**An Implementation of List Sieving in C to Solve The Exact Shortest Vector Problem.**

The shortest vector problem is an NP-hard problem associated with lattice based cryptography. This report documents the implementation and analysis of a Heuristic solution to this problem.

1. Algorithm

List sieving is in the category of heuristic algorithms for solving the exact Shortest Vector Problem (SVP). This approach involves random sampling of lattice vectors and exploits the characteristic that any linear combination of lattice vectors is also a lattice vector. The method iteratively reduces sampled vectors until a shorter vector cannot be identified once 2cn samples have been made. As stated by (Micciancio, 2021), List Sieving has an optimal time complexity of O(21.325n), where n denotes the dimension of the lattice basis.

* 1. Alterations
     1. Addition of CosScore subroutine

The speed in which the shortest vector can be found is dependent on the orthogonality of the basis (Galbraith, 2014 ). More orthogonal bases can be solved by taking fewer sample vectors. To address this the CosScore function computes the mean cosine similarity of the input basis and maps it to an integer c – used by ListSieve to bound the number of sample vectors to be produced. The aim of doing this is to save time where more vectors are sampled than are necessary for low orthogonality bases.

* + 1. Alteration to Sample Subroutine

Rather than creating a random perturbation vector (e) to subsequently compute a vector (p) within the lattice's parallelepiped, Sample produced random linear combinations of the basis vectors. These linear combinations are generated in such a way that the length falls within the range of 0 to 2 times the length of the longest basis vector. This modification maintains ListSieve's ability to obtain a sample vector approximately as long as the longest basis but offers a simpler implementation.

* + 1. Alteration to ListSieve

ListSieve aims to find a vector such that it has a Euclidean norm less than μ. At the entry point to ListSieve μ is set to the Euclidean norm of the shortest basis vector. When a new vector has been found with a length shorter than μ, ListSieve recurses into this length until the number of samples has reached its limit. Once the limit has been reached the current value for μ is the shortest vector.

1. Optimisation Techniques

List Sieving stores all reduced lattice vectors in an array (L), as the function runs L becomes very large, to minimise the impact of memory reallocation on running time it was decided that memory for L should be allocated and deallocated outside the ListSieve function. To further reduce the impact of memory management on running time, where possible blocks of memory were allocated once before the loop they are needed in and reused until termination. As a rule, memory management for specific functions was performed in their caller functions (where possible) to improve pointer safety and simplify deallocation.

1. Performance analysis

Figure 1 shows a greater than exponential increase in running time from eight to nine dimensions.



Figure : Timing results for orthogonal unit basis vectors. data was produced by running and timing the program with a bash script, from the output file data was read and plotted with python.

To find the cause of this unexpected increase, the C “time.h” header was used with a bash script to collect data on the running times of the Sample and Reduce subroutines every time they are run on the same bases as above. Figure 2 summarises the data collected. From figure 2, it can be concluded that the sample subroutine is responsible for disproportionately increasing the running time.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Reduce | | | | Sample | | | |
| Dimension | **Mean running time** | **Standard deviation** | **Min** | **Max** | **Mean running time** | **Standard deviation** | **Min** | **Max** |
| 2 | 2.00x10-7 | 0 | 2.00x10-7 | 2.00x10-7 | 3.50x10-6 | 1.50x10-6 | 2.00x10-6 | 5.00x10-5 |
| 3 | 3.00x10-6 | 8.94x10-7 | 2.00x10-7 | 4.00x10-6 | 1.94x10-5 | 1.57x10-5 | 3.00x10-6 | 4.90x10-5 |
| 4 | 4.25x10-6 | 8.29x10-7 | 2.00x10-7 | 5.00x10-6 | 5.23x10-5 | 4.94x10-5 | 3.00x10-6 | 1.58x10-4 |
| 5 | 7.30x10-6 | 2.02x10-6 | 4.00x10-6 | 1.20x10-5 | 5.62x10-4 | 4.95x10-4 | 2.20x10-5 | 2.40x10-3 |
| 6 | 8.74x10-6 | 2.98x10-6 | 4.00x10-6 | 1.40x10-5 | 2.05x10-3 | 1.95x10-3 | 2.40x10-5 | 1.10x10-2 |
| 7 | 1.90x10-5 | 1.34x10-5 | 4.00x10-6 | 7.80x10-5 | 1.43x10-2 | 1.27x10-2 | 3.91x10-4 | 6.21x10-2 |
| 8 | 3.37x10-5 | 1.85x10-5 | 4.00x10-6 | 1.10x10-4 | 8.72x10-2 | 9.08x10-2 | 8.20x10-4 | 0.566134 |
| 9 | 7.07x10-5 | 4.04x10-5 | 4.00x10-6 | 2.60x10-4 | 0.680 | 0.680 | 4.98x10-4 | 4.445898 |

Figure : Summary of running time for Reduce and Sample subroutines. data was produced using C's 'time' library and a bash script to run on different inputs.

1. Summary

Overall, I have produced a program that performs better on space requirements than on time requirements. To improve my time complexity I would use the original form of the sample subroutine to create an array of sample vectors. To further improve running time I would implement a solution whereby the sample array is split into smaller chunks and ListSieve is run simultaneously by on these smaller arrays using a common array L to store reduced lattice vectors.

# Works Cited

Galbraith, S. (2014 , January 16). *Chapter 17 Lattice Basis Reduction.* Retrieved November 2023 , from University of Auckland: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjMn7LLjduDAxXAUUEAHXbfCHwQFnoECDkQAQ&url=https%3A%2F%2Fwww.math.auckland.ac.nz%2F~sgal018%2Fcrypto-book%2Fch17.pdf&usg=AOvVaw0YS3PSAoRHNTCuTBp6TuLZ&opi=89978449

Micciancio, D. (2021). *CSE 206A: Lattice Algorithms and Applications.* Retrieved November 2023, from University of California San Diego: https://cseweb.ucsd.edu/classes/fa21/cse206A-a/LecSieve.pdf