International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 5 (2013) pp. 317-325

Search Graph Formulation and Hasting's Generalization of Metropolis Algorithm For Solving SVP

Ajitha Shenoy K B[1], Somenath Biswas[2] and Piyush P Kurur[3]

[1,2,3]Department of Computer Science and Engineering,
Indian Institute of Technology, Kanpur, INDIA.
[1]*ajith@cse.iitk.ac.in,* [2]*sb@cse.iitk.ac.in,* [3]*ppk@cse.iitk.ac.in*

[1]Department of MCA, Manipal Institute of Technology,
Manipal University, Manipal, INDIA
ajith.shenoy@manipal.edu

Abstract: **Shortest Lattice Vector Problem (SVP) has numerous applications spanning from robotics to computational number theory, viz., polynomial factorization. At the same time, SVP is a notoriously hard problem. Not only it is NP-hard, there is not even any polynomial approximation known for the problem that runs in polynomial time. What one normally uses is the LLL algorithm which, although a polynomial time algorithm, may give solutions which are an exponential factor away from the optimum. In this paper, we have defined an appropriate search space for the problem which we use for implementation of the Hasting's generalization of the Metropolis algorithm. We have defined a suitable neighbourhood structure which makes the diameter of the space polynomially bounded, and we ensure that each search point has only polynomially many neighbours. We also proved that our search space graphs for SVP has magnification greater than half. We have implemented the Metropolis algorithm and Hasting's generalization of the Metropolis algorithm for the SVP. Our results are quite encouraging in all instances when compared with LLL algorithm.**

Keywords: SVP, Search Space, Metropolis Algorithm, Hasting's Generalization, LLL.

I. Introduction

We investigate in this paper the suitability of using the Metropolis algorithm to solve the shortest lattice vector problem, SVP, [1] for short. The Metropolis algorithm [2][3] is a widely used randomized search heuristic and is often used in practice to solve combinatorial optimization problems. It is known that the algorithm performs surprisingly well even for some provably hard problems; e.g, [3] showed that the Metropolis algorithm is efficient for random instances of the graph bisection problem. It is, therefore, of interest to investigate the performance of the algorithm for SVP, which is another hard problem of great interest, both from the theoretical and practice considerations.

Van Emde Boas [4] proved in 1981 that SVP is NP-hard for the ∞ norm and mentioned that the same should be true for any p norm. However, proving hardness in the 2 norm (or in any finite p norm) was an open problem for a long time. A breakthrough result by Ajtai[5] in 1998 finally showed that SVP is NP-hard under randomized reductions. Another breakthrough by Micciancio[6] in 2001 showed that SVP is hard to approximate within some constant factor, specifically for any factor less than $\sqrt{2}$. This was the best result known so far leaving a huge gap between the $\sqrt{2}$ hardness factor and the exponential approximation factors achieved by Lenstra et. al. [7] in 1982, Schnorr [8] in 1988, and Ajtai et. al. [9] in 2003. At the same time, there are many situations in practice which require us to get at least a good solution for SVP, because this is an essential step in most algorithms for factorizing polynomials.

The structure of the paper is as follows: in the following section, we define SVP, in Section 3 we give quick description of LLL algorithm [7], in Section 4 we define an appropriate search space for our approach to solve SVP, in Section 5 we show how we use our search space to implement the Metropolis and Hasting's generalization of the Metropolis algorithm, we also mention why the latter is more appropriate for our search space. Section 6 provides some experimental data on how our implementation compares with a standard implementation of the LLL algorithm. The paper ends with some concluding remarks.

II. Shortest Vector Problem (SVP)

Definition II.1 (Lattices)**.** *Let $B = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$ be a set of linearly independent vectors in m-dimensional Euclidean space $\mathbb{R}^m$ where $m \geq n$. The set $L(B)$ of all vectors $a_1\mathbf{b}_1 + \ldots + a_n\mathbf{b}_n$, a_i's varying over integers, is called the integer lattice, or simply, the lattice with basis B (or generated by B) and n is the dimension of $L(B)$. If $m = n$, we say that the lattice is of full dimension.*

In this paper, we consider only full dimensional lattices. Furthermore, we consider only lattices whose basis vectors have rational components. In such a case, we can clear the denominators and assume that the each of the basis element is a vector in $\mathbb{Z}$ instead of $\mathbb{R}$.

The basis can be compactly represented as an $n \times n$ matrix (also denoted as B) with columns being the basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ as its columns. Then we can write $L(B) = \{B\mathbf{a} : \mathbf{a} \in \mathbb{Z}^n\}$

Problem II.1 (Shortest Lattice Vector Problem (SVP))**.** *Given a lattice $L(B)$ contained in $\mathbb{Z}^n$ specified by linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$, the SVP problem is to find a shortest (in Euclidean norm) non-zero vector of $L(B)$.*

Without loss of generality, we consider the *decision (and the search) version* of the above problem: Given a a basis B as above, and a rational number $K > 0$, the problem is to decide if there is non-zero vector $\mathbf{v}$ that belongs to $L(B)$ such that $||\mathbf{v}|| < K$ where $||\mathbf{v}||$ denotes the Euclidean norm of $\mathbf{v}$, and if the answer is 'yes', output such a vector.

Clearly, SVP can be solved by logarithmically many applications of the decision version of the problem. In the next section we will give quick description of LLL algorithm [7] since we compare our algorithm with the existing LLL algorithm.

III. LLL Algorithm[7]

The LLL algorithm is a generalization of Gauss's algorithm to higher dimension. LLL is the well celebrated and extensively used polynomial time approximation algorithm for finding shortest lattice vector. To give quick description of LLL algorithm, we need the following definitions.

Definition III.1 (Gram-Schmidt orthogonalization process)**.** *Given n linearly independent vectors $b_1, b_2, \ldots, b_n \in \mathbf{R^n}$, the Gram-Schmidt orthogonalization of $b_1, b_2, \ldots, b_n$ is defined by $\widetilde{b_i} = b_i - \sum_{j=1}^{i-1} \mu_{i,j}\widetilde{b_j}$, where $\mu_{i,j} = \frac{<b_i, \widetilde{b_j}>}{<\widetilde{b_j}, \widetilde{b_j}>}$*

Definition III.2 (δ - LLL Reduced Basis [7])**.** *A basis $B = \{b_1, b_2, \ldots, b_n\} \in \mathbf{R^n}$ is a δ - LLL Reduced basis ($\frac{1}{4} < \delta < 1$) if the following holds :*

1. $\forall 1 \leq i \leq n$, $j < i$, $\mu_{i,j} \leq \frac{1}{2}$ 2. $\forall 1 \leq i < n$, $\delta \left\|\widetilde{b_i}\right\|^2 \leq \left\|\mu_{i+1,i}\widetilde{b_i} + \widetilde{b}_{i+1}\right\|^2$

Using the above definitions, we can give a description of LLL algorithm:

Algorithm 1 LLL Algorithm

```
Input : Lattice basis B = {b_1, b_2, ..., b_n} ∈ Z^n
Output : δ - LLL reduced basis for L(B)
Compute b̃_1, ..., b̃_n
for i = 2 to n do
    for j = i − 1 to 1 do
        b_i ← b_i − c_{i,j} b_j, where c_{i,j} = ⌈ <b_i, b̃_j> / <b̃_j, b̃_j> ⌋
if ∃i : δ ||b̃_i||^2 > ||μ_{i+1,i} b̃_i + b̃_{i+1}||^2 then
    b_i ↔ b_{i+1}
    Goto Step 3
end if
output b_1, b_2, ..., b_n
```

The LLL algorithm computes a vector which is at most $2^{\frac{n-1}{2}}$ times the shortest vector of the lattice and, is of time complexity of $O\left(n^6 \cdot \log^3 \alpha\right)$, where α is $max_{1 \leq i \leq n}||b_i||$. Although it does not compute the shortest vector, the output vector is short enough for applications like polynomial factoring. In the next section we will define search space for SVP.

IV. Search Space for SVP

A randomized search heuristic algorithm, like the Metropolis algorithm, for a combinatorial optimization problem works, given an instance, on a suitably defined search space associated with the instance. This space can be considered as a graph, possibly directed, where each vertex of the graph is a *feasible solution* of the instance, and an edge from u to v shows that from the feasible solution represented by u one can move to another feasible solution v through some inexpensive computation. Each vertex has a *cost* or a *value*, the goal of the search algorithm is to arrive at the vertex which represents the optimum solution for the instance. For a minimization problem, as SVP, the algorithm tries to find the solution with minimum cost. Usually, the number of feasible solutions for an problem instance will be exponential in the size of the instance. The search algorithm, therefore, will consider the search space only implicitly. At any given iteration of the algorithm, it will be at one feasible solution, the heuristic defines how to select a neighbouring vertex, at which the algorithm will be at the beginning of the next iteration.

The working of the Metropolis algorithm on an instance can be viewed as a random walk with a bias on a finite neighbourhood structure of *states*. Each state of the structure represents a feasible solution of the optimization problem instance being solved, and the structure has a *goal state*, the optimum point, which the algorithm intends to locate. For minimization problems, as our problem, SVP is, each point in the structure has a *cost*, and the goal state is the state with minimum cost (or, for decision versions, the state with cost less than or equal to a given specified cost). At any given step, the algorithm is at one of the points in the search space, it selects one of the neighbouring points and then transits to that point. The point is selected probabilistically, the bias ensures that the algorithm would reach the goal state eventually without getting stuck at local minima. For the Metropolis algorithm to run efficiently, it is necessary that the neighbourhood structure for an instance to satisfy

1. There should be at most exponentially (in instance size) many elements in the structure,
2. the diameter of the structure should be bounded above by a fixed polynomial in the instance size,
3. the set of neighbours of any element should be computable in time polynomial in the instance size, and
4. the cost of any state also should be computable efficiently.

For justifying the way we define our search space for the SVP [1], we need the following result.

Proposition IV.1. *[1] Let B be an $n \times n$ non-singular matrix and let $\mathbf{u}$ be a $n \times 1$ vector, $\| \mathbf{u}\| \leq K$, K being a non-zero constant. If there is an integer vector $\mathbf{w}$ such that $B\mathbf{w} = \mathbf{u}$, then the magnitude of every component of $\mathbf{w}$ is bounded above by M, $M = (\alpha n)^n$, where α denotes the largest value*

amongst the magnitudes of all the elements of B and K taken together.

The proof follows easily from the Cramer's rule, noting that, first, the determinant of B is $\neq 0$, and second, if β is the largest magnitude of all the entries of an $n \times n$ matrix Y, then $\det Y \leq (n\beta)^n$

Definition IV.1. *[Search Space for SVP [1]] Let B, an $n \times n$ matrix, be the basis of a lattice L and K be a given constant. Our goal is to look for a lattice vector of norm K or less. The search space for this instance of the SVP is as follows, where M is as in Proposition IV.1, and m is a parameter as fixed in the implementation. (m can be a fixed as a constant for all instances, or, more usually, it will be a fixed multiple of n.)*

1. *[Search space elements] The search space elements consist of matrices of the form $A' = [A|I]$, where I is the $n \times n$ identity matrix and A is an $n \times m$ matrix with all entries of magnitude bounded above by M, M as in Proposition IV.1.*
2. *[Definition of neighbourhood] For two elements R' and S', the latter is a neighbour of the former if S can be obtained from R by any of the following* elementary operations*:*
   (a) *By swapping two columns of R,*
   (b) *By multiplying a column of R by -1,*
   (c) *By adding a power of 2 multiple of one column of R' to another column of R, provided the resultant column satisfies that the magnitude of each of its components is less than equal to M. In particular, $r_i \leftarrow r_i^{'} \pm c \times r_j^{'}, (i \neq j,\ 1 \leq i \leq m, 1 \leq j \leq m+n$, and c, a positive integer, where $c = 2^0, \ldots, 2^k$, $k = n \cdot \log(\alpha n)$. (For a matrix R, r_i denotes its ith column.)(As stated already, this operation is allowed only if the component magnitude condition is satisfied.)*
3. *[Cost associated with a search space element] For an element $A' = [A|I]$ of the search space, its cost $c(A')$ is defined to be t, where t is the norm of that vector $\mathbf{v}$ which has the smallest norm amongst the $(m+n)$ vectors of $B[A|I]$. (In other words, by pre-multiplying the basis matrix B to $[A|I]$, we obtain an $n \times (m+n)$ matrix; t is the norm of the column vector with least norm amongst these $m+n$ column vectors of the matrix $B[A|I]$.)*

The following Proposition follows easily from the the way we have defined our search space.

Proposition IV.2. *[1] Let the $n \times n$ matrix B be a basis of the lattice L, and K is a given constant for the SVP instance.*

1. *For every element $[A|I]$ of the search space, each of the $(m+n)$ (column) vectors of the matrix $B[A|I]$ is a vector of the lattice L. Also, the $(m+n)$ column vectors of $B[A|I]$ generate the lattice L.*
2. *If L contains a vector of norm K or less, then the search space for SVP with L, K, will contain an element $[A|I]$ such that one of the $(m+n)$ column vectors of $B[A|I]$ will be of norm K or less. (The search space uses M as defined in Proposition IV.1.)*

The first part is obvious, as every column vector of $B[A|I]$ is an integer linear combination of the n lattice vectors. Also, as the vectors of B are contained in the vectors of $B[A|I]$, therefore, B and $B[A|I]$ both generate the same lattice (Here, we use the assumption that L is full dimensional). The second part of the Proposition also follows easily: we know from Proposition IV.1 that if there is a lattice vector of norm K or less, then there is a $\mathbf{v}$, the magnitude of each component of $\mathbf{v}$ bounded by M, such that the norm of $B\mathbf{v}$ is K or less. Our search space will have a member $[A|I]$ with A containing $\mathbf{v}$, as the latter can be obtained from the elementary operations we allow from the identity matrix I.

We now show that our search space definition satisfies the requirements we stated at the start of the Section. First we prove that every search space element has at most polynomially many neighbours.

Theorem IV.1. *[1] Let n, m be as in Definition IV.1 and M be as in Proposition IV.1. The number of neighbours for any node $A^{'}$ in the search space is $O(m^2 \log M)$. (As $\log M$ is $n \log(\alpha n)$, and $\log \alpha$ being the number of bits required to specify the largest magnitude number in the problem instance, we therefore have that every element has at most polynomially many neighbours.)*

The proof follows by noting that a search space element has at most ${}^mC_2$ neighbours through the first kind of elementary operation, m through the second kind, and ${}^mC_2 \times 2\,(k+1) + 2mn\,(k+1)$ of the third kind, where k is $O(\log M)$.

Next, our goal is to show that the search space has a polynomially bounded diameter.

Theorem IV.2. *[1] There is an $O(mn \log M)$-length path between the elements $A' = [A|I]$ and $B' = [B|I]$ (and vice versa), where A' and B' are any two elements in the search space.*

Proof. Let us first show how we can replace the i^{th} column a_i of A with b_i, the i^{th} column of B. In the first stage, using elementary operations, we get e_j in place of a_i, where e_j denote j^{th} column of the Identity matrix I (Since A has $m \geq n$ columns and m is multiple of n, $i = qn + j$ for some $q \in \mathbb{Z}$, where $1 \leq j \leq n$, $j = n$ if $i = qn$). We have to set j^{th} component of a_i to 1 and other component of a_i to 0. Suppose that the r^{th} component of a_i was x. Let $x = c_0x_0 + \ldots + c_kx_k$, where each c_i is 2^i and each x_i is 0 or 1, and k is $O(\log M)$. For $r \neq j$ we set the r^{th} component to zero by performing the elementary operations $a_i \leftarrow a_i - xe_r$ in atmost $k+1$ elementary operation. For $r = j$ we set the component to one by performing the elementary operation $a_i \leftarrow a_i - (x-1)e_j$ in at most $(k+1)$ elementary operations, since $x - 1 = y = 2^0y_0 + \ldots + 2^ky_k$, where each y_t is 0 or 1, $0 \leq t \leq k$. So total number of elementary operations to set each component of a_i is bounded by $n(k+1)$ elementary operation. Therefore the total number of elementary operations to set a_i for $1 \leq i \leq m$ is bounded by $mn(k+1)$. Now in the second stage, using elementary operations, we get b_i in place of $a_i = e_j$. Let r^{th} component of b_i be $z = 2^0z_0 + \ldots + 2^kz_k$, where each z_t is 0 or 1, $0 \leq t \leq k$. If $r = j$ by performing the elementary operation $a_i \leftarrow a_i + (z-1)e_j$, we can set j^{th} component of a_i to j^{th} component of b_i in atmost $k+1$ elementary operations. For $r \neq j$ we set the r^{th} component of a_i to r^{th} component of b_i by performing elementary operation $a_i \leftarrow a_i + ze_r$. This

implies that we can set a_i to b_i in atmost $n(k+1)$ elementary operations. Hence we can set a_i to b_i for all $1 \leq i \leq m$ in atmost $nm(k+1)$ elementary operations. Therefore we can set A to B in atmost $2nm(k+1)$ elementary operations. i.e. $O(mn \log M)$. Hence the proof. □

We shall see in Section VI that the Metropolis algorithm for SVP seems to perform surprisingly well. We provide below a result which to a certain extent explains this good performance, we prove that our serach graphs have large magnifications. Let G be a graph, $G = (V, E)$. For any subset A of V, let $E(A, \overline{A})$ denote the set of edges from A to $V - A$. The *magnification* $\mu(G)$ of G is defined as

$$\mu(G) = \min_{0<|A|\leq|V|/2} \frac{|E(A,\overline{A})|}{|A|}$$

We now show that our search space graphs for SVP have magnifications at least half.

Theorem IV.3. *A search space graph for SVP has magnification* $\geq \frac{1}{2}$

Proof. For any two verices A', B' of the search graph let us define the *canonical path* [10][11] from A' to B' to be the path as defined in the proof of Theorem IV.2. We prove that the number of canonical paths that pass through any given transition (i.e., edge) is bounded by the total number of states (i.e N) in the search space. Let $C' = [C|I]$ and $D' = [D|I]$ be two adjacent vertices in a canonical path, and consider the transition (edge) taken in the path to go from C' to D'. Such a transition adds or subtracts a power of 2 to a component of a vector in C. For example, let

$C = d_1, d_2, \ldots d_{k-1} \boxed{d_k^1, 0 \ldots 0} c_{k+1} \ldots c_m$ and

$D = d_1, d_2, \ldots, d_{k-1} \boxed{d_k^1, d_k^2, 0 \ldots 0} c_{k+1} \ldots c_m$

where $d_k^i = 2^j$ for some $j = 0, 1, \ldots, k$. Here we are using the an elementary operation of the kind that adds a power of two times an identity matrix column to a column of C. Clearly, the number of canonical paths that share this transition is less than equal to the number of ways to choose $d_1, \ldots, d_k \times$ the number of ways to choose $c_{k+1}, \ldots, c_m$, which is equal to the number of ways to choose a sequence of m columns,
in other words, the total number of states N.
Hence, number of canonical paths that pass through any edge is bounded by the total number of states in the search space. For a subset S of vertices, let $E(S, \overline{S})$ denote the edges going out of S to the rest of the vertices, i,e., $\overline{S}$. Let S be the subset of states which defines $\mu(G)$, for our search space graph G. Therefore, $\mu(G) = \frac{E(S,\overline{S})}{|S|}$. However, $|S| \times |\overline{S}|$ canonical paths go from S to $|\overline{S}|$. Each of these paths passes through one of the edges of $E(S, \overline{S})$. As no edge can have more than N canonical paths passing through it, $N \times E(S, \overline{S}) \geq |S| \times |\overline{S}|$.
As $|S^{'}| \geq \frac{N}{2}$, $N \times E(S, \overline{S}) \geq |S| \times \frac{N}{2}$,
which implies $\frac{E(S,\overline{S})}{|S|} \geq \frac{1}{2}$. Therefore $\mu(M) \geq \frac{1}{2}$. □

The significance of the above result is as follows. Sanyal, Raja, and Biswas proved in [2] that a Markov chain family used for an optimization problem will be able to find the optimum with high probability within a number of steps bounded by a fixed polynomial in the input instance size if and only if each chain in the family is rapidly mixing[10][11] and the stationary probability of each goal state is high, i.e., larger than an inverse polynomial in instance size. Now, it is well known that a time reversible chain (as in the case of the Metropolis) is rapidly mixing iff it has high conductance[10][11], that is, from every subset S of vertices, where the probability of being in a state in S in the stationary distribution being no more than half, there is a large ergodic flow out of S. Now, we have shown in the above result that there will be a large number of edges going out of sets of vertices. It is, therefore, likely that these large number of edges together will also carry a large ergodic flow. If that is the case for an instance, the Markov chain for the instance will be rapidly mixing.
We have proved that the entries of A is bounded by $O(\alpha n^n)$ and Theorem IV.2 suggest that there exists a path using which we can reach any node in the search space. Hence our search space Definition IV.1 ensures that entries of intermediate matrices will not grow exponentially and we can also reach from one state to another with in polynomial number of steps. We are always interested in finding shortest non zero vector in the lattice but there are chances that we may get zero vector while applying the elementary operations defined in Definition IV.1 on the matrix $A^{'} = [A|I]$. To avoid this, we can define a cost of zero vector as infinity which prevents from moving to such neighbours due to its very high cost. Let us now define the Metropolis algorithm for SVP.

V. Metropolis Algorithm

The pseudo-code of the Metropolis algorithm is given below (Algorithm 2). As mentioned before, the metropolis algorithm is the execution of a Markov process. It is therefore completely defined once the transition probabilities are defined.
Consider a search space and neighbourhood structures as defined in Definition IV.1. Observe that only one row of current solution will be changed by the elementary operations performed on it. The cost function can be modified as follows: Let $R^{'}$ be the current solution and $S^{'}$ be the new solution. $S^{'}$ is obtained from $R^{'}$ by applying one of the elementary transformation as defined in the Definition IV.1 to the r^{th} column of R. Hence, the cost function $c(R^{'})$ is the Euclidian norm of the r^{th} column vector of $B * R$ and $c(S^{'})$ is the Euclidian norm of the r^{th} column vector of $B * S$.
The Metropolis algorithm on instance $R^{'} = [R|I]$ runs a Markov chain $X^{R^{'}} = (X_1^{R^{'}}, X_2^{R^{'}}, \ldots)$, using the temperature parameter T. The state space of the chain is the set $S^{R^{'}}$ of the feasible solutions of $R^{'}$. Let d denote the degree of the node in a search graph where $d = O(m^2 \cdot \log M)$ as in Theorem IV.1. Let $R^{'}$ and $S^{'}$ denote any two feasible solutions and neighbourhood of $R^{'}$ is denoted by $N(R^{'})$. Then the transition probabilities are as follows:

$$q_{R'S'} = \begin{cases} 0, \text{ if } R^{'} \neq S^{'} \ \& \ S^{'} \notin N(R^{'}) \\ \frac{e^{-\left(c(S^{'})-c(R^{'})\right)/T}}{d}, \text{ if } c(S^{'}) > c(R^{'}) \& R^{'} \in N(R^{'}) \\ \frac{1}{d}, \text{ if } c(R^{'}) \geq c(S^{'}) \ \& \ S^{'} \in N(R^{'}) \\ 1 - \sum_{J' \neq R'} q_{J'R'}, \text{ if } R^{'} = S^{'} \end{cases}$$

The complete algorithm (Algorithm 2) is given below [1].

Algorithm 2 Metropolis Algorithm

1: Input : $B \leftarrow$ Basis for the lattice L and a rational number K
2: Output : Matrix $R^{'}$ such that $B * R^{'}$ contains a vector $\mathbf{v}$ with $||\mathbf{v}|| \leq K$.
3: Let $I \leftarrow n \times n$ Identity matrix. Let $R^{'} = [R|I]$ be the starting state in the search space as in Definition IV.1 and $c(R^{'})$ denote cost of $R^{'}$ as defined in the beginning of this section.
4: Set $BestNorm = c(R^{'})$
5: **while** $BestNorm > K$ **do**
6: Select any one of the neighbour $S^{'}$ of $R^{'}$ uniformly at random by performing one of the elementary operation as defined in Definition IV.1
7: **if** $BestNorm > c(S^{'})$ **then**
8: $BestNorm = c(S^{'})$
9: **end if**
10: Set $R^{'} = S^{'}$ with probability

$$\alpha = \min\left(\frac{e^{-c(S^{'})/T}}{e^{-c(R^{'})/T}}, 1\right)$$

11: **end while**

Hasting's Generalization of Metropolis Algorithm

The way the Metropolis algorithm decides about moving from the current state s_i to a state in the neighbourhood can be seen as a two stage process: first, choose a neighbour s_j uniformly at random (*the proposal stage*), and then, with a probability α which depends upon the relative costs of the solutions associated with s_j and s_i, move to s_j or remain at s_i (*the acceptance stage*).
In our case, the neighbours of a state $[A|I]$ are $[A'|I]$'s where A' is obtained by performing an elementary operation using the vectors in A and I. Some of the elementary operations represent what we call *long jumps* because a vector $\mathbf{v}$ is replaced by another $\mathbf{u}$ where there is a large difference in the norms of $\mathbf{v}$ and $\mathbf{u}$. This happens when $\mathbf{v}$ is replaced by $\mathbf{v} \pm c\mathbf{w}$ when the constant c is large. It is desirable to have a control on how extensively our algorithm will make use of such long jumps. This is not possible in the standard Metropolis algorithm as the proposal stage will chose a neighbour uniformly at random.
To overcome this problem, we make use of the Hasting's generalization [12][1] of the Metropolis algorithm. In this generalization, we can use any probability to select the neighbour of a state in the proposal stage. Let S be any state space. First we define a Markov chain M_1 on S. The transition probabilities of M are as follows. Let q_{xz} denote the probability by which we select a neighbour z when the current state is x. Let x be a state. If $y_1, \ldots, y_{n_x}$ be neighbours the neighbours of x. Then

$$q_{xz} = \begin{cases} 0 & \text{if } x \neq z \text{ and } z \notin N(x) \\ \theta & \text{if } x = z, \\ r_i & \text{if } z = y_i \end{cases},$$

where the values r_i can be chosen appropriately depending on how much we want to invest on each of the strategy.

The Hasting's generalized metropolis algorithm M_2 runs on the same state space but has a different transition probability: Suppose the chain M_2 is at a state the state x at some step. Then

1. With probability q_{xz}, M_2 selects a state z in the neighbourhood.
2. If $z = x$ then the next state of M_2 is x.
3. If $z = y_i$, we first compute α defined as

$$\alpha = \min\left(\frac{e^{-c(y_i)/T} \cdot q_{y_i x}}{e^{-c(x)}/T \cdot q_{x y_i}}, 1\right)$$

Here, for any state z, $c(z)$ represents the cost of the candidate solution of z and T is a fixed temperature parameter.
4. We move to y_i with probability α else we remain in the present state x.

It can be verified easily that the chain M_2 is time-reversible and then its stationary distribution, the probability of x, π_x is given by:

$$\pi_x = \frac{e^{-c(x)/T}}{Z},$$

where Z is the normalizing factor $\sum_i \pi_i$. Clearly, when M_2 is in x, the next state is either x itself, or one of its neighbours y_i. Let $\frac{\pi_{y_i} q_{y_i x}}{\pi_x q_{x y_i}} < 1$. Then p_{xy_i}, the probability of moving from x to y_i in M_2 is given by $p_{xy_i} = q_{xy_i} \times \frac{\pi_{y_i} q_{y_i x}}{\pi_x q_{x y_i}}$. We now verify that $\pi_x p_{xy_i} = \pi_{y_i} p_{y_i x}$.
LHS = $\pi_x p_{xy_i} = \pi_x q_{xy_i} \frac{\pi_{y_i} q_{y_i x}}{\pi_x q_{x y_i}} = \pi_{y_i} q_{y_i x}$
For RHS, note that $\alpha = 1$ Therefore $p_{y_i x} = q_{y_i x}$. Hence, $\pi_{y_i} p_{y_i x} = \pi_{y_i} q_{y_i x}$.
Thus, the time reversibility for M_2 holds and its stationary distribution probability for any x is π_x.
The chain M_2 is the Hasting's generalization. This chain has the same stationary distribution as the usual Metropolis algorithm, but has the flexibility of fine tuning the probability of choosing a neighbour to reflect the structure of the problem at hand. In our implementation, we shall keep q_{xy_i} the same as $q_{y_i x}$. The detailed algorithm (Algorithm 3) is given below [1]. In the next section we will compare the results of our algorithm with that of LLL algorithm.

VI. Results

In this section we describe how our algorithm compares with the celebrated LLL [7] algorithm on benchmark instances SVPs'. We have tested our algorithm for basis **B** with different dimension n. We implemented our algorithm using NTL-5.5.2 [13] and compared the result with NTL's in built optimized function LLL. We now describe the benchmark lattices that we ran this algorithm on. A class of SVP instances are generated using the techniques developed by Richard Lindner and Michael Schneider [14]. They have given sample bases for Modular, Random, ntru, SWIFT and Dual Modular lattices of dimension 10. We have tested our code for all these instances and found that our algorithm works faster and gives shorter lattice vector when compared to LLL. The tested results are given in the Table 1 and Table 2

Algorithm 3 Hasting's Generalization

Input : $B \leftarrow$ Basis for the lattice L and a rational number K

Output : Matrix $R^{'}$ such that $B * R^{'}$ contains a vector $\mathbf{v}$ with $||\mathbf{v}|| \leq K$.

Let $I \leftarrow n \times n$ Identity matrix. Let $R^{'} = [R|I]$ be the starting state in the search space as in Definition IV.1 and $c(R^{'})$ denote cost of $R^{'}$ as defined in the beginning of this section. Let d denote total number of neighbours as in Theorem IV.1

Set $BestNorm = c(R^{'})$

while $BestNorm > K$ **do**

Select any one of the neighbour $S^{'}$ of $R^{'}$ by performing one of the elementary operations defined below.

- Swap two columns of R with probability $\frac{^mC_2}{d}$,
- Multiply a column of R by -1 with probability $\frac{m}{d}$
- Add a power of 2 times a column of R' to another column of R i.e. in particular, $r_i \leftarrow r_i^{'} \pm c \times r_j^{'}, (i \neq j, 1 \leq i \leq m, 1 \leq j \leq m+n$, where $c = 2^0, \ldots, 2^k, k = n \cdot \log(\alpha n))$ with probability $\frac{d-^mC_2-m}{d} \cdot P_i$, where P_i denote probability of selecting the value of $c = 2^i$ and $\sum_{i=0}^{k} P_i = 1$.

  [We can use more than one probability distribution to select values for c. In our implementation we have selected two probability distributions $P_i = \frac{1}{k+1}$ and $Q_i = 2(k+1-i)/(k+1)(k+2)$ to select values for c. We will keep on changing our selection probability distribution with P_i and Q_i for every selected number of steps(500 steps).]

if $BestNorm > c(S^{'})$ **then**

$BestNorm = c(S^{'})$

end if

Set $R^{'} = S^{'}$ with probability

$$\alpha = \min\left(\frac{e^{-c(S^{'})/T}}{e^{-c(R^{'})/T}}, 1\right)$$

end while

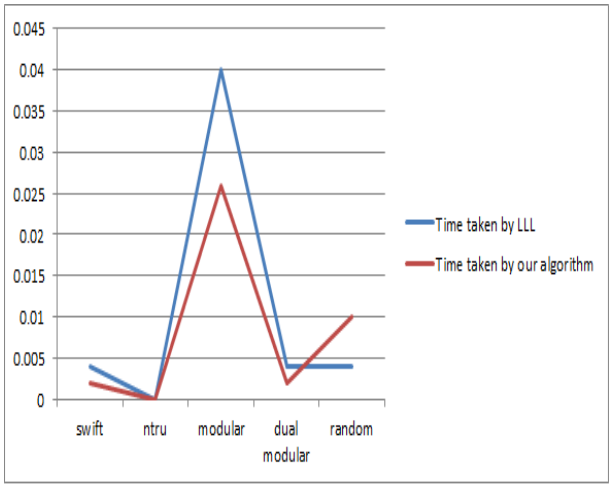

Figure. 1: Type of Lattice Vs Time Taken by LLL and Our Algorithm

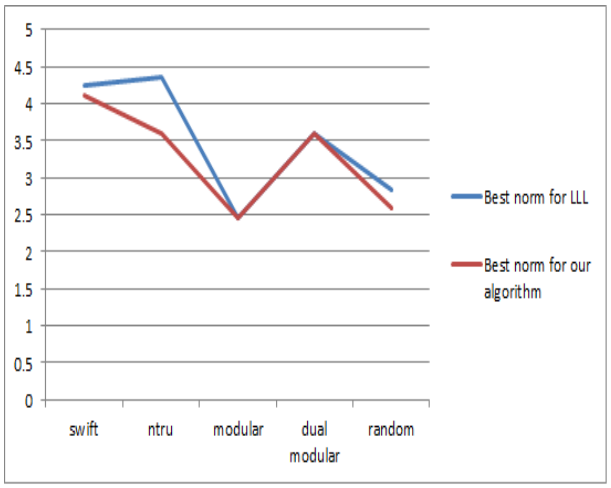

Figure. 2: Type of Lattice Vs Best Norm found by LLL and Our Algorithm

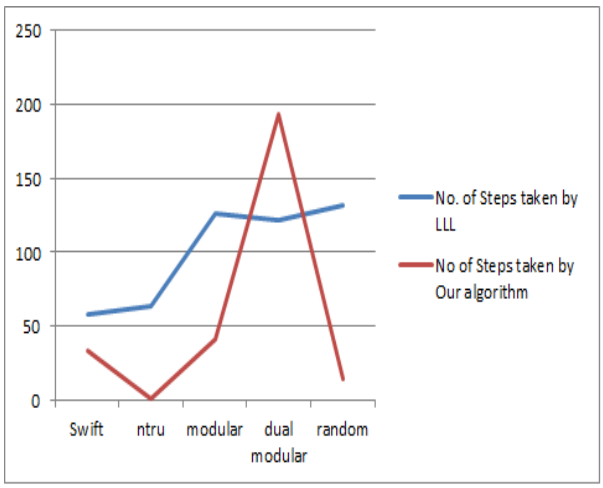

Figure. 3: Type of Lattice Vs Number of steps taken by LLL and Our Algorithm

Table 1: Results obtained by applying LLL(Data taken from [14][1])

Lattice Type	Dim. n	Best Norm Found	CPU Time in seconds	Input size in bits	No. of Steps Taken
Swift	8	4.242	0.04	8	58
NTRU	8	4.358	0	8	64
Modular	10	2.449	0.04	8	126
Dual Modular	10	3.6055	0.004	8	122
Random	10	2.828	0.004	8	132

Table 2: Results obtained by applying Hasting's Generalization (Data taken from [14][1])

Lattice Type	Dim. n	Best Norm Found	CPU Time in seconds	Input size in bits	No. of Steps Taken
Swift	8	4.12	0.002	8	33
NTRU	8	3.6	0.002	8	1
Modular	10	2.449	0.026	8	41
Dual Modular	10	3.6	0.002	8	193
Random	10	2.6	0.01	8	14

The Fig. 1 shows that our algorithm takes less time when compared to LLL. Fig. 2 shows that Our algorithm gives shorter vector than LLL algorithm for the given instances. Fig 3 shows that the number of steps taken by our algorithm is less than LLL algorithm for most of the given instances.

Based on the result by Ajtai [15], Johannes Buchmann, Richard Lindner, Markus Ruckert and Michael Schneider [16] [17] constructed a family of lattices for which finding the short vector implies being able to solve difficult computational problems in all lattices of a certain smaller dimension. For completeness we give a quick description of these family.

Definition VI.1. *Let n be any positive integer greater than 50, c_1, c_2 be any two positive real numbers such that $c_1 > 2$ and $c_2 \leq c_1 \ln 2 - \frac{\ln 2}{50 \cdot \ln 50}$. Let $m = c_1 \cdot n \cdot \ln n$ and $q = n^{c_2}$. For a matrix $X \in \mathbb{Z}^{n \times m}$, with column vectors $x_1, \ldots, x_m$, let*

$$L(c_1, c_2, n, X) = \left\{ (v_1, \ldots, v_m) \in \mathbb{Z}^m | \sum_{i=1}^{m} v_i \mathbf{x}_i \equiv \mathbf{0} \ mod \ q \right\}$$

All lattices in the set $L(c_1, c_2, n, .) = \{L(c_1, c_2, n, X) | X \in \mathbb{Z}_q^{n \times m}\}$ are of dimension m and the family of lattices $\mathbb{L}$ is the set of all $L(c_1, c_2, n, .)$

They[16][17] also proved that all lattices in $L(c_1, c_2, n, .)$ of the family $\mathbb{L}$ contain a vector with Euclidean norm less than $\sqrt{m}$ and it is hard to find such vector. The challenge is to try different means to find a short vector. The Challenge is defined in the following definition:

Definition VI.2 (Lattice Challenge:)**.** *Given lattice basis of lattice L_m, together with a norm bound ν. Initially set $\nu = \lceil \sqrt{m} \rceil$. The goal is to find a vector $\mathbf{v} \in L_m$, with $||\mathbf{v}||_2 \leq \nu$. Each solution $\mathbf{v}$ to the challenge decreases ν to $||\mathbf{v}||_2$.*

Table 3: Results obtained by applying LLL (Data taken from [16][17][1]: Toy Challenge)

Dimension n	Best Norm Found	CPU Time in seconds	Input size in bits	No. of Steps Taken
10	2.49	0	8	78
15	1234.6	0.016	150	2183
20	3	0.27	8	320
25	1.73	0.008	8	373
30	4.123	0.004	8	364
50	20.49	0.11	100	3920

Table 4: Results obtained by applying Hasting's Generalization (Data taken from [16][17][1]: Toy Challenge)

Dimension n	Best Norm Found	CPU Time in seconds	Input size in bits	Number of steps
10	2.23	0	8	18
15	1147.2	185.6	150	123567
20	2.83	0.05	8	640
25	1.73	-1.41624e-18	8	1
30	3.464	0.001	8	10
50	8.66	294.2	100	256789

We have tested our algorithm for toy challenges(i.e. with $m \leq 50$) and the comparison results with LLL is listed in Table 3 and 4.

An important step in the LLL algorithm is the computation of the Gram-Schmidt orthogonalisation of the basis in hand. In practical implementations, this computation is done using floating point numbers instead of multiprecision arithmetic to speed up computation. We apply a similar technique here. At each step our transition probabilities are based on the value of the objective function, which is the length of the smallest vector in the current solution. We compute this length using floating point arithmetic instead of the full multiprecission arithmetic.

Our results are very encouraging. For all the examples considered, we found that our algorithm performs well either in value or in time and often in both than LLL. When the number of bits used to represent integer value is more than 100 bits we found that LLL is more faster than our algorithm but our algorithm gives shorter vector than LLL.

The Fig. 4 shows that our algorithm takes less time than LLL for some instances. Fig. 5 shows that Our algorithm gives shorter vector than LLL algorithm for the given instances. Fig. 6 shows that for some instances our algorithm takes less number of steps than LLL but when number of bits used to represent input values is greater than 100 bits LLL takes less number of steps than our algorithm.

VII. Conclusion

In this paper we have defined the search space for svp and also considered the use of the Metropolis algorithm and its generalization due to Hastings, for solving SVP, a well-known, hard combinatorial optimization problem. To the best of our knowledge, this is the first such attempt. Our

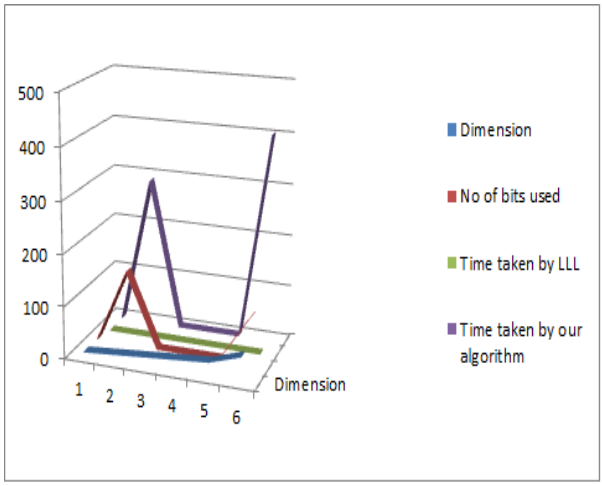

Figure. 4: Number of Bits, Dimension Vs Time Taken by LLL and Our Algorithm

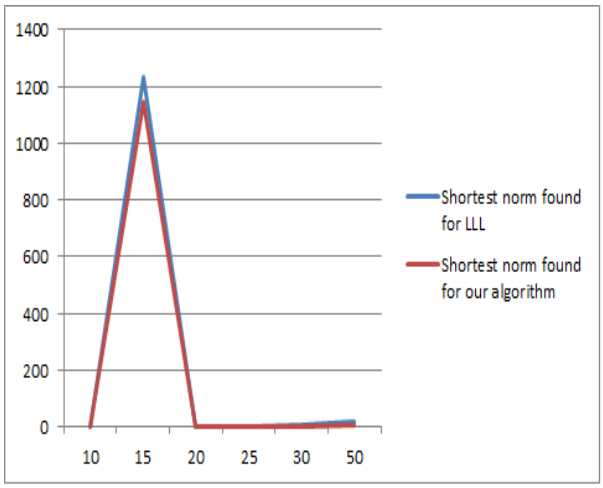

Figure. 5: Dimension Vs Best Norm found by LLL and Our Algorithm

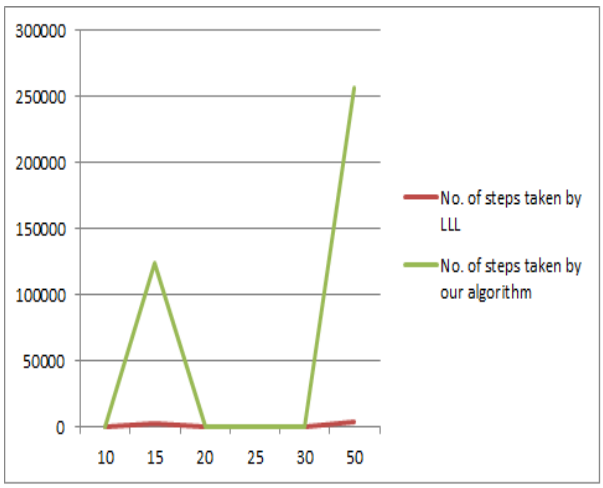

Figure. 6: Dimension Vs Number of steps taken by LLL and Our Algorithm

approach rests on an appropriate definition of a search space for the problem, which can be used for some other classes of evolutionary algorithms as well, e.g., genetic algorithm and the go-with-the-winner algorithm. We have also proved that our Search Space graph for SVP has magnification greater than half. We have compared the performance of our implementation with that of a standard implementation of the LLL algorithm; and the results we have obtained are fairly encouraging. Given this experience, it is worth while to explore if it can be shown that our approach is efficient for random instances of SVP.

Acknowledgment

This work was carried out by Ajitha Shenoy K B in the Department of Computer Science and Engineering, Indian Institute of Technology (IIT), Kanpur, India. Ajitha Shenoy K B would like to thank Research-I foundation, Department of Computer Science and Engineering, Indian Institute of Technology (IIT), Kanpur, India and Manipal University, Manipal, India for their support in pursuing this research work.

References

[1] Ajitha Shenoy K. B. and Somenath Biswas and Piyush P Kurur, "Metropolis algorithm for solving shortest lattice vector problem (svp)," in *Proceedings of* 11^{th} *international conference on Hybrid Intelligent Systems*, Melacca, Malaysia, 2011, pp. 442–447.

[2] Swagato Sanyal and S. Raja and Somenath Biswas, "Necessary and sufficient conditions for success of the metropolis algorithm for optimization," in *Proceedings of the tenth ACM GECCO'10*, Portland, OR, USA, 2010, pp. 1417–1424.

[3] Ted Carson, "Emperical and analytic approaches to understanding local search heuristics," in *PhD Thesis*, University of California, San Diego, 2001.

[4] P. Van Emde Boas, "Another np-complete problem and the complexity of computing short vector in a lattice," in *Tech. rep 8104*, University of Amsterdam, Department of Mathematics, Netherlands, 1981.

[5] M. Ajtai, "The shortest vector problem in l_2 is np-hard for randomized reductions," in *STOC 98: Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, New York, NY, USA, 1998, pp. 10–19.

[6] D. Micciancio, "The shortest vector in a lattice is hard to approximate to within some constant," *SIAM Journal of Computing*, vol. 30(6), pp. 2008–2035, 2001.

[7] A.K. Lenstra and H. W. L. Jr. and L. Lovasz, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261(4), pp. 515–534, 1982.

[8] C. Schnorr, "A more efficient algorithm for lattice basis reduction," *Journal of Algorithms*, vol. 9(1), pp. 47–62, 1988.

[9] M. Ajtai, "The worst-case behavior of schnorr's algorithm approximating the shortest nonzero vector in a lattice," in *STOC-03: Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, ACM Press, 2003, pp. 396–406.

[10] Mark Jerum and Alistair Sinclair, "Approximating the permanent," *SIAM Journal of Computing*, vol. 18, pp. 1149–1178, 1989.

[11] Jerum and Alistair Sinclair, "conductance and rapid mixing of property for markov chain : The approximation of the permanent resolved," in *Proceedings of the symposym on Theory of Computer Science*, 1998.

[12] Michael Mitzenmacher and Eli Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Page 269: Cambridge University Press, 2005.

[13] Victor Shoup, "http://www.shoup.net/ntl/."

[14] Sage reference v4.7, "Cryptography," www.sagemath.org/doc/reference/sage/crypto/lattice.html.

[15] M. Ajtai, "Generating hard instances of lattice problems (extended abstract)," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, STOC'96, ACM*, New York, NY, USA, 1996.

[16] J. Buchmann, R. Lindner, M. Ruckert, and M. Schneider, "Explicit hard instances of the shortest vector problem," in *PQ Crypto,* 2^{nd} *Internation Workshop on Post Quantum Cryptography*, LNCS 5299, 2008, pp. 79–94.

[17] TU Darmstadt, "Lattice challenge," www.latticechallenge.org.

Author Biographies

Mr. Ajitha Shenoy K B is a research scholar in the Department of Computer Science and Engineering, Indian Institute of Technology (IIT), Kanpur, India. He obtained M.Tech (Computer and Information Science) degree from Cochin University of Science and Technology, Cochin, India in the year 2003. He also obtained MSc(Mathematics) degree from Kannur University, Kannur, India in the year 1999. He is working as an Assistant Professor in the Department of MCA, Manipal Institute of Technology, Manipal Universtity, Manipal, India. His research interests are Randomized Local search algorithm, Combinatorial Optimization, Image Compression and Graph Algorithms.

Dr. Somenath Biswas is a Professor, Department of Computer Science and Engineering, Indian Institute of Technology (IIT), Kanpur, India. His research interests are Randomized algorithms, computational biology, computational complexity and logic in computer science.

Dr. Piyush P Kurur is a Associate Professor, Department of Computer Science and Engineering, Indian Institute of Technology (IIT), Kanpur, India. His research interests are computational algebra, quantum computation and computational number theory.