Perturbed Analysis of LLL algorithm

Nikhil Mattapally - EE17B138
Adithya Swaroop - EE17B115
June 15, 2021

Abstract

The Lenstra–Lenstra–Lovász (LLL) lattice basis reduction algorithm is a polynomial time lattice reduction algorithm. It is a practical method with enough accuracy in solving integer linear programming, factorizing polynomials over integers and breaking cryptosystems. In this project we implement the LLL algorithm and analyse its run time for perturbed inputs.

1 Introduction

An n-dimensional lattice L is a set of vectors $\{\Sigma_{i=1}^{n} b_i v_i | b_i \in Z\}$ where $v_1, v_2, ..., v_n \in R_m$ is a set of linearly independent vectors called the basis for the lattice. The same lattice could have many bases. Given a basis for an n-dimensional lattice, the Shortest Vector Problem asks for the shortest non- zero vector in the lattice. The length of the vectors can be measured in any l_p norm ($p \geq 1$) and the corresponding problem is denoted by SVP_p. For our project we are using $p = 2$. Finding shortest non-zero vector in 2-dimensional lattices was solved by Gauss in the 19th century, but there was not efficient algorithm of solving SVP in higher dimensions until 1980s. In 1981, mathematician Perter van Emde Boas conjectured that SVP is a NP-hard problem.

Up to date, there is no algorithm can solve SVP exactly and run efficiently, but Arjen Lenstra, Hendrik Lenstra and Laszlo Lovász posted a well-known approximation algorithm of SVP which is called the LLL algorithm in 1982, which can approximate a non-zero lattice vector in n-dimension lattice L in a polynomial runtime w.r.t n.

2 LLL algorithm

Lenstra-Lenstra-Lovász (LLL) Algorithm is an approximation algorithm of the shortest vector problem. It runs in polynomial time and finds an approximation within an exponential factor of the correct answer. To find the shortest vector the algorithm tries to reduce the basis.

Basis reduction: Basis reduction is a process of reducing the basis of a lattice L to a shorter basis while keeping L the same. For a lattice with n vector basis we are trying to find n shortest possible vectors belonging to lattice which are linearly independent.

For a 2 dimensional lattice a basis (b_1, b_2) is said to be reduced if it satisfies the following condition

$$||b_1|| \leq ||b_2||$$

$$u_{1,2} = \frac{b_1 * b_2}{||b_1||^2} \leq 0.5$$

u is called the orthogonal projection coefficient. We can prove that the b_1 of this reduced basis is the shortest vector. Gauss has developed an algorithm which uses these conditions to find the shortest vector.

The idea of basis reduction in two dimensional lattice is to find the orthogonal basis based on the given basis. The basis we found in Gauss algorithm is not exactly orthogonal, but it is the nearest basis we can get. To generalize the algorithm to n-dimensions, we need to find a way to construct n-dimensional orthogonal basis based on the given basis, which leads us to Gram-Schmidt Orthogonalization

Gram-Schmidt Orthogonalization: Given a basis $\{b_1, b_2, ..., b_m\}$of a subspace $H_m \epsilon R^n$

$$\begin{aligned} {b_1}^* &= b_1 \\ {b_2}^* &= b_2 - u_{1,2} b_1 \\ &\vdots \\ {b_m}^* &= b_m - \textstyle\sum_{i<m} u_{i,m} b_i \end{aligned} \tag{1}$$

Then the basis $\{{b_1}^*, {b_2}^*, ..., {b_m}^*\}$ is the correspond orthogonal basis which is also called as Gram-Schmidt Orthogonal(GSO) basis. In this process, one vector is kept constant(first one) and for other vectors, we remove components of previous vectors.

By observing in the two dimensional case, we found that in the process of reducing the basis, we want to make the basis vectors as the most orthogonal as possible. The orthogonal basis cannot be reduced any more. With the Gram-Schmidt orthogonalization in higher dimensions, A. Lenstra, H. Lenstra, and L. Lovász proposed an approximation algorithm of basis reduction in higher dimensions in 1982, which is called LLL basis reduction algorithm. To begin with, they defined LLL reduced basis.

δ-LLL Reduced Basis: A basis $B = \{b_1, ..., b_n\} \in R_n$ is a δ-LLL Reduced Basis if the following holds:

$$\forall\ 1 \leq i \leq n, j < i, |u_{i,j}| \leq 0.5$$

$$\forall\ 1 \leq i \leq n, ||{b_{i+1}}^* + u_{i,i+1} {b_i}^*||^2 \geq \delta ||{b_i}^*||^2$$

since ${b_i}^*, {b_{i+1}}^*$ are orthogonal above equation becomes

$$||{b_{i+1}}^*||^2 \geq (\delta - u_{i,i+1}^2) ||{b_i}^*||^2$$

The first condition is called the Size condition and the second condition is called Lovász condition.

Size condition: Size condition checks if the vector b_k is almost orthogonal to all the vectors before it. If the condition is not satisfied for b_j, LLL algorithm removes the component of b_j from b_k. This operation reduces the size of the vector as short as possible and keep it as orthonormal as possible to all the $k-1$ vectors.

Lovász condition: This condition makes sure that vector b_{i+1} is sufficiently large compared to its predecessor. In LLL algorithm if this condition is failed, the vectors b_i and b_{i+1} are exchanged. Note that the new ${b_i}^*$ is simply ${b_{i+1}}^* + u_{i,i+1}{b_i}^*$. So we can also say Lovász condition ensures swapping b_i and b_{i+1} does not decrease norm of b_i by too much. the factor δ is typically taken as $\frac{3}{4}$. Also we can see Lovász condition brings ordering in the basis. Shortest vector will be swapped until it reaches first position. It solves Shortest Vector Problem.
The LLL algorithm basically implements these two conditions iterativly.

3 Implementation of LLL algorithm

This is the pseudo code of LLL algorithm

Algorithm 1 : LLL Algorithm

Input $\{b_1, b_2, ..., b_m\}$
Find orthogonal basis matrix for input (GSO)
$k = 1$
while $k < 1$ **do**
  for j from $k-1$ to 0 **do**
    if $|u_{k,j}| \leq 0.5$ (if size condition fails) **then**
      $m \leftarrow$nearest integer of $u_{k,j}$
      $b_i \leftarrow b_i - m * b_j$
      UpdateGramSchmidt$(b_1, b_2, ..., b_m)$
    end if
  end for
  if $||{b_{i+1}}^*||^2 \geq (\delta - u_{i,i+1}^2)||{b_i}^*||^2$ **then**
    $k \leftarrow k + 1$
  else
    swap(b_i, b_{i+1})
    UpdateGramSchmidt$(b_1, b_2, ..., b_m)$
  end if
end while

Algorithm 2 : LLL Algorithm Improved

Input $\{b_1, b_2, ..., b_m\}$
Find orthogonal basis matrix for input (GSO)
$k = 1$
while $k < 1$ **do**
  for j from $k-1$ to 0 **do**
    if $|u_{k,j}| \leq 0.5$ (if size condition fails) **then**
      $m \leftarrow$nearest integer of $u_{k,j}$
      $b_i \leftarrow b_i - m * b_j$
    end if
  end for
  if $||{b_{i+1}}^*||^2 \geq (\delta - u_{i,i+1}^2)||{b_i}^*||^2$ **then**
    $k \leftarrow k + 1$
  else
    swap(b_i, b_{i+1})
    update ${b_i}^*$ and ${b_{i+1}}^*$
  end if
end while

In the original code, After the correction due to size condition we are updating the orthogonal matrix. But we have observed the orthogonal matrix does not change due to the operation $b_i \leftarrow b_i - u_{k,j} * b_j$ So we have updated the code so that orthogonal matrix calculation is skipped for size condition. Also, For orthogonalization if Lovasz condition fail, we only need to update k-1 and kth vectors in the orthogonal matrix since only $(k-1)^{th}$ and k^{th} vectors were swapped. This have improved runtime performance by a heavy margin.

4 Runtime Analysis

It's a polynomial time approximation algorithm. Given a basis with d linearly independent n-dimensional integer coordinates vectors, algorithm calculates reduced lattice basis in time $O(d^5 n log^3(B))$, where B is largest vector in basis under Euclidean norm.

For each size of matrix, we will create an array of random integers from the range [-500, 500] uniformly. Then do LLL reduced basis algorithm and note the runtime after repeating the process 100 times. We change the size in steps of 2 and after crossing 50, we will change in steps of 5 (Because it's too time consuming).

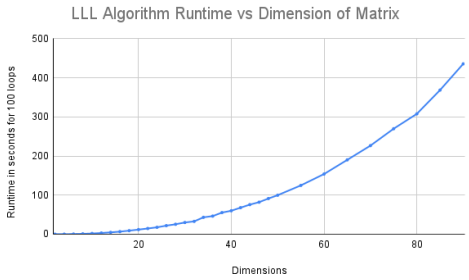

Figure 1: Runtime vs Dimension

For calculating the exponent by which runtime varies with the dimension, let us try to plot *loglog* plot of previous plot. We can calculate the exponent from the slope.

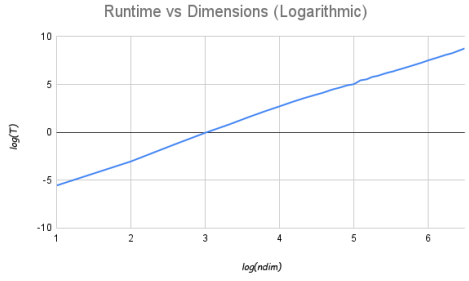

Figure 2: Runtime vs Dimension (*loglog*)

Slope was around 2-2.5 (Worst case is 6 theoretically)

5 Perturbation Analysis

For perturbation analysis, we will add an array of random numbers devised from Gaussian distribution with zero mean and analyze for different standard deviations. We have to keep in mind that it's an integer algorithm, so we need to round it to nearest integers after perturbing. Set of deviations we tried are 5, 50, 100, 150.

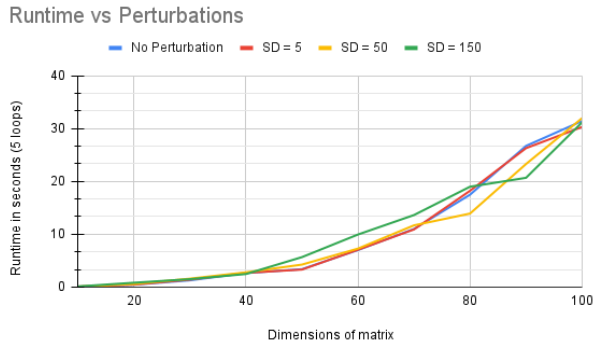

Figure 3: Runtime vs Perturbations (5 loop average)

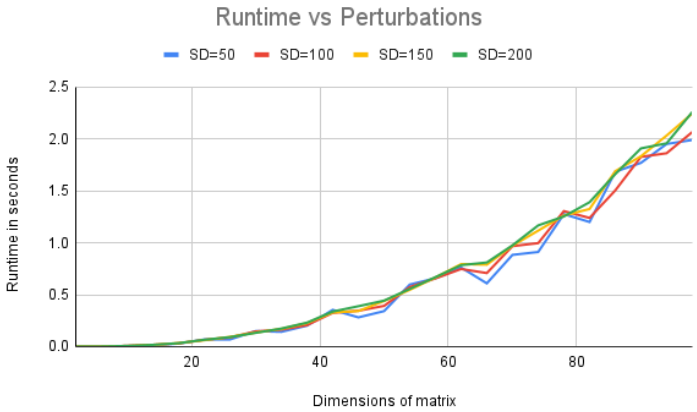

Figure 4: Runtime vs Perturbations (100 loop average)

As we can observe, Algorithm runtime not only depends on dimensions, but it also depends heavily on the relation between vectors (Degree of Orthogonality). If also depends on initialization of random numbers. If the vectors of integers generated are close to orthogonality, it takes less time, otherwise more. From the above graph, we can observe that runtime does not change much for perturbation with standard deviation 5, but for deviation of 150, as there is a possibility of drastic degree of orthogonality change, runtime changes a lot. And also runtime variation in 5 loop average is more compared to change in 100 loop average as if you randomly generate Gaussian random variables 100 times their sum tend to mean, so change is less in runtime (Compared to 5 loop case).

6 Conclusion

In this paper, we introduced an algorithm, LLL basis reduction algorithm, for approximating the shortest vector in higher dimensional space in polynomial time and analyzed it's performance. Although the algorithm was developed in early 1980s, it still has various applications in number theory, integer programming and cryptography because of its performance and accuracy. The appearance of LLL algorithm inspired new development of cryptography since the existing encryption systems can be easily broken by LLL algorithm. Also, it is a fundamental algorithm for solving lattice problems.
The worst-case behavior of LLL is reasonably well understood, and it turns out that the analysis above is tight in the worst-case. However, according to some extensive experiments, for "typical" lattices, the LLL algorithm (and its generalizations) appear to behave much better than the worst-case analysis suggests. Although the dependence on the dimension is still exponential, the base of the exponent is much smaller. Explaining this phenomenon, even heuristically, is still an open question. Another outstanding open question is to improve on LLL and its generalizations for special families of lattices.