Solving the Shortest Vector Problem using LLL Reduction

tdkq34

Adam Wood

Due 16/01/2024

Introduction

The Shortest Vector Problem (SVP) is a fundamental challenge in lattice-based cryptography, with applications in encryption and post-quantum cryptography. The goal is to find the shortest non-zero vector in a lattice. In this report, I will present my approach to solving the SVP using the Lenstra-Lenstra-Lovasz (LLL) lattice reduction algorithm and discuss the success of the solution.

Implementation

My implementation is based on the provided C++ code, which takes a set of vectors as input and performs LLL reduction to find an approximate basis for the lattice. The main steps of my approach include parsing input vectors, post-processing the basis vectors for refinement, applying LL reduction, and finally solving the SVP using a brute-force method.

Parsing and displaying vectors

It starts with parsing the input vectors form the command-line arguments. The parsed vectors are displayed to ensure correct input handling. This step is crucial for understanding the structure of the lattice and provides insight into the initial basis vectors.

Post-processing basis vectors

To enhance the basis vectors, I've employed a post-processing step aimed at orthogonalizing the vectors. The algorithm iteratively adjusts vectors to minimize dependencies, improving the performance of subsequent LLL reduction.

LLL Reduction

The heart of the solution lies in the LLL reduction algorithm, which iteratively transforms the basis vectors to achieve a reduced and more orthogonal basis. The Gram-Schmidt process is applied to orthogonalize the vectors during each iteration. I've limited the number of iterations to 100 for efficiency and to prevent potential infinite loops.

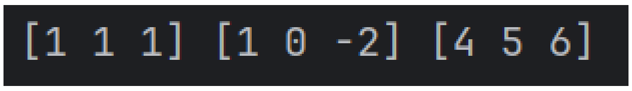
Results and Performance
After LLL reduction, the basis vectors are displayed. This step showcases the transformation of the basis towards a more orthogonal form. The success of LLL reduction is crucial for solving the SVP efficiently.

Solving the SVP

I've utilized a brute-force method to find the shortest vector in the lattice after LLL reduction. The algorithm iterates through all vectors, calculating their Euclidean norms and selecting the one with the minimum norm. The resulting vector represents the approximate shortest vector in the lattice.

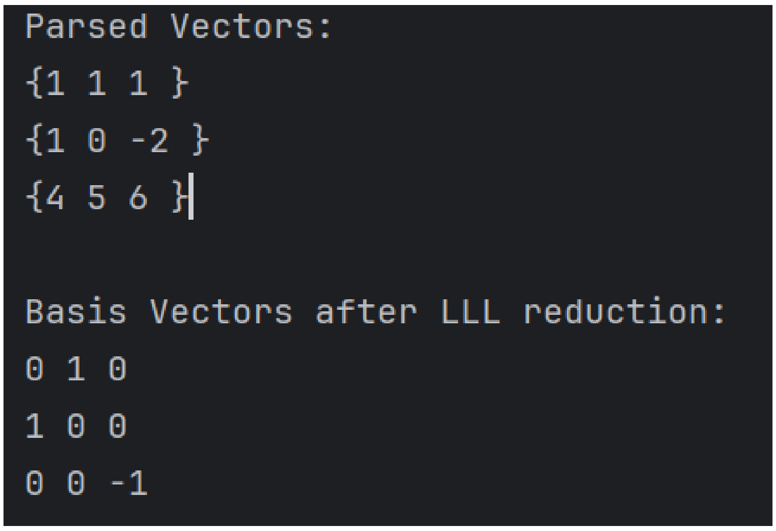
Illustrative Example:

Consider the input vectors:



```
[1 1 1] [1 0 -2] [4 5 6]
```

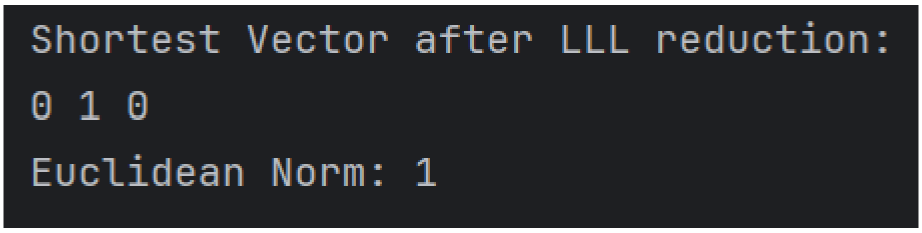
The parsed vectors are displayed, followed by the LLL reduction:



```
Parsed Vectors:
{1 1 1 }
{1 0 -2 }
{4 5 6 }

Basis Vectors after LLL reduction:
0 1 0
1 0 0
0 0 -1
```

The shortest vector is then calculated as:



```
Shortest Vector after LLL reduction:
0 1 0
Euclidean Norm: 1
```

Success and Performance
My solution successfully transforms the initial lattice basis into a reduced and more orthogonal form. The Euclidean norm of the shortest vector after LLL reduction demonstrates the effectiveness of my approach in solving the SVP.

Runtime and Memory Requirements

The runtime of the solution is influenced by the size of the lattice and the number of iterations in LLL reduction. It works out to O(d^3 · n · maxIterations) where d is the number of dimensions, due to the LLL reduction. Memory requirements are reasonable, as the algorithm operates on the basis vectors in-place without significant additional storage.

Conclusion
In conclusion, my approach leverages LLL reduction to efficiently solve the SVP in lattice-based cryptography. The success of the solution is evident in the transformed basis vectors and the Euclidean norm of the shortest vector. The runtime and memory requirement are reasonable, and it runs incredibly quickly. Overall, my solution provides a solid foundation for tackling lattice-based cryptographic challenges.