1 The LLL Algorithm

Recall the definition of an LLL-reduced lattice basis, as defined by Lenstra, Lenstra, and Lovász [LLL82], and how it approximates a short nonzero lattice vector. (In all that follows, recall that $\mathbf{B} = \widetilde{\mathbf{B}} \cdot \mathbf{U}$ is the Gram–Schmidt decomposition of a lattice basis $\mathbf{B}$, where the columns of $\widetilde{\mathbf{B}}$ are orthogonal and $\mathbf{U} \in \mathbb{R}^{n \times n}$ is upper unitriangular, with entries $\mu_{i,j}$.)

Definition 1.1. A lattice basis $\mathbf{B}$ is *LLL-reduced* if the following two conditions are met:

1. $|\mu_{i,j}| \leq \frac{1}{2}$ for all $i < j$. (Such a basis is said to be "size reduced.")
2. $\frac{3}{4}\|\widetilde{\mathbf{b}}_i\|^2 \leq \|\mu_{i,i+1}\widetilde{\mathbf{b}}_i + \widetilde{\mathbf{b}}_{i+1}\|^2$ for all $i < n$. (This is the "Lovász condition.")

Lemma 1.2. *In an LLL-reduced lattice basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$, we have that $\|\widetilde{\mathbf{b}}_{i+1}\|^2 \geq \frac{1}{2}\|\widetilde{\mathbf{b}}_i\|^2$ for all $i < n$, and hence $\|\mathbf{b}_1\| \leq 2^{(n-1)/2} \min_i \|\widetilde{\mathbf{b}}_i\| \leq 2^{(n-1)/2} \cdot \lambda_1(\mathcal{L}(\mathbf{B}))$.*

Algorithm 1 defines the LLL algorithm, which transforms any lattice basis into an LLL-reduced one of the same lattice. The algorithm simply alternates between two steps: *size-reducing* the current basis (i.e., making Item 1 hold), and checking whether the Lovász condition (Item 2) holds; if not, it simply swaps a pair of offending basis vectors. From this description, we can immediately see that the output is indeed LLL-reduced, so the algorithm is correct—if it ever terminates. The key challenge is in showing that it does indeed terminate, and in time polynomial in the input size.

Algorithm 1 The LLL algorithm.

```
1:  function LLL(B ∈ Z^{n×n})                ▷ Outputs an LLL-reduced basis of the lattice L(B)
2:      let B ← SizeReduce(B)
3:      if there exists an i < n for which the Lovász condition is violated then
4:          swap b_i and b_{i+1} and go to Line 2
5:      return B
6:  function SizeReduce(B ∈ Z^{n×n})         ▷ Outputs a size-reduced basis of L(B), preserving B̃
7:      compute (or update) the Gram–Schmidt orthogonalization B̃ of B
8:      for j = 2, ..., n (in any order) do
9:          for i = j − 1 down to 1 do
10:             let b_j ← b_j − ⌊u_{i,j}⌉ · b_i, where u_{i,j} = ⟨b_j, b̃_i⟩/⟨b̃_i, b̃_i⟩
```

The SizeReduce($\mathbf{B}$) subroutine deserves some elaboration. Its purpose is, in the Gram–Schmidt decomposition $\mathbf{B} = \widetilde{\mathbf{B}} \cdot \mathbf{U}$, to shift the entries in the upper triangle of $\mathbf{U}$ by integers, so that they lie in $[-\frac{1}{2}, \frac{1}{2})$. Because we must preserve the lattice, this is done by subtracting appropriate integer multiples of the basis vectors $\mathbf{b}_i$ (*not* the Gram–Schmidt vectors $\widetilde{\mathbf{b}}_i$!) from the $\mathbf{b}_j$, for $i < j$. In matrix form, the (i,j)th iteration lets $\mathbf{B} \leftarrow \mathbf{B} \cdot \mathbf{W} = \widetilde{\mathbf{B}} \cdot (\mathbf{UW})$, where $\mathbf{W}$ is the upper unitriangular matrix with just one possibly nonzero off-diagonal entry $-\lfloor u_{i,j} \rceil$, at position (i,j). This implicitly updates the (upper unitriangular) matrix $\mathbf{U} \leftarrow \mathbf{UW} \in \mathbb{R}^{n \times n}$. This matrix is identical to $\mathbf{U}$, except possibly at position (i,j) *and the entries above it, but not below it* in the same column. This is why we need to make the changes going *upward* in each column (see Line 9).

Lemma 1.3. *Given an integral lattice basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$ with Gram–Schmidt orthogonalization $\widetilde{\mathbf{B}}$, the* SizeReduce *algorithm outputs a basis $\mathbf{B}'$ of $\mathcal{L} = \mathcal{L}(\mathbf{B})$ having Gram–Schmidt decomposition $\mathbf{B}' = \widetilde{\mathbf{B}} \cdot \mathbf{U}'$, where $u'_{i,j} \in [-\frac{1}{2}, \frac{1}{2})$ for all $i < j$.*

Proof. First, even though SizeReduce may change $\mathbf{B}$, the Gram–Schmidt vectors $\widetilde{\mathbf{B}}$ are preserved throughout, because the only changes to $\mathbf{B}$ are via multiplication by upper-unitriangular matrices, i.e., if $\mathbf{B} = \widetilde{\mathbf{B}} \cdot \mathbf{U}$ is the Gram–Schmidt decomposition prior to some iteration, then $\mathbf{B} = \widetilde{\mathbf{B}} \cdot (\mathbf{UW})$ is the decomposition afterward, since $\mathbf{UW}$ is upper unitriangular. Second, the (i, j)th iteration ensures that the value $\langle \mathbf{b}_j, \widetilde{\mathbf{b}}_i \rangle / \langle \widetilde{\mathbf{b}}_i, \widetilde{\mathbf{b}}_i \rangle \in [-\frac{1}{2}, \frac{1}{2})$, and that value never changes in later iterations, because for all $h < i$, the basis vector $\mathbf{b}_h$ (a multiple of which may later be subtracted from $\mathbf{b}_j$) is orthogonal to $\widetilde{\mathbf{b}}_i$. □

We now state the main theorem about the LLL algorithm.

Theorem 1.4. *Given an integral lattice basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$, the LLL algorithm outputs an LLL-reduced basis of $\mathcal{L} = \mathcal{L}(\mathbf{B})$ in time $\mathrm{poly}(n, |\mathbf{B}|)$, where $|\mathbf{B}|$ denotes the bit length of the input basis.*

The remainder of this section is dedicated to an (almost complete) proof of this theorem. First, it is clear that the LLL algorithm, if it ever terminates, is correct: all the operations on the input basis preserve the lattice it generates, and the algorithm can only output an LLL-reduced basis.

We next prove that the number of iterations is $O(N)$ for some $N = \mathrm{poly}(n, |\mathbf{B}|)$. This uses a "potential argument," which assigns a value to all the intermediate bases produced by the algorithm. We show three facts: that the potential starts out no larger than 2^N, that it never drops below 1, and that each iteration of the algorithm decreases the potential by a factor of at least $\sqrt{4/3} > 1$. All this implies that the number of iterations is at most $\log_{\sqrt{4/3}} 2^N = O(N)$.

The potential function is defined as follows: for a basis $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$, let $\mathcal{L}_i = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_i)$ for each $i \leq n$. The potential is the product of the determinants of these lattices:

$$\Phi(\mathbf{B}) := \prod_{i=1}^{n} \det(\mathcal{L}_i) = \prod_{i=1}^{n} \left(\|\widetilde{\mathbf{b}}_1\| \cdots \|\widetilde{\mathbf{b}}_i\| \right) = \prod_{i=1}^{n} \|\widetilde{\mathbf{b}}_i\|^{n-i+1}.$$

Claim 1.5. *The potential of the input basis $\mathbf{B}$ is at most 2^N, and every intermediate basis the algorithm produces has potential at least 1.*

Proof. Because $\|\widetilde{\mathbf{b}}_i\| \leq \|\mathbf{b}_i\|$ and $\|\mathbf{b}_i\| \geq 1$ for all i, the potential of the input basis $\mathbf{B}$ is bounded by

$$\Phi(\mathbf{B}) \leq \prod_{i=1}^{n} \|\mathbf{b}_i\|^{n-i+1} \leq \prod_{i=1}^{n} \|\mathbf{b}_i\|^{n} \leq \max_i \|\mathbf{b}_i\|^{n^2} = 2^{\mathrm{poly}(n, |\mathbf{B}|)}.$$

Every intermediate basis is integral, hence the lattices $\mathcal{L}_i$ associated with that basis have determinant at least 1.[1] Therefore, the potential of that basis is at least 1. □

We next analyze how the potential changes when we perform a swap in Line 4.

Claim 1.6. *Suppose that $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ are swapped in Line 4, and let the resulting basis be denoted $\mathbf{B}'$. Then $\widetilde{\mathbf{b}}'_j = \widetilde{\mathbf{b}}_j$ for all $j \notin \{i, i+1\}$, and $\widetilde{\mathbf{b}}'_i = \mu_{i,i+1} \widetilde{\mathbf{b}}_i + \widetilde{\mathbf{b}}_{i+1}$.*

Note that this claim lets us optimize Line 7: following a swap, we need not orthogonalize $\mathbf{B}'$ from scratch, but only need to update two of the vectors from $\widetilde{\mathbf{B}}$.

[1] This requires some care to verify, because the lattices $\mathcal{L}_i$ are *not full rank* for $i < n$. Letting $\mathbf{B}_i = (\mathbf{b}_1, \ldots, \mathbf{b}_i) \in \mathbb{Z}^{n \times i}$ be a basis for $\mathcal{L}_i$, we have $\det(\mathcal{L}_i) = \sqrt{\det(\mathbf{B}_i^t \mathbf{B}_i)} \geq 1$, because the Gram matrix $\mathbf{B}_i^t \mathbf{B}_i \in \mathbb{Z}^{i \times i}$ is integral and nonsingular.

Proof. For all $j < i$, we have $\widetilde{\mathbf{b}}'_j = \widetilde{\mathbf{b}}_j$, because it is the component of $\mathbf{b}'_j = \mathbf{b}_j$ orthogonal to $\mathrm{span}(\mathbf{b}'_1, \ldots, \mathbf{b}'_{j-1}) = \mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{j-1})$. Similarly, for any $j > i+1$, we have $\widetilde{\mathbf{b}}'_j = \widetilde{\mathbf{b}}_j$, because it is the component of $\mathbf{b}'_j = \mathbf{b}_j$ orthogonal to $\mathrm{span}(\mathbf{b}'_1, \ldots, \mathbf{b}'_{j-1}) = \mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{j-1})$, where here the equality holds because both $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ are in the lists of arguments (in reversed order). Finally, $\widetilde{\mathbf{b}}'_i$ is the component of $\mathbf{b}'_i = \mathbf{b}_{i+1}$ orthogonal to $\mathrm{span}(\mathbf{b}'_1, \ldots, \mathbf{b}'_{i-1}) = \mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})$, which is $\mu_{i,i+1}\widetilde{\mathbf{b}}_i + \widetilde{\mathbf{b}}_{i+1}$ by construction. □

Lemma 1.7. *Suppose that $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ are swapped in Line 4, and let the resulting basis be denoted $\mathbf{B}'$. Then $\Phi(\mathbf{B}')/\Phi(\mathbf{B}) < \sqrt{3/4}$.*

Proof. Let $\mathcal{L}_i = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_i)$ and $\mathcal{L}'_i = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_{i+1})$. By Claim 1.6,

$$\frac{\Phi(\mathbf{B}')}{\Phi(\mathbf{B})} = \frac{\det(\mathcal{L}'_i)}{\det(\mathcal{L}_i)} = \frac{\|\widetilde{\mathbf{b}}_1\| \cdots \|\widetilde{\mathbf{b}}_{i-1}\| \|\mu_{i,i+1}\widetilde{\mathbf{b}}_i + \widetilde{\mathbf{b}}_{i+1}\|}{\|\widetilde{\mathbf{b}}_1\| \cdots \|\widetilde{\mathbf{b}}_{i-1}\| \|\widetilde{\mathbf{b}}_i\|} = \frac{\|\mu_{i,i+1}\widetilde{\mathbf{b}}_i + \widetilde{\mathbf{b}}_{i+1}\|}{\|\widetilde{\mathbf{b}}_i\|} < \sqrt{3/4},$$

where the last inequality follows from the fact that the swap occurs because the Lovász condition (Item 2) is *not* satisfied. □

This completes the proof that the number of iterations is $O(N) = \mathrm{poly}(n, |\mathbf{B}|)$. Moreover, we can see by inspection that each iteration of the algorithm is polynomial time in the bit length of the current basis. However, this does *not* necessarily guarantee that the LLL algorithm is polynomial time overall, since the bit length of the intermediate bases could possibly *increase* with each iteration. (For example, if the bit length doubled in each iteration, then by the end, the bit length would be exponential in n.)

To complete the full proof that LLL is polynomial time, it suffices to show that the sizes of *all* intermediate bases are some fixed polynomial in the size of the original basis. This turns out to be the case, specifically due to the size-reduction step (which we have not used up to this point!). The proof of this fact is somewhat grungy, though, so we won't cover it.

We conclude with some final remarks about the LLL algorithm. The factor $3/4$ in the Lovász condition is just for convenience of analysis. We can use any constant between $1/4$ and 1, which yields a tradeoff between the final approximation factor and the number of iterations, but these will still remain exponential and polynomial (in n), respectively. By choosing a value very close to 1, we can obtain an approximation factor of $(2/\sqrt{3})^n$ in polynomial time, but we cannot do any better using LLL. We can get slightly better approximation factors of $2^{O(n(\log\log n)^2/\log n)}$, still in polynomial time, using Schnorr's generalization [Sch87] of LLL, where the analogue of the Lovász condition deals with blocks of $k \geq 2$ consecutive vectors.

References

[LLL82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, December 1982. Page 1.

[Sch87] C.-P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987. Page 3.