Programming Paradigms Report - Shortest Vector Problem (SVP)

Alex Dinan, vwzn63

1 Initial Approach

Initially, I implemented a naive brute-force search algorithm. This entailed iteratively trying all integer coefficient combinations within a certain range. However, as input dimensionality increased, this approach quickly took an unreasonable time to terminate. Also, the range I was searching in was arbitrary and did not guarantee an optimal solution.

2 Choice of exact SVP algorithm

Through research [1], I discovered 3 main approaches for solving SVP exactly:

Algorithm	Time complexity	Space complexity
Enumeration	super exponential, $n^{\mathcal{O}(n)}$	polynomial, $\mathcal{O}(n^2)$
Voroni cell	doubly exponential, $\mathcal{O}(2^{2n})$	exponential, $\mathcal{O}(2^n)$
Seiving	exponential, $\mathcal{O}(2^n)$	exponential, $\mathcal{O}(2^n)$

I chose to use enumeration. It has the best space complexity and despite an inferior asymptotic time complexity, it is the fastest approach in practice, particularly for moderately small dimensions. Also, there are numerous heuristics and optimisations which further improve running time.

Specifically, I implemented the Schnorr-Euchner algorithm [2]. This improves performance by enumerating lattice points in an optimal order using a zig-zag search pattern.

3 Lattice reduction

I incorporated Lenstra-Lenstra-Lovasz (LLL) lattice reduction [3] as a pre-processing step. This polynomial time algorithm produces an equivalent lattice with shorter, more orthogonal basis vectors which can then be used as an input to the enumeration algorithm. This significantly improves the efficiency of subsequent enumeration to $2^{\mathcal{O}(n^2)}$ by allowing a smaller initial search radius to be used.

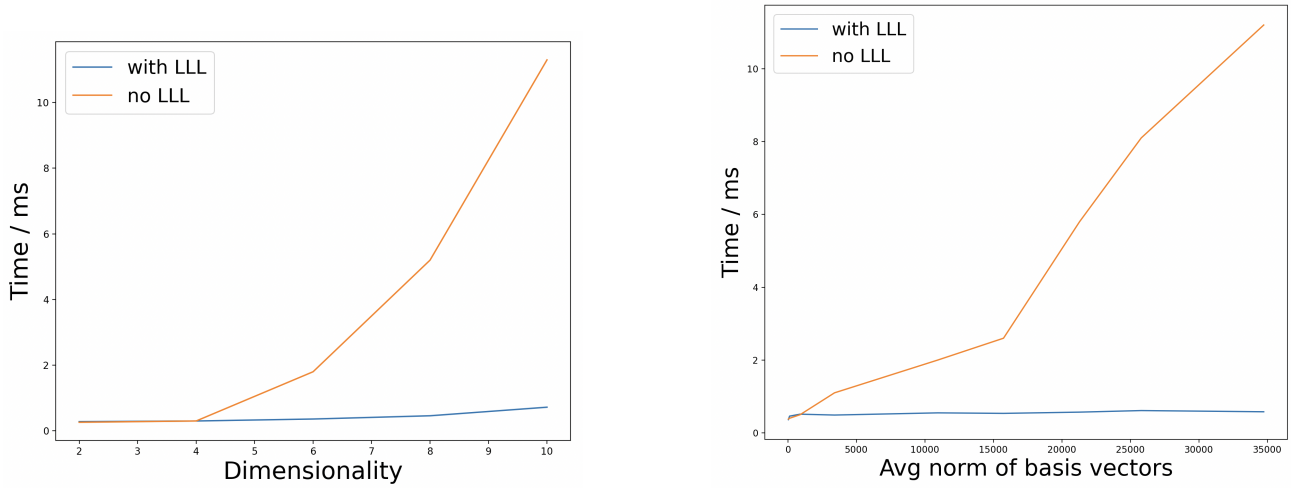


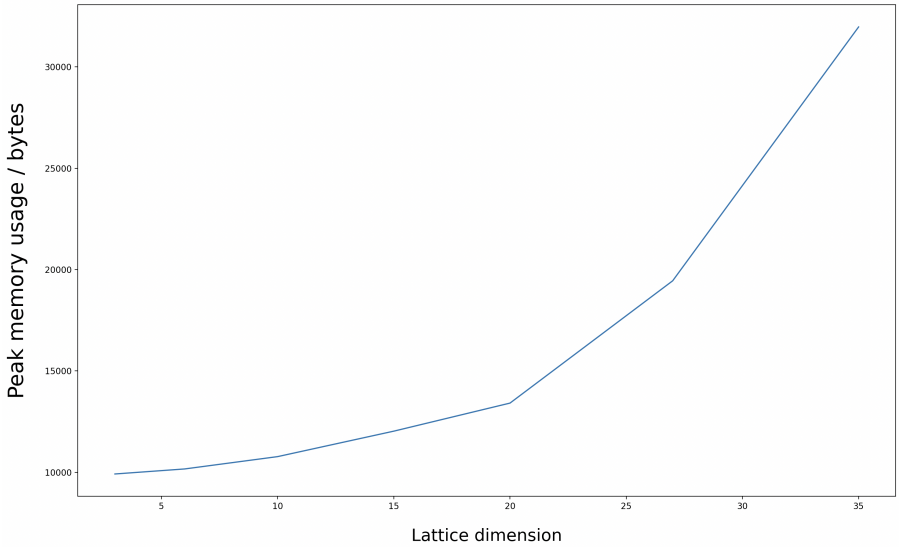
Figure 1: Runtime performance with/without LLL for nearly parallel basis vectors

When the lattice basis is relatively orthogonal, LLL has little impact on runtime and actually runs more slowly for small vector norms and dimensions. However, as shown by Figure 1, it provides significant performance increases for more parallel vectors.

4 Optimising space complexity

Aside from by implementing enumeration, memory requirements were minimised using the Valgrind tool. This allowed me to efficiently allocate/deallocate memory and avoid leaks. It also aided the discovery of an issue where I was using dynamically allocated lists/vectors prior to initialisation, leading to undefined behaviour.

I also used pointers as arguments to functions and minimised the number of parameters to decrease memory overhead. Furthermore, I avoided the use of recursion. For example, when checking the linear (in)dependence of a set of vectors, I initially implemented a determinant test recursively and later optimised this to Gaussian elimination/row reduction [4].



From the figure above, peak memory usage seems to grow in a quadratic fashion with lattice dimensionality, reinforcing the information presented in Section 2 and suggesting that the enumeration algorithm has been implemented correctly.

5 Handling invalid inputs

- Before parsing the input, I ensure that the argument count is valid; In this case it must be a perfect square.
- The parsing process checks that '[' / ']' are used correctly and no non-numerical characters are used innapropriately.
- Finally, the basis vectors are checked for linear independence.
- Exception handling is used throughout the program e.g. for unsuccessful memory allocation and file opening errors.

In each of the above cases, a unique, helpful message is displayed and the program terminates.

6 Overall system success

Overall, the system performs well in solving the exact shortest vector problem. It handles invalid input and incorporates error handling. The table below summarises some performance measures for typical lattice examples randomly generated within the ranges specified.

Dimension	**Vector component range**	**Avg. execution time**	**Avg. peak memory usage**
10	[-1,000,000 , 1,000,000]	$558\mu s$	$10,776$ bytes
50	[-100 , 100]	$0.850s$	$62,424$ bytes
75	[-20 , 20]	$12.8s$	$140,448$ bytes

References

[1] Lattice cryptography. `https://cseweb.ucsd.edu/~daniele/LatticeLinks/SVP.html`.

[2] M Yasuda. A survey of solving svp algorithms and recent strategies for solving the svp challenge. *In: Takagi, T., Wakayama, M., Tanaka, K., Kunihiro, N., Kimoto, K., Ikematsu, Y. (eds) International Symposium on Mathematics, Quantum Theory, and Cryptography. Mathematics for Industry*, vol 33:275, 2021.

[3] Lenstra–lenstra–lovász lattice basis reduction algorithm. `https://en.wikipedia.org/wiki/Lenstra%E2%80%93Lenstra%E2%80%93Lov%C3%A1sz_lattice_basis_reduction_algorithm`.

[4] Gaussian elimination. `https://mathworld.wolfram.com/GaussianElimination.html`.