$ is the set of head terms of $Poly$. The

reason to do this step is to ensure that for any $r \in R$ (line 17), r cannot be reduced by G anymore. We can then reduce the $Poly$ to $Poly'$ by Gaussian Elimination after discarding those p in $Poly'$ that can be divided by G , after which we get the new remainders R and finish the Reduction step. The UpdateGP step is the same as in Buchberger algorithm.

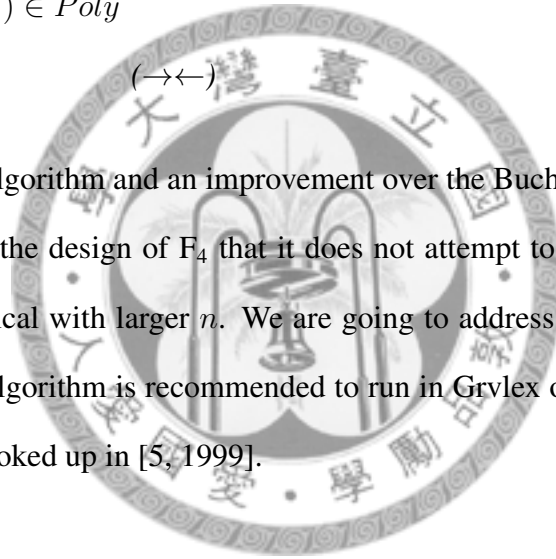
Proof 1 *Reduction Step of F_4*

Suppose $\exists r \in R, \exists g \in G, g \mid r$, then obviously, $HT(r) \in \text{monomials of } Poly$

\implies By the designed of Reduction step, $g^* \frac{m}{HT(g)} \in Poly$

$\implies m = HT(g^* \frac{m}{HT(g)}) \in Poly$

$\implies m \notin R$



F_4 is really a fast algorithm and an improvement over the Buchberger algorithm. However we can see from the design of F_4 that it does not attempt to control memory usage. This makes it impractical with larger n . We are going to address this shortcoming in the next chapter. The F_4 algorithm is recommended to run in Grvlex order. More detail about F_4 algorithm can be looked up in [5, 1999].

Algorithm 2.5 Pseudocode of F_4 Algorithm

Input: F

REM Initialization

2 $F \leftarrow \text{ReducedRowEchelon}(F)$

3 $G, P \leftarrow \text{UpdateGP}(G, P, F)$

REM Main program - check pairs and let G satisfied the condition

5 **while** $P \neq \{\}$ **do**

REM step : Select Pair

7 $Poly, P \leftarrow \text{SelectPairs}(P)$

REM step : Reduction

9 $M \leftarrow \text{Monomials}(Poly)$

10 $Done \leftarrow \text{Headterms}(Poly)$

11 **while** $\exists m \in M \setminus Done, \exists g \in G, g|m$ **do**

12 $Done \leftarrow Done \cup \{m\}$

13 $Poly \leftarrow Poly \cup \{g * \frac{m}{HT(g)}\}$

14 $M \leftarrow \text{Monomials}(Poly)$

15 **end**

16 $Poly' \leftarrow \text{ReducedRowEchelon}(Poly)$

17 $R \leftarrow \{p \mid p \in Poly', HT(p) \notin HT(Poly)\}$

REM step : UpdateGP

19 $G, P \leftarrow \text{UpdateGP}(G, P, R)$

20 **end**

Output: G



Algorithm 2.6 Pseudocode of SelectPairs function in F_4

Input: P

1 $Pair \leftarrow \text{SelectStrategy}(P)$

2 $P \leftarrow P \setminus Pair$

3 $Poly \leftarrow \{pair.f_i \text{mult} * pair.f_i, pair.f_j \text{mult} * pair.f_j \mid pair \in Pair\}$

Output: $Poly, P$



Chapter 3

Modified Algorithm

In the last chapter we have introduced the F_4 algorithm. It is a fast algorithm to compute Groebner Basis. However, F_4 algorithm is not designed to have any mechanisms to control memory consumption. Due to the exponential space complexity of F_4 , when the multivariate polynomial system is large, for example, $n = 40$, most computers will suffer the problem that there is not enough memory to support the execution of the algorithm.

In Section 2.4, we see that the `SelectPair` function can provide some flexibility in F_4 . If we select a smaller number of pairs in `SelectPair`, then the memory needed in one iteration would be less. However, this is not a good strategy in reducing memory consumption since the design of Reduction step in F_4 algorithm may produce a large matrix. Moreover, if we select too few pairs here, the running time of the program would be much longer, making the algorithm impractical.

What we want to achieve here in this thesis is to design a new scheme based on the structure of F_4 algorithm that controls less memory than F_4 without unnecessarily sacrificing performance. We would like to run on the platform with minimum memory require-

ment (the memory needed to store G and P) using the modified scheme. Moreover, under the memory limitation less than the maximum memory requirement of F_4 , the modified scheme should work better than the original F_4 algorithm. In summary, we would like a new scheme that provides a better time-memory trade-off for the user.

In Section 3.1, we will show readers the three main parts we have modified based on the F_4 algorithm to achieve our goals:

1. Modification of SelectPair function in Subsection 3.1.1
2. Modification of Reduction step in Subsection 3.1.2
3. Adding new ReduceG function in Subsection 3.1.3

Section 3.2 is going to show the users the correctness of each modification of F_4 algorithm. Readers will see that even we modify some structures of F_4 , the modified algorithm still computes the correct Groebner basis as F_4 does.

3.1 New scheme based on F_4

To show our modifications to the F_4 algorithm, we first show the entire algorithm of new scheme in Algorithm 3.1 and then introduce each part subsequently. As Algorithm 3.1 shows, we modify the SelectPair and Reduction parts of the original F_4 algorithm and add one more function ReduceG in the UpdateGP step.

In the new scheme, we provide a new parameter *size*, which controls how much memory can be used. Since the number of pairs selected will affect the matrix size, we should make some changes of the select pair strategy, as shown in Algorithm 3.2, to control the

number of pairs selected in one loop iteration by taking a parameter mat_size based on which the function can select pairs that fit the size.

We also need to modify the Reduction step. According to the design of F_4 shown in Algorithm 2.5, we see that the matrix size would become larger and larger as we check the divisibility of each monomial m by adding extra polynomials into $Poly$. This introduces two problems. First, the matrix size will become much larger than the size before checking monomials, and second, it is hard to limit the matrix size due to the uncertainty. We try to overcome these problems by splitting the work into several parts and working with smaller matrices. More detail will be introduced in Subsection 3.1.2.

We not only need to consider the matrix size, but also the size of G as well as the size of P . Indeed, the least amount of memory required is the memory to store G and P , which is the essential piece of information that needs to be stored. Though it is hard to do anything with size of P , we can always reduce the size of G by reducing the non-heading terms. That is, we can make the length of g in G shorter without reducing its head terms. We can reduce the size of G in many ways, and here we are going to introduce a simple algorithm in Subsection 3.1.3.

We are going to introduce the detail of these modifications in the following subsections.

3.1.1 Select Pair step

Select Pair is an open strategy in the F_4 algorithm. This is the most direct factor to decide how large the matrix size is. Since there is memory limitation in our algorithm, we cannot use the same select pair strategy in F_4 .

Let $size$ be the total memory usable to the program, G_size be the memory used to

store G , and P_size be the memory used to store P . Then the remained memory can be used for the step Reduction step is $mat_size = size - G_size - P_size$. We have to decide how to use the remaining free memory wisely.

After the analysis of the design of F_4 , we decide to fill up the memory with pre-SPolynomials. That is, we select as many pairs as we can to maximize the throughput to be as close to F_4 as possible. The reason to do this is that the throughput is important to efficiency. We can eliminate more duplicated operations if we check more similar pairs together. Another reason is that other polynomials added to $Poly$ is prepared for the reduction of SPolynomials. However it might not be used since its corresponding monomial m might not be a head term of any remainders. It is not an absolutely necessary information during this stage, so we may not want to pre-fetch this information under the condition of limited memory.

The new pair selection strategy in line 2 of Algorithm 3.2 is simple. First, we pick the pairs one by one from P with the least degree for their least common multiples, as one would do in the original F_4 . However, when the corresponding matrix of $Poly$ exceeds mat_size , we break the loop and return $Poly$. After this step, we have the pre-SPolynomials in $Poly$ consisting of those selected pairs. Indeed, $Poly$ here is a subset of the $Poly$ in the F_4 .

In most cases when the memory is limited, the number of selected pairs is much smaller than the recommended number. Though this limits the reduction throughput, it would still save more duplicated calculations than if we selected more pairs. Since each loop of the algorithm will produce new basis elements, and the new basis will have some pairs removed which we do not need to check any more according to the Buchberger Criteria. Thus, the running time is not slowed as much as it would given the reduced number of pairs selected.

Notice that we fill up the remaining memory here not with other polynomials that might be added into $Poly$ during the divisibility check of monomials, as shown in line 9 to line 15 in Algorithm 2.5. This means that we need to modify the Reduction step to achieve the functionality of what has been omitted here, which will be shown in Algorithm 3.1.2. We select the most pairs that can be selected under the same memory limitation, and this is one of the important factors to determine how fast our modified algorithm can run.

3.1.2 Reduction Step

The Reduction Step from line 10 to line 15 in Algorithm 3.1 is the main modification. Remember that, in the Select Pair step, We fill up the matrix with all pre-SPolynomials. However, we still need to check the non-heading terms, or in other words, the heading terms after reduction of pre-SPolynomials, according to the Buchberger Theorem.

What we are going to do here looks much like Buchberger Algorithm. Instead of reducing the SPolynomials in batch as F_4 does, we check the divisibility of new polynomials after each matrix elimination. The reason is that we could save the memory to store the pre-computed information which might not be useful in the reduction. Besides, this design is more helpful to controlling the matrix size. If we want to control the matrix size, we have to make the maximum matrix size be predictable. Our target is clear: we only care about the head terms instead of all the monomials in $Poly$. This makes the number of polynomials that can be added into $Poly$ at most the number of polynomials in $Poly$ since there are at most $\#Poly$ head terms that need to be reduced.

First, we reduce $Poly$ with Gaussian Elimination at line 11. After this reduction, we will get $Poly'$, the remaining polynomials of $Poly$. Then we select those p in $Poly'$ such

that the head terms of which do not appear in $Poly$ and denote them as $Poly''$, as in line 12. These polynomials are new polynomials produced by $Poly$, not the ones we add into $Poly$ before. Then we need to reduce $Poly''$ by G again to ensure that the remaining polynomials finally output cannot be divided by G any more. We check every p in $Poly''$ in line 13. If p can be divided by some g in G , then we need to reduce this p again. Thus we put both p and extended g into $Poly$. This operation means that we are going to divide p by g . For those p that cannot be divided by G any more, we are going to output it as new remainders that will need to be added into G to make G satisfy the condition of Buchberger Theorem. Before finishing the Reduction step, we store these p in R , as shown in line 14. The loop will break when $Poly$ is empty, which means that we have checked all the polynomials produced during the reduction of SPolynomials, and there are no more polynomials that can be divided by G in this turn. Notice that, the number of polynomials in $Poly$ will be always even.

We can see that every matrix appearing in each loop iteration here is indeed a submatrix of the large matrix in the Reduction step in the original F_4 . The new scheme splits the whole matrix into several smaller, limited matrices. Though we save some memory here, there is an overhead of going between polynomials and matrices. Of course there are also duplicated operations in polynomial multiplication and matrix reduction since the smaller matrices overlap one another. All these factors might make the efficiency of Reduction step of the modified scheme worse than F_4 .

On the other hand, we can save some operations. In each loop iteration of checking $Poly''$, we use the new remainders R , as shown in line 13, which is going to be added into G immediately. If there is a divisibility relationship between members in R , the modified scheme will do the reduction in the same loop iteration of Reduction step. In the original

F_4 algorithm, this divisibility of new remainders cannot be discovered, and the new pairs would be produced again in the next iteration, resulting some unnecessary computation and memory usage.

After doing the Reduction step, we will finally reduce all the Spolynomials of selected pairs by G and reach at R , the set of new remainders. We are going to add R into G and update G and P likes F_4 does in the UpdateGP step.

3.1.3 Reduction of G

The size of G is a new problem that we need to solve. In F_4 , since the matrix in reduction is large, doing Gaussian Elimination not only reduces the head terms of SPolynomials by G but also eliminates many non-heading monomials. The larger the matrix is, the more monomials can be eliminated. Thus, p in G is shorter in length.

In our modified scheme, though spanning the same ideal with G , the length of p in G is longer than it is in the original F_4 algorithm. There are two reasons for it. First, we select limited pairs, thus the monomials contained in these SPolynomials are limited, so the monomials that can be eliminated are also limited. Furthermore, in the modified scheme, we only care about the head terms, so only the monomials appearing as head terms are eliminated. All these make G larger.

The procedure reducing non-heading terms in G do not affect the correctness of algorithm since it does not change the solution space of $\langle F \rangle$ nor the head terms in G . It is recommended for efficiency purposes.

The size of G is an important factor for memory consumption. First, since we have less memory than in the original F_4 algorithm, we can not afford to waste a lot of memory on

G . Besides, the larger G is, the less free memory the matrix reduction can use. Moreover, under the same matrix size, the longer g in G is, the more columns we have when transforming into matrix form. Thus, the number of row is decreased, which means that less SPolynomials can be put into the matrix and the less pairs can be checked in one iteration. Due to these reasons, we have to reduce the size of G to a tolerable size.

How to reduce G is an open strategy here. We can reduce the non-heading monomial of G at any stage by any algorithm in the new scheme since it does not change anything but the size of G .

Here we show an example of ReduceG function. We will do this step after UpdateGP function in line 18 of Algorithm 3.1. We do the reduction of G here so that every time G changes, we reduce it immediately. The sample algorithm of ReduceG is shown in Algorithm 3.3.

Let us see the sample code in Algorithm 3.3. We reduce g in G one by one. Let max_length be the maximum length of g in G . If the length of g is less than $\frac{1}{2}max_length$, then we ignore g and go to the next one. Otherwise, we reduce g by G , as shown in line 2. Of course, g cannot be divided by G anymore according to the algorithm, so the only thing we can do now is to reduce non-heading term of g here. The next thing to do is to check each monomial in g to see if any other g_i in G can reduce monomial m in g and add the extension of g_i into $Poly$, as shown in line 4 to line 6. After the monomial check, we get a set of polynomials $Poly$. Notice that g is the largest polynomial in $Poly$. After the Gaussian Elimination in line 7, the corresponding polynomial g' of g will have the same head term. By replacing g by g' , we finish the reduction of g .

The reason that we ignore those p that have length less than $\frac{1}{2}max_length$ is for efficiency. If we reduce every g in G , then it would take too much time. Besides, what we want

is to reduce the size of G , and reducing monomials in g does not ensure reduced length of g . Take reducing $g = x^3 + x^2$ with $x^2 + x + 1$ as an example. After the reduction, $g = x^3 - x - 1$, but length of g is longer than before. Thus, we give the parameter $\frac{1}{2}max_length$ that g will have more probability having monomials that can be divided by G as well as are shorter after reduction. The parameter should be modified depending on the size of G . If G is larger and denser with its monomials (considering the density of matrix of G), we should set the parameter lower; for example, $1/3$. And if necessary, we can do the ReduceG function more than one time. From the experiment result, we recommend that this size of G use no more than half of the total memory.

3.2 Proof of Correctness

We have shown a new modified scheme of the F_4 algorithm that provides a good mechanism of trading off between time and space. Here we prove the correctness of the new scheme. To prove the correctness of the modified algorithm, we only need to prove that it satisfy the Buchberger Theorem. Since the original F_4 algorithm satisfy the Buchberger Theorem, and our scheme is based on it, equivalently, we shall prove that the new scheme will produce the same result as the F_4 algorithm.

The following subsections will show readers the correctness of the different steps of the modified scheme.

3.2.1 Select Pair step

SelectPair function only decides how many pairs will be selected in this iteration. Nevertheless, every pair will be checked sooner or later before output of the Groebner basis G .

Every strategy of SelectPair function, as long as each pair would be selected eventually, does not affect the correctness of the algorithm. It only affects the efficiency.

Our modification of Select Pair strategy looks like the one in F_4 but with one more parameter mat_size . Those pairs not checked due to the limitation of memory would be check in a later iteration.

3.2.2 Reduction step

The step Reduction of Spolynomials is the main modification in the new scheme, which really changes the structure of F_4 . In the original F_4 algorithm, this step will calculating a set of new remainders R that is not reducible by G any more. We will show that the new scheme also generates the new set of remainders R in ideal of $\langle G \rangle$ that are exactly those remainders with respect to G corresponding to the SPolynomials of selected pair, as F_4 does.

The proof works as follows.

Proof 2 *Reduction Step of the modified scheme*

Let $g_i, g_j \in G$ and $p = Spoly(g_i, g_j)$. By definition of SelectPair function, we will have $pre-SPoly_i(g_i, g_j)$ and $pre-SPoly_j(g_i, g_j)$ in $Poly$ at line 8 of Algorithm 3.1. After doing Gaussian Elimination of $Poly$ at line 11 in the first iteration, we get p .

1. Let $p' \in R$ be the new polynomial produced with respect to p after Reduction step to output. Suppose p' is divisible by G .

$\implies p' \in Poly$, by definition of modified scheme in line 13.

$\implies p' \notin R$, since $Poly$ and R are disjoint.

$\implies p'$ is not output, since only R would be output. ($\rightarrow \leftarrow$)

2. Let $p' \in R$ be the new polynomial produced with respect to p after reduction not to output. Suppose $p' \neq 0$ is not divisible by G .

$\implies p' \notin Poly$, since p' is not divisible by G .

$\implies p' \in R$, since $Poly$ and R are disjoint.

$\implies p'$ is output, since R would be output. ($\rightarrow\leftarrow$)

Thus, p' with respect to SPolynomial p is output if and only if $p' \neq 0$ is not reducible by G .

Notice that, $\#Poly$ will become less and less since $Poly''$ are those polynomials having their head-terms not appearing in $Poly$. Besides, after every reduction, p in $Poly$ becomes smaller and smaller. Eventually, the loop of checking $Poly$ will stop since $Poly$ will be empty, or there will be a p that is always divisible by G after each reduction. But the last condition can not happen since G is finite, and head terms of p are getting smaller.

Thus we have proved the correctness of Reduction step by ensuring that it is doing exactly the work done in F_4 .

3.2.3 UpdateGP step

We use the same UpdateGP function as the original F_4 algorithm, so there should not be any problem. Now let us look at the ReduceG function. This operation does not change the ideal $\langle G \rangle$. And of course, it does not change the head terms for all g in G . Thus this step does not do anything with the generated pairs, either. It produces no new head terms. This step will not affect the correctness of the algorithm, but only efficiency. We can think of this step as achieving the same effect as computing the reduced row echelon form of the large matrix in the original F_4 algorithm.

Indeed, every step in the modified scheme achieves similar goals as it does in F_4 , although they might look somewhat different. Therefore if F_4 works, then the modified scheme should work too.



Algorithm 3.1 Pseudocode of modified scheme

Input: $F, size$

REM Initialization

2 $F \leftarrow \text{ReducedRowEchelon}(F)$

3 $G, P \leftarrow \text{UpdateGP}(G, P, F)$

REM Main program - check pairs and let G satisfied the condition

5 **while** $P \neq \{\}$ **do**

REM step : Select Pair

7 $mat_size \leftarrow size - G_size - P_size$

8 $Poly, P \leftarrow \text{SelectPairs}(P, mat_size)$

REM step : Reduction

10 **while** $Poly \neq \{\}$ **do**

11 $Poly' \leftarrow \text{ReducedRowEchelon}(Poly)$

12 $Poly'' \leftarrow \{p \mid p \in Poly', HT(p) \notin HT(Poly)\}$

13 $Poly \leftarrow \{p, h^*g \mid p \in Poly'', g \in G \cup R, HT(p) = HT(h^*g)\}$

14 $R \leftarrow \{p \mid p \in Poly'', p \notin Poly\}$

15 **end**

REM step : UpdateGP

17 $G, P \leftarrow \text{UpdateGP}(G, P, R)$

18 $G \leftarrow \text{ReduceG}(G)$

19 **end**

Output: G

Algorithm 3.2 Pseudocode of SelectPairs function in modified scheme

Input: P, mat_size

```
1 while  $P \neq \{\}$  do
2    $pair \leftarrow \text{SelectStrategy}(P)$ 
3    $P \leftarrow P \setminus pair$ 
4    $Poly \leftarrow Poly \cup \{pair.f_i mult * pair.f_i, pair.f_j mult * pair.f_j\}$ 
5   if  $\#Monomials(Poly) * \#Poly \geq mat\_size$  then
6     break
7   end
8 end
```

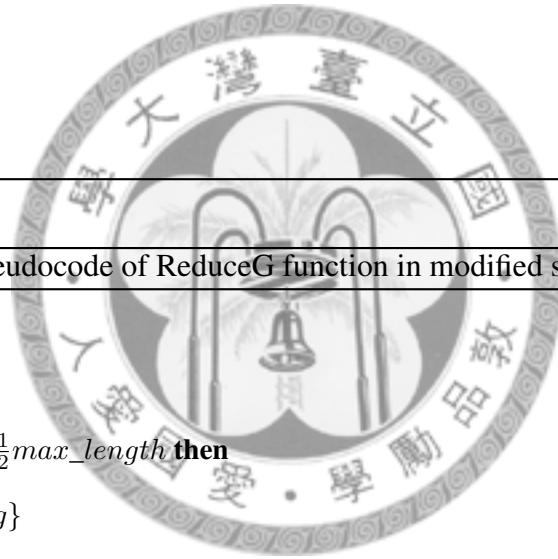
Output: $Poly, P$

Algorithm 3.3 Pseudocode of ReduceG function in modified scheme

Input: G

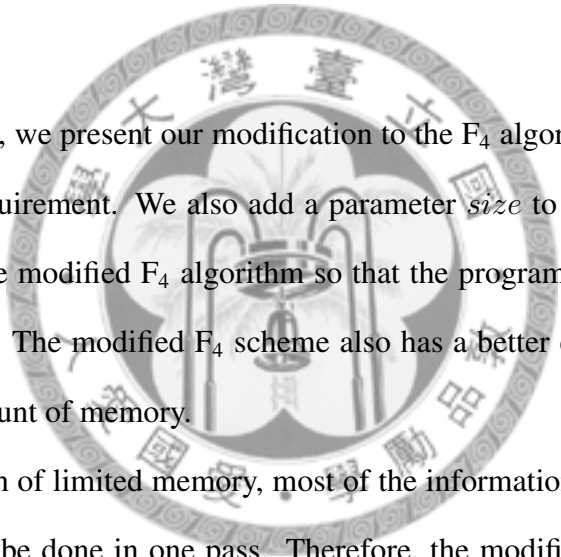
```
1 for  $g \in G$  do
2   if length of  $g \geq \frac{1}{2} max\_length$  then
3      $Poly \leftarrow \{g\}$ 
4     for  $m \in Monomials(g)$  do
5        $Poly \leftarrow Poly \cup \{h * g' \mid g' \in G, m = HT(h * g')\}$ 
6     end
7      $Poly' \leftarrow \text{ReducedRowEchelon}(Poly)$ 
8      $g \leftarrow g' \in Poly, HT(g') = HT(g)$ 
9   end
10 end
```

Output: G



Chapter 4

Experiment Results



In the previous chapter, we present our modification to the F_4 algorithm that decreases the minimum memory requirement. We also add a parameter *size* to control the total memory consumption of the modified F_4 algorithm so that the program can run with different memory requirements. The modified F_4 scheme also has a better efficiency than original F_4 given the same amount of memory.

Under the condition of limited memory, most of the information cannot be stored, and the calculation cannot be done in one pass. Therefore, the modified scheme needs more computation, for example, more transformation between matrix and polynomial forms. All the overhead makes the modified scheme slower than the original F_4 when there is enough memory. However, we will show in this chapter that the modified scheme has better time-memory product than the original F_4 algorithm. This means that if we can spend a little more time and save a lot of memory. Furthermore, users can trade time for memory at various degrees, making the modified scheme extremely flexible.

In this chapter, we will show our experiment result and compare it with that of the

original F_4 algorithm.

First, we implement both F_4 and the modified scheme in C++ language. Both of these two programs use the same data structures and lower-level field operations. The Polynomial Ring of the input data is $\text{GF}_{16}\langle X \rangle$ with $X = \langle x_1, x_2, \dots, x_n \rangle$.

According to the custom of multivariate cryptography, we let n be the number of variables and m be the number of input polynomials. The experiment is performed on the following three different input parameters to see how the modified scheme works in the ordinary systems as well as overdetermined systems:

1. $m = n + 2$
2. $m = 1.5n$
3. $m = 2n$

Section 4.1 shows the experiment result of the input parameter $m = n+2$ with n ranging from 6 to 13. Section 4.2 shows the experiment result of the input parameter $m = 1.5n$ with n ranging from 8 to 15. Section 4.3 shows the experiment result of the input parameter $m = 2n$ with n ranging from 10 to 18. The last Section 4.4 presents a generalized analysis of the three different input parameters.

The experiments shown in this chapter are performed on a computer with two Intel Xeon E5620 CPUs running at 2.40 GHz. Each processor has 128 KB of L1 cache, 1 MB of L2 cache, and 8 MB of L3 cache. The main memory is 24 GB running at 1333 MHz. Each data point here represents the average of 20 runs.

4.1 $m = n + 2$

We will check the case $m = n + 2$ in this section. We compare the running time of F_4 algorithm from $n = 8$ to $n = 15$ with our modified scheme.

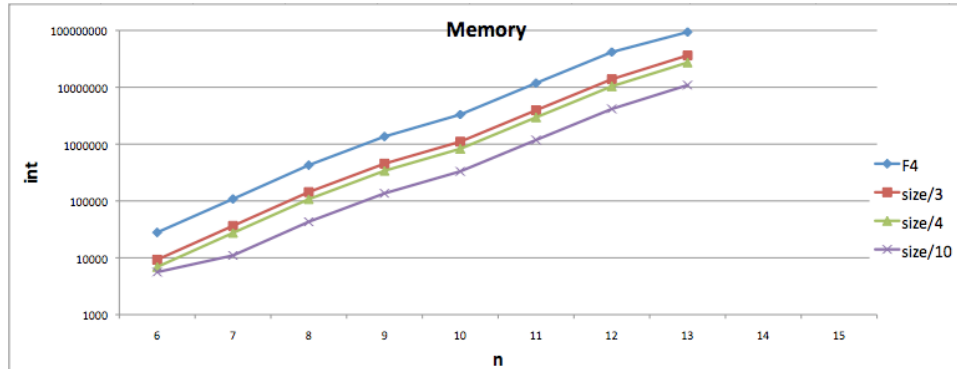


Figure 4.1: Memory curve for $m = n+2$ (unit:int)

First, we see from Figure 4.1 that the space complexity of F_4 algorithm is indeed exponential by noting that the y-axis is in logarithmic scale. Here we run the F_4 program first and record the memory usage. We then try to run the modified scheme with $1/3$, $1/4$, and $1/10$ memory. The reason why we use these ratios $1/3$ and $1/4$ is that we want to see that how the program performs under different memory budgets. The reason for the ratio $1/10$ is that we want to see how the program performs when the memory budget is extremely low. Notice that for $n = 6$, instead of using $1/10$ -memory of F_4 , we use $1/5$ -memory of F_4 , as the memory usage of F_4 at $n = 6$ is too small to have the enough memory to run the modified program.

From Table 4.1 and Figure 4.2, we see how the programs performs in different n and amounts of memory. In Figure 4.2, we notice that the time complexity of F_4 algorithm is again exponential, as the y-axis is in logarithmic scale, and so is the modified scheme. The

Table 4.1: Running Time for $m = n+2$ (unit:s)

n	6	7	8	9	10	11	12	13
F_4	0.01	0.10	0.57	3.29	16.14	76.24	868.29	4567.00
size/3	0.05	0.14	0.62	3.38	23.43	114.76	1041.52	4956.00
size/4	0.01	0.10	0.57	3.19	25.90	119.10	1031.48	5140.00
size/10	0.05	0.25	1.00	5.10	37.67	183.95	1334.67	7578.50

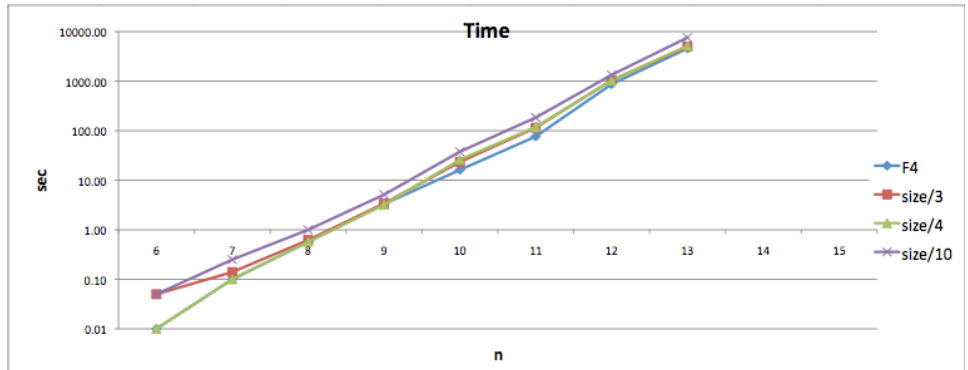


Figure 4.2: Time curve for $m = n+2$ (unit:s)

two programs have the same time complexity.

In Table 4.1, we also see that the running time of 1/3-memory and 1/4-memory do not differ very much, as their memory usages are also similar. However, when the memory is extremely low, we see that the running time apparently becomes longer. It takes more than 2 times running time when n is small, and 1.5 times when n is larger.

In Figure 4.3, we use the time-memory product as the metric to see how the programs trade off between time and memory. We calculate the time-memory product and compare this value with that of F_4 algorithm at every point. At $n = 6$, since the time is small (less than one second), the ratio is large, and we got unusual value here. In most cases, if we use 1/3 memory, we will need to run 1.2 times longer. For 1/4 memory, we also need around

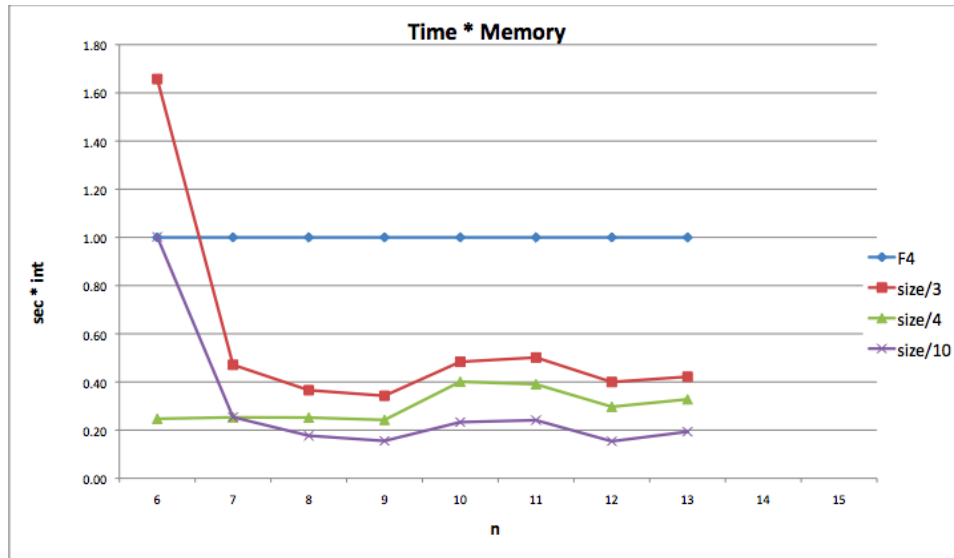


Figure 4.3: Time * Memory curve for $m = n+2$

1.2 times running time, which means it takes almost the same calculating time if we do not change the memory usage much. However, for 1/10 memory, it takes 2 times longer.

We also see from Figure 4.3 that there is a small peak between $n = 9$ and $n = 12$. This is because we do not have a good control of the size of G , and this influences the free memory we can use to check SPolynomials. However, it is not obvious since the memory usage of F_4 in the case $m = n + 2$ is large and grows very fast. When n is larger, the ratio decreases again.

Notice that the metric function can vary. If the memory is more important to the user, then it can be changed to, e.g., $time \cdot memory^2$ or $time \cdot memory^3$.

4.2 $m = 1.5n$

We will discuss the case $m = 1.5n$ in this section. We compare the running time of F_4 algorithm from $n = 8$ to $n = 15$ with the modified scheme using 1/3, 1/4 and 1/10

memory.

Table 4.2: Running Time for $m = 1.5n$ (unit:s)

n	8	9	10	11	12	13	14	15
F_4	0.14	0.86	1.95	8.81	21.95	92.57	211.33	2805.24
size/3	0.14	1.05	2.43	11.71	31.38	123.24	395.33	3010.90
size/4	0.10	1.00	2.52	11.71	32.62	182.43	647.62	3392.95
size/10	0.19	0.86	4.05	19.48	60.14	358.62	976.48	4703.29

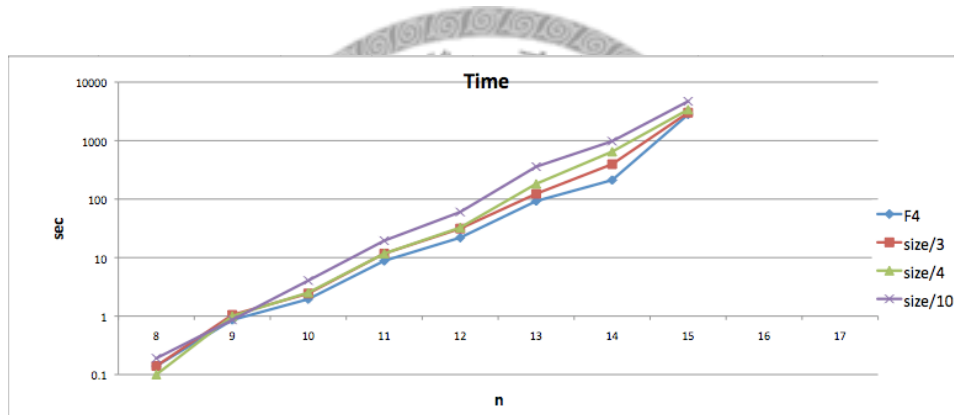


Figure 4.4: Time curve for $m = 1.5n$ (unit:s)

From Table 4.2 and Figure 4.4, we see the curves of F_4 and the modified scheme in overdetermined systems. Obviously, the time complexity is exponential, and the exponents are similar in both schemes.

In Table 4.1, we also see that the running time of 1/3-memory and 1/4-memory cases do not differ very much except when $n = 14$. When the memory usage is 1/10, it takes longer executing time than $m = n + 2$; when n is small, it takes 2 to 3 times longer, and when n is larger, it take 1.5 to 2 times longer.

Notice that when $n = 14$, the ratio of running time is larger. This is because in overdetermined systems, it takes more memory to record G and P , and thus the memory for storing the matrix decreases. However, when n is larger it decreased again.

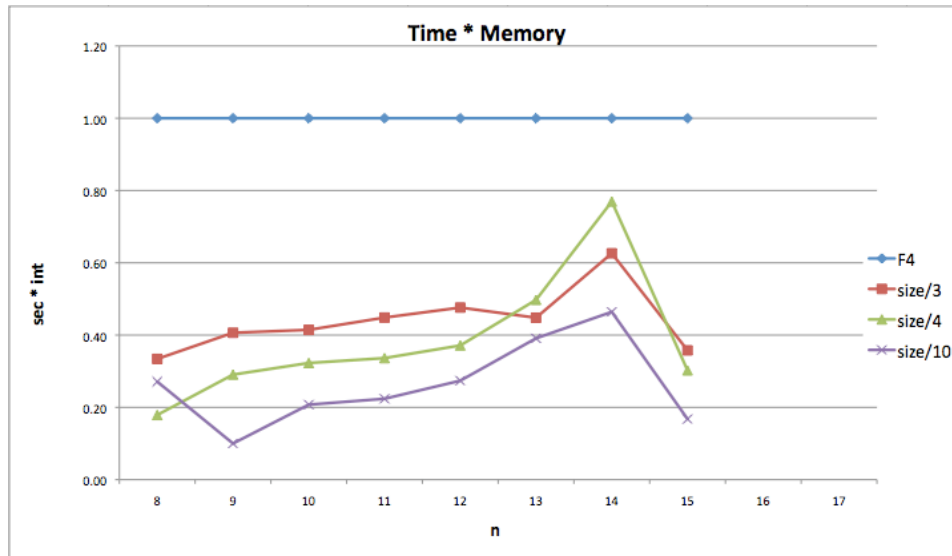


Figure 4.5: Time * Memory curve for $m = 1.5n$

Figure 4.5 shows the time-memory product ratio. We see that there is a larger peak at $n = 14$, as the ratio of time here is larger. And the ratio decreases again. The ratio of the 1/3-memory case is around 0.4, 1/4-memory, around 0.3, and 1/10, around 0.2. This is the same as $m = n + 2$. That is, in overdetermined systems of $m = 1.5n$, to use 1/10 memory, we will get 2 times longer, too.

4.3 $m = 2n$

We will discuss the case $m = 2n$ in this section. These are also overdetermined systems. We have more information here in input data than $m = n + 2$ and $m = 1.5n$. We compare

the running time of F_4 algorithm from $n = 10$ to $n = 18$ with that of the modified scheme using $1/3$, $1/4$ and $1/8$ memory. The reason why we do not use $1/10$ memory is that it is simply too small. It cannot even record all information of G and P .

Table 4.3: Running Time for $m = 2n$ (unit:s)

n	10	11	12	13	14	15	16	17	18
F_4	0.57	1.38	3.57	7.67	18.10	42.05	163.00	406.10	1707.86
size/3	0.57	1.33	3.43	7.24	21.81	75.10	415.95	965.43	1906.29
size/4	0.52	1.24	3.10	7.00	40.48	108.76	459.00	943.05	1919.38
size/8	0.43	1.19	5.19	18.67	69.86	223.67	569.95	1059.05	3120.95

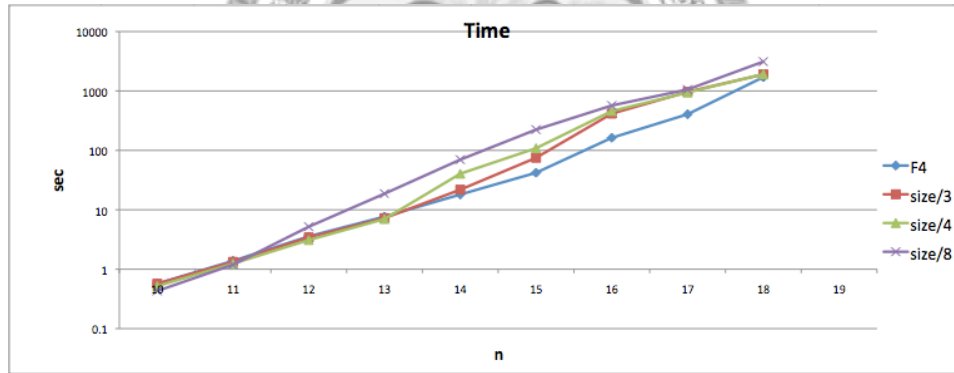


Figure 4.6: Time curve for $m = 2n$ (unit:s)

From Table 4.3 and Figure 4.6, we see the curves of F_4 and modified scheme in overdetermined systems for $m = 2n$. The time complexity is exponential, and the exponents are also similar.

There is also a peak in Figure 4.7, and this peak is larger than that in $m = n + 2$ and $m = 1.5n$. However, when n becomes larger, it decreases again. And the ratio of $1/3$

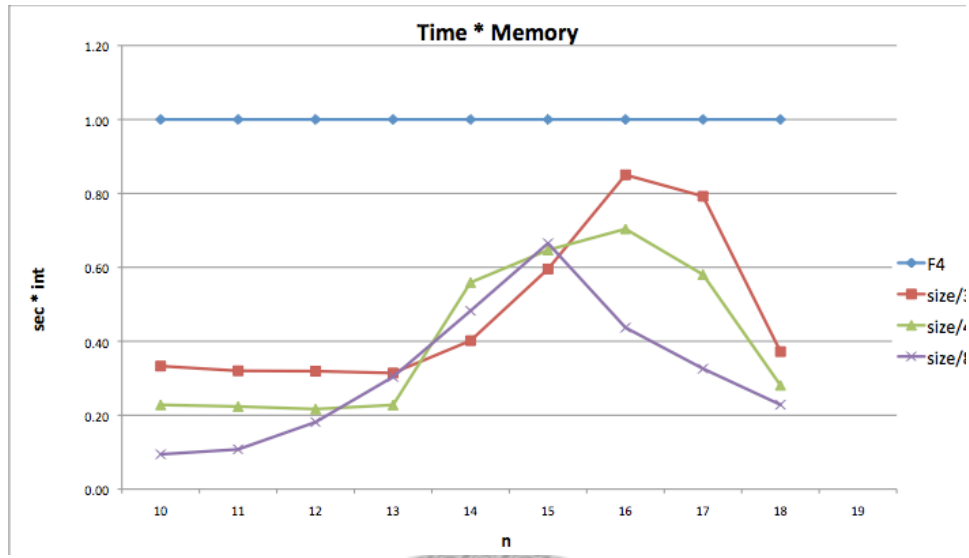


Figure 4.7: Time * Memory curve for $m = 2n$

memory is around 0.3 to 0.4, ratio of 1/4 memory, around 0.2 to 0.3, and the ratio of 1/8 memory, around 0.1 to 0.2.

4.4 Analysis

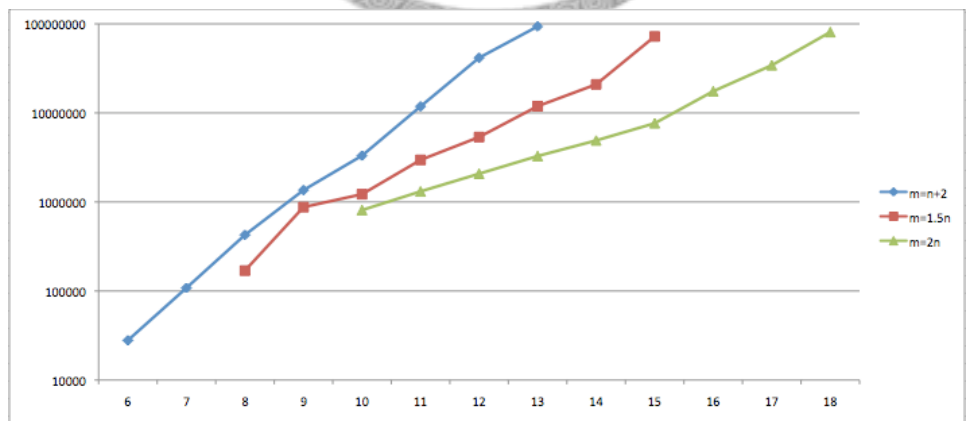


Figure 4.8: Memory curve for each case (unit:int)

Figure 4.8 shows the total memory usage of F_4 algorithm in the cases $m = n + 2$, $m = 1.5n$, and $m = 2n$. We see that in each of these cases, the space complexity of F_4 algorithm is all exponential. However, when there is more information in Input F , F_4 needs less memory. Solving overdetermined systems $m = 2n$ needs less memory and has a smaller exponent in the exponential time complexity.

Since it use less memory in $m = 1.5n$ and $m = 2n$, there is fewer memory stress, and less space to reduce the memory. The modified scheme does not work as well as it does in $m = n + 2$, since there are obvious peak. Nevertheless, it still has a small ratio of $time*memory$ than F_4 algorithm.

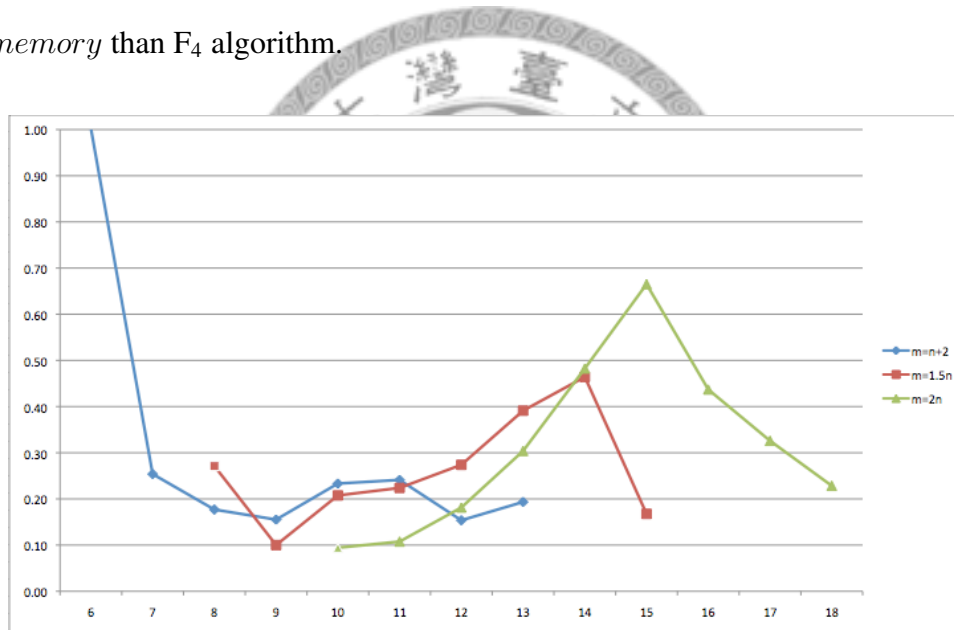
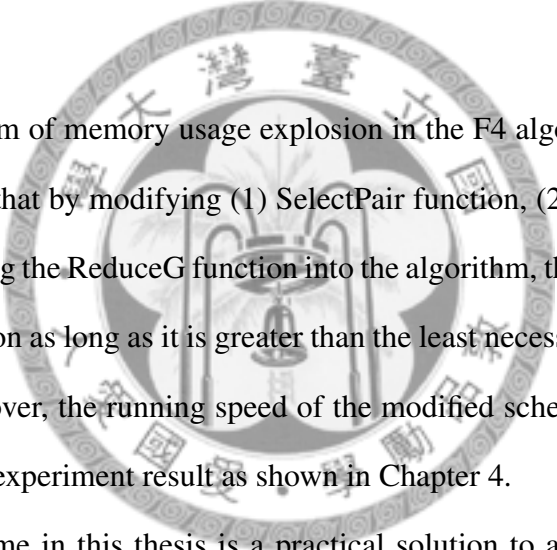


Figure 4.9: Time * Memory curve for each case for 1/10 and 1/8 memory

In our implementation, we focus on the matrix size and do not have a better control over the size of G as we have for matrix. But we do have an efficient way in control matrix size.

Chapter 5

Conclusion



We mitigate the problem of memory usage explosion in the F4 algorithm in this thesis. In Chapter 3, we can see that by modifying (1) SelectPair function, (2) Reduction step of the algorithm and (3) adding the ReduceG function into the algorithm, the modified scheme can run in memory limitation as long as it is greater than the least necessary memory (size of G plus size of P). Moreover, the running speed of the modified scheme is not unendurable, which is supported by experiment result as shown in Chapter 4.

The modified scheme in this thesis is a practical solution to add some memory consumption control mechanisms to the F4 algorithm. The modified scheme now can fit in most of memory limitation. Users now may do a time-memory trade-off based on their demand and the hardware support they have.

In the modified scheme, not only the checking of SPolynomials in one Reduction loop decreases, the check steps and the transformation between matrix and polynomial forms also increase. Due to this overhead, the modified scheme is not as fast as the original F4 algorithm when there is enough memory. However, when the memory is not sufficient, the

modified scheme works better than the original F4 algorithm. The running time of original F4 algorithm is too long that to be practical under limited memory. On the other hand, the modified F4 scheme can be executed under 1/10 memory and have only 2 times running time. This achieves our goal in this thesis — let F4 can be executed on different hardware and still efficient.

Our scheme in this thesis is not perfect. Indeed, the scheme shown here is only a concept of the basic structure. There is room for improvement to make the program run faster. Users can optimize many parts of the program to improve the efficiency. For example, users may optimize the transformation between polynomials and matrices since the operation would be executed many times.

Users may also like to optimize the ReduceG function, which is not in the original F4 algorithm. Indeed, we do not have an optimized operation of reducing G in this thesis. Users can change every step in ReduceG as long as G is minimized and does not change the head terms in G . This is an important step of the modified scheme. If we can optimize it, then it would give a great improvement on the efficiency of the modified scheme.

In the Reduction step of the modified F4 scheme, we can see that the size of the matrix will become smaller and smaller. Though we do not use this free space in the scheme in this thesis, users can try to use this extra memory to enhance the performance, like trying to check the divisibility of monomials next to head terms, which has the highest probability to become head terms of new remainders. Efficient use of this memory would help the performance a lot.

There are also some applications of the modified scheme that can be done in the future. Having the advantage of less memory consumption, the modified scheme can be ported into computation systems with limited memory like FPGA. We even can port the program

into a parallel or distributed system that has only very little resource. For example, we can do the design — select some pairs and assign the Reduction step to a node, which only needs a small amount of computation resource, and passes the set of new remainders R to the main program.



Bibliography

- [1] T. Becker, H. Kredel, and V. Weispfenning. *Gröbner bases: a computational approach to commutative algebra*. Springer-Verlag, London, UK, 0 edition, 4 1993.
- [2] B. Buchberger. A theoretical basis for the reduction of polynomials to canonical forms. *SIGSAM Bull.*, 10(3):19–29, 1976.
- [3] Bo-Yin Yang and Jiun-Ming Chen. All in the xl family: Theory and practice. In *ICISC*, pages 67–86, 2004.
- [4] Nicolas Courtois, Nicolas Courtois, Er Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. *IN ADVANCES IN CRYPTOLOGY, EUROCRYPT'2000, LNCS 1807*, 1807:392–407, 2000.
- [5] Jean-Charles Faugère. A new efficient algorithm for computing gröbner bases (f4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999.