第五届（2020年）

全国高校密码数学挑战赛

参赛论文

赛题三：子集和问题

战队名称 covteam

学 校 西华大学

学 院 计算机与软件工程学院

专 业 信息安全

参赛选手 杨佳立,白泽兴,赵之奕

指导教师 周洁

二零二零六月

摘 要

子集和问题又称背包问题，其已经被证明为NP-完全问题，在PNP的基本假设下，不可能找到求解子集和问题的多项式时间算法。但是，精心设计的求解算法往往能降低求解的计算复杂度。维数越高的子集和问题越难求解，根据维数的不同，本论文采用三种方法对赛题进行求解。对于维数n=40到n=80的低维子集和问题，通过对原始的Schroeppel-Shamir算法进行改进，将赛题给定的汉明重量k应用到算法中，大大降低原始算法的时间和空间复杂度。此算法主要基于生日攻击碰撞的思想，其对于某个集合只需要遍历一半的搜索空间即可找到解，因此对于低维度的背包问题能准确的得出答案并且其时间和空间复杂度较低。对于维数n=100到n=180的中高维子集和问题，采用基于给定汉明重量k的HGJ (Howgrave-Graham and Joux )改进算法，降低原始HGJ算法的时间和空间复杂度，在构造搜索空间时，我们采用组合进行构建，该构造方法可以减少搜索空间的大小，提高找到碰撞的机率。最后利用HGJ改进算法，在可接受的时间和空间消耗中得到了中高维子集和问题的解。对于n=200的高维子集和问题，论文采用格基规约的方法，通过构造向量组，把子集和问题转化为最短向量问题（SVP），通过不断的处理和规约，最终我们求得了200维的较小范数。

关键词：子集和问题，背包问题，格基规约，LLL归约，最短向量问题

目 录

[第一章 引言 1](#_Toc47548631)

[1.1 赛题分析 1](#_Toc47548632)

[1.2 解决方法 1](#_Toc47548633)

[1.3 结题结果 2](#_Toc47548634)

[第二章 子集和问题 6](#_Toc47548635)

[2.1 子集和问题研究现状 6](#_Toc47548636)

[2.2 Schroeppel-Shamir算法 6](#_Toc47548637)

[2.2.1 Schroeppel-Shamir算法 6](#_Toc47548638)

[2.2.2 子集和低维（n<=80）问题求解 9](#_Toc47548639)

[2.3 Howgrave-Graham-Joux算法 10](#_Toc47548640)

[2.3.1 Howgrave-Graham-Joux算法 10](#_Toc47548641)

[2.3.2 子集和中高维（80<n<=180）问题求解 11](#_Toc47548642)

[2.4 格基规约算法 15](#_Toc47548643)

[2.4.1 格的定义 15](#_Toc47548644)

[2.4.2 格中的困难问题 15](#_Toc47548645)

[2.4.3 LLL算法 16](#_Toc47548646)

[2.4.4 BKZ算法 16](#_Toc47548647)

[2.4.5 子集和高维（n>180）问题求解 17](#_Toc47548648)

[第三章 结论 19](#_Toc47548649)

[第四章 参考文献 20](#_Toc47548650)

[第五章 谢辞 21](#_Toc47548651)

[第六章 附录 22](#_Toc47548652)

[6.1 Schroeppel-Shamir改进算法代码 22](#_Toc47548653)

[6.2 Howgrave-Graham-Joux改进算法代码 30](#_Toc47548654)

[6.3 BKZ规约算法代码 39](#_Toc47548655)

# 引言

## 赛题分析

定义子集和问题为：给定一个n元的正整数集合和同为n元的0-1集合以及，其中集合已知，已知，求解使得对于或，满足。

集合中的的值为0，1则确定了相应的是否需要放到背包里。通俗一点来说，相当于背包总重量，相当于备选装入背包的值，而则决定是否应该装入背包中。解决背包问题就是要从全部的值中选出其中的一部分使其加起来等于。为0时，表示该值并未被选中，为1时表示该值被选中。题中总共有九个挑战，从40维到200维，每20维一个等级。每个等级根据密度d分为四个题目，并且密度越高，与的比特位数越小。

## 解决方法

求解子集和问题的方法大致可以分为确定性算法和概率性算法两类。确定性算法在理论上一定可以求得子集和问题的解，但一般为非多项式时间算法，时间和空间复杂度高，仅适用于维数较低的子集和问题。概率性算法则以一定的概率得到子集和问题的解，效率高但不一定能得到解。

根据子集和问题的维数n，我们将赛题分为三类：低维子集和问题（n<=80）,中高维子集和问题（80<n<=180）,高维子集和问题(n>180)。

对于低维子集和问题，本文采用确定性的改进的Schroeppel-Shamir算法[5]。其算法首先对背包进行整数划分,随机分为四个子背包并完全枚举四个子背包的所有可能的和值，并通过four-way-merge算法从中有效且确定地寻找答案，此外如果题目已知汉明重量可以进一步缩小范围从而提高找到解的可能性。

对于中高维子集和问题，采用确定性的Howgrave-Graham-Joux改进算法[6]。其算法更关注子背包的构造情况，如果假定汉明重量为（可证明此时已经是最复杂的情况）那么我们从中构建所有的组合情况在其中选四个组合数，那么这样同样可以得到答案并且如此构造时间和空间复杂度相比Schroeppel-Shamir都要减少很多。

对于高维子集和问题，采用概率性的BKZ格基规约算法。因为确定性算法对于低维问题并不需要太多时间，并且总能找到答案，但缺点是时间复杂度太高，此外，空间复杂度对于高维度的问题同样无法承受。而格基规约算法则是通过背包元素来构造向量矩阵，之后运用格基规约算法对其进行规约将问题转化为SVP问题，并使得得到的归约基进尽可能可正交化，而当得到的归约基只包含0和1时，就是我们的背包的解。该算法只使用于高密度的子集和问题，在使用规约算法时，我们主要采用BKZ算法

## 结题结果

我们求出了第一级到第八级挑战的0-1解，第九级挑战的非0-1解。按照赛题的计分原则，总得分为：

200+400+600+800+1000+1200+1400+1600+(1800/2+25-223)=7902，

其中25为第九级挑战题目9.4给定的汉明重量，223为题目9.4所得解的欧几里德范数平方。每一级挑战的题目及答案如下。

1. **挑战难度一：题目1.1.**

**n=40，d=0.8，k=5，s=2038015735279982.**

本题的解向量为:

**X** =[0,**1**,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,**1**,

**1**,0,0,0,0,0,0,0,**1**,0,

0,0,0,**1**,0,0,0,0,0,0]

1. **挑战难度二：题目2.4.**

**n=60，d=1.1，k=7，s=75355236542335061.**

本题的解向量为:

**X** = [0,0,0,0,0,0,0,0,**1**,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,**1**,0,0,0,0,0,0,

0,0,0,0,0,0,**1**,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,**1**,**1**,0,0,**1**,0,**1**,0]

1. **挑战难度三：题目3.1.**

**n=80，d=0.8，k=10，s=3808748539414823520119275163007.**

本题的解向量为:

**X**=[0,0,0,0,0,0,**1**,0,0,0,

**1**,0,0,0,0,**1**,**1**,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,**1**,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,**1**,0,

0,0,**1**,0,0,0,0,**1**,**1**,0,

0,0,**1**,0,0,0,0,0,0,0]

1. **挑战难度四：题目4.1.**

**n=100，d=0.8，k=12，s=214326896127070329471158810547025670819.**

本题的解向量为:

**X**=[0,0,**1**,0,0,**1**,0,0,**1**,0,

0,**1**,0,0,0,0,0,0,0,0,

0,0,0,0,**1**,0,0,0,0,0,

**1**,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,**1**,0,0,0,0,0,

0,0,0,0,0,0,**1**,0,0,0,

0,0,0,0,0,**1**,0,0,0,0,

0,0,0,0,**1**,0,**1**,0,0,0,

0,0,0,0,**1**,0,0,0,0,0]

1. **挑战难度五：题目5.4.**

**n=120，d=1.1，k=15，s=5267459431779443110988747083086266.**

本题的解向量为:

**X**=[0,0,**1**,0,0,0,0,0,0,**1**,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,**1**,0,0,0,**1**,

**1**,0,0,**1**,**1**,0,**1**,0,0,0,

0,0,**1**,0,0,**1**,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,**1**,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,**1**,0,0,0,0,0,0,0,0,

**1**,0,**1**,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,**1**,0]

1. **挑战难度六：题目6.4.**

**n=140，d=1.1，k=17，s=1624091102952514246122199359089257364515.**

本题的解向量为:

**X**=[**1**,0,0,0,**1**,0,0,0,**1**,**1**,

0,0,0,0,0,**1**,0,0,0,0,

**1**,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,**1**,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,**1**,

0,0,0,0,0,0,0,0,**1**,**1**,

0,0,0,**1**,0,0,0,**1**,0,0,

0,0,0,0,0,0,0,**1**,0,0,

0,0,0,0,0,0,0,0,0,**1**,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,**1**,**1**,0,0,**1**,0,

0,0,0,0,0,0,0,0,0,0]

1. **挑战难度七：题目7.4.**

**n=160，d=1.1，k=20，s=354338660700758611344106165240137362898447564.**

本题的解向量为:

**X**=[0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,**1**,0,0,0,0,

0,0,0,0,0,0,0,**1**,0,0,

**1**,0,0,**1**,0,0,0,0,0,**1**,

0,0,0,**1**,0,0,0,0,0,0,

**1**,**1**,**1**,0,0,0,0,0,0,0,

0,**1**,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,**1**,0,0,**1**,0,

0,0,0,0,0,0,0,0,**1**,0,

**1**,**1**,0,0,0,0,**1**,0,**1**,0,

0,0,**1**,0,0,0,0,0,0,0,

0,0,**1**,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,

**1**,0,0,0,0,0,0,0,0,0]

1. **挑战难度八：题目8.4.**

**n=180，d=0.8，k=22，**

**s=757680399933334153634722410993690867586002859113929552263090603594409**

本题的整数解向量为:

**X**=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 1, 0, 1, 1, 0, 0, 0,

0, 0, 1, 0, 1, 0, 0, 0, 0, 0,

0, 0, 1, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

1, 0, 0, 1, 0, 0, 0, 1, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 1,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

1, 0, 0, 0, 1, 0, 0, 0, 1, 0,

0, 0, 0, 0, 0, 1, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 1, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 1, 0,

0, 1, 0, 1, 0, 0, 0, 1, 0, 0,

1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

1. **挑战难度九：题目9.4.**

**n=200，d=1.1，k=25，**

**s=34194585308509120772031528249247006704390121910932930512**

本题的整数解向量为:

**X**= [ **1**, 0, 0, **1**,**-1**, **3**, 0, **3**,**-1**, 0,

**2**, 0,**-3**, 0, 0, **1**, 1, **1**, 0, **2**,

**1**, **1**, 0,**-1**, **4**,**-1**,**-2**,**-2**, 0,**-1**,

**-4**, 0, 0,**-1**, **2**, 0, 0, **2**, **2**,**-1**,

0, **2**,**-2**, 0, 0,**-1**,**-2**, **2**,**-1**, **2**,

**-1**, 0, **1**, 0,**-2**, 0, **2**, 0,**-5**,**-1**,

**2**, 0, **1**, **2**, 0, 1, 1, 0, 0,-1,

0, **1**,**-1**, **1**, **2**, **3**, **1**, **2**, **1**, 0,

0, 0, **1**, **1**, 0, **2**, **1**,-**1**, 0, 0,

**1**, 0, 0, 0, **1**,**-1**,**-1**, 0,**-2**, **2**,

0, 0, 0, 0, **3**, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0,**-1**, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

解的欧几里德范数平方为||x||22=223。

# 子集和问题

## 子集和问题研究现状

子集和问题或0-1背包问题是一个著名的NP-完全问题[4]，其经常被用于构建加密体系。近年来，出现了许多针对于背包加密体系的攻击算法，根据背包的特性，其攻击算法可分为两大类。一类是通用算法，该算法适用于任何背包问题。从Schroeppel-Shamir的生日攻击算法[5]，到更加优化的Howgrave-Graham-Joux的通用算法[6]和Becker的改进的背包通用算法[7]。这类算法的优点是，其降低了破解所需花费的时间，是一种确定性的算法，并且可以适用于任何的背包问题。缺点是，因该算法的时间复杂度和空间复杂度是指数时间的，所需要的时间和空间呈指数增长，当背包的维数太大时，无法在有效时间和空间内进行求解。另一类是算法是基于格基规约来进行求解。Coster等人证明了[8]，当背包的密度时，子集和问题的解以极大的概率可以通过寻找某个格上的最短向量来进行求解。此外，Ping等人证明了[9]，低汉明重量、高密度的背包问题可以确定性的规约到格上的最短向量问题。Schnorr等人[10]提出了基于格基规约的算法来求解子集和问题，其构建了基于LLL格基规约和基于BKZ格基规约的求解子集和问题的算法，并对其进行了改进。此后，Schnorr等人[11]又提出了使用随机化的BKZ归约来求解密度接近于1的子集和问题。周娜[1]阐述了，通过合理的构建向量矩阵，使用LLL算法或BKZ算法可以有效的解决低密度的子集和问题。此外，近年来又出现了一些新的解法，例如王蔚[2]等人提出的求解子集和的快速算法。王保仓等人[3]提出的基于联立丢番图逼近的子集和问题的启发式求解算法等。

## Schroeppel-Shamir算法

### Schroeppel-Shamir算法

Schroeppel-Shamir算法[5]是Schroeppel和Shamir于1981年提出的一个背包算法，可以在的内存和在的时间内解决n个元素的一般整数背包问题。该算法是对Horowitz和Sahni的生日算法[12]的改进，使其能够应用于背包问题。生日算法的目是按照如下的等式重写背包:



其中所有的都是0或1。因此，为了解决背包问题，我们先构造集合，其包含前n/2个元素的的所有可能和，是目标值S减去最后一个[n/2]元素的所有可能和得到的集合。通过寻找两个集合之间的碰撞，我们就能找到背包问题的所有解。该算法可以在的时间内通过排序和查找碰撞完全计算如上两个集合。这是最传统的生日攻击方法，从的复杂度可以大大减少到。但是此方法虽然减少了指数级的时间复杂度，但是对于空间复杂度仍然是不可接受的。对此，在文献[5]中，Schroeppel和Shamir指出，为了找到这些碰撞，不需要存储和集合，而使用(基于堆或Adelson-Velsky和Landis树)优先级队列，通过构造四个子集，而不是两个子集，这样就只需要使用的内存，其算法可以表示为如下所示。

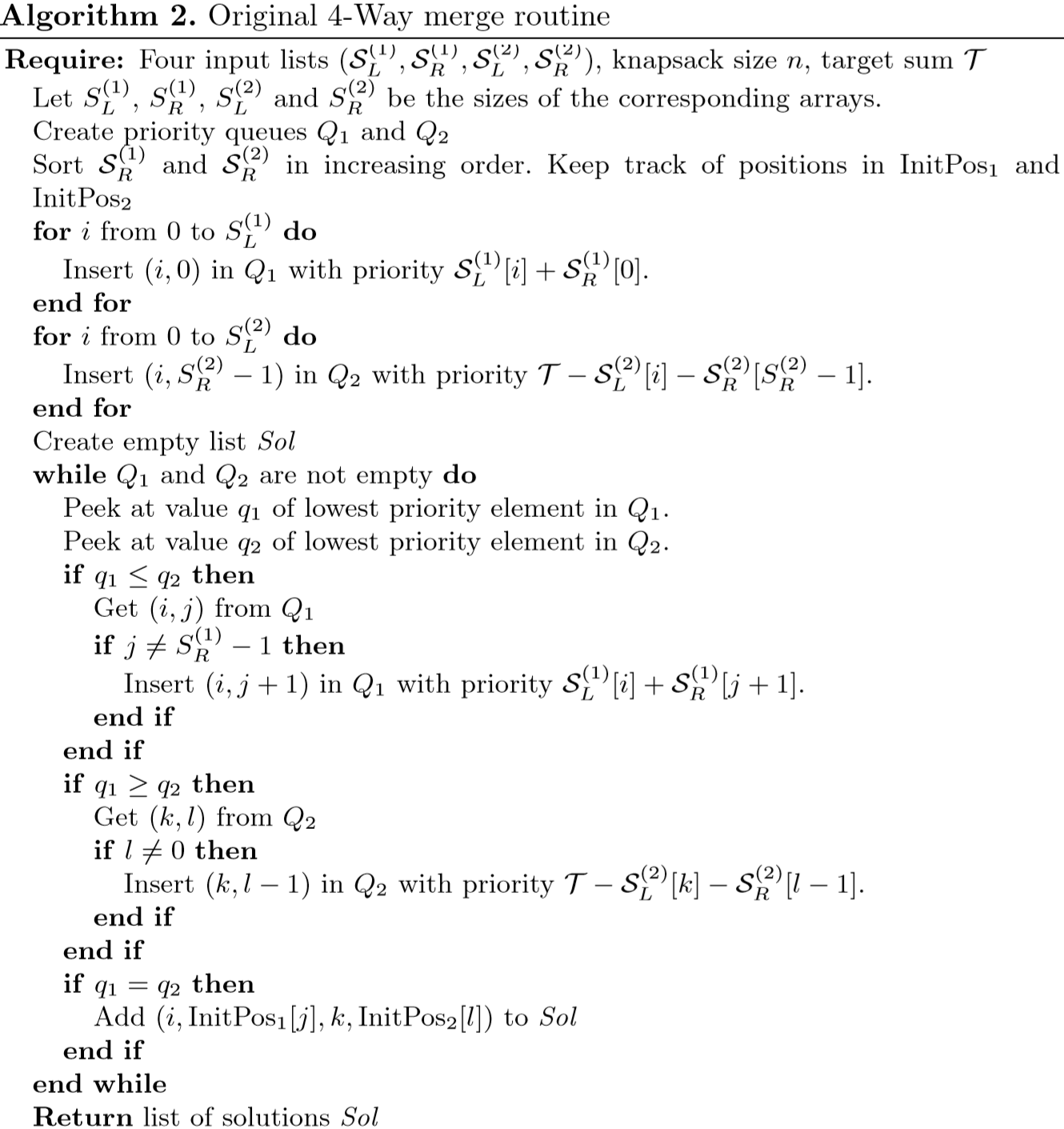


图2-1 Schroeppel-Shamir算法

但是此方法同样存在着不足，因为在实际应用中，由于对大型优先队列的需求，使得Schroeppel-Shamir算法难以实现和优化。实际上，使用大型优先级队列要么会在内存使用中引入额外的因素，要么会有额外的缓存信息，使得内存信息冗余。因此，我们希望完全避免采用优先队列，找到使用少量且简单工具的方法。为了做到这一点，我们使用基于文献[13]的算法，该算法解决了从4个不同的列表中寻找4个元素的问题。但从理论角度来看，对于一些特殊背包需要更多的内存。这个新算法的基本思想是，我们首先选择一个在附近的模数，随后我们把背包表示为如下等式：



在对于第一和第二个数列的时候，我们找到是否存在这样一个满足:



对此我们可以通过循环0到M-1的所有的值实现。首先我们对第二个数列进行排序并全部模M，随后我们循环每一个第一个数列中的值减去然后在第二个列表中寻找是否存在这样的差值。如果找到我们则把它们压入S中，于是我们能够构建一个数列S，由第一个列表和第二个列表中的某值构成且满足其和模掉M后等于。对于列表3和列表4我们采用同样的方法，不过这次是寻找目标S减去数列中的和值后，其是否在列表4中存在。改进的Schoreppel-Shamir算法的具体步骤见图2-2。

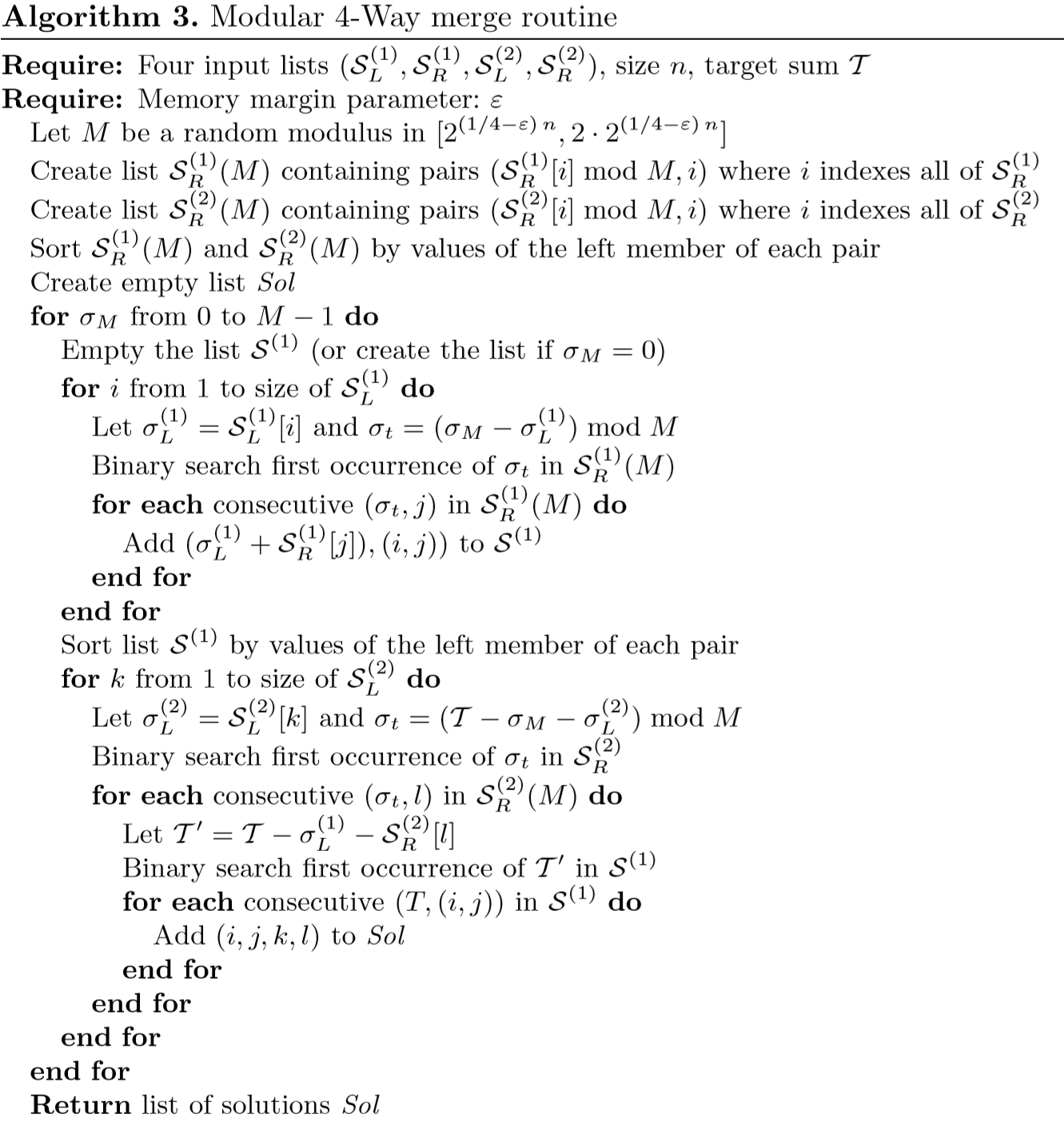


图2-2 改进的Schroeppel-Shamir算法

### 子集和低维（n<=80）问题求解

在求解低维的子集和问题时，我们采用Schoreppel-Shamir算法，但是240的时间复杂度仍然是一个不小的数字，但是由于题目给出了汉明重量这一条件，我们可以通过此条件进行一定的优化，并进一步减少时间和空间复杂度。根据赛题中所给出的汉明重量k，在改进算法构造四个集合的时候，从原本的完全构造四个集合的所有可能，转变为带有限制条件k的四个集合，即只有满足在所给的汉明重量的集合才能被选取出来进行碰撞，并在构造的时候附属上汉明重量这一信息，并在中间从四个集合中选取碰撞的时候，考察当前可能解的汉明重量是否为题目所给出的汉明重量。经过分析该算法适合求解80维及以下的所有整数背包问题，且其算法能忽视集合元素密度带来的影响，确定地在,时间和空间复杂度内完成计算。但当维数达到100时，算法便不再适用，其时间复杂度和空间复杂度难以接受。我们通过计算和实际测验得出内存与维数在Modular-4-way-merge算法下的关系图。

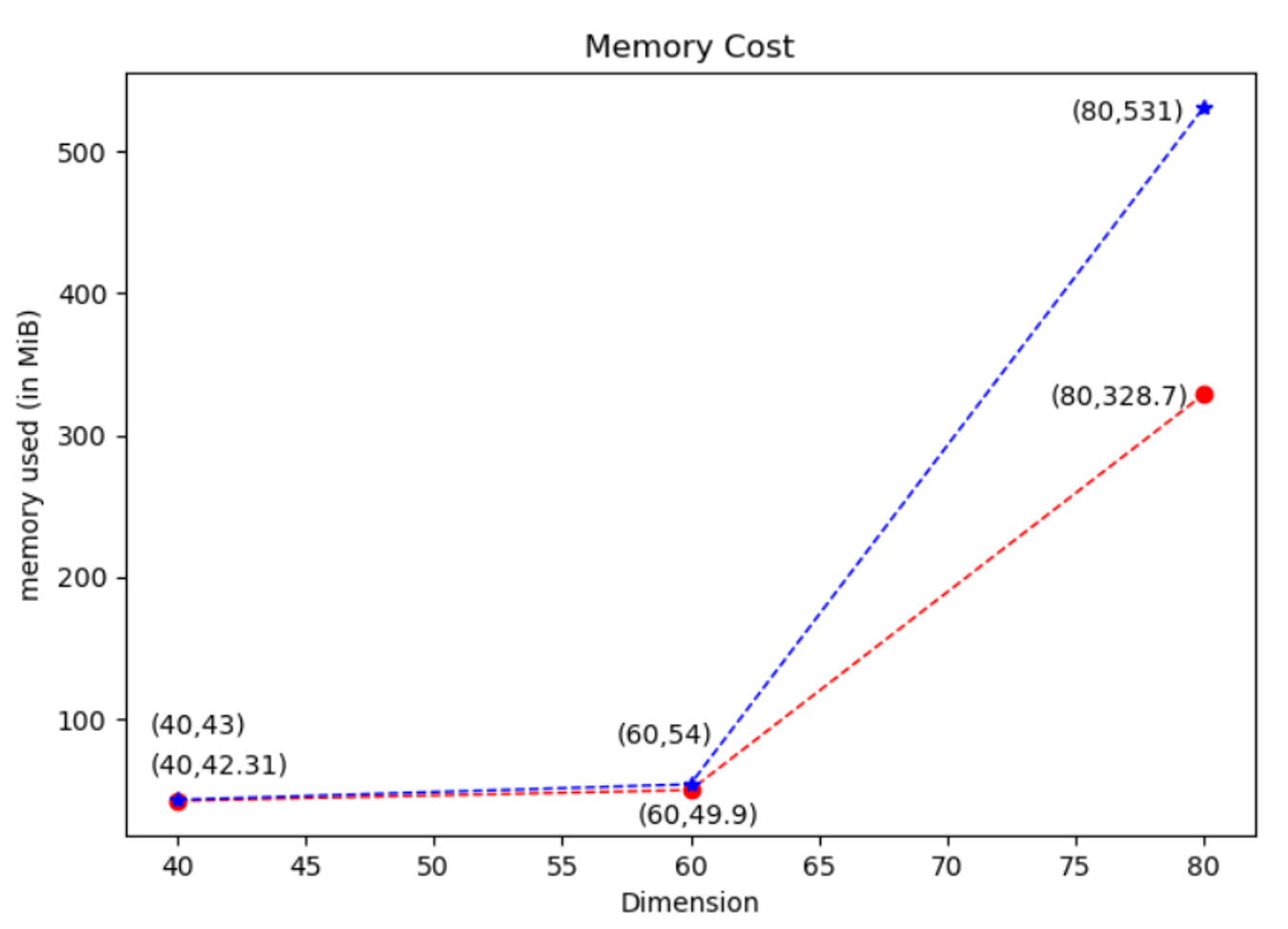


图2-3在Modular-4-way-merge算法下内存与维数的关系

其中蓝色为不考虑汉明重量所需要的内存，红色为考虑汉明重量所需要的内存。可以发现对于此算法，因本次数据的的情况下，在内存方面可以减少约40%左右，根据算法原理，所需要的时间大约也能减少40%。考虑到时间效率问题，我们采用C++来实现该算法，因所给数据均为大整数，我们选择导入GMP库来进行编程。

最后在我们解出的赛题的时间和空间消耗情况如下：对于40维（题目1.1）在42.31MB和 0.1s 内完成了计算；对于60维（题目2.4），在49.9MB的内存和59s 内完成了计算；对于 80维（题目3.1），我们在intel® 至强® E5-2403处理器上，使用20个线程并行计算，在328.7MB的内存和1h 24m 25s时间内完成了计算。具体实现代码见附录“Schroeppel-Shamir改进算法代码”。

## Howgrave-Graham-Joux算法

### Howgrave-Graham-Joux算法

Howgrave-Graham-Joux（HGJ）算法[6]是Howgrave-Graham和Joux于2010年提出的通用背包算法。该算法可以解决通用的背包问题，即形如：



其汉明重量和密度可任意给定，我们假定其汉明重量为。

定义集合为或个背包元素的部分和。则存在一对集合中的元素，其满足S=。实际上，根据S可能的分解，存在许多的这样的配对。当为偶数时，存在这样的分解的数量=，而当为奇数时，其数量=。

HGJ的基本思想是关注于集合，因为通过组合数构造的子集一定在其中会找到我们所需要的解。对于汉明重量为最坏的情况下，我们将原本的集合仍然是分解成四个子集，并分别构造其所有组合情况，因为，故我们每一个子集是一个1/8不平衡背包（已知汉明重量的背包），所以我们每一个子集的大小为，随后我们随机指定这四个集合的子目标值为，那么再通过此条件的约束着能极大的减少时间复杂度从而更快速的求解。随后我们同样Modular-4-way-merge算法进行搜索答案，上一节所提到的Modular-4-way-merge算法也便是HGJ所提到并改进的。同样我们首先在附近选择一个素数M，和一个随机的元素R模M。则有一定的可能性存在一个S的分解，其满足并且。这样的集合的预期大小为：



该指数是假定最坏的时候，即时所得出的。一旦这两个子集和构建好后。我们就去寻找在和之间的一个碰撞，其中属于，属于。接下来，使用排序和查找算法即可在时间内完成搜索。

### 子集和中高维（80<n<=180）问题求解

在求解中高维的子集和问题时，此前的Schroeppel-Shamir算法因空间复杂度较高，普通计算机已无法完整的执行该算法。于是，我们找到了Howgrave-Graham-Joux（以下称为HGJ）算法，其算法内部也使用了Modular-4-way-merge算法。正如上文所述，其通过构造子集然后再进行碰撞，将其时间复杂度降低到了，可惜的是，其算法只对第一轮碰撞有效，因其在开始构建部分子集和时，便已经把值固定了，故这样并不能完全的实现元素全覆盖。如果想要遍历所有情况，那么其总数为，在本次的数据中随着目标值S的不断变大，这个数字已经变的很大从而不可接受，所以HGJ算法能够找出子集和解的概率概率仍然并不高。但是我们发现题目的汉明重量约为n/8，分解到每个子集时实际为n/32不平衡背包，这样的话其总数只有，即使对于160维这样的高维问题，我们估算其空间复杂度是在现在的技术下也是可接受的。通过计算和实际检测在Howgrave-Graham-Joux算法中的运行情况，我们得出了维数和内存之间的关系如下图。

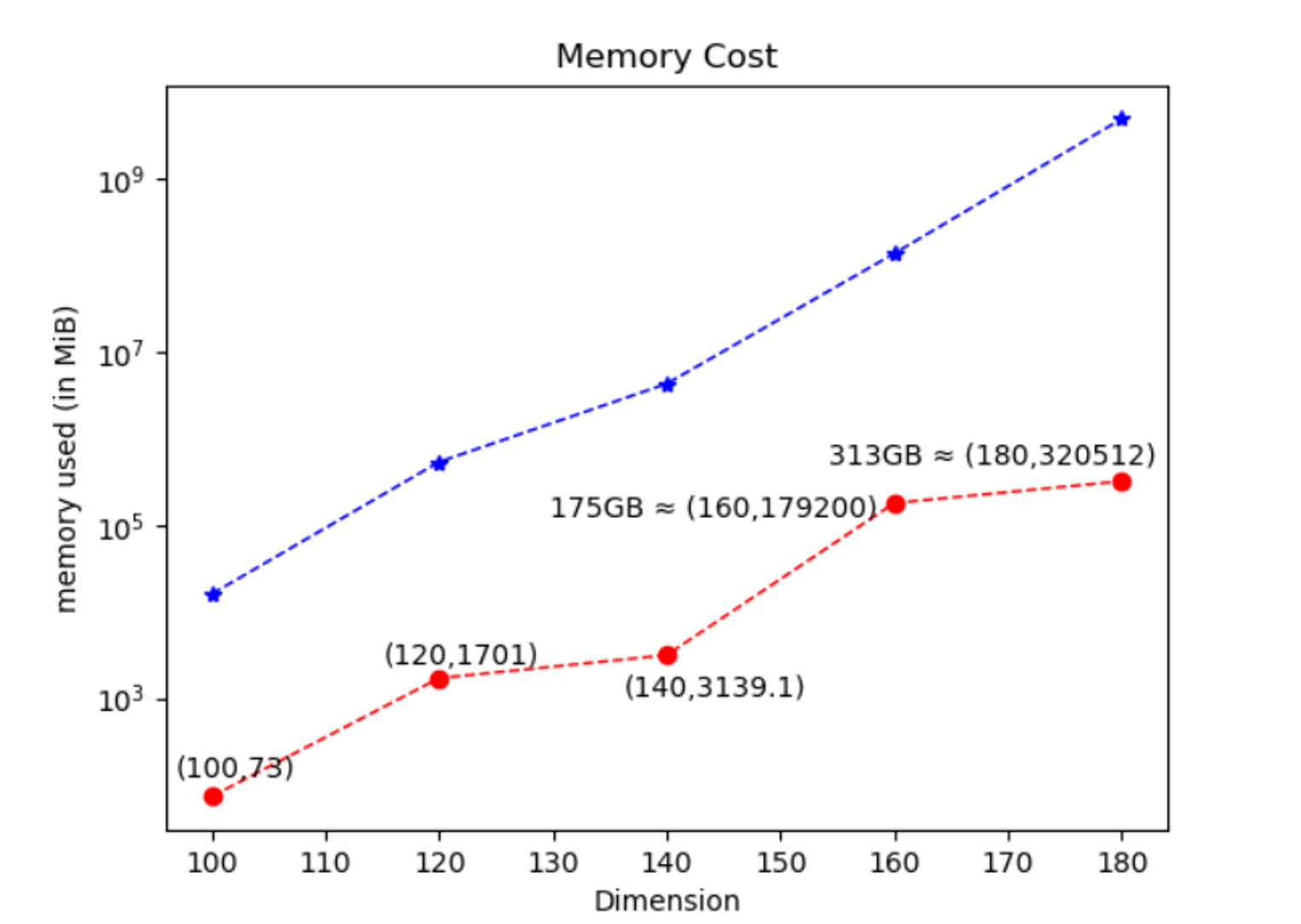


图2-4在Howgrave-Graham-Joux算法下内存与维数的关系

蓝色的线Schroeppel-Shamir在相同的维度下所需要的内存。我们可以明显看出，在纵坐标为指数的情况下对于目前的技术已经无法容纳那么多的信息，从而对于高维问题Schroeppel-Shamir即使带有汉明重量等条件求解也是不可能的。

红色的线为HGJ算法所需要的内存情况。可以从图中知道对于100到140维之间，所需要的内存是较低的，对于普通的笔记本电脑足以胜任。对于160维则需要179200MB的内存。由于算法原理，我们对于k mod 4 == 0的情况是最合适的，与此同时如果 k mod 4 == 2我们称其为最坏的情况，如果k mod 4==1 或者 k mod 4 == 3，则为较合适的其情况。因为我们可以提前假设一个构成目标和S中的元素或者一个不属于目标和S的元素，从而删去或者添加到S中使得汉明重量成为四的倍数，最后成为我们最合适的情况，对于余2的情况只是比较合适情况尝试的次数变多了而已。那么由于160维的k为20，180维的k = 22仅仅比160维的k多2，所以在构建集合的时候我们使其k降为20，那么180维所需要的内存对于160维只是翻倍了，需要320512MB的内存，这对我们完成180维提供了有利的条件。

所以最后我们采取了HGJ算法的思想，通过构建组合数，但是提前不指定目标和S，随后使用Modular-4-way-merge算法进行搜索。发现通过构造组合数去寻找道碰撞的几率极高，程序往往不需要遍历整个空间，此外在构建的时候内存和相比Schroeppel-Shamir有了指数级的减少，并且再构造所有组合数的时候，我们的答案序列同样进行了组合，其数量为。例如当我们在第一个列表选择了一个组合并准备进行检查时，我们仍然存在个答案组合在剩余的列表里面，故这对我们找到碰撞的概率有了巨大的提升。此外对于HGJ算法中，存在比较关键的M参数选择的问题，我们同样做出了讨论和研究，为了方便取样数据和分析，我们选择了维数n=100维的子集和问题。我们通过简单遍历M的取值去观察到了运行时内存和计算出了所需要的计算时间的关系，如图所示。

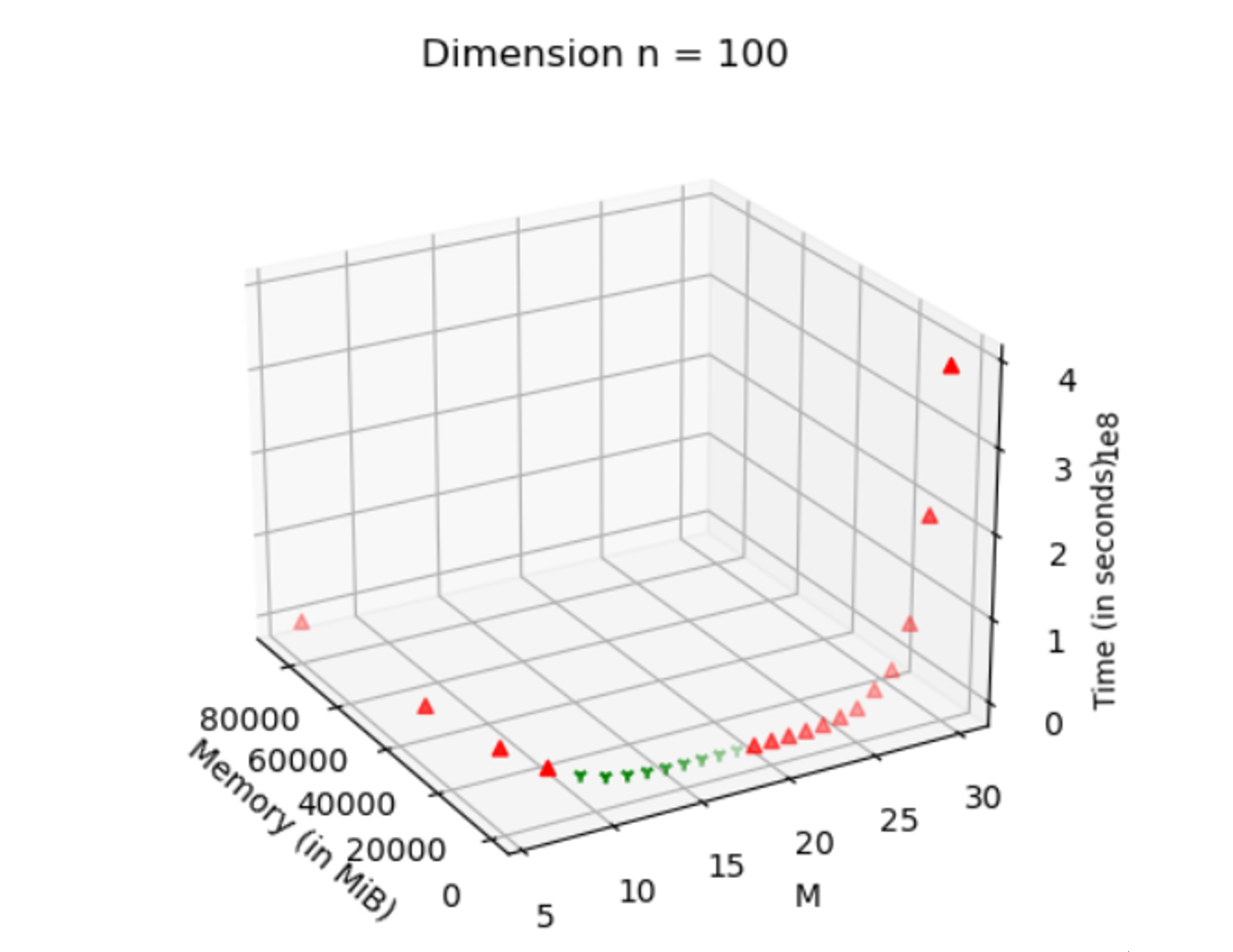


图2-5 M的取值与运行内存和计算时间的关系

图像的走势，我们可以从算法算法的原理得出。在M=230的时候，时间复杂度变得不可接受。究其原因是当模数M变得很大时，单次找到碰撞的概率变低，中间的S集合需要的内存减少，故内存只包含初始构建的四个集合所需要的内存。但是与此同时由于需要遍历0至M-1，所以当遍历次数增加时，时间复杂度也变得异常巨大，从而无法有效求解。在M变低的过程中，可以发现内存走势逐渐变高。通过分析可以得出在单次循环中，由于M变低，发生求和取模后数值相同的概率变大，从而使得中间可能解集集合S变大，从而使得内存消耗变大。在M变低过程中，这对内存影响是非常明显的，在图中可见对于100维，M如果小于210，则所需的内存空间是不可接受的。综上两个方面的考虑，对于某个维度我们存在一个较为合适的M的取值，即为图中的绿色部分。通过规划，对于100维，我们选择了M=214，并使用python语言构建的程序在1s内即可大概率求解。

在循环的过程中，由于我们构建的四个子集的长度较小，单次循序需要的时间复杂度仅需为，即为我们构造的组合总数，这一步耗时较少，但是随着模数M的不断变大，我们需要循环的次数急剧增加，遍历所有情况成为不可能。故我们通过n=100维的情况，完整运行了4次并观察了对于特定排列的的取值对能否求出解的影响，如下所示。

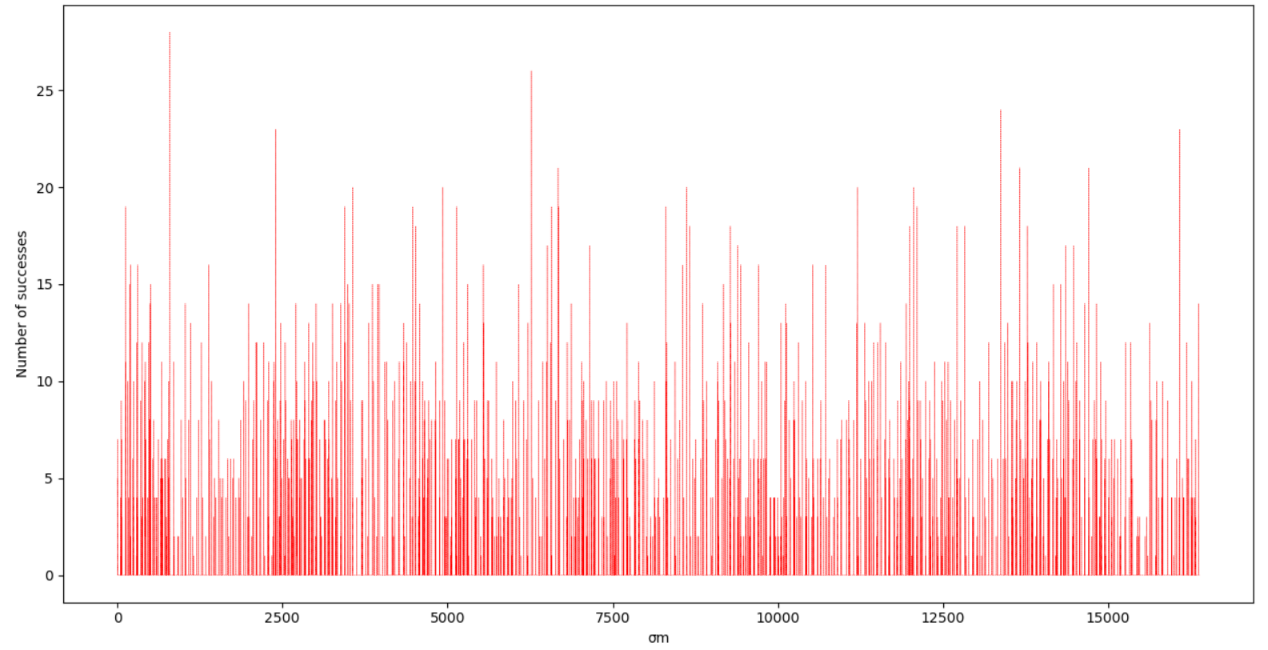


图2-6 不同取值下的成功次数

在算法运行中，如果四个子集排列确定了，那么如果该数据有解，那么一定存在某个在那一次循环过程中能够找到解。例如当n=100时，能求出解的的分布呈现一种随机的排列，并且同一能求出的解通常不止一个，有时甚至在同一次循环中找到7个答案的不同排列。但是由于数据的随机性，我们不能预测哪种排列对应的能够求解，故我们使用并行计算将M的区间分为多个小区间，每一个线程负责一个小区间，并且从线程中随机取值，从而增加找到合适的的概率。需要说明的是，如果我们构造的四个子集其中某个集合不包含我们的答案的组合的话，那么该算法是无法有效求解的，此时我们只需要重新排列即可。此外如果对于k mod 4不等于0的情况，我们首先将k降为最近模4为0的数，我们同样可以采用并行技术，设置每一个cpu的目标和为不同的和，同时共享相同的四个子集，那么即可同时运行且不需要其他内存。最后我们成功解决100-180维的中高维的子集和问题。

赛题的时间和空间消耗情况如下：对于100维（题目4.1），我们选择了M= 214，它在intel® Xeon® E5-240（以下均使用此配置）上使用了小于73MB的内存和0.18s时间内完成了计算；对于120 维（题目5.4），我们选择了 M = 218，它在10个线程中在1496MB的内存和3h 10m 25s 时间内完成了计算；对于140维（题目6.4），我们选择了 M = 220，它在20个线程中在3584MB内存和 3h 54m 21s时间内完成了计算；对于 160维（题目7.4）, 我们选择了M= 229，它在15个线程中在191728MB内存和1h 4m 48s时间内完成了计算，对于180维（题目8.1），我们选择M=231，他在48线程中，在332432MB内存和82h 44m 44s内完成了计算。

## 格基规约算法

### 格的定义

定义格为：格是实数域上 m 维向量空间中的一个离散子群，它是由向量空间中 n 个线性独立向量的整数倍的线性组合生成的：



集合 L 中以为基的格，n 为格的维数。若 m 等于 n ，可得 L 为满秩格。

格的基可由任意一组可以生成格的线性无关的向量组成，一个格可以包含很多组基，而这很多组基只能生成唯一的格。不同的基之间也可以通过向量的线性组合相互转换。若每个向量的坐标都是整数，则称之为整数格。

一个格中可包含不同的基，并且基之间可以相互表示，同一个格中不同基的行列式相等，因此，在不同基之间寻找一个包含最短向量的基的过程就称为格基规约。

### 格中的困难问题

格中存在 NP 困难问题，最基本的两个就是：在一个格中寻找最短非零向（Nonzero-SVP）,使得它的 Euclidean 范数最小（Shortest Vector Problem）；在一个格中寻找与指定的一个非格中的向量距离最近的向量，使得两个向量相减的 Euclidean范数最小（Closest Vector Problem）。通常表示欧几里德范数，表示向量的内积，它们之间存在的关系。表示格中最短非零向量的长度。 n 维格的单位体积是一个常量，其数值等于格的行列式。因此，从几何角度可知 n 维格的行列式：表示向量与前i 个向量组成的超平面所组成的夹角。显而易见，夹角越大，的值越小，所对应基向量的正交性就越好，基向量的长度越短。

格在实数域 m 维向量空间中的一个基础域的集合可定义为:



格L的行列式或体积可定义为：



Hadamard 不等式：假设L是一个格，对于它的任意一个基和基础域 F 来说，存在。若基向量越接近正交，上式就越接近等式。

Hadamard比率：用来描述一组向量的垂直正交程度，它的范围是(0,1].向量之间越接近垂直正交，Hadamard比率的值就越接近 1。

### LLL算法

1982 年，Lenstra A. K.，Lenstra H. W. 和Lovász L. [14]提出了经典的 LLL 格基规约算法。之后，Lagarias和Odlyzko[15]表明把特定的背包问题（密度为 0.64）转换成格中的最短向量问题（SVP）。紧接着，他们描述了使用 LLL 格基规约算法来求解该问题的思想。十九世纪二十年代初，Lagarias-Odlyzko 的方法被Coster等人[8]进一步将能被解决的背包问题的背包密度提高到 0.94，利用低密度子集和攻击就能在格中能够寻找到最短非零向量，那么算法的一个简单修改将解决几乎密度小于 0.94 的所有问题。

我们使用的格基规约算法来源于Shoup上的NTL库中的函数。我们使用库中的chnorr-Euchner基规约算法，使用了其中的LLL算法。在更新文档中，作者说明了这里的LLL不是最原始的LLL，而是LLL衍生出的deep LLL算法。在deep LLL算法中，当条件



成立时，继续下面两个相邻向量之间的比较; 如果不成立，则需要将当前向量与更前面的一个向量进行比较，直到时为止。在文献[6]中，实际使用的是的改进算法()，这样可以避免在改变格基时的正确性。deepLLL相较于一般的LLL算法在交换环节的次数更少，所以速度更加快。

### BKZ算法

常见格基规约算法除了的LLL算法外，还有BKZ(Block Korkin Zolotarev reduction)算法。

BKZ(Base Korkin Zolotarev)是分块的deepLLL算法。它将标准LLL算法中的交换步骤用一个分组约减步骤代替，该方法得到的约减基被成为KZ约减基。格中一组基L()可以被称为KZ约减基应该当且仅当满足下面三条性质:

（1）是格L中的最短非零向量。

（2）对于i=2,3…,n的向量使得为中最短非零向量。

（3）对于所有的，都有。

通常，KZ约减基的第一个向量就是SVP问题的一个解。

其相比与LLL算法有了更近一步的优化，主要思想就是对向量分块来进行规约，当分块越小时，其规约速度越快，但其规约后格基的规约效果较差。我们可以通过调整其分块大小来规约出较高质量的基向量，当期分块大小较大时，将会有很大的概率得到只包含0和1的基向量，此时的基向量便是我们子集和问题的解向量。

### 子集和高维（n>180）问题求解

当面对高维的子集和问题时，之前的Schroeppel-Shamir算法与Howgrave-Graham-Joux算法因时间复杂度和空间复杂度过高，无法进行求解。因此，我们采用格基规约的方法来尝试求解。

使用LLL算法解题时，需要先使用题目中的信息来构造出一个矩阵，将矩阵的每一行堪称一个向量，一共n+1个向量，构成格的一个基。参考文献[1]中构建矩阵的方式，矩阵总共n+1行，主对角线上除最后一个元素为S外，其余全部为2。最后一列除最后一个元素为S外，其余为。最后一行除最后一个元素为S外，其余元素均为S。



将矩阵构造完成之后，如果将矩阵最后一列的最后一行乘以Constan倍，能更有希望找到格中最短向量。在使用该算法时，可以使用Hadamard比率来评价一个矩阵是否是已经足够被规约，Hadamard取值范围在(0,1]之间，当该值越接近于1，说明规约出的格基越优质。找到其中最短的向量,



其中t为最短向量，而为所需要求的0,1解。

由于BKZ的规约效果通常比LLL的规约效果好，因此我们在解题时，主要是使用的BKZ算法来进行求解。通过用背包元素构建恰当的矩阵来进行规约，调整BKZ的分块大小来保证能规约出01基向量的同时保证时间小。在实际运行中，我们的格基采用文献[10]中的如下方式构建，其中Δ定义为大于汉明重量k的整数。



如此一来我们得到了n+1个线性无关向量，如果我们把解m表示为m[n-1], 0, 0)，那么我们一定有。

我们采用文献[6]中的子集合问题的算法如图2-4所示

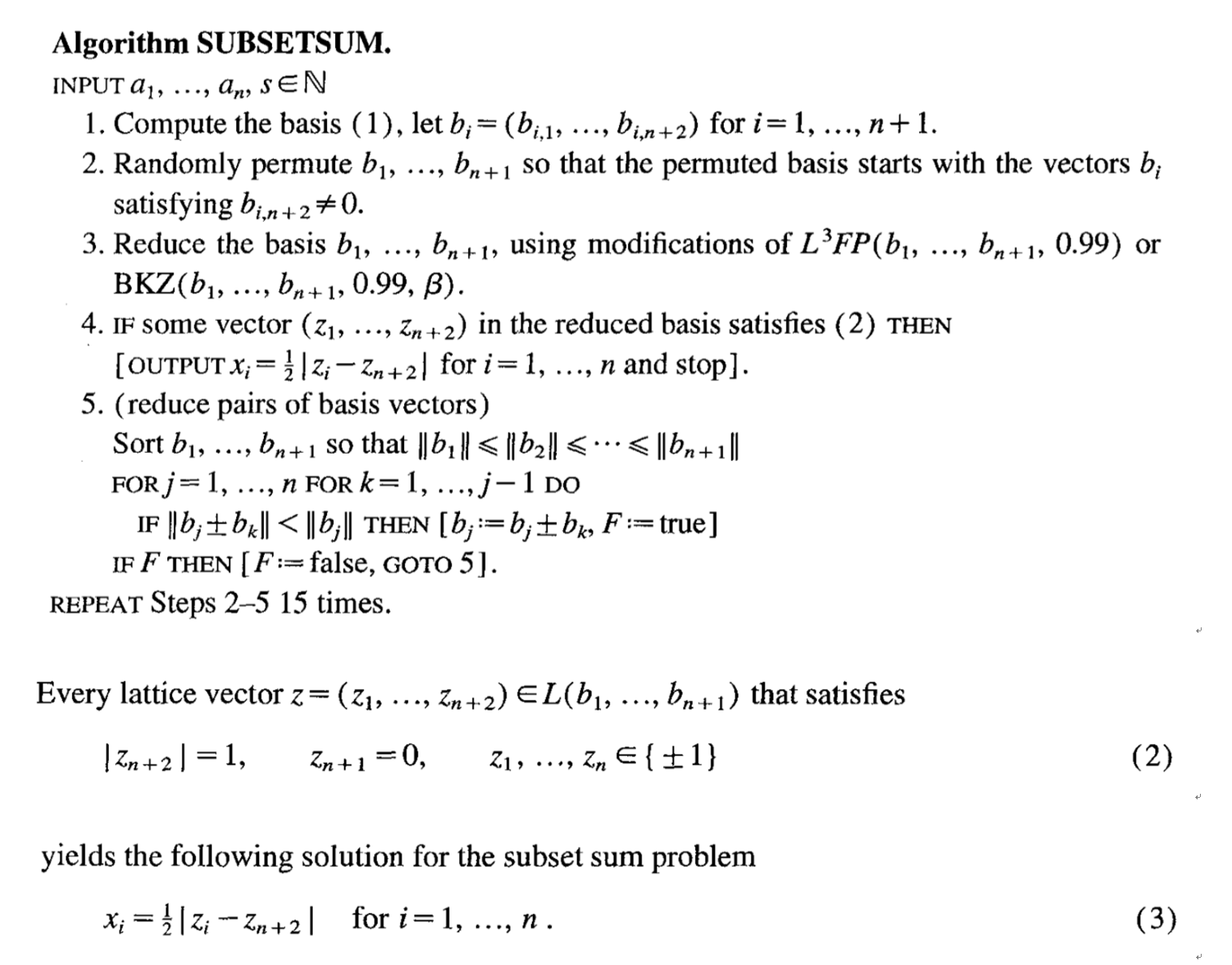


图2-7 重复子集合算法

该算法首先构造格子并使其随机排列，在规约之后进行线性组合，如果存在线性组合使得组合的范数小于之前的范数，那么我们将其另赋新值。该算法通过不断的组合和规约，能够不断逼近当前格的最短向量，并最终求解，但遗憾的是，当我们求解高维的子集和问题时，比赛即将结束，对于140维即以下，我们成功规约出01整数解，但对于200维（赛题9.4）我们进行了分区大小为24的BKZ规约，在可忽略的内存内求出范数平方为233的近似解。

# 结论

本文基于Schroeppel-Shamir（Schroeppel and Shamir）改进算法、HGJ（Howgrave-Graham-Joux）改进算法和格基规约算法，成功解决了180维以内的子集和问题。通过实验发现维数较低Schroeppel-Shamir算法可以有效且快速的完成计算,如果知道汉明重量已知，改进的HGJ算法更为高效。这两种方法可以有效忽略背包元素密度带了的影响，从而可以解决通用的整数背包，但是对于k较大或者维数较高的情况会显得力不从心。故对于一般高维问题，目前最有效的方法是把该问题转为SVP问题，然后使用规约算法进行求解。该方法对于维度低，密度为0.8的数据仍然能够快速求解，对于其他密度存在一定的概率性。对于180至200的高维问题，需要的时间复杂度依然较大，且由于格基构建时参数的不确定性，仍然需要较多的时间去尝试。所以如何构建一个时间复杂度和空间复杂度小，能够解决高维的子集和问题的算法，仍值得我们继续进行研究。

# 参考文献

[1] 周娜. 格基规约相关算法的研究[D]. 深圳大学, 2017.

[2] 王蔚, 邱伟星. 求解子集和问题的快速算法[J]. 南京邮电大学学报: 自然科学版, 2012, 32(6): 92-95.

[3] 王保仓, 卢珂. 基于联立丢番图逼近的子集和问题启发式求解算法[J]. 密码学报, 2017, 4(5): 498-505.

[4] 周福才，徐剑. 格理论与密码学. 北京:科学出版社.

[5] van Emde Boas P. Another NP-complete problem and the complexity of computing short vectors in a lattice[J]. Tecnical Report, Department of Mathmatics, University of Amsterdam, 1981.

[6] Schroeppel R, Shamir A. A T=O(2^n/2), S=O(2^n/4) algorithm for certain NP-complete problems[J]. SIAM journal on Computing, 1981, 10(3): 456-464.

[7] Howgrave-Graham N, Joux A. New generic algorithms for hard knapsacks[C]//Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2010: 235-256.

[8] Becker A., Coron J., Joux A., Improved generic algorithms for hard knapsacks, in Advances in Cryptology-EUROCRYPT 2011, Tallinn, Estonia, May 15-19, 2011 (LNCS, 6632), pp. 364-385.

[9] Coster M.J., Joux A., LaMacchia B.A., et al., Improved low-density subset sum algorithms, Computational Complexity, 1992, 2, (2), pp. 111-128.

[10] Yuan Ping, Baocang Wang, Shengli Tian, Yuehua Yang, Genyuan Du, Deterministic lattice reduction on knapsacks with collision-free properties, IET Information Security, 2018, 12(4), pp. 375-380.

[11] Schnorr C P, Euchner M. Lattice basis reduction: Improved practical algorithms and solving subset sum problems[J]. Mathematical programming, 1994, 66(1-3): 181-199.

[12] Schnorr C P, Shevchenko T. Solving Subset Sum Problems of Densioty close to 1 by" randomized" BKZ-reduction[J]. IACR Cryptol. ePrint Arch., 2012, 2012: 620.

[13] Horowitz E, Sahni S. Computing partitions with applications to the knapsack problem[J]. Journal of the ACM (JACM), 1974, 21(2): 277-292.

[14] Chose P, Joux A, Mitton M. Fast correlation attacks: An algorithmic point of view[C]//International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2002: 209-221.

[15] Lenstra A K, Lenstra H W, Lovász L. Factoring polynomials with rational coefficients[J]. Mathematische annalen, 1982, 261(ARTICLE): 515-534.

[16] Lagarias J C, Odlyzko A M. Solving low-density subset sum problems[J]. Journal of the ACM (JACM), 1985, 32(1): 229-246.

# 谢辞

在本文的撰写过程中，周洁老师作为我们的指导老师，她治学严谨，学识渊博，视野广阔，为我们营造了一种良好的学术氛围。正是由于她在百忙之中多次为我们解答疑惑，并为本文的撰写提供了许多中肯而且宝贵的意见，本文才得以成型。另外在实际解题过程中，中国科学院软件所给与了我们宝贵的计算资源，因为有了他们的帮助我们的想法才能得以实现并最终解决问题。

在此向中国科学院软件所的同学表以衷心的谢意！此外向周洁老师的无可挑剔的敬业精神、严谨认真的治学态度、深厚的专业修养和平易近人的待人方式表示深深的敬意！

# 附录

## Schroeppel-Shamir改进算法代码

1. #include <iostream>
2. #include <gmp.h>
3. #include <vector>
4. #include <random>
5. #include <string>
6. #include <ctime>
7. #include <algorithm>
8. #include <time.h>
9. #include <thread>
10. #include <stdlib.h>
11. #include <fstream>
12. #include <unistd.h>
13. using namespace std;
14. #define PBSTR "||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||"
15. #define PBWIDTH 60
16. typedef long long int ll;
17. typedef unsigned long int unint;
18. const unint N ;
19. const unint Hamingweight;
20. const int threadNumber;
21. const char \*target\_str;
22. bool result = false;
23. struct QuarterKnapsack
24. {
25. mpz\_t sum;
26. unint weight;
27. };
28. struct ModuleKnapsck
29. {
30. unint sum;
31. unint index;
32. };
33. struct IndexPair
34. {
35. unint i;
36. unint j;
37. };
38. class Solution
39. {
40. public:
41. mpz\_t sum;
42. unint sl1\_index;
43. unint sr1\_index;
44. Solution() { mpz\_init(this->sum); };
45. ~Solution();
46. };
47. Solution::~Solution()
48. {
49. // mpz\_clear(this->sum);
50. // mpz\_realloc2(this->sum, 0);
51. }
52. vector<QuarterKnapsack> init(int l, int r, vector<mpz\_t \*> elements)
53. {
54. unint last, lent = r - l + 1;
55. vector<QuarterKnapsack> slist;
56. for (ll i = 0; i < pow(2, lent); i++)
57. {
58. unint count = 0, value = i, HMW = 0;
59. mpz\_t sum;
60. mpz\_init(sum);
61. while (value > 0)
62. {
63. last = value & 0x01;
64. if (last == 1)
65. {
66. mpz\_add(sum, sum, \*elements[r - count]);
67. HMW++;
68. }
69. count++;
70. value = value >> 1;
71. }
72. if (HMW > Hamingweight)
73. {
74. continue;
75. }
76. QuarterKnapsack oneKapk;
77. mpz\_init(oneKapk.sum);
78. mpz\_set(oneKapk.sum, sum);
79. oneKapk.weight = HMW;
80. slist.push\_back(oneKapk);
81. mpz\_clear(sum);
82. }
83. return slist;
84. }
85. vector<unint> binarySearch(vector<ModuleKnapsck> &slist, unint x)
86. {
87. ll l = 0, r = slist.size() - 1, mid;
88. vector<unint> ans;
89. while (l <= r)
90. {
91. mid = (l + r) / 2;
92. if (slist[mid].sum == x)
93. {
94. for (ll i = mid - 1; i > -1; i--)
95. {
96. if (slist[i].sum == x)
97. {
98. ans.push\_back(slist[i].index);
99. }
100. else
101. {
102. break;
103. }
104. }
105. ans.push\_back(slist[mid].index);
106. for (ll i = mid + 1; i < slist.size(); i++)
107. {
108. if (slist[i].sum == x)
109. {
110. ans.push\_back(slist[i].index);
111. }
112. else
113. {
114. break;
115. }
116. }
117. return ans;
118. }
119. else if (slist[mid].sum > x)
120. {
121. r = mid - 1;
122. }
123. else
124. {
125. l = mid + 1;
126. }
127. }
128. return ans;
129. }
130. vector<IndexPair> binarySearchSol(vector<Solution> &slist, mpz\_t x)
131. {
132. int l = 0, r = slist.size() - 1, mid;
133. vector<IndexPair> ans;
134. IndexPair solpair;
135. while (l <= r)
136. {
137. mid = (l + r) / 2;
138. if (mpz\_cmp(slist[mid].sum, x) == 0)
139. {
140. for (ll i = mid - 1; i > -1; i--)
141. {
142. if (slist[i].sum == x)
143. {
144. solpair.i = slist[i].sl1\_index;
145. solpair.j = slist[i].sr1\_index;
146. ans.push\_back(solpair);
147. }
148. else
149. {
150. break;
151. }
152. }
153. solpair.i = slist[mid].sl1\_index;
154. solpair.j = slist[mid].sr1\_index;
155. ans.push\_back(solpair);
156. for (ll i = mid + 1; i < slist.size(); i++)
157. {
158. if (slist[i].sum == x)
159. {
160. solpair.i = slist[i].sl1\_index;
161. solpair.j = slist[i].sr1\_index;
162. ans.push\_back(solpair);
163. }
164. else
165. {
166. break;
167. }
168. }
169. return ans;
170. }
171. else if (mpz\_cmp(slist[mid].sum, x) == 1)
172. {
173. r = mid - 1;
174. }
175. else
176. {
177. l = mid + 1;
178. }
179. }
180. }
181. bool cmp\_mod(ModuleKnapsck x, ModuleKnapsck y)
182. {
183. return x.sum < y.sum;
184. }
185. bool cmp\_sol(Solution x, Solution y)
186. {
187. return (mpz\_cmp(x.sum, y.sum) < 0);
188. }
189. void algorithm3(vector<QuarterKnapsack> &sl1, vector<QuarterKnapsack> &sr1, vector<QuarterKnapsack> &sl2,
190. vector<QuarterKnapsack> &sr2, \
191. vector<ModuleKnapsck> &sr1m, vector<ModuleKnapsck> &sr2m,
192. mpz\_t target,ll M, ll Ml, ll Mr, \
193. int id, ofstream &record)
194. {
195. clock\_t start, end;
196. start = clock();
197. cout << "algorm3 started" << endl;
198. ll step = 0;
199. // cout << Ml << ' ' << Mr << endl;
200. for (unint m = Ml; m <= Mr; m++)
201. {
202. step++;
203. if (step % 10 == 0)
204. {
205. double percentage = ((m - Ml) \* 1.0) / (Mr - Ml);
206. int val = (int)(percentage \* 100);
208. cout << '\n'
209. << "thread: " << id << ' ' << val << "% " << m << ' ' << Ml << '/' << Mr << endl;
210. if (step % 500 == 0){
211. record << '\n'
212. << "thread: " << id << ' ' << val << "% " << m << ' ' << Ml << '/' << Mr << endl;
213. }
215. }
216. vector<Solution> s;
217. for (unint sl1\_index = 0; sl1\_index < sl1.size(); sl1\_index++)
218. {
219. mpz\_t tmp;
220. mpz\_init(tmp);
221. mpz\_ui\_sub(tmp, m, sl1[sl1\_index].sum);
222. vector<unint> return\_list = binarySearch(sr1m, mpz\_fdiv\_ui(tmp, M));
223. mpz\_clear(tmp);
224. for (int i = 0; i < return\_list.size(); i++)
225. {
226. unint sr1\_index = return\_list[i];
227. unint HMW = sl1[sl1\_index].weight + sr1[sr1\_index].weight;
228. if (HMW <= Hamingweight)
229. {
230. mpz\_t sum;
231. mpz\_init(sum);
232. mpz\_add(sum, sl1[sl1\_index].sum, sr1[sr1\_index].sum);
233. Solution sol;
234. mpz\_set(sol.sum, sum);
235. sol.sl1\_index = sl1\_index;
236. sol.sr1\_index = sr1\_index;
237. s.push\_back(sol);
238. mpz\_clear(sum);
239. }
240. }
241. }
242. // gmp\_printf("%Zd\n", s[i].sum);
243. sort(s.begin(), s.end(), cmp\_sol);
244. for (int sl2\_index = 0; sl2\_index < sl2.size(); sl2\_index++)
245. {
246. mpz\_t tmp;
247. mpz\_init(tmp);
248. mpz\_sub\_ui(tmp, target, m);
249. mpz\_sub(tmp, tmp, sl2[sl2\_index].sum);
250. vector<unint> return\_list = binarySearch(sr2m, mpz\_fdiv\_ui(tmp, M));
251. mpz\_clear(tmp);
252. for (int i = 0; i < return\_list.size(); i++)
253. {
254. unint sr2\_index = return\_list[i];
255. unint HMW = sl2[sl2\_index].weight + sr2[sr2\_index].weight;
256. if (HMW > Hamingweight)
257. {
258. continue;
259. }
260. mpz\_t tt;
261. mpz\_init(tt);
262. mpz\_sub(tt, target, sl2[sl2\_index].sum);
263. mpz\_sub(tt, tt, sr2[sr2\_index].sum);
264. vector<IndexPair> return\_sol = binarySearchSol(s, tt);
265. mpz\_clear(tt);
266. for (int sol\_index = 0; sol\_index < return\_sol.size(); sol\_index++)
267. {
268. FILE \*file;
269. file = fopen("./outT", "w");
270. end = clock();
271. cout << "\rsolved in " << (double)(end - start) / CLOCKS\_PER\_SEC << endl;
272. mpz\_out\_str(file, 10, sl1[return\_sol[sol\_index].i].sum), fputc('\n', file);
273. mpz\_out\_str(file, 10, sr1[return\_sol[sol\_index].j].sum), fputc('\n', file);
274. mpz\_out\_str(file, 10, sl2[sl2\_index].sum), fputc('\n', file);
275. mpz\_out\_str(file, 10, sr2[sr2\_index].sum);
276. fclose(file);
277. result = true;
278. return;
279. }
280. }
281. }
282. for (int i = 0; i < s.size(); i++)
283. mpz\_clear(s[i].sum);
284. s.clear();
285. }
286. }
287. void shamir(vector<mpz\_t \*> elements, mpz\_t target)
288. {
289. int quarter = elements.size() / 4;
290. cout << "qater :" << quarter << endl;
291. vector<QuarterKnapsack> sl1 = init(0, quarter - 1, elements);
292. cout << "int 1 " << sl1.size() << endl;
293. vector<QuarterKnapsack> sr1 = init(quarter, 2 \* quarter - 1, elements);
294. cout << "int 2 " << sr1.size() << endl;
295. vector<QuarterKnapsack> sl2 = init(2 \* quarter, 3 \* quarter - 1, elements);
296. cout << "int 3 " << sl2.size() << endl;
297. vector<QuarterKnapsack> sr2 = init(3 \* quarter, elements.size() - 1, elements);
298. cout << "int 4 " << sr2.size() << endl;
299. double ll = pow(2, N / 4.0), rr = pow(2, (N / 4.0) + 1);
300. default\_random\_engine e(time(0));
301. uniform\_real\_distribution<double> u(ll, rr);
302. int M = (int)u(e);
303. cout << "Mod: " << M << endl;
304. cout << "init module" << endl;
305. vector<ModuleKnapsck> sr1m;
306. for (int i = 0; i < sr1.size(); i++)
307. {
308. ModuleKnapsck \_t;
309. \_t.sum = mpz\_fdiv\_ui(sr1[i].sum, M), \_t.index = i;
310. sr1m.push\_back(\_t);
311. }
312. vector<ModuleKnapsck> sr2m;
313. for (int i = 0; i < sr2.size(); i++)
314. {
315. ModuleKnapsck \_t;
316. \_t.sum = mpz\_fdiv\_ui(sr2[i].sum, M), \_t.index = i;
317. sr2m.push\_back(\_t);
318. }
319. cout << "sorting" << endl;
320. sort(sr1m.begin(), sr1m.end(), cmp\_mod);
321. sort(sr2m.begin(), sr2m.end(), cmp\_mod);
322. cout << "sort completed" << endl;
323. int ml = 0, mr = M / threadNumber;
324. // algorithm3(sl1, sr1, sl2, sr2, sr1m, sr2m, target, M, 0, M - 1, file);
325. // thread t1(algorithm3, ref(sl1), ref(sr1), ref(sl2), ref(sr2), ref(sr1m), ref(sr2m), target, M, M /2 + 1, M-1, file);
326. ofstream record;
327. record.open("./record");
328. for (int i = 0; i < threadNumber; i++)
329. {
330. thread t(algorithm3, ref(sl1), ref(sr1), ref(sl2), ref(sr2), ref(sr1m), ref(sr2m), target, M, ml, mr, i + 1, ref(record));
331. t.detach();
332. ml = mr + 1;
333. mr += M / threadNumber + 1;
334. if (abs(M - mr) < threadNumber)
335. {
336. mr = M - 1;
337. }
338. }
339. while (result == false)
340. {
341. sleep(5);
342. /\* code \*/
343. }
344. }
345. int main()
346. {
347. FILE \*fp;
348. char StrLine[1024];
349. if ((fp = fopen("./data100", "r")) == NULL)
350. {
351. cout << ("[-]Error, can't open file") << endl;
352. return -1;
353. }
354. vector<mpz\_t \*> elements;
355. mpz\_t arr[N];
356. for (int i = 0; i < N; i++)
357. {
358. mpz\_init(arr[i]);
359. mpz\_set\_str(arr[i], fgets(StrLine, 1024, fp), 10);
360. elements.push\_back(arr + i);
361. }
362. fclose(fp);
363. mpz\_t target;
364. mpz\_init(target);
365. mpz\_set\_str(target, target\_str, 10);
366. shamir(elements, target);
367. mpz\_clear(target);
368. for (int i = 0; i < N; i++){
369. mpz\_clear(arr[i]);
370. }
371. return 0;
372. }

## Howgrave-Graham-Joux改进算法代码

1. #include <iostream>
2. #include <gmp.h>
3. #include <vector>
4. #include <random>
5. #include <string>
6. #include <ctime>
7. #include <algorithm>
8. #include <time.h>
9. #include <thread>
10. #include <stdlib.h>
11. #include <fstream>
12. #include <set>
13. #include <unistd.h>
14. using namespace std;
15. #define PBSTR "||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||"
16. #define PBWIDTH 60
17. typedef long long int ll;
18. typedef unsigned long int unint;
19. const unint N = 140;
20. const unint Hamingweight = 16;
21. bool result = false;
22. struct QuarterKnapsack
23. {
24. mpz\_t sum;
25. vector<int> index;
26. };
27. struct ModuleKnapsck
28. {
29. unint sum;
30. unint index;
31. };
32. struct IndexPair
33. {
34. unint i;
35. unint j;
36. };
37. class Solution
38. {
39. public:
40. mpz\_t sum;
41. unint sl1\_index;
42. unint sr1\_index;
43. Solution() { mpz\_init(this->sum); };
44. ~Solution();
45. };
46. Solution::~Solution()
47. {
48. // mpz\_clear(this->sum);
49. // mpz\_realloc2(this->sum, 0);
50. }
51. vector<unint> binarySearch(vector<ModuleKnapsck> &slist, unint x)
52. {
53. ll l = 0, r = slist.size() - 1, mid;
54. vector<unint> ans;
55. while (l <= r)
56. {
57. mid = (l + r) / 2;
58. if (slist[mid].sum == x)
59. {
60. for (ll i = mid - 1; i > -1; i--)
61. {
62. if (slist[i].sum == x)
63. {
64. ans.push\_back(slist[i].index);
65. }
66. else
67. {
68. break;
69. }
70. }
71. ans.push\_back(slist[mid].index);
72. for (ll i = mid + 1; i < slist.size(); i++)
73. {
74. if (slist[i].sum == x)
75. {
76. ans.push\_back(slist[i].index);
77. }
78. else
79. {
80. break;
81. }
82. }
83. return ans;
84. }
85. else if (slist[mid].sum > x)
86. {
87. r = mid - 1;
88. }
89. else
90. {
91. l = mid + 1;
92. }
93. }
94. return ans;
95. }
96. vector<IndexPair> binarySearchSol(vector<Solution> &slist, mpz\_t x)
97. {
98. int l = 0, r = slist.size() - 1, mid;
99. vector<IndexPair> ans;
100. IndexPair solpair;
101. while (l <= r)
102. {
103. mid = (l + r) / 2;
104. if (mpz\_cmp(slist[mid].sum, x) == 0)
105. {
106. for (ll i = mid - 1; i > -1; i--)
107. {
108. if (slist[i].sum == x)
109. {
110. solpair.i = slist[i].sl1\_index;
111. solpair.j = slist[i].sr1\_index;
112. ans.push\_back(solpair);
113. }
114. else
115. {
116. break;
117. }
118. }
119. solpair.i = slist[mid].sl1\_index;
120. solpair.j = slist[mid].sr1\_index;
121. ans.push\_back(solpair);
122. for (ll i = mid + 1; i < slist.size(); i++)
123. {
124. if (slist[i].sum == x)
125. {
126. solpair.i = slist[i].sl1\_index;
127. solpair.j = slist[i].sr1\_index;
128. ans.push\_back(solpair);
129. }
130. else
131. {
132. break;
133. }
134. }
135. return ans;
136. }
137. else if (mpz\_cmp(slist[mid].sum, x) == 1)
138. {
139. r = mid - 1;
140. }
141. else
142. {
143. l = mid + 1;
144. }
145. }
146. }
147. bool cmp\_mod(ModuleKnapsck x, ModuleKnapsck y)
148. {
149. return x.sum < y.sum;
150. }
151. bool cmp\_sol(Solution x, Solution y)
152. {
153. return (mpz\_cmp(x.sum, y.sum) < 0);
154. }
155. void stdout\_vector(vector<int> v)
156. {
157. for (int i = 0; i < v.size(); i++)
158. cout << v[i] << ' ';
159. }
160. void file\_vector(vector<int> &v, ofstream &outfile)
161. {
162. for (int i = 0; i < v.size(); i++)
163. outfile << v[i] << ' ';
164. }
165. void stdout\_vector(vector<mpz\_t \*> element)
166. {
167. for (int i = 0; i < element.size(); i++)
168. gmp\_printf("%Zd\n", \*element[i]);
169. }
170. void stdout\_vector(vector<QuarterKnapsack> v)
171. {
172. for (int i = 0; i < v.size(); i++)
173. {
174. gmp\_printf("%Zd ", v[i].sum);
175. for (int j = 0; j < v[i].index.size(); j++)
176. {
177. cout << v[i].index[j] << ' ';
178. }
179. cout << endl;
180. }
181. cout << endl;
182. }
183. void insertset(set<int> &s, vector<int> &p)
184. {
185. for (int i = 0; i < p.size(); i++)
186. s.insert(p[i]);
187. }
188. void algorithm3(vector<QuarterKnapsack> &sl1, vector<QuarterKnapsack> &sr1, vector<QuarterKnapsack> &sl2,
189. vector<QuarterKnapsack> &sr2,
190. vector<ModuleKnapsck> &sr1m, vector<ModuleKnapsck> &sr2m,
191. mpz\_t target, ll M, ll Ml, ll Mr, int id)
192. {
193. clock\_t start, end;
194. start = clock();
195. cout << "algorm3 started" << endl;
196. ll step = 0;
197. // cout << Ml << ' ' << Mr << endl;
198. for (unint m = Ml; m <= Mr; m++)
199. {
200. step++;
201. if (step % 1 == 0)
202. {
203. double percentage = ((m - Ml) \* 1.0) / (Mr - Ml);
204. int val = (int)(percentage \* 100);
205. cout << '\n'
206. << "thread: " << id << ' '
207. << val << "% " << m << ' ' << Ml << '/' << Mr << endl;
208. }
209. vector<Solution> s;
210. for (unint sl1\_index = 0; sl1\_index < sl1.size(); sl1\_index++)
211. {
212. mpz\_t tmp;
213. mpz\_init(tmp);
214. mpz\_ui\_sub(tmp, m, sl1[sl1\_index].sum);
215. vector<unint> return\_list = binarySearch(sr1m, mpz\_fdiv\_ui(tmp, M));
216. mpz\_clear(tmp);
217. for (int i = 0; i < return\_list.size(); i++)
218. {
219. unint sr1\_index = return\_list[i];
220. mpz\_t sum;
221. mpz\_init(sum);
222. mpz\_add(sum, sl1[sl1\_index].sum, sr1[sr1\_index].sum);
223. Solution sol;
224. mpz\_set(sol.sum, sum);
225. sol.sl1\_index = sl1\_index;
226. sol.sr1\_index = sr1\_index;
227. s.push\_back(sol);
228. mpz\_clear(sum);
229. }
230. }
231. // gmp\_printf("%Zd\n", s[i].sum);
232. sort(s.begin(), s.end(), cmp\_sol);
233. // cout << s.size() << endl;
234. // for (int i = 0; i < s.size(); i++)
235. // {
236. // gmp\_printf("%Zd ", s[i].sum);
237. // stdout\_vector(sl1[s[i].sl1\_index].index);
238. // cout << endl;
239. // }
240. // cout << "---" << endl;
241. for (int sl2\_index = 0; sl2\_index < sl2.size(); sl2\_index++)
242. {
243. mpz\_t tmp;
244. mpz\_init(tmp);
245. mpz\_sub\_ui(tmp, target, m);
246. mpz\_sub(tmp, tmp, sl2[sl2\_index].sum);
247. vector<unint> return\_list = binarySearch(sr2m, mpz\_fdiv\_ui(tmp, M));
248. mpz\_clear(tmp);
249. for (int i = 0; i < return\_list.size(); i++)
250. {
251. unint sr2\_index = return\_list[i];
252. mpz\_t tt;
253. mpz\_init(tt);
254. mpz\_sub(tt, target, sl2[sl2\_index].sum);
255. mpz\_sub(tt, tt, sr2[sr2\_index].sum);
256. vector<IndexPair> return\_sol = binarySearchSol(s, tt);
257. mpz\_clear(tt);
258. // cout << return\_sol.size() << endl;
259. for (int sol\_index = 0; sol\_index < return\_sol.size(); sol\_index++)
260. {
261. // set<int> checkset;
262. // insertset(checkset, sl1[return\_sol[sol\_index].i].index);
263. // insertset(checkset, sr1[return\_sol[sol\_index].i].index);
264. // insertset(checkset, sl2[sl2\_index].index);
265. // insertset(checkset, sr2[sr2\_index].index);
266. // if (checkset.size() != Hamingweight){
267. // continue;
268. // }
269. end = clock();
270. // gmp\_printf("%Zd\n", sl1[return\_sol[sol\_index].i].sum);
271. // gmp\_printf("%Zd\n", sr1[return\_sol[sol\_index].i].sum);
272. // gmp\_printf("%Zd\n", sl2[sl2\_index].sum);
273. // gmp\_printf("%Zd\n", sr2[sr2\_index].index);
274. cout
275. << "\rsolved in " << (double)(end - start) / CLOCKS\_PER\_SEC << endl;
276. stdout\_vector(sl1[return\_sol[sol\_index].i].index);
277. stdout\_vector(sr1[return\_sol[sol\_index].j].index);
278. stdout\_vector(sl2[sl2\_index].index);
279. stdout\_vector(sr2[sr2\_index].index);
280. cout << endl;
281. ofstream outfile;
282. outfile.open("./outH", ios::app);
283. file\_vector(sl1[return\_sol[sol\_index].i].index, outfile);
284. file\_vector(sr1[return\_sol[sol\_index].j].index, outfile);
285. file\_vector(sl2[sl2\_index].index, outfile);
286. file\_vector(sr2[sr2\_index].index, outfile);
287. outfile << endl;
288. result = true;
289. return;
290. }
291. }
292. }
293. for (int i = 0; i < s.size(); i++)
294. mpz\_clear(s[i].sum);
295. s.clear();
296. }
297. }
298. int combination(int n, int k)
299. {
300. if (n == k || k == 0)
301. return 1;
302. else
303. return combination(n - 1, k) + combination(n - 1, k - 1);
304. }
305. void initKnapsack(int n, int r, vector<mpz\_t \*> &elements,
306. vector<QuarterKnapsack> &sl1, vector<QuarterKnapsack> &sr1,
307. vector<QuarterKnapsack> &sl2, vector<QuarterKnapsack> &sr2)
308. {
309. vector<QuarterKnapsack> tt;
310. std::vector<bool> v(n);
311. std::fill(v.end() - r, v.end(), true);
312. int total\_comb = combination(n, r);
313. int comb\_step = 0;
314. do
315. {
316. double percentage = comb\_step \* 1.0 / total\_comb;
317. int val = (int)(percentage \* 100);
318. if (comb\_step % 1000 == 0)
319. {
320. cout << '\r' << val << '%';
321. }
322. QuarterKnapsack tmp;
323. mpz\_init(tmp.sum);
324. for (int i = 0; i < n; ++i)
325. {
326. if (v[i])
327. {
328. mpz\_add(tmp.sum, tmp.sum, \*elements[i]);
329. tmp.index.push\_back(i);
330. }
331. }
332. tt.push\_back(tmp);
333. comb\_step++;
334. } while (std::next\_permutation(v.begin(), v.end()));
335. srand(time(0));
336. random\_shuffle(tt.begin(), tt.end());
337. for (comb\_step = 0; comb\_step < tt.size(); comb\_step++)
338. {
339. if (comb\_step < total\_comb / 4)
340. {
341. sl1.push\_back(tt[comb\_step]);
342. }
343. else if (comb\_step < total\_comb / 2)
344. {
345. sr1.push\_back(tt[comb\_step]);
346. }
347. else if (comb\_step < 3 \* total\_comb / 4)
348. {
349. sl2.push\_back(tt[comb\_step]);
350. }
351. else
352. {
353. sr2.push\_back(tt[comb\_step]);
354. }
355. }
356. }
357. void howgrave(vector<mpz\_t \*> elements, mpz\_t target, int K, int threadNumber)
358. {
359. vector<QuarterKnapsack> sl1, sr1, sl2, sr2;
360. cout << "init..." << endl;
361. initKnapsack(N, K / 4, elements, sl1, sr1, sl2, sr2);
362. cout << '\n';
363. cout << "quarter size: " << sl1.size() << ' ' << sr1.size() << ' '
364. << sl2.size() << ' ' << sr2.size() << ' ' << endl;
365. double ll = pow(2, N / 4.0), rr = pow(2, (N / 4.0) + 1);
366. default\_random\_engine e(time(0));
367. uniform\_real\_distribution<double> u(ll, rr);
368. int M = (int)u(e);
369. M = pow(2, 20);
370. cout << "Mod: " << M << endl;
371. cout << "init module" << endl;
372. vector<ModuleKnapsck> sr1m;
373. for (int i = 0; i < sr1.size(); i++)
374. {
375. ModuleKnapsck \_t;
376. \_t.sum = mpz\_fdiv\_ui(sr1[i].sum, M), \_t.index = i;
377. sr1m.push\_back(\_t);
378. }
379. vector<ModuleKnapsck> sr2m;
380. for (int i = 0; i < sr2.size(); i++)
381. {
382. ModuleKnapsck \_t;
383. \_t.sum = mpz\_fdiv\_ui(sr2[i].sum, M), \_t.index = i;
384. sr2m.push\_back(\_t);
385. }
386. cout << "sorting" << endl;
387. sort(sr1m.begin(), sr1m.end(), cmp\_mod);
388. sort(sr2m.begin(), sr2m.end(), cmp\_mod);
389. cout << "sort completed" << endl;
390. // stdout\_vector(sl1);
391. // stdout\_vector(sr1);
392. // stdout\_vector(sl2);
393. // stdout\_vector(sr2);
394. // algorithm3(sl1, sr1, sl2, sr2, sr1m, sr2m, target, M, 0, M - 1, 1);
395. int ml = 0, mr = M / threadNumber;
396. // algorithm3(sl1, sr1, sl2, sr2, sr1m, sr2m, target, M, 0, M - 1, file);
397. // thread t1(algorithm3, ref(sl1), ref(sr1), ref(sl2), ref(sr2), ref(sr1m), ref(sr2m), target, M, M /2 + 1, M-1, file);
398. for (int i = 0; i < threadNumber; i++)
399. {
400. thread t(algorithm3, ref(sl1), ref(sr1), ref(sl2), ref(sr2), ref(sr1m), ref(sr2m), target, M, ml, mr, i + 1);
401. t.detach();
402. ml = mr + 1;
403. mr += M / threadNumber + 1;
404. if (abs(M - mr) < threadNumber)
405. {
406. mr = M - 1;
407. }
408. }
409. while (result == false)
410. {
411. sleep(5);
412. /\* code \*/
413. }
414. }
415. int main(int argc, char \*\*argv)
416. {
417. FILE \*fp;
418. char StrLine[1024];
419. if ((fp = fopen("./data140", "r")) == NULL)
420. {
421. cout << ("[-]Error, can't open file") << endl;
422. return -1;
423. }
424. vector<mpz\_t \*> elements;
425. mpz\_t arr[N];
426. for (int i = 0; i < N; i++)
427. {
428. mpz\_init(arr[i]);
429. mpz\_set\_str(arr[i], fgets(StrLine, 1024, fp), 10);
430. elements.push\_back(arr + i);
431. }
432. fclose(fp);
433. mpz\_t target;
434. mpz\_init(target);
435. if (argc < 2){
436. cout << "[-]Error, no target" << endl;
437. return 0;
438. }
439. char \*target\_str = argv[1];
440. mpz\_set\_str(target, target\_str, 10);
441. gmp\_printf("target %Zd\n", target);
442. if (argc < 3)
443. {
444. cout << "[-]Error, no threadNumber" << endl;
445. return 0;
446. }
447. int threadNumber = stoi(argv[2]);
448. cout << "threadNumber: " << threadNumber << endl;
449. howgrave(elements, target, Hamingweight, threadNumber);
450. // mpz\_clear(target);
451. // for (int i = 0; i < N; i++)
452. // {
453. // mpz\_clear(arr[i]);
454. // }
455. return 0;
456. }

## BKZ规约算法代码

1. import random
2. import math
3. from tqdm import tqdm
4. from functools import cmp\_to\_key
5. def clone\_runoob(li1):
6. li\_copy = li1[:]
7. return li\_copy
8. def compare(x, y):
9. return norm(x) - norm(y)
10. def listadd(x, y):
11. assert (len(x) == len(y))
12. return [x[i] + y[i] for i in range(len(x))]
13. def listsub(x, y):
14. assert (len(x) == len(y))
15. return [x[i] - y[i] for i in range(len(x))]
16. def checkz(zlist):
17. pass
18. def norm(vs):
19. res = 0
20. for i in range(len(vs)):
21. res += vs[i] \* vs[i]
22. return math.sqrt(res)
23. def solve(alist, n, k, s):
24. N = int(sqrt(n))
25. blist = [[] for j in range(n + 1)]
26. for i in range(len(blist)):
27. if i < len(blist) - 1:
28. blist[i] = [1 if i == j else 0 for j in range(n)]
29. blist[i] += [N \* alist[i], N]
30. else:
31. blist[i] = [0 for i in range(n)]
32. blist[i] += [-(N \* s), -(k \* N)]
33. shuffle(blist)
34. while True:
35. # step3 - reduce
36. print("\rstep3...")
37. m = matrix(ZZ, blist)
38. zlist = m.BKZ(block\_size=24)
39. print('norm', norm(zlist[0]))
40. print(zlist[0])
41. #step4 - check
42. if norm(zlist[0]) ^ 2 - k < 0.5:
43. print('found SVP', zlist[0])
44. break
45. # step5 - find
46. print("\rstep5...", end='')
47. blist = []
48. for i in range(n):
49. blist.append(list(zlist[i]))
50. f = True
51. while f:
52. f = False
53. blist = sorted(blist, key=cmp\_to\_key(compare))
54. for j in range(n):
55. for kk in range(j):
56. if norm(listadd(blist[j], blist[kk])) < norm(blist[j]):
57. blist[j] = listadd(blist[j], blist[kk])
58. f = True
59. elif norm(listsub(blist[j], blist[kk])) < norm(blist[j]):
60. blist[j] = listsub(blist[j], blist[kk])
61. f = True
62. #def main():
63. # alist
64. # s
65. # k
66. # n
67. # if \_\_name\_\_ == '\_\_main\_\_':
68. # main()