分类号 0153.1 学校代码 10590
UDC 51 密 级 公开

深圳大学硕士学位论文

格基规约相关算法的研究

学位申请人姓名 周 娜

专 业 名 称 计算机科学与技术

学院（系、所） 计算机与软件学院

指导教师姓名 陈剑勇教授、王平讲师

深圳大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文 格基规约相关算法的研究 是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写的作品或成果。对本文的研究所做出重要贡献的个人和集体，均已在文中以明确的方式标明。本声明的法律后果由本人承担。

论文作者签名：周娜 日期：2017年5月17日

学位论文使用授权说明

（必须装订在印刷本首页）

本学位论文作者完全了解深圳大学关于收集、保存、使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属深圳大学。学校有权保留学位论文并向国家主管部门或其他机构送交论文的电子版和纸质版，允许论文被查阅和借阅。本人授权深圳大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复印手段保存、汇编学位论文。

（涉密学位论文在解密后适用本授权书）

论文作者签名：周娜 导师签名：

日期：2017年5月17日 日期：2017年5月17日

摘　要

随着信息时代的快速发展，信息安全问题与我们密切相关，因而密码学逐渐成为人们日益关注的焦点。数据的安全性与保密性是密码技术的核心要素，而格基规约算法是密码学中核心要素的重要体现，它是一种典型的密码学分析技术。

当前，LLL(Lenstra，Lenstra，Lovasz) 是最经典的格基规约算法。而以 LLL 算法为基础的变体数不胜数，其目的都是为了优化。格基规约算法具有广泛的应用，例如：数论，整数规划，丢番图逼近以及密码学等方面。而本文主要从优化格基规约算法、用规约思想来初始化参数去解决球译码算法、格理论在 0-1 整数背包中的应用以及背包算法的并行化方面来阐述，具体内容如下：

（一）格基规约是求解格中非零近似最短向量的问题，著名的 LLL 算法可在多项式时间内求解出可证明的约减基。l 次规约是 LLL 算法的变体，有着更高质量的约减基，但是运行时间大幅度增加。而分块 LLL 规约则是约减基略差但运行时间减少明显的算法。因此，本文提出在分块 LLL 规约中加入 l 次规约的思想。有效汲取二者的优势，生成分块 l 次规约算法，其在时间−质量方面可达到相对权衡。

（二）球译码算法是解决整数最小二乘问题的有效方法，而球译码算法的关键问题是搜索空间中初始化半径的选择。论文首先提出利用阿达玛比率选取一组高质量的基；其次提出用 QR(orthogonal matrix，upper triangular matrix) 分解，LLL 优化规约以及带预测技术的宽度优先搜索算法 K-Best(BFS+PEDS) 叠加的方式来约减所选取的一组基；最后再利用球译码算法来求解，其过程等价于在格中求解最近向量问题。

（三）背包问题是一个 NPC(non-deterministic polynomial complete) 问题，同时也是经典的组合优化问题。利用（一）中优化规约算法来提高密码分析的效率，缩短密码分析的时间。规约算法同样可应用在基于格的背包密码系统的分析上。为了深入了解背包原理，研究其常规算法的思想。因此，针对 0-1 整数背包，提出新生成算法的并行化来加速解决背包问题。

关键词： 格基规约；LLL 规约；分块规约；球译码算法；背包问题

Abstract

With the rapid development of information age, information security problem is closely related to us. Thus cryptography has gradually become the focus that people increasingly concern about. The security and privacy of data elements is the core of the password techniques, and lattice reduction algorithm is the important embodiment of the core elements in cryptography. It is a kind of typical analysis technology of cryptography.

At present, LLL is the most classic lattice reduction algorithm. The improved algorithms based on LLL are numerous. Their purposes are to optimize algorithms. Lattice reduction algorithm has a wide range of applications, such as number theory, an integer programming, diophantine approximation, cryptography and so on. This article mainly elaborates from the optimized algorithm for lattice reduction. Parameter initialized through lattice reduction algorithm is to solve the sphere decoding algorithm(SDA). Lattice theory is used in 0-1 integer knapsack and parallelization of knapsack algorithm. Detailed content is as follows:

Firstly, lattice reduction is to solve the nonzero approximate shortest vector problem (appr-SVP) in lattice. The famous LLL algorithm can solve the provable reduction base in polynomial time. l reduction is a variant of the LLL algorithm, which has a higher quality of reduction base. However, the running time significantly increases. The effect of the block LLL is slightly inferior, but the running time obviously decreases a lot. Therefore, we put forward the thought that l reduction is added to the block LLL lattice reduction. It can effectively absorb the advantages of both, and form a relative balance algorithm of time and quality.

Secondly, the sphere decoding algorithm is an effective method to solve the problem of integer least squares, and the key problem of the sphere decoding algorithm is the choice of initial radius in the search space. First, this paper puts forward that using the Hadamard ratio is to select a group of high quality base. Second, using QR decomposition, LLL optimization reduction and breadth first search algorithm (K-Best, that is BFS+PEDS) with prediction

technology to reduce the selected the base. The last, reusing sphere decoding algorithm(SDA) is to find a solution. The process is equivalent to solve the Closest Vector Problem(CVP) in lattice.

Thirdly, the knapsack problem is a NP-complete problem which is also a classic combinatorial optimization problem. Using the optimized reduction algorithm of above is not only to improve the efficiency of the cryptographic analysis, but also to shorten the time of password analysis. Reduction algorithm can be used in knapsack password system analysis based on lattice as well. In order to further understand the knapsack principle, the idea of the conventional algorithm is studied. Thus, for 0-1 integer knapsack problem, we put forward the parallel of a new generation algorithm to accelerate the speed of solving knapsack problem.

Key word: Lattice Reduction; LLL Reduction; Block Reduction; Sphere Decoding Algorithm; Knapsack Problem

目　录

第 1 章 绪论

1.1 研究背景

格基规约不仅在格研究领域有着非常重要的作用，同时也是密码学的有力工具。许多格上的问题都可以直接通过格基规约来解决，甚至一些 NP 完全问题都可应用格基规约来近似求解。另一方面，一些密码分析也可以被约减成格基规约问题。因此，格基规约算法的创新设计在格研究领域以及实践应用方面有着很重要的价值。

格基规约算法在通信、数论、整数规划、丢番图逼近、凸体分析以及密码设计与分析等方面有着广泛的应用。格基规约的概念首先在 Lagrange[1]，Gauss[2]和 Hermite[3]中出现，之后 Korkine、Zolotarev[4,5]以及一些学者研究二次方形式的规约。Minkowski[6]在几何数论方面也做了一些特定的研究。

格是典型的线性代数架构，其中的计算是简单的线性代数形式。基于格的问题通常是一个 NPC 问题，而格基规约是研究格中困难问题的有效方法，其中困难问题主要包括求解 SVP 和 CVP 等。譬如：Gauss 规约[7]能在多项式时间内发现二维格的最短向量。而求解 SVP 的算法有两大类：近似算法和精确算法[8]。近似算法的所得向量是一个长度满足某个界限的非零短向量，由于 SVP 是 NP-Hard 的，所以在高维格中，通常只能应用近似算法；而精确算法是能找到可证明的最短向量。这两类算法各有价值，相互协作，相辅相成。

近年来，利用格上的理论知识来研究公钥密码系统似乎已成为密码学界的热潮，吸引众多科研人员的高度关注和青睐。在利用格上难题构造公钥密码策略以前，密码学界的研究者则更侧重研究格中的 SVP、CVP、格理论在密码分析中的应用等。研究格中的困难问题 SVP 和 CVP 的一个重要方法就是格基规约算法，当前学者们研究格基规约算法的主要原因有两个：其一，格基规约算法对当前基于传统数学难题的公开密钥密码体制（典型算法：RSA）的攻击，使得当前成熟的公钥密码算法在安全性上存在着很大的威胁。其二，由于密码体制的分析结果是衡量密码体制安全性的重要依据，密码设计者和使用者都密切关注密码分析问题，因此，可以从密码分析的角度说明基于格上难题设计的公钥密码方案是较为安全的，可以抵抗量子攻击。在密码界，利用一个不可能很快得到解的数学难题来构造安全的密码方案，可理解为很多密码方法并没有严格的安全性证明，只是这种加密方案在经过多次攻击之后，并没有对其安全性造成本质上的威胁，就可认为此方案是安全的。

格基规约算法在格密码分析中发挥了很大的价值，除此之外，在求解整系数多元线性方程或者同余多元线性方程较小的根时，格基规约理论能够在有效的复杂度范围内进行求解，因此，可以将格基规约理论有效地应用到基于传统困难问题的公钥密码分析当中，如破解背包加密算法，截短的线性同余随机序列生成器以及对基于大整数分解的公钥加密方案 RSA 的分析和攻击等。目前，格基规约理论对低指数或者存在安全漏洞的 RSA 加密方案的安全性可能造成很大的威胁，但没有彻底破解 RSA 加密方案，因此，若对 RSA 方案进行加密之前，对明文进行填充或者增大方案指数等采取进一步加密的策略，就可有效抵抗可能的攻击，从而提高方案的安全性。

本文研究格基规约主要考虑进一步优化算法，在保持求得相对较短约减基的基础上，可以花费更少的时间，特别是维数越大的时候，效果越明显。优化的格基规约算法是综合考虑约减效率和运行时间，比其它改进的规约算法更加实用。对于改进的算法，为了发挥它的利用价值，可进一步应用到球译码算法的初始化参数中来约减半径，进而可以缩小搜索空间，加快找解速度。

研究球译码算法的主要原因是其有着重要的应用价值。无线通信是当今世界上较热门的科研领域之一，它突破了有限通信的物理障碍，使得用户可以自由地在任何无线电波能够到达的地方进行通信，这大大扩展了通信的空间和活力。正如 20 世纪 90 年代有限通信迅速发展一样，目前无线通信面临的主要问题就是如何提供高效的传输速率。由于无线通信借以提高数据传输的传统资源-频率带宽和发射功率-目前都已濒临饱和。因此，仅依靠这两种资源的损耗来提高传输速率是行不通的。而用户对图像传输和电话会议等高传输速率数据业务的需求量越来越迫切，总体来说是根本无法满足用户的这类需求。

近几年来，信息理论的研究表明：在存在丰富散射的无线信道中，收发两端均采用多天线，即多输入多输出（MIMO）系统可以获得比单发单收系统更高的容量。因此，MIMO 技术引起了研究者的极大关注并且得到了快速的发展。由于贝尔实验室垂直分层空时码（V-BLAST）的结构可以充分利用 MIMO 的系统容量。因此，随即出现了关于 MIMO 接收系统的一系列检测算法。

球译码是一种公认的比较好的算法，它是一种频谱利用率高，误码性能高的检测方法。在 V-BLAST 系统检测中，目前采用的迫零（ZF）算法，最小均方误差（ MMSE）算法，排序连续干扰抵消（OSIC）或最大似然（ML）等准则来进行译码，前三种算法实现起来相对简单，但误码率性能差，而使用 ML 检测能得到更好的

性能，但是其复杂度较高，不易于实现。而基于 ML 检测的球译码算法（SDA）是一种性能优化，复杂度适中的检测算法。已经证明，采用穷尽搜索的 ML 检测算法的复杂度随天线数呈指数增长，而 SDA 的复杂度在很大信噪比范围内与天线数呈多项式关系，故 SDA 可以用较少的计算量来获得最大似然译码性能。

球译码算法带来的优点在于它不需要像传统的最大似然译码算法那样在整个格内对所有的格点进行搜索，而只需要在一个事先设定的有限球形区域进行搜索，如果该区域所包含的点数相对于整个格内的总点数是相当小时，那么其搜索时间就会大大减少。因此，有效减少搜索半径成为 SDA 的一个研究目标。

1.2 研究现状

格基规约的经典算法是 LLL，1982 年 Lenstra，Lenstra，Lovasz[9]提出了能够在多项式时间内获得可证明输出质量的约减基，其中参数δ的值通常取 0.75，使用 LLL 约减算法在多项式时间内可求解到n维格中的一个短向量，短向量的欧几里德范数小于格中最短向量欧几里德范数的$2^{(n-1)/2}$倍。由于它的高效性和简易性，LLL 算法的应用是 NP 困难问题中最具突破性的问题之一。受 LLL 算法的启发，之后出现许多更快更精确的算法，并且把这突破性应用在不同的领域。Schnorr[10]总结了 LLL 算法，提出通过调整参数K，可在运算速度和近似因子之间达到一个平衡的约减算法。使用一个合理的参数，它的近似因子可能达到$2^{n(\log\log n)^2/\log n}$。

Seysen[11]设计了一种算法，在计算过程中得到的整数小于原始输入的整数，也能实现格基规约的意图，在维度不高的情况下，Seysen 的算法执行效果很好。Schnorr、Euchner[12]商讨了在运用算术来计算基向量时，使用浮点运算去实现格拉姆-施密特正交化的可能性，这种方法可以提高他们所提出算法[10]的实际性能，但缺陷是使得约减不稳定。除此之外，他们又提出新形式的格基规约算法，运用“深插入”的方式[12]。

Schnorr[13]使得思想普通化，即在 LLL 算法的基础上，提出了格基的划分，为了使得b_j^*能够达到最短，向量b_j可以与其后的连续k个向量交换位置，这种思想称为 BKZ 约减算法。分块约减算法可以获得更短的向量，但需要以更多的时间为代价。针对这个特征，Schnorr、Horner[14]提出了一个提高版本的 BKZ 算法，使用“修剪”技术来加速块的搜索进程，同时，它可确保约减的最终结果基本保持不变。尽管 LLL 算法和它的各种变体被广泛应用，但是当维度n大于 350 的时候，在实际过程中，这些算法将

变得非常困难。于是 Koy、Schnorr 又一次改进 LLL，提出分块 LLL 约减算法[15]。算法的新类型可实现维度 n 小于 1000 的有效约减。之后，他们又提出了浮点正交化的分块 LLL 约减算法的形式[16]。余位驰等人[17]在 LLL 的基础上提出 l 次约减，使得约减效率更高。2014 年，Fontein、Schneider 和 Wanger 提出 PotLLL 算法[18]。2016 年，Espitau 和 Joux 提出应用区间运算的自适应精确的 LLL 和 PotLLL 算法[19]，主要就是用区间运算代替浮点运算，使得运行速度更快。

当前，存在几种主要的格基约减算法，这些算法都有比较广泛的应用。而且，格基规约算法的变体也是多样性的，其目的都是为了使得约减效果或运行时间更加优化。l 次约减相比原始 LLL 算法能够获得更加有效的约减基。而分块 LLL 约减相比原始 LLL 算法可以节俭一定的运行时间，这两种变体都有自己的优势。因此，不管是高斯 LLL、深插 LLL、分块 LLL、PotLLL、还是 l 次规约算法等，这些版本有的存在使用的限制条件，很多算法只是单一地考虑了时间快速或者约减基相对较好的结果，而忽略了两者中的另一方面，这在实际应用中不是很理想。所以，现有技术中缺少一种有效考虑二者的综合因素，既能保证时间相对短，又能获得较好约减基的算法。

格基规约算法主要用来求解 SVP，其属于近似算法。而精确算法也有很大的应用价值，与近似算法互补应用，又可分为确定性算法和随机性算法。

1981 年，Pohst、Fincke[20,21]提出了时间复杂度为 $2^{O(n^2)}$ 的 Enum 算法。两年后，Kannan[22]又提出了一个 Enum 算法，其时间复杂度为 $2^{O(n\log n)}$，Helfrich 对该算法进行深度分析，得到一个复杂度为 $n^{n/2+O(n)}$ 的上界[23]。2007 年，Kannan 算法的时间复杂度上界被 Hanrot 和 Stehle 限制为 $n^{n/2e+O(n)}$[24]。之后，他们又证明了，存在一类特殊基可以使得 Kannan 算法的时间复杂度上界至少为 $n^{n/2e+O(n)}$[25]。

实际常用的枚举算法（ENUM）[13,26]都是应用 Pruning 技术来实现。2010 年，Gama 提出了 Extremely Pruning 技术[27]，该技术观点有效地提高了枚举运行的效率。同年，Micciancio 等人针对格点的 Voronoi 细胞结构，给出一种可解决 SVP、CVP、SIVP 等问题的新型确定算法[28]，该算法的时间复杂度为 $2^{2n+O(n)}$，空间复杂度约为 $2^{n+O(n)}$。

精确算法的另一种类型为随机算法，其也能有效地求解 SVP。2001 年，Ajtai 提出了 AKS 算法，其时间复杂度为 $2^{O(n)}$[29]。2004 年 Regev[30]和 2008 年 Nguyen、Vidick[31]分别对 AKS 的时间复杂度进行修正。2010 年，Micciancio、Voulgaris 运用估计方法对 AKS 筛法的时空复杂度上界进行有效估计：$2^{3.4n+O(n)}$，$2^{1.97n+O(n)}$[32]，该估计源于球面覆

盖相关问题的经典结论，还提出时空复杂度相继为$2^{3.199n+O(n)}$和$2^{1.325n+O(n)}$的 ListSieve 筛法。2008 年，Nguyen 等人基于 AKS 筛提出了时空复杂度相继为$2^{0.415n+O(n)}$和$2^{0.2075n+O(n)}$的第一个随机启发式算法[31]。2011 年，两层筛的启发式随机筛算法在[33]中被提出，此技术实现了时空复杂度的有效平衡。2013 年，Zhang 在此基础上，进一步提出三层筛的思想[34]，时空复杂度分别为$2^{0.3778n+O(n)}$和$2^{0.2833n+O(n)}$。2014 年，Hayasaka 等人[35]提出基于有限域上$GF(p^n)$的数域筛构建 3 维格筛的策略，相对于 Zajac 的有限域$GF(p^6)$的 3 维线筛，其运行时间大幅度减少。2016 年， Becker1 和 Laarhoven[36]提出使用高维超正体的位置敏感哈希函数族联合启发格筛算法，展现了在任意格和理想格中的加速求解最短向量问题的方法，可得n维中最短向量问题的时间复杂度为$2^{0.298n+O(n)}$。

格中最主要的困难问题是解决 SVP 和 CVP。求解 SVP 的近似算法和精确算法不胜枚举。而求解 CVP 的方法却屈指可数，最常用的就是 Babai 估计的最近平面算法[37]。其次是经常用于解码中的 BDD 问题，即目标向量距离格相对较近的最近向量问题，可参考相关文献[38,39,40]。

1.3 主要贡献

本文主要研究经典格基规约 LLL 算法，从理论层面深入剖析 LLL 算法的原理，并从规约基的约减特点方面提出相应的改进算法，从而提高约减效率且降低运行时间。同时，进一步研究格基规约算法的应用领域，譬如：初始化球译码算法的搜索半径，解决 0-1 整数背包问题等。主要工作内容如下：

（一）针对经典格基规约 LLL 算法的约减方式进行改进，格基规约是求解格中非零近似最短向量的问题，LLL 算法可在多项式时间内求解出可证明的约减基。l次规约和分块 LLL 规约都是 LLL 算法的变体，前者有着更高质量的约减基但是运行时间大幅度增加，后者则呈现相反的特点。因此，结合二者的优点，提出在分块 LLL 规约中加入l次规约的思想，形成一个时间质量相对权衡的改进算法。与l次规约算法类似的有 DeepLLL 和 PotLLL，都是针对 LLL 算法的改进，DeepLLL 的思想如果当前两个向量满足条件（2），就进行后面两个向量之间的规约，而没有回溯规约，如果当前两个向量不满足条件，才进行回溯规约；PotLLL 的思想如果当前两个向量满足条件（2），就进行回溯规约，如果一直满足就要回溯规约到$k=1$。前者使得相应的向量没有回溯规约到，从而就没有进一步规约；后者使得回溯规约到 1，浪费大量的时间。因此，从

上述两种方案比较来看，l次规约的特点是l的值可以动态确定和变化，具体可以考虑使用 Hadamard 比率来判断，若比率的值相对大于 0.7 以上，l的值可以取得小些，而如果值小于等于 0.7，l的值可以相对大些。可适当通过基的正交性来做参考，这样可以减少一定的运行时间。实验表明，所提出的算法在保持获得l次规约基不变或更好的情况下，可减少一定的运行时间，维数越大，效果越明显。

（二）针对格基规约的应用来提高球译码算法（SDA），首先利用 Hadamard 比率来选取一组高质量的基；其次提出用 QR 分解, LLL 优化规约以及带预测技术的宽度优先搜索 K-Best 算法（BFS+PEDS）组合的方式来约减所选取的这组基；最后用约减的结果作为初始化参数，再利用球译码算法（SDA）去求解整数最小二乘问题，其过程等价于在格中求解最近向量问题（CVP）。实验表明，所提出的思想可达到预期的效果，并且可减少大约 61.8%的运行时间。

（三）基于格构建的密码系统可直接应用格基规约的算法来分析，常用的密码系统有 GGH、同余、背包、NTRU 等。基于此，可用我们的第一个创新点——分块l次规约来分析背包密码系统的问题，能准确并快速地获得加密过的明文；针对 0-1 整数背包，也存在很多常规的算法来解决。因此，本文最后针对 0-1 整数背包中新生成算法来实现它的并行化，用并行思想来加速解决背包问题。实验可知，并行实现与串行的相比，可减少大量时间，效果非常明显。

1.4 组织结构

本文组织结构如下：

第 1 章主要介绍格基规约的研究背景，对格基规约算法的国内外现状以及发展进程进行详细阐述，同时简述主要贡献以及本文的组织结构。

第 2 章主要介绍格中的基础概念，格基规约的经典算法 LLL，BKZ 以及格基规约所涉及的应用领域——球译码算法，0-1 整数背包算法的相关概念。

第 3 章首先引出 LLL 算法以及它的变体，并提出相应的改进算法以及分析，展现所改算法的详细流程图。之后从数据的选取，规约参数δ，分块大小k，回溯次数l分别取不同值时以及不同算法的约减效果方面进行比较。

第 4 章首先引出球译码算法，并提出球译码的关注点——初始化半径。之后从 Hadamard 比率，QR 分解（R 对角线转正），LLL 优化约减，K-Best(BFS+PEDS)方面

综合约减初始化参数，进一步分析算法的复杂度，展现算法的流程图。下一个阶段进行算法仿真，从数据的选取到不同维度的访问节点数，执行时间以及初始化半径方面进行比较。

第 5 章首先引出 0-1 整数背包问题，进而讨论优化格基规约算法在背包密码系统中应用的详细步骤。之后针对常规算法求解 0-1 整数背包的问题进行研究，提出相应算法的并行思想，通过算法的设计，实现与分析，进行算法的仿真。进一步从数据的选取，串、并行效果的比较以及并行算法的找解情况进行展现。

第 6 章总结本文所做的主要工作，并对相应的内容提出几点建议或者可行的方案，为下一步研究明确方向。

第 2 章 基础知识

本章将介绍论文中可能涉及到的基本格理论知识以及数学符号。2.1 小节主要介绍格的基本概念，格基的相关定义和应用；2.2 小节主要介绍经典格基规约算法的规则，包括 LLL、分块 LLL、BKZ 等；2.3 小节简单介绍格基规约相关算法的应用领域，充分了解本章内容对接下来的章节起到关键作用。

2.1 格理论概念

格 $L \subset R^m$ 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 $b_1, b_2, \cdots, b_n$ 的整数倍的线性组合生成的：

$$L(b_1, b_2, \cdots, b_n) = \{\sum_{i=1}^{n} x_i b_i, x_i \in Z\} \tag{2-1}$$

集合 L 为以 $b_1, b_2, \cdots, b_n$ 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量（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 $\|v_i\|$ 表示欧几里德范数， $< v_i, v_i >$ 表示向量 v_i 的内积，它们之间存在 $\|v_i\| = < v_i, v_i >^{1/2}$ 的关系。 $\lambda_1(L)$ 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[41]。因此，从几何角度可知 n 维格的行列式： $\det(L) = CellVol(L) = \|v_1\|\|v_2\|\cdots\|v_n\|\sin\theta_1 \sin\theta_2 \cdots \sin\theta_{n-1}$ ， θ_i ， $1 \le i \le n-1$ 表示向量 v_{i+1} 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， $\|v_1\|\|v_2\|\cdots\|v_n\|$ 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（商 R^n / L 的基础域）的集合可定义为: $F(L) = \{t_1 v_1 + t_2 v_2 + \cdots + t_n v_n, 0 \le t_i < 1\}$ ； 格 L 的行列式或体积可定义为：

$$Disc(L) = Volume(F(L)) = \det(v_1 \mid v_2 \mid \cdots \mid v_n) = \prod_{i=1}^{n} \|v_i^*\| \tag{2-2}$$

Hadamard 不等式：假设 L 是一个格，对于它的任意一个基 $v_1, v_2, \cdots, v_n$ 和基础域 F 来说，存在 $\det L = vol(F) \le \|v_1\| \|v_2\| \cdots \|v_n\|$。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。

正交缺陷度（Orthogonal Defect）:用于衡量一组基 $b_1, b_2, \cdots, b_n$ 的正交化程度：

$$OD(b_1, b_2, \cdots, b_n) = \frac{\|b_1\| \|b_2\| \cdots \|b_n\|}{\det(L)} \tag{2-3}$$

2.2 格基规约算法

2.2.1 LLL 算法

格雷姆-施密特正交化（GSO）是把 R^m（实数域上 m 维）线性空间中的任意一组基转换成一组正交基，在初级线性代数中是一个基本的问题，之后把正交化的思想应用到 LLL 算法中。设 $v_1, v_2, \cdots, v_n$ 是 R^m 中的一个基，则它的 GSO 基为 $v_1^*, v_2^*, \cdots, v_n^*$，即：

$$\begin{aligned} &v_1^* = v_1, \\ &v_i^* = v_i - \sum_{j=1}^{i-1} u_{ij} v_j^* (2 \le i \le n), u_{ij} = v_i \cdot v_j^* / v_j^* \cdot v_j^* (1 \le j < i \le n) \end{aligned} \tag{2-4}$$

详细过程可见[42]中。GSO 的基向量 $v_1^*, v_2^*, \cdots, v_n^*$ 通常不在向量 $v_1, v_2, \cdots, v_n$ 生成的格中，因为 GSO 的过程不是 $v_1, v_2, \cdots, v_n$ 这 n 个向量的整数倍的线性组合。基于此特点，通常用格中与 v_i^* 距离最近的格中的某个向量来代替 v_i^*，这样所求的格基就非常接近正交，可称此过程为准正交化。

设 $u_{ii} = 1, 1 \le i \le n$，可得：

$$v_i = \sum_{j=1}^{i} u_{ij} v_j^* \tag{2-5}$$

把 $v_i = (v_{i1}, \cdots, v_{in})$ 写成矩阵 $V = (v_{ij})$ 中每一行的向量 v_i 形式，同理 $V^* = (v_{ij}^*)$。设 $u_{ij} = 0, 1 \le i < j \le n$，可得矩阵等式如下：$V = MV^*, M = (u_{ij})$，矩阵 M 是 $u_{ii} = 1, 1 \le i \le n$ 的下三角矩阵，因此也可得 $V^* = M^{-1}V$。

上述准正交化的结果通常可以使向量之间最接近正交，向量的乘积达到最小，但不能保证正交化后的这组基中，每一个向量都相对较短。因此，为了使基中的每一个向量都相对较短，可通过宽限正交化的程度，从而对向量的长度做出一定的条件限制，此过程可称为 LLL 格基规约的基本思想。

如果给定一组基，若满足以下两个条件：

（1）$\left|\mu_{ij}\right| \le 1/2, 1 \le j < i \le n$；

（2）$\left\|b_i^* + \mu_{i,i-1} b_{i-1}^*\right\|^2 \ge \delta \left\|b_{i-1}^*\right\|^2, 2 \le i \le n$ （2-6）

则称这组基为 LLL 约减基。其中约减参数 $\delta(1/4 < \delta < 1)$ 是一个实数，通常取其值为 0.75。条件（1）的主要作用是使得向量之间两两“几乎正交”，条件（2）是对向量之间的长度进行限制，如果不满足条件，就交换向量之间的位置，其主要作用是重新计算 GSO 生成一个更短的向量，条件（2）又可被写成：

（2’）$\left\|b_i^*\right\|^2 \ge (\delta - \mu_{i,i-1}^2) \left\|b_{i-1}^*\right\|^2, 2 \le i \le n$ （2-7）

定义 1：假定 $b_1, b_2, \cdots, b_n \in R^m$ 是一组参数为 δ 的约减基，$\alpha = 1/(\delta - 1/4)$，可得 $\left\|b_i^*\right\|^2 \le \alpha^{j-i} \left\|b_j^*\right\|^2, 1 \le i \le j \le n$。

定义 2：假定 $b_1, b_2, \cdots, b_n \in R^m$ 是一组参数为 δ 的约减基，$\alpha = 1/(\delta - 1/4)$，可得 $\left\|b_1\right\| \le \alpha^{(n-1)/2} \lambda_1(L)$，$\lambda_1(L)$ 表示格中的最短向量。

定义 3：假定 $b_1, b_2, \cdots, b_n \in R^m$ 是一组参数为 δ 的约减基，$\alpha = 1/(\delta - 1/4)$，可得 $\left\|b_1^*\right\|^{2n} \le \alpha^{n(n-1)/2} (\det(L))^2$。

2.2.2 分块规约算法

（1）分块 LLL 算法

分块 LLL 约减是 LLL 算法的一个变体，约减结果与 LLL 相比较弱，但提高了约减速度。分块 LLL 的约减思想是：给定一个维度为 n 的基 $b_1, b_2, \cdots, b_n$，$n = km$ 表示把给定的基划分成 m 块，每块大小为 k 个连续的基向量。两个连续块的局部约减使用 $2k \times 2k$ 的子矩阵 R_l，对应于两个连续的块 B_{l-1}, B_l。每块大小为 k 约减基的目的是使得全局的代价最小化。

定义：我们称一组有序基 $b_1, b_2, \cdots, b_n \in Z^d$ 为大小等于 k 的分块 LLL 约减基，其中 $\delta \in (1/4, 1]$，$(n = km)$，如果它是 Size 约减，且满足以下三个条件，其中 $\alpha = 1/(\delta - 1/4)$：

（1）$\delta \left\|b_i^*\right\|^2 \le \left\|b_{i+1}^*\right\|^2 + u_{i+1,i}^2 \left\|b_i^*\right\|^2, i \ne 0\%k$；

（2）$D(l) \le (\alpha/\delta)^{k^2} D(l+1), l = 1, \cdots, m-1$；

（3）$\delta^{k^2} \left\|b_{kl}^*\right\|^2 \le \alpha \left\|b_{kl+1}^*\right\|^2, l = 1, \cdots, m-1$ （2-8）

其中 $D(l) = \left\|b_{k(l-1)+1}^*\right\|^2 \cdots \left\|b_{kl}^*\right\|^2$ 表示块 B_l 的局部格兰姆行列式。

（2）BKZ 算法

设$b_1,\cdots,b_m$是格$L=L(b_1,\cdots,b_m)\subset R^n$的一组有序基。$Span(b_1,\cdots,b_{i-1})$为由$b_1,\cdots,b_{i-1}$组成的向量空间，$\pi_i:R^n\to Span(b_1,\cdots,b_{i-1})^{\perp}$表示垂直映射，可得$b-\pi_i(b)\in Span(b_1,\cdots,b_{i-1})$。令$L_i$表示格$\pi_i(L)$，即将格$L$投影到与前$i-1$个向量垂直空间生成的新格，其秩为$m-i+1$。

格L的一组有序基$b_1,\cdots,b_m$是一个KZ基，如果它是Size约减，同时满足$\left\|b_i^*\right\|=\lambda_1(L_i),i=1,\cdots,m$。每一个$KZ$基[43]都满足$4/(i+3)\le\left\|b_i\right\|^2/\lambda_i^2\le(i+3)/4,i=1,\cdots,m$。

Schnorr[9]引进了BKZ约减基，设β为一个整数，$2\le\beta<m$。一个格基$b_1,\cdots,b_m$为β的约减基，如果它为Size约减，同时满足$\left\|b_i^*\right\|\le\lambda_1(L_i(b_1,...,b_{\min(i+\beta-1,m)}))$，$i=1,\cdots,m-1$。

格L的每一个β约减基$b_1,\cdots,b_m$满足$\left\|b_1\right\|^2\le\alpha_\beta^{(m-1)/(\beta-1)}\lambda_1(L)^2$，如果$\beta-1$可除$m-1$。我们称基$b_1,\cdots,b_m$为参数$\delta,1/4<\delta\le 1$的$\beta$约减基，如果它为Size约减，同时满足$\delta\left\|b_i^*\right\|^2\le\lambda_1(L_i(b_1,...,b_{\min(i+\beta-1,m)}))^2$，$i=1,\cdots,m-1$。

2.3 格基规约应用的相关算法

2.3.1 球译码算法（SDA）

整数最小二乘问题（ILSP）可应用在许多领域，例如：通信[44]，密码学[45]，GPS[46]以及许多工程上等问题。球译码算法是解决整数最小二乘问题的一种有效方法。ILSP的定义如下：

$$\min_{x\in Z^n}\left\|Ax-y\right\|_2^2,A\in R^{m*n},m\ge n,y\in R^m \qquad (2\text{-}9)$$

假定A是列满秩矩阵，即A的列向量是一组基，x是整数，Ax是这组基整数倍的线性组合，它们组成了一个格空间，目的就是找到一个与y向量最相近的格点。

在线性代数中，矩阵的QR分解是将矩阵A分解成$A=QR$的乘积，Q是正交矩阵，R是上三角矩阵。QR分解通常被应用到求解线性最小二乘问题（LLSP），针对这组基的QR分解方法有特征值法，QR分解法等。

通常情况下，我们将矩阵A用QR分解来转换，整数最小二乘问题可被约减成上三角整数最小二乘问题：

$$\min_{x\in Z^n}\left\|Rx-\hat{y}\right\|_2^2 \qquad (2\text{-}10)$$

对于上三角整数最小二乘问题，首先，在实数域中解决线性方程 $Rx=\hat{y}$，可得一个实数解 x。然后四舍五入向量中每一子项值，可得到一个解称为 Babai 估计[47]。但是 Qiao[48]指出 Babai 估计与实际的解之间有一个差距。M.Ajtai[49]提出在整个格空间中求解 ILSP 是一个困难问题。之后 Phost[50]考虑了球译码算法；Fincke 和 Phost 提出 Fincke-Phost 球译码算法，并解释其期望的时间复杂度是多项式时间[51]。2015 年，Li 和 Chen[52]提出基于任意维度自适应半径的减少复杂度的球译码算法，针对解决极大似然问题提出 3 个提高算法的方案。2016 年，Karthikeyan 和 Saraswady[53]提出使用基于 Schnorr-Euchner 的枚举概率阈值来减少球译码算法的时间复杂度，主要应用在 MIMO 无线通信系统中。

2.3.2 0-1 整数背包问题求解算法

背包问题也称为子集合问题，被证明是 NPC 问题，即首先是一个 NP 问题，且如果所有 NP 问题都能在多项式的时间内转化成某个 NP 问题，则称该 NP 问题为 NPC 问题；可在多项式时间内验证一个解是否正确的问题称为 NP 问题；可在多项式时间内解决的问题（存在多项式时间算法的一类问题）称为 P 问题；满足 NPC 问题定义的第二条，但不一定满足第一条的称为 NP-Hard 问题。原则上，它通常被应用到密码系统中。给定一个整数序列 $(a_1,a_2,\cdots,a_n),a_i>0$， S 是一个非负整数满足：

$$S=\sum_{i=1}^{n}x_i\cdot a_i,x_i\in\{0,1\} \tag{2-11}$$

找二进制序列 $(x_1,x_2,\cdots,x_n)$ 的过程被称为背包问题，这个问题是 NP 完全。

1978 年，Merkle 和 Hellman 把背包问题引入到密码系统领域，主要考虑：在序列是超递增的情况下，即：

$$a_i>\sum_{j=1}^{i-1}a_j,1\le i\le n \tag{2-12}$$

这个背包被称为简易背包，可容易在多项式内解决。1981 年，Schroeppel、Shamir[54]提出了一个算法，该算法可在时间复杂度为 $O(2^{n/2})$，空间复杂度为 $O(2^{n/4})$ 的情况下求解简易背包问题。然而，该问题通常不能在多项式下被解决。之后，Merkle 和 Hellman 的密码系统被 Shamir 使用 Lenstra 的整数规划破译。1982 年，Lenstra，Lenstra，Lovasz[9]提出了经典的 LLL 格基规约算法。之后，Lagarias 和 Odlyzko[55]表明把特定的背包问题（密度为 $d<0.64$）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 的攻击被 Coster 等人[56]进一步提高到背包密度为$d<0.94$。如果在格中能找到最短非零向量，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。

1996 年，Impagliazzo 和 Naor[57]以密度为标准划分背包问题，它的定义如下：

$$d=\frac{n}{\log_2(\max_i a_i)},1\le i\le n \qquad (2\text{-}13)$$

背包密度d的定义可理解为：背包$(a_1,a_2,\cdots,a_n)$的长度与背包中最大元素的二进制长度的比值。如果$d<1$，背包可被应用于密码系统，而$d>1$时，背包通常被应用于哈希意图。然而，密度接近于 1 的n维背包问题是最困难的，这些问题对于任意n维来说是 NP-hard，解决这类问题要么通过优化生日算法，要么通过格基规约。2009 年，基于 Schroeppel 和 Shamir 的算法，出现了解决困难背包问题的简单但启发变体的算法，有同样的时间复杂度$O(2^{n/2})$和空间复杂度$O(2^{n/4})$。2010 年，HGJ [58]在生日算法的基础上提出两个新的生成算法来解决密度接近于 1 的困难背包问题，这些算法可以减少运行时间到$O(2^{0.337n})$，同时保持一个低存储空间为$O(2^{0.256n})$。2011 年，BCJ [59]又提出了生成算法的提高版，其约减时间为$O(2^{0.291n})$。2012 年底，Schnorr 和 Shevchenko[60]提出通过优化的 BKZ 约减来解决密度接近于 1 的随机背包问题。由此可见，格基规约求解困难背包问题的能力越来越强。

2.4 本章小结

格是数学中的概念，格基规约的思想起源于十八世纪初。如今，格基规约算法被应用到不同领域，它的优化以及创新版本不断被提出，验证了它的发展历程。本节主要介绍了格中的基本知识以及格基规约算法中的相关概念，为了进一步了解它的相关内容，可参考如下文献[61]。

本章除了介绍格中的相关概念外，还引进了格基规约的经典算法 LLL 及分块算法 BKZ，同时介绍了格基规约算法可以应用解决的问题：用于球译码算法的初始化部分以及求解困难背包问题。

第 3 章 格基规约算法的改进

3.1 引言

基于格的密码学应用都是希望通过格基规约来找到最短的非零格向量。因此，格中最短非零向量的欧几里德范数被当作衡量算法质量的一个重要指标。

LLL 算法的优化版本很多，有几种非常相似。如 deepLLL[12], 若条件（2）成立，则 $k=k+1$ 继续下两个相邻向量之间的比较；若不成立，$k=k-1$ 当前向量与相隔一个的前一个向量进行判断，如果还是不成立，当前向量依旧与更前一个向量进行判断比较，以此类推直到 $k=1$ 为止，若成立，就把第 k 个向量插入到与之比较的向量位置，而与之比较的向量及之后的向量保持不变的次序依次后移一位，k 就更新成插入的相应位置的值。PotLLL[18]由 Fontein、Schneider 和 Wanger 在 2014 年提出，其思想是：当前向量与前一个向量进行条件（2）的判断，若成立，当前向量与相隔一个的前一个向量继续进行判断，若成立，一直向前回溯判断直到 $k=1$，若不成立，把第 k 个向量插入到条件不成立向量的前一个位置处，k 的值更新成插入相应位置的值。2016 年，Espitau 和 Joux[19]提出了应用区间运算的自适应精确的 LLL 和 PotLLL 算法，主要变换就是把浮点运算更新成区间运算的方式，其运行速度更快。而 BKZ[13]是分块的 deepLLL 算法。l 次规约[17]与前两种变体思想相比的最大好处就是，l 的值是动态控制的，可以设置 1 到 n（向量个数）之间的任何值，$l=1$ 属于 LLL 约减，$l>1$ 属于深度回溯规约。值越大，约减效果越好，但有些好基已经足够约减，并不用回溯到 $k=1$ 而浪费大量时间，仅需简单回溯约减。因此，动态控制回溯次数的值可以清晰地比较约减基的变化。

基于前人的研究，论文提出一个汲取 LLL 变体优点的算法。通过 l 次约减，可获得更短的约减基；分块 LLL 规约，可花费更少的时间。因此，在分块 LLL 的思想上应用 l 次规约，即分块 l 次格基规约，所提想法的难点在于如何有效确定分块大小 k 的值以及回溯规约 l 的值，才能保证获得较好的约减效果，结论见下文。大量实验表明，所提算法可在获得更短约减基的同时，减少一定的运行时间，维数越大，效果越明显。

3.2 提出的规约算法

3.2.1 格基规约

定义：假定格 L 的一组基 $b_1,b_2,\cdots,b_n\in Z^d$，$n=km$，$k$ 是分块大小，m 是块数；$\delta\in(1/4,1]$，$\alpha=1/(\delta-1/4)$，$D(loc)=\left\|b^*_{k(loc-1)+1}\right\|^2\cdots\left\|b^*_{k\cdot loc}\right\|^2$，如果满足以下四个条件，则称这组基为分块 l 次规约基。

（1）$\left|u_{ij}\right|\le 1/2, 1\le j<i\le n$；

（2）$\delta\left\|b^*_{i-r}\right\|^2\le\left\|b^*_i\right\|^2+u^2_{i,i-r}\left\|b^*_{i-r}\right\|^2$，for $i\ne 1\bmod k$，$1\le r\le l$，$1\le l\le k-1$；

（3）$D(loc)\le(\alpha/\delta)^{k^2}D(loc+1)$，for $loc=1,\cdots,m-1$；

（4）$\delta^{k^2}\left\|b^*_{k\cdot loc}\right\|^2\le\alpha\left\|b^*_{k\cdot loc+1}\right\|^2$，for $loc=1,\cdots,m-1$。

格兰姆行列式 D_{kl} 是前 l 个局部块行列式的乘积 $D_{kl}=D(1)\cdots D(l)$，loc-l reduction(l) 表示两块 B_{l-1},B_l 的局部 l 次约减，它仅仅改变 $D_{k(l-1)}$。因此，两个块作为一个整体，调用局部 l 次格基规约算法：loc-l reduction(l)；调用返回之后，对比第 l 和第 $l-1$ 块，第 $k(l-1)$ 和第 $k(l-1)+1$ 两个向量。如果满足 $D(l-1)>(\alpha/\delta)^{k^2}D(l)$ 条件，交换两个块并且令 $l=l-1$；如果满足 $\delta^{k^2}\left\|b^*_{k(l-1)}\right\|^2>\alpha\left\|b^*_{k(l-1)+1}\right\|^2$ 条件，交换两个相邻向量并且令 $l=l-1$。只有这两个条件都不满足时，才执行 $l=l+1$。

分块部分控制了整个算法的执行过程。在分块算法内部，应用局部算法 loc-l reduction(l) 进行条件（1）和条件（2）的判断，首先进行的是相邻两个向量之间的判断，如果满足条件，根据参数 l 的值，继续进行 l 次回溯规约。当运行完局部 l 次规约后，回到分块的主算法中，对 l 次局部规约的块，进行条件（3）和条件（4）的判断，条件（3）是判断两个相邻的小块之间的大小，条件（4）是判断两个小块之间的相邻向量的大小，只有条件（3）、（4）同时满足，规约回到程序的分块主算法处，继续进行下一个相邻块之间的约减。

3.2.2 规约算法

分块 LLL[15]算法是 LLL 的有效变体，基于此约减，把分块中的 LLL 算法用 l 次规约[17]来实现，也可以理解成在 l 次规约的算法中加入分块约减的思想。算法 3.1 利用分块约减的优势展示了主算法。它主要负责划分块的大小以及遍历块，并且管理每一块以及相邻块之间向量的关系。

算法中标号 1，2 的条件如果成立，继续往下执行相应的语句。分块 l 的约减思想是用 l 次规约算法运算向量个数为 $2k$ 的分块向量。明显地，分块 l 可获得向量个数为 $2k$ 的 l 次约减基。分块 l 的主要工作是由 Loc-l reduction$(segment-1, segment)$，两个相邻的子块执行的全局总开销。

算法 3.1 分块规约的主算法

输入： $b_1,\cdots,b_n \in Z^d, k, m, n=km, \delta$ 注：k 为分块大小，m 为块数，n 为线性无关的向量个数，δ 为参数

输出： $b_1,\cdots,b_n$ 是分块 l 次规约基

1. segment=2;
2. while segment<=m do
3. [b, b^*]=Loc-l reduction $(b_{1:n,k*segment-2*k+1:k*segment})$;注：返回对应的约减基 b 和施密特正交化 b^*
4. if $(D(segment-1) \le (\alpha/\delta)^{k^2} D(segment)) \&\& (\delta^{k^2} \left\|b^*_{k(segment-1)}\right\|^2 \le \alpha \left\|b^*_{k(segment-1)+1}\right\|^2)$ then
5. segment=segment+1;
6. else
7. if $(D(segment-1) > (\alpha/\delta)^{k^2} D(segment))$
8. 交换两块 $(B(segment), B(segment-1))$；
9. $segment = \max(segment-1, 2);$
10. else
11. 交换两列 $(b_{k(segment-1)}, b_{k(segment-1)+1})$；
12. $segment = \max(segment-1, 2);$
13. end if
14. end if
15. end while

算法 3.2 局部 l 块约减算法 Loc-l Reduction Algorithm

输入： 一组格基 $b_1,\cdots,b_n \in Z^m$，约减参数 $\delta(1/4 < \delta \le 1)$，回溯规约次数 l

输出： 带参数 δ 的 l 次规约基

1. $k=2$;
2. while $(k \le n)$ do
3. 用施密特正交化计算正交化系数 u_{ki} 和正交向量的平方 $\left\|b_i^*\right\|^2$，$i=1,2,\cdots,k-1$;
4. for $(j=1; j<k; j++)$ do
5. if $(|u_{kj}|) > 1/2$ then
6. $u_{kj} = \lceil u_{kj} - 1/2 \rceil$;
7. else
8. $u_{kj} = 0$;
9. end if
10. end for
11. $b_k = b_k - \sum_{j=1}^{k-1} u_{kj} b_j$;
12. 用施密特正交化重新计算正交化系数 u_{kj} 和正交向量的平方 $\left\|b_j^*\right\|^2$，$j=1,2,\cdots,k-1$;
13. Stepk=k; sl=1; 保存当前的规约位置 标记当前规约等级，回溯层次
14. [b, k]=ReduceCheck $(b, b^*, u, Stepk-sl, Stepk)$;表示 LLL 约减
15. while $((k == Stepk+1) \&\& (sl < l))$ do
16. if $(Stepk-sl) < 1$ then
17. break;
18. else
19. sl++;
20. [b, k]=ReduceCheck $(b, b^*, u, Stepk-sl, Stepk)$;表示 l 次规约，但还没有达到 l 层
21. end if
22. end while
23. end while

算法 3.2 是局部 l 次约减算法，标号 1 到 15 实现 LLL，即 $l=1$ 的情况；当 $l>1$ 时，标号 16 到 23 进行深度约减。算法 3.3 是算法 3.2 中调用 l 次规约的一个子进程。在深度约减的过程中，如果满足标记 5 的条件，则把 b_j 插入到 b_i 的前面；令 $k=\max(i,2)$，并重新约减 $b_1,\cdots,b_j,b_i,\cdots,b_{j-1},b_{j+1},\cdots,b_n$。依次递归执行，可获得更短的基向量。

算法 3.3 l 次规约的子例程

输入：int ReduceCheck(b,b^*,u,i,j)

输出： 规约矩阵 b，规约位置 k

```
1. if (i ≥ j) then
2.    k=j;error;
3. end if
4. if (δ‖b_i^*‖^2 > ‖b_j^*‖^2 + u_ji^2 ‖b_i^*‖^2) then
5.    把 b_j 放到 b_i 之前;
6.    k = max(i,2);
7. else
8.    k = j+1;
9. end if
```

3.2.3 算法的流程图

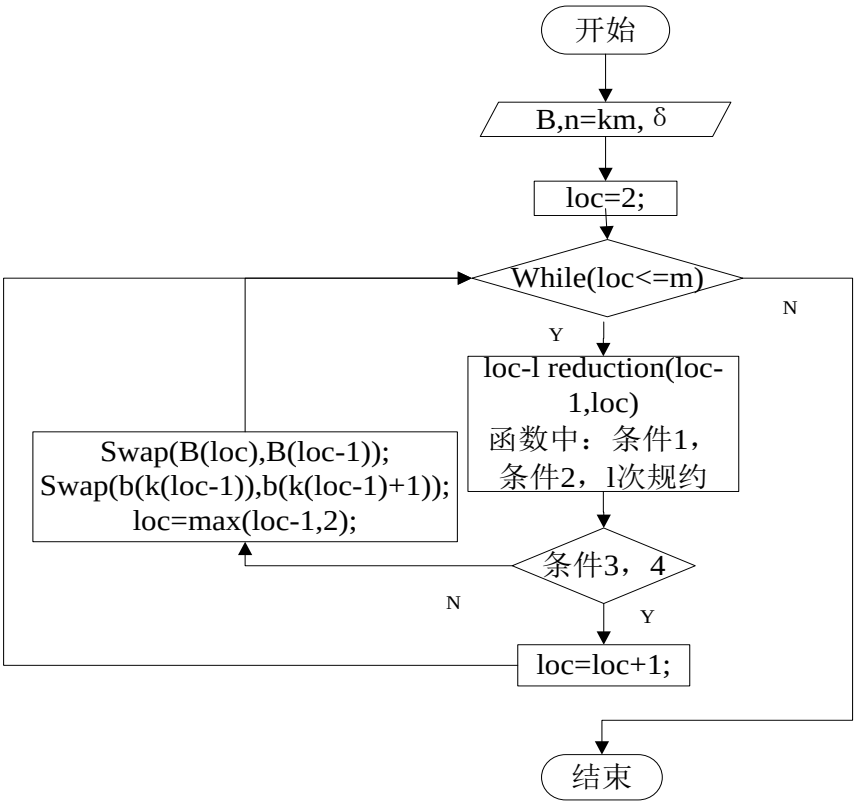

图 3.1 分块 l 主程序的流程图

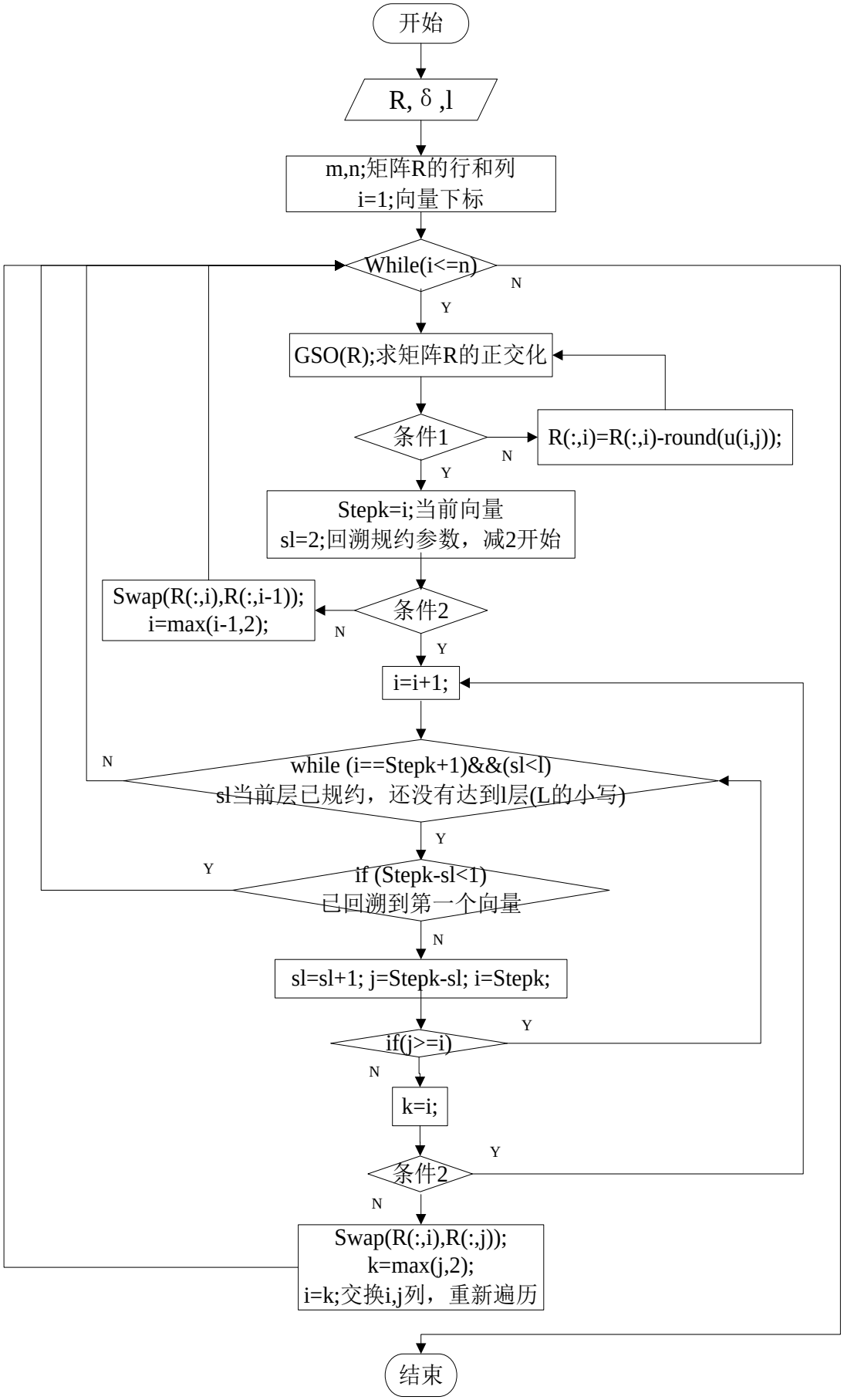

图 3.2 局部 l 次规约的流程图

从图 3.1 中可知，while 循环控制了程序的整个流程，主要控制块之间的约减关系。当条件成立时，调用局部 l 次规约算法。在此规约算法中，主要进行条件 1，条件 2 的判断与规约。而该分块算法中，主要判断和约减条件 3、条件 4。当条件 3 不满足时，交换两个相邻的块，且规约位置回退到 $loc=\max(loc-1,2)$，并重新规约相邻的块；当条件 4 不满足时，交换两个块之间相邻两个向量的位置，且规约位置回退到 $loc=\max(loc-1,2)$，并重新规约相邻的块。直到条件 3 和条件 4 同时满足条件时，令 $loc=loc+1$，并继续规约后面两个相邻的块，直到 while 循环不成立，则退出程序。

从图 3.2 中可知，描述了局部 l 次规约的详细流程。While 循环控制着两个相邻块中向量之间的规约关系。首先对向量之间进行施密特正交化的求解，获得正交化系数和正交化向量，进而进行条件 1 的判断。若条件 1 不满足，进行适当的向量调整，然后重新求解判断；若条件 1 满足，继续条件 2 的判断，若条件 2 不满足，交换两个向量之间的位置，回退到前一个向量的位置，从条件 1 再重新开始上述的判断。若条件 2 满足，进到下一个 while 循环中，此 while 循环是用来判断 l 次规约的。

3.2.4 复杂度分析

改进算法可在获得更短约减基的同时，减少一定的运行时间。针对 l 次规约，l 值越大，约减效果越好，但执行时间大幅度增加。分块 LLL 与 LLL 规约相比，从整体来说，约减时间减少，但规约效果略差；如果 k 取合适的值，如 $k=n/4$ 时，可获得与 LLL 算法的约减效果非常接近，同时运行时间大幅度降低。因此，分块与 l 次规约结合，可以在获得更短约减基的同时，获得比 l 次规约更好的结果，实际效果更加可观。

在维度 n 越高的情况下，算法需要大量的运行时间。下面两种假设可以适当地减少运行时间。针对 l 次规约算法，可通过适当减小 δ 的值来减少迭代次数以至于花费更少的时间，由于放松高维约减的限制，因此约减基向量的长度比 LLL 约减效果略差；其次，针对分块约减，可通过有效控制分块大小 k 的值，k 越大，所需运行时间越多，但约减效果越好，k 越小时，则与上述情况相反。

3.3 算法仿真

3.3.1 数据的选取

通过大量的实验测试约减算法，实验首先选择了维度为 10 的一个随机格，元素是 0 到2^8范围内的随机整数。通过比较参数l不同时、k不同时、l与k不同时，k与δ不同时，规约效果的变化情况。其次，为了让实践更具说服力，选取维度为 64 的随机格，元素范围为[1，256]，在 LLL 约减，l次约减，分块 LLL 约减以及新算法之间比较实际性能，以此来证明理论分析的准确性。下面列举的数据，每一列作为格中的一个列向量。由于$\det(L)$不为零，所以可以构成一个格基，格L中的一组基被列举如下：

$$
\begin{bmatrix}
12 & 124 & 98 & 52 & 63 & 9 & 93 & 249 & 30 & 70 \\
37 & 96 & 245 & 173 & 223 & 130 & 138 & 78 & 143 & 60 \\
23 & 152 & 47 & 31 & 90 & 29 & 164 & 74 & 224 & 114 \\
183 & 65 & 201 & 192 & 24 & 68 & 125 & 107 & 64 & 98 \\
171 & 109 & 154 & 86 & 157 & 218 & 139 & 146 & 135 & 142 \\
58 & 253 & 251 & 254 & 232 & 62 & 119 & 163 & 20 & 177 \\
114 & 186 & 30 & 125 & 229 & 228 & 229 & 66 & 36 & 239 \\
99 & 10 & 233 & 108 & 110 & 78 & 76 & 74 & 209 & 177 \\
95 & 29 & 101 & 53 & 165 & 79 & 129 & 181 & 110 & 233 \\
182 & 204 & 8 & 163 & 4 & 142 & 3 & 24 & 239 & 56
\end{bmatrix}
$$

3.3.2 l不同的约减比较

接下来，l次规约算法，通过更改参数l来比较。假定回溯约减参数l从 1 变化到 9 时，比较规约时间以及约减基的变化，上述格基的 Hadamard 比率为 0.4248。

表 3.1：参数l不同时的规约时间和约减基

算法	LLL	l次规约							
	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$	$l=7$	$l=8$	$l=9$
规约时间	0.0190	0.0230	0.0220	0.0230	0.0250	0.0120	0.0240	0.0270	0.0130
Hadamard	0.8595	0.8595	0.8253	0.8331	0.8331	0.8331	0.8331	0.8331	0.8331
约减基范数	199	199	201	177	177	177	177	177	177
	177	177	198	187	187	187	187	187	187
	187	187	177	199	199	199	199	199	199
	200	192	187	192	192	192	192	192	192
	198	198	199	218	218	218	218	218	218
	192	200	208	200	200	200	200	200	200
	220	220	232	220	220	220	220	220	220
	220	220	220	220	220	220	220	220	220
	254	254	298	320	320	320	320	320	320
	315	315	384	315	315	315	315	315	315

表 3.1：通过改变参数l，来比较l次规约算法的约减效果。规约时间表示算法的执行时间，Hadamard 比率表示规约基的正交程度，规约基范数列出约减基每一列的长度，为了容易比较长度变化的大小，特意省略精度数，本章以下表格约减长度均省略精度。表中列出在l不同条件下，整个约减基的变化情况。LLL 算法本身就是约减基的正交性与约减向量的长度相互妥协的一种规约算法。l次规约在 LLL 算法的基础上，进行回溯迭代。由于迭代次数不断增加，因此规约时间也相应增加，但约减基的效果更好，即约减基的范数更短。

从表中可知，当l为 2、3、4 时，约减效果不断变好。而当l为 5-9 时，后面几乎没变，此情况可看成，规约效果已达到相对约减。$l=3$时的约减效果似乎没有更约减，可看成是局部不规则的情况所造成的。这种现象本身随着l次格基约减计算时间的规律性以及数据的缓存性，可看成是不矛盾的。因此，此表有力地解释了l次规约通过牺牲运行时间来换取更好规约基的特性。

3.3.3 k不同的约减比较

我们知道l次约减基通常比 LLL 约减基更短，但运行时间是增加的。因此，为了减少运行时间，我们再分析一下分块规约的思想。下面，同样利用上述格基矩阵来测试理论分析。

表 3.2：分块大小k不同时的规约时间和约减基

算法	LLL	分块 LLL		
	$l=1$	$k=1$	$k=2$	$k=5$
规约时间	0.0190	0.0150	0.0130	0.0180
Hadamard	0.8595	0.5596	0.6850	0.8595
约减基范数	199	364	240	199
	177	326	261	177
	187	251	198	187
	200	297	251	200
	198	293	267	198
	192	244	294	192
	220	397	260	220
	220	360	300	220
	254	370	301	254
	315	456	364	315

表 3.2 列出了分块大小k不同时，运行时间和约减基的变化。与 LLL 算法相比，分块 LLL 是减少了运行时间，但约减基略差。而分块 LLL 的规约效果和k本身的大小有一定的关系。由表可知，k值越大，格基就越短，运行时间则越多。通常情况下，

$k \gg m$的效果好，大量实验表明$k = n/4$时相对较好。由于此处维数有限，在 3.3.5 小节算法之间比较时，会很容易看出效果。分块 LLL 是通过牺牲约减效果来获得更短的运行时间。

表 3.3：k 与l 取不同值时的运行时间、约减效果

$k \backslash l$	2	3	4	5	6	7	8	9
2 时间	0.0140	0.0090	0.0200	0.0150	0.0180	0.0240	0.0210	0.0190
Hadamard	0.6850	0.6850	0.6850	0.6850	0.6850	0.6850	0.6850	0.6850
约减基	240	240	240	240	240	240	240	240
	261	261	261	261	261	261	261	261
	198	198	198	198	198	198	198	198
	251	251	251	251	251	251	251	251
	267	267	267	267	267	267	267	267
	294	294	294	294	294	294	294	294
	260	260	260	260	260	260	260	260
	300	300	300	300	300	300	300	300
	301	301	301	301	301	301	301	301
	364	364	364	364	364	364	364	364
5 时间	0.0150	0.0190	0.0220	0.0220	0.0330	0.0390	0.0450	0.0240
Hadamard	0.8595	0.8253	0.8331	0.8331	0.8331	0.8331	0.8331	0.8331
约减基	199	201	177	177	177	177	177	177
	177	198	187	187	187	187	187	187
	187	177	199	199	199	199	199	199
	192	187	192	192	192	192	192	192
	198	199	218	218	218	218	218	218
	200	208	200	200	200	200	200	200
	220	232	220	220	220	220	220	220
	220	220	220	220	220	220	220	220
	254	298	320	320	320	320	320	320
	315	384	315	315	315	315	315	315

表 3.3 比较了分块l 次规约算法在分块大小k 与回溯规约次数l 取不同值时的约减效果。$k = 2$时，由于l 从 2 增加到 9，运行时间呈现递增状态。又由于k 越大越好，因此，如果值太小了，约减效果的变化会不明显。而当$k = 5$时，随着l 的增加，运行时间整体呈现上升趋势，约减基的效果比较明显，二者整体上都保持一定的变化趋势。虽然会出现局部不规则现象，主要是因为使用随机生成的格，且l 次规约需要不断的交换向量之间的位置。因此，对一组基使用不同的l，基向量的顺序就会有一些差异，从而会出现局部无规律现象，该现象与整体的规律性并不矛盾。所以，当l 从 2 到 3 时，约减效果就很容易理解。

3.3.4 δ 不同的约减比较

表 3.4 列举了分块l次规约算法在分块大小k与规约参数δ取不同值时的运行时间和约减基的变化情况，其中$l=5$。从表中可知，参数δ越大，运行时间就越大，但约减效果越好。块数k越大，运行时间也增大，约减效果越好。当$\delta=0.75,k=5$时，约减基的效果最好，其它情况则通过改变这两个参数来对比约减基的变化效果。

表 3.4：k与δ取不同值时的运行时间、约减效果

$k\backslash\delta$	0.35	0.55	0.75
1 规约时间	0.0170	0.0170	0.0190
Hadamard	0.5176	0.5596	0.5596
约减基范数	364	364	364
	326	326	326
	251	251	251
	446	297	297
	297	293	293
	397	244	244
	244	397	397
	392	360	360
	381	456	370
	477	370	456
2 规约时间	0.0210	0.0180	0.0240
Hadamard	0.6220	0.6850	0.6850
约减基范数	240	240	240
	261	261	261
	198	251	198
	265	198	251
	364	267	267
	278	294	294
	201	260	260
	221	364	300
	456	300	301
	425	301	364
5 规约时间	0.0210	0.0240	0.0280
Hadamard	0.6918	0.8439	0.8331
约减基范数	240	198	177
	261	199	187
	199	217	199
	289	177	192
	198	198	218
	198	200	200
	220	225	220
	232	220	220
	457	254	320
	373	327	315

3.3.5 不同算法的约减比较

为了使得不同算法之间的约减效果更加明显，特选取维数为 64 的一个随机格来比较。由于维数太大，仅列出具有代表性的 30 列约减向量范数。

表 3.5：不同算法之间的比较

算法	LLL	BKZ							
		$k=4$	$k=8$	$k=12$	$k=24$	$k=32$	$k=48$	$k=54$	$k=62$
规约时间	3.6480	10.4690	11.8970	13.0490	16.8760	19.3520	22.7810	23.9540	24.6420
Hadamard	0.0450	0.2947	0.4055	0.4272	0.4272	0.4272	0.4272	0.4272	0.4272
约减基范数	882	882	882	882	882	882	882	882	882
	789	789	789	789	789	789	789	789	789
	718	718	718	718	718	718	718	718	718
	792	792	792	792	792	792	792	792	792
	666	666	666	666	666	666	666	666	666
	709	709	709	709	709	709	709	709	709
	867	702	702	702	702	702	702	702	702
	864	740	740	740	740	740	740	740	740
	805	805	805	805	805	805	805	805	805
	834	834	834	834	834	834	834	834	834
31 列	843	831	831	831	831	831	831	831	831
	819	776	776	776	776	776	776	776	776
	880	811	811	811	811	811	811	811	811
	844	829	829	829	829	829	829	829	829
	841	789	789	789	789	789	789	789	789
	892	787	787	787	787	787	787	787	787
	835	773	773	773	773	773	773	773	773
	3042	785	785	785	785	785	785	785	785
	2733	742	742	742	742	742	742	742	742
	4838	949	949	949	949	949	949	949	949
55 列	3391	1142	1142	1142	1142	1142	1142	1142	1142
	5615	950	950	950	950	950	950	950	950
	32070	865	865	865	865	865	865	865	865
	37168	1012	989	989	989	989	989	989	989
	25640	1032	1032	1032	1032	1032	1032	1032	1032
	18180	1636	1788	1004	1004	1004	1004	1004	1004
	29762	1951	954	954	954	954	954	954	954
	42318	3038	1185	858	858	858	858	858	858
	6797	1441	997	997	997	997	997	997	997
	14942	7418	1961	1018	1018	1018	1018	1018	1018

为了便于衡量算法的效率，采用同样的 64 维矩阵，下表列出多种不同算法之间的约减效果，每种算法的取值情况都给予明确标注，由于篇幅有限，仅列出 64 维约减矩阵中的前 30 列进行比较。

表 3.6：不同算法之间的比较

算法	LLL	l	Segment LLL					Segment l (new)	
	$l=1$	$l=2$	$k=2$	$k=4$	$k=8$	$k=16$	$k=32$	$l=2$ $k=16$	$l=3$ $k=16$
规约时间	3.6480	6.2810	0.0490	0.1010	0.1710	0.7870	3.6930	1.0330	1.3880
Hadamard	0.0450	0.0644	0.4539	0.4486	0.4527	0.4529	0.0450	0.4576	0.4386
约减基范数	882	718	882	882	882	882	882	718	666
	789	789	789	789	789	789	789	789	705
	718	762	718	718	718	718	718	762	718
	792	776	869	792	792	792	792	776	771
	666	666	874	709	709	666	666	666	709
	709	709	792	846	846	709	709	709	891
	867	867	823	857	857	867	867	867	702
	864	864	811	811	805	864	864	864	777
	805	1054	791	791	713	805	805	1054	811
	834	777	878	780	721	834	834	777	722
11 列	747	747	721	721	747	747	747	747	721
	733	733	894	894	733	733	733	733	786
	721	721	864	888	816	721	721	721	748
	816	748	793	793	809	816	816	748	976
	793	842	724	760	831	793	793	842	715
	861	799	811	852	739	861	861	799	1180
	747	747	883	798	805	747	747	747	687
	794	752	841	796	775	794	794	752	753
	755	1074	948	838	827	755	755	766	683
	766	766	814	807	737	766	766	711	760
21 列	711	750	849	756	843	711	711	737	770
	789	789	791	795	765	737	789	775	658
	802	711	763	784	687	731	802	729	798
	731	703	822	687	788	802	731	683	740
	802	802	687	741	822	828	802	777	827
	913	802	698	698	740	782	913	793	705
	782	731	856	796	846	740	782	749	814
	828	793	797	797	793	870	828	743	695
	847	777	789	789	789	853	847	782	729
	804	698	866	866	809	771	804	842	726

表 3.5 列出 LLL 与 BKZ[62]两种约减算法的比较。由于 BKZ 算法中块大小 k 的取值可为 $2<k<64$，因此，此处选取具有不同代表性的值来分析结果，BKZ 算法比 LLL 算法要耗费大量的时间，因为其中包含不同阶段的 LLL 算法与 ENUM 算法，但其约减效果要比 LLL 的更约减。k 越大，所需时间越多，基更约减。k 值从 4 到 12 的约减变化较明显，后面的变化较小，由于基达到了一定的约减，所以基本就不会改变了，只有约减时间在递增。因此，可根据实际情况考虑应用取值，通常 k 取 $n/4$ 左右。

最后，表 3.6 中列出几种不同规约算法的比较。由表可知，l 次规约与 LLL 相比，虽然规约基更约减，但规约时间大幅度增加，仅仅考虑单一的优势不可取。从整体来看，Segment LLL 与 LLL 算法相比，规约时间大幅度减少，但约减效果略差；且由 k 的取值效果来看，似乎越大，更约减；但细看可发现，当 $k=n/4$ 时，约减效果和 LLL 算法非常相近，且运行时间大幅度减少；因此，当 k 取合适的值时，Segment LLL 完全可近似代替 LLL 算法。BKZ 算法可看成是 DeepLLL 的分块形式，因此，通过比较，并没有所提算法的运行效果优。

Segment l 次规约是我们提出的思想，从运行结果来看，约减效果非常明显，且与 l 次规约相比，运行时间大幅度降低。l 的值可动态控制，如果约减基的 Hadamard 比率很高，就可适当减小 l 的取值，如果比率很低，可适当增加其值。根据具体分析，可选择相对优的 l 值，对程序的运行速度会有一定的改善。因此，当算法被综合考虑时，不难发现，新提出算法的效率在综合情况下是最好的。

3.4 本章小结

本章首先提出算法的思想，并列出其伪代码；然后进行算法仿真，验证理论分析与实验的一致性。实验首先选取一个 10 维的随机格，分别从 l 取不同值、k 取不同值、δ 取不同值时的运行时间及约减基效果来比较它们的特性，从而可得三个参数之间的相应关系。其次，为了更好地比较算法的约减效果，选取一个 64 维的随机格来比较四种不同算法的运行结果，进而从理论分析与实践的一致性方面来详细阐述。

第 4 章 球译码算法的改进

4.1 引言

格基规约算法可以应用在球译码算法中，使得球译码算法（SDA）的效率有很大提升；而球译码算法是解决整数最小二乘问题的有效方法，它的主要关注点就是搜索超球面初始化半径的选择问题。格基规约可以约减初始化参数，获得一个包含较短向量的矩阵，应用在球译码算法中可以缩小初始化半径，提高遍历效率。

SDA 有两个主要问题，其一：怎样有效选择半径的范围是非常重要的？其次，如何判断一个点是否在一个球内？按球译码算法的搜索方式来分，它可被划分成深度优先和宽度优先。深度优先可以提供优化性能[63,64,65]，但是复杂度的转化范围是非常大的。宽度优先算法的性能是稍微逊色些，但其计算的复杂度是固定的，它被广泛应用到 MIMO 通信系统的最大似然检测中，如果 K 值很大，将导致大量的计算复杂度，其中包括路径扩展、排序以及更新。所以为了降低总的计算复杂度，出现很多改进的策略，例如动态位排序[66]，基于路径规划[67]，稀疏感知（对解进行范数限制）[68]，递归的格译码[69]等。因此，各有优点和弊端，都受到了广泛地关注[70]。本章针对 K-Best 算法，K 取较小的值且每层采用预测技术来确定半径 x 的分向量 x_i 。

球译码算法的搜索过程就好像遍历一棵树，因此，我们通常会采用深度优先或者宽度优先的策略。它们可分别参考球译码算法（SDA）[71,72]和 K-Best 算法（KBA）[73,74]。SDA 就好像一棵树的先序遍历，从根节点到叶子节点，找到一条满足初始条件的有效路径。而 KBA 就好像一棵树的层次遍历，从第一层一直遍历到最后一层，再找出一条相对较短且满足条件的路径。

4.2 提出的优化算法

4.2.1 Hadamard 比率

Hadamard 比率的定义在第 2 部分已介绍。之所以在此提出，是因为它给数据的正确性带来一定的保证。例如：用 Babai 估计求解最近向量问题（CVP），若一组基的 Hadamard 比率越大，向量之间的误差就会越小。因此，我们应该选择 Hadamard 比率尽可能高的一组基，找解的正确性才会越高。接下来，我们将使用 Hadamard 比率去生成满足特定条件的一组高质量的基[75]。

算法 4.1 Hadamard 比率判定

输入：一个格基所构成的方阵 m
输出：判定格基 m 的 Hadamard 比率

1. $n = size(m)$ 表示矩阵 m 的行数和列数
2. $n = n(1), product = 1$ 取矩阵的行数
3. for $i = i:n$ do
4. $product = product * \|m(i,:)\|$
5. end for
6. $result = (abs(\det(m)) / product)^{1/n}$

算法 4.2 生成高质量基函数

输入：基向量坐标中绝对值最大的值：N，向量个数：v，Hadamard 比率的下限：h
输出：满足 Hadamard 比率设置的一组高质量基

1. result=unidrnd$(2*N,v)-N$ 生成维数为$v*v$的随机矩阵，且向量元素的范围在 1 到N中
2. while $H(result) < h$ do
3. result=unidrnd$(2*N,v)-N$
4. end while

算法 4.2 是一个生成高质量基的函数。在标记 2 处调用算法 4.1，通过连续的随机生成和判断来获得满足条件的一组基。h 的选择是非常重要的：其值越接近 1，生成的基找解的结果就越精确，但是执行程序的时间由于随机过程的特点也就越长。

4.2.2 QR 分解（R 对角线转正）

第 2 部分整数最小二乘问题的不等式通过 QR 分解可被简化成：

$$\left\| Rx - \hat{y} \right\|_2^2 \le d^2, \quad d^2 = d^2 - \left\| Q_2^{-1} y \right\|_2^2 \tag{4-1}$$

因为 R 是上三角，所以也可以把上式转换成如下形式：

$$\left(r_{m,m} x_m - \hat{y}_m \right)^2 + \left(r_{m-1,m-1} x_{m-1} + r_{m-1,m} x_m - \hat{y}_{m-1} \right)^2 + \cdots \le d^2 \tag{4-2}$$

上式左边第一项相对于格点 x 仅仅有一个未知数 x_m，第二项有两个未知数，以此类推。因此我们可以根据：

$$(r_{m,m} x_m - \hat{y}_m)^2 \le d^2 \tag{4-3}$$

（4-3）式求得 x_m 的取值范围。为了更精确地表达 x 子项的取值范围，此处考虑把上三角矩阵 R 的对角线上的数据全部转变成正数。现列出对角线转正的算法如下：

算法 4.3 对角线转正（DPR）算法

输入：矩阵 H 的 QR 分解：（H，Q，R）
输出：把矩阵 R 的对角线元素转正，获得新的矩阵：（Q，R）

```
1. Nr 矩阵 H 的行数
2. for n=1:Nr do
3.   if (R(n,n)<0) then
4.     R(n,:)=-R(n,:)
5.     Q(:,n)=-Q(:,n)
6.   end if
7. end for
8. 输出转正后的矩阵
```

通过转变上三角矩阵 R 对角线的元素值，可以很容易地获得 x_m 的取值范围：

$$\left\lceil \frac{-d+\hat{y}_m}{r_{m,m}} \right\rceil \le x_m \le \left\lfloor \frac{d+\hat{y}_m}{r_{m,m}} \right\rfloor \tag{4-4}$$

此外，对于每一个满足上式的整数 x_m，我们可以定义：$d_{m-1}^2 \triangleq d^2-(r_{m,m}x_m-\hat{y}_m)^2$，$\hat{y}'_{m-1} \triangleq \hat{y}_{m-1}-r_{m-1,m}x_m$。如果：

$$d_{m-1}^2 \ge \left(r_{m-1,m-1}x_{m-1}-\hat{y}'_{m-1} \right)^2 \tag{4-5}$$

可得 x_{m-1} 的取值范围：

$$\left\lceil \frac{-d_{m-1}+\hat{y}'_{m-1}}{r_{m-1,m-1}} \right\rceil \le x_{m-1} \le \left\lfloor \frac{d_{m-1}+\hat{y}'_{m-1}}{r_{m-1,m-1}} \right\rfloor \tag{4-6}$$

根据以上的求解过程，以此类推，可依次求得 $x_{m-2},x_{m-3},\cdots,x_1$ 的范围，进而可以获得以半径为 d 的超球体中的所有格点。

4.2.3 LLL 规约及 K-Best(BFS+PEDS)思想

本小节，LLL 算法用来约减上三角矩阵 R，把 R 分解成 $\hat{Q}\hat{R}T^{-1}$，$\hat{Q}$ 是正交矩阵，$\hat{R}$ 是上三角矩阵，T 是单模矩阵。因此，$\hat{R}$ 的列向量形成一组约减基。第一列向量的范数足够小，几乎呈递增的趋势。所以，最后一列最后一行的 $r_{n,n}$ 通常会变大。根据公式（4-3），$r_{n,n}$ 越大，x 的子项 x_n 的范围越小。因此，可获得更小的初始化半径，进而使得球体内的格点数也越少。利用 LLL 算法去初始化一个矩阵的过程可以参考文章[76]。而如果应用第三章优化的规约算法，可以获得更好的约减效果。

LLL 约减后，可以获得一个新的上三角矩阵 $\hat{R}$，把它带入原表达式中，可得一个新的 $\hat{y}=\hat{Q}^{-1}y$。解等式 $\hat{R}x=\hat{y}$，可得 x 的解并且四舍五入 x。然后，可求得一个更小

的半径：$d=\left\|\hat{R}round(x)-\hat{y}\right\|$。初始化半径后，通过 SDA 找解，如果求得一个解，左乘一个T，就可以获得和仅仅用 QR 分解后求的解一样。最重要的是，由于应用 LLL 约减后，使得半径缩小了，找解的速度相对提高了。

在 LLL 算法的基础上，继续考虑球译码算法的宽度优先策略作为初始化。宽度优先算法的缺点是：如果k比较大，球的半径不能及时被更新直到$i=1$层，且对每一层的节点都需要k个扩展，之后通过路径的排序以及更新，其运行时间是相当大的。因此，为了平衡效率，k通常取相对较小的值。我们考虑k取较小值时的两种情况：其一，先选取最高层的最左侧节点，沿着当前节点遍历最左侧孩子节点，用同样的方式，获得每一层的节点直到$i=1$，可得到一条路径；其次，使用预测技术来决定每一层节点的选取，而不用计算所有节点的 PEDS 比较后再确定当前层的节点，我们可以根据下列公式获得每一层最优节点的扩展：

$$x_k=\frac{y_k-\sum_{j=k+1}^{m}r_{k,j}x_j}{r_{k,k}} \tag{4-7}$$

在求得x_k的值后，从求得每个节点的范围内，选择一个与x_k最接近的整数。这种预测方法的时间复杂度明显减少，且找到一个解的成功率也较高。

4.2.4 优化算法

球译码的宽度优先搜索算法（K-Best），主要通过预测技术来找相对较短的半径。预测技术使得：

$$r_{k,k}x_k+\sum_{j=k+1}^{m}r_{k,j}x_j-y_k=d_k \tag{4-8}$$

即令半径d的子项d_k等于 0 时，求出x_k的值。算法 4.4 中，标记 9 循环控制路径x的每一个子项，从m开始直到递减为 1，可找到一条路径。标记 10 应用了所提出的预测技术，可避免求解所有节点的 PEDS，进而节省大量的时间。标记 11 是对预测所求的x_k取整，12-17 是判断x_k的值是否在所求的范围内，如果不在，取与范围最接近的边缘值。18-26 首先判断子项k是否大于 1，如果大于，递减之后继续判断子项的取值情况，若$LB_k>UB_k$，说明子项范围内没解-即可跳出，否则，按照上述要求取相应的值。如果子项k等于 1，说明找到一条路径，标记 27 之后是判断所求得路径的长度是否小于初始化半径，如果小于，则找到更小的半径且更新当前半径，否则当前初始化半径不变。

算法 4.4 K-Best 算法

输入：上三角矩阵：r（上文中的 R），球心：y（上文中的 $\hat{y}$），球半径：radius
输出：寻找一个解作为半径：x 或为初始半径

1. m,n 矩阵 r 的行和列
2. $d_m = radius$
3. $LB_m = \lceil (-d_m + y_m)/r_{m,m} \rceil, UB_m = \lfloor (d_m + y_m)/r_{m,m} \rfloor$
4. $\min R = d_m, \text{int}erSum_m = y_m$
5. if $LB_m > UB_m$ then
6. radius= $\min R$ 初始化半径太小
7. end if
8. k=m
9. while $0 < k \le m$ do
10. $x_k = (y_k - r_{k,k+1:m} * x_{k+1:m})/r_{k,k}$
11. $x_k = round(x_k)$
12. if $x_k > UB_k$ then
13. $x_k = UB_k$
14. end if
15. if $x_k < LB_k$ then
16. $x_k = LB_k$
17. end if
18. if $k > 1$ then
19. $k = k - 1$
20. $d_k = \sqrt{d_{k+1}^2 - (\text{int}erSum_{k+1} - r_{k+1,k+1} * x(k+1))^2}$
21. $\text{int}erSum_k = y_k - r_{k,k+1:m} * x(k+1:m)$
22. $LB_k = \lceil (-d_k + \text{int}erSum_k)/r_{k,k} \rceil, UB_k = \lfloor (d_k + \text{int}erSum_k)/r_{k,k} \rfloor$
23. if $LB_k > UB_k$ then
24. 若间距为空，则跳出 K-Best 算法
25. radius= $\min R$, break
26. end if
27. else
28. k=1
29. if $\min R > \|r * x - y\|$ then
30. 找到一个解
31. $\min R = \|r * x - y\|$
32. radius= $\min R$, return
33. end if
34. break
35. end if
36. end while

4.2.5 复杂度分析（流程图）

这个部分，我们讨论所提出的 K-Best 算法作为初始化半径的时间复杂度，在传统的 K-Best 算法中，从所求得的 $number = UB_k - LB_k + 1$ 个候选节点中选取 k 个最小的

值，至少需要 $f(k,number)=number-k+\sum_{number+1-k<j}\leq number\lceil\log_2 j\rceil$ 次成对的比较，且需要执行排序方法。当采用预测技术且 k 取值为很小的正整数时，我们仅仅需要检查 x_k 是否在 $[LB_k,UB_k]$ 的范围内。如果存在，可以直接应用；如果不存在，直接分配与 x_k 最接近的在 $[LB_k,UB_k]$ 范围内的整数，这个过程的时间复杂度是一个常量。

在 k 取很小的正整数时，程序可以更快的找到解。而球以更小的半径去初始化球译码算法可使得访问节点的数量和运行时间的复杂度大幅度减小。为了更清晰地展现程序的执行过程，特列出整个算法的流程图，使得执行过程一目了然，如下图 4.1：

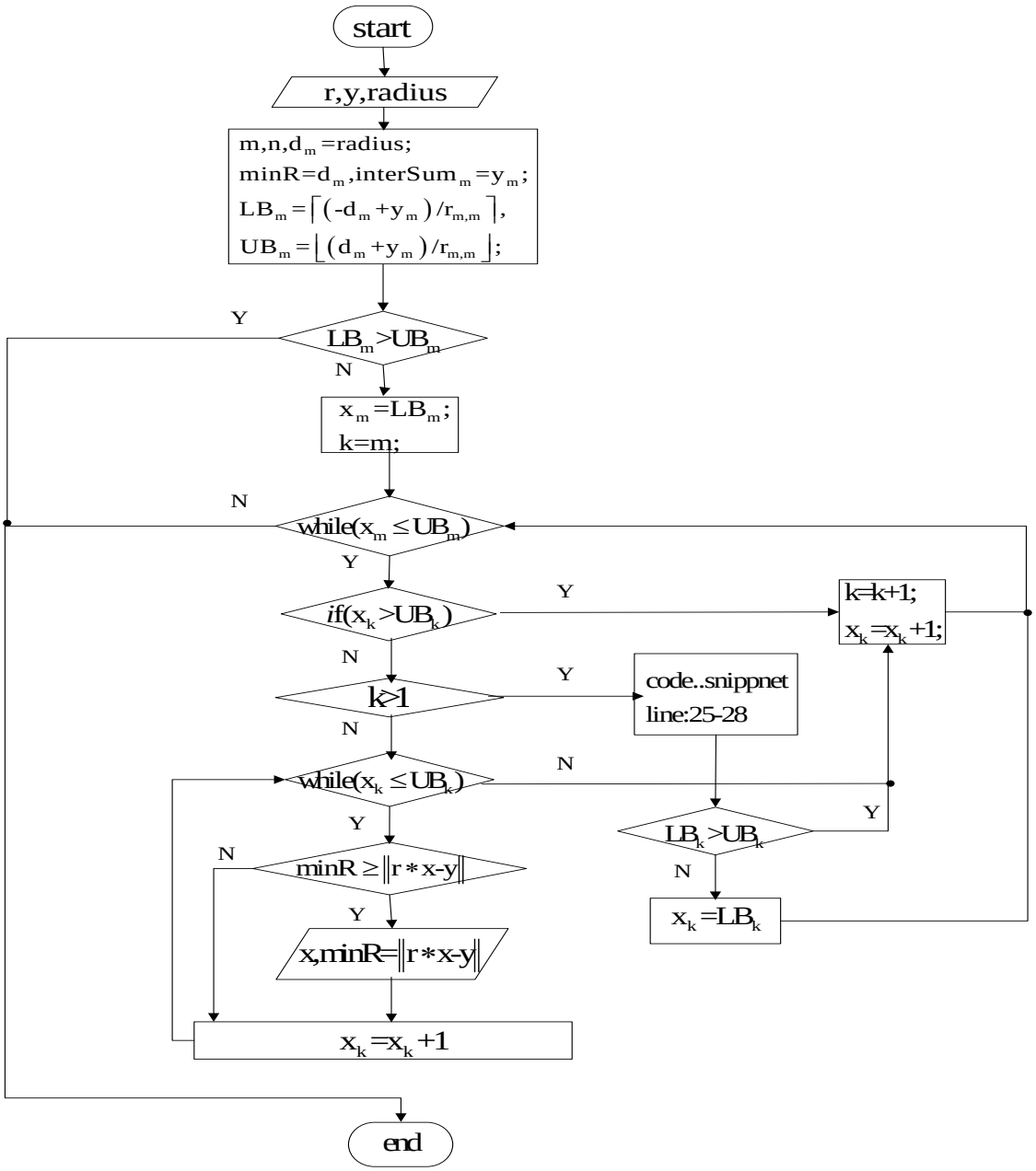

图 4.1 K-Best 算法流程图

4.3 算法仿真

4.3.1 数据的选取

本小节，通过获取不同初始化半径的方法来比较程序的执行结果。QR 分解，再通过 Babai 估计求得半径，会存在误差。QR+ERROR，加上存在的误差，可明显比较两者的效果。QR+LLL，通过 LLL 算法来约减，可明显比较出规约算法的重要性。后面三种，主要区别就是加上了宽度优先搜索策略。

实验主要按照以下步骤来实现，首先，实验选择满足 Hadamard 比率至少为 0.7 的高质量基，只有用这种方法才尽可能使得误差降低些，比较时更有说明性。其次，矩阵 A 通过 QR 分解之后，再采用 LLL 算法进一步约减，最后使用 k 取较小正整数的带预测技术的 K-Best 算法来求解更小的初始化半径。

通过大量实验数据去测试不同情况下的访问节点数、执行时间以及初始化半径。用 1 到 32 维的随机矩阵依次读入算法中（超过一万的数据量），比较了不同情况下不同维数的性能。通过实验，可验证所提思想理论分析的正确性与一致性。

4.3.2 不同维度的访问节点数

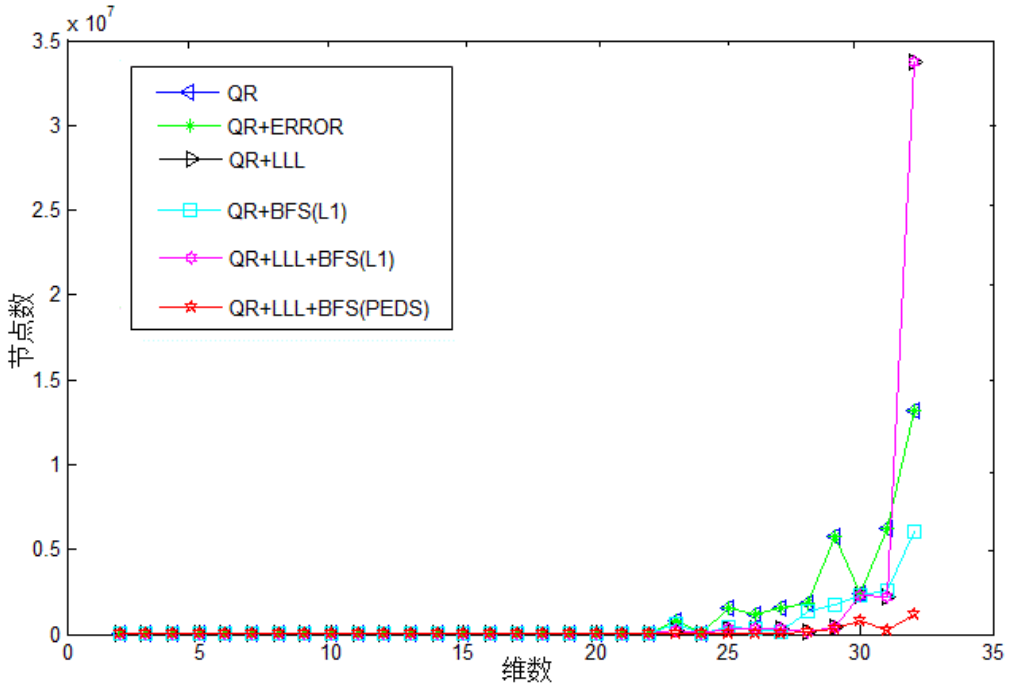

图 4.2 不同维数下，6 种初始化半径的访问节点数

后三种初始化半径方法加上了宽度优先搜索策略进一步优化，其主要区别是：QR+BFS(L1)，宽度优先搜索中 k 取较小的值，且每一层按照深度优先选取的策略定值，即每层选取最左侧分支节点形成一条路径；QR+LLL+BFS(L1)在前一种方法的基础上加了规约算法，其效果差的主要原因是 BFS(L1)遍历方式的选取；最后一种方法 QR+LLL+BFS(PEDS)是集所有优化特点的综合策略，在利用规约的基础上，加上带预测技术的宽度优先搜索的遍历方法。

图 4.2 展示了在不同维数下，访问节点的数量。通常，维数越大，访问节点的数量就越多。当然，如果所选矩阵的种类（特点）不同，也会有不同的现象。总的来说，图像整体呈现一个上升的趋势，从图中可清晰地看出，大红色五角星折线的访问节点数是最少的，即效果最好。由于数据量偏大，25 维之前的数据，几乎成一个重叠的状态，其之间还是有差距的。下面，通过一个列表来详细展示所执行的部分数据。

表 4.1：不同维数下，访问节点的数量

Curve\dimensions	4	8	12	16	20	24	28	32
QR	11	89	647	3298	29310	50325	1807199	13151570
QR+ERROR	11	89	647	3301	29322	50818	1807822	13164777
QR+LLL	6	8	14	161	841	89790	148578	33727341
QR+BFS(L1)	11	89	89	3298	25351	50325	1379257	6034934
QR+LLL+BFS(L1)	6	8	14	161	841	89790	148578	33727341
QR+LLL+BFS(PEDS)	6	8	14	161	841	4198	106630	1140275

表 4.1 列举了不同维数下，六种初始化半径的访问节点情况。第一种情况，仅仅应用了 QR 分解的思想，由于 Babai 估计，此种情况存在误差，所以计算的节点数量可参考。第二种情况加上估计误差，访问节点数量通常是大于等于第一种的。第三种由于加入 LLL 算法的思想，缩小了初始化半径，因此访问节点的数量明显比第一，二种情况减小了。

第四种情况是：带有宽度优先搜索但没有 LLL 算法的初始化，很明显，结果不是很乐观，因此，可得出 LLL 算法的应用效果是不容忽视的。第五种和第六种的区别就是宽度优先搜索的不同，即在每一层中找点的方式不同，前者是选择最左侧的几个节点，然后求出它的 PEDS 值，从中再选出一个 PEDS 最小的节点，继续用同种方法向下遍历直到 $k=1$，即找到一条路径；后者是每一层都采用预测技术，求出使得当前层拥有最小 PEDS 的 x 子项值，这样可以节省每一层求众多 PEDS 值，再找出一个最小的 PEDS 的大量时间。综上所述，可知最后一种执行效果是最优的。

4.3.3 不同维度的执行时间

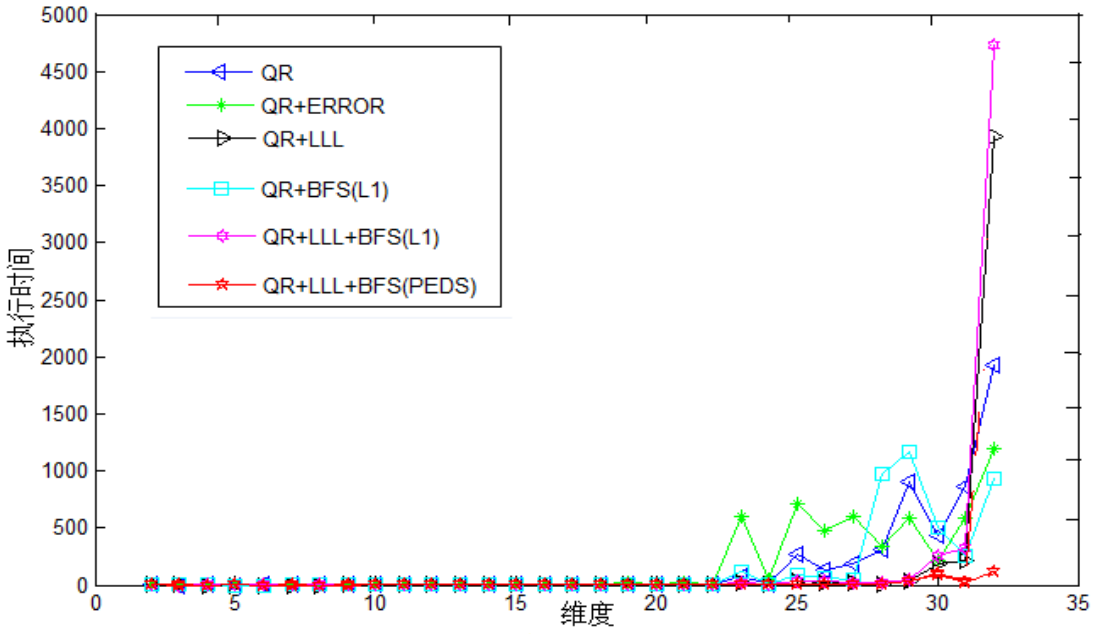

图 4.3 不同维度下的执行时间

图 4.3 展示了不同维度下，六种初始化半径算法下的程序执行时间。根据访问节点数量的图可知，访问的节点越多，执行的时间越长。六条曲线清晰地列出了执行时间的趋势。可清晰的看出，大红色曲线是运行时间最短的，即效果最好的！由于数据是比较接近的，所以维度 23 之前看起来有着重叠的状态。当然，还是有细微区别的，同样通过表格来详细列举执行时间的数据。

表 4.2：不同维度下的执行时间

Curve\dimensions	4	8	12	16	20	24	28	32
QR	0.0010	0.0250	0.0790	0.4210	4.2880	4.9010	306.8830	1.9314e+003
QR+ERROR	0.0010	0.0560	0.0710	0.3920	6.2080	43.3980	344.8400	1.2006e+003
QR+LLL	0	0	0.0030	0.2830	0.4600	6.7330	11.0480	3.9267e+003
QR+BFS(L1)	0.0010	0.0070	0.0050	2.4120	4.7540	10.4730	976.8480	933.5260
QR+LLL+BFS(L1)	0.0010	0.0010	0.0010	0.0530	0.0400	8.3480	15.1740	4.7372e+003
QR+LLL+BFS(PEDS)	0	0	0.0010	0.0060	0.0440	0.3910	9.8640	121.4000

表 4.2 列举了详细的执行时间，每种情况有轻微的不同。因为程序的初始化半径不同，所以每种运行时间也是不同的。从所列数据中可知，最后一种效果是最好的，且维数越大，效果越明显。

4.3.4 不同维度的初始化半径

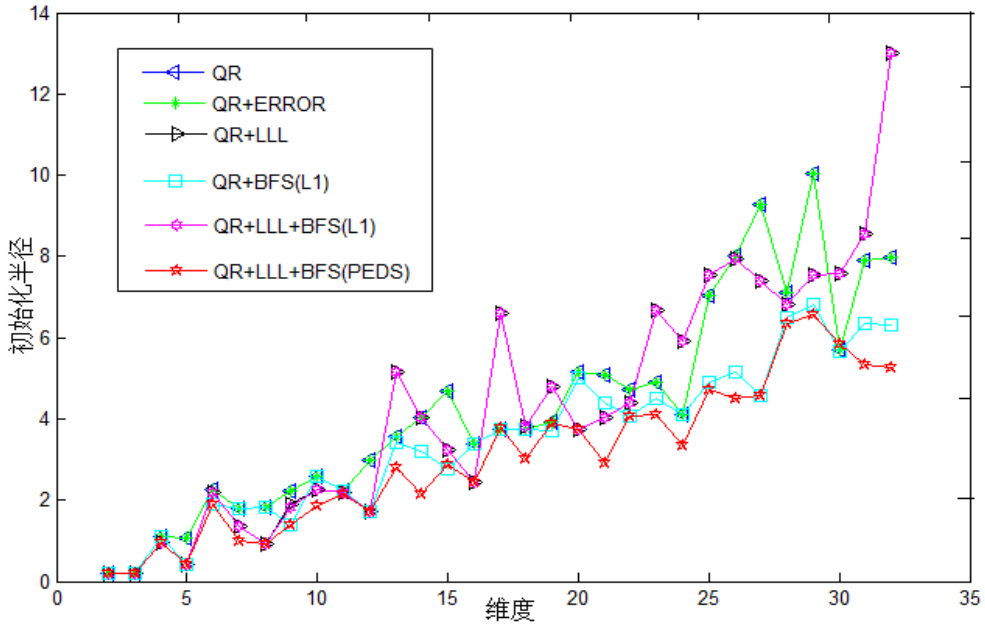

图 4.4 不同维度的初始化半径

图 4.4 列出了不同维度的初始化半径，由于半径的数值相对较小，每一个阶段的改变都是非常清晰的。显而易见，大红色曲线是效果最好的。

表 4.3：不同维度的初始化半径数值

Curve\dimensions	4	8	12	16	20	24	28	32
QR	1.0883	1.8232	2.9946	3.3671	5.1670	4.1056	7.0974	7.9660
QR+ERROR	1.0883	1.8234	2.9947	3.3677	5.1677	4.1146	7.0981	7.9687
QR+LLL	0.9738	0.9079	1.7257	2.4439	3.7528	5.9133	6.8091	13.0146
QR+BFS(L1)	1.0883	1.8232	1.7257	3.3671	4.9971	4.1056	6.5092	6.3111
QR+LLL+BFS(L1)	0.9738	0.9079	1.7257	2.4439	3.7528	5.9133	6.8091	13.0146
QR+LLL+BFS(PEDS)	0.9738	0.9079	1.7257	2.4439	3.7528	3.3579	6.3320	5.2763

表 4.3 记录了初始化半径曲线值的变化。初始化半径决定了运行程序的效果，包括访问的节点数，运行时间和复杂度。明显地，最后一行的效果是非常可观的。

4.4 本章小结

本章中，我们考虑应用球译码算法来解决（ILSP）。在球译码性能中，我们提出了一个关键的问题就是减少初始半径。同时，我们引用了 Hadamard 比率来获取高质量的基。我们还使用了新的思想展现了较好的执行时间和约减半径。实验显示了在保持同样高质量解的情况下，所提出的思想在平均情况下，初始化半径可减少大约 61.8%。

第 5 章 格理论在 0-1 整数背包中的应用及背包算法的并行化

5.1 引言

由于格中的计算都是简单的线性代数运算，因此非常容易在软硬件上实现。且格理论在密码学领域有着广泛地应用，因此通常应用格中的难题——高维格中求 SVP、CVP 等来设计新型的密码方案。由此可知，此类密码方案具有稳定性、安全性更高，计算速度更快等优点。除此之外，格基规约也可用在基于格的密码系统的分析上，如著名的格基规约 LLL 算法，BKZ 算法等。

基于格构建的密码系统主要有几下几种：GGH、同余、背包、NTRU 等。密码系统的分析可直接应用格基规约算法来求解。例如：本章将介绍格基规约算法在背包密码系统中的应用过程，即恢复明文的详细步骤。在此基础上，深入研究常规 0-1 整数困难背包的求解原理以及优化的思路[56,57]。2012 年，Nagashima 和 Kunihiro 提出针对适当消息长度更快地解决困难背包的算法[77]，此算法是在 BCJ[57]的基础上进行优化改进的。通过比较与了解，在熟悉 HGJ[56]生成算法的基础上，实现了它的并行化，并给出详细的并行思路。最后，通过实验进行比较串行和并行的运行时间及并行算法的找解情况。

5.2 LLL 优化算法在 0-1 整数背包中的应用

普通的背包问题是一个 NP 完全问题，因此想必是非常困难的。针对背包体制的几种攻击算法，常规求解方法是穷举 2^n 种可能（暴力攻击）。更好一点的方法是排序下面的两个集合，并寻找一个冲突：$\{\sum_{j\le n/2} x_j a_j : x_j = 0 or 1\}, \{t - \sum_{j>n/2} x_j a_j : x_j = 0 or 1\}$，这种情况需要 $O(n2^{n/2})$ 个步骤的操作。低密度子集合攻击，当密度小于 0.9408 时，可通过格基规约算法来求解格基中的最短向量，从而找到解。Shamir 密钥恢复攻击以及联立丢番图逼近攻击等。

基于困难背包问题建立一个密码系统，此类系统一般都会存在陷门问题，如 Shamir 利用整数规划的方法对 MH 背包密码体制在多项式时间内成功地恢复了密钥。而使用规约算法进行分析背包是不用考虑任何额外信息的，对于任意的背包序列都可以进行分析，且分析的效率只与背包的长度及大小有关系。

假设给定背包序列 $A=(a_1,a_2,\cdots,a_n)$ 和一个整数和 S，按照下列方式构造矩阵，且把矩阵的每一行看成一个向量，共 $n+1$ 个向量，构成格 L 的一个基：

$$B=\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_n \\ v_{n+1} \end{bmatrix}=\begin{bmatrix} 2 & 0 & 0 & \cdots & 0 & a_1 \\ 0 & 2 & 0 & \cdots & 0 & a_2 \\ 0 & 0 & 2 & \cdots & 0 & a_3 \\ \vdots & \vdots & \vdots & 2 & 0 & \vdots \\ 0 & 0 & 0 & \cdots & 2 & a_n \\ 1 & 1 & 1 & \cdots & 1 & S \end{bmatrix} \tag{5-1}$$

可设背包难题的解为 x=$(x_1,x_2,\cdots,x_n)$，其中的每个子项取值只能为 0 或者 1。格 L 中一定包含一个向量：

$$\begin{aligned} t &= (x_1,x_2,\cdots,x_n,-1)\cdot B \\ &= x_1v_1+x_2v_2+\cdots+x_nv_n-v_{n+1} \\ &= (2x_1-1,2x_2-1,\cdots,2x_n-1,0) \end{aligned} \tag{5-2}$$

向量 t 是一个非常短的向量，其长度为 $\sqrt{n}$，格中几乎不可能再找出比 t 还短的非零向量。在求解的过程中，为了更容易地找到最短向量，通常会把矩阵 B 的最后一列乘以一个很大的常数 $Cons\tan t$，矩阵行列式的值将是原来的 $Cons\tan t$ 倍，但求最短向量的 Gauss 期望将增大 $Cons\tan t^{1/(n+1)}$ 倍，但所求的最短向量 t 将不会改变。如果找到此向量，用规约算法分析背包密码系统，就很容易还原明文。

给定一个背包序列 $A=(a_1,a_2,\cdots,a_n)$ 和一个密文 S；求解明文步骤如下：

1．根据所给的值，构造一个矩阵；

2．求解矩阵的 Hadamard 比率，如果值越接近 0，说明这个基越差，是劣质基，反之，越接近 1 的基是优质基；

3．对这个矩阵 B 进行 LLL 优化规约，可求得一个相对约减的基；

4．对这组基求解 Hadamard 比率，如果比之前的大，说明基越来越好；

5．从约减基矩阵中找出第一行的向量作为 t，即格中相对最短的非零向量；

6．用等式（5-2）来求解明文 x；把向量 t 乘以-1，即可恢复明文：$x_1,x_2,\cdots,x_n$。

第 3 章提出的分块 l 次约减算法，通过实验验证可知，其约减效果比 LLL 原本算法好。如果把它用到此处，将会更快地恢复加密的明文内容，特别是维数较高的情况下，难度也会越大，而应用优化的规约算法可以加速提升效率。

格基规约算法，由于其高效灵活的实用特点，在密码系统中占据非常重要的地位，且采用格密钥体制。因此，在格中寻找一个近似最短的向量来代替私钥，以实现

对加密内容的解密，针对密钥体制的攻击，Nguyen 和 Regev[78]提出利用规约算法对 GGH、NTRU 签名的密码分析，且非常有效。

5.3 0-1 整数背包问题求解算法的并行化

5.3.1 算法的设计

在分析背包问题的同时，主要从格领域进行着手，但也关注了它的常规求解方法，即 HGJ 的新生成困难背包算法。已知该算法的时间空间复杂度，为了进一步减少时间复杂度，我们改变了算法处理数据的执行方式。在本小节中，描述了 HGJ 算法的并行实现方式，并给出具体实现的方案。

在串行算法中，基于 n 个元素的生成背包问题，从 n 个元素中选择至少一半的元素组成和 S：

$$S=\sum_{i=1}^{n}x_i a_i \tag{5-3}$$

即元素个数：

$$\ell=\sum_{i=1}^{n}x_i \geq \lceil n/2 \rceil \tag{5-4}$$

（假定选 $n/2$ 个元素）；采用四分法，即从 n 个元素中任意选择 $n/8$ 个元素组成和，共有 $C_n^{n/8}$ 种组合数，存在一个列表 $List$ 中。任意选择三个值，即 R_1,R_2,R_3，用 $R_1,R_2,R_3,S-R_1-R_2-R_3$ 分别取余 M， 其值可为 $C_\ell^{\ell/4}$，把 $List$ 中的数据分别取余 M，$\delta_1 \equiv R_1(\mathrm{mod}\,M)$，$\delta_2 \equiv R_2(\mathrm{mod}\,M)$，$\delta_3 \equiv R_3(\mathrm{mod}\,M)$，$\delta_4 \equiv (S-R_1-R_2-R_3)(\mathrm{mod}\,M)$。把获得相同余数的放在同一个列表中，可划分成四个列表：$\delta_1,\delta_2,\delta_3,\delta_4$，这样可缩小一定的找解范围。然后从不同列表中选择一个数进行相加，如满足 $S=\delta_1+\delta_2+\delta_3+\delta_4$ 的条件，即找到一组解。因为解有可能不唯一，因此找到一个解即可退出程序。

在并行算法中，多个 CPU 同时执行不同范围内的代码。因此，可能多个 CPU 中都会有解，但同一个解只能在一个 CPU 上出现。与串行方法类似，如果任意一个 CPU 找到解，即可退出全部程序。并行就雷同多个串行同时执行，只不过每个 CPU 分配的范围更小，找解会更快一些。总的来说，并行效率提高的非常明显。

并行实现时，首先确定 CPU 的个数 N，根据 2^i=N 来确定取前 i 个比特位（前 i 个元素 $a_1,\cdots,a_n$）表示 N 种不同的情况，然后根据参数确定不同 CPU 执行的范围。为了更清晰地说明并行处理器的执行情况，下图列出了 8 个处理器并行执行程序时的大致

情况。如选择2^3=8个处理器，需要用n个元素中的前三位来表示这 8 种情况。这 8 个处理器同时执行算法。可能有的有解，有的无解；可能所有情况只有一个解，也可能有多解；可能一开始很快就找到解，也可能最后才找到解。每个问题不同，每种情况的运行时间也是不同的。

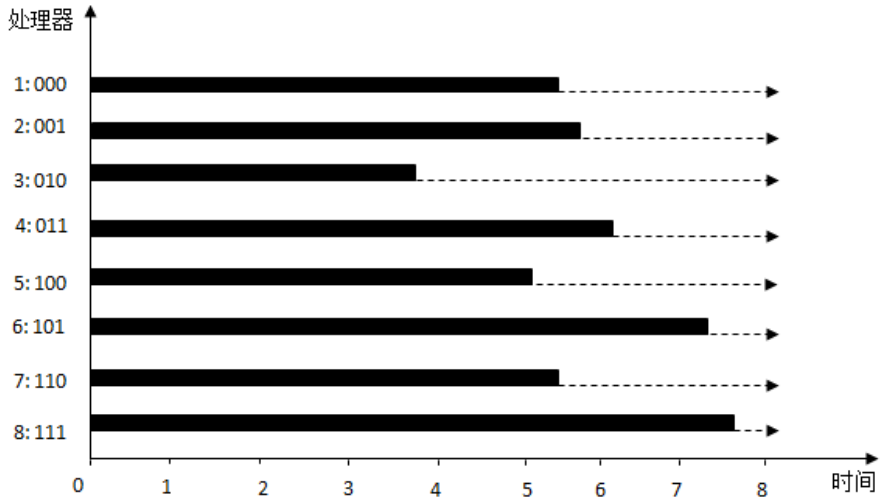

图 5.1：多处理器并行执行时间

5.3.2 算法的实现

算法 5.1 HGJ 算法的串行

输入：n 个背包元素 $(a_1, a_2, \cdots, a_n)$，背包和 S。设 M 是一个接近 $2^{\beta n}$ 的随机素数，参数 $\beta \in (1/4, 1/3)$，R_1, R_2, R_3 是随机数。

输出：$(x_1, x_2, \cdots, x_n)$ 序列。

1. 选择 $n/2$ 个元素组合成和 S
2. 利用四分法，$C_n^{n/8}$ 组成一个列表 List
3. 利用 $R_1, R_2, R_3, S - R_1 - R_2 - R_3$ 分别取余 M
4. List 列表中的值也分别取余 M，把对应相等的值放在同一个列表中，$\delta_1, \delta_2, \delta_3, \delta_4$
5. $\delta_1 \equiv R_1 (\mathrm{mod}\, M),\ \delta_2 \equiv R_2 (\mathrm{mod}\, M),\ \delta_3 \equiv R_3 (\mathrm{mod}\, M),\ \delta_4 \equiv (S - R_1 - R_2 - R_3) (\mathrm{mod}\, M)$
6. 分别从四个不同列表中 $\delta_1, \delta_2, \delta_3, \delta_4$ 选取一个值进行相加，和为 S，即找到一个解
7. 还原解对应的序列值：$(x_1, x_2, \cdots, x_n)$

为了更清晰地比较算法的改进思路，首先列出串行实现的方式，其中对 List 列表取余的主要原因是：由于解的组合情况有多种，如 96 个背包元素中选取 48 个组成和 S，进行四分法保证每个部分由 12 个元素组成，即 C_{96}^{12} 种组合情况，若从中找解，则需要枚举 $C_{96}^{12}(C_{96}^{12}-1)(C_{96}^{12}-2)(C_{96}^{12}-3)/4$ 种完全判断，因此它所花费的时间代价很高。而如果对 List 列表中的 C_{96}^{12} 种情况进行取余，可获得四个新的列表：$\delta_1, \delta_2, \delta_3, \delta_4$，所

以，花费的时间大幅度减少，只需要这四个列表中的个数相乘。我们是在选中背包元素个数平衡的情况下考虑的，如果不是平衡的，效果则更差。

算法 5.2 HGJ 算法的并行

输入：n 个背包元素 $(a_1,a_2,\cdots,a_n)$，背包和 S。设 M 是一个接近 $2^{\beta n}$ 的随机素数，$\beta\in(1/4,1/3)$，R_1,R_2,R_3 是随机数。

输出：$(x_1,x_2,\cdots,x_n)$ 序列。

1. 选择 $n/2$ 个元素组合成和 S
2. 先确定 CPU 的个数 N，在通过 $2^i=N$ 确定需要用元素的前几位来表示 CPU 的不同情况
3. 利用四分法，$C_{n-i}^{n/8},C_{n-i}^{n/8-1},C_{n-i}^{n/8-2},C_{n-i}^{n/8-3},\cdots,C_{n-i}^{n/8-i}$ 对应列表 List1，List2，List3，List4，…，List(N)
4. 根据 CPU 个数，可分为如下情况 $0,\cdots,0,a_{i+1},\cdots,a_n$，$0,\cdots,1,a_{i+1},\cdots,a_n$，…，$1,\cdots,1,a_{i+1},\cdots,a_n$，相应位置为 0，表示和 S 中不包含此元素；反之，则包含
5. 用参数分配确定不同 CPU 执行不同情况
6. 例如第一种：$0,\cdots,0,a_{i+1},\cdots,a_n$，首先确定前 i 个元素所对应的值，都为 0 表示前 i 个元素不在 S 中，则需要确定除了前 i 个元素之外的其它 $n/2$ 个元素组成的 S
7. Integer=（n-前 i 个元素中为 1 的个数）/4，Module=（n-前 i 个元素中为 1 的个数）%4
8. 从 Integer 和 Module 的取值中判断还需要确定组成和 S 的元素个数
9. 再确定是从哪个 *List* 列表中选值，如第一种，Integer=4，Module=0，4 个全部从列表 List1 中选取
10. $\delta_1\equiv R_1(\mathrm{mod}\,M),\delta_2\equiv R_2(\mathrm{mod}\,M),\delta_3\equiv R_3(\mathrm{mod}\,M),\delta_4\equiv(S-R_1-R_2-R_3)(\mathrm{mod}\,M)$
11. 以下同串行找解思路
12. 例如第 N 种：$1,\cdots,1,a_{i+1},\cdots,a_n$，前 i 个元素都在 S 中，需确定除前 i 个元素之外的其它 $n/2-i$ 个
13. Integer=（$n/2-i$）/4，Module=（$n/2-i$）%4
14. 假如 Integer=3，Module=1，需要确定 $\delta_1,\delta_2,\delta_3,\delta_4$，前三个从 List1 中取余，最后一个从 Listi 中
15. $\delta_1\equiv R_1(\mathrm{mod}\,M),\delta_2\equiv R_2(\mathrm{mod}\,M),\delta_3\equiv R_3(\mathrm{mod}\,M),\delta_4\equiv(S'-R_1-R_2-R_3)(\mathrm{mod}\,M)$
16. 上式的 $S'=S-(a_1+\cdots+a_i)$，只要找到一个解，就可输出，且终止程序

并行运行的情况，需注意以下两点：根据 CPU 的个数把范围划分成不同的情况后，每种情况再根据前 i 个元素进行不同的分析。如 96 个元素中选取 48 个组成和 S，进行四分法保证每个部分包含 12 个元素，假如 CPU 的个数为 16，即选择前 4 个元素来表示。首先，确定 List1，…，List12，对应 C_{92}^{12}，C_{92}^{11}，…，C_{92}^{1}。单 List1 中的组合数就比串行算法中少 96*95*94*93 种判断情况，且 List2 到 List12 中都比串行中 List 列表中的组合数少，因此在分析从哪个列表中确定 $\delta_1,\delta_2,\delta_3,\delta_4$，相对于串行可节省大量的运行时间。其次，每种情况的和 S 也不同，具体要根据不同 CPU 的前 i 个元素的选取情况，减去比特位为 1 的对应背包的元素值，形成一个新的背包和 S。

5.3.3 算法的复杂度分析

假定λ=$2^{-\varepsilon n}$，ε是非常小的，δ表示可分解的集合，对应的和取模$M^{'}>|B|$。我们发现对于n个元组的一个指数小部分λ，成对的数量每次迭代大致都是$O(2^{\varepsilon n}M^{'})$。乘以迭代次数后的总时间是$O(2^{\varepsilon n}M^{'2}/|B|)$=$0(2^{(0.0872+\beta)n})$。测试了四倍的数量一致是$O(2^{0.3113n})$，通过组合，当$1/>3\beta>$，算法总的运行时间为$O(\max(2^{0.3113n},2^{(0.0872+\beta)n}))$，内存空间为$O(2^{(0.5436-\beta)n})$。

因此，我们可获得时间空间权衡。现在，我们通过选择β来举例说明这个简单的算法。第一种选择是最小化内存所需数量，取β为任意接近$1/3$的值，可产生运行时间$O(2^{0.421n})$，内存空间$O(2^{0.211n})$。第二种是寻找一个最小的β值，例：$\beta\approx0.2972$。在这种情况下，我们可得运行时间$O(2^{0.385n})$，内存空间$O(2^{0.247n})$。第三种启发选择是与Schroeppel-Shamir需要同样的内存$O(2^{n/4})$，对于$\beta\approx0.2936$时，运行时间为$O(2^{0.381n})$。最后，我们可以通过取β接近于1/4来获得算法的时间复杂度$O(2^{0.338n})$，内存为$O(2^{0.294n})$。从以上讨论中可知，当$\beta<1/4$，时间复杂度变成$O(2^{(0.5872-\beta)n})$，再次增大。

以上的分析考虑的是串行的情况，接下来，介绍并行系统。当N个CPU同时执行程序的时候，因为并行系统共享总的代码量，所以串行中总的计算量被划分成不同的范围进行分散处理，列表的组合数从串行到并行可节省$C_n^{n/8}-C_{n-i}^{n/8}$，且不同CPU执行确定不同的取余列表$\delta_1,\delta_2,\delta_3,\delta_4$时，时间也不同，最少可节省$4(C_n^{n/8}-C_{n-i}^{n/8})$，最多可节省$4C_n^{n/8}$，且元素个数$n$越大，效果越明显。根据并行执行的特点，相对于串行来说，总运行时间可大致减少到$O(\max(2^{0.3113n},2^{(0.0872+\beta)n}))/N$，但实际效果比理论分析更好。

5.4 算法仿真

5.4.1 数据的选取

通过大量实验去测试串行、并行的算法，对比了两者的效果。为了方便展示效率，在1000以内随机选择了16个数字作为例子。这16个数字如下：752，52，90，23，49，302，221，67，809，398，78，567，237，943，532，909。

从所给的16个元素中随机选择8个相加作为和S。首先，进行四分法，组合数有$C_{16}^2=120$，随机确定R_1,R_2,R_3，分别取余M后，获得四个列表$\delta_1,\delta_2,\delta_3,\delta_4$，从中循环遍历找解。可通过串行算法找到解，有可能找到多解，但程序只要找到一个就可退出。为了便于比较结果，并行运算时使用同样的元素和S来找解。并行通过不同的

CPU 分配空间后，在不同范围内寻找，直到找到一个解即可退出，也可以找出所有的解，直到所有 CPU 程序全部运行结束再退出。

5.4.2 串行和并行运行时间的比较

表 5.1 串行、并行两种方式的运行时间

Transmission mode\processor	Id1	Id2	Id3	Id4	Id5	Id6	Id7	Id8
Serial(2260)	232							
Parallel(2260)	11	8	21	6	6	6	5	5
Serial(1911)	38530							
Parallel(1911)	14	6	10	8	6	8	5	8
Serial(2839)	241							
Parallel(2839)	16	7	7	5	6	6	8	5
Serial(2177)	219							
Parallel(2177)	21	7	19	8	6	7	5	7
Serial(3380)	568							
Parallel(3380)	6	8	6	7	7	7	6	6

表 5.1 列出了串行、并行算法的运行时间。串行只需要一个处理器，直到找到解即可退出；而并行根据范围划分为在 8 个处理器上同时执行，只要找到一个解就可退出，同时记录不同处理器所需时间。从表中明显可以看出，并行的时间远远小于串行时间，说明并行的效率大大提高。串行时间根据所取的 S 不同，执行结果也不同。背包个数 n 越大，串并行的效果就会越明显。

5.4.3 并行算法的找解情况

表 5.2：列出不同 S 时，并行算法找解的情况

S	Solution
(2260)	1:398 532 237 67 78 49 809 90 2:52 237 532 78 49 67 302 943 2:52 49 567 302 23 237 221 809
(1911)	2:52 221 398 49 67 78 809 237
(2839)	0:23 943 398 237 302 78 49 809 2:398 532 49 23 67 809 909 52 6:221 237 78 23 567 909 752 52 7:398 532 49 23 943 752 52 90
(2177)	0:23 78 221 398 49 67 809 532 2:52 49 398 78 23 67 567 943 2:52 49 532 67 23 78 809 567 3:52 90 221 237 78 23 567 909 5:752 90 67 237 78 23 398 532
(3380)	1:90 302 909 237 23 67 809 943 6:752 52 567 532 49 221 809 398

表 5.2 列出了在不同的S下，解可能是唯一或是多个。当解是唯一的时候，我们只能在一个处理器上找到一个解；而当是多解的时候，可能在同时执行的多个处理器上都能找到解，或者在一个处理器上找到多个解。产生这种情况的原因就是 S 有多种组合情况。如何有效地避免多解，可考虑选取元素的时候，使得值之间分散的大些。Coster 等人[57]证明当背包密度小于 0.9408 时，背包方案易受低密度子集合的攻击；大于 1 的时候，会造成解密的不唯一性。因此，安全的背包密度一般需要保持在（0.9408，1）范围内。

5.5 本章小结

本章首先介绍格基规约算法用来分析背包加密系统的方法，并给出详细的分析步骤，然后用第 3 章改进的规约算法再次分析，由于分块l次规约效果好于 LLL 规约算法，因此可用它来优化和提升效率。其次引进了 HGJ 的算法来求解 0-1 整数背包问题，就此算法提出了并行的实现方式，并给出详细的算法实现和分析，然后通过实验比较了两种方式的运行时间以及并行算法的找解情况。

第 6 章 结论

本章将对文章的主要内容进行总结，首先在 6.1 小节对第 3、4、5 三章进行概括，分别叙述了格基规约算法的提高，在球译码算法的初始化半径中以及密码分析中的应用和好处。6.2 小节将对格基规约算法以及其应用部分的研究和发展提供方向。

6.1 总结

本文主要围绕格基规约的思想进行叙述和展开，开篇先介绍格基规约的研究背景，发展现状及其主要贡献领域；其次介绍格基规约的相关理论知识，原理及可用来解决的问题；后 3 章主要介绍格基规约算法的改进以及可以应用的领域以及好处，即本文所做的主要贡献。

第 3 章主要对格基规约的 LLL 算法进行改进。针对分块 LLL 算法的特点，可发现规约时间减少，但约减基的效率略差；而 l 次规约算法的优点是约减效率提高，但规约时间有所增加。基于二者的利与弊，综合考虑汲取其优点，即形成分块 l 次格基规约算法。就是在分块 LLL 算法中用 l 次规约方法来代替 LLL 规约方式，以达到约减效率和运行时间相对平衡的状态。l 次规约的规律是在约减完两个相邻的向量之后，用后面一个向量与相隔一个的前一个向量继续进行规约，一直往前重复直到回溯到 l 次规约，$l=1$相当于 LLL 规约，$l>1$表示往前规约的次数，其规约的复杂度与规约次数 l 直接相关。分块 LLL 是把一组向量分成向量个数为 k 的 m 个相邻的块，然后对每两个相邻的块进行 l 次规约，其规约时间和分块的大小直接相关。因此，分块 l 次规约算法可通过参数 l 和 k 来共同控制，达到约减效率和运行时间相对平衡。后面通过参数的不同变化来控制约减效果，以及通过不同算法的比较，可获得较好的规约基，且可以达到理论分析的效果。

第 4 章主要讲述了规约算法 LLL 应用在球译码算法的初始化半径部分，以及对球译码算法的半径在 LLL 算法的基础上进一步约减的方法。本章首先要通过 Hadamard 比率选取一组高质量的基来使得误差尽可能的小；其次，提出利用 QR 分解、LLL 规约、以及带预测技术的宽度优先搜索算法（K-Best）叠加的方式进行约减这组基，以至于找到一个尽可能短的约减向量作为球译码算法（SDA）的初始化半径；最后再通过 SDA 算法来求解，其运算过程等价于在格中寻找最近向量的问题。实验选取一万多的数据来进行比较六种不同初始化半径下的算法效率，通过比较不同维度下的访问节

点数量、不同维度下的程序运行时间以及不同维度下的初始化半径的结果。运行结果一目了然，同时也验证了理论分析与实践的一致性。

第 5 章先讲述了格基规约 LLL 算法用来分析背包密码系统的过程。然后为了更深入地了解背包的概念及原理，研究了其常规的生成背包算法。背包问题是一个 NP-complete 问题，于是当背包元素个数及背包和非常大时，快速求解背包问题是相对比较困难的。所以，针对此问题，提出对应算法的并行化来加速解决背包问题。实验比较了串行和并行算法的运行时间，可清晰地看出并行效率的优势，这当然和所选择的 CPU 数量也有很大的关系，$2^i = N$ 个 CPU 数量越多，并行运行的分块也就越多，而每一块运行找解的速度也就越快，应用的时间也就越少，成反比的一个状态。实验又列出了，当和 S 确定后的找解情况，若 S 是由若干个元素生成的，则一定有解，且解有可能不唯一；若 S 是随机生成的一个数，则可能有解，也可能没有。

6.2 展望

本文对格基规约 LLL 算法的变体进行改进，寻找约减效率和运行时间相对平衡的一种算法。针对 LLL 算法的改进，仍存在很多变体只是单一地考虑了约减效率或者运行时间，从而导致另一方面的不如意，这在实际应用中的参考价值不大。因此，改进算法应尽可能的综合考虑，片面虽然也是一种思想，但可以再完美些，以达到利用的价值。改进观点价值更高的话，利用价值相对就会更高，以至于产生更多优质的学术创作，进而利益最大化。

对于创新或改进效率明显的算法，可考虑它的实际应用。譬如：分块 l 次规约就可以应用在球译码算法的半径初始化中以及基于格的 GGH 密码系统的分析中等，前者可以获得更小的半径，因此运行时间也随之减少，后者可以更快速且更准确地恢复明文内容。而改进的 SDA 也可以考虑应用在 V-BLAST 系统检测中，即无线通信。像这样类似的应用还有很多，都可以给具体应用带来明显的效果。在之后的研究中，针对格基，通信等领域，我们会综合考虑算法的创新价值，同时把更多优化算法运用在实际中，以便给学术界带来可供参考的价值。

参 考 文 献

[1] J. L. Lagrange, Recherches d’arithmétique[M], Nouv. Mém, Acad, 1773：265-312.

[2] C. F. Gauss, Disquisitiones Arithmeticae, Leipzig, 1801.

[3] C. Hermite, Extraits de letters de M.Ch. Hermite a M.Jacobi sur differents objects de la theorie des nombres, deuxieme lettre. J. Reine Angew. Math., 40, 1850: 279-290.

[4] A. Korkine, G. Zolotareff, Sur les formes quadratiques, Math. Ann., 1873: 336-389.

[5] A. Korkine, G. ZolotareIf, Sur les formes quadratiques positives quaternaires, Math. Ann., 5, 1872: 581-583.

[6] H. Minkowski, Geometrie der Zahlen, Teubner, Leipzig, 1910.

[7] J. W. S. Cassels, An Introduction to the Geometry of Numbers, Berlin Gottingen Heidelberg: Springer, 1959.

[8] 王小云, 刘明洁. 格密码学研究[J]. 密码学报, 2014, 1(1): 13－27.

[9] A. K. Lenstra, H. W. Lenstra and L. Lovasz, Factoring polynomials with rational coefficients, Mathematiche Annalen 261, 1982：515-534.

[10] C. P. Schnorr, A hierarchy of polynomial time basis reduction algorithm, Theoretical Computer Science, 53, 1987: 201-224.

[11] M. Seysen, Simultaneous reduction of a lattice basis and its reciprocal basis, Combinatorica 13, 1993: 363-376.

[12] C. P. Schnorr, M. Euchner, Lattice basis reduction: improved practical algorithms and sovling subset sum problems, Mathematical Programming 66, 1994: 181-199.

[13] C. P. Schnorr, Block reduced lattice bases and successive minima, Combinatorics, Probability and Computing, 3, 1994: 507-533.

[14] C. P. Schnorr, H. H. Horner, Attacking the Chor-Rivest cryptosystem by improved lattice reduction, In: Advances in Cryptology -EUROCRYPT’95, LNCS 921,1995: 1-12.

[15] H. Koy, C. P. Schnorr, Segment LLL-Reduction of lattice bases, Cryptography and Lattices, Jose: ph H. Silverman (Ed.), Springer-Verlag, 2001: 67-80.

[16] H. Koy, C. P. Schnorr, Segment LLL Reduction with floating point ortho-, Cryptography and Lattices, Joseph H.Silverman (Ed.), Springer.Verlag, 2001: 81-96.

[17] 余位驰, 何大可, 一种新型 l 次格基规约算法, 铁道学报, 29(1), 2007: 50-54.

[18] F. Fontein, M. Sichael, U. Wagner, PotLLL: a polynomial time version of LLL with deep insertions, Des, Codes Cryptography, 73(2): 2014: 355-368.

[19] T. Espitau, A. Joux, Adaptive precision LLL and potential-LLL reductions with interval arithmetic. IACR Cryptology ePrint Archive, 2016: 528.

[20] U. Fincke, M. Pohst. A procedure for determining algebraic integers of given norm[C]. In: Hulzen J A ed. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1983: 194-202.

[21] M. Pohst. On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications[J]. ACM Sigsam Bulletin, 1981: 37-44.

[22] R. Kannan, Improved algorithms for integer programming and related lattice problems[C]. In: Proceedings of the 15th Annual ACM Symposium on Theory of Computing. New York: ACM, 1983: 193-206.

[23] B. Helfrich. Algorithms to construct Minkowski reduced and Hermite reduced lattice bases[J]. Theoretical Computer Science, 41, 1985: 125-139.

[24] G. Hanrot, D. Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm[C]. In: Menezes A ed. Advances in Cryptology-CRYPTO 2007. Springer Berlin Heidelberg, 2007: 170-186.

[25] G. Hanrot, D. Stehlé. Worst-case Hermite-Korkine-Zolotarev reduced lattice bases[J]. arXiv preprint arXiv:0801, 3331, 2008.

[26] D. Stehlé, M. Watkins, On the extremality of an 80-dimensional lattice[M]. In: Hanrot G, Morain F, Thomé E eds. Algorithmic Number Theory. Springer Berlin Heidelberg, 2010: 340-356.

[27] N. Gama, P. Q. Nguyen, O. Regev. Lattice enumeration using extreme pruning[C]. In: Gilbert H ed. Advances in Cryptology-EUROCRYPT 2010, Springer Berlin Heidelberg, 2010: 257-278.

[28] D. Micciancio, P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations[J]. Siam Journal on Computing, 42(3), 2013: 1364-1391.

[29] M. Ajtai, R. Kumar, D. Sivakumar. A sieve algorithm for the shortest lattice vector problem[C]. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing. New York: ACM, 2001: 601-610.

[30] O. Regev, Lecture Notes on Lattices in computer science[EB/OL], http://www.cs.tau.ac.il/odedr/teaching/lattices fall 2004/index. html. 2004.

[31] P. Q. Nguyen, T. Vidick, Sieve algorithms for the shortest vector problem are practical[J]. Journal of Mathematical Cryptology, 2008, 2(2): 181-207.

[32] D. Micciancio, P. Voulgaris. Faster exponential time algorithms for the shortest vector problem[C]. In: Charikar M ed. Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2010: 1468-1480.

[33] X. Y. Wang, M. J. Liu, C. L. Tian, et al. Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem[C]. In:Bruce S N, Cheung L, Chi Kwong Hui, eds. Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2011. New York: ACM, 2011: 1-9.

[34] F. Zhang, Y. B. Pan, G. R. Hu. A three-level sieve algorithm for the shortest vector problem[C]. In: SAC 2013-20th International Conference on Selected Areas in Cryptography, 2013.

[35] K. Hayasaka, K. Aoki, T. Kobayashi, T. Takagi, A construction of 3-dimensional lattice sieve for number field sieve over $GF(p^n)$ [J], IACR Cryptology ePrint Archive, 2015: 1179.

[36] A. Becker, T. Laarhoven, Efficient (ideal) lattice sieving using cross-polytope LSH[C]//International Conference on Cryptology in Africa, Springer International Publishing, 2016: 3-23.

[37] L. Babai, On Lovász' lattice reduction and the nearest lattice point problem[J]. Combinatorica, 6(1), 1986: 1-13.

[38] P. Klein, Finding the closest lattice vector when it's unusually close[C]. In: Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms. San Francisco: Society for Industrial and Applied Mathematics, 2000: 937-941.

[39] D. Aharonov, O. Regev. Lattice problems in NP∩coNP[J]. Journal of the ACM (JACM), 52(5), 2005: 749-765.

[40] Y. K. Liu, V. Lyubashevsky, D. Micciancio, On bounded distance decoding for general lattices[C], Ninth International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2006 and Tenth International Workshop on Randomization and Computation, RANDOM 2006. Springer Berlin, Heidelberg, 2006: 450-461.

[41] C. Dwork, Lecture notes: Lattices and their application to cryptography. Stanford University, Spring Quarter, 1998:pp36 (α version) .

[42] 北京大学数学系几何与代数教研室代数小组, 高等代数（第二版）[M], 高等教育出版社, 1988.

[43] J. C. Lagarias, Hendrik W. Lenstra Jr., C. P. Schnorr, Korkin-Zolotarev bases and successive minima of a lattice and its reciprocal lattice, Combinatorica 10(4), 1990: 333-348.

[44] B. Hassibi, H. Vikalo. On the sphere-decoding algorithm I. Expected complexity, IEEE Transactions on Signal Processing, 53, 2005: 2806-2818.

[45] O. Goldreich, S. Goldwasser, S. Halevi. Public-Key cryptosystems from lattice reduction problems, Advances in Cryptology-Crypto’97, 1997: 112-131.

[46] A. Hassibi, S. Boyd. Integer parameter estimation in linear models with applications to GPS, IEEE Trans. Signal Process, vol. 46, no. 11, Nov. 1998: 2938-2952.

[47] M. Grotschel, L. Lovasz, A. Schriver, Geometric algorithms and combinatorial optimization, Springer Verlag, 2nd ed., 1993.

[48] S. Qiao, Integer least squares: sphere decoding and the LLL algorithm. Proceedings of C3S2E-08, ACM International Conference Proceedings Series, edited by Bipin C. Desai. Concordia University, Montreal, Qc, May 12-13, 2008:3-28.

[49] M. Ajtai, The shortest cector problem in L2 is NP-hard for randomized reductions, Proceedings of the 30th Annual ACM symposium On Theory of Computing, 1998: 10-19.

[50] M. Pohst, On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications, ACMSIGSAM Bull, 15, 2008: 37-44.

[51] U. Fincke, M. Pohst, Improved methods for calculating vectors of short length in a Lattice, Including a complexity analysis, MathComp44, 1985: 463-471.

[52] J. Li, S. Chen, Reduced-complexity sphere decoding algorithm based on adaptive radius in each dimension[C]//Computer and Communications (ICCC), 2015 IEEE International Conference on. IEEE, 2015: 309-313.

[53] M. Karthikeyan, D. Saraswady, Reduced complexity sphere decoding using probabilistic threshold based Schnorr-Euchner enumeration[J], AEU-International Journal of Electronics and Communications, 70(4), 2016: 449-455.

[54] R. Schroeppel, A. Shamir, A T = O(2 n/2), S = O(2 n/4) algorithm for certain NP-Complete problems. SIAM J. Comput., 10(3), 1981: 456-464.

[55] J. C. Lagarias, A. M. Odlyzko, Solving low-density subset sum problems, Proceedings of IEEE symposium on the foundations of Computer Science, 1983: 1-10.

[56] M. J. Coster, A. Joux, B. A. LaMacchia, A. M. Odlyzko, C. P. Schnorr, J. Stern, Improved low-density subset sum algorithms, Computational Complexity,2, 1992: 111-128.

[57] R. Impagliazzo, M. Naor. Efficient cryptographic schemes provably as secure as subset sum. Journal of Cryptology, 9(4), 1996: 199-216.

[58] N. Howgrave-Graham, A. Joux, New generic algorithms for hard knapsacks. In EUROCRYPT’2010, 2010: 235-256.

[59] A. Becker, J-S. Coron, A. Joux, Improved generic algorithm for hard knapsacks, 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings, EUROCRYPT 2011: 364-385.

[60] C. P. Schnorr, T. Shevchenko, Solving subset sum problems of density close to 1 by “randomized” BKZ-reduction, IACR Cryptology ePrint Archive 2012: 620.

[61] H. Cohen, A course in computatonal algebric number theory[M], Second edition, Berlin: Springer-Verlag, 1993.

[62] Y. Chen, P. Nguyen, BKZ 2.0: Better lattice security estimates[J]. Advances in Cryptology-ASIACRYPT 2011, 2011: 1-20.

[63] C. P. Schnorr, M. Euchner, Lattice basis reduction: improved practical algorithms and sovling subset sum problems, Mathematical Programming 66, 1994: 181-199.

[64] U. Fincke, M. Pohst, Improved methods for calculating vectors of short length in a lattice, Including a complexity analysis, MathComp44, 1985: 463-471.

[65] E. Viterbo, J. Boutros, A universal lattice vode fecoder for fading channels,

Information Theory, 45(5), 1999: 1639-1642.

[66] X. Lou, Q. Zhou, Y. Chen, Research on low complexity K-best sphere decoding algorithm for MIMO systems, Wireless Personal Communications, 84(1), 2015: 547-563.

[67] K. Niu, K. Chen, J. Lin, Low-complexity sphere decoding of polar codes based on optimum path metric, IEEE Communications Letters, 18(2), 2014: 332-335.

[68] S. Barik, H. Vikalo, Sparsity-aware sphere decoding: algorithms and complexity analysis, IEEE Trans. Signal Processing 62(9), 2014: 2212-2225.

[69] A. Meyer, On the number of lattice points in a small sphere and a recursive lattice decoding algorithm, Des. Codes Cryptography, 66(1-3), 2013: 375-390.

[70] D. Seethaler, J. Jalden, C. Studer, et al, On the complexity distribution of sphere decoding. Information Theory, IEEE Transactions, 57(9), 2011: 5754-5768.

[71] C. Studer, A. Burg, H. Bolcskei, Soft-output sphere decoding: algorithms and VLSI implementation. IEEE Journal on Selected Areas in Communications, 26(2), 2008: 290-300.

[72] C. Studer, A. Burg, H. Bolcskei, Soft-input soft-output single tree-search sphere decoding. IEEE Transactions on Information Theory, 56(10), 2010: 4827-4842.

[73] Z. Guo, P. Nilsson, Algorithm and implentation of the K-best sphere decoding for MIMO detection[J]. IEEE Journal on Selected Areas in Commu., 24(3), 2006: 491-503.

[74] M. Shabany, P. G. Gulak, Scalable VLSI architecture for K-best lattice decoders, IEEE International Symposiumon Circuits and Systems(ISCAS), Seattle, WA: IEEE Press, 2008: 940-943.

[75] 周福才, 徐剑. 格理论与密码学, 北京：科学出版社, 2013: 95-100.

[76] S. Qiao, Integer least squares: sphere decoding and the LLL algorithm. Proceedings of C3S2E-08, ACM International Conference Proceedings Series, edited by Bipin C. Desai. Concordia University, Montreal, Qc, May 12-13, 2008: 23-28.

[77] Y. Nagashima, N. Kunihiro, Faster algorithm for solving hard knapsacks for moderate message length, Information Security and Privacy. ACISP 2012. Lecture Notes in Computer Science, vol 7372. Springer, Berlin, Heidelberg, 2012: 43-56.

[78] P. Q. Nguyen, O. Regev, Learning a parallelepiped: cryptanalysis of GGH and NTRU signatures. EUROCRYPT 2006, Lecture Notes in Computer Science, vol 4004, 2006: 271-288.

致 谢

在毕业论文即将完成之际，也意味着我在深圳大学的三年研究生学习生涯即将结束。从论文的选题，资料的搜集，以及整个论文的编排撰写中，非常感谢王平导师的悉心指导。读研期间，王平老师为我提供了优质的科研条件，学习环境。每周的学术交流会、不定期的学术观点分享与讨论对我的学术研究有着潜移默化的影响，使我对学术研究有了明确的方向和目标。王平老师严谨务实的态度、具有钻研性、预见性的学术认知以及和蔼可亲的魅力一直影响和感染着我，使我受益匪浅。

感谢王小民副教授对我的培养方案指点迷津，时刻关注我的学术研究进展，并及时给予宝贵意见。由于王小民副教授 3 月底临近退休离职，所以，也非常感谢陈剑勇教授愿意接收我，并成为我的新导师，认真负责我的毕业论文相关事宜。

感谢我实验室的同学们、师弟师妹们，你们的存在让实验室多了浓厚的学习氛围，使我对实验室有着依赖的感情；你们对我的学术研究也提供了宝贵的建议，评价以及支持，使我对学术研究产生了莫大的热情。同时，也感谢在生活中，宿舍中与我共同分享经验的舍友们，因为有你们，使我在深圳大学的生活变得丰富多彩。

在此，我还要特别感谢我的父母，他们总是给予我无尽的关怀和无私的爱。无条件地培养支持我，让我在漫长的人生旅途中使心灵有了虔敬的归依。在未来的日子里，我会努力用热情与爱来回报我的家人。

最后，我想把最真挚的感谢再次送给每一个帮助过我的人。

攻读硕士学位期间的研究成果

[1] Na Zhou, Ping Wang, Sphere Decoding Algorithm Based on Improved Radius Selection[C],The 12th International Conference on Information Security and Cryptology, 2016/12/1：571-583.

[2] Na Zhou, Ping Wang, A Time-Quality Tradeoff Lattice Reduction Based on Segment Reduction, International Journal of Information Security (IJIS)(投稿中).

[3] Na Zhou, Ping Wang, Parallel Generic Algorithms for Hard Knapsacks，Computational and Applied Mathematics（CAM）(投稿中).

[4] 周娜，王平，一种基于格的密码分析方法及装置，发明专利，专利号：201710083915.7，（申请受理中）。