CSE 206A: Lattice Algorithms and Applications Spring 2007

Lecture 4: The LLL Algorithm

Lecturer: Daniele Micciancio *Scribe: Daniele Micciancio*

No efficient algorithm is known to find the shortest vector in a lattice (in arbitrary dimension), or even just computing its length λ_1. A central tool in the algorithmic study of lattices (and their applications) is the LLL algorithm of Lenstra, Lenstra and Lovasz. The LLL algorithm runs in polynomial time and finds an approximate solution $\mathbf{x}$ to the shortest vector problem, in the sense that the length of the solution $\mathbf{x}$ found by the algorithm is at most $\gamma \cdot \lambda_1$, for some approximation factor γ. The approximation factor $\gamma = 2^{O(n)}$ achieved by LLL is exponential in the dimension of the lattice. Later in the course, we will study algorithms that achieve (slightly) better factors. Still, the approximate solutions found by LLL are enough in many applications. For example, in the last lecture we proved that any prime p congruent to 1 modulo 4 can be written as the sum of two squares, but the proof was not effective, in the sense that it didn't give an efficient way to find the two squares. We will show that using LLL one can efficiently find a and b such that $p = a^2 + b^2$. We design and analyze the LLL algorithm in two steps:

1. We first define a notion of "reduced" basis, and show that the first vector of a reduced basis is an approximately shortest vector in the lattice.

2. Next, we give an efficient algorithm to compute a reduced basis for any lattice.

1 Reduced basis

Remember the Gram-Schmidt orthogonalization process:

$$\mathbf{b}_i^* \quad = \quad \mathbf{b}_i - \textstyle\sum_{j<i} \mu_{i,j} \mathbf{b}_j^* \qquad \text{where} \quad \mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle}$$

We define projection operations π_i from $\mathbb{R}^m$ onto $\sum_{j \geq i} \mathbb{R}\mathbf{b}_j^*$:

$$\pi_i(\mathbf{x}) = \sum_{j=i}^{n} \frac{\langle \mathbf{x}, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \mathbf{b}_j^*.$$

Notice that the Gram-Schmidt orthogonalized vectors can be expressed as $\mathbf{b}_i^* = \pi_i(\mathbf{b}_i)$.

We can now define LLL-reduced basis. For reasons that will be clear in the running time analysis of the algorithm, we introduce a real parameter $1/4 < \delta < 1$ and define LLL-reduced basis with respect to δ.

Definition 1 *A basis $B = (\mathbf{b}_1 \ldots \mathbf{b}_n) \in \mathbb{R}^{m \times n}$ is a* ***LLL Reduced Basis*** *with parameter δ if:*

1. $|\mu_{i,j}| \leq \frac{1}{2}$ *for all* $i > j$

2. for any any pair of consecutive vectors $\mathbf{b}_i, \mathbf{b}_{i+1}$*, we have*

$$\delta \|\pi_i(\mathbf{b}_i)\|^2 \le \|\pi_i(\mathbf{b}_{i+1})\|^2.$$

The first condition (usually called "size reduction") is easy to achieve using a variant of the Gram-Schmidt procedure, and it is discussed in details in the next section. In order to understand the second condition it is useful to consider the case when $i = 1$ and $\delta = 1$. For $i = 1$, the projection π_1 is just the identity function (over the linear span of the lattice) and the condition becomes simply $\|\mathbf{b}_1\| \le \|\mathbf{b}_2\|$, i.e., the first two vectors in an LLL reduced basis are sorted in order to increasing length. (Introducing a factor $\delta < 1$, relaxes the increasing condition to hold approximately within a factor δ.) For $i > 1$, the LLL reduced basis definition requires a similar condition to hold for the projected basis $\pi_i(\mathbf{B})$.

Another geometric interpretation of the second condition is the following. Notice that

$$\|\pi_i(\mathbf{b}_{i+1})\|^2 = \|\mathbf{b}^*_{i+1} + \mu_{i+1,i}\mathbf{b}^*_i\|^2 = \|\mathbf{b}^*_{i+1}\|^2 + \|\mu_{i+1,i}\mathbf{b}^*_i\|^2 = \|\mathbf{b}^*_{i+1}\|^2 + (\mu_{i+1,i})^2\|\mathbf{b}^*_i\|^2.$$

So, the second condition in the definition of LLL-reduced basis can be equivalently rewritten as

$$(\delta - \mu^2_{i+1,i})\|\mathbf{b}^*_i\|^2 \le \|\mathbf{b}^*_{i+1}\|^2.$$

So, although the Gram-Schmidt vectors $\mathbf{b}^*_i$ can get shorter and shorter, their length cannot decrease too quickly. Specifically, for any $1/4 < \delta < 1$, if we set $\alpha = \frac{1}{\delta - \frac{1}{4}}$, then

$$\|\mathbf{b}^*_i\|^2 \le \alpha \|\mathbf{b}^*_{i+1}\|^2. \tag{1}$$

For example, if $\delta = 3/4$, then $\alpha = 2$ and each $\|\mathbf{b}^*_{i+1}\|$ is least half the length of $\|\mathbf{b}^*_i\|$. Using Equation 1 repeatedly, we get

$$\|\mathbf{b}^*_1\|^2 \le \alpha^{i-1}\|\mathbf{b}^*_i\|^2 \le \alpha^{n-1}\|\mathbf{b}^*_i\|^2.$$

Since this is true for all $i = 1, \ldots, n$, the first vector in an LLL reduced basis satisfies

$$\|\mathbf{b}_1\| \le \alpha^{(n-1)/2} \min \|\mathbf{b}^*_i\| \le \alpha^{(n-1)/2}\lambda_1$$

where we have used the lower bound $\lambda_1 \le \min_i \|\mathbf{b}^*_i\|$ on the length of the shortest vector in a lattice. In particular, if $\delta = 3/4$, the first vector in an LLL reduced basis is a $\gamma = 2^{(n-1)/2}$ approximate solution to SVP.

In many applications, the length of the shortest lattice vector λ_1 is not known, the its length can only be estimated using Minkowski's theorem $\lambda_1 \le \sqrt{n}\det(\mathbf{B})^{1/n}$. Combining the bound $\|\mathbf{b}_1\| \le \alpha^{(n-1)/2}\lambda_1$ with Minkowski's theorem we get $\|\mathbf{b}_1\| \le \sqrt{n}\alpha^{(n-1)/2}\det(\mathbf{B})^{1/n}$. A stronger bound can be obtained relating the length of $\mathbf{b}_1$ in an LLL reduced basis directly to the determinant of the lattice as follows. Take the product of (1) for $i = 1, \ldots, n$, to get

$$\|\mathbf{b}_1\|^n \le \prod_i \alpha^{(i-1)/2}\|\mathbf{b}^*_i\| = \alpha^{n(n-1)/4}\det(\mathbf{B}).$$

So, we have

$$\|\mathbf{b}_1\| \le= \alpha^{(n-1)/4}\det(\mathbf{B}^{1/n}),$$

which can be interpreted as a weak (but algorithmic) version of Minkowski's theorem.

Our analysis of LLL reduced basis is summarized in the following theorem.

Theorem 2 *For any $1/4 < \delta \leq 1$, if $\mathbf{B}$ is a δ-LLL reduced basis, then*

- $\|\mathbf{b}_1\| \leq \alpha^{(n-1)/2}\lambda_1$
- $\|\mathbf{b}_1\| \leq \alpha^{(n-1)/4}\det(\mathbf{B})^{1/n}$

where $\alpha = 1/(\delta - 1/4) \geq 4/3$.

2 Size reduction

The size reduction condition ($|\mu_i j| \leq 1/2$) in the definition of LLL reduced basis can be achieved using a variant of the Gram-Schmidt orthogonalization procedure. This condition has also a nice geometric interpretation we are going to explain first.

It is easy to see that any point in the linear span of a lattice can be written as the sum of a lattice point $\mathbf{x} \in \mathcal{L}(\mathbf{B})$ plus a vector in the fundamental parallelepiped

$$\mathbf{y} \in \mathcal{P}(\mathbf{B}) = \{\mathbf{Bx} : 0 \leq x_i < 1\}.$$

Moreover, such a decomposition is unique. In other words, the sets $\mathbf{x} + \mathcal{P}(\mathbf{B})$ (indexed by $\mathbf{x} \in \mathcal{L}(\mathbf{B})$) form a partition of $\mathrm{span}(\mathbf{B})$. A subset $S \subseteq \mathrm{span}(\mathbf{B})$ such that $\{\mathbf{x}+S : \mathbf{x} \in \mathcal{L}(\mathbf{B})\}$ form a partition of $\mathrm{span}(\mathbf{B})$ is called a *fundamental region* for the lattice, and $\mathcal{P}(\mathbf{B})$ is an example of fundamental region.

There are many other examples of interesting fundamental regions. For example, one can consider the *centered* half open parallelepiped

$$\mathcal{C}(\mathbf{B}) = \{\mathbf{Bx}: -1/2 \leq x_i < +1/2\}.$$

Another important fundamental region is the Voronoi cell of the lattice $\mathcal{V}(\mathbf{B})$, i.e., the set of all points that are closer to the origin than to any other lattice point.[1]

Notice that the partition associated to $\mathcal{P}(\mathbf{B})$ can be easily computed, in the sense that given a target point $\mathbf{Bt}$ (where $\mathbf{t}$ is a real vector), one can efficiently find the lattice point $\mathbf{Bx}$ such that $\mathbf{t} \in \mathbf{Bx} + \mathcal{P}(\mathbf{B})$. (Just set $\mathbf{x} = \lfloor \mathbf{t} \rfloor$.) The partition associated to the centered parallepiped $\mathcal{C}(\mathbf{B})$ can also be computed similarly, rounding the entried of $\mathbf{t}$ to the closest integer. On the other hand, the partition associated to the Voronoi cell seems hard to compute: given a point in space $\mathbf{Bt}$, determining which voronoi cell $\mathbf{Bx} + \mathcal{V}(\mathbf{B})$ it belongs to is easily shown to be equivalent to solving the closest vector problem, i.e., finding the lattice point closest to $\mathbf{Bt}$.

The size reduction condition in the definition of LLL reduced basis can be easily interpreted as partitioning the space according to still another fundamental region: the orthogonalized centered parallelepiped $\mathcal{C}(\mathbf{B}^*)$, where $\mathbf{B}^*$ is the Gram-Schmidt matrix of $\mathbf{B}$. (Prove, as an excercise, that $\mathcal{C}(\mathbf{B}^*)$ is a fundamental region for lattice $\mathcal{L}(\mathbf{B})$. See figure 1 for an illustration.) Observe that if $\mathbf{b}_i - \mathbf{b}_i^* \in \mathbf{Bx} + \mathcal{C}(\mathbf{B}^*)$, then $-1/2 \leq \mu_{i,j} < 1/2$ for all $j < i$. So, given a lattice basis $\mathbf{B}$, a size reduced basis for the same lattice can be easily obtained as follows:

[1] In order to get a partition, one need also to include boundary points, with some care to avoid including the same point in multiple regiones. For example, the norm relation $\|\mathbf{x}\| \leq \|\mathbf{y}\|$ can be extended to a total order by defining $\mathbf{x} \leq \mathbf{y}$ if and only if $\|\mathbf{x}\| < \|\mathbf{y}\|$ or $\|\mathbf{x}\| = \|\mathbf{y}\|$ and $\mathbf{x}$ preceeds $\mathbf{y}$ lexicographically. Then, the *half-open* Voronoi cell can be defined as the sef of all points $\mathbf{x}$ such that $\mathbf{x} \leq (x - \mathbf{y})$ for any lattice point $\mathbf{y} \in \mathcal{L}(\mathbf{B})$.

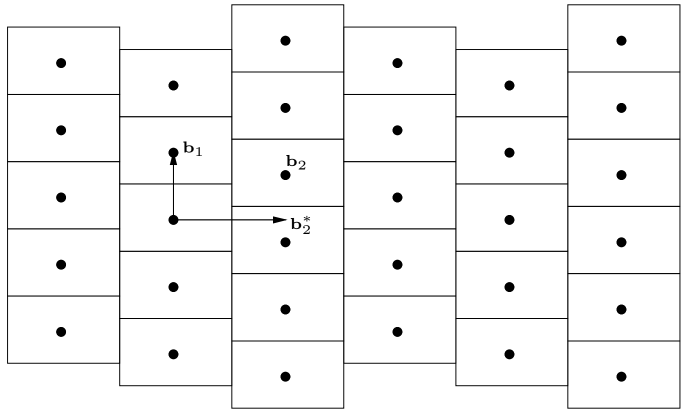

Figure 1: The centered orthogonalized parallelepiped $\mathcal{C}(\mathbf{B}^*)$ if a fundamental region for lattice $\mathcal{L}(\mathbf{B})$.

1. For $i = 2, \ldots, n$ do
2. Find a lattice point $\mathbf{x} \in \mathcal{L}([\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}])$ such that $\mathbf{b}_i - \mathbf{b}_i^* \in \mathbf{x} + \mathcal{C}[\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}]^*$
3. subtract $\mathbf{x}$ from $\mathbf{b}_i$.

It is clear that the final $\mathbf{B}$ is a basis for the original lattice because we only executed elementary integer column operations in step 3. Moreover, it is easy to verify that after iteration i, the size reduction condition holds for all $j < i$. Finally, at iteration i, the Gram-Schmidt coefficients $\mu_{i',j}$ with $j < i' < i$ do not change. So, upon termination, the basis is size reduced.

We will need to show how to find the cell $\mathbf{Bx} + \mathcal{C}(\mathbf{B}^*)$ containing a given target $\mathbf{t}$. This can be easily achieved by a simple, still very useful, variant of the Gram-Schmidt algorithm:

Step 0. $i := n$
Step 1. $x_i := \left\lfloor \frac{\langle \mathbf{t}, \mathbf{b}_i^* \rangle}{\langle \mathbf{b}_i^*, \mathbf{b}_i^* \rangle} \right\rceil$
Step 2. $\mathbf{t} = \mathbf{t} - x_i \mathbf{b}_i$
Step 3. $i := i - 1$, if $i \geq 1$ goto Step 1.

(The operation $\lfloor x \rceil$ denotes choosing the nearest integer to x, choosing the smaller when there is a tie.) The output of the algorithm is the lattice vector $\sum_i x_i \mathbf{b}_i$. We want to prove that the algorithm is correct, i.e., $\mathbf{t} = \mathbf{Bx} + \mathbf{B}^*\mathbf{z}$, where $z_i \in \left[-\frac{1}{2}, \frac{1}{2}\right)$. For each $j = 1,$ ... n, let $\mathbf{t}^{(j)} = \mathbf{t} - \sum_{i>j} x_i \mathbf{b}_i$ denote the value of $\mathbf{t}$ after the jth iteration. Next, consider the component of $B^*\mathbf{z} = B\mathbf{x} - \mathbf{t}$ along $\mathbf{b}_j^*$:

$$(\mathbf{b}_j^*)^T B^* \mathbf{z} = \|\mathbf{b}_j^*\|^2 z_j.$$

We also have

$$(\mathbf{b}_j^*)^T (B\mathbf{x} - \mathbf{t}) \quad = \quad (\mathbf{b}_j^*)^T \left(\sum_{i=1}^{n} \mathbf{b}_i x_i - \mathbf{t} \right)$$

$$
\begin{aligned}
&= (\mathbf{b}_j^*)^T \left(\mathbf{b}_j x_j - \left(\mathbf{t} - \sum_{i>j} \mathbf{b}_i x_i\right)\right) \\
&= (\mathbf{b}_j^*)^T \left(\mathbf{b}_j x_j - \mathbf{t}^{(j)}\right).
\end{aligned}
$$

Recall that $\frac{\langle \mathbf{b}_j^*, \mathbf{b}_j \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} = 1$, and we see that

$$
z_j = \frac{\langle \mathbf{b}_j^*, \mathbf{b}_j \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} x_j - \frac{\langle \mathbf{b}_j^*, t^{(j)} \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} = x_j - \frac{\langle \mathbf{b}_j^*, t^{(j)} \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \in [-\frac{1}{2}, \frac{1}{2})
$$

because $x_j = \lfloor \frac{\langle \mathbf{b}_j^*, t^{(j)} \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \rceil$.

3 The LLL algorithm

The LLL algorithm alternates two steps, aimed at achieving the two properties of an LLL reduced basis. Once we have size-reduced the input basis $\mathbf{B}$, there is only one way $\mathbf{B}$ can fail to be LLL reduced: violate the second condition, i.e., $\delta \|\pi_i(\mathbf{b}_i)\|^2 > \|\pi_i(\mathbf{b}_{i+1})\|^2$ for some index i. If this happens, the algorithm swaps $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$. Several pairs might violate the second property. Which one is selected for the swapping does not matter. In the original LLL algorithm i was chosen to be the smallest unordered pair, but any selection is equally good. In fact, one can even swaps several disjoint pairs at the same time, leading to a parallel variant of the LLL algorithm. After the swap, the basis is not necessarily size reduced anymore. So, one must repeat the whole process from the reduction step.

In summary, the LLL algorithm can be described as follows:

1. Size reduce $\mathbf{B}$

2. If for some i we have
$$\delta \|\pi_i(\mathbf{b}_i)\|^2 > \|\pi_i(\mathbf{b}_{i+1})\|^2,$$
then swap $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ and go back to step 1.

3. Otherwise terminate.

Cleary, upon termination the basis is LLL-reduced because it is size reduced and no pairs need to be swapped. So, if the algorithm terminates, then it is correct. We now prove that the algorithm terminates and it is actually polynomial time.

In order to show that the algorithm is polynomial time, we have to prove that the number of iterations is polynomial in the input size, and each iteration takes polynomial time. We first bound the number of iterations.

3.1 Bounding number of iterations

We now bound the number of iterations performed by the algorithm, i.e., we analyze the maximum number that a swap can occur. This is accomplished by associating a positive integer to the basis B, and showing that each time we swap two vectors this integer decreases by at least a constant factor.

Remember the definition of determinant

$$\det(B) = \prod \|\mathbf{b}_i^*\|.$$

or equivalently

$$\det(B) = \sqrt{|\det(B^T B)|}.$$

From this second definition it is clear that if B is an integer matrix, then the square of the determinant is an integer.

Lemma 3 *For every integer basis $B \in \mathbb{Z}^{m\times n}$, we have* $\det(\mathcal{L}(B))^2 \in \mathbb{Z}$

Therefore, we can associate to the basis B the following integer.

Definition 4 $\mathcal{D} = \prod_{k=1}^{n} det(\mathcal{L}(\mathbf{b}_1, \cdots, \mathbf{b}_k))^2 \in \mathbb{Z}$.

We want to show that $\mathcal{D}$ decreases at least by a factor δ at each iteration. First we will show that step 1 of the algorithm doesn't change $\mathcal{D}$. Then we will show that step 2 decreases $\mathcal{D}$ at least by δ.

To prove that step 1 doesn't change $\mathcal{D}$ we remember that the size reduction does not affect the $\mathbf{b}_i^*$'s. Since $\mathcal{D}$ can be expressed as a function of the $\mathbf{b}_i^*$'s, step 1 does not change the value of $\mathcal{D}$.

At this point we need to look at the effect that a swap has on $\mathcal{D}$. Let us look at the effect of a single swap say between $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$. Let $\mathcal{D}$ be the integer associated to the basis B before a swap, and $\mathcal{D}'$ the corresponding integer after the swap.

Notice that for all $j \neq i$ the lattice $\mathcal{L}(\mathbf{b}_1, \cdots, \mathbf{b}_j)$ is not changed by the swap. To prove this look at the two cases $j < i$ and $j > i$. When $j < i$ then there is no change in the basis $[\mathbf{b}_1, \cdots, \mathbf{b}_j]$, so the value of $\det(\mathcal{L}(\mathbf{b}_1, \cdots, \mathbf{b}_j))$ remains the same. On the other hand, if $j > i$ the only change is that two basis vectors in $[\mathbf{b}_1, \cdots, \mathbf{b}_j]$ are swapped, so the lattice $\mathcal{L}(\mathbf{b}_1, \cdots, \mathbf{b}_j)$ does not change and the determinant $\det(\mathcal{L}(\mathbf{b}_1, \cdots, \mathbf{b}_j))$ stays also the same.

So, the only factor in $\mathcal{D}$ that is affected by the swap is the one corresponding to lattice $\mathcal{L}(\mathbf{b}_1, \cdots, \mathbf{b}_i)$. Here we are replacing the last vector $\mathbf{b}_i$ by $\mathbf{b}_{i+1}$. Therefore

$$\frac{\mathcal{D}}{\mathcal{D}'} = \frac{\prod_{j\leq i} \|\mathbf{b}_j^*\|^2}{(\prod_{j<i} \|\mathbf{b}_j^*\|^2) \cdot \|\pi_i(\mathbf{b}_{i+1})\|^2} = \frac{\|\pi_i(\mathbf{b}_i)\|^2}{\|\pi_i(\mathbf{b}_{i+1})\|^2} \geq \delta^{-1}$$

because swaps are performed only when $\|\pi_i(\mathbf{b}_{i+1})\|^2 < \delta\|\pi_i(b_i)\|^2$.

This proves that

$$\mathcal{D}' \leq \delta\mathcal{D}$$

and by induction on n,

$$\mathcal{D}^{(n)} \leq \delta^n \mathcal{D}$$

where $\mathcal{D}$ is the value associated to the initial basis and $\mathcal{D}^{(n)}$ is the value after n iterations.

Since $\mathcal{D}$ is a positive integer, $\mathcal{D} \geq 1$ and

$$(\tfrac{1}{\delta})^n \leq \mathcal{D}$$

or equivalently

$$n \leq \log_{\frac{1}{\delta}} \mathcal{D} \tag{2}$$

This proves an upper bound on the number of iterations as a function of the initial value of $\mathcal{D}$. Since $\mathcal{D}$ is computable in polynomial time from the input basis, then its size must be polynomial in the input size. An estimate of how big $\mathcal{D}$ is can be easily obtained using Hadamard inequality.

3.2 Bounding the numbers

We proved that the number of iterations is bounded by a polynomial in the input size. In order to bound the running time of the algorithm we still need to show that each iteration also takes polynomial time. The number of arithmetic operations performed at each iteration is clearly polynomial. So, in order to prove a polynomial bound on the running time we only need to show that the size of the numbers involved in the entire computation also is polynomially bounded.

The LLL algorithm uses rational numbers, so we need to bound both the precision required by this number and their magnitude.

From the Gram-Schmidts orthogonalization formulas we know that

$$\mathbf{b}_i^* = \mathbf{b}_i + \sum_{j<i} \nu_{i,j} \mathbf{b}_j$$

for some reals $\nu_{i,j}$. Since $\mathbf{b}_i^*$ is orthogonal to $\mathbf{b}_t$ for all $t < i$ we have

$$\langle \mathbf{b}_t, \mathbf{b}_i \rangle + \sum_{j<i} \nu_{i,j} \langle \mathbf{b}_t, \mathbf{b}_j \rangle = 0.$$

In matrix notation, if we let $B_{i-1} = [\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}]$ and $(\nu_i)_j = \nu_{i,j}$ this can be written as:

$$(B_{i-1}^T \cdot B_{i-1}) \cdot \nu_i = -B_{i-1}^T \cdot \mathbf{b}_i$$

which is an integer vector. Solving the above system of linear equations in variables ν_i using Cramer's rule we get

$$\nu_{i,j} \in \frac{\mathbb{Z}}{\det(B_{i-1}^T \cdot B_{i-1})} = \frac{\mathbb{Z}}{\det(B)^2}.$$

We use this property to bound the denominators that can occur in the coefficients $\mu_{i,j}$ and orthogonalized vectors $\mathbf{b}_i^*$. Let $D_i = \det(B_{i-1})^2$ and notice that

$$D_{i-1} \cdot \mathbf{b}_i^* = D_{i-1} \cdot \mathbf{b}_i + \sum_{j<i} (D_{i-1} \nu_{i,j}) \mathbf{b}_j$$

is an integer combination of integer vectors. So, all denominators that occur in vector $\mathbf{b}_i^*$ are factors of D_{i-1}. Let's now compute the coefficients:

$$\begin{aligned}
\mu_{i,j} &= \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \\
&= \frac{D_{j-1} \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{D_{j-1} \langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \\
&= \frac{\langle \mathbf{b}_i, D_{j-1} \mathbf{b}_j^* \rangle}{D_j} \in \frac{\mathbb{Z}}{D_j}
\end{aligned}$$

and the denominators in the $\mu_{i,j}$ must divide D_j. Notice that $\mathcal{D} = \prod D_j$ and therefore all entries in μ_i, j and $\mathbf{b}_i^*$ can be written as an integer divided by $\mathcal{D}$.

We already know that that $\mu_{i,j}$ are at most $1/2$ in absolute value, so their bit-size is bounded by $\log \mathcal{D}$. We now bound the length of the vectors. Using $D_i = \prod_{j=1}^{i} \|\mathbf{b}_i^*\|^2$, we get

$$\|\mathbf{b}_i^*\|^2 = \frac{D_i}{D_{i-1}} \leq D_i \leq \mathcal{D}.$$

Finally,

$$\|\mathbf{b}_i\|^2 = \|\mathbf{b}_i^*\|^2 + \sum_{j<i} \mu_{i,j}^2 \|\mathbf{b}_j^*\|^2 \leq \mathcal{D} + (n/4)\mathcal{D} \leq n\mathcal{D}.$$

So, all numerators and denominators of the numbers occurring in the execution of the algorithm have bit-size polynomial in $\log \mathcal{D}$.

4 Applications

We conclude with a simple application of LLL to algorithmic number theory: showing that we can *efficiently* write any prime $p \equiv 1 \pmod 4$ as the sum of two squares. Remember the proof that any such prime is the sum of two squares: all vectors $[a, b]^T$ in the lattice

$$\mathbf{B} = \begin{bmatrix} 1 & 0 \\ i & p \end{bmatrix}$$

have the property that $a^2 + b^2$ is a multiple of p. So if we can find a nonzero lattice vector of squared norm less than $2p$, it must be $a^2 + b^2 = p$. Minwoski's theorem assures us that such short vectors exist. The question is: how can we find it? Answer: using LLL!

Run the LLL algorithm on the lattice basis, to obtain a reduced basis $\mathbf{b}_1, \mathbf{b}_2$ for the same lattice. We know, from theorem 2, that $\|\mathbf{b}_1\| \leq \alpha^{1/4} \det(\mathbf{B})^{1/2}$. Squaring, and using $\det(\mathbf{B}) = p$, we get $\|\mathbf{b}_1\|^2 \leq \sqrt{\alpha} p < 2p$ as desired, provided $\delta > 3/4$.