1 Shortest Vector

- *Algorithm:* Lenstra-Lenstra-Lovász Algorithm (algo. 1)
- *Input:* A basis $B = [b_1, b_2, ..., b_n]$, a norm calculation method $\|v\|$
- *Complexity:* $\mathcal{O}(n^6 \log(max(\|b_i\|))^3)$
- *Data structure compatibility:* N/A
- *Common applications:* Linear Algebra, Cryptosystems

Problem. Shortest Vector

Given a basis B for a vector space and a function to calculate vector norm(length), find the shortest distance between lattice points in that basis.

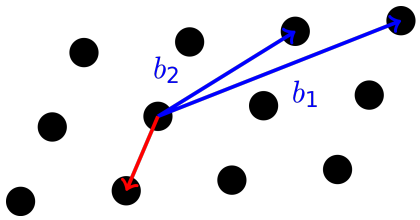
Description

From the name of the name "Shortest Vector Problem"(denoted as SVP), one can easily figure out that the problems requires to find the shortest vector in a vector space. However the form of the vectors to be chosen is specified: They should be connecting the "lattice points" in that vector space.

Suppose the basis of the vector space is B, the lattice points are points who has a vector when connected with the original point in the following form

$$v = Bz$$

where z is a vector of integers. It is called so because the points align like a lattice. An example from the 2-dimension Euclidean space can show that.



To simplify this problem, just change it to finding the shortest distance from original to any non-original lattice points in that vector space. Another simplification to make is to assume that the dimension of the given basis is the same to the dimension of the vector space(which means B is square). However, the hardness of the original problem is still $\mathcal{NP}$-hard

In many cases the problem can be further simplefied to be γ-approximation SVP, denoted as SVP_γ. This approximated problem intends to find a vector in the lattice whose length is at most $\gamma\lambda$, where γ is a specified number larger than 1 and λ is the theoretical shortest length. This simplication make the requirement of the optimization to be looser and in that case, there is a polynomial-time algorithm called Lenstra-Lenstra-Lovász(LLL) Algorithm, which is designed for $\gamma = 2^{(n-1)/2}$

The first step is to orthogonalize the basis. Orthogonalization of a basis B is to build another basis B^* which spans the same space(but possibly not same lattice) with B, and the vectors in B^* are all orthogonal(perpendicular) to each other. Here the Gram-Schimidt algorithm (whose introduction is offered in reference) will be used for orthogonalization.

Next is to create a coefficient matrix μ, in which $\mu_{i,j}$ means the weight of b_j^* in the vector b_i $(1 \leq j < i \leq n)$

The following step is to adjust the original basis B using column opeartion to make the B^* and μ to maintain the LLL-Condition:

- For $1 \leq j < i \leq n$, $|\mu_{i,j}| \leq 0.5$
- Specify a real number $\delta \in (0.25, 1)$ (usually 0.75), then for $k = 2, ..., n, \delta \|b_{k-1}^*\|^2 \leq \|b_k^*\|^2 + \mu_{k,k-1}^2 \|b_{k-1}^*\|^2$

The first condition can be satisfied by subtracting $|\mu_{i,j}|b_j$ from b_i, and the second condition can be satisfied by just swapping columns. After the LLL-conditions are satisfied for all columns, the first column is just the vector we want to find. The mathematical proof can be found in the first reference url.

Algorithm 1: Lenstra-Lenstra-Lovász Algorithm

Input : A basis B of dimension n, a norm $\|v\|$

Output: A vector in the lattice whose length is at most $2^{(n-1)/2}\lambda$

ortho $\leftarrow$GramSchimidt$(B) = \{b_1^*, b_2^*, ...b_n^*\}$;
k $\leftarrow 1$;
$\delta \leftarrow 0.75$;
for $j \leftarrow 1$ *to* $n-1$ **do**
  for $j \leftarrow j+1$ *to* n **do**
    $\mu_{i,j} \leftarrow \frac{<b_i, b_j^*>}{<b_j^*, b_j^*>}$;
  end for
end for
while $k < n$ **do**
  for $j \leftarrow n-1$ *downto* 1 **do**
    if $|\mu_{k,j}| > 0.5$ **then**
      $b_k \leftarrow b_k - |\mu_{k,j}|b_j|$;
      ortho $\leftarrow$GramSchimidt$(B) = \{b_1^*, b_2^*, ...b_n^*\}$;
      for $j \leftarrow 1$ *to* $n-1$ **do**
        for $j \leftarrow j+1$ *to* n **do**
          $\mu_{i,j} \leftarrow \frac{<b_i, b_j^*>}{<b_j^*, b_j^*>}$;
        end for
      end for
  end for
  if $\|b_k^*\| \geq (\delta - \mu_{k,k-1}^2)\|d_{k-1}^*\|$ **then**
    $k \leftarrow k+1$;
  else
    Swap b_k and b_{k-1};
    ortho $\leftarrow$GramSchimidt$(B) = \{b_1^*, b_2^*, ...b_n^*\}$;
    for $j \leftarrow 1$ *to* $n-1$ **do**
      for $j \leftarrow j+1$ *to* n **do**
        $\mu_{i,j} \leftarrow \frac{<b_i, b_j^*>}{<b_j^*, b_j^*>}$;
      end for
    end for
    $k \leftarrow max(k-1, 1)$;
  end if
end while
return b_1

References

- https://web.eecs.umich.edu/~cpeikert/lic13/lec02.pdf

- https://en.wikipedia.org/wiki/Lattice_problem
- https://www.math.hmc.edu/calculus/tutorials/gramschmidt/gramschmidt.pdf
- Lenstra, A. K.; Lenstra, H. W., Jr.; Lovsz, L. (1982). "Factoring polynomials with rational coefficients".