南開大學
Nankai University

计算机学院

算法导论实验报告

基于状态压缩方法的 Buchberger 算法高斯消元化

姓名：林逸典

学号：2213917

专业：计算机科学与技术

2024 年 3 月 29 日

目录

1 实验概要

在本次实验中，我们聚焦于特殊高斯消元法——Buchberger 算法在计算 Grobner 基过程中的优化问题。首先，我们对问题背景进行了具体的描述，明确了研究的重要性和实际需求。随后，我们系统地查阅了相关问题的研究现状，分析了当前方法的优势和不足，为后续的研究奠定了坚实的基础。

在尝试优化过程中，我们起初采用了稀疏矩阵链式储存的常规优化方法。然而，实验结果表明，这种优化方法的效果并不理想，无法满足我们的预期需求。这使我们意识到，对于这类特殊问题，需要采用更为精准和高效的优化策略。

基于问题的特殊性，我们最终采取了状态压缩的思想进行优化。通过状态压缩，我们成功地将问题的规模进行了有效的缩减，从而显著提高了计算效率。实验结果表明，这种方法在时间和空间上均取得了很好的优化效果，为 Buchberger 算法在计算 Grobner 基方面的应用提供了新的思路和方法。

本次实验的代码已上传至 GitHub。

2 问题描述

2.1 特殊高斯消元法——Buchberger 算法计算 Grobner 基

2.1.1 背景描述

Grobner 基在代数几何和计算机科学中占据着举足轻重的地位，它作为多项式环中一组特殊的生成元，确保了通过这组生成元进行除法运算时结果的唯一性。在计算机代数和符号计算的广阔领域中，Grobner 基发挥着不可或缺的作用，常用于求解多项式方程组、进行理想成员判断以及多项式约化等任务。

作为计算 Grobner 基的核心算法，Buchberger 算法在算法设计领域具有显著地位。该算法可以被视为欧几里得算法和线性系统中高斯消元法的泛化实现。它通过对初始基中的多项式进行 S-多项式运算和约化，逐步生成新的多项式，直至满足 Grobner 基的条件。

2.1.2 算法描述

高斯消元法是一种求解线性方程组的经典算法，它通过对线性方程组的增广矩阵进行一系列初等行变换，从而实现方程组的求解。

高斯消元法主要包含前向消元和回代求解两个核心步骤，其中前向消元在我们本次实验中起到重要作用：

Algorithm 1 高斯消元法: 前向消元

for $I \leftarrow 1$ to n **do**
    选择第 I 列中绝对值最大的元素作为主元 $pivot$
    找到主元所在的行 $pivotcol$
    交换第 I 行和第 $pivotcol$ 行
    将第 I 行的主元元素变为 1（行内元素除以 $pivot$）
    for $i \leftarrow I+1$ to n **do**
        计算乘数 $k = C[i][I]$
        for $j \leftarrow I$ to $n+1$ **do**
            $C[i][j] \leftarrow C[i][j] - k \times C[I][j]$
        end for

11: **end for**
12: **end for**

Buchberger 算法一般采用符号运算的形式进行，在本次实验中我们将它转化为一种特殊形式的高斯消元算法。其核心思想在于构建一个初始的消元矩阵，该矩阵的元素仅为 0 和 1，且呈现出高度的稀疏性。在算法的执行过程中，仅需要使用被巧妙地简化为异或运算的减法运算，大大提升了计算效率。同时，消元矩阵的行被精心划分为消元子和消元行，其中消元子的特性在于其首个非零元素的列均不相同，这与常规高斯消元法中的主元行概念相契合。通过前向消元处理，消元行逐步转化为新的消元子或被简化为全 0 行并被舍弃，最终当所有消元行均经过处理后，我们得到一个可能存在置空行的类上三角消元子矩阵，算法得以完成。

	普通高斯消元法	Buchberger 算法
元素	实数	0 和 1
矩阵稀疏性	均可	稀疏
运算类型	常规加减乘除运算	退化为异或的减法运算
回代求解	是	否
运算结果	解向量	类上三角消元子矩阵
主元处理	需要选择主元并且通过行交换将其置于主对角线上	消元子为主元，前向消去后将新消元子加入消元子矩阵

表 1: 普通高斯消元法与 Buchberger 算法对比

2.1.3 算法实现

为了将 Buchberger 算法有效地转化为高斯消元形式，我们在初步实现中采用了基于前向消元的朴素策略，我们称之为朴素 Buchberger 高斯消元化。这一策略主要涉及消元行和消元子的划分与匹配运算。

Algorithm 2 朴素 Buchberger 高斯消元化: 前向消元

1: **for** $I \leftarrow 0$ to $row - 1$ **do**
2: **if** A 的第 I 行被置空 **then**
3: **for** $i \leftarrow 0$ to $col2 - 1$ **do**
4: **if** B 的第 i 行未被删除且 B[i,I] 为 1 **then**
5: 将 B 的第 i 行赋值给 A 的第 I 行
6: 从 B 中删除第 i 行
7: **break**
8: **end if**
9: **end for**
10: **if** A 的第 I 行仍为空 **then**
11: **continue**
12: **end if**
13: **end if**
14: **for** $i \leftarrow 1$ to $col2 - 1$ **do**
15: **if** B 的第 i 行未被删除且 B[i,I] 为 1 **then**
16: **for** $j \leftarrow I$ to row **do**
17: $B[i][j] \leftarrow B[i][j] \oplus A[I][j]$

```
            end for
        end if
    end for
end for
```

3 研究现状

高斯消元法作为线性代数中的基础算法，其改进和优化工作已经取得了丰富的成果 [3] [1]。特别是在处理稀疏矩阵时，高斯消元法的优化显得尤为重要。当前的研究主要聚焦于链式存储结构的应用以及消元顺序的选择，通过这些手段有效地提高了算法在处理稀疏矩阵时的效率 [4] [7] [6]。

另一方面，Buchberger 算法在计算 Grobner 基方面拥有坚实的数学理论基础。然而，关于其高斯消元化的研究相对较为稀缺。尽管 Grobner 基的计算在多项式代数和计算机科学等领域具有广泛的应用，但如何高效地实现这一计算过程仍是一个具有挑战性的问题 [2] [5]。

在本次研究中，我们计划结合稀疏矩阵高斯消元优化的现有成果，首先设计并实现基于这些成果的算法。随后，我们将根据 Buchberger 高斯消元化算法的特殊性，进一步探索并应用新的优化策略。

4 实验环境

4.1 实验平台

本实验采用华为鲲鹏 920 处理器作为核心计算平台，其基于 ARM 架构，CPU 主频高达 2.6GHz。该平台提供了稳定可靠的计算环境，为实验的顺利进行奠定了坚实基础。此外，实验平台配备了 191GB 的内存（RAM），确保了在处理大规模数据集时的高效性能。

处理器架构	ARM
CPU 型号	华为鲲鹏 920 处理器
CPU 主频	2.6GHz
内存（RAM）	191GB

表 2: 实验环境配置

4.2 实验数据集

本实验所采用的数据集来源于相关研究，经过严格的格式规范化和数据清洗过程，确保了数据集的准确性和可靠性。数据集包含多个测试样例，每个样例具有不同的消元矩阵列数、消元子行数和消元行行数，以全面评估算法在不同规模问题上的性能表现。由于实验条件限制，我们将在较小的数据集上进行时间相关性能的测试，而在较大的数据集上则侧重于空间使用效率的评估。

测试编号	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10	test11
消元矩阵列数	130	254	562	1011	2362	3799	8399	23045	37960	43577	85401
消元子行数	22	106	170	539	1226	2759	6375	18748	29304	39477	5724
消元行行数	8	53	53	263	453	1953	4345	14325	14921	54274	756

表 3: 实验数据集

5 链式 Buchberger 高斯消元化

在稀疏矩阵的运算中，链式存储的优化方法是一种常见且有效的策略。在应用于 Buchberger 算法时，利用消元行和消元子的不同，我们能够将退化为异或运算的减法运算进一步简化，使其降低至原始运算量的一半。

5.1 算法实现

Algorithm 3 链式 Buchberger 高斯消元化: 前向消元

```
for I ← 0 to row − 1 do
    if A 的第 I 行被置空 then
        for i ← 0 to col2 − 1 do
            if B 的第 i 行未被删除且 B 的首项索引为 I then
                将 B 的第 i 行赋值给 A 的第 I 行
                从 B 中删除第 i 行
                break
            end if
        end for
    end if
    if A 的第 I 行仍为空 then
        continue
    end if
    for i ← 0 to col2 − 1 do
        if B 的第 i 行未被删除且 B 的首项索引为 I then
            If A 的第 I 行中存在某项而 B 的第 i 行中不存在该项, then 将该项添加到 B 的第 i 行
            If A 的第 I 行和 B 的第 i 行中均存在某项, then 从 B 的第 i 行中删除该项
        end if
    end for
end for
```

5.2 性能测试与结果分析

通过对朴素算法和链式算法进行性能测试，我们记录了两种算法的执行时间和最大占用空间，并计算了优化比。以下是对测试结果的详细分析。

	test1	test2	test3	test4	test5	test6
simple	0.0360	1.8864	4.3903	169.78	1504.0	22364
list	0.2875	13.744	28.026	1314.2	12356	229848
rate	0.13	0.14	0.16	0.13	0.12	0.10

表 4: 朴素算法和链式算法的时间对比 (单位: 毫秒)

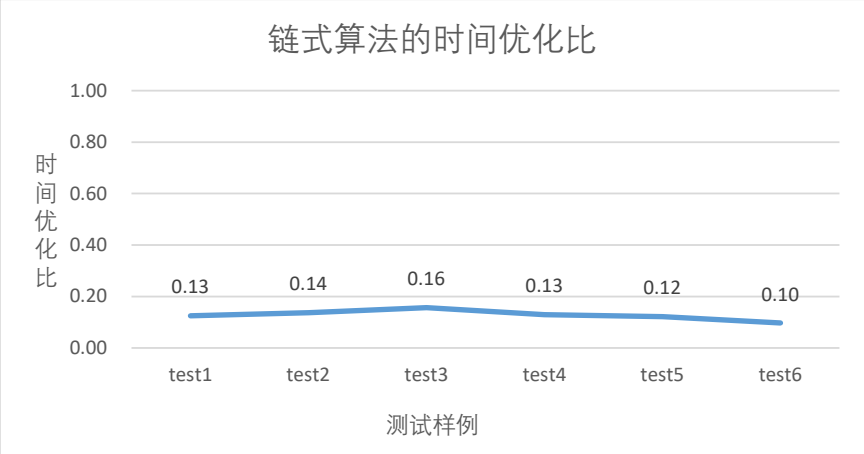

图 5.1: 链式算法的时间优化比

	test7	test8	test9	test10	test11
simple	111296	848448	1966016	4172160	7193024
list	>541568	>2080512	>3114496	>8064576	>849920
rate	0.21	0.41	0.63	0.52	8.46

表 5: 朴素算法和链式算法的空间对比 (单位:KB)

从图表和表格中可以看出，在大部分情况下，链式算法在时间和空间上的表现均不如朴素算法。这主要是由于 Buchberger 算法的特性所致。在朴素算法中，消元矩阵的元素 0 和 1 可以以布尔类型存储，而在链式算法中，这些元素必须以较大的整数类型存储。随着 Buchberger 算法的进行，消元矩阵的元素会迅速收敛于 0 和 1 的等比例分布，这导致链式算法在存储效率上不具有优势。此外，链表的指针域也会占用额外的空间。在运算过程中，消元行的元素会频繁发生变化，导致链表节点被反复创建和删除，进一步降低了运算效率。

此外，值得注意的是，链式算法在某些特定情况下（如 test11）可能会表现出较高的优化比。这可能是由于测试案例的特性或算法实现的具体细节所致。然而，在大多数情况下，朴素算法仍然表现出更好的性能。

综上所述，对于当前的测试案例和算法实现，朴素算法在时间和空间效率上均优于链式算法。对此，我们将充分考虑以上特性，采用状态压缩 Buchberger 高斯消元化算法对朴素算法进行优化。

6 状态压缩 Buchberger 高斯消元化

考虑到 Buchberger 算法的消元矩阵仅包含 0 和 1 两种元素，并且其主要运算为退化为异或的减法运算，我们提出了一种基于状态压缩的优化方法。尽管初始的消元矩阵表现为稀疏矩阵，但随着运算的进行，消元矩阵的元素分布会迅速趋向于 0 和 1 的等比例分布。在这种背景下，我们采用状态压缩技术，使用 unsigned short 类型的变量来存储数据。这种存储方式可以充分利用计算机的内存空间，并通过按位异或运算来实现消元操作。这种优化方法不仅减少了内存占用，还提高了运算速度，从而实现了时间和空间上的双重优化。

6.1 算法实现

Algorithm 4 状态压缩 Buchberger 高斯消元化: 前向消元

```
for I ← 0 to row − 1 do
    if A 的第 I 行被置空 then
        for i ← 0 to col2 − 1 do
            if B 的第 i 行未被删除且 B 的对应状态位为 1 then
                将 B 的第 i 行赋值给 A 的第 I 行
                从 B 中删除第 i 行
                break
            end if
        end for
    end if
    if A 的第 I 行仍为空 then
        continue
    end if
    for i ← 0 to col2 − 1 do
        if B 的第 i 行未被删除且 B 的对应状态位为 1 then
            for j ← I/16 to row/16 do
                B[i][j] ← B[i][j] ⊕ A[I][j]
            end for
        end if
    end for
end for
```

6.2 性能测试与结果分析

为了评估朴素算法与状态压缩算法的性能，我们进行了详细的性能测试，并记录了两种算法的执行时间和最大占用空间。随后，我们计算了优化比，以便对两种算法的性能进行直观的比较。以下是针对测试结果的深入分析。

	test1	test2	test3	test4	test5	test6	test7
simple	0.0359	1.8864	4.3902	169.78	1503.9	22364	303784
state	0.0081	0.2180	0.4804	13.232	106.62	1540.1	20489
rate	4.40	8.65	9.13	12.83	14.10	14.52	14.82

表 6: 朴素算法和状态压缩的时间对比 (单位: 毫秒)

	test7	test8	test9	test10	test11
simple	111296	848448	1966016	4172160	7193024
state	18560	110912	251648	527232	849920
rate	6.00	7.65	7.81	7.91	8.46

表 7: 朴素算法和状态压缩的空间对比 (单位:KB)

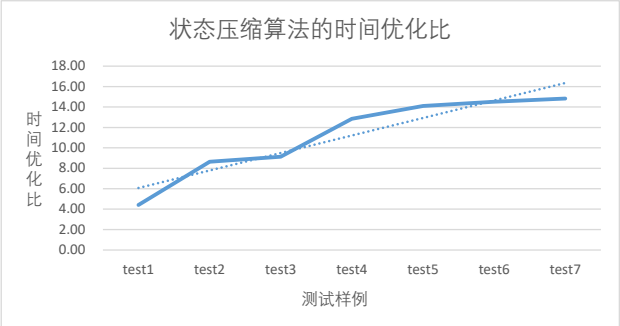

图 6.2: 状态压缩算法的时间优化比

图 6.3: 状态压缩算法的空间优化比

根据表6和表7中的数据，我们可以看出状态压缩算法在时间和空间性能上均优于朴素算法。具体而言，状态压缩算法的时间优化比随着问题规模的增大而稳定上升，这表明在更大规模的问题上，状态压缩算法可以发挥更好的优化效果。而空间优化比则相对稳定在 8 左右，这与状态压缩算法每 1Byte 存储 8 个状态位的理论预期相符。这一结果证明了我们对 Buchberger 算法特性的深入理解和状态压缩算法设计的有效性。

综上所述，状态压缩算法在性能上相较于朴素算法具有显著优势，这得益于对问题特性的深入挖掘和算法设计的优化。

7 总结

本实验旨在优化特殊高斯消元法——Buchberger 算法在计算 Grobner 基时的性能。通过实施链式 Buchberger 高斯消元化和状态压缩 Buchberger 高斯消元化两种方法，并对它们的性能进行了深入的测试与分析，我们取得了显著的优化效果。

首先，我们尝试了链式 Buchberger 高斯消元化方法。尽管该方法在理论上有减少冗余计算的潜力，但在实际性能测试中，我们发现其效果并不如预期显著。这促使我们进一步深入研究算法的特性和行为模式，以寻找更有效的优化手段。

随后，我们转向状态压缩技术的探索，并成功实现了状态压缩 Buchberger 高斯消元化方法。通过利用状态压缩技术，我们在不增加计算复杂度的前提下，显著降低了算法的时间消耗和空间占用。性能测试结果表明，随着问题规模的增大，状态压缩算法的时间优化比稳步提升，空间优化比也保持稳定，这与我们的理论预期相符。

综上所述，通过本实验的研究，我们成功利用状态压缩技术优化了 Buchberger 算法计算 Grobner 基的性能。这不仅提升了算法本身的效率，也为解决类似问题提供了新的思路和技术手段。

我们还将继续探索优化状态压缩 Buchberger 高斯消元化的方法，特别是结合并行计算技术，进一步提升算法的执行效率。我们将继续完善相关实验，并将最新的研究成果及时分享到 GitHub。

参考文献

[1] 刘单, 万新儒, 彭丽君, and 陈恳. 规格化对高斯消元法计算速度的影响. 南昌大学学报: 理科版, 39(2):135–135, 2015.

[2] 张韶华, ZHANG, Shaohua, School, of, mathematics, , , statistics, Yangtze, and Normal. Euclid 算法,guass 消元法以及 buchberger 算法研究 (英文). 应用数学, 31(1):5, 2018.

[3] 彭朝英. 高斯消元法的改进及其在工程上的应用. 邵阳学院学报: 自然科学版, 8(2):5, 2011.

[4] 林首位, 徐宏, 侯华, 褚忠, and 龚荣良. 大型稀疏矩阵线性化方程组的数值解法. 华北工学院学报, 23(004):265–269, 2002.

[5] 段绍华. 高级数据加密标准的代数攻击方法研究. PhD thesis, 中南大学.

[6] 王玉卿. 高斯消元的顺序和稀疏矩阵的图解. 沈阳工业大学学报, 15(3):9, 1993.

[7] 董哲林, 陈国华, 蒋炎坤, 王春发, and 庄瑞宜. 一种稀疏矩阵算法及其在燃烧室传热计算中的应用. 车用发动机, (4):6, 2011.