Metropolis Algorithm for solving Shortest Lattice Vector Problem(SVP)

Ajitha Shenoy K B
Department of Computer Science and Engineering
Indian Institute of Technology
Kanpur, INDIA
ajith@cse.iitk.ac.in

Somenath Biswas
Department of Computer Science and Engineering
Indian Institute of Technology
Kanpur, INDIA
sb@cse.iitk.ac.in

Piyush P Kurur
Department of Computer Science and Engineering
Indian Institute of Technology
Kanpur, INDIA
ppk@cse.iitk.ac.in

***Abstract*—In this paper we study the suitability of the Metropolis Algorithm and its generalization for solving the shortest lattice vector(SVP) 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 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 can use this search space formulation for some other classes of evolutionary algorithms, e.g., for genetic and go-with-the-winner algorithms. We have implemented the Metropolis algorithm and Hasting's generalization of Metropolis algorithm for the SVP. Our results are quite encouraging in all instances when compared with LLL algorithm.**

***Index Terms*—SVP, Search Space, Metropolis Algorithm, Hasting's Generalization, LLL.**

I. Introduction

We investigate in this paper the suitability of using the Metropolis algorithm [1][2] to solve the shortest lattice vector problem (SVP, for short). The Metropolis algorithm is a widely used randomized search heuristic and is often used in practice to solve optimization problems. It is known that the algorithm performs surprisingly well even for some provably hard problems; e.g, [2] 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 [3] 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 NP-hardness in the 2 norm (or in any finite p norm) was an open problem for a long time. A breakthrough result by Ajtai[4] in 1998 finally showed that SVP is NP-hard under randomized reductions. Another breakthrough by Micciancio [5]in 2001 showed that SVP is hard to approximate within some constant factor, specifically 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. [6] in 1982, Schnorr [7] in 1988, and Ajtai et. al. [8] 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 define an appropriate search space for our approach to solve SVP, in Section 4 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 5 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 2.1 (Lattices): Let $B = \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n\}$ 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 $\mathbf{b}_1, \dots, \mathbf{b}_n$ as its columns. Then we can write $L(B) = \{B\mathbf{a} : \mathbf{a} \in \mathbb{Z}^n\}$

Problem 2.2 (Shortest Lattice Vector Problem (SVP)): Given a lattice $L(B)$ contained in $\mathbb{Z}^n$ specified by linearly independent vectors $\mathbf{b}_1, \dots, \mathbf{b}_n$, 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.

III. Search Space for SVP

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 (a) there should be at most exponentially (in instance size) many elements in the structure, (b) the diameter of the structure should be bounded above by a fixed polynomial in the instance size, (c) the set of neighbours of any element should be computable in time polynomial in the instance size, and similarly, the cost of any state also should be computable efficiently.

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

Proposition 3.1: 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 3.2: [Search Space for SVP] 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 3.1, 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 3.1.
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 $(n + m)$ vectors $B[A|I]$. (In other words, by pre-multiplying the basis matrix B to $[A|I]$, we obtain an $n \times m$ matrix; t is the norm of the column vector with least norm amongst these $n + m$ column vectors of the matrix $B[A|I]$.)

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

Proposition 3.3: 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 $(n + m)$ (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 3.1.)

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 3.1 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 3.4: Let n, m be as in Definition 3.2 and M be as in Proposition 3.1. 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 ${}^{m}C_2$ neighbours through the first kind of elementary operation, m through the second kind, and ${}^{m}C_2 \times 2\,(k+1) + 2mn\,(k+1)$ 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 3.5: 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 have proved that the entries of A is bounded by $O(\alpha n^n)$ and Theorem 3.5 suggest that there exists a path using which we can reach any node in the search space. Hence our search space Definition 3.2 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 3.2 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.

IV. Metropolis Algorithm

The pseudo-code of the Metropolis algorithm is given below (Algorithm 1). 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 3.2. 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 3.2 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 Eulidian 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 3.4. 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:

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

The complete algorithm (Algorithm 1) is give below.

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

Algorithm 1 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 3.2 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 3.2
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
```

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 [9] of the Metropolis algorithm. In this generalization, we can use any probability to select the neighbour of a state in the proposal stage. 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 the in 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 $\sum_i \pi_i$. 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 2) is given below. In the next section we will compare the results of our algorithm with that of LLL algorithm.

Algorithm 2 Hasting's Generalization

```
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 3.2 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 3.4
4: Set $BestNorm = c(R^{'})$
5: while $BestNorm > K$ do
6:     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).]
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
```

TABLE I
COMPARISON OF RESULTS LLL : METROPOLIS (DATA TAKEN FROM [10])

Lattice Type	Dim. n	Best Norm Found (LLL : Our Algo)	CPU Time in seconds (LLL : Our Algo)	Input size in bits
Swift	8	4.242 : 4.123	0.04 : 0.004	8
NTRU	8	4.358 : 3.6055	0 : 0.004	8
Modular	10	2.449 : 2.449	0.04 : 0.024	8
Duarl Modular	10	3.6055 : 3.6055	0.004 : 0.004	8
Random	10	2.828 : 2.645	0.004 : 0.012	8

TABLE II
COMPARISON OF RESULTS LLL : METROPOLIS (DATA TAKEN FROM [13][12]: TOY CHALLENGE)

Dimension n	Best Norm Found (LLL : Our Algo)	CPU Time in seconds (LLL : Our Algo)	Input size in bits
15	1234.58 : 1147.2279	0.016 : 201.651	150
20	3 : 2.8284	0.2680 : 0.052	8
25	1.73 : 1.73	0.008 : 0.004	8
30	4.123 : 3.8729	0.008 : 0.008	8
50	20.49 : 8.66	0.108 : 291.892	100

V. RESULTS

In this section we describe how our algorithm compares with the celebrated LLL [6] algorithm on benchmark instances SVPs'. The LLL algorithm computes a vector which is at most $2^{\frac{n-1}{2}}$ of the shortest vector of the lattice and has a 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.

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 [10]. 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 I

Based on the result by Ajtai [11], Johannes Buchmann, Richard Lindner, Markus Ruckert and Michael Schneider [12] [13] 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 5.1: 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 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 5.2 (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 $||\mathbf{v}||_2$.

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

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.

VI. CONCLUSION

In this paper we have 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 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 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.

REFERENCES

[1] S. Sanyal, S. Raja, and S. Biswas, "Necessary and sufficient conditions for success of the metropolis algorithm for optimization," in *GECCO'10, Copyright ACM*, Portland, USA, 2010.

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

[3] P. V. E. 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.

[4] 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.

[5] 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.

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

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

[8] 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.

[9] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Page 269: Cambridge University Press, 2005.

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

[11] 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.

[12] 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.

[13] T. Darmstadt, "Lattice challenge," www.latticechallenge.org.