The LLL reduction algorithm

Suppose that we are given a basis $b_1, \ldots, b_k$ of a lattice L, and we wish to find a reduced basis, as defined by the conditions (1) and (2) of §2.6. The first

thing that we do is to compute the associated Gram–Schmidt basis $b_1^*, ..., b_k^*$ and the constants μ_{ij}. Let $m = 2$. (We'll see the significance of m later.) If $k = 1$ then we are finished. Otherwise we do the following:

Step A
Reduce $|\mu_{m,m-1}|$ to at most $\frac{1}{2}$, by adding a suitable multiple of b_{m-1} to b_m.

For example, take

$$b_1 = \begin{pmatrix} 7 \\ 19 \end{pmatrix}, \quad b_2 = \begin{pmatrix} 6 \\ 16 \end{pmatrix}.$$

Then $b_1^* = b_1$ and

$$\mu_{2,1} = \frac{1}{49 + 361}\begin{pmatrix} 7 & 19 \end{pmatrix}\begin{pmatrix} 6 \\ 16 \end{pmatrix} = \frac{346}{410} = \frac{173}{205} > \frac{1}{2}.$$

So our basis is not yet LLL-reduced. Moreover

$$b_2^* = \begin{pmatrix} 6 \\ 16 \end{pmatrix} - \frac{173}{205}\begin{pmatrix} 7 \\ 19 \end{pmatrix} = -\frac{1}{205}\begin{pmatrix} 19 \\ -7 \end{pmatrix}.$$

To carry out Step A, we replace b_2 by

$$b_2 - b_1 = \begin{pmatrix} -1 \\ -3 \end{pmatrix}.$$

Note that b_1^* and b_2^* are unchanged.

Step B
If condition (2) holds, with $i = m$, then go to Step C.
Otherwise, swap b_{m-1} and b_m; and make appropriate modifications to the Gram–Schmidt basis, and then to the μ_{ij}.
If $m > 2$ replace m by $m - 1$.
Return to Step A.

Note that condition (2) requires us only to know the lengths of the b_i^*, so one needs only to keep track of their current lengths, and of the μ_{ij}. If a swap is performed in Step B, then the lengths of b_{m-1}^* and b_m^* are interchanged, but no others, and the μ_{ij} may change for $i \geq m - 1$.

Step C
For $j = m-2, m-3, \ldots, 1$, reduce $|\mu_{m,j}|$ to at most $\frac{1}{2}$, by adding a suitable multiple of b_j to b_m.

Increase m by 1. If $m = k+1$ stop, and otherwise return to Step A.

The significance of the parameter m is that whenever we go to Step A, the vectors $b_1, \ldots, b_{m-1}$ are an LLL-reduced basis for the lattice which they span. Initially m is set to 2, and the aim is to reach $m = k+1$. When going from Step B to Step A, the value of m decreases, unless it is already equal to 2. In contrast, when going from Step C to Step A the value of m increases. The reader will find it helpful to draw a flowchart for the process!!

Since the value of m can go down as well as up it is not obvious that the algorithm terminates. To investigate this we look closely at the way any changes to the basis affect various sublattices. For each $i = 1, \ldots, k$ we define L_i as the sublattice of L spanned by $b_1, \ldots, b_i$.

The first thing to notice is that neither Step A nor Step C change any of the lattices L_i. Consider what happens in Step B. If a swap is performed the only lattice which changes is L_{m-1}. The new value of b^*_{m-1} is

$$\text{old}\, b^*_m + \mu_{m,m-1}.\text{old}\, b^*_{m-1}.$$

The swap is performed only when condition (2) fails, and we see that this implies that the process multiplies $||b^*_{m-1}||^2$ by a factor which is at most $\frac{3}{4}$. However we have

$$\begin{aligned} \Delta(L_{m-1})^2 &= \det((b_1 \ldots b_{m-1})^t (b_1 \ldots b_{m-1})) \\ &= \det((b^*_1 \ldots b^*_{m-1})^t (b^*_1 \ldots b^*_{m-1})) \\ &= \prod_{i=1}^{m} ||b^*_i||^2. \end{aligned}$$

(For the second equality one needs to remember that the matrix relating a basis to its Gram–Schmidt orthogonalization is upper triangular, with 1's on the diagonal.) This equality shows that a swap at Step B has the effect of reducing $\Delta(L_{m-1})^2$ by a factor of at most $\frac{3}{4}$, while leaving $\Delta(L_i)^2$ fixed for all $i \neq m-1$. However the corollary in §2.4 shows that these $\Delta(L_i)$ are all bounded below by a positive constant, so the algorithm must eventually terminate.

An example

As before, we consider the lattice spanned by

$$b_1 = \begin{pmatrix} 7 \\ 19 \end{pmatrix}, \quad b_2 = \begin{pmatrix} 6 \\ 16 \end{pmatrix}.$$

Then $\Delta(L_1)^2 = 49 + 361 = 410$, while

$$\Delta(L_2) = \Delta(L) = \left|\det \begin{pmatrix} 7 & 6 \\ 19 & 16 \end{pmatrix}\right| = 2.$$

Property (c) in §2.7 shows that an LLL-reduced basis would have

$$||b_1||^2 \leq 2^{(k-1)/4}\Delta(L)^{1/k} = 2^{1/4}.2^{1/2} = 2^{3/4},$$

so that our basis is very far from reduced.

We compute

$$b_1^* = b_1, \quad \mu_{2,1} = \frac{173}{205}, \quad b_2^* = \frac{1}{205}\begin{pmatrix} 19 \\ -7 \end{pmatrix}$$

and go to Step A, with $m = 2$.

Step A: Replace b_2 by

$$b_2 - b_1 = \begin{pmatrix} -1 \\ -3 \end{pmatrix},$$

so that $\mu_{2,1}$ is replaced by $\frac{-7-57}{410} = \frac{-64}{410}$, which has absolute value at most $\frac{1}{2}$.

Step B: Condition (2) fails (it must do, since we know that $||b_1||$ is not small enough), so we swap b_1 and b_2 to get

$$b_1 = \begin{pmatrix} -1 \\ -3 \end{pmatrix}, \quad b_2 = \begin{pmatrix} 7 \\ 19 \end{pmatrix}.$$

Observe that we now have $\Delta(L_1)^2 = 10 < \frac{3}{4} \cdot 410$. We return to Step A, still with $m = 2$.

Step A: This time we replace b_2 by

$$b_2 + 6b_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

and go to Step B.

Step B: Condition (2) still fails, unsurprisingly since $||b_1||$ is still too large. So we swap the basis vectors to reach

$$b_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b_2 = \begin{pmatrix} -1 \\ -3 \end{pmatrix}.$$

Again $\Delta(L_1)^2$ has been reduced, and one checks that $\Delta(L_1)^2 = 2 < \frac{3}{4} \cdot 10$. We return to Step A.

Step A: Replace b_2 by

$$b_2 + 2b_1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix},$$

and go to Step B.

Step B: At last condition (2) is satisfied, so we go to step C.

Step C: Increase m to 3, and stop, returning

$$b_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

The knapsack problem

We are given a knapsack which can carry a limited weight and we have a given set of objects of known weights. We want to pack as much into the knapsack as possible.

Example The given weights are $1, 2, 4, 8, 16, 32, 64$ and the knapsack can hold weight 12. There is a unique optimal packing, with weights 4 and 8. Indeed for a knapsack of any given capacity there is a unique optimal packing, which is easy to spot.

In general, if the weights $w_1, \ldots, w_k$ are in "superincreasing order", such that

$$w_i > \sum_{j=1}^{i-1} w_j \quad \text{for } i = 2, \ldots, k,$$

then the knapsack problem is easy: one uses the **greedy algorithm**, packing the heaviest weight possible, then the next heaviest possible, and so on.

However in general the knapsack problem is hard. Indeed it is known to be NP-complete. Given that it is *provably* hard, cryptographers have tried

to devise coding systems based on knapsack problems. Unfortunately, while we know that the knapsack problem in general is hard, it is not so easy to produce, and use, particular examples of hard knapsack problems.