Technical Report 1283

Basis Reduction Algorithms and Subset Sum Problems

Brian A.LaMacchia

MIT Artificial Intelligence Laboratory

Basis Reduction Algorithms and Subset Sum Problems

by

Brian A. LaMacchia

Artificial Intelligence Laboratory
and
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology

Abstract

This thesis investigates a new approach to lattice basis reduction suggested by M. Seysen. Seysen's algorithm attempts to globally reduce a lattice basis, whereas the Lenstra, Lenstra, Lovász (LLL) family of reduction algorithms concentrates on local reductions. We show that Seysen's algorithm is well suited for reducing certain classes of lattice bases, and often requires much less time in practice than the LLL algorithm. We also demonstrate how Seysen's algorithm for basis reduction may be applied to subset sum problems. Seysen's technique, used in combination with the LLL algorithm, and other heuristics, enables us to solve a much larger class of subset sum problems than was previously possible.

Thesis Supervisor:	Silvio Micali Associate Professor of Computer Science and Engineering
Company Supervisor:	Andrew M. Odlyzko AT&T Bell Laboratories

This report is a revised version of a thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology in May, 1991.

Acknowledgments

This thesis could not have been completed without the support of a great many people. I wish to take this opportunity to express my appreciation for their help throughout this project.

First, my sincerest thanks to **Andrew M. Odlyzko**, my company supervisor at AT&T Bell Laboratories. This thesis would not have been begun, let alone finished, without Andrew's guidance. Throughout my time at Bell Labs Andrew has been my supervisor, mentor and friend; his initial suggestion that I look at applications of Seysen's algorithm to subset sum problems was the basis for this work. Andrew's support and encouragement were without bound.

I also thank **Matthijs J. Coster** for his assistance throughout this project. Matthijs suggested numerous variants of Seysen's algorithm and helped design Algorithm **SL** for solving subset sum problems. His comments on early drafts of this thesis were also quite helpful.

Many people have commented on different aspects of this project while work was in progress. In particular, I'd like to thank **Jeff Lagarias**, **Claus Schnorr** and **Warren Smith** for their assistance.

This work was performed under the auspices of the Center for Mathematical Sciences Research at AT&T Bell Laboratories. I thank **Mike Garey**, director, and the other members of the Math Center for the research environment they maintain and their constant support during my tenure at Bell Labs.

Many thanks to **Professor Silvio Micali**, my thesis supervisor at MIT, for overseeing this work. Silvio's enthusiasm for this project never wavered, and his comments when I began writing this thesis were very useful.

Thanks also **Professors Harold Abelson** and **Gerald J. Sussman**, and the members of **Project MAC** (Mathematics and Computation), for their support upon my return from Bell Labs. In particular, thanks to **Andy Berlin**, **Elizabeth Bradley**, **Mike Eisenberg**, **Mark Freidman**, **Arthur Gleckler**, **Chris Hanson**, **Elmer Hung**, **Stefan Kozlowski**, **Bill Rozas**, **Sameer Shalaby**, **Thanos Siapas**, **Panayotis Skordos**, **Franklyn Turbak**, **Henry Wu**, and **Feng Zhao**. They are a very special group of people.

This project was performed as part of the MIT VI-A Internship Program. The director of the VI-A Program is **Kevin O'Toole**. **Professor Frederic Morgenthaler** serves as faculty liaison between the VI-A Office and AT&T Bell Laboratories. At Bell Labs, the VI-A Program is administered by **J. Michael Milner** and **José L. Reyes**. My thanks to all of them for making VI-A at Bell Labs so successful.

This paper describes research conducted at AT&T Bell Laboratories, as part of the MIT VI-A Internship Program, and at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-89-J-3202.

Contents

List of Tables

X

List of Figures

Chapter 1

Introduction

In 1985 Lagarias and Odlyzko [26] developed a general attack on knapsack cryptosystems which reduces solving subset sum problems to the problem of finding the Euclidean-norm shortest nonzero vector in a point lattice. Recent improvements to this attack [12, 19] have stimulated interest in finding lattice basis reduction algorithms well-suited to the lattices associated with subset sum problems. This thesis studies a new approach to lattice basis reduction originally developed by M. Seysen [38]. Seysen's reduction algorithm was initially developed to find simultaneously good bases of a lattice and its dual lattice. However, it may also be successfully applied to solving subset sum problems, especially when combined with other known reduction methods. Using a collection of techniques, including Seysen's algorithm, we show that it is possible to solve in practice a much larger class of subset sum problems than was previously possible.

1.1 Point Lattices

Let B be a set of vectors $(\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n})$ in $\mathbb{R}^n$. If these vectors are independent, then they form a basis of $\mathbb{R}^n$ and any point $\mathbf{x}$ in n-space may be written as a linear combination of vectors in B:

$$\mathbf{x} = \sum_{i=1}^{n} r_i \mathbf{b}_i, \quad \text{for } r_i \in \mathbb{R}, 1 \leq i \leq n.$$

Consider the set of points $L \subset \mathbb{R}^n$ which may be written as the sum of *integer* multiples of the basis vectors:

$$L = \left\{ \mathbf{x} = (x_1, x_2, \ldots, x_n) \,:\, \mathbf{x} = \sum_{i=1}^{n} z_i b_i, \quad \text{for } z_i \in \mathbb{Z}, 1 \leq i \leq n \right\}.$$

We call this set L the *point lattice* (or just *lattice*) described by the basis B.

Point lattices are pervasive structures in mathematics, and have been studied extensively. See [25], for example, for a survey of the field. In the area of combinatorial mathematics alone it is possible to phrase many different problems as questions about lattices. Integer programming [20], factoring polynomials with rational coefficients [27], integer relation finding [16], integer factoring [35], and diophantine approximation [36] are just a few of the areas where lattice problems arise. In some cases, such as integer programming existence problems, it is necessary to determine whether a convex body in $\mathbb{R}^n$ contains a lattice point (for some specific lattice). In other cases the items of interest are short vectors in the lattice. As we shall see below, for certain cryptographic applications, we would like to be able to quickly determine the Euclidean-norm shortest nonzero vector in a lattice.

It is important to note that the difficulty of finding the Euclidean-norm shortest nonzero vector in a lattice is an open question. If $\mathbf{x} = (x_1, \ldots, x_n)$, then the *sup-norm* of $\mathbf{x}$, denoted $||\mathbf{x}||_\infty$, is defined as

$$||\mathbf{x}||_\infty = \max_{1 \leq i \leq n} |x_i|.$$

It is known that finding the sup-norm shortest nonzero vector in a lattice is NP-hard [5]. Based on this evidence, we suspect that finding the Euclidean-norm shortest nonzero vector for any given lattice is also computationally difficult. However, it may be possible to find the shortest nonzero vector for many lattices quickly. Indeed, current techniques for finding short vectors are slow in theory but often perform well in practice.

The remainder of this chapter establishes the environment for our study of lattices and specific applications to cryptography. Section 1.2 discusses *reduced* bases of lattices and *lattice reduction theory*. Section 1.3 mentions some of the algorithms which currently exist for computing a reduced lattice basis B' given a basis B of a particular point lattice. In particular, we detail the operation of the Lenstra-Lenstra-Lovász (LLL) basis reduction algorithm [27], which is currently the best known method for finding short vectors in a lattice.

1.2 Reduced Lattice Bases

Any lattice L may be described by many different lattice bases. Let $B_1, B_2, \ldots$ be distinct sets of vectors, all of which form bases of lattice L. We can imagine that there exists some ordering or ranking of the bases B_i, and thus one or more of the B_i might be considered "good" lattice bases of L. *Lattice reduction theory* deals with identifying "good" lattice bases for a particular lattice. If we are given a basis B which describes L, we would like to *reduce* B to basis B', also describing L, where B' is a "good" lattice basis in the sense of some reduction theory.

There are two classical lattice reduction theories, one due to Korkin and Zolotarev [23, 24] and one due to Minkowski [29]. A basis $B = (\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n})$ of lattice L is said to be *Minkowski-reduced* if

1. $\mathbf{b_1}$ is the shortest nonzero vector in L.

2. For $2 \leq i \leq n$, $\mathbf{b_i}$ is the shortest vector in L such that $(\mathbf{b_1}, \ldots, \mathbf{b_i})$ may be extended to a basis of L.

Minkowski-reduced lattice bases always contain the shortest nonzero vector in the lattice. Subsequent basis vectors $\mathbf{b_i}$ are selected by choosing the shortest lattice vector in

L which is not a linear combination of the already selected basis vectors $\mathbf{b_1}, \ldots, \mathbf{b_{i-1}}$. If $\mathbf{b_i} = \sum_{j=1}^{i-1} z_j \mathbf{b_j}, z_j \in \mathbb{Z}$, then it would be impossible to extend $(\mathbf{b_1}, \ldots, \mathbf{b_i})$ to be a basis of L.

The definition of *Korkin-Zolotarev reduction* is similar to that of Minkowski. We say a basis $B = (\mathbf{b_1}, \ldots, \mathbf{b_n})$ is Korkin-Zolotarev reduced if it satisfies the following three conditions:

1. $\mathbf{b_1}$ is the shortest nonzero vector in L.

2. For $2 \leq i \leq n$, let S_i be the $(i-1)$-dimension subspace spanned by the basis $(\mathbf{b_1}, \ldots, \mathbf{b_{i-1}})$. Let $S_i^{\perp}$ be the orthogonal complement of S_i in $\mathbb{R}^n$. Finally, let $P_i(L)$ denote the orthogonal projection of L onto $S_i^{\perp}$. Then choose $\mathbf{b_i}$ such that $P_i(\mathbf{b_i})$ is the shortest nonzero vector in $P_i(L)$.

3. *Size reduction condition.* For $1 \leq i < j \leq n$,
$$|\langle P_i(\mathbf{b_i}), P_i(\mathbf{b_j})\rangle| \leq \frac{1}{2}\|P_i(\mathbf{b_i})\|^2,$$
where $P_1(\mathbf{x}) = \mathbf{x}$.

In the definition of Minkowski reduction, successive basis vectors $\mathbf{b_i}$ are added to the lattice basis only if $\mathbf{b_i}$ is the shortest vector in the lattice which will allow the basis to be extended. In Korkin-Zolotarev reduction, though, successive basis vectors $\mathbf{b_i}$ are chosen based on their length in the orthogonal complement of the space spanned by the previous basis vectors $\mathbf{b_1}, \ldots, \mathbf{b_{i-1}}$.

Depending on the specific problem, we may find either, both, or neither of the above definitions of "good" lattice bases is sufficient. Certainly if the goal is to find the shortest nonzero vector in the lattice, as is the case with subset sum problems (see Chapter 3), we could use either Minkowski reduction or Korkin-Zolotarev reduction as a measure of how "good" a particular lattice basis is. If the item of interest involves multiple vectors in the lattice, one definition may be preferred over the other for that particular application.

1.3 Lattice Basis Reduction Algorithms

Although both Minkowski reduction and Korkin-Zolotarev reduction provide frameworks for studying lattice basis reduction, computing a reduced lattice basis (in either sense) is in general a difficult problem. Currently, there are no known polynomial-time algorithms for finding either a Minkowski or a Korkin-Zolotarev reduced basis for a given lattice L. If such an algorithm existed, then we would be able to find the Euclidean-norm shortest nonzero vector in L in polynomial time by finding the reduced basis, for which $\mathbf{b_1}$ is the desired vector. Thus, any polynomial-time lattice basis reduction algorithm we use will not be able to satisfy the strict conditions of Minkowski or Korkin-Zolotarev reduction.

Techniques for finding relatively small vectors in a lattice have been known for some time (see [22] for example); it was not until recently, though, that a fast algorithm was known which was guaranteed to produce lattice bases with relatively short vectors. In [27] Lenstra, Lenstra and Lovász described a polynomial-time algorithm for transforming a give lattice basis $B = (\mathbf{b_1}, \ldots, \mathbf{b_n})$ of lattice L into an *LLL-reduced* lattice basis $B' = (\mathbf{b'_1}, \ldots, \mathbf{b'_n})$. A basis B' is LLL-reduced if it has the following two properties:

$$|\mu_{i,j}| \leq \frac{1}{2} \qquad \text{for } 1 \leq j < i \leq n,\ \mu_{i,j} = \frac{(\mathbf{b'_i}, \mathbf{b^*_j})}{(\mathbf{b^*_j}, \mathbf{b^*_j})}, \tag{1.1}$$

$$\left\|\mathbf{b^*_i} + \mu_{i,i-1}\mathbf{b^*_{i-1}}\right\|^2 \geq y \left\|\mathbf{b^*_{i-1}}\right\|^2 \quad \text{for } 1 < i \leq n, \tag{1.2}$$

where the parameter $y \in (\frac{1}{4}, 1)$ and $\mathbf{b^*_j} = \mathbf{b'_j} - \sum_{i=1}^{j-1} \mu_{i,j}\mathbf{b^*_i}$ (That is, $B^* = (\mathbf{b^*_1}, \ldots, \mathbf{b^*_n})$ is a Gram-Schmidt orthogonalized basis generated from B). Notice that LLL reduction is similar to but weaker than Korkin-Zolotarev reduction.

The Lenstra-Lenstra-Lovász basis reduction algorithm converts lattice basis B into B' by performing two types of transformations. In a *size-reduction* step, LLL finds the largest j such that there exists an $i > j$ and $\mu_{i,j}$ violates Equation 1.1. By performing the transformation

$$\mathbf{b_i} \longleftarrow \mathbf{b_i} - \lceil \mu_{i,j} \rfloor \mathbf{b_j},$$

where $\lceil \cdot \rfloor$ denotes the nearest-integer function, we find that the new value of $\mu_{i,j}$ is $\leq \frac{1}{2}$. In the *exchange* step, LLL searches for the smallest value of i such that $\mathbf{b_{i-1}}$ and $\mathbf{b_i}$ fail to satisfy Equation 1.2. Here LLL swaps the two vectors ($\mathbf{b_{i-1}} \longleftarrow \mathbf{b_i}$ and $\mathbf{b_i} \longleftarrow \mathbf{b_{i-1}}$) to force compliance with the second LLL-reduced condition. The LLL basis reduction algorithm alternately performs size-reduction and exchange steps until both Equations 1.1 and 1.2 are satisfied, at which point the algorithm halts.

LLL-reduced lattice bases satisfy a number of bounds. In particular, if y is the global LLL constant, then the first vector in the reduced lattice basis $\mathbf{b_1}$ satisfies:

$$\|\mathbf{b_1}\| \leq \left(\frac{4}{4y-1}\right)^{n-1} \|\mathbf{x}\|, \quad \text{for all } \mathbf{x} \in L, \mathbf{x} \neq \mathbf{0}.$$

In particular, for $y = \frac{3}{4}$ (the value used in [27]), we have

$$\|\mathbf{b_1}\| \leq 2^{n-1} \|\mathbf{x}\|, \quad \text{for all } \mathbf{x} \in L, \mathbf{x} \neq \mathbf{0}.$$

Thus the length of $\mathbf{b_1}$ is at most an exponential multiple of the length of the shortest nonzero vector in the lattice. (Similar bounds exist for the other vectors in the reduced basis.) In practice, the LLL algorithm usually performs much better than this exponential bound, although example lattice bases are known which cause the LLL algorithm to exhibit worst-case performance.

Most of the work on lattice basis reduction algorithms since the introduction of LLL has focused on improving and extending the technique. For example, the version of LLL described above requires rational arithmetic (the $\mu_{i,j}$ variables in particular must be stored as fractions); multiprecision floating-point numbers are usually used to reduce the computation requirements, but may introduce error into the calculations. One method of reducing the multiprecision requirements is described in [33]. Similarly, [32] showed how to modify the LLL algorithm so that the set of input vectors can be linearly dependent. Hierarchies of LLL-type algorithms have been investigated [34], stretching from LLL-reduction at one end of the spectrum to Korkin-Zolotarev reduction at the other. However, little effort has been expended on looking at algorithms not derived from or similar to LLL.

This thesis examines an approach to lattice basis reduction of different structure than that of the LLL algorithm and related methods. This method was originally suggested by M. Seysen [38] to simultaneously produce a reduced basis and a reduced dual basis for some lattice L. Where the LLL algorithm concentrates on local optimizations to produce a reduced lattice, the Seysen approach considers the entire basis for methods of optimization. (Recall that the LLL *size-reduction* steps only consider $\mu_{i,j}$ for the smallest possible j, and that only adjacent vectors $\mathbf{b_{i-1}}$ and $\mathbf{b_i}$ may be *exchanged.*)

Chapter 2 describes the first phase of the research, in which Seysen's basis reduction algorithm and multiple variants were implemented and examined. Theoretical and empirical analyses of the fixed components of Seysen's algorithm are given. For those parts of the algorithm which are permitted to vary, we examine some of the possible variations, and look at the effect of these changes on the performance of the algorithm. Possible extensions of the algorithm are also discussed.

The motivation behind our study of Seysen's lattice basis reduction algorithm is presented in Chapter 3. It is known that certain *subset sum problems* may be solved by finding the shortest nonzero vector in a particular lattice (the lattice is generated based on the specific construction of the subset sum problem). The best methods previously known for reducing subset sum problem lattices [26, 33] involve the LLL algorithm and some other heuristics, and are not very successful for $n > 25$ (n is the size of the subset sum problem to be solved). Chapter 3 details experiments which used Seysen's algorithm in combination with the LLL algorithm and other heuristics to solve a much greater range of subset sum problems.

Chapter 2

The Seysen Basis Reduction Algorithm

In 1990, Martin Seysen proposed a new method for performing lattice basis reduction [38]. *Seysen's basis reduction algorithm* (or just *Seysen's algorithm*) differs from the LLL algorithm and its variants in that it considers all vectors in the lattice simultaneously, and performs operations on those vectors which will reduce the lattice according to some measure. Recall that the LLL algorithm works locally on the lattice it is reducing; LLL will only perform an operation on two vectors which are adjacent to each other in the ordered lattice basis.

Seysen was motivated to create a new method for basis reduction by a desire to find a better way to simultaneously reduce a lattice and its reciprocal (or dual) lattice. If lattice L is defined by basis vectors $\mathbf{b_1}, \ldots, \mathbf{b_n}$, then the dual lattice L^* of L is defined by basis vectors $\mathbf{b_1^*}, \ldots, \mathbf{b_n^*}$, where

$$\begin{aligned} (\mathbf{b_i}, \mathbf{b_i^*}) &= 1, \\ (\mathbf{b_i}, \mathbf{b_j^*}) &= 0, \quad \text{for } i \neq j. \end{aligned} \tag{2.1}$$

Now consider what happens in the dual lattice when we perform a *row move* on $\mathbf{b_i}$ and $\mathbf{b_j}$ in L. (A *row move* is any operation which adds a constant multiple of one lattice basis vector to another basis vector.) Let $\mathbf{b_j'} = \mathbf{b_j} + \lambda \mathbf{b_i}$, where $\lambda \in \mathbb{Z}$. We consider what changes must occur in the dual lattice basis vectors $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$, since Equation 2.1 must hold at all times. For $k \neq i$, we find that:

$$\mathbf{b}_k^{*\prime} = \mathbf{b}_k^*,$$

since

$$\begin{aligned}(\mathbf{b}_j', \mathbf{b}_k^*) &= (\mathbf{b}_j + \lambda \mathbf{b}_i, \mathbf{b}_k^*),\\ &= (\mathbf{b}_j, \mathbf{b}_k^*) + \lambda(\mathbf{b}_i, \mathbf{b}_k^*),\\ &= 0.\end{aligned}$$

For $k = i$, however, this is not the case:

$$\mathbf{b}_i^{*\prime} = \mathbf{b}_i^* - \lambda \mathbf{b}_j^*,$$

since

$$\begin{aligned}(\mathbf{b}_j', \mathbf{b}_i^{*\prime}) &= (\mathbf{b}_j + \lambda \mathbf{b}_i, \mathbf{b}_i^{*\prime}),\\ &= (\mathbf{b}_j, \mathbf{b}_i^{*\prime}) + \lambda(\mathbf{b}_i, \mathbf{b}_i^{*\prime}),\\ &= (\mathbf{b}_j, \mathbf{b}_i^* - \lambda \mathbf{b}_j^*) + \lambda(\mathbf{b}_i, \mathbf{b}_i^* - \lambda \mathbf{b}_j^*),\\ &= (\mathbf{b}_j, \mathbf{b}_i^*) - \lambda(\mathbf{b}_j, \mathbf{b}_j^*) + \lambda(\mathbf{b}_i, \mathbf{b}_i^*) - \lambda^2(\mathbf{b}_i, \mathbf{b}_j^*),\\ &= 0 - \lambda + \lambda - 0,\\ &= 0.\end{aligned}$$

Thus, when we add $\lambda \mathbf{b}_i$ to $\mathbf{b}_j$ in the basis of lattice L, we must subtract $\lambda \mathbf{b}_j^*$ from $\mathbf{b}_i^*$ in the basis of L^*. It is easy to see that if lattice L is reduced with the LLL algorithm, the resulting reduced lattice may have a dual in which some basis vector is quite large, as no attempt is ever made to consider the size of the dual basis when row moves are being performed. Seysen's algorithm attempts to choose row moves that reduce both the lattice and its dual.

We now outline the basic operation of Seysen's algorithm. Let L and L^* be a lattice and its dual which we wish to simultaneously reduce. Let A and A^* be the associated quadratic forms of L and L^*, respectively:

$$A = [a_{i,j}]_{1\le i,j\le n} = [(\mathbf{b_i}, \mathbf{b_j})]_{1\le i,j\le n},$$
$$A^* = \left[a^*_{i,j}\right]_{1\le i,j\le n} = \left[(\mathbf{b^*_i}, \mathbf{b^*_j})\right]_{1\le i,j\le n}.$$

The element $a_{i,j}$ of matrix A is the inner product of the basis vectors $\mathbf{b_i}$ and $\mathbf{b_j}$ of lattice L. Notice that A and A^* are inverses of each other and are symmetric.

If L is the lattice defined by basis B, then any other basis B' of L may be written as:

$$B' = B\,T,$$

where $T \in SL_n(\mathbb{Z})$ (i.e. T is an $n \times n$ integer matrix with determinant 1). The quadratic form A' associated with B' may similarly be derived from A:

$$A' = T^t\,A\,T.$$

For any quadratic form A, define the *Seysen measure* $S(A)$ as follows:

$$S(A) = \sum_{i=1}^{n} a_{i,i}a^*_{i,i} = \sum_{i=1}^{n} \|\mathbf{b_i}\|^2 \|\mathbf{b^*_i}\|^2. \tag{2.2}$$

A basis B is then *S-reduced* if:

$$S(A) \le S(T^t\,A\,T), \quad \text{for all } T \in SL_n(\mathbb{Z}). \tag{2.3}$$

We suspect that it is computationally difficult to find the optimal transformation matrix T for a given basis B. Consider, however, the class of transformation matrices $T^\lambda_{i,j}$ defined by:

$$T^\lambda_{i,j} = I_n + \lambda U_{i,j}, \quad \text{where } i \ne j,\ \lambda \in \mathbb{Z},$$

I_n is the n-dimensional identity matrix, and

$$U_{i,j} = [u_{k,l}]_{1\le k,l\le n}, \quad \text{where } u_{k,l} = \begin{cases} 1 & \text{if } k = i \text{ and } l = j, \\ 0 & \text{if } k \ne i \text{ or } l \ne j. \end{cases}$$

(The matrix $U_{i,j}$ has exactly one nonzero entry. Matrix $T_{i,j}^{\lambda}$ has diagonal entries of 1 and exactly one nonzero off-diagonal entry.) Right-multiplying B by any $T_{i,j}^{\lambda}$ simply adds λ times the i^{th} column of B to the j^{th} column of B. If the columns of B are the basis vectors $\mathbf{b_i}$, then $B\ T_{i,j}^{\lambda}$ is simply the transformed basis $B' = (\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_{j-1}}, \mathbf{b_j} + \lambda\mathbf{b_i}, \ldots, \mathbf{b_n})$.

Since it is easy to perform calculations with $T_{i,j}^{\lambda}$ transformations, we focus our attention on products of one or more $T_{i,j}^{\lambda}$ transformation matrices. It can be shown that every $T \in SL_n(\mathbb{Z})$ may be written as such a product:

$$SL_n(\mathbb{Z}) = \left\{ T \ : \ T = \prod_k T_{i_k,j_k}^{\lambda_k}, 1 \leq k < \infty \right\}.$$

We call a quadratic form S_2-*reduced* if

$$S(A) \leq S(T_{j,i}^{\lambda}\ A\ T_{i,j}^{\lambda}), \quad \text{for } 1 \leq i,j \leq n, \text{ for } \lambda \in \mathbb{Z}.$$

Seysen suggests the following algorithm for S_2-reducing a quadratic form:

```
while (A is not $S_2$-reduced)
  do
    choose $i,j$ such that $\exists\ \lambda \in \mathbb{Z}$ with

        $S\left(T_{j,i}^{\lambda}\ A\ T_{i,j}^{\lambda}\right) < S(A)$

    let

        $\lambda = \left\lceil \frac{1}{2}\left(\frac{a_{i,j}^*}{a_{j,j}^*} - \frac{a_{i,j}}{a_{i,i}}\right) \right\rfloor$

    let

        $A = T_{j,i}^{\lambda}\ A\ T_{i,j}^{\lambda}$
```

where $\lceil \cdot \rfloor$ denotes the nearest-integer function. This procedure for S_2-reducing a quadratic form is Seysen's basis reduction algorithm.

To date, little has been proven concerning the performance of Seysen's algorithm. There are no known bounds on the Seysen measure of an S_2-reduced basis (although bounds have been proven for S-reduced lattice bases), nor on the length of the shortest nonzero vector in the basis. The running time of Seysen's algorithm is clearly bounded if the lattice basis consists of only integer vectors, but it is not known if the algorithm even terminates for basis vectors with real coefficients. However, preliminary experiments performed by Seysen on lattices of dimension $n \leq 30$ suggest that this technique may be faster than the LLL algorithm and yield bases with shorter vectors. Based on these observations, a comprehensive investigation of theoretical and practical aspects of Seysen's algorithm was undertaken. This chapter details the results of our study of Seysen's basis reduction algorithm.

Section 2.1 below discusses the theoretical underpinnings of Seysen's algorithm. Empirical tests performed with various versions of the Seysen algorithm are detailed in Section 2.2. Section 2.3 mentions some modifications which may be made to Seysen's algorithm when the performance of the basic version breaks down. Finally, Section 2.4 discusses possible extensions to Seysen's algorithm.

2.1 Theoretical Analysis

We consider first the theoretical foundations of Seysen's basis reduction algorithm. There are a number of questions concerning the actual structure of the algorithm which immediately arise. For a given quadratic form A, how might the S-reduced and S_2-reduced forms derived from A differ? Is it even sufficient to consider only $T_{i,j}^{\lambda}$ transformation matrices, or are there lattices for which it is impossible to find the S-reduced form using only $T_{i,j}^{\lambda}$ transformations? How do we choose the order in which to apply the $T_{i,j}^{\lambda}$ transformations, or equivalently how do we choose pairs of basis vectors for row moves? Is the Seysen measure function $S(A) = \sum_{i=1}^{n} a_{i,i}a_{i,i}^{*}$ a "good" way to rank different bases of a lattice? Finally, given that $S(A)$ is an acceptable measure function, is our choice of $\lambda = \lceil \frac{1}{2}(\frac{a^*_{i,j}}{a^*_{j,j}} - \frac{a_{i,j}}{a_{i,i}}) \rfloor$, given i and j, optimal? This section considers theoretical justifications for all of these questions. Section 2.2 below considers these questions from an empirical point of view.

2.1.1 Sufficiency of $T^{\lambda}_{i,j}$ Matrices

As defined above, a basis B is S-reduced if and only if its associated quadratic form A satisfies:

$$S(A) \leq S(T^t\, A\, T), \quad \text{for } T \in SL_n(\mathbb{Z}). \tag{2.4}$$

Thus, in order to Seysen-reduce a given lattice L which we know has basis B, we need to find a transformation matrix $T \in SL_n(\mathbb{Z})$ such that for all other $T' \in SL_n(\mathbb{Z})$ we have $S(T^t\, A\, T) \leq S((T')^t\, A\, T')$. As $SL_n(\mathbb{Z})$ is the set of all $n \times n$ matrices of unit determinant, we suspect that it is computationally difficult to find the desired matrix T directly. However, there are ways to avoid having to compute matrix T directly. Specifically, we can restrict our attention to a set of generating matrices for $SL_n(\mathbb{Z})$, as we show below.

Initially, let us assume that $n = 2$ and that A is the quadratic form associated with a lattice basis we wish to reduce. $SL_2(\mathbb{Z})$ thus contains all 2×2 matrices $[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}]$ with $ad - bc = 1$. Now, it is known [2, 37] that the set G_2 is a set of *generating matrices* for $SL_2(\mathbb{Z})$, where:

$$G_2 = \left\{ \begin{bmatrix} 1 & \lambda \\ 0 & 1 \end{bmatrix} : \lambda \in \mathbb{Z} \right\} \bigcup \left\{ \begin{bmatrix} 1 & 0 \\ \lambda & 1 \end{bmatrix} : \lambda \in \mathbb{Z} \right\}$$

That is, if $T \in G_2$, then there exists a sequence of matrices $T_1, T_2, \ldots, T_k$ such that

$$T = T_1\, T_2\, \cdots T_k, \; T_i \in G_2 \quad \text{for } 1 \leq i \leq k.$$

(Actually, the set $\{[\begin{smallmatrix}1 & \lambda\\ 0 & 1\end{smallmatrix}], [\begin{smallmatrix}1 & 0\\ \lambda & 1\end{smallmatrix}] : \lambda \in \{-1, 0, 1\}\}$ is sufficient, since

$$\begin{bmatrix}1 & \pm 1\\ 0 & 1\end{bmatrix}^{\lambda} = \begin{bmatrix}1 & \pm\lambda\\ 0 & 1\end{bmatrix},$$

$$\begin{bmatrix}1 & 0\\ \pm 1 & 1\end{bmatrix}^{\lambda} = \begin{bmatrix}1 & 0\\ \pm\lambda & 1\end{bmatrix}.$$

Section 2.2.2 below discusses the performance of Seysen's algorithm when we restrict $\lambda = \pm 1$.)

Notice that the matrices $[\begin{smallmatrix}1 & \lambda\\ 0 & 1\end{smallmatrix}]$ and $[\begin{smallmatrix}1 & 0\\ \lambda & 1\end{smallmatrix}]$ describe all possible row moves which can be performed on a 2×2 matrix. As an example, note that the matrix $S_0 = [\begin{smallmatrix}0 & 1\\ -1 & 0\end{smallmatrix}]$ is generated by:

$$S_0 = \begin{bmatrix}1 & 0\\ -1 & 1\end{bmatrix}\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ -1 & 1\end{bmatrix}.$$

(S_0 corresponds to swapping the two rows in a matrix.) Thus, the set of matrices $T_{i,j}^{\lambda}$ for $n = 2, i \neq j$ is exactly a set of generating matrices for $SL_2(\mathbb{Z})$. Therefore, it is sufficient for $n = 2$ for Seysen's algorithm to consider only products of matrices of the form $T_{i,j}^{\lambda}$. The difficulty is in choosing the right matrices and the right order of operations.

Our analysis above assumed $n = 2$, but similar results are known where n is an arbitrary integer [30]. For fixed $n > 0$,

$$G_n = I_n \bigcup \left\{ \bigcup_{\substack{1 \le i,j \le n\\ i \neq j}} \left\{T_{i,j}^{1}, T_{i,j}^{-1}\right\} \right\}$$

is a set of generating matrices for $SL_n(\mathbb{Z})$. Thus, it would be sufficient for Seysen's algorithm to consider only $T_{i,j}^{1}$ and $T_{i,j}^{-1}$ transformation matrices if it could pick the proper triples $(i, j, \pm 1)$ at every step. In practice, Seysen's algorithm chooses triples (i, j, λ) where $\lambda \in \mathbb{Z}$, but the basic problem is still choosing the right triples in the right order. Choosing the correct (i, j) pairs to reduce and the value of λ for that pair are discussed below.

2.1.2 Choosing Vector Pairs to Reduce

Seysen's algorithm does not specify how to choose which pair of basis vectors $(\mathbf{b_i}, \mathbf{b_j})$ to reduce on each iteration of the algorithm. At every iteration, it is necessary to find an (i, j) pair for which there exists a transformation matrix $T_{i,j}^{\lambda}$, $\lambda \neq 0$, such that:

$$S(T_{j,i}^{\lambda}\, A\, T_{i,j}^{\lambda}) < S(A).$$

Therefore, given that initially there are likely to be many pairs of vectors which may be reduced, we must decide how to select the best pair.

Two options appear immediately as candidate vector selection methods: *lazy* selection and *greedy* selection. A lazy selection scheme simply chooses any available (i, j) pair in the easiest possible manner. For example, we can imagine two nested loops which generate (i, j) pairs and stop at the first pair for which $\lambda(i, j) \neq 0$, where

$$\lambda(i, j) = \left\lceil \frac{1}{2} \left(\frac{a_{i,j}^*}{a_{j,j}^*} - \frac{a_{i,j}}{a_{i,i}} \right) \right\rfloor .$$

Once such a pair is found, a $T_{i,j}^{\lambda(i,j)}$ transformation can be performed on the lattice basis. Then the algorithm could search for another (i, j) pair, perhaps continuing the search at the first pair lexicographically after (i, j).

The second possible candidate selection method is a greedy approach. Here we calculate $\Delta(i, j, \lambda)$ for each possible pair (i, j), where $\Delta(i, j, \lambda)$ is defined:

$$\Delta(i, j, \lambda) = S(T_{j,i}^{\lambda}\, A\, T_{i,j}^{\lambda}) - S(A).$$

Thus, any transformation matrix $T_{i,j}^{\lambda}$ will have $\Delta(i, j, \lambda) < 0$. The algorithm then uses the pair of vectors $(\mathbf{b_i}, \mathbf{b_j})$ which minimizes $\Delta(i, j, \lambda)$ in the next row move.

One immediate disadvantage to a greedy approach is that it requires more extensive computations than the lazy selection method. This is true of any set of selection criteria which attempts to choose vector pairs to reduce in some fashion which performs better than random selection. If the two selection methods yield reduced bases of comparable Seysen measure, the added cost of an "intelligent" method may be greater than the time saved by reducing the number of row operations. However, if one method should yield lattices with lower Seysen measure, the extra costs may be justified.

We should point out that there is a distinction between choosing a pair of vectors to reduce and actually performing the reduction. Choosing a pair of vectors to reduce because they have the greatest potential to reduce the Seysen measure does not necessarily imply that we should perform the entire reduction and use the largest possible value of λ. It may be wise to perform only a fraction of the possible row moves and reevaluate other possible pairs of vectors. We run the risk, if we are too greedy, of getting stuck too soon in a local minimum.

There are reasons both to favor and to suspect the value of intelligent vector pair selection methods. One of the advantages that Seysen's method has over the LLL family of basis reduction algorithms is that it looks at all the vector pairs simultaneously. The LLL algorithm works in a fashion similar to a bubble sort and LLL only considers row operations involving "adjacent" basis vectors (i.e. $\mathbf{b_i}$ and $\mathbf{b_{i-1}}$ for $2 \leq i \leq n$). The cost of intelligent selection methods in terms of additional operations is certainly a disadvantage, but only if the cost is a significant fraction of the total running time. Section 2.2.1 below discusses these issues and presents empirical evidence of the cost and performance of a number of selection schemes for Seysen's algorithm. From our experiments, the greedy selection scheme performs better than the lazy scheme, and the additional computation required to implement greedy selection is small.

2.1.3 The $S(A)$ Function

Equation 2.2 above introduced the Seysen measure function $S(A) = \sum_{i=1}^{n} ||\mathbf{b_i}||^2||\mathbf{b_i^*}||^2$; the entire operation of Seysen's lattice basis reduction algorithm centers on this quantization of relative reduction of two bases of lattice L. It is natural to question whether Seysen measures are indeed a reasonable means of ranking different lattice bases. We mention here some of the theoretical evidence which suggests that ranking lattice bases by their Seysen measure is appropriate.

The use of the quantity $||\mathbf{b_i}||||\mathbf{b_i^*}||$ derives from elementary n-dimensional geometry. Recall the definition of the dual lattice $B^* = (\mathbf{b_1^*}, \ldots, \mathbf{b_n^*})$ of lattice B:

$$(\mathbf{b_i}, \mathbf{b_j^*}) = \delta_{i,j}, \quad \text{for } 1 \leq i, j \leq n,$$

where $\delta_{i,j}$ is the Dirac delta function ($\delta_{i,j} = 1$ if $i = j$, $\delta_{i,j} = 0$ otherwise). Now, fix i and notice that

$$\begin{aligned}
(\mathbf{b_i}, \mathbf{b_i^*}) &= 1, \\
||\mathbf{b_i}||||\mathbf{b_i^*}|| \cos(\alpha) &= 1, \\
||\mathbf{b_i}||||\mathbf{b_i^*}|| &= \frac{1}{\cos(\alpha)},
\end{aligned} \tag{2.5}$$

where α is the angle between $\mathbf{b_i}$ and $\mathbf{b_i^*}$. (Note that $-\frac{\pi}{2} \leq \alpha \leq \frac{\pi}{2}$ beacuse of the way in which $\mathbf{b_i^*}$ is defined.)

Let S_i denote the $(n-1)$-dimensional hyperplane spanned by the basis vectors $\mathbf{b_1}, \ldots, \mathbf{b_{i-1}}, \mathbf{b_{i+1}}, \ldots, \mathbf{b_n}$. Notice that $\mathbf{b_i^*}$ is perpendicular to S_i by definition. Thus, given that α is the angle between $\mathbf{b_i}$ and $\mathbf{b_i^*}$, the angle between $\mathbf{b_i}$ and S_i is $\pi - \alpha$. Thus, if $\beta = \pi - \alpha$, Equation 2.5 becomes

$$||\mathbf{b_i}||||\mathbf{b_i^*}|| = \frac{1}{\sin(\beta)}.$$

If basis vector $\mathbf{b_i}$ is relatively dependent of the other vectors $\mathbf{b_j}, 1 \leq j \leq n, j \neq i$, then the angle between $\mathbf{b_i}$ and the hyperplane S_i will be relatively small, and thus $\frac{1}{\sin(\beta)}$ will be large. Conversely, if $\mathbf{b_i}$ is relatively independent of the other basis vectors, β will be close to $\frac{\pi}{2}$ radians, and the product $||\mathbf{b_i}||||\mathbf{b_i^*}||$ will be close to one[1].

These geometric arguments lead directly to a measure which is a function of the products $||\mathbf{b_i}||||\mathbf{b_i^*}||$ where $1 \leq i \leq n$. Since $|\beta_i| \leq \frac{\pi}{2}$, we could choose the function $S_1(A) = \sum ||\mathbf{b_i}||||\mathbf{b_i^*}|| = \sum \frac{1}{\sin(\beta_i)}$ as the measure function. Unfortunately,

[1] Note that because of the duality between B and B^*, we could also have considered β to be the angle between $\mathbf{b_i^*}$ and the hyperplane spanned by $\mathbf{b_1^*}, \ldots, \mathbf{b_{i-1}^*}, \mathbf{b_{i+1}^*}, \ldots \mathbf{b_n^*}$.

as Section 2.4.4 points out below, there is no simple formula for finding the optimal value of λ for a row move involving the basis vectors $\mathbf{b_i}$ and $\mathbf{b_j}$. Seysen is able to avoid these computational difficulties by using

$$\begin{aligned} S(A) &= \sum_{i=1}^{n} ||\mathbf{b_i}||^2 ||\mathbf{b_i^*}||^2, \\ &= \sum_{i=1}^{n} \frac{1}{\sin^2(\beta_i)}, \end{aligned}$$

as the measure function, which does yield a simple formula for λ. Since $\sin(\beta_i) \in [0,1]$, the squared terms in the $S(A)$ function are guaranteed to be larger on a term-by-term basis than the corresponding terms in the $S_1(A)$ sum. Thus, if lattice basis B_1 has smaller measure than basis B_2 using the S_1 measure function, B_1 will also have smaller measure than B_2 when compared using the Seysen measure S.

An additional advantage to using a function of the $||\mathbf{b_i}|| ||\mathbf{b_i^*}||$ product terms is that bounds exists on the size of the individual terms. In [17] Håstad and Lagarias show that the following bound applies for some primal basis B and dual basis B^* of a given lattice:

$$\max_{1 \leq i \leq n} \{||\mathbf{b_i}||, ||\mathbf{b_i^*}||\} \leq \exp(O(n^{\frac{1}{3}})). \tag{2.6}$$

This bound immediately implies that there exists a basis of L with Seysen measure bounded by $\exp(O(n^{\frac{1}{3}}))$, since:

$$\begin{aligned} \max_{1 \leq i \leq n} \{||\mathbf{b_i}||, ||\mathbf{b_i^*}||\} &\leq \exp(O(n^{\frac{1}{3}})), \\ \max_{1 \leq i \leq n} ||\mathbf{b_i}|| ||\mathbf{b_i^*}|| &\leq \exp(O(n^{\frac{1}{3}})) + \exp(O(n^{\frac{1}{3}})) = \exp(O(n^{\frac{1}{3}})), \\ \sum_{i=1}^{n} ||\mathbf{b_i}|| ||\mathbf{b_i^*}|| &\leq n \exp(O(n^{\frac{1}{3}})), \\ S(A) &\leq \exp(O(n^{\frac{1}{3}})). \end{aligned}$$

Seysen shows in [38] that the bound in Equation 2.6 may be improved to

$$\max_{1 \leq i \leq n} \{||\mathbf{b_i}||, ||\mathbf{b_i^*}||\} \leq \exp(O((\ln n)^2)), \tag{2.7}$$

which reduces the bound on $S(A)$ for an S-reduced lattice to:

$$\begin{aligned}\sum_{i=1}^{n} \|\mathbf{b_i}\|\|\mathbf{b_i^*}\| &\leq n \exp(O((\ln n)^2), \\ &\leq \exp(\ln n) \exp(O((\ln n)^2), \\ S(A) &\leq \exp(O((\ln n)^2).\end{aligned}$$

To date, this is the best known bound on the Seysen-measure of an S-reduced lattice basis. However, as is the case with the LLL algorithm, in some cases Seysen's algorithm produces S_2-reduced lattice bases which have measures much lower than the theoretical bound.

2.1.4 Choosing λ Values

We now consider the choice of λ values in Seysen's basis reduction algorithm. Assume that $S(A)$ is as in Equation 2.3 above, and that only two-vector row moves are considered (i.e. transformation matrices of the form $T_{i,j}^{\lambda}$ for integer values of i, j and λ). We first show that

$$\lambda = \frac{1}{2}\left(\frac{a_{i,j}^*}{a_{j,j}^*} - \frac{a_{i,j}}{a_{i,i}}\right), \tag{2.8}$$

yields the maximum possible reduction in $S(A)$ for fixed values of i and j where $\lambda \in \mathbb{R}$. Further, we show that if we require $\lambda \in \mathbb{Z}$, then

$$\lambda = \left\lceil \frac{1}{2}\left(\frac{a_{i,j}^*}{a_{j,j}^*} - \frac{a_{i,j}}{a_{i,i}}\right)\right\rfloor, \tag{2.9}$$

is indeed the value which yields the best possible reduction in the Seysen measure of the lattice basis.

Let i, j be fixed integers with $1 \leq i, j \leq n$; $\mathbf{b_i}, \mathbf{b_j}, \mathbf{b_i^*}$ and $\mathbf{b_j^*}$ are the basis vectors on which we will perform a row move. Without loss of generality, we assume that $\lambda\mathbf{b_i}$ will be added to $\mathbf{b_j}$ and $\lambda\mathbf{b_j^*}$ will be subtracted from $\mathbf{b_i^*}$. Define $\mathbf{b_j'}$ and $\mathbf{b_i^*}'$ to be the values of $\mathbf{b_j}$ and $\mathbf{b_i^*}$ after the row move is performed:

$$\mathbf{b_j'} = \mathbf{b_j} + \lambda \mathbf{b_i},$$
$$\mathbf{b_i^{*\prime}} = \mathbf{b_i^*} - \lambda \mathbf{b_j^*}.$$

Let A and A^* be the quadratic forms associated with the lattice and its dual before the row move occurs, and let A' and $A^{*\prime}$ be the associated quadratic forms after the row move. Then

$$A' = T_{j,i}^{\lambda}\, A\, T_{i,j}^{\lambda},$$
$$A^{*\prime} = T_{i,j}^{-\lambda}\, A^*\, T_{j,i}^{-\lambda}.$$

Now, given that $T_{i,j}^{\lambda}$ transition matrices have exactly one off-diagonal nonzero entry, it is easy to see that A' differs from A only in the values in the i^{th} row, the j^{th} row, the i^{th} column, and the j^{th} column. The same is also true for $A^{*\prime}$. Since the Seysen measure function $S(A)$ only depends upon the diagonal elements in A and A^*, we know that

$$S(A') - S(A) = \sum_{i=1}^{n} a_{i,i}' a_{i,i}^{*\,\prime} - \sum_{i=1}^{n} a_{i,i} a_{i,i}^*, \tag{2.10}$$

$$= a_{i,i}' a_{i,i}^{*\,\prime} + a_{j,j}' a_{j,j}^{*\,\prime} - a_{i,i} a_{i,i}^* - a_{j,j} a_{j,j}^*. \tag{2.11}$$

When A is right-multiplied by $T_{i,j}^{\lambda}$, the i^{th} column of A is added to the j^{th} column of A. When this matrix is subsequently left-multiplied by $T_{j,i}^{\lambda}$, the i^{th} row is added to the j^{th} row. Thus, after these two transformations, the value of $a_{i,i}$ is unchanged, but

$$a_{j,j}' = a_{j,j} + 2\lambda a_{i,j} + \lambda^2 a_{i,i}. \tag{2.12}$$

If we perform a similar analysis in the dual quadratic form A^*, we find that

$$a_{i,i}^{*\,\prime} = a_{i,i}^* - 2\lambda a_{i,j}^* + \lambda^2 a_{j,j}^*. \tag{2.13}$$

Using Equations 2.12 and 2.13, Equation 2.11 becomes:

$$\begin{aligned} S(A') - S(A) &= a_{i,i}\left(a^*_{i,i} - 2\lambda a^*_{i,j} + \lambda^2 a^*_{j,j}\right) + a^*_{j,j}\left(a_{j,j} + 2\lambda a_{i,j} + \lambda^2 a_{i,i}\right) \\ &\quad - a_{i,i}a^*_{i,i} - a_{j,j}a^*_{j,j}, \\ &= 2\lambda^2 a_{i,i}a^*_{j,j} + 2\lambda a_{i,j}a^*_{j,j} - 2\lambda a_{i,i}a^*_{i,j}. \end{aligned}$$

Differentiating with respect to λ and setting the result equal to 0, we find that:

$$\begin{aligned} \frac{\partial}{\partial\lambda}\left(S(A') - S(A)\right) = 4\lambda a_{i,i}a^*_{j,j} + 2a_{i,j}a^*_{j,j} - 2a_{i,i}a^*_{i,j} &= 0, \\ 4\lambda a_{i,i}a^*_{j,j} &= 2a_{i,i}a^*_{i,j} - 2a_{i,j}a^*_{j,j}, \\ \lambda &= \frac{a_{i,i}a^*_{i,j} - a_{i,j}a^*_{j,j}}{2a_{i,i}a^*_{j,j}}, \\ &= \frac{1}{2}\left(\frac{a^*_{i,j}}{a^*_{j,j}} - \frac{a_{i,j}}{a_{i,i}}\right). \end{aligned}$$

Thus, if λ could take on any real value, for fixed i and j the minimum value of $S(A')$ is obtained with $\lambda = \frac{1}{2}\left(\frac{a^*_{i,j}}{a^*_{j,j}} - \frac{a_{i,j}}{a_{i,i}}\right)$.

We have shown that the minimum value of $S(A)$ with $\lambda \in \mathbb{R}$ is obtained when λ satisfies Equation 2.8. Our goal now is to show that if λ is restricted to integer values, Equation 2.9 yields that value of λ for which $S(A)$ is minimized. Let

$$\begin{aligned} \lambda &= \lambda_z + \lambda_r, \quad \text{where } \lambda_z \in \mathbb{Z}, 0 \le \lambda_r < 1, \\ \Delta(i,j,\lambda) &= S(T^{\lambda}_{j,i}\, A\, T^{\lambda}_{i,j}) - S(A). \end{aligned}$$

We know that for fixed i, j, $\lambda = \frac{1}{2}\left(\frac{a^*_{i,j}}{a^*_{j,j}} - \frac{a_{i,j}}{a_{i,i}}\right)$ minimizes the value of Δ. Furthermore, as Δ is a quadratic function of λ, at least one of λ_z and $\lambda_z + 1$ must minimize Δ for fixed, integer values of λ.

Consider $\Delta(i,j,\lambda_z)$ and $\Delta(i,j,\lambda_z+1)$:

$$\begin{aligned} \Delta(i,j,\lambda_z) &= 2\lambda_z^2 a_{i,i}a^*_{j,j} + 2\lambda_z a_{i,j}a^*_{j,j} - 2\lambda_z a_{i,i}a^*_{i,j}, \\ \Delta(i,j,\lambda_z+1) &= 2(\lambda_z+1)^2 a_{i,i}a^*_{j,j} + 2(\lambda_z+1)a_{i,j}a^*_{j,j} - 2(\lambda_z+1)a_{i,i}a^*_{i,j}, \\ &= 2\lambda_z^2 a_{i,i}a^*_{j,j} + 2\lambda_z a_{i,j}a^*_{j,j} - 2\lambda_z a_{i,i}a^*_{i,j} \\ &\quad + 4\lambda_z a_{i,i}a^*_{j,j} + 2a_{i,i}a^*_{j,j} + 2a_{i,j}a^*_{j,j} - 2a_{i,i}a^*_{i,j}, \end{aligned}$$

Thus,

$$\Delta(i,j,\lambda_z+1)-\Delta(i,j,\lambda_z)=4\lambda_z a_{i,i}a^*_{j,j}+2a_{i,i}a^*_{j,j}+2a_{i,j}a^*_{j,j}-2a_{i,i}a^*_{i,j}. \quad (2.14)$$

As $S(A)$ is a non-negative valued function which we want to minimize, we are interested in large, negative Δ values (i.e. $|\Delta|$ should be large, $\Delta < 0$). Thus, if Equation 2.14 is > 0, we should choose $\lambda = \lambda_z$; similarly, if Equation 2.14 is < 0, set $\lambda = \lambda_z + 1$.

When is Equation 2.14 greater than zero?

$$\begin{aligned} \Delta(i,j,\lambda_z+1)-\Delta(i,j,\lambda_z) &> 0, \\ \Longrightarrow 4\lambda_z a_{i,i}a^*_{j,j}+2a_{i,i}a^*_{j,j}+2a_{i,j}a^*_{j,j}-2a_{i,i}a^*_{i,j} &> 0, \\ 4\lambda_z a_{i,i}a^*_{j,j} &> 2a_{i,i}a^*_{i,j}-2a_{i,i}a^*_{j,j}-2a_{i,j}a^*_{j,j}, \\ \lambda_z &> \frac{1}{2}\left(\frac{a^*_{i,j}}{a^*_{j,j}}-\frac{a_{i,j}}{a_{i,i}}-1\right), \\ \lambda_z &> \lambda-\tfrac{1}{2}, \\ \lambda_z &> \lambda_z+\lambda_r-\tfrac{1}{2}, \\ \Longrightarrow \quad \lambda_r &< \tfrac{1}{2}. \end{aligned}$$

Thus, if $\lambda_r < \frac{1}{2}$, then Equation 2.14 is positive, and we should choose $\lambda' = \lambda_z$. If $\lambda > \frac{1}{2}$, Equation 2.14 is negative, and we should set $\lambda' = \lambda_z + 1$. Thus,

$$\begin{aligned} \lambda' &= \lceil \lambda_z + \lambda_r \rfloor, \\ &= \left\lceil \frac{1}{2}\left(\frac{a^*_{i,j}}{a^*_{j,j}}-\frac{a_{i,j}}{a_{i,i}}\right)\right\rfloor, \end{aligned}$$

which proves Equation 2.9.

2.2 Empirical Analysis

In the previous section we attempted to provide some theoretical justification for Seysen's basis reduction algorithm. The basic analysis suggests that Seysen's technique is viable, but as yet there are no significant bounds on the running time of the algorithm. In this section we detail numerical experiments which were performed using Seysen's algorithm. These experiments yield greater insight into the classes of lattices best suited for reduction by Seysen's algorithm, as well as an indication of the effectiveness of Seysen's technique.

Before we begin detailing the empirical results it is appropriate to detail our test conditions. All implementations of Seysen's basis reduction algorithm and the LLL algorithm were written in FORTRAN. Multiprecision floating point arithmetic operations were performed by a package of routines written by David Bailey at the NASA Ames Research Center [4]. Tests were run on Silicon Graphics, Inc., IRIS-4D workstations; the IRIS uses the MIPS R3000 chip set as its main processor.

The experiments described below explore many of the same aspects of Seysen's algorithm discussed in the previous section. Section 2.2.1 compares lazy and greedy schemes for choosing the row move to perform on each iteration of the algorithm. The effects of restricting λ choices are discussed briefly in Section 2.2.2. Sections 2.2.3 and 2.2.4 compare the performance of Seysen's algorithm with the LLL lattice on two classes of lattice bases.

2.2.1 Lazy vs. Greedy Selection Methods

In Section 2.1.2 above we outlined the differences between lazy and greedy methods for selecting a row move to perform on each iteration of the Seysen `while` loop. In this section we describe the results of test lattice reductions on small-dimension lattices using both the lazy and greedy approaches.

All experimental tests were performed on random integer lattices of determinant one. Integer lattice bases were generated as follows. Let B be an $n \times n$ integer matrix where the i^{th} column of B corresponds to basis vector $\mathbf{b_i}$. The elements of matrix B

Table 2.1: Comparison of Lazy and Greedy Selection Methods

n	Avg. # Steps (Lazy)	Avg. # Steps (Greedy)	Ratio (Lazy/Greedy)
20	2079.90	758.50	2.74
25	4096.40	1624.25	2.52
30	7444.80	3279.45	2.27
35	8787.35	3094.25	2.84

are:

$$B = [b_{i,j}]_{1 \leq i,j \leq n} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i > j, \\ \left\lceil \text{rand}(x) - \frac{1}{2}x \right\rfloor & \text{if } i < j. \end{cases}$$

The function rand(x) generates a random number chosen uniformly from the interval $[0, x]$; in these experiments $x = 4$. Notice that $\det(B) = 1$ since B is upper-triangular and all diagonal entries $b_{i,i} = 1$.

To generate a random, dense lattice which is not upper-triangular yet still has determinant equal to 1, we perform random row moves on matrix B to generate matrix B'. We choose n^2 pairs of integers (i, j) with $1 \leq i, j \leq n$ and $i \neq j$. For each such pair, λ is chosen to be $+1$ or -1 with equal probability. Then, the current $\mathbf{b}_i$ is scaled by λ and added to $\mathbf{b}_j$. (That is, we set $B = B\ T_{i,j}^{\lambda}$.) The result of these n^2 random row moves is matrix B', which is a basis for our test lattice L. Since $T_{i,j}^{\lambda}$ transformations preserve the determinant of the lattice, we know that $\det(L) = 1$. We may thus measure the performance of Seysen's algorithm on lattice L by how close reduced lattice L' is to I_n.

Table 2.1 summarizes the results of tests comparing the performance of lazy and greedy selection methods. Twenty test lattices were generated for each dimension $n \in \{20, 25, 30, 35\}$. In all cases where $n \leq 30$, both lazy and greedy algorithms were able to completely reduce all test lattices to I_n. The table shows the average number of row moves required by the lazy and greedy methods to reduce a lattice to I_n. On average, the lazy selection scheme required over twice as many row reductions as the greedy scheme did to reduce a given test lattice.

At $n = 35$ the lazy algorithm was able to reduce to I_n only two of the twenty attempted lattices; the remaining problems all encountered local minima during the reduction, thus halting Seysen's algorithm. The greedy implementation was unable to completely reduce any of the $n = 35$ test lattices. The two versions of the algorithm performed about equally well if we look at the Seysen measure of the reduced $n = 35$ test lattices.

These experimental results tell us two things concerning the relative merits of lazy and greedy selection schemes. First, when both lazy and greedy methods are likely to produce lattice bases with similar Seysen measures, the greedy selection methods will save at least a factor of two in the number of reduction steps. Second, based on the $n = 35$ data, using greedy instead of lazy does not appear to significantly reduce the performance of the algorithm as a whole. For our test lattices neither method performed significantly better than the other in terms of the Seysen measure of the S_2-reduced lattice bases.

One might argue that it is not reasonable to compare only the number of reduction steps required to reduce lattices using greedy and lazy selection methods, since that measure fails to take into account the cost of selecting the two basis vectors to reduce. A naive implementation of the greedy algorithm might require $O(n^2)$ time, as there are $\frac{1}{2}n(n-1)$ possible pairs of basis vectors $(\mathbf{b_i}, \mathbf{b_j}), i \neq j$ which must be considered. However, it turns out that, after an initial $O(n^2)$ precomputation phase, only $O(n)$ time is required to greedily select the next row move. Assume that we have computed $\Delta(i, j, \lambda(i, j))$ values for all pairs of integer $(i, j), 1 \leq i, j \leq n$. Now, after a specific row move involving basis vectors i' and j' is performed, the only previously computed values of Δ which need to be updated are those for which $i = i', i = j', j = i'$ or

$j = j'$. (If you consider Δ to be an array of values, the $(i')^{\text{th}}$ and $(j')^{\text{th}}$ rows and columns of Δ are all that need to be recomputed.) Thus, this recomputation can be performed in $O(n)$ time.

Storing Δ values can reduce the cost of a greedy selection method from $O(n^2)$ to $O(n)$. However, even $O(n)$ cost would be prohibitive if the actual amount of computation required to select a pair of vectors was comparable to the cost of performing a row move. This is not the case; performing a row move requires $O(n)$ multiprecision math operations, whereas the stored Δ values need only be stored as single- or double-precision floating point numbers. (The values are generally different enough that 32 or 64 bits will provide more than enough precision.) Since multiprecision operations take significantly more time than double-precision operations, and since each row-move requires $O(n)$ operations, it seems reasonable to discount the added cost of performing the greedy selection as noise when compared to the cost of implementing the main portion of the algorithm.

Based on these experimental results, we chose to use a greedy selection strategy in all subsequent implementations of Seysen's basis reduction algorithm. For lattices where we know that Seysen's algorithm will be able to perform a significant amount of basis reduction, such as the sparse lattices associated with subset sum problems (see Chapter 3), the greedy selection method and its expected reduced execution time are preferred.

2.2.2 Choosing λ Values

As shown in the previous section, the selection method for choosing row moves in Seysen's algorithm can affect both the algorithm's performance and running time. In our comparison of lazy and greedy selection methods above, however, we implicitly sidestepped another such issue: the method by which values of λ are chosen for a specific row move. Section 2.1.4 showed that for specified lattice basis vectors $(\mathbf{b_i}, \mathbf{b_j})$, the value of λ which minimizes $\Delta(i,j,\lambda)$ is:

$$\lambda = \left\lceil \frac{1}{2}\left(\frac{a^*_{i,j}}{a^*_{j,j}} - \frac{a_{i,j}}{a_{i,i}}\right)\right\rfloor . \tag{2.15}$$

However, recall from Section 2.1.1 that any transformation matrix $T^{\lambda}_{i,j}$ may be represented as the product of λ $T^{\pm 1}_{i,j}$ matrices (+1 if $\lambda > 0$, -1 otherwise). Thus, when a $T^{\lambda}_{i,j}$ transformation is applied to lattice basis B, we are implicitly performing λ row moves at once. It may be the case that for some lattices, a finer degree of control is required; that is, a greedy algorithm might perform even better if it was restricted to performing only $T^{1}_{i,j}$ and $T^{-1}_{i,j}$ transformations. That way the algorithm would have the finest possible degree of control over row operations.

It is also important to note that a greedy selection mechanism uses λ values in two distinct places. First, in order to select a pair of basis vectors to reduce, the greedy approach calculates $\Delta(i,j,\lambda(i,j))$ for all possible values of $i,j,i \neq j$ ($\lambda(i,j)$ is the function in Equation 2.15 above). Once a pair of vectors has been chosen, a $T^{\lambda}_{i,j}$ transformation is applied. In the first case, λ is used as part of the scoring mechanism in order to choose a set of basis vectors to reduce. In the second case λ plays a different role, the number of times to add vector $\mathbf{b_i}$ to $\mathbf{b_j}$. Because λ values have these two distinct functions, it is important that we distinguish between those roles when testing methods of choosing λ values.

We consider in this section three versions of Seysen's basis reduction algorithm and their performance on a set of randomly generated integer lattices[2]. All three versions use a greedy selection scheme to choose the next row move to perform. They differ only in the set of allowed values of λ in the scoring and transformation phases. These version are:

1. $(\mathbb{Z},\mathbb{Z})$: λ may take on any integer value when choosing a set of basis vectors for the next row move, and any $T^{\lambda}_{i,j}$ may be performed on those vectors.

[2] In fact, we use the same set of integer lattices used in the previous section for comparing the lazy and greedy selection mechanism.

2. $(\mathbb{Z}, \pm 1)$: λ may take on any integer value when choosing a set of basis vectors for the next row move. However, only $T^{1}_{i,j}$ and $T^{-1}_{i,j}$ actual transformations are allows. (If $\lambda > 0$ we add $\mathbf{b_i}$ to $\mathbf{b_j}$. If $\lambda < 0$ we subtract $\mathbf{b_i}$ from $\mathbf{b_j}$.)

3. $(\pm 1, \pm 1)$: λ may only be ± 1 when choosing the next row move, and only $T^{\pm 1}_{i,j}$ transformations may be performed on the basis.

The $(\mathbb{Z}, \mathbb{Z})$ version is identical to our "greedy" implementation of the previous section; it will serve as our control. The $(\mathbb{Z}, \pm 1)$ version of Seysen's algorithm is designed to greedily select the best possible row move based on unlimited λ values, but to perform the least possible number of changes to the lattice basis before recomputing what to do next. The $(\pm 1, \pm 1)$ version also restricts lattice basis changes to the minimum amount possible on each step, but this version selects a row move based only on what it can do immediately to reduce the $S(A)$ measure, not on any "future potential."

Table 2.2 compares the performance of the $(\mathbb{Z}, \mathbb{Z})$, $(\mathbb{Z}, \pm 1)$ and $(\pm 1, \pm 1)$ versions of Seysen's basis reduction algorithm. For each value of n, twenty test lattices of dimension n were generated and Seysen-reduced by each of the three methods. The table lists aggregate information for each value of n (all numbers shown are the geometric mean of the experimental values obtained for each of the twenty test lattices):

- n: The dimension of the test lattice in this group.

- L_{10}: $\sum_{i=1}^{n} \log_{10} \|\mathbf{b_i}\|$ before reduction.

- $S(A)$: The Seysen measures before reduction.

- L^*_{10}: $\sum_{i=1}^{n} \log_{10} \|\mathbf{b^*_i}\|$ before reduction.

- Method: The restructions placed of λ values.

- L'_{10}: $\sum_{i=1}^{n} \log_{10} \|\mathbf{b_i}\|$ after reduction.

- $S(A')$: The Seysen measure after reduction.

Table 2.2: Comparison of $(\mathbb{Z},\mathbb{Z})$, $(\mathbb{Z},\pm 1)$ and $(\pm 1,\pm 1)$ Options

n	L_{10}	$S(A)$	L^*_{10}	Method	L'_{10}	$S(A')$	$L^{*\prime}_{10}$	# Steps
20	82.4	$4.7 \cdot 10^{11}$	118.2	$(\mathbb{Z},\mathbb{Z})$	0.	20.	0.	757.4
				$(\mathbb{Z},\pm 1)$	0.	20.	0.	767.0
				$(\pm 1,\pm 1)$	0.	20.	0.	787.3
25	132.0	$5.7 \cdot 10^{14}$	192.9	$(\mathbb{Z},\mathbb{Z})$	0.	25.	0.	1622.1
				$(\mathbb{Z},\pm 1)$	0.	25.	0.	1603.4
				$(\pm 1,\pm 1)$	0.	25.	0.	1610.3
30	195.8	$9.3 \cdot 10^{17}$	287.2	$(\mathbb{Z},\mathbb{Z})$	0.	30.	0.	3275.3
				$(\mathbb{Z},\pm 1)$	0.	30.	0.	3176.1
				$(\pm 1,\pm 1)$	0.	30.	0.	3316.1
35	269.2	$6.2 \cdot 10^{20}$	390.0	$(\mathbb{Z},\mathbb{Z})$	110.4	$1.0 \cdot 10^{8}$	112.6	3037.0
				$(\mathbb{Z},\pm 1)$	99.0	$4.0 \cdot 10^{7}$	102.2	3022.0
				$(\pm 1,\pm 1)$	112.3	$1.3 \cdot 10^{8}$	114.6	2920.8
40	343.8	$1.8 \cdot 10^{23}$	507.0	$(\mathbb{Z},\mathbb{Z})$	216.9	$5.2 \cdot 10^{12}$	227.1	2240.2
				$(\mathbb{Z},\pm 1)$	206.5	$1.6 \cdot 10^{12}$	217.0	2399.4
				$(\pm 1,\pm 1)$	205.0	$1.5 \cdot 10^{12}$	217.3	2527.2
45	434.0	$2.0 \cdot 10^{26}$	656.7	$(\mathbb{Z},\mathbb{Z})$	304.6	$4.2 \cdot 10^{15}$	323.5	2369.9
				$(\mathbb{Z},\pm 1)$	290.6	$1.2 \cdot 10^{15}$	312.3	2569.8
				$(\pm 1,\pm 1)$	296.5	$1.9 \cdot 10^{15}$	315.8	2739.2
50	541.2	$4.5 \cdot 10^{29}$	833.5	$(\mathbb{Z},\mathbb{Z})$	402.8	$2.3 \cdot 10^{18}$	429.2	2698.1
				$(\mathbb{Z},\pm 1)$	386.1	$6.3 \cdot 10^{17}$	418.4	3276.4
				$(\pm 1,\pm 1)$	393.8	$1.1 \cdot 10^{18}$	422.4	3491.1

- $L^{*'}_{10}$: $\sum_{i=1}^{n} \log_{10} \|\mathbf{b}_i^*\|$ after reduction.

- # Steps: The number of row moves performed during the reduction.

For $n \leq 30$, all three implementation were able to completely reduce all test lattices to I_n. The only difference in the performance of the three methods was in the number of reduction steps required to reduce a test lattice, and these differences were minor (no more than 5% variation among any of the values for a given dimension).

More differences among the methods appeared once $n \geq 35$ and Seysen's algorithm was no longer able to reduce any of the test lattices to I_n. For the majority of the test lattices, the $(\mathbb{Z}, \pm 1)$ appears to yield the most Seysen-reduced lattice basis, although it requires significantly more row moves to perform this reduction than the $(\mathbb{Z}, \mathbb{Z})$ method. This improvement is expected; after all, the only difference between the $(\mathbb{Z}, \mathbb{Z})$ and $(\mathbb{Z}, \pm 1)$ methods is that that latter looks more frequently at the lattice basis and takes smaller "steps" while performing a reduction. However, as the dimension of the lattice basis increased, the ratio of row moves required by $(\mathbb{Z}, \pm 1)$ to row moves required by $(\mathbb{Z}, \mathbb{Z})$ also increased. By the time $n = 50$, the $(\mathbb{Z}, \pm 1)$ method required approximately 30% more reduction steps to reduce test lattices than the $(\mathbb{Z}, \mathbb{Z})$ method did.

The performance of the $(\pm 1, \pm 1)$ method fell somewhere between that of $(\mathbb{Z}, \pm 1)$ and $(\mathbb{Z}, \mathbb{Z})$ for test lattices with $n \geq 45$. This is somewhat surprising in and of itself, since the capability of the $(\pm 1, \pm 1)$ method to consider future reduction is severely limited. This unexpected performance may be an artifact of our method of generating test lattices; we only performed n^2 row operations on the initial upper-triangular lattice basis and used only $T_{i,j}^{\pm 1}$ transitions to modify the lattice basis. This could account for the relatively good performance of the $(\pm 1, \pm 1)$ method, and also the differences between the $(\mathbb{Z}, \mathbb{Z})$ and $(\mathbb{Z}, \pm 1)$ methods.

The experiments summarized in Table 2.2 do not indicate that any one of the tested methods consistently performs better than the others. Without any clear indication that one method would yield significantly better results (especially as $n \to 100$), we were reluctant to use any method in Seysen's algorithm except the $(\mathbb{Z},\mathbb{Z})$ one implicitly defined by Seysen himself. For special types of lattices, it is probably worthwhile to compare these methods again. It may be that increased performance by the $(\mathbb{Z},\pm 1)$ method more than offsets the increase in the number of row operations. However, for our experiments in subsequent sections (and the subset sum lattices of Chapter 3) we continued to use the $(\mathbb{Z},\mathbb{Z})$ form of Seysen's algorithm.

2.2.3 Testing the B_θ Lattice

The previous sections detailed small tests which compared the relative performance of Seysen's algorithm when a few minor changes were made to its structure. In this section and the following one we investigate more fully the performance of the algorithm itself as a reduction technique for self-dual lattice bases and random integer lattices. Our next series of tests was suggested by J. C. Lagarias (personal communication) to test the self-dual reduction performance of the algorithm. Let $B_\theta, 0 < \theta < \frac{1}{2}$ be the lattice basis consisting of the following basis vectors:

$$\begin{aligned}
\mathbf{b_1} &= (1,0,0,0,\dots),\\
\mathbf{b_2} &= (\theta,1,0,0,\dots),\\
\mathbf{b_3} &= (-\theta,\theta,1,0,\dots),\\
\mathbf{b_4} &= (\theta,-\theta,\theta,1,\dots),\\
&\dots
\end{aligned}$$

If B_θ is represented as a matrix with $\mathbf{b_i}$ as the i^{th} column, then we have:

$$B_\theta = [b_{i,j}]_{1\le i,j\le n} = \begin{cases} 1 & \text{if } i = j,\\ 0 & \text{if } i > j,\\ (-1)^{j-i+1}\theta & \text{if } i < j. \end{cases}$$

Basis B_θ has the property that $\|\mathbf{b}_i^*\|$ grows exponentially with n, but rather slowly for small dimensions.

Tests were performed on B_θ lattices with $\theta = 0.4$ using Seysen's basis reduction algorithm for dimensions $5 \leq n \leq 105$. Based on the experimental results of Sections 2.2.1 and 2.2.2 above, we used a greedy selection method to choose which pairs of vectors to reduce on each iteration of the algorithm, and λ was allowed to be any integer value during all stages of the algorithm. The results of these tests are given in Table 2.3. For each test lattice, the following information is given:

- n: The dimension of the lattice.
- L_{10}: $\sum_{i=1}^{n} \log_{10} \|\mathbf{b}_i\|$ before reduction.
- $S(A)$: The Seysen measure of B_θ before reduction.
- L_{10}^*: $\sum_{i=1}^{n} \log_{10} \|\mathbf{b}_i^*\|$ before reduction.
- L_{10}': $\sum_{i=1}^{n} \log_{10} \|\mathbf{b}_i\|$ after reduction.
- $S(A')$: The Seysen measure of B_θ after reduction.
- $L_{10}^{*\prime}$: $\sum_{i=1}^{n} \log_{10} \|\mathbf{b}_i^*\|$ after reduction.
- # Steps: The number of row moves performed during the reduction.

The exponential growth of $\|\mathbf{b}_i^*\|$ may be easily seen by looking at the rate of growth of the L_{10}^* column in the tables. Remember that L_{10}^* is the sum of the base 10 logarithms of the lengths of the dual basis vectors. This sum grows at least linearly with respect to n; thus, the $\|\mathbf{b}_i^*\|$ grow exponentially in n.

For the B_θ lattice Seysen's basis reduction algorithm yields little improvement in the vector lengths $\|\mathbf{b}_i\|$. Indeed, for some values of n we have $L_{10} < L_{10}'$. This is not the case, though, for the dual lattice; Seysen's reduction algorithm greatly decreases the lengths of the vectors in the dual basis. When the algorithm completes,

Table 2.3: Performance of Seysen's Algorithm on B_θ for $\theta = 0.4, 5 \leq n \leq 105$

n	L_{10}	$S(A)$	L^*_{10}	L'_{10}	$S(A')$	$L^*_{10}{}'$	# Steps
5	0.30	7.36e+00	0.50	0.30	6.59e+00	0.30	3
10	1.10	3.65e+01	3.80	1.10	1.61e+01	1.00	12
15	2.30	1.88e+02	10.60	2.10	2.78e+01	1.90	29
20	3.70	1.00e+03	21.10	3.40	4.28e+01	3.10	45
25	5.30	5.37e+03	35.20	4.90	5.80e+01	4.20	70
30	7.10	2.89e+04	53.00	6.30	7.88e+01	6.20	135
35	9.10	1.55e+05	74.40	8.20	1.03e+02	8.00	177
40	11.20	8.35e+05	99.50	10.40	1.30e+02	9.90	240
45	13.40	4.49e+06	128.30	13.10	1.72e+02	12.80	363
50	15.70	2.42e+07	160.70	15.80	2.32e+02	16.90	509
55	18.20	1.30e+08	196.70	18.60	2.62e+02	18.30	698
60	20.70	6.99e+08	236.40	22.30	3.53e+02	22.50	772
65	23.30	3.76e+09	279.70	25.00	4.01e+02	25.70	1067
70	25.90	2.02e+10	326.80	26.40	4.40e+02	28.10	1467
75	28.70	1.09e+11	377.40	32.70	6.93e+02	36.60	1702
80	31.50	5.85e+11	431.70	31.90	6.00e+02	35.70	2123
85	34.40	3.15e+12	489.70	35.70	7.25e+02	40.40	2775
90	37.30	1.69e+13	551.30	39.70	8.50e+02	45.40	3345
95	40.30	9.10e+13	616.60	44.10	1.03e+03	51.00	4839
100	43.30	4.89e+14	685.60	49.00	1.20e+03	55.50	6363
105	46.40	2.63e+15	758.10	50.80	1.19e+03	56.70	8303

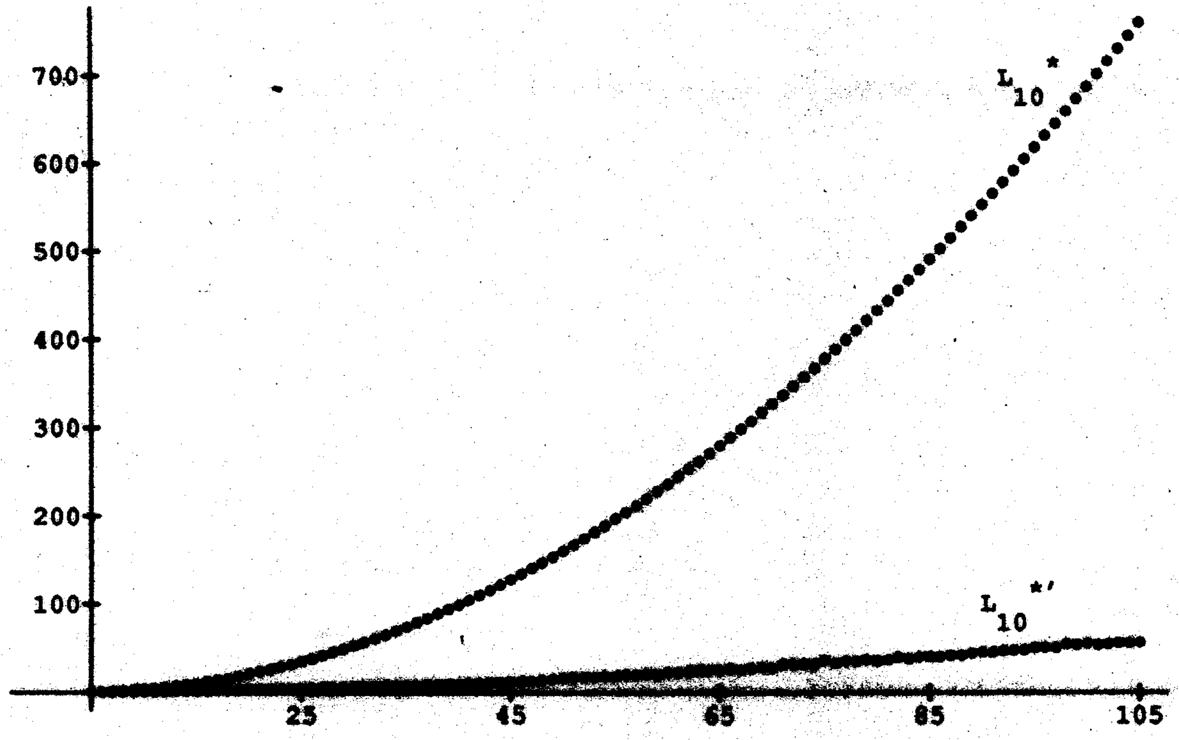

Figure 2-1: Performance of Seysen's Algorithm on $B_{0.4}$ Lattice: L_{10}^{*} and $L_{10}^{*}{}'$ vs. n

we have $L'_{10} \approx {L^*_{10}}'$ and the lengths of the vectors in the prime and dual lattice bases are comparable. Figure 2-1 shows graphically the improvement made by Seysen's algorithm to $\|\mathbf{b}^*_i\|$.

The results obtained from application of Seysen's algorithm to B_θ lattices are quite promising. The algorithm was able to significantly reduce the lengths of the dual basis vectors $\mathbf{b}^*_i$ without significantly increasing the lengths of the basis vectors for B_θ themselves. In fact, the resulting primal and dual bases have basis vector lengths which are comparable. Certainly this suggests that Seysen's algorithm is a viable technique for applications in which we wish to simultaneously reduce a lattice and its dual.

2.2.4 Testing Random Integer Lattices

The reductions of B_θ lattices tested application of Seysen's algorithm to a very narrow class of lattices. From a cryptographic point of view (see Chapter 3 below), and in many other cases, our goal is to reduce random lattices with integer basis vectors. In general, we do not know that our lattice conforms to some specific structure; we are give only the basis vectors themselves. Thus, it is appropriate to investigate the performance of Seysen's algorithm on randomly generated integer lattices.

When we considered in Section 2.2.1 the choice of a lazy or greedy selection method for Seysen's algorithm, we ran tests on random integral lattices L with $\det(L) = 1$ of small dimension ($n \leq 35$). We utilize the same technique for generating random lattices as used before, except now we consider lattices up to dimension 50. In addition to running Seysen's algorithm over these test cases, each lattice was also reduced using the LLL algorithm with $y \lesssim 1.0$.

Table 2.4 summarizes the results of these experiments. For each value of $n \equiv 0 \bmod 5$, $20 \leq n \leq 50$, twenty different test lattices were generated. The columns n, L_{10}, $S(A)$, L^*_{10}, L'_{10}, $S(A')$, and ${L^*_{10}}'$ identify the same quantities as in Table 2.2 above. The column labeled "# Steps (Seysen)" reports the average number of reduction steps (row moves) performed by Seysen's algorithm for each test lattice. The average number of row operations performed by the LLL algorithm is listed in the "# Steps (Lovasz)" column. (LLL reduction was not performed on lattices with $n \geq 45$ because of the excessive amount of computer time required to obtain results).

The LLL basis reduction algorithm was able to reduce all tested lattice bases to the n-dimensional cubic lattice basis (i.e. $B' = I_n$), which has Seysen measure zero. Seysen's algorithm performed similarly on all lattice bases with $n \leq 31$. (Values in parentheses in the L'_{10} column indicate the number of lattice bases which were Seysen-reduced to the n-dimensional cubic lattice.) For $32 \leq n \leq 34$ the Seysen algorithm was able to completely reduce only some of the attempted lattice bases; no lattice basis with $n \leq 35$ was ever reduced by the Seysen algorithm to I_n.

The degradation in the performance of Seysen's algorithm as n increases from 30 to 35 is quite surprising. Apparently, for these types of lattices, the probability of reaching a lattice basis which is a local minimum of the Seysen measure function increases substantially over that range. The LLL reduction algorithm does not exhibit any decrease in performance, except for an overall increase in the number of reduction steps required to convert the given lattice basis into the n-dimensional cubic lattice basis. Of course, the LLL algorithm does take significantly longer to run than the Seysen algorithm, but these tests suggest that Seysen's algorithm alone will not be sufficient to reduce lattice bases in higher dimensions. We may need to combine Seysen's algorithm with other lattice basis reduction techniques to efficiently reduce large lattice bases. Alternately, it may be possible to use some heuristic technique to reduce the probability of reaching a lattice basis which is a local minimum of $S(A)$, or to "kick" a Seysen-reduced basis out of a local minimum. The following section suggests a few possible methods which may be employed when Seysen's algorithm fails to S-reduce a lattice basis and stops at a local minimum of $S(A)$.

Table 2.4: Performance of Seysen's Algorithm on Random Integer Lattices

n	L_{10}	$S(A)$	L^*_{10}	L'_{10}	$S(A')$	$L^*_{10}{}'$	# Steps (Seysen)	# Steps (Lovasz)
20	82.37	$4.71 \cdot 10^{11}$	118.22	0. (20)	20.	0.	757.37	4424.0
25	131.95	$5.67 \cdot 10^{14}$	192.93	0. (20)	25.	0.	1622.11	12977.4
30	195.81	$9.28 \cdot 10^{17}$	287.24	0. (20)	30.	0.	3275.30	32718.5
31	205.86	$3.05 \cdot 10^{18}$	308.06	0. (20)	31.	0.	3690.51	38085.2
32	222.41	$1.04 \cdot 10^{19}$	323.66	0. (11)	1695.8	0.	3626.00	44770.0
33	237.86	$5.07 \cdot 10^{19}$	348.19	0. (5)	86999.0	0.	3408.27	53407.4
34	243.86	$4.48 \cdot 10^{19}$	358.86	0. (1)	$4.26 \cdot 10^{6}$	0.	3075.63	60299.7
35	269.25	$6.24 \cdot 10^{20}$	390.02	110.39	$1.00 \cdot 10^{8}$	112.57	3036.95	70047.2
36	287.16	$5.26 \cdot 10^{21}$	423.62	138.11	$2.48 \cdot 10^{9}$	141.13	2738.39	80836.3
37	291.86	$5.23 \cdot 10^{21}$	438.94	162.17	$3.20 \cdot 10^{10}$	167.33	2400.10	91491.5
38	310.35	$3.17 \cdot 10^{22}$	469.18	175.52	$1.07 \cdot 10^{11}$	182.50	2520.02	103624.0
39	329.32	$1.78 \cdot 10^{23}$	500.89	196.26	$7.45 \cdot 10^{11}$	203.87	2555.72	118338.0
40	343.84	$1.82 \cdot 10^{23}$	507.02	216.88	$5.21 \cdot 10^{12}$	227.10	2240.23	132901.0
45	434.01	$2.02 \cdot 10^{26}$	656.67	304.56	$4.20 \cdot 10^{15}$	323.53	2369.90	–
50	541.20	$4.53 \cdot 10^{29}$	833.54	402.80	$2.33 \cdot 10^{18}$	429.15	2698.11	–

2.3 When Seysen's Algorithm Fails

We have seen above that for random, dense lattices L with $\det(L) = 1$, Seysen's algorithm starts to break down for $30 \leq n \leq 35$. As n increases beyond 35, the number of local minima for the $S(A)$ function apparently increases, and thus the chance that Seysen's algorithm will reduce lattice L to I_n decreases. As the number of local minima increases, it is increasingly likely that in the process of reducing lattice L the $S(A)$ function (where A is the quadratic form associated with L) will encounter one of these local minima. As described above, Seysen's algorithm cannot tell whether it has reached a local or a global minimum. Thus, it stops as soon as all possible $T_{i,j}^{\lambda}$ transformations cause $S(A)$ to increase.

For many types of lattices, such as the sparse lattices generated by subset sum problems (see Chapter 3), Seysen's algorithm has performed sufficient work by the time it encounters a local minimum that it is acceptable for it to stop. However, for many lattice reduction problems Seysen's algorithm stops too soon. We would like the algorithm to be able to detect local minima and overcome them. If one considers the surface described by $S(A)$ values, local minima are "wells" or "depressions" in the surface which are large enough to contain all points reachable by performing one row move on the lattice. In this section we discuss possible techniques for "kicking" the reduced lattice out of these wells; methods of enhancing Seysen's algorithm so that it may consider other options when it encounters a local minimum.

There are many methods which could conceivably be applied to a lattice to move it out of a local minimum; we consider only some of these options. Section 2.3.1 considers an obvious possibility, which is to consider row moves involving 3 or 4 vectors at once (general n-moves are discussed in Section 2.4.1 below). In Section 2.3.2 we investigate simulated annealing and rapid quenching approaches to the problem. Finally, Section 2.3.3 discusses using Hadamard matrices to permute the entire lattice basis.

2.3.1 Row Moves Involving Three or Four Vectors

As initially described in [38], Seysen's basis reduction algorithm only considers row moves involving two basis vectors. That is, we only consider moves of the form:

$$\mathbf{b_j} \leftarrow \mathbf{b_j} + \lambda \mathbf{b_i}. \tag{2.16}$$

(These moves, of course, correspond to the set of transformation matrices $T^{\lambda}_{i,j}$.) However, there is no real theoretical reason to restrict ourselves to row moves of the form given in Equation 2.16. We consider here possibilities for row moves of the form

$$\mathbf{b_j} \leftarrow \mathbf{b_j} + \lambda \mathbf{b_{i_1}} + \kappa \mathbf{b_{i_2}},$$
$$\mathbf{b_j} \leftarrow \mathbf{b_j} + \lambda \mathbf{b_{i_1}} + \kappa \mathbf{b_{i_2}} + \mu \mathbf{b_{i_3}},$$

and their dual basis counterparts

$$\mathbf{b^*_j} \leftarrow \mathbf{b^*_j} + \lambda \mathbf{b^*_{i_1}} + \kappa \mathbf{b^*_{i_2}},$$
$$\mathbf{b^*_j} \leftarrow \mathbf{b^*_j} + \lambda \mathbf{b^*_{i_1}} + \kappa \mathbf{b^*_{i_2}} + \mu \mathbf{b^*_{i_3}}.$$

Before we begin, it should be noted that there are practical reasons for not considering row moves involving more than two vectors at any given time. First, if we are using a greedy selection method to choose the vectors upon which to operate, more work is required to choose the correct n-tuple. (If we use a lazy implementation this is less of a concern). Second, 2-moves[3] exhibit a symmetry between the prime and dual lattice which is lost when we consider n-moves with $n > 2$. When we perform a $T^{\lambda}_{i,j}$ transformation, $\mathbf{b_j} \leftarrow \mathbf{b_j} + \lambda \mathbf{b_i}$ *and* $\mathbf{b^*_i} \leftarrow \mathbf{b^*_i} - \lambda \mathbf{b^*_j}$. Thus we need not explicitly consider operations on the dual lattice, since every dual lattice transformation has an equivalent transformation in the prime lattice (with i, j swapped and λ multiplied by -1). For n-moves with $n > 2$, however, this duality is lost. The 3-move

$$\mathbf{b_j} \leftarrow \mathbf{b_j} + \lambda \mathbf{b_{i_1}} + \kappa \mathbf{b_{i_2}},$$

[3] We use "k-move" to designate a row operation in which multiples of $k-1$ vectors are added simultaneously to another vector.

for example, has the following effects in the dual basis:

$$\mathbf{b}^*_{\mathbf{i_1}} \leftarrow \mathbf{b}^*_{\mathbf{i_1}} - \lambda \mathbf{b}^*_{\mathbf{j}},$$
$$\mathbf{b}^*_{\mathbf{i_2}} \leftarrow \mathbf{b}^*_{\mathbf{i_2}} - \kappa \mathbf{b}^*_{\mathbf{j}}.$$

Thus, even if we use a lazy approach to choose which vectors to use in a row operation, there are many more candidate operations which must be considered since there is no longer any overlap between moves in the primal and dual lattices.

Calculating the best possible values of $\lambda, \kappa, \mu, \ldots$ for a given set of basis vectors is also more complicated as the number of vectors involved in the move increases. As an example, let us consider the computation required to choose λ and κ for the 3-move

$$\begin{aligned} \mathbf{b_j} &\leftarrow \mathbf{b_j} + \lambda \mathbf{b_{i_1}} + \kappa \mathbf{b_{i_2}}, \\ \mathbf{b}^*_{\mathbf{i_1}} &\leftarrow \mathbf{b}^*_{\mathbf{i_1}} - \lambda \mathbf{b}^*_{\mathbf{j}}, \\ \mathbf{b}^*_{\mathbf{i_2}} &\leftarrow \mathbf{b}^*_{\mathbf{i_2}} - \kappa \mathbf{b}^*_{\mathbf{j}}. \end{aligned}$$

We assume without loss of generality that $i_1 < i_2 < j$. Then we may represent this transformation by a $T^{\lambda,\kappa}_{i_1,i_2,j}$ transformation matrix, where:

$$T^{\lambda,\kappa}_{i_1,i_2,j} = I_n + \lambda U_{i_1,j} + \kappa U_{i_2,j}.$$

Similar to Section 2.1.4 above, let us define

$$\begin{aligned} \Delta(i_1, i_2, j, \lambda, \kappa) &= S((T^{\lambda,\kappa}_{i_1,i_2,j})^t \; A \; T^{\lambda,\kappa}_{i_1,i_2,j}) - S(A) \\ &= \left(-2a_{i_2,i_2}a^*_{i_2,j} + 2a_{i_2,j}a^*_{j,j}\right)\kappa + 2a_{i_2,i_2}a^*_{j,j}\kappa^2 \\ &\quad + \left(-2a_{i_1,i_1}a^*_{i_1,j} + 2a_{i_1,j}a^*_{j,j}\right)\lambda + 2a_{i_1,i_2}a^*_{j,j}\kappa\lambda + 2a_{i_1,i_1}a^*_{j,j}\lambda^2 \end{aligned}$$

Now, we compute the partial derivatives of $\Delta(i_1, i_2, j, \lambda, \kappa)$ with respect to κ and λ,

$$\begin{aligned} \frac{\partial}{\partial\kappa}\Delta(i_1, i_2, j, \lambda, \kappa) &= -2a_{i_2,i_2}a^*_{i_2,j} + 2a_{i_2,j}a^*_{j,j} + 4a_{i_2,i_2}a^*_{j,j}\kappa + 2a_{i_1,i_2}a^*_{j,j}\lambda, \\ \frac{\partial}{\partial\lambda}\Delta(i_1, i_2, j, \lambda, \kappa) &= -2a_{i_1,i_1}a^*_{i_1,j} + 2a_{i_1,j}a^*_{j,j} + 2a_{i_1,i_2}a^*_{j,j}\kappa + 4a_{i_1,i_1}a^*_{j,j}\lambda, \end{aligned}$$

set them equal to 0, and solve for κ and λ:

$$\frac{\partial\Delta}{\partial\kappa} = 0, \frac{\partial\Delta}{\partial\lambda} = 0 \Longrightarrow$$

$$\lambda = \frac{-4a_{i_2,i_2}\left(-2a_{i_1,i_1}a^*_{i_1,j} + 2a_{i_1,j}a^*_{j,j}\right)a^*_{j,j} - 2a_{i_1,i_2}a^*_{j,j}\left(-2a_{i_2,i_2}a^*_{i_2,j} + 2a_{i_2,j}a^*_{j,j}\right)}{4(a_{i_1,i_2})^2(a^*_{j,j})^2 - 16a_{i_1,i_1}a_{i_2,i_2}(a^*_{j,j})^2},$$

$$\kappa = \frac{2a_{i_1,i_2}\left(-2a_{i_1,i_1}a^*_{i_1,j} + 2a_{i_1,j}a^*_{j,j}\right)a^*_{j,j} - 4a_{i_1,i_1}a^*_{j,j}\left(-2a_{i_2,i_2}a^*_{i_2,j} + 2a_{i_2,j}a^*_{j,j}\right)}{4(a_{i_1,i_2})^2(a^*_{j,j})^2 - 16a_{i_1,i_1}a_{i_2,i_2}(a^*_{j,j})^2}.$$

If we wish to implement a version of Seysen's algorithm which allows 3-moves in general, we must calculate *six* sets of (λ, κ) values; one set for each of

$$\mathbf{b_j} \leftarrow \mathbf{b_j} + \lambda\mathbf{b_{i_1}} + \kappa\mathbf{b_{i_2}} \qquad \mathbf{b^*_j} \leftarrow \mathbf{b^*_j} + \lambda\mathbf{b^*_{i_1}} + \kappa\mathbf{b^*_{i_2}}, \tag{2.16}$$

$$\mathbf{b_{i_1}} \leftarrow \mathbf{b_{i_1}} + \lambda\mathbf{b_j} + \kappa\mathbf{b_{i_2}} \qquad \mathbf{b^*_{i_1}} \leftarrow \mathbf{b^*_{i_1}} + \lambda\mathbf{b^*_j} + \kappa\mathbf{b^*_{i_2}}, \tag{2.17}$$

$$\mathbf{b_{i_2}} \leftarrow \mathbf{b_{i_2}} + \lambda\mathbf{b_{i_1}} + \kappa\mathbf{b_j} \qquad \mathbf{b^*_{i_2}} \leftarrow \mathbf{b^*_{i_2}} + \lambda\mathbf{b^*_{i_1}} + \kappa\mathbf{b^*_j}. \tag{2.18}$$

Clearly including the six possible 3-moves and the eight possible 4-moves in Seysen's algorithm is computationally expensive. However, there are reasons for wishing to do so. When Seysen's algorithm reaches a local minimum of $S(A)$ for some lattice L it is reducing, it has reached a point where any single row move increases $S(A)$. By allowing the algorithm to look at 3-moves when it has run out of 2-moves to perform, we increase the number of configurations Seysen's algorithm must investigate before it gives up. It is quite possible that one or more configurations which are 3-move attainable but not 2-move attainable will have Seysen measure smaller than $S(A)$. These reduced lattices would then permit the algorithm to move out of the local minimum and continue its reduction steps.

We added routines to our implementation of Seysen's basis reductions algorithm to implement three- and four-vector row operations. In all cases one vector was designated the "target" vector and integer multiples of the other two or three vectors were added to the target. We did not notice any significant improvement in the performance of the algorithm on random integer lattices of determinant 1 when these additional moves were allowed to occur on any iteration of the algorithm. In some cases, the greedy selection scheme actually performed worse when allowed to use 3-moves and 4-moves; usually this decrease in performance occurred because the algorithm "jumped ahead" too quickly by using the 3-move and would have done better by using a sequence of 2-moves. Three- and four-vector operations were helpful, however, when a lattice reduction reached a local minimum. In many of these cases a few 3-moves (or 4-moves, if considered) existed which would carry the lattice out of the local minimum and allow the algorithm to resume processing. Given these experiences and the increased complexity required to operation on more than two vectors at a time, we would suggest using n-moves when $n > 2$ only to move the lattice being reduced out of a local minimum.

2.3.2 Simulated Annealing and Rapid Quenching

Many combinatorial optimization problems have been successfully attacked using *simulated annealing*, which was initially developed independently by [10, 21]. Simulated annealing approaches resemble local optimum algorithms, except that a random component is introduced which allows occasional "uphill" moves (moves which worsen the current solution to the problem according to a cost schedule). As simulated annealing methods have been successfully applied to a wide variety of problems, it seems reasonable to consider adding simulated annealing techniques to Seysen's algorithm in the hope of reducing the number of lcoal minima which cause the algorithm to stop before reaching a global minimum.

Modifying Seysen's algorithm to work along the lines of a simulated annealing approach would not be difficult. In the implementation of the algorithm, we simply need to accept row moves which increase the Seysen measure of the lattice basis. The probability of accepting a move which increases $S(A)$ will depend upon the *temperature* of the reduced lattice, which starts high and decreases according to some *cooling schedule* and the reduction proceeds. It thus remains to specify the initial temperature of a lattice basis, the probability (as a function of temperature) of accepting a row move which increases $S(A)$, and a cooling schedule which describes how temperature decreases with time/reduction steps.

Another technique, based on physical systems, for solving combinatorial optimization problems is the *rapid quenching* approach. Simulated annealing slowly reduces the temperature of the solution, thus gradually reducing the probability of accepting a move to a higher energy/cost state. Rapid quenching, on the other hand, quickly reduces the temperature in the model, bringing the system to a minimum quickly. The system is then reheated to a temperature lower than the initial temperature, and the process is repeated. Seysen's algorithm itself can be viewed as one iteration of a rapid quenching process. The heated system is the initial lattice basis, and the algorithm itself, by greedily reducing the Seysen measure of the lattice basis, decreases the temperature of the system.

We modified our implementation of Seysen's algorithm to simulate multiple rapid quenching iterations. When a lattice basis reached a minimum of the Seysen measure function and no single two-vector row move could decrease $S(A)$, a randomization function was performed on the lattice to "heat" it and Seysen's algorithm was subsequently applied to the heated lattice basis. Our randomizing function chose a linear number of pairs of vectors $(\mathbf{b_i}, \mathbf{b_j}), i \neq j$, and (with equal probability) either added $\mathbf{b_i}$ to or subtracted $\mathbf{b_i}$ from $\mathbf{b_j}$. (This is the same randomizing operation used previously to generate random integer lattices, except that we perform $O(n)$ randomizations here instead of $O(n^2)$ as was done before.) Multiple iterations of the heating/Seysen-reducing process did successfully reduce lattice bases more than Seysen-reduction alone, although it is unclear as to how much benefit can be gained from repeated applications of this process.

2.3.3 Using Hadamard Matrices to Permute Lattice Bases

Our third and last suggestion for moving lattice bases out of local minima was suggested by Matthijs Coster (personal communication). Instead of randomly permuting the lattice basis as random quenching does, Coster suggests using a *Hadamard matrix* H to transform a Seysen-reduced lattice basis B into lattice basis $B' = B\ H$. Recall that matrix H is a Hadamard matrix if it satisfies the following two properties:

1. Each entry $h_{i,j}$ of H is either $+1$ or -1.

2. If $\mathbf{h_1}, \ldots, \mathbf{h_n}$ are the rows of H, then for all i, j with $i \neq j$, $(\mathbf{h_i}, \mathbf{h_j}) = 0$.

It is not difficult to show that if H is an $n \times n$ Hadamard matrix, then $n = 2$ or $n \equiv 0 \bmod 4$ ([1], 2.21 Exercise 10).

Now, consider the lattice basis B' obtained by multiplying B by a Hadamard matrix H. (If $n \not\equiv 0 \bmod 4$ we may consider B and the corresponding lattice L to be sublattices of an n'-dimension space with $n' > n, n' \equiv 0 \bmod 4$.) Each basis vector in B' is a linear combination of all the basis vectors in B, but no two B' vectors have similar constructions, since $\mathbf{h_i}$ and $\mathbf{h_j}$ differ in $\frac{1}{2}n$ coordinates if $i \neq j$. The basis vectors in B' will have lengths $\approx \sqrt{n}$ times the lengths of the basis vectors in B, so while we obtain a good randomization of the lattice, the lengths of the basis vectors are still manageable.

We should point out that the matrix H is not a linear transformation matrix; $\det(H) \neq 1$. This means that the lattice generated by B is not the same lattice generated by B'. However, all n-dimensional Hadamard matrices H_n satisfy:

$$H_n\ H_n^t = n\ I_n.$$

Thus,

$$B\ H_n\ H_n^t = n\ B$$

and the net result of the operation is to scale B (and the associated lattice L) by a factor of n. Thus we can divide out the factor of n and we are left with the lattice L we started with.

So, our plan of attack should be as follows. When Seysen's algorithm stops and reports that lattice basis B is a local minimum of $S(A)$, create $B' = B\ H$ where H

is a Hadamard matrix. Now, Seysen-reduce lattice basis B' until a local minimum is reached. Then compute $B'' = \frac{1}{n}B'H_n^t$. Finally, Seysen-reduce basis B'', producing basis B'''. Bases B and B''' describe the same lattice L. Note that there is no guarantee that $S(A''') < S(A)$, where A, A''' and the quadratic forms associated with B, B''' respectively. Further study is required before we may conclude whether Hadamard permutations provide a reasonable method for "kicking" lattice basis B.

2.4 Extending Seysen's Algorithm

The description Seysen gave in [38] of his algorithm was only an outline of a lattice basis reduction technique. We have tried in this chapter to give both theoretical and empirical reasons for the choices made in implementing Seysen's algorithm. However, we have only touched upon a few of the many possible combinations of techniques. As the next chapter shows, these choices are effective as reducing lattice bases derived from subset sum problems. For other lattices, their effectiveness may be in question. We briefly mention here some of the other possible choices for the various components of Seysen's algorithm.

2.4.1 General n-vector Row Operations

Section 2.3.1 above discussed the possibility of extending Seysen's algorithm to consider row operations involving three and four vectors at once. It is possible to extend these operations to encompass arbitrary k-moves where integer multiples of $k-1$ basis vectors are added to another basis vector. For fixed k, let $\mathbf{b_j}$ be the target basis vector (the vector to be modified) and let $\mathbf{b_{i_1}}, \ldots, \mathbf{b_{i_{k-1}}}$ be the basis vectors to be added to $\mathbf{b_i}$ in multiples of $\lambda_1, \ldots, \lambda_{k-1}$ respectively. Then, after the row move, we will have:

$$\mathbf{b_j} \leftarrow \mathbf{b_j} + \sum_{m=1}^{k-1} \lambda_m \mathbf{b_{i_m}},$$
$$\mathbf{b^*_{i_m}} \leftarrow \mathbf{b^*_{i_m}} - \lambda_m \mathbf{b^*_j}, \quad \text{for } 1 \le m \le k-1.$$

Now, we may solve for the new values of the quadratic forms A and A^* after the row move has occurred. In A, the only value that changes is $a_{j,j}$. Its new value is:

$$a_{j,j} \leftarrow a_{j,j} + \left(\sum_{m=1}^{k-1} a_{i_m,i_m}\lambda_m^2\right) + \sum_{m=1}^{k-1} 2\lambda_m a_{i_m,j} + \sum_{\substack{m,p=1\\ m\neq p}}^{k-1} 2\lambda_m\lambda_p a_{i_m,i_p}. \tag{2.20}$$

In the dual lattice quadratic form A^*, the values $a^*_{i_m,i_m}$ change for $1 \le m \le k-1$. Their new values are:

$$a^*_{i_m,i_m} \leftarrow a^*_{i_m,i_m} - 2\lambda_m a^*_{i_m,j} + \lambda_m^2 a^*_{j,j}, \quad \text{for } 1 \le m \le k-1. \tag{2.21}$$

If we compute Δ, the change in $S(A)$ resulting from this row move, we find that:

$$\begin{aligned}\Delta = {} & a^*_{j,j}\left(\left(\sum_{m=1}^{k-1} a_{i_m,i_m}\lambda_m^2\right) + \sum_{m=1}^{k-1} 2\lambda_m a_{i_m,j} + \sum_{\substack{m,p=1\\ m\neq p}}^{k-1} 2\lambda_m\lambda_p a_{i_m,i_p}\right)\\ & + \sum_{m=1}^{k-1} a_{i_m,i_m}\left(\lambda_m^2 a^*_{j,j} - 2\lambda_m a^*_{i_m,j}\right),\end{aligned} \tag{2.22}$$

$$\begin{aligned}\Delta = {} & 2\sum_{m=1}^{k-1} \lambda_m^2 a^*_{j,j} a_{i_m,i_m} + 2\sum_{m=1}^{k-1} \lambda_m \left(a^*_{j,j} a_{i_m,j} - a_{i_m,i_m} a^*_{i_m,j}\right)\\ & + 2\sum_{\substack{m,p=1\\ m\neq p}}^{k-1} \lambda_m\lambda_p a_{i_m,i_p} a^*_{j,j}.\end{aligned} \tag{2.23}$$

Thus, for all $1 \le m \le k-1$ we have:

$$\frac{\partial}{\partial\lambda_m}\Delta = 4\lambda_m a_{i_m,i_m} a^*_{j,j} + 2(a^*_{j,j} a_{i_m,j} - a_{i_m,i_m} a^*_{i_m,j}) + 2\sum_{\substack{p=1\\ p\neq m}}^{k-1} \lambda_p a_{i_m,i_p} a^*_{i,i}. \tag{2.24}$$

We may thus compute formulae for all λ_m for $1 \le m \le k-1$ if we solve the simultaneous system of equations obtained by setting the $k-1$ derivatives defined in

Equation 2.24 equal to zero. Of course, we still must solve the problem of finding optimal *integer* values for the λ_m. For the 2-move case we showed that $\lceil \lambda_r \rfloor$ was the best integer choice for λ. It is not clear that rounding will work in the k-move case. The real values derived for the λ_m may only indicate some range of integer values which need to be searched.

2.4.2 Alternate Selection Criteria

Seysen's original implementation of his basis reduction algorithm used a "lazy" approach for choosing pairs of basis vectors to reduce. Pairs of integers $(i, j), 1 \leq i, j \leq n$ were searched in lexicographic order until a suitable row move involving $\mathbf{b_i}$ and $\mathbf{b_j}$ was found. We have presented above empirical evidence which favors a "greedy" approach, even when the extra computation time required to implement the "greedy" method is considered.

Selection methods other than the "greedy" and "lazy" approaches were not considered in our experiments, but are certainly possible. For example, in addition to taking into account the reduction in $S(A)$ which will result from a row move, we might also wish to consider the other row moves which will be blocked by performing this move. That is, if $\lambda \mathbf{b_j}$ is added to $\mathbf{b_i}$, the potential $S(A)$ reductions of all other row moves which involve either $\mathbf{b_i}$ or $\mathbf{b_j}$ will be modified. Perhaps we should choose row moves so that the moves they block have minimum $S(A)$ reduction potential. We could combine this idea with the "greedy" method; selecting a row move with the greatest difference between the amount it reduces $S(A)$ and the average $S(A)$ reduction of all the moves it blocks.

2.4.3 Alternate Choices of λ

In Section 2.2.2 above we looked at the effect of placing restrictions on the possible values of λ on the performance of Seysen's algorithm. In particular, λ was allowed either to be any integer value, or to only take on the values ±1. We found that the algorithm worked slightly better if row moves were chosen with $\lambda \in \mathbb{Z}$ but only $T_{i,j}^{\lambda}$ moves with $\lambda = \pm 1$ were actually performed, probably because the changes made to the lattice basis on any single row move are smaller. We pay for the improved performance, however, though an increase in the running time of the overall algorithm, as it takes more row operations with the restriction in place to add a large integer multiple of one basis vector to another basis vector.

As an example of other possible methods of choosing or restricting λ values, consider the following set of restrictions:

- When choosing a pair of vectors for a row move, λ may take on any integer value.

- When the row move is actually performed, if $\lambda > 0$ use the largest power of two strictly less than λ (unless $\lambda = 1$, in which case λ should be used). If $\lambda < 0$ use the smallest power of two strictly greater than λ (again, unless $\lambda = -1$, in which case -1 should be used).

We may abbreviate this set of conditions as $(\mathbb{Z}, 2^k)$. What we are doing is computing the best possible value of λ, but instead of performing one row move to compute $\mathbf{b_j} \leftarrow \mathbf{b_j} + \lambda \mathbf{b_i}$, we perform a logarithmic number of moves. In this way we might be able to combine the benefits of the $(\mathbb{Z}, \mathbb{Z})$ approach (fast running time) and the $(\mathbb{Z}, \pm 1)$ approach (better overall performance). This approach has not been tested, and judging from the relative differences noticed between the $(\mathbb{Z}, \mathbb{Z})$ and $(\mathbb{Z}, \pm 1)$ cases is not likely to produce very large changes in reduction performance. However, it is an example of other possible λ restrictions which could be tried.

2.4.4 Alternate $S(A)$ Functions

Section 2.1.3 above gave reasons for using functions of the product terms $\|\mathbf{b_i}\|\|\mathbf{b_i^*}\|$. In particular, the function

$$S(A) = \sum_{i=1}^{n} a_{i,i}a^*_{i,i} = \sum_{i=1}^{n} \|\mathbf{b_i}\|^2\|\mathbf{b_i^*}\|^2$$

was selected as the Seysen measure function because it yielded a closed form solution for the optimal value of λ given i and j. However, other functions could certainly be employed as the method of comparing different lattice bases. In this section we briefly describe how Seysen's algorithm would have to be modified to accommodate another measure function.

We restrict our attention to versions of Seysen's algorithm which use only row moves involving two basis vectors (i.e. $\mathbf{b_j} \leftarrow \mathbf{b_j} + \lambda\mathbf{b_i}$). Recall that the formula in Equation 2.9 for the optimal choice of λ was derived by maximizing the change in the Seysen measure function caused by a row move involving two particular basis vectors. In the Seysen measure function is changed, the only direct impact it will have upon the operation of the algorithm is that the optimal value of λ for basis vectors $(\mathbf{b_i}, \mathbf{b_j})$ will be computed in a different manner.

In [38] Seysen mentions two possible replacements for the $S(A)$ function:

$$S_1(A) = \prod_{i=1}^{n} a_{i,i}a^*_{i,i} = \prod_{i=1}^{n} \|\mathbf{b_i}\|^2\|\mathbf{b_i^*}\|^2,$$
$$S_2(A) = \sum_{i=1}^{n} \sqrt{a_{i,i}a^*_{i,i}} = \sum_{i=1}^{n} \|\mathbf{b_i}\|\|\mathbf{b_i^*}\|.$$

Replacing $S(A)$ with $S_1(A)$ implies that our choice of λ must minimize:

$$\begin{aligned}\Delta(i,j,\lambda) &= S_1(T^\lambda_{j,i}\, A\, T^\lambda_{i,j}) - S_1(A), \\ &= \left(\prod_{\substack{m=1 \\ m\notin\{i,j\}}}^{n} a_{i,i}a^*_{i,i}\right)\left(a'_{i,i}{a^*_{i,i}}'a_{j,j}{a^*_{j,j}}' - a_{i,i}a^*_{i,i}a_{j,j}a^*_{j,j}\right).\end{aligned}$$

Solving $\frac{\partial\Delta}{\partial\lambda} = 0$ yields an extremely complex expression for λ. Similar results occur when we try to substitute $S_2(A)$ for $S(A)$. In both cases, no simple closed-form solution for λ exists as was the case with $S(A)$.

It may be still be possible to utilize measure functions in Seysen's algorithm with no simple closed-form solution for λ if we are willing to sacrifice some performance. If the range of possible integer λ values is bounded, for given i and j we can compute $\Delta(i, j, \lambda)$ for all possible λ values in the permitted range. The λ which provides the greatest change may then be selected. The cost of this procedure is that we can no longer guarantee that the maximal λ for a pair of basis vectors $(\mathbf{b_i}, \mathbf{b_j})$ may be found in constant time. If the range of λ values to consider is usually small, then we will probably notice little more than a linear slowdown in the running time of the algorithm. For large ranges of possible λ values, further heuristics might be applied, such as only considering λ values which are near a power of two.

In our experiments we noticed that large λ values tended to occur only during the first few row reduction performed by Seysen's algorithm. After this initial burst in reduction of $S(A)$ row moves tended only to involve small integer λ values; it was quite rare to find $|\lambda| > 10$. If similar conditions occur for the lattice bases in question, it is probably reasonable to use a more complex measure function than $S(A)$ and use a small exhaustive search over a bounded range of possible λ values to find the optimal λ coefficient for a row move.

Chapter 3

Solving Subset Sum Problems

3.1 Introduction

In the previous chapter we discussed the theoretical and empirical implications of Seysen's basis reduction algorithm. As Chapter 1 pointed out, many problems in discrete mathematics may be reduced to problems involving lattices and basis reduction. In this chapter we use Seysen's algorithm to solve *subset sum problems.* The subset sum problem in the general case is NP-complete, and many *knapsack-type cryptosystems* have been suggested which depend on the difficulty of solving subset sum problems. We show below that Seysen's algorithm, when used in conjunction with the LLL algorithm and other techniques, allows us to solve a large class of subset sum problems in polynomial time.

We begin our analysis with the definition of a *subset sum problem.*

Definition. *Let $A = \{a_1, \ldots, a_n\}$ be a set of positive integers (the weights). Let $A' \subset A$ be some subset of A, and let s be the sum of the elements of A'. Then*

$$s = \sum_{i=1}^{n} e_i a_i, \quad \text{for } e_i \in \{0,1\}, 1 \leq i \leq n. \tag{3.1}$$

The subset sum or knapsack problem is to find, given the set A of weights and the sum s, some subset A' of A. Equivalently, one may find a set of values for the 0-1 *variables $e_1, \ldots, e_n$, where $e_i = 1$ if and only if $a_i \in A'$.*

The subset sum problem is known to be NP-complete [15]. In the INSTANCE-QUESTION format often used to phrase NP-complete decision problems, the subset sum decision problem would be described as follows:

> INSTANCE A set of positive integers $A = \{a_1, \ldots, a_n\}$ and an integer s.
>
> QUESTION Does there exist a subset A' of A such that the sum of the elements of A' is s?

Clearly, if we can solve subset sum problems in polynomial time, we can answer the subset sum decision problem in polynomial time. The converse is also true; we can ask an oracle which answers the subset sum decision problem in polynomial time whether there is a solution to the subset sum problem with weights $a_2, \ldots, a_n$ and sum $s - a_1$. If there is such a solution, then there exists a solution to the original problem that included a_1 (i.e. we can set $e_1 = 1$). If the oracle say no such solution exists, then a_1 cannot be in the subset which sums to s, and we know that $e_1 = 0$. We can then recurse and determine $e_2, e_3, \ldots, e_n$ in sequence.

Many public-key cryptosystems have been proposed with the difficulty of solving subset sum problems as the basis for their security. (See [7, 8, 13, 31] for surveys of this field.) Almost all of these cryptosystems have been shown to be insecure; the Chor-Rivest one [11] is perhaps the most widely known system which has not yet been broken. The majority of the attacks on knapsack-based cryptosystems have involved discovering the secret information hidden in the weights which allows the receiver A to decrypt the message quickly. However, there have been two independent attacks, one due to Brickell [6] and one due to Lagarias and Odlyzko [26], which attempt to solve all subset sum problems of a certain type, independent of the method in which the weights were chosen. These methods (and the newer result in [12]) depend in theory only on the *density* of the subset sum problem to be solved. In practice, however, the success rate of these methods is bounded by the performance of the basis reduction technique used in the attack.

Section 3.2 below outlines currently known methods for solving subset sum problems, and describes a new method which significantly increases the class of problems which may be attacked in practice. Section 3.3 discusses current methods for actually solving a specific subset sum problem, and the limits of these methods. In Section 3.4 we show how to use Seysen's algorithm in conjunction with multiple versions of the LLL algorithm and other heuristics to solve a larger class of subset sum problems. Section 3.5 presents empirical results obtained by solving a large number of subset sum problems using Seysen's algorithm. These results give experimental evidence that it is possible to solve a much larger class of subset sum problems in polynomial time than was previously thought possible.

3.2 Theoretical Bounds on Solving Subset Sum Problems

The majority of attacks on knapsack-based cryptosystems exploit the specific constructions of the cryptosystems. Two algorithms have been proposed, however, which depend only on the properties of the subset sum problem and not on any specific method of construction. These algorithms, one by Brickell [6] and one by Lagarias and Odlyzko [26], show that almost all *low-density* subset sum problem may be solved in polynomial time. The *density* d of a set of weights $a_1, \ldots, a_n$ is defined by

$$d = \frac{n}{\log_2 \max\limits_{1 \leq i \leq n} a_i}. \tag{3.2}$$

For $d > 1$ there will in general be many subsets of weight with the same sum s, so from an encryption-decryption point of view we are interested in subset sum problem instances with $d \leq 1$. The Brickell and Lagarias-Odlyzko algorithms show that it is possible to solve almost all subset sum problems with d sufficiently small.

The Brickell and Lagarias-Odlyzko attacks reduce the subset sum problem to the problem of finding the Euclidean-norm shortest nonzero vector in a lattice. As was mentioned in Section 1.1 above, finding short vectors in lattices may be very hard in general. The theoretical worst-case bounds for the LLL algorithm and its variants are not encouraging, and no bound currently exists for Seysen's algorithm. However, these techniques tend to perform much better in practice than in theory; the performance of Seysen's algorithm on the Korkin-Zolotarev test lattice with $\theta = 0.4$ (Section 2.2.3) is one such empirical example. Thus, it seems important to separate the efficiency of lattice reduction and finding short nonzero vectors from the difficulty in reducing subset sum problems to lattice reduction questions.

We consider a *Euclidean-norm lattice oracle* (or *lattice oracle* for short) that, when given a lattice basis as its input, with high probability finds in polynomial time the Euclidean-norm shortest nonzero vector in the given lattice. We do not know how to construct such an oracle, but it might be possible to do so. Data provided in [26, 33] show that at low densities the LLL algorithm behaves as a lattice oracle, and our results in Section 3.5 below show that this is also the case for a combination of the Seysen and LLL algorithms, even for significantly larger and denser subset sum problems. Given the existence of a lattice oracle, the analysis in [26] shows that it is possible to solve almost all subset sum problems of density $d < 0.6463\ldots$ in polynomial time. Recently, Coster, LaMacchia, Odlyzko and Schnorr [12] and Joux and Stern [19] independently demonstrated via different techniques that this bound could be improved to $d < 0.9408$. In fact, if we assume the existence of a *sup-norm lattice oracle* instead of a Euclidean-norm lattice oracle, [12] showed that the density bound then becomes $d < 1$.

The Lagarias-Odlyzko attack proceeds as follows. Let $\{a_1, \ldots, a_n\}$ be a set of weights with $0 \leq a_i \leq A$ for some positive integer A and for all $1 \leq i \leq n$. Let

$\mathbf{e} = (e_1, \ldots, e_n) \in \{0,1\}^n, \mathbf{e} \neq (0, 0, \ldots, 0)$ be fixed and depend only on n. Then

$$s = \sum_{i=1}^{n} e_i a_i, \quad \text{for } e_i \in \{0,1\},$$

is the sum of the subset of weights, where a_i is in the subset if and only if $e_i = 1$. Now, define basis vectors $\mathbf{b_1}, \ldots, \mathbf{b_{n+1}}$ as follows:

$$\begin{aligned} \mathbf{b_1} &= (1, 0, \ldots, 0, Na_1), \\ \mathbf{b_2} &= (0, 1, \ldots, 0, Na_2), \\ &\vdots \\ \mathbf{b_n} &= (0, 0, \ldots, 1, Na_n), \\ \mathbf{b_{n+1}} &= (0, 0, \ldots, 0, Ns), \end{aligned}$$

where N is a positive integer $> \sqrt{n}$. Let L be the lattice defined by the basis vectors $\mathbf{b_1}, \ldots, \mathbf{b_{n+1}}$. That is,

$$L = \left\{ \sum_{i=1}^{n+1} z_i \mathbf{b_i} \colon z_i \in \mathbb{Z}, \quad \text{for } 1 \leq i \leq n+1 \right\}.$$

Notice that lattice L contains the vector $\hat{\mathbf{e}} = (e_1, e_2, \ldots, e_n, 0)$, the solution vector to the subset sum problem, since

$$\hat{\mathbf{e}} = \sum_{i=1}^{n} e_i \mathbf{b_i} - \mathbf{b_{n+1}}.$$

Let P denote the probability that there exists another vector $\hat{\mathbf{x}} \in L$ such that $||\hat{\mathbf{x}}|| \leq ||\hat{\mathbf{e}}||$ and $\hat{\mathbf{x}} \notin \{\mathbf{0}, \hat{\mathbf{e}}, -\hat{\mathbf{e}}\}$. The simplified analysis of the Lagarias-Odlyzko attack presented in [14] shows that this probability is bounded:

$$P \leq n \left(2n\sqrt{\tfrac{1}{2}n} + 1 \right) \frac{2^{c_0 n}}{A}, \quad \text{for } c_0 = 1.54724\ldots \tag{3.3}$$

Thus, if the bound on the size of the weights $A = 2^{cn}$ with $c > c_0$, $\lim_{n \to \infty} P = 0$. If the density of a subset sum problem is less than $0.6463\ldots$, then

$$\begin{aligned} \frac{n}{\log_2 \max\limits_{1 \leq i \leq n} a_i} < 0.6463\ldots &\Longrightarrow \max_{1 \leq i \leq n} a_i > 2^{n/0.6463\ldots} \\ &\Longrightarrow A > 2^{c_0 n}. \end{aligned}$$

Thus, all subset sum problems with density $< 0.6463\ldots$ could be solved in polynomial time, given the existence of a lattice oracle.

Recently, two independent improvements [12, 19] to the Lagarias-Odlyzko attack have been developed, both of which increase the density bound to $d < 0.9408\ldots$. The modification suggested in [12] is to replace the $\mathbf{b_{n+1}}$ basis vector in L with

$$\mathbf{b'_{n+1}} = (\tfrac{1}{2}, \tfrac{1}{2}, \ldots, \tfrac{1}{2}, Ns).$$

Let L' be the lattice spanned by the vectors $\mathbf{b_1}, \ldots, \mathbf{b_n}, \mathbf{b'_{n+1}}$. Lattice L' does not contain the solution vector $\hat{\mathbf{e}}$, but it contains a similar vector $\hat{\mathbf{e}}'$:

$$\hat{\mathbf{e}}' = (e'_1, \ldots, e'_n, 0), \quad \text{where } e'_i = e_i - \tfrac{1}{2}.$$

We know that $e'_i \in \{-\frac{1}{2}, \frac{1}{2}\}$ for $1 \leq i \leq n$ since $e_i \in \{0, 1\}$ for $1 \leq i \leq n$. Then $||\hat{\mathbf{e}}'||^2 \leq \frac{1}{4}n$ independent of the number of e_i's which are equal to 1.

Using lattice L' we are now interested in the probability P' that there exists a vector $\hat{\mathbf{x}}' \in L'$ such that:

$$\begin{aligned} ||\hat{\mathbf{x}}'|| \leq ||\hat{\mathbf{e}}'|| &\leq \tfrac{1}{2}\sqrt{n}, \\ \hat{\mathbf{x}}' &\notin \{\mathbf{0}, \hat{\mathbf{e}}', -\hat{\mathbf{e}}'\}. \end{aligned} \tag{3.4}$$

Utilizing similar techniques to those in [14, 26, 28], [12] shows that the probability P' is bounded above by:

$$P' \leq n\left(4n\sqrt{n} + 1\right)\frac{2^{c'_0 n}}{A} \quad \text{for } c'_0 = 1.0628\ldots. \tag{3.5}$$

This bound is similar to that in Equation 3.3 above. Since $1/c'_0 = 0.9408\ldots$, any subset sum problem with density $d < 0.9408\ldots$ may be solved in polynomial time, given the existence of a lattice oracle.

3.3 Previous Empirical Methods

Along with their theoretical results, Lagarias and Odlyzko [26] presented in 1985 the results of the first empirical attacks on general subset sum problems. Their method was to apply a multiprecision version of the LLL algorithm to the basis L presented in Section 3.2 above ($\mathbf{b_{n+1}} = (0, 0, 0, \ldots, 0, s)$). Lagarias and Odlyzko set the LLL parameter $y = 1$, which they said yielded better results than $y = 0.75$ but tripled execution time in practice. Also, five random orderings of the basis vectors for lattice L were tried, since different initial orderings yield different LLL-reduced bases. Experiments were conducted on various subset sum problems with $n \leq 50$ and densities d between 0.5 and 0.875. Figure 3-1 graphically shows the performance of the Lagarias-Odlyzko method. For each tested value of n, the labeled curve connects (density,success rate) points in the plot obtained by attempting to solve subset sum problems of size n.

For $n \leq 26$, the LLL algorithm appeared to function almost as well as a square-norm lattice oracle; all subset sum problems with density $d \leq 0.6408\ldots$ and $n \leq 26$ were solved when LLL was used on five random permutations of the basis vectors. However, performance degrades quickly as n grows above 30. For $n = 40$, Lagarias and Odlyzko were able to solve all attempted subset sum problems only for density 0.5. At $n = 50$, only two-thirds of the attempted density 0.5 problems were solved.

In [33] Radziszowski and Kreher reported the results of extensive attempts to solve subset sum problems. Their reduction algorithm, based on LLL, differed from that used in Lagarias-Odlyzko in two important ways. First, Radziszowski and Kreher modified the LLL algorithm to reduce the number of required multiprecision calculations. In essence, instead of running LLL once on a lattice with numbers k bits long, they ran LLL m times with numbers k/m bits long (a divide-and-conquer approach). This modification sped up their algorithm and allowed them to attack larger subset sum problems than Lagarias and Odlyzko (larger here means greater value of n).

The second change Radziszowski and Kreher made to the Lagarias-Odlyzko attack was to use another algorithm in conjunction with LLL to find short vectors in the lattice. This second algorithm, called *Weight-Reduction*, searches for pairs of vectors

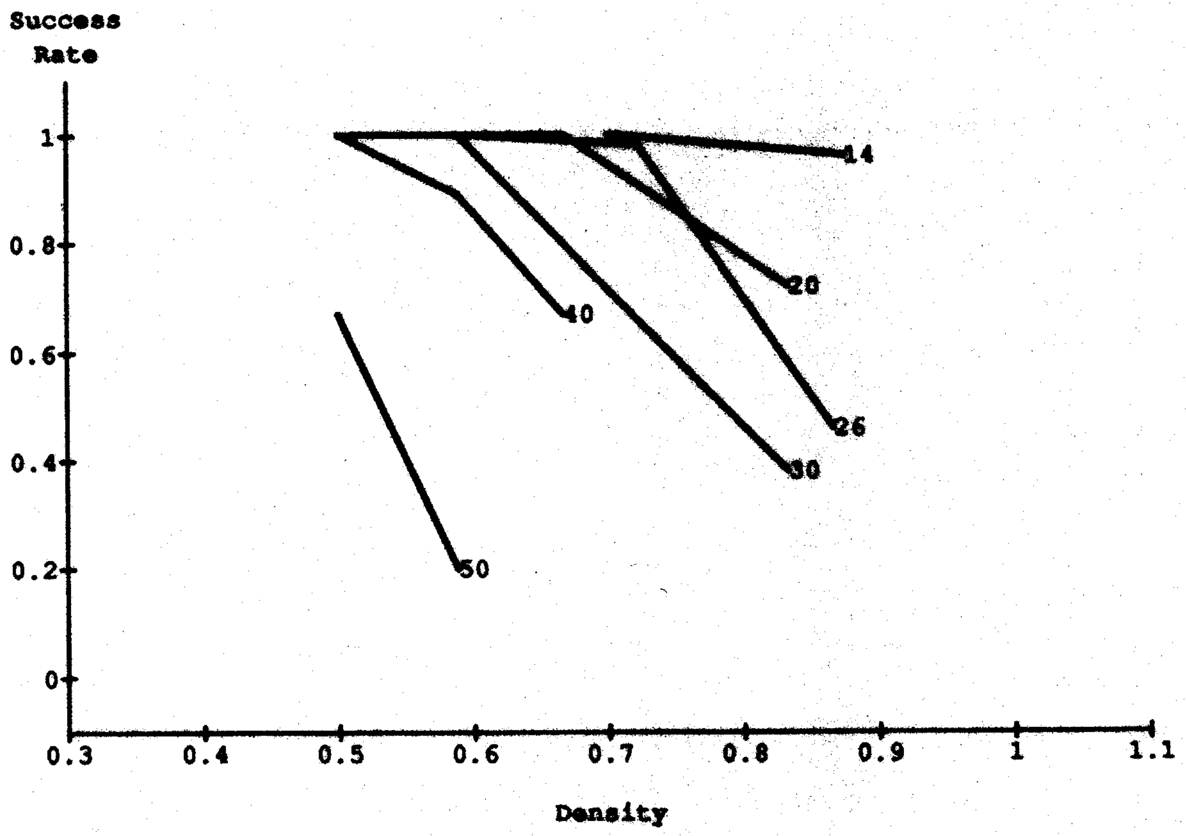

Figure 3-1: Lagarias-Odlyzko Results: Success Rate vs. Density

$(\mathbf{b_i}, \mathbf{b_j})$ in the lattice for which:

$$\|\mathbf{b_i} + \epsilon \mathbf{b_j}\| < \max\{\|\mathbf{b_i}\|, \|\mathbf{b_j}\|\}, \quad \text{for } \epsilon = \pm 1. \tag{3.6}$$

When *Weight-Reduction* finds such a pair, it replaces the larger of $\mathbf{b_i}$ and $\mathbf{b_j}$ (in terms of square-norm) by the sum $\mathbf{b_i} + \epsilon \mathbf{b_j}$. Equation 3.6 is satisfied by a pair $(i, j), i \neq j$ if and only if

$$\max\{\|\mathbf{b_i}\|^2, \|\mathbf{b_j}\|^2\} < 2 \cdot \|(\mathbf{b_i}, \mathbf{b_j})\|.$$

The *Weight-Reduction* algorithm can easily be implemented in $O(n^2)$ time to search for all pairs of vectors $(\mathbf{b_i}, \mathbf{b_j})$ which satisfy Equation 3.7.

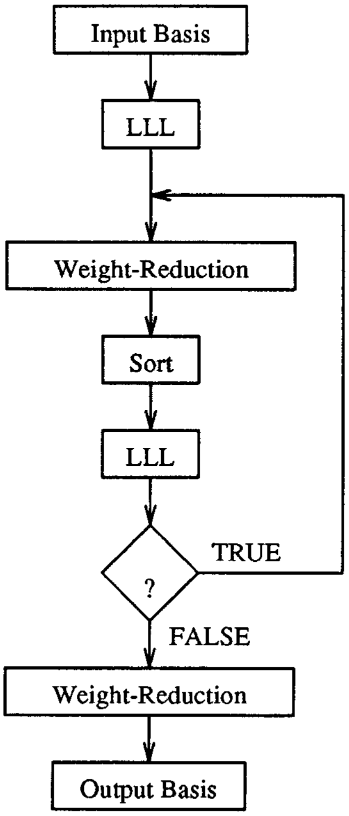

Figure 3-2: Radziszowski-Kreher Inner Loop

Radziszowski and Kreher alternated calls to the modified LLL algorithm with calls to *Weight-Reduction* and a sorting procedure to improve the search for short vectors in the lattice. Figure 3-2 is a flowchart-like representation of the process. The initial basis L is LLL-reduced ($y = 0.99$ for all invocations of LLL). The output of LLL is then *Weight-Reduced*, sorted by length, and then fed back into LLL. This process continues until some termination condition is met. In theory, "no decrease in $\sum_{i=1}^{n} \|\mathbf{b_i}\|$" could be used as the termination condition. In their experiments, Radziszowski and Kreher set an explicit bound on the maximum number of loops which could be performed. The output of this iterative process is then *Weight-Reduced* one more time.

One important feature of the Radziszowski-Kreher loop is the insertion of the sorting phase before LLL is run each time. By sorting the vectors in the lattice by length, the length of the shortest vector in L is guaranteed not to increase by application of LLL. If L is a lattice output by the sort procedure, then vector $\mathbf{b_1}$ in L is the shortest vector in the lattice. Now, LLL can replace $\mathbf{b_1}$ with some vector $\mathbf{b}_2'$ only if $\|\mathbf{b}_2'\| < \|\mathbf{b_1}\|$. (This does not hold for $\mathbf{b_i}$ in general but is true for $\mathbf{b_1}$.) Thus, we are guaranteed that the shortest vector in L before LLL is applied will not disappear from the basis unless an even shorter vector is found.

Figure 3-3 shows graphically the results of the experiments carried out in [33]. For these experiments, only 15 loop iterations were allowed (9 if all the vectors in the basis were of length $< \frac{1}{2}n$). For values of $n \leq 34$, this algorithm was able to solve all attempted subset sum problems of density $d \leq 0.654$, which is above the Lagarias-Odlyzko bound. As n increases from 42 to 98 the density at which all attempted problems were solved decreases. As was reported in [26], the effectiveness of LLL as a lattice oracle drops off as the size of the lattice grows. Radziszowski and Kreher's results show however that it is possible to increase the effective range of LLL by combining it with other heuristic techniques.

The Radziszowski-Kreher algorithm was the best known method to date for solving subset sum problems. In the following section, we show how to to combine the Seysen basis reduction algorithm with the LLL algorithm, other heuristics, and the theoretical improvements in [12] (Section 3.2) to greatly extend the range of subset

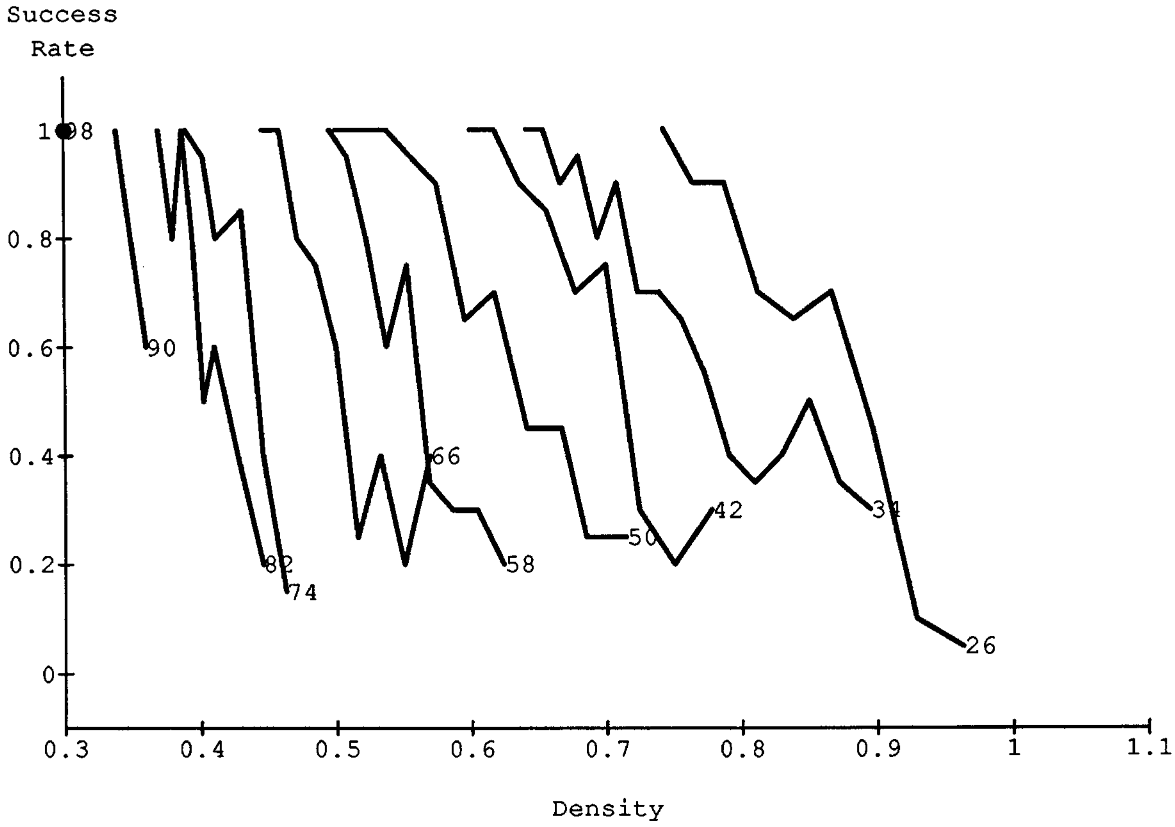

Figure 3-3: Radziszowski-Kreher Results: Success Rate vs. Density

sum problems which may be empirically attacked and successfully solved.

3.4 Using Seysen's Algorithm to Solve Subset Sum Problems

In both [26] and [33] the LLL algorithm was used almost exclusively to perform the required lattice basis reductions. Given the existence of Seysen's basis reduction algorithm and knowledge as to how it performs relative to LLL, it is natural to wonder how Seysen's algorithm could be applied to solving subset sum problems. We know from some of the comparisons performed between LLL and Seysen's algorithm in Chapter 2 that Seysen's algorithm tends to perform fewer row operations than LLL. If we used Seysen's algorithm then as part of an attack on subset sum problems, we should be able to perform more reduction operations (or other work) without increasing the overall execution time of our method.

There are also theoretical and empirical reasons to suspect that Seysen's algorithm alone will *not* perform well on subset sum problems. Seysen's algorithm was originally suggested for simultaneously reducing a lattice and its dual lattice, not for finding the shortest vector in a lattice. If a lattice and its dual are of comparable size, the Seysen algorithm is not likely to perform row operations that generate short vectors in one lattice if that move increases the lengths of vectors in the dual lattice. If the vectors in the dual are significantly larger than those in the prime lattice, Seysen's algorithm may actually produce vectors larger than those originally input. Also, the algorithm's operation is not dependent on the ordering of the basis vectors, which means that the benefits gained by running LLL multiple times on different randomizations of the subset sum lattice disappear.

These predicted benefits and deficiencies of the Seysen algorithm suggest that the best place to use Seysen is at the beginning of our attempt to solve a subset sum problem. The initial subset sum lattice L has primal basis vectors which are much larger than those in the dual; this favors the generation of shorter vectors in L and longer vectors in L^*. Also, as Seysen tends to perform many fewer row moves than LLL to reach the same reduced lattice basis, using Seysen's algorithm initially will reduce the total number of multiprecision row moves as compared to using only LLL. Once Seysen's algorithm stops at a local minimum, we can use LLL and other techniques to further reduce just the primal lattice L and look for short vectors.

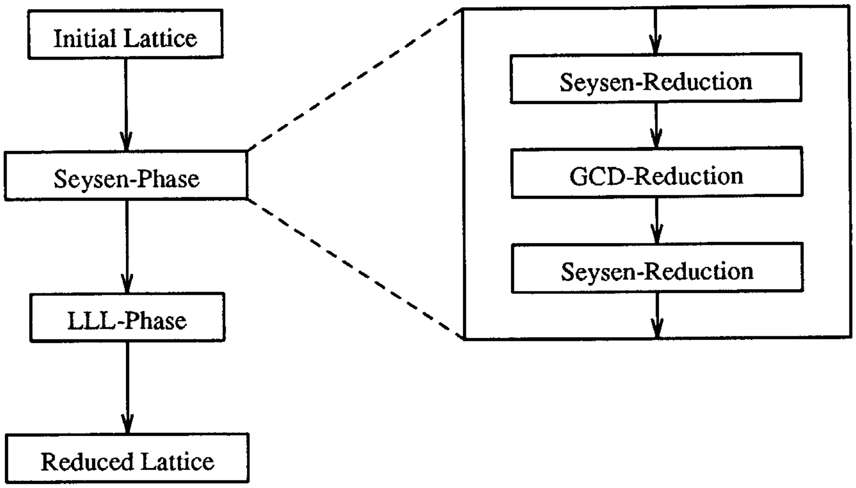

Figure 3-4: Overview of Algorithm **SL**

Figure 3-4 shows the basic outline of the **SL** (Seysen-Lovász) algorithm. We start with the initial lattice basis suggested in [26]. If the weights of the subset sum problem are sufficiently large, then the lengths of the basis vectors in L will be much greater than the corresponding lengths of the basis vectors in the dual lattice L^*. We apply the Seysen algorithm to the L, L^* self-dual pair of lattice bases, using a greedy algorithm (Section 2.1.2) to choose at each step the pair of basis vectors $(\mathbf{b_i}, \mathbf{b_j})$ to be reduced. The measure function $S(A)$ is the one suggested by Seysen [38]:

$$S(A) = \sum_{i=1}^{n} \|\mathbf{b_i}\|^2 \|\mathbf{b_i^*}\|^2.$$

When computing the change in $S(A)$ for each pair of vectors $(\mathbf{b_i}, \mathbf{b_j}), i \neq j$, λ is always set to its maximum value of:

$$\lambda = \left\lceil \left(\frac{a^*_{i,j}}{a^*_{j,j}} - \frac{a_{i,j}}{a_{i,i}} \right) \right\rfloor$$

Let L_{S_1} be the lattice basis which is produced by this (first) application of the Seysen algorithm.

Recall that the goal of the **SL** algorithm is to find a basis vector which describes the solution to the given subset sum problem. In [26, 33] the desired solution vector **e** was always of the form $(e_1, ..., e_n)$ with e_i equal to either 0 or a fixed constant κ. We will call all such vectors **e** *Type-I* solution vectors, to designate them from another class of solution vectors which will appear below. At the completion of the first Seysen reduction in the **SL** algorithm, it is possible that lattice basis L_{S_1} contains a Type-I solution vector. If so, then we are finished, and Algorithm **SL** halts. We assume that L_{S_1} does not contain a Type-I solution vector, and continue with the next stage of the algorithm.

The algorithms of [26, 33] searched for short vectors of the form $(e_1, \ldots, e_n, e_{n+1} = 0)$ in the LLL-reduced bases, where $e_i \in \{0, \kappa\}$ for $1 \leq i \leq n$ and $\kappa \in \mathbb{Z}$ is any fixed integer. One of the problems with these methods is that often the short vectors produced by LLL reduction had $e_{n+1} \neq 0$; that is, the sum $\sum e_i a_i$ was not of the form $s + yt$ and did not describe any relation involving the target subset sum. One method of reducing the probability that LLL (or Seysen) will include a vector of this form in the reduced basis is to scale the a_i and s by some constant factor N, which increases the length of any vector having a nonzero e_{n+1} by about $\sqrt{N}$ if N is sufficiently large. However, this approach has the drawback that it increases the size of numbers which are already quite large and require multi-precision arithmetic. We suggest another method for elimination from consideration all lattice vectors with $e_{n+1} \neq 0$: *GCD-reduction.*

The idea behind *GCD-Reduction* is to perform row moves on the lattice basis so that the entire $(n+1)^{\text{th}}$ column contains exactly one nonzero element, the GCD of the weights $a_1, \dots, a_n$. If (for example) vector $\mathbf{b_{n+1}}$ contains the GCD, we can remove $\mathbf{b_{n+1}}$ from the basis and remove the $(n+1)^{\text{st}}$ column completely from the lattice basis. This reduces what was an $(n+1)$-dimension lattice to an n-dimension lattice, and guarantees that any lattice vector generated by reducing this basis would have had its $(n+1)^{\text{st}}$ component equal to 0 in the $(n+1)$-dimension space.

Implementing *GCD-Reduction* is easy. The basic algorithm we use was described by Brun [9]. Basis vectors are sorted in order of decreasing $b_{i,n+1}$. Then $\mathbf{b_2}$ is repeatedly subtracted from $\mathbf{b_1}$ until $b_{2,n+1} > b_{1,n+1}$. The vectors are then resorted in order of decreasing $b_{i,n+1}$ and the process loops. In reality, only vector $\mathbf{b_1}$ needs to be inserted into the previously generated order. This can be performed very fast by using an auxiliary pointer array; the actual basis vectors don't even need to be moved around. Eventually, the only nonzero element in the $(n+1)^{\text{st}}$ will be $b_{1,n+1}$, at which point both the vector $\mathbf{b_1}$ and all $b_{i,n+1}, 2 \leq i \leq n+1$ can be removed from the lattice.

We apply *GCD-Reduction* to the lattice basis L_{S_1} output by the first application of Seysen's algorithm. We do not use *GCD-Reduction* on the original lattice basis L because the resulting lattice basis matrix would be quite dense (which means the Seysen algorithm will take longer to run) and the vectors in the dual lattice would also be much larger. After *GCD-Reduction* is applied to basis L_{S_1} (yielding output basis L_G in n dimensions) we again search for the presence of a Type-I solution vector. Assuming we do not find such a vector, the Seysen algorithm is applied to L_G to reduce the lattice to a local minimum of the measure function $S(A)$. Call this lattice L_{S_2}, the output of the second Seysen application.

For small values of n ($n \lesssim 20$), it is possible that L_{S_2} will contain a Type-I solution vector. For larger values of n and sufficiently high densities d, however, it is unlikely that the Seysen-*GCD-Reduction*-Seysen portion of the algorithm will have found the desired vector. Thus we begin the second portion of the algorithm: the LLL phase.

Figure 3-5 shows the first level of detail in the LLL-phase of Algorithm **SL**. The input to this phase of the algorithm is the lattice basis L_{S_2} which was the result of

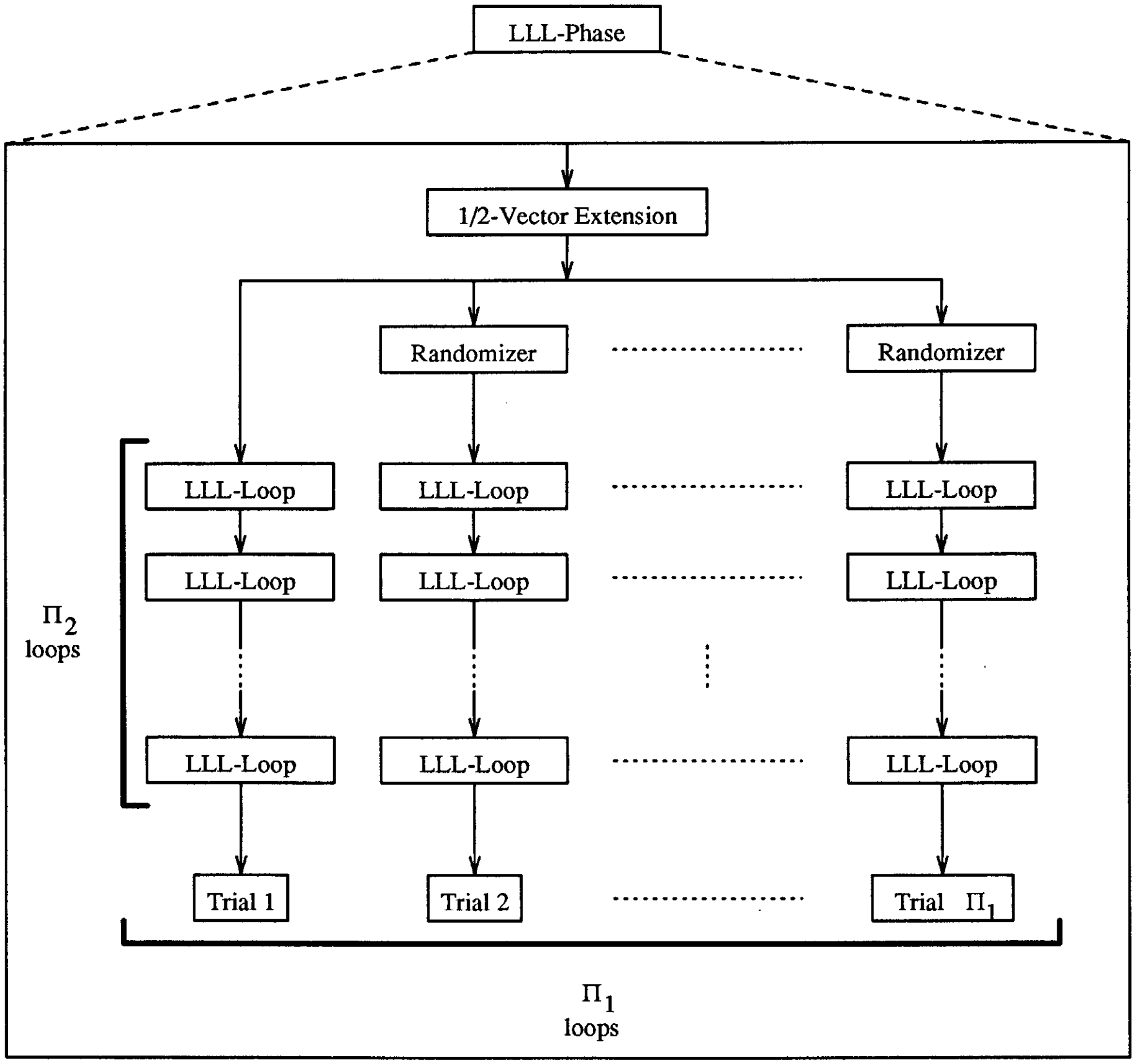

Figure 3-5: Algorithm **SL**: LLL-Phase

the second application of Seysen reduction. We now take advantage of the theoretical improvements in the density bound given in [12] and Section 3.2 above. (The introduction of the "one-half" vector is delayed until after the Seysen stage of Algorithm **SL** has completed to increase the sparsity of the lattice bases which are Seysen-reduced.) We extend the $\mathbb{R}^n$ lattice described by L_{S_2} into a lattice in $\mathbb{R}^{n+1}$ by adding an $(n+1)^{\text{st}}$ component to each of the basis vectors $\mathbf{b_1}, \ldots, \mathbf{b_n}$:

$$\begin{aligned} \text{old } \mathbf{b_i} &= (b_{i,1}, b_{i,2}, \ldots, b_{i,n}) \\ \text{new } \mathbf{b_i} &= (b_{i,1}, b_{i,2}, \ldots, b_{i,n}, b_{i,n+1} = \sum_{j=1}^{n} b_{i,j}) \end{aligned}$$

The new $(n+1)^{\text{st}}$ component of each basis vector is simply the sum of the first n components. To complete the extension, we need to add one more vector to the lattice basis. New basis vector $\mathbf{b_{n+1}}$ is defined as follows:

$$\mathbf{b_{n+1}} = (\underbrace{\tfrac{1}{2}, \tfrac{1}{2}, \ldots, \tfrac{1}{2}}_{n \text{ terms}}, \tfrac{1}{2}(n-1))$$

For $\mathbf{b_{n+1}}$, the $(n+1)^{\text{st}}$ component is the sum of the first n terms minus $\frac{1}{2}$. The $-\frac{1}{2}$ correction is needed because otherwise the $(n+1)^{\text{st}}$ column of the basis would be dependent. Let L_{in} designate this extended lattice.

With the introduction of vector $\mathbf{b_{n+1}}$ we have also introduced a second class of solution vectors. If $\mathbf{e}$ is a Type-I solution vector for a subset sum problem with $\frac{1}{2}n$ elements in the subset, then there now exists a *Type-II* vector in the lattice of the form

$$\hat{\mathbf{e}} = (\hat{e}_1, \ldots, \hat{e}_n, \hat{e}_{n+1}), \quad \text{for } \hat{e}_i \in \{-\tfrac{1}{2}, \tfrac{1}{2}\}, 1 \le i \le n+1.$$

Also, if $\mathbf{e}$ was a solution vector of the n-dimensional lattice, in the extended lattice L_{in} vector $\mathbf{e}$ will be transformed into $\mathbf{e}' = (e_1, \ldots, e_n, e_{n+1})$ where e_{n+1} equals κ times the number of elements in the subset which sums to s. Thus, during the LLL phase of Algorithm **SL**, we search for both extended Type-I solution vectors and Type-II solution vectors.

The structure of the LLL phase of Algorithm **SL** is based upon the successful attacks of [26, 33]. Lagarias and Odlyzko showed that using LLL multiple times on random orderings of the input basis improved the chances of finding a solution vector. Thus, Algorithm **SL** initially tries to reduce lattice L_{in}, and upon failure randomizes L_{in} and tries again. This process continues until a total of π_1 reduction attempts have been made (L_{in} and $\pi_1 - 1$ random orderings of L_{in}). In [26] $\pi_1 = 5$, but there is nothing special about that particular value.

Recall from Section 3.3 above that the Radziszowski-Kreher approach, while setting $\pi_1 = 1$, repeatedly called LLL in conjunction with *Weight-Reduction* and *Sort* (*Sort* just sorts the basis vectors by length.). They showed that repeated iterations of the LLL-*Weight-Reduction-Sort* loop shown in Figure 3-2 yielded better results than a single application of LLL. Algorithm **SL** incorporates this approach, applying a number of iterations of LLL-*Weight-Reduction-Sort* before giving up and trying a new randomization of L_{in} (Figure 3-5). Each ordering of L_{in} passes though up to π_2 iterations of the loop before Algorithm **SL** gives up. As stated above, in [33] $\pi_2 = 15$, or $\pi_2 = 9$ if all the vectors in the lattice were of length $< \frac{1}{2}n$. Again, there is nothing special about these particular values. Theoretically, one could choose π_1 and π_2 to be quite large. Any practical algorithm, though, would probably want to use reasonable constants for both π_1 and π_2.

Although both [26] and [33] run the LLL algorithm with the parameter $y \approx 0.99$, Lenstra, Lenstra, and Lovász show that y may be any value in the range $\frac{1}{4} < y < 1$ [27]. For $y \approx \frac{1}{4}$, LLL will only exchange vectors in the lattice (Equation 1.2) for relatively large differences between vectors $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$. As $y \to 1$, the amount of improvement required to trigger the LLL exchange step decreases, and LLL will swap vectors and continue running for minimal improvement. Thus, larger values of y will likely lead to improved results, but LLL will also take longer to run. This suggests that instead of running the entire LLL algorithm with $y = 0.99$, as was done previously, a version of LLL which started with $y \approx \frac{1}{4}$ and adjusted it upwards as the

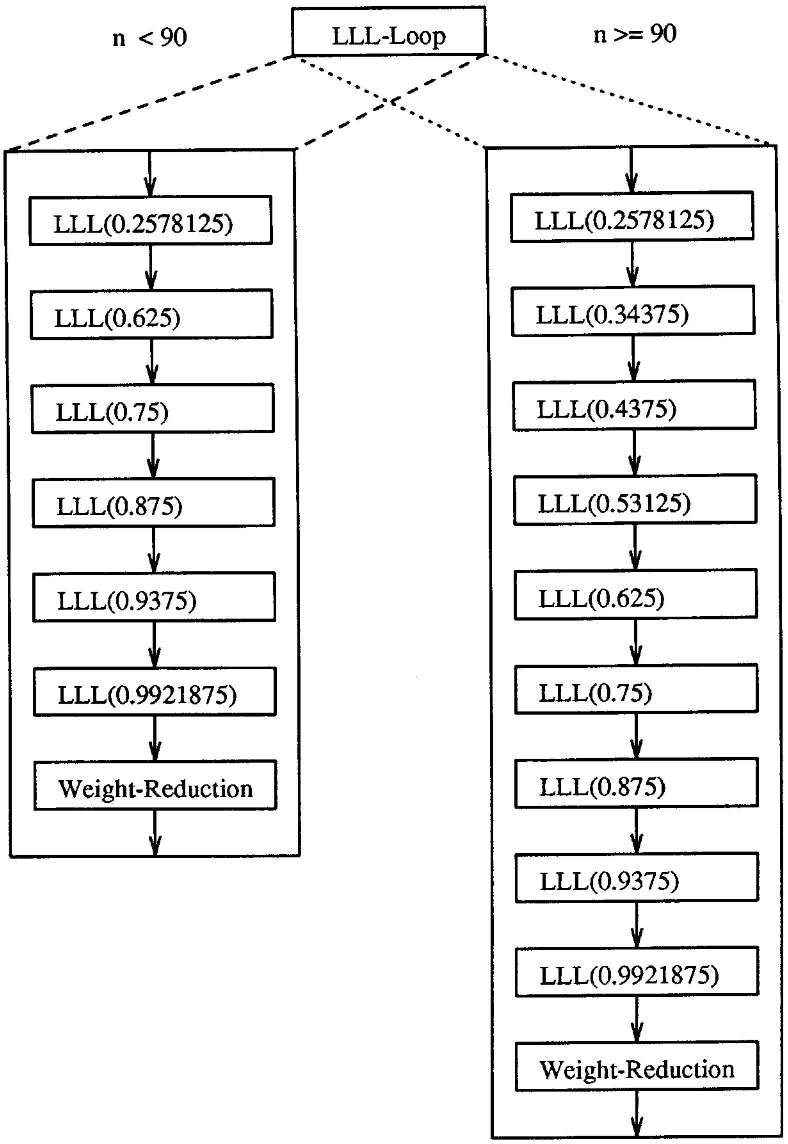

Figure 3-6: Algorithm **SL**: LLL-Loop Structure

algorithm ran might work just as well but require fewer row operations [18].

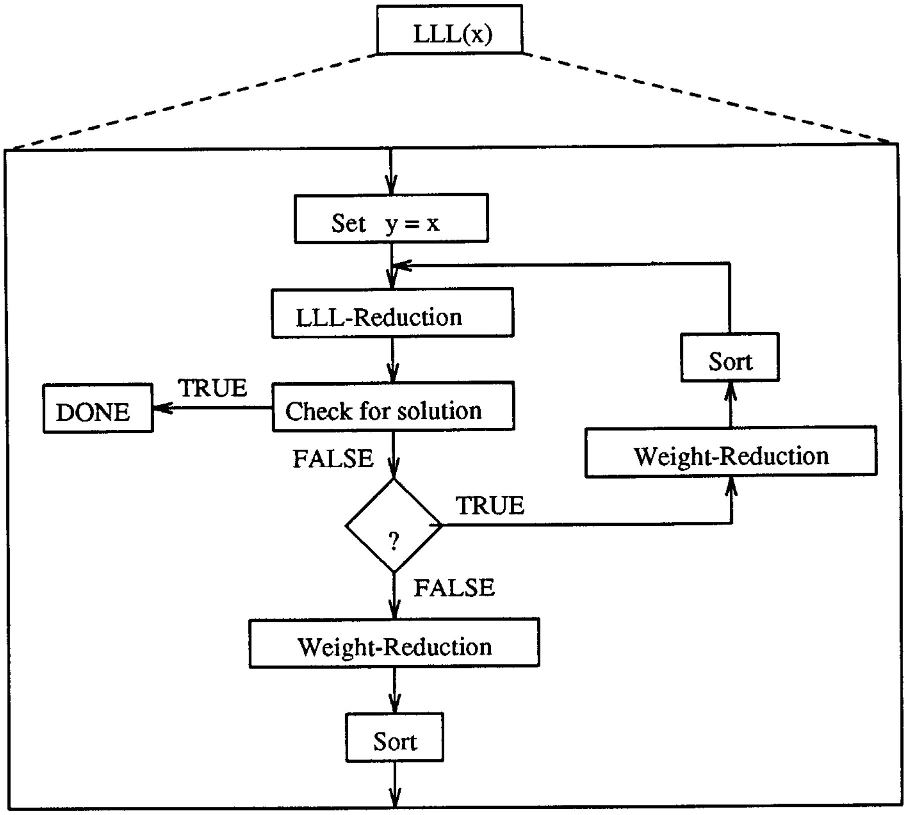

Figure 3-7: Algorithm **SL**: LLL(x) Internal Structure

Algorithm **SL** uses a version of the LLL algorithm which varies the parameter y among a finite number of values. Figure 3-6 shows what happens each time **SL** calls LLL-Loop. For subset sum problems with $n < 90$, each call to LLL passes through six stages (i.e. six distinct values for y)[1]. Upon entry to each stage, the y parameter is updated, and LLL-reduction is performed upon the lattice basis using the new y value (see Figure 3-7). Let $L_{y_0,\text{in}}$ and $L_{y_0,\text{out}}$ represent respectively the lattice bases input to and output from an application of the LLL algorithm with $y = y_0$. We now

[1] Three additional stages are used for subset sum problems with $n \geq 90$ to help reduce error.

compute the boolean value of the following expression:

$$\begin{aligned} &(\ \min\{\|\mathbf{b_i}\| \in L_{y_0,\mathrm{out}}\} < \min\{\|\mathbf{b_i}\| \in L_{y_0,\mathrm{in}}\}) \\ &\quad \mathbf{OR}\,(\ (\min\{\|\mathbf{b_i}\| \in L_{y_0,\mathrm{out}}\} = \min\{\|\mathbf{b_i}\| \in L_{y_0,\mathrm{in}}\}) \\ &\qquad\quad \mathbf{AND}\ (\max\{\|\mathbf{b_i}\| \in L_{y_0,\mathrm{out}}\} < \max\{\|\mathbf{b_i}\| \in L_{y_0,\mathrm{in}}\}))\,. \end{aligned} \tag{3.7}$$

(This decision point is represented by the diamond in Figure 3-7.) If the boolean value of Equation 3.7 is **TRUE**, then the *Weight-Reduction* procedure is applied, the *Sort* procedure is invoked, and the output is input again into LLL with $y = y_0$. Thus, we perform LLL-*Weight-Reduction-Sort* loops until either the length of the shortest vector in the lattice has increased after LLL-reduction, or if the length of the shortest vector has remained constant and the length of the longest vector has not decreased[2].

By using the recursive structure in Figure 3-7 and the termination condition represented by Equation 3.7, we attempt to have the LLL algorithm perform as many reduction steps as possible with small values of y_0. We delay increasing the value of y until LLL fails to make any improvement in the length of the largest vector in the lattice basis. In this way we restrict the set of row swaps and moves which LLL will consider, which improves the running time of the algorithm. Initially, for y small (say $y < 0.75$) LLL will only consider row moves which will "significantly" reduce the current lattice basis. Later, for values of $y \approx 1$, LLL will consider any row move which reduces the lattice.

For $n < 90$, six specific y values are used: 0.2578125, 0.625, 0.75, 0.875, 0.9375, and 0.9921875. These values were chosen for two reasons. First, all of the numbers have exact fractional binary representations in double-precision floating point. This meant that error would not be introduced into the LLL algorithm by performing arithmetic operations on y. Second, experiments with small subset sum problems ($n \leq 24$) showed that the number of row moves performed by LLL for each of the four middle values were approximately equal. That is, work was evenly distributed across

[2] The loop termination condition in Equation 3.7 is strictly a heuristic. Other terminating conditions could certainly be used.

all intermediate stages represented in Figure 3-6. (The endpoint values 0.2578125 and 0.9921875 were fixed). For $n \geq 90$, three additional values were used: 0.34375, 0.4375, and 0.53125. These values evenly divide the interval $[0.2578124, 0.625]$ into four equal pieces. As discussed in Section 3.5 below, our initial implementation of the LLL algorithm did not work correctly for lattice bases with $n \geq 90$ because of rounding error introduced into the computations. We were able to reduce the effects of rounding error to acceptable levels by modifying the LLL algorithm itself and by introducing the 0.34375, 0.4375 and 0.53125 stages into the algorithm.

We have detailed the operation of Algorithm **SL**, a combination of the Seysen and the Lenstra-Lenstra-Lovász basis reduction algorithms which also utilizes the *GCD-Reduction*, *Half-Vector*, *Weight-Reduction*, and *Sort* heuristics. In the next section we present the results of our experiments with this algorithm and show how the combination of these techniques greatly increases the range of subset sum problems which may be solved.

3.5 Empirical Tests Using Algorithm SL

We now present the results of our experimental attempts to solve subset sum problems using Algorithm **SL** as describes in Section 3.4 above. Following the tabulated results of Radziszowski and Kreher, we attempted to solve random subset sum problems of size n, where:

$$n \in \{42, 50, 58, 66, 74, 82, 90, 98, 106\}.$$

For each value of n, a set of b values (representing the number of bits in the binary representation of the weights a_i) was chosen. Algorithm **SL** was then run on a number of randomly generated subset sum problems for each pair (n, b). Where possible, values of b were chosen to coincide with values used in [26, 33].

Algorithm **SL** was implemented in FORTRAN and utilized Bailey's multiprecision floating-point package [4]. The Seysen reduction algorithm stored all lattice bases in multiprecision floating point and used both multiprecision and double-precision floating point operations. The size of the multiprecision floating-point representation was changed for each value of n so that b was at most one-half the size of the representation in bits. In many cases significantly more bits were available in the representation to help reduce rounding error.

The LLL portion of Algorithm **SL** was also implemented in FORTRAN but used only integer and double precision operations. One advantage of the Seysen-LLL structure of Algorithm **SL** is that by the time the LLL stage is reached, the lattice basis to be reduced contains only relatively small integers. In all cases investigated, the input lattice basis to the LLL phase of the algorithm did not contain any single coefficient larger than 2^{16}. This meant that we could safely store the basis vector coefficients as 32-bit integers, and we could use integer and double-precision floating point to carry out all calculations necessary to run LLL. Avoiding multiprecision row moves during the LLL phase greatly decreases the execution time of Algorithm **SL**.

For values of $n < 90$, these implementations of the Seysen and LLL algorithms were sufficient for our purposes. However, for $n \in \{90, 98, 106\}$, rounding error started to significantly alter the operation of the algorithm. Recall that the LLL algorithm stores the values of the $\mu_{i,j}$ coefficients as rational numbers; our implementation of LLL used double-precision floating point approximations to decrease the running time of the algorithm. Since such an approximation will introduce error, code was included to detect and attempt to correct errors resulting from rounding in double-precision calculations. In addition, while not changing the operation of the LLL algorithm, we performed a change of variables to reduce the rounding error which is necessarily introduced. LLL as defined in [27] uses two arrays of variables to hold information about the lattice being reduced:

$$\mu_{i,j} = \frac{(\mathrm{b_i}, \mathrm{b_j^*})}{(\mathrm{b_i}, \mathrm{b_i})},$$

$$B_i = \|\mathrm{b_i^*}\| = (\mathrm{b_i^*}, \mathrm{b_i^*}).$$

(Note that the $\mathbf{b}_i^*$ are orthogonalized versions of the $\mathbf{b}_i$ in LLL, not the basis vectors of the dual lattice as in Seysen's reduction algorithm.) Our version of LLL used three arrays of variables to maintain the same information:

$$\begin{aligned} m_{i,j} &= \frac{(\mathbf{b}_i, \mathbf{b}_j^*)}{\|\mathbf{b}_i\| \cdot \|\mathbf{b}_j^*\|} &&= \frac{\|\mathbf{b}_j^*\|}{\|\mathbf{b}_i\|}\mu_{i,j} \\ c_i &= \|\mathbf{b}_i^*\| &&= \sqrt{B_i} \\ bb_i &= (\mathbf{b}_i, \mathbf{b}_i) \end{aligned}$$

It is not difficult to transform the LLL relations between values of $\mu_{i,j}$ and B_i into equations involving $m_{i,j}, c_i$ and bb_i. The advantage gained is that:

$$\begin{aligned} m_{i,j} &= \frac{(\mathbf{b}_i, \mathbf{b}_j^*)}{\|\mathbf{b}_i\| \cdot \|\mathbf{b}_j^*\|}, \\ &= \frac{\|\mathbf{b}_i\| \cdot \|\mathbf{b}_j^*\| \cdot \cos\alpha}{\|\mathbf{b}_i\| \cdot \|\mathbf{b}_j^*\|}, \\ &= \cos\alpha, \end{aligned}$$

where α is the angle between the vectors $\mathbf{b}_i$ and $\mathbf{b}_j^*$. In the original version of LLL,

$$\mu_{i,j} = \frac{\|\mathbf{b}_i\|}{\|\mathbf{b}_j^*\|} m_{i,j},$$

and thus could take on a wide range of values, depending on the ratio of the lengths of $\mathbf{b}_i$ and $\mathbf{b}_j^*$. Limiting $m_{i,j} \in [-1, 1]$ and keeping the length information separate in c_i and bb_i reduces the error introduced by double-precision arithmetic operations. (Notice that bb_i values are integers and can be regenerated at will from the integer basis vectors.) Of course, with sufficient multiprecision representation for all variables there is no need for this transformation. It is simply an implementation-specific modification which allowed us to use a fast, double-precision version of LLL to reduce the lattices which arose while solving 106-element subset sum problems.

Tables 3.1 through 3.3 show the results of experiments performed on random subset sum problems with $n \leq 106$. For $n \leq 74$ twenty subset sum problems were

Table 3.1: Test Results Using Algorithm **SL** for $42 \leq n \leq 58$

n	b	$d = n/b$	RM_{S}	RM_{LLL}	TL	$\mathrm{S}_5(\mathrm{S}_1)$	% Solved	Average Time
20 trials for each length b for each n								
42	42	1.000	36907	420357	21	20 (19)	1.00	98.39
42	44	0.955	38955	276031	20	20 (20)	1.00	94.67
42	46	0.913	40264	362848	21	20 (19)	1.00	99.88
42	48	0.875	42288	234513	20	20 (20)	1.00	97.65
42	50	0.840	43826	222350	20	20 (20)	1.00	99.67
42	52	0.808	45144	178153	20	20 (20)	1.00	100.29
42	54	0.778	46405	209717	20	20 (20)	1.00	102.56
50	50	1.000	55122	3206204	42	20 (9)	1.00	283.93
50	52	0.962	56243	2672637	36	19 (10)	0.95	262.24
50	54	0.926	58249	2112480	31	20 (14)	1.00	243.22
50	56	0.893	59721	1596051	26	20 (16)	1.00	224.80
50	58	0.862	61594	1098460	21	20 (19)	1.00	208.40
50	60	0.833	63604	1220857	22	20 (18)	1.00	217.68
50	62	0.806	65563	978497	21	20 (19)	1.00	210.71
58	58	1.000	73587	15664277	83	7 (1)	0.35	808.68
58	60	0.967	77371	13477268	70	10 (5)	0.50	744.32
58	63	0.921	80574	13869151	74	13 (2)	0.65	837.96
58	66	0.879	82284	12949088	67	12 (5)	0.60	803.07
58	69	0.841	87569	7581281	43	19 (10)	0.95	594.55
58	72	0.806	90773	6234735	38	20 (10)	1.00	548.75
58	75	0.773	93896	4408841	29	20 (14)	1.00	480.70
58	78	0.744	98289	3411933	23	20 (18)	1.00	450.66
58	81	0.716	101104	3572104	25	20 (15)	1.00	463.48
58	84	0.690	105325	2090306	20	20 (20)	1.00	412.65
58	87	0.667	107711	1792670	20	20 (20)	1.00	406.44

attempted per b value. Only ten such problems were attempted per b value for $82 \leq n \leq 106$. For each problem, $\frac{1}{2}n$ elements were chosen from among the weights $a_1, \ldots, a_n$ to be in the subset. During the LLL phase of the algorithm, parameter $\pi_1 = 5$ and $\pi_2 = 8$. Six distinct values of y were used for $n < 90$; nine were used for $n \geq 90$.

All tests were run on Silicon Graphics 4D-220 workstations with four or eight MIPS Computers, Inc., R3000 processors per workstation. Each workstation was equipped with (32Mbytes· # processors) of main memory. As Algorithm **SL** requires significantly less than 32Mbytes of memory per process, all processors in a given workstation could be used simultaneously to work on different subset sum problems. The majority of the R3000 processors ran at 33MHz; the remainder operated with a clock frequency of 25MHz. All running times reported in Tables 3.1 through 3.3 have been adjusted to reflect the running time on a 33MHz processor.

The columns in Tables 3.1 through 3.3 show the value of the following variables for each (n, b) pair:

- n: The number of elements in the set of weights $(a_1, \ldots, a_n)$. Exactly one-half of these elements were chosen to form the subset sum s.

- b: The number of bits in the binary representation of the a_i's. Each a_i was generated randomly by choosing b random $0 - 1$ variables and concatenating the bits.

- d: The density of this class of subset sum problems. $d = n/b$.

- RM_{S}: The number of row moves performed by the Seysen phase of Algorithm **SL** on all trials.

- RM_{LLL}: The number of row moves perform by the LLL phase of Algorithm **SL** on all trials.

- TL: The total number of lattices reduced by LLL during all attempted trials, where we consider each randomization of the input lattice to be a new lattice. Thus, for 20 trials, $20 \leq TL \leq 20\pi_1 = 100$.
- S_5: The number of subset sum problems successfully solved with $\pi_1 = 5$.
- S_1: The number of subset sum problems successfully solved with $\pi_1 = 1$. This number is useful when comparing the results of Algorithm **SL** to the Radziszowski-Kreher method.
- % Solved: The success rate for $\pi_1 = 5$ measured as a fraction of the number of attempted problems.
- Average Time: The average amount of CPU time (in seconds) required to run Algorithm **SL** on a single trial, adjusted to reflect a 33MHz R3000 processor.

It should be noted that the running times for Algorithm **SL**, while an improvement over the Radziszowski-Kreher method, could be improved. The code for **SL** was not tuned to the Silicon Graphics machine (although Bailey's multiprecision code was written to work with SGI workstations). Careful recoding and tuning of crucial subroutines could probably yield significant improvements without changing the overall operation of the algorithm.

A quick look at the results summarized in Table 3.1 shows that Algorithm **SL** greatly improves upon the performance of previous methods for small values of n ($n \leq 58$). In [33] Radziszowski and Kreher were able to solve all attempted subset sum problems for $(n, d) = (42, 68)$, but only managed to solve 30% of the $(42, 54)$ problems. Compare this result to **SL**, which was able to solve all attempted subset sum problems with $n = 42$ and density $d \leq 1.0$. In fact, **SL** appears to work as well as a lattice oracle for all $n \leq 50$. The algorithm first shows signs of degradation at $n = 58$, where a 100% success rate was not reached until $b = 72$. A true lattice oracle (using the Euclidean norm) should have been able to achieve 100% success at $b = 63$, where the density $d = 0.926$ is less than the critical 0.9408 bound. However, **SL** was still able to solve over half of the attempted subset sum problems at $d = 0.962$; in [33] the 0.50 success rate was not reached until $d \approx 0.56$.

Columns four and five of the Table 3.1 show the number of row moves performed by the Seysen and LLL phases of **SL**. These numbers are a good first-order approximation of the amount of work performed by each phase, although it should be pointed out that there is no direct correspondence between the absolute magnitudes of the numbers. For fixed n, as b increases the Seysen phase performs more reduction steps on the lattice basis, and less work is required by the LLL phase to find the solution to the subset sum problem. As the density of the system decreases, Seysen's reduction algorithm comes closer to finding the desired solution vector. Indeed, in ten test cases with $(n, d) = (24, 24)$, the LLL phase was able to find solutions using only the $y = 0.2578125$ and $y = 0.625$ reductions. For higher densities of subset sum problems, the Seysen phase of **SL** reached a local minimum earlier, and LLL had a greater reduction to perform to find the solution. Note that once a Seysen reduction reached a local minimum it was considered finished; no attempt was made to "kick" or "heat" the reduced basis and then reapply Seysen. Application of one or more of the techniques described in Section 2.3 might yield better results, and is certainly a topic which should be investigated in the future.

Although Algorithm **SL** works exceptionally well for small values of n, its performance degrades quickly with respect to n once $n \geq 60$. Table 3.2 shows the performance of **SL** for n in the range $[66, 82]$. At $n = 66$, **SL** did not obtain a 100% success rate until the density dropped to $d = 0.66$, although the fact that **SL** solved 19 out of 20 attempts at $b = 92$ suggests that incrementing π_1 and/or π_2 might yield a 100% success rate at $d = 0.717$. If we consider only successes on the first attempt, **SL** reached the 100% level at $b = 112$, compared to $b = 144$ for Radziszowski and Kreher. (Note too that the method in [33] uses $\pi_1 = 1, 9 \leq \pi_2 \leq 15$, whereas in these experiments $\pi_2 = 8$ for **SL**.) Similar improvements may be seen for the cases where

Table 3.2: Test Results Using Algorithm **SL** for $66 \le n \le 82$

n	b	$d = n/b$	RM_S	RM_{LLL}	TL	$S_5(S_1)$	% Solved	Average Time
20 trials for each length b for each n								
66	76	0.868	109671	36111021	90	5 (0)	0.25	1774.39
66	80	0.825	116604	33233636	84	7 (1)	0.35	1670.55
66	84	0.786	120172	23212075	60	16 (5)	0.80	1465.55
66	88	0.750	128015	24791753	63	17 (2)	0.85	1560.71
66	92	0.717	134563	17049844	45	19 (9)	0.95	1249.72
66	96	0.688	139469	14280958	40	19 (10)	0.95	1140.53
66	100	0.660	142870	10472664	30	20 (11)	1.00	993.57
66	104	0.635	148379	6026607	21	20 (19)	1.00	819.64
66	108	0.611	155150	5688732	22	20 (18)	1.00	825.37
66	112	0.589	162192	4621556	20	20 (20)	1.00	803.30
74	106	0.698	175993	67316342	85	8 (2)	0.40	3416.51
74	112	0.661	183540	55347194	71	12 (4)	0.60	3095.85
74	118	0.627	189322	51147665	66	13 (5)	0.65	2850.97
74	124	0.597	202651	23245962	33	19 (13)	0.95	1852.46
74	130	0.569	211750	17077901	25	20 (16)	1.00	1600.57
10 trials for each length b for each n								
82	134	0.612	122353	74911469	50	0 (0)	0.00	8499.82
82	140	0.586	126724	54569482	37	5 (1)	0.50	6144.08
82	146	0.562	132948	63474238	42	5 (0)	0.50	6315.46
82	152	0.539	139374	22873751	17	10 (5)	1.00	3375.16
82	158	0.519	144353	18199583	14	10 (7)	1.00	3154.05

$n = 74$ and $n = 82$. Although at $n = 82$ Algorithm **SL** did not attain a success rate of 1.00 until the density had decreased to $d = 0.539$, similar results were not reported in [33] until d had dropped to below 0.39.

Table 3.3 shows the performance of **SL** on subset sum problems with $n \approx 100$. The density at which 100% a success rate is reached is steadily and significantly decreasing for small increases in n. By the time $n = 106$, **SL** is able to solve all attempted problems with density ≈ 0.35, but densities above 0.4 appear unobtainable.

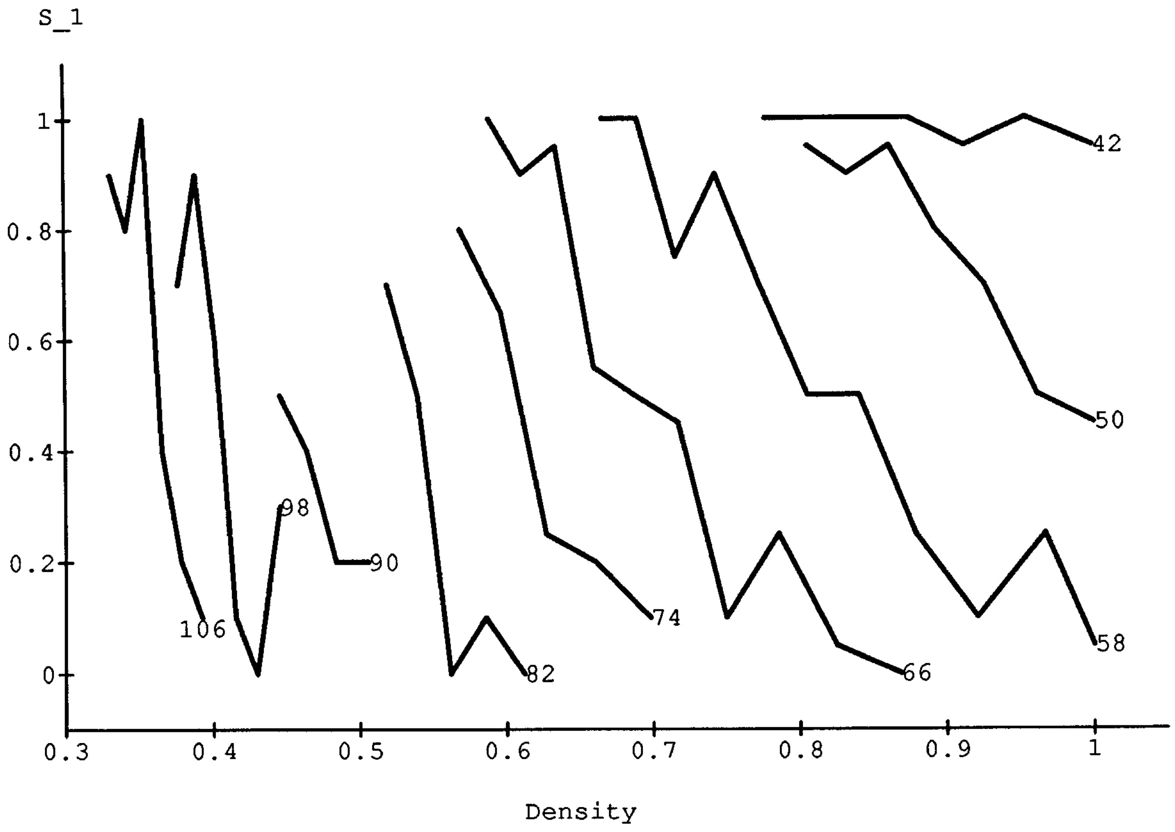

Figure 3-8: Algorithm **SL**: S_1 vs. Density

Figure 3-8 shows the success rate for solving subset sum problems on the first randomization (S_1) versus density for all the test cases described in Tables 3.1 through 3.3.

Table 3.3: Test Results Using Algorithm **SL** for $90 \leq n \leq 106$

n	b	$d = n/b$	RM_S	RM_{LLL}	TL	$S_5(S_1)$	% Solved	Average Time
10 trials for each length b for each n								
90	178	0.506	180739	111553580	40	3 (2)	0.30	14300.87
90	186	0.484	188560	88163144	32	8 (2)	0.80	11241.77
90	194	0.464	196090	67630320	25	8 (4)	0.80	9362.65
90	202	0.446	198909	48109448	19	10 (5)	1.00	7277.74
98	220	0.445	238948	177962307	38	3 (3)	0.30	21725.83
98	228	0.430	249432	187347238	39	6 (0)	0.60	21841.45
98	236	0.415	255213	125193960	28	9 (1)	0.90	17064.47
98	244	0.402	264404	74096833	17	10 (6)	1.00	12386.50
98	252	0.389	277285	39984465	11	10 (9)	1.00	9142.08
98	260	0.377	280876	42340342	13	10 (7)	1.00	8560.10
106	270	0.393	322902	265583840	37	5 (1)	0.50	34147.08
106	280	0.379	335435	177869835	26	9 (2)	0.90	25219.92
106	290	0.366	347679	175847501	26	8 (4)	0.80	26034.16
106	300	0.353	357841	43319759	10	10 (10)	1.00	12137.60
106	310	0.342	369215	49294587	12	10 (8)	1.00	13184.20
106	320	0.331	375548	39512954	11	10 (9)	1.00	11652.82

Compare the curves depicted in this figure to those for the Lagarias-Odlyzko (Figure 3-1) and the Radziszowski-Kreher (Figure 3-3) methods. Algorithm **SL** shows significant improvement over these previous methods and extends the "frontier" of solvable subset sum problems. Figure 3-9 shows the success rate versus density when **SL** was allowed to perform five attempted reductions (S_5). The improvement made by Algorithm **SL** over previous methods may clearly be seen by comparing this figure to those shown previously. The shift in the frontier caused by increasing π_1 from one to five is also apparent.

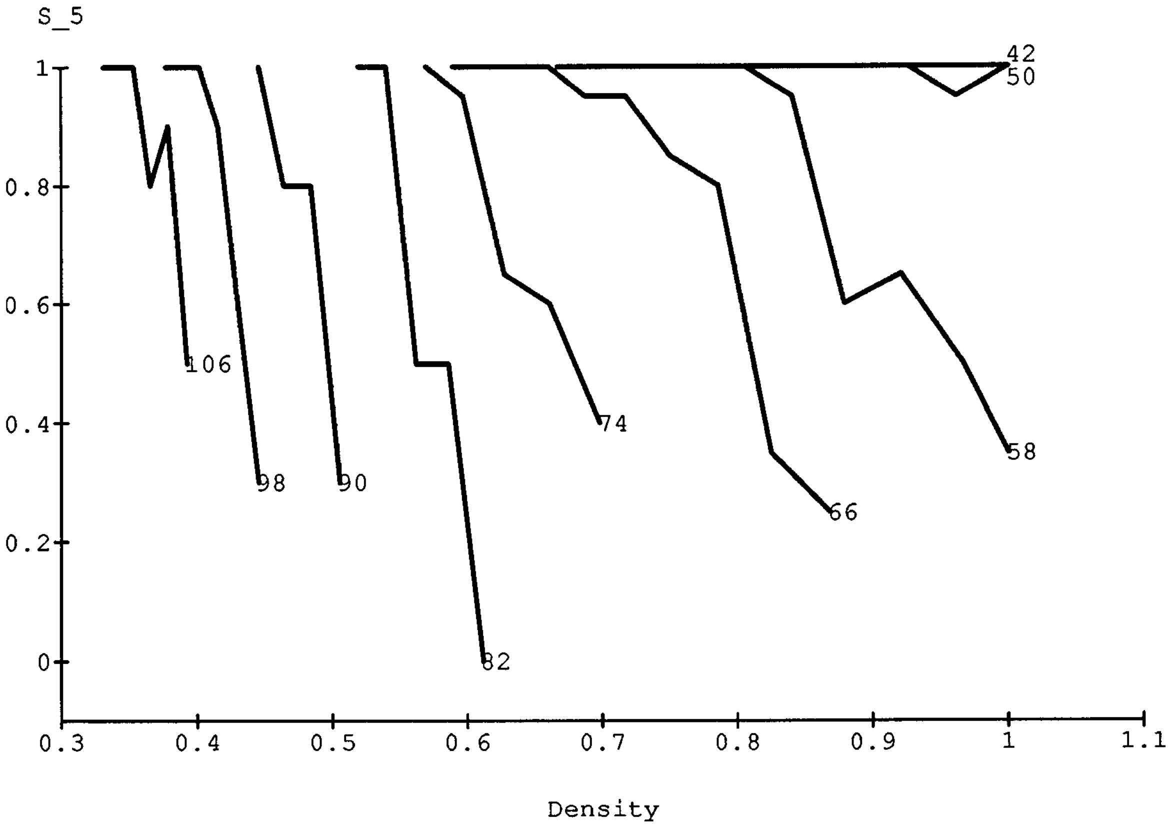

Figure 3-9: Algorithm **SL**: S_5 vs. Density

The performance of Algorithm **SL** is a vast improvement over the techniques used in [26, 33]. Not only is the success rate higher for Algorithm SL, but its improved running time allows larger-sized subset sum problems to be effectively attacked. Nevertheless, for $n \gtrsim 100$ Algorithm SL is still only able to attack subset sum problems with densities below 0.4. Subset sum problems with larger n and higher densities are unsolvable from a practical point of view.

Chapter 4

Conclusions

We have shown in this thesis that Seysen's lattice basis reduction algorithm performs much better than other currently available techniques in a limited number of circumstances. In particular, we have demonstrated that Seysen's algorithm may be combined with the LLL algorithm to solve subset sum problems. As a general lattice basis reduction tool, however, Seysen's algorithm leaves much to be desired. The lack of theoretical bounds on the running time and performance of the algorithm is discouraging. Empirical tests performed using Seysen's algorithm also highlight weaknesses with using this method to solve general lattice reduction problems. As the first part of this thesis demonstrated, the performance of Seysen's algorithm on randomly generated "dense" lattice bases degrades quickly as the dimension of the lattice increases above a certain critical bound. There are numerous local minima of the $S(A)$ function for these lattices, and once Seysen's algorithm encounters one it is unable to escape (without some external influence acting upon the lattice basis).

Seysen's algorithm should not be immediately discounted, however. In certain specific cases the algorithm performed quite well. In general Seysen's algorithm performs only a fraction of the row moves required by the LLL algorithm to reduce a lattice basis. For certain lattices, such as the B_θ lattices of Section 2.2.3, the LL-L algorithm was unable to perform any row reductions, whereas Seysen's algorithm enjoyed great success. Even for dense lattices, if the dimension of the lattice is relatively small, Seysen's algorithm obtains the same results as LLL in only a fraction of the time. Applications which utilize basis reduction in few dimensions are good candidates for Seysen's method.

The experiments performed on lattices derived from subset sum problems highlight one of the main advantages of Seysen's technique: its ability to very quickly perform significant reductions on a lattice basis. Using Seysen's algorithm as a first reduction stage allowed us to convert lattice bases involving large, multiprecision values into other bases with vector coefficients which could be represented in single- or double-precision. The LLL algorithm could then be run without having to use multiprecision arithmetic, which greatly improved its execution time. For larger lattices (with $n \approx 100$) one must be aware of possible problems due to rounding and truncation errors, but these difficulties can be overcome.

Although the primary goal of this thesis was to investigate applications of Seysen's basis reduction algorithm, one should not overlook the other techniques which were used (in addition to Seysen's algorithm) to solve subset sum problems. In particular, the theoretical improvements to the Lagarias-Odlyzko attack in [12] may be directly incorporated into practical methods of solving subset sum problems. Also, the use of multiple values of the y parameter in the LLL algorithm significantly reduced the total number of row moves LLL performed. This modification does not appear to decrease the reduction performance of LLL in any way over LLL with constant $y \approx 0.99$, although it does significantly reduce the running time of the algorithm. We would suggest using multiple, increasing values of y in the future whenever LLL-reduction is performed.

We have demonstrated that Seysen's algorithm is a good basis reduction technique for certain types of lattices, outperforming the basic LLL algorithm in terms of the number of row operations required. Furthermore, we have seen how Seysen's algorithm may be combined with variants of the LLL algorithm and heuristic methods to successfully attack many subset sum problems with $n \lesssim 100$. Yet these specific instances are but a fraction of the type of lattice reduction problems which arise. It is therefore natural to consider what other types of lattice reduction problems are suitable for attack by Seysen's algorithm. We conclude this chapter, and the thesis itself, with some remarks on other Seysen-suitable lattices, and also suggestions for modifying Algorithm **SL** to solve subset sum problems of larger dimensions and higher densities.

4.1 Candidate Lattices for Seysen Reduction

The number of "classes" of lattice bases which may be successfully reduced using only Seysen's basis reduction algorithm appears to be quite small. It was shown in Chapter 2 that Seysen's algorithm, which was designed for finding simultaneously good reductions of a lattice basis and its dual, indeed works well in such cases. Furthermore, if the goal of a lattice reduction is to minimize some cost function or measure of the lattice (and perhaps its dual), an appropriately modified version of Seysen's algorithm incorporating the cost function will likely perform much better than an LLL-type algorithm. While there are empirical reasons which suggest the use of the $S(A)$ measure as a particular cost function, Seysen himself mentioned in [38] that other function had been used in the algorithm in place of $S(A)$ with about the same degree of success.

Replacing $S(A)$ with another cost function may involve some difficulty. In particular, it may not be possible to solve for the coefficients λ_i in closed form using an alternate cost function. If some range of acceptable values can be determined, then a search considering all integers in the range of interest may be feasible. Such an approach was used in experiments with the cost function

$$\begin{aligned} S'(A) &= \sum_{i=1}^{n} \frac{1}{\sqrt{a_{i,i}a^*_{i,i}}}, \\ &= \sum_{i=1}^{n} \frac{1}{\|\mathbf{b_i}\|\|\mathbf{b_i^*}\|}. \end{aligned}$$

For a proposed row move $\mathbf{b_j} \leftarrow \mathbf{b_j} + \lambda\mathbf{b_i}$ it is possible to compute bounds on the range of possible λ values. For sufficiently small ranges $\lambda \in [\lambda_1, \lambda_2]$ we compute the change in $S'(A)$ after applying the row move with λ taking on every integer value in the range. Then (assuming a greedy approach) we choose the value for λ which maximized the decrease in $S'(A)$. Thus, although it may not be possible to solve for the best choice for λ for some proposed cost function, numerical methods may allow consideration of the metric anyway.

The class of lattice bases which may be successfully reduced by using only Seysen's algorithm appears quite limited, assuming that we ignore differences in execution time. The randomly generated lattice bases with unit determinant from Chapter 2 exemplify this point: Seysen's algorithm began to perform poorly for $n \geq 35$, whereas the LLL algorithm continued to correctly reduce lattices to the n-dimensional cubic lattice for all tested cases. As a stand-alone technique, Seysen's algorithm may not be considered that useful; it works quite well for some lattices of low dimension, but it tends to stop early for larger-dimensioned lattice bases. To reduce these lattice bases some combination of Seysen's algorithm and other reduction methods is probably required.

4.2 Modifying Algorithm SL

The combination of the Seysen and LLL reduction algorithms used in Chapter 3 to solve subset sum problems is a significant improvement over previously tried techniques. There is still plenty of room for improvement. The performance of Algorithm **SL** declines steadily over the range $50 \leq n \leq 100$. For subset sum problems with

$n \approx 100$, Algorithm **SL** had difficulty solving instances with density $d \approx 0.4$, well below the $0.9408\ldots$ theoretical density bound.

There are a number of ways in which Algorithm **SL** could be extended in order to attack subset sum problems with larger n, d values. For example, any of the techniques mentioned in Section 2.3 above could be applied to the Seysen phases of the algorithm. We can clearly imagine that another measure function $S'(A)$ might perform better for sparse, subset-sum generated lattices when compared to the performance of $S(A) = \sum_{i=1}^{n} \|\mathbf{b}_i\|^2 \|\mathbf{b}_i^*\|^2$.

Another method of extending Algorithm **SL** would be to increase the values of the π_1 and π_2 parameters. We have seen that increasing π_1 (the number of different orderings of the initial LLL lattice basis vectors) from one to five yields markedly better performance. Along similar lines, one could also allow more LLL-*Weight-Reduction-Sort* loops to occur for each randomization. Recall that for Algorithm **SL**, π_2, the maximum number of loops, was set to eight. After eight iterations without finding the desired solution vector in the lattice basis, Algorithm **SL** would "give up" and select a new randomization of the LLL-Phase input lattice basis. The choice to set $\pi_2 = 8$ was made arbitrarily; Radziszowski and Kreher used $\pi_2 = 15$ in their experiments. The majority of the work performed during the LLL-Phase of Algorithm **SL** occurs during the first LLL-Loop on each of the π_1 iterations; the second through π_2^{th} LLL-Loops perform only a fraction of the number of row moves performed by the first loop. In our experiments, subsequent LLL-Loops tended to perform only $\frac{1}{10}$ the row operations performed in the initial LLL-*Weight-Reduction-Sort* iteration. This means that π_2 could probably be increased to around 20 and the overall running time of the LLL-Phase would probably double.

Increasing the number of LLL-Loops appears to be an inexpensive way to allow Algorithm **SL** to perform more reduction operations on a lattice basis. One must consider, however, how much an increase in π_2 will improve the rate at which Algorithm **SL** solves subset sum problems. We know from [33] that Radziszowski and

Kreher were able to improve upon the Lagarias-Odlyzko algorithm by significantly increasing π_2. Furthermore, in some very recent experiments Euchner and Schnorr (personal communication) were able to obtain performance close to that of Algorithm **SL** with $\pi_1 = 1$ and $\pi_2 = 16$. The Euchner-Schnorr reduction algorithm utilizes some other heuristics not included in Algorithm **SL** but uses only the LLL algorithm as the main reduction technique. To date, Euchner and Schnorr have only reported on experiments involving subset sum problems of size $n \leq 66$; the performance of their technique on lattice bases in higher dimensions is unknown. Even their preliminary results, though, suggest that it might be worthwhile to increase the number of reduction stages in Algorithm **SL**, even if it is necessary to reduce π_1 in order to maintain comparable running times. Also, as their methods uses only the LLL algorithm, a combination of their techniques and Algorithm **SL** is certainly possible.

Another factor to consider in Algorithm **SL** is the structure of our version of LLL. Recall that instead of running LLL with $y \lesssim 1.0$ we used a fixed number of LLL stages with varying y values. The first stage used a value of y slightly greater than $\frac{1}{4}$, and subsequent stages used larger and larger values of y. For the first stage of the π_2 reduction stages, the multiple-value version of LLL significantly reduced the overall running time of the algorithm. However, later reduction stages were generally unable to benefit from this structure; in fact, in many cases the lattice was reduced further only by the stages with $y \geq 0.875$. Future implementations might consider removing the LLL stages with small y values for the second through $(\pi_2)^{\text{th}}$ reduction loops.

In Section 2.3.3 we discussed how Hadamard matrices could be used to randomly permute a lattice basis which was locally Seysen-reduced in the hope that reapplying Seysen's algorithm would yield a better reduced lattice basis. This same technique could be applied to both the Seysen and LLL phases of Algorithm **SL** and might yield significant improvements. Hadamard matrices could be used to permute a Seysen-reduced lattice basis whose Seysen measure is a local minimum of the $S(A)$ function. Such a permutation might then permit Seysen's algorithm to reach another lattice basis with smaller $S(A)$ value, which in turn could increase the ratio of work performed by the Seysen phase to that performed by the LLL phase. Similarly, recall that under the current scheme once a lattice basis has been LLL-reduced π_2 times without yielding a solution vector, that basis is forgotten and a new iteration is started using a random permutation of the output of the Seysen phase. Instead of "throwing away" the LLL-reduced basis, which has shorter basis vectors than the Seysen-reduced basis, a Hadamard permutation could be applied to the output of the π_2 stages. This permuted basis would then be used as the input basis to the next big iteration. Assuming that the Hadamard permutation method sufficiently "scrambles" the lattice basis, we could thus avoid the overhead of having to reduce the output of the Seysen phase more than once.

Finally, it is possible to modify Algorithm **SL** to take into account additional information related to the creation of the specific subset sum problem. In [12] it is shown how to tailor the input lattice basis for a subset sum problem if it is known that the number of elements in the desired subset of weights is bounded by a fraction of n. That is, if it is known that

$$\sum_{i=1}^{n} e_i = \beta n,$$

then the lattice basis can be modified so that a solution vector exists of length $\sqrt{n\beta(1-\beta)}$. (In the worst case, $\beta = \frac{1}{2}$ and the solution vector is the familiar $\hat{\mathbf{e}}'$ vector with $\|\hat{\mathbf{e}}'\| = \frac{1}{2}\sqrt{n}$.) For instances of the general subset sum problem no information is known concerning $\sum e_i$. Some knapsack cryptosystem, such as the Chor-Rivest system [11], do use subsets with relatively few weights. When attacking such systems, Algorithm **SL** should be modified to use the tailored lattice basis described in [12].

Bibliography

[1] T. M. Apostol, *Calculus, Vol. II*, New York: John Wiley & Sons (1969).

[2] T. M. Apostol, *Modular Functions and Dirichlet Series in Number Theory*, *Graduate Texts in Mathematics* **41**, New York: Springer-Verlag (1976).

[3] D. H. Bailey, Comparison of two new integer relation algorithms, manuscript in preparation.

[4] D. H. Bailey, MPFUN: A Portable High Performance Multiprecision Package, NASA Ames Research Center, preprint.

[5] P. van Emde Boas, Another NP-complete partition problem and the complexity of computing short vectors in a lattice, Rept. 81-04, Dept. of Mathematics, Univ. of Amsterdam, 1981.

[6] E. F. Brickell, Solving low density knapsacks, *Advances in Cryptology, Proceedings of Crypto '83*, Plenum Press, New York (1984), 25-37.

[7] E. F. Brickell, The cryptanalysis of knapsack cryptosystems. *Applications of Discrete Mathematics*, R. D. Ringeisen and F. S. Roberts, eds., SIAM (1988), 3-23.

[8] E. F. Brickell and A. M. Odlyzko, Cryptanalysis: a survey of recent results, *Proc. IEEE* **76** (1988), 578-593.

[9] Brun, Algorithmes euclidiens pour trois et quatre nombres, 13eme Congr. Math. Scand., Helsinki, 45-64.

[10] V. Cerny, A thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm, *J. Optimization Theory and Appl.* **45**, 41-51.

[11] B. Chor and R. Rivest, A knapsack-type public key cryptosystem based on arithmetic in finite fields, *Advances in Cryptology: Proceedings of Crypto '84*, Springer-Verlag, NY (1985), 54-65. Revised version in *IEEE Trans. Information Theory* **IT-34** (1988), 901-909.

[12] M. J. Coster, B. A. LaMacchia, A. M. Odlyzko and C. P. Schnorr, An improved low-density subset sum algorithm, *Advances in Cryptology: Proceedings of Eurocrypt '91*, D. Davies, ed., to appear.

[13] Y. Desmedt, What happened with knapsack cryptographic schemes?, *Performance Limits in Communication, Theory and Practice*, J. K. Skwirzynski, ed., Kluwer (1988), 113-134.

[14] A. M. Frieze, On the Lagarias-Odlyzko algorithm for the subset sum problem, *SIAM J. Comput.* **15(2)** (May 1986), 536-539.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company (1979).

[16] J. Håstad, B. Just, J. C. Lagarias, and C. P. Schnorr, Polynomial time algorithms for finding integer relations among real numbers, *SIAM J. Comput.* **18(5)** (October 1989), 859-881.

[17] J. Håstad and J. C. Lagarias. Simultaneously good bases of a lattice and its reciprocal lattice, *Math. Ann.* **287** (1990), 163-174.

[18] D. He, Solving low-density subset sum problems with modified lattice basis reduction algorithm, Northwest Telecommunication Engineering Institute (Xi'an, China), preprint.

[19] A. Joux and J. Stern, Improving the critical density of the Lagarias-Odlyzko attack against subset sum problems, to be published.

[20] R. Kannan, Improved algorithms for integer programming and related lattice problems, *Proc.* 15th *Symp. Theory. of Comp.* (1983), 193-206.

[21] S. Kirkpatrick, C. D. Gelatt Jr. and M. P. Vecchi, Optimization by simulated annealing, *Science* **220** 671-680.

[22] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd ed., Addison-Wesley 1981.

[23] A. Korkin and G. Zolotarev, Sur les formes quadratiques, *Math. Ann* **6**, 366-389.

[24] A. Korkin and G. Zolotarev, Sur les formes quadratiques, *Math. Ann* **11**, 242-292.

[25] J. C. Lagarias, Point lattices, manuscript in preparation.

[26] J. C. Lagarias and A. M. Odlyzko, Solving low-density subset sum problems, *J. Assoc. Comp. Mach.* **32(1)** (January 1985), 229-246.

[27] A. K. Lenstra, H. W. Lenstra, and L. Lovász, Factoring polynomials with rational coefficients, *Math. Ann.* **261** (1982), 515-534.

[28] J. E. Mazo and A. M. Odlyzko, Lattice points in high-dimensional spheres, *Monatsh. Math.* **110** (1990), 47-61.

[29] H. Minkowski, Diskontinuitsbereich fur arithmetic Aquivalenz, *J. reine Angew.* **129**, 220-224.

[30] J. R. Munkres, *Analysis of Manifolds*, Redwood City: Addison-Wesley (1991).

[31] A. M. Odlyzko, The rise and fall of knapsack cryptosystems, *Cryptology and Computational Number Theory*, C. Pomerance, ed., Am. Math. Soc., Proc. Symp. Appl. Math. **42** (1990), 75-88.

[32] M. Pohst, A modification of the LLL reduction algorithm, *J. Symb. Comp.* **4** (1987), 123-127.

[33] S. Radziszowski and D. Kreher, Solving subset sum problems with the L^3 algorithm, *J. Combin. Math. Combin. Comput.* **3** (1988), 49-63.

[34] C. P. Schnorr, A hierarchy of polynomial time lattice basis reduction algorithms, *Theoretical Computer Science*, **53** (1987), 201-224.

[35] C. P. Schnorr, Factoring integers and computing discrete logarithms via diophantine approximation, *Advances in Cryptology: Proceedings of Eurocrypt '91*, D. Davies, ed., to appear.

[36] A. Schönhage, Factorization of univariate integer polynomials by diophantine approximation and by an improved basis reduction algorithm, 11th *International Colloquium on Automata, Languages and Programming (ICALP '84)*, J. Paredaens, ed., *Lecture Notes in Computer Science* **172**, Springer-Verlag, NY (1984), 436-447.

[37] J.-P. Serre, *A Course in Arithmetic*, *Graduate Texts in Mathematics* **7**, New York: Springer-Verlag (1973).

[38] M. Seysen, Simultaneous reduction of a lattice basis and its reciprocal basis, to be published.

CS-TR Scanning Project
Document Control Form

Date : 6 / 15 / 95

Report # AI-TR-1283

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
□ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) □ Technical Memo (TM)
□ Other:

Document Information

Number of pages: 108 (115-images)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
□ Double-sided

Intended to be printed as :
□ Single-sided or
☒ Double-sided

Print type:
□ Typewriter □ Offset Press ☒ Laser Print
□ InkJet Printer □ Unknown □ Other:

Check each if included with document:

☒ DOD Form □ Funding Agent Form ☒ Cover Page
☒ Spine □ Printers Notes □ Photo negatives
□ Other:

Page Data:

Blank Pages (by page number):

Photographs/Tonal Material (by page number):

Other (note description/page number):

Description : Page Number:

IMAGE MAP (1-2) UN#'ED TITLE PAGE & BLANK PAGE
(3-10) PAGES #'ED i-VIII
(11-108) PAGES #'ED 1-98
(109-112) SCAN CONTROL, COVER, SPINE (BACK OF COVER), DOD
(113-115) TRGT'S (3)

Scanning Agent Signoff:
Date Received: 6 / 15 / 95 Date Scanned: 6 / 16 / 95 Date Returned: 6 / 22 / 95

Scanning Agent Signature: Michael W. Cook

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
June 1991

3. REPORT TYPE AND DATES COVERED
technical report

4. TITLE AND SUBTITLE
Basis Reduction Algorithms and Subset Sum Problems

5. FUNDING NUMBERS
N00014-89-J-3202

6. AUTHOR(S)
Brian A. LaMacchia

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139

8. PERFORMING ORGANIZATION REPORT NUMBER
AI-TR 1283

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
Information systems
Arlington, Virginia 22217

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AD-A259492

11. SUPPLEMENTARY NOTES
None

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution of this document is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis investigates a new approach to lattice basis reduction suggested by M. Seysen. Seysen's algorithm attempts to globally reduce a lattice basis, whereas the Lenstra, Lenstra, Lovász (LLL) family of reduction algorithms concentrates on local reductions. We show that Seysen's algorithm is well suited for reducing certain classes of lattice bases, and often requires much less time in practice than the LLL algorithm. We also demonstrate how Seysen's algorithm for basis reduction may be applied to subset sum problems. Seysen's technique, used in combination with the LLL algorithm, and other heuristics, enables us to solve a much larger class of subset sum problems than was previously possible.

14. SUBJECT TERMS

subset sum problems	integer lattice
knapsack cryptosystems	Seysen's algorithm
public key cryptography	lattice basis reduction

15. NUMBER OF PAGES
110

16. PRICE CODE
$8.00

17. SECURITY CLASSIFICATION OF REPORT
unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
unclassified

20. LIMITATION OF ABSTRACT
unclassified