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 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 after 2^{cn} samples have been made. As stated by (Micciancio, 2021), List Sieving has an optimal time complexity of $O(2^{1.325n})$, where n denotes the dimension of the lattice basis.

1.1. Alterations

1.1.1. 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 therefore be solved by taking fewer sample vectors. To address this the CosScore function computes the mean cosine similarity of the input basis vectors 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 and memory where more vectors are sampled than are necessary for high orthogonality bases.

1.1.2. 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 is less than twice 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.

1.1.3. 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 of μ is the shortest vector.

2. 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.

3. Performance analysis

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

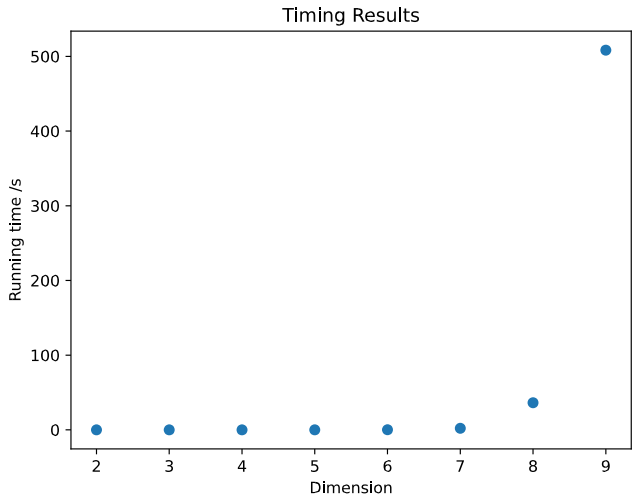

Figure 1: Timing results for orthogonal unit basis vectors. data was produced by running and timing the program with a bash script, from the output file data was read and plotted with python.

The root of this issue was suspected to be due to either the Sample or Reduce subroutines as they are executed the most. Data collected from timing these functions is summarised in Figure 2 From figure 2, it can be concluded that the sample subroutine is responsible for disproportionate increase in 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 2: 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 input Lattice Bases.

4. Summary

Overall, I have produced a program that correctly solves SVP up to ten dimensions. To improve time complexity, I would use the original form of the sample subroutine to create an array of sample vectors. This array could then be split into smaller chunks optimised for using multi-threading to perform the List Sieve algorithm simultaneously. Developing a communication system whereby the current shortest vector updated in every thread as soon as it's discovered for one would also improve extensibility by increasing the number of samples processed per unit of time. For large and complicated lattices this would speed up the solution.

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