000828625 - ldzc78 - CC CW - 2021

I. Introduction

The main heuristic used was a genetic algorithm. Within this other heuristics and optimizations were also used to maximise the possibility of creating a shorter vector.

II. SVP - design choices

A. Lattice implementation

Instead of storing each vector as simply a 1 dimensional array, I instead stored each as a coefficient matrix that when multiplied by the basis matrix, the rows of the resulting matrix summed together to produce the overall vector.

This was effective as a vector basis' relatively smaller coefficients could be operated on instead of the overall vector in sieving by averages and differences. This also saved computations as the matrix multiplications now only had to occur when a euclidean norm for a vector needed to be found. This also guaranteed that every vector computed with integer coefficients was guaranteed to be a valid point in that lattice

B. Genetic algorithm framework

A genetic algorithm method was used for the framework of the implementation. The SVP lent itself well to it as a vector in the lattice could be represented by the vector's basis coefficients. Basis coefficients became the modifiable parts of the vector which could be changed by the genetic algorithm to create shorter vectors.

less desirable coefficients being bred out and the more desirable ones (leading to shorter vectors) being kept and having a higher chance of being passed forward to the next generation of vectors. Therefore though the populations shorter vectors would be produced

C. breeding

Each generation the population is scored (i.e. the euclidean norm of the vector) with the lower scores having a higher chance to be bred with each other. Breeding produces two children from the parents. One though modified sieving by averages and the other form modified sieving by differences (see Heuristics). Each generation is also supported by a small percentage of randomised vectors to add more diversity to the population that also helps with exploring the vector space to a greater extent.

D. Mutations

Each new vector created though the breeding process has a chance of one of two mutations occurring: Two random coefficients could change places to effect each others corresponding basis vector A random coefficient could be changed by a small integer amount These mutations allow for exploration of the vector space to find new shorter vectors. This exploration also helps the algorithm escape local minima

These mutations had a fixed rate of occurring, although this could be changed so that more mutations occurred when the population is stuck in a local minima to try and explore enough to find better tours outside this local minima.

Note: mutation can be turned off so that new generations are only a product of breeding. this removes any randomisation from mutations. Shortest vector submitted was found without mutations.

III. heuristics

A. Basis change

Basis change A roughly orthogonal basis was found from the original basis. This was done so that the shortest vector would have a better chance of being closer to the origin of the lattice, meaning that the coefficients of the basis vectors (creating the vector) would be relatively smaller. The increased chance of the coefficients being smaller allowed for a smaller search space of vector coefficients to find the shortest vector. This roughly orthogonal basis change is found by running the LLL [1] algorithm with a delta of 0.3 and using the Gram–Schmidt algorithm as a subroutine.

implementation in Python is based upon the pseudocode from ”An introduction to mathematical cryptography” [2]

This orthogonal basis was used in the genetic algorithm, but when the euclidean norm and coefficients for each of the vectors were calculated the vectors basis was changed back to the original basis (provided for us in latticeBasis.txt).

This roughly orthogonal basis change is found by running the LLL algorithm with a delta of 0.3 and using the gram Schmidt algorithm as a subroutine.

B. Modified sieving by differences

This slightly modifies sieving by differences by running sieving by differences on the vector in question with every other vector in the population. The shortest vector found in this finds is taken as the overall result

IV. optimisations

A. Orthogonal basis

orthogonalisation of basis makes it more likely for the shortest vector to be closer to the origin and thus the search space is likely to be smaller. thus the initially randomly generated population of vectors coefficients can be bounded within a certain range that is more likely to be closer to the shortest vector, allowing shorter vectors to be found much faster

B. *Coefficients of LLL algorithm*

The mu table (coefficients of how different each vector in a basis are different to one another, based on the dot product) is not completed in full for each pair of basis vectors. Instead they are created dynamically when needed in the LLL algorithm. This saves a substantial amount of computation when creating the near orthogonal basis

V. Conclusion

A. *Limitations*

1) LLL: The reason for the conversion back to the original basis for finding the euclidean norm was that the new basis has a chance of creating vectors where the euclidean norm is so small that they are rounded to 0.0 . This was because when running the LLL algorithm certain parts were changed so that the algorithm worked with integers instead of floats to create a valid integer basis. This will have introduced errors in the new basis that affects how linearly independent each of the vectors in the basis are from each other.

In the LLL algorithm, the variable delta controls the strength of the reduction in the new basis and thus extent to which the created basis vectors are reduced and are orthogonal to one another.

The condition for the LLL algorithm to move onto the next vector in the basis is that the current vector's magnitude is a certain ratio greater than the previous basis vector's magnitude (this ratio is controlled by delta). A greater delta leads to a longer run time as the algorithm loops along each of the vectors trying to make them roughly orthogonal to each other. Theoretically this has polynomial time complexity when delta = (0.25, 1] , but in practicality a delta larger than 0.4 took far too long to run to be used. Therefore a limitation that could be rectified with more time would be to run LLL with a larger delta to create a basis where the basis vectors were reduced more.

B. *shortest vector found*

u = [[8.], [14.], [32.], [-16.], [1.] [-23.], [-24.], [-30.], [37.], [9.], [14.], [-2.],])

norm = 72.0832851637

x = [[-2], [2], [-1], [1], [-3], [1], [2], [2], [-2], [1], [2], [-3]])

References

[1] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261(ARTICLE):515–534, 1982.

[2] Joseph H Silverman, Jill Pipher, and Jeffrey Hoffstein. *An introduction to mathematical cryptography*. Springer, 2008.