

## 1 The Shortest and Closest Vector Problems

Recall the definition of the approximate Shortest Vector Problem. (The exact version is obtained by taking  $\gamma = 1$ , which is implicit when  $\gamma$  is omitted.)

**Definition 1.1.** For  $\gamma = \gamma(n) \geq 1$ , the  $\gamma$ -approximate Shortest Vector Problem  $\text{SVP}_\gamma$  is: given a basis  $\mathbf{B}$  of a lattice  $\mathcal{L} = \mathcal{L}(\mathbf{B}) \subset \mathbb{R}^n$ , find some nonzero  $\mathbf{v} \in \mathcal{L}$  such that  $\|\mathbf{v}\| \leq \gamma(n) \cdot \lambda_1(\mathcal{L})$ .

A closely related inhomogeneous variant is the approximate Closest Vector Problem.

**Definition 1.2.** For  $\gamma = \gamma(n) \geq 1$ , the  $\gamma$ -approximate Closest Vector Problem  $\text{CVP}_\gamma$  is: given a basis  $\mathbf{B}$  of a lattice  $\mathcal{L} = \mathcal{L}(\mathbf{B}) \subset \mathbb{R}^n$  and a point  $\mathbf{t} \in \mathbb{R}^n$ , find some  $\mathbf{v} \in \mathcal{L}$  such that  $\|\mathbf{t} - \mathbf{v}\| \leq \gamma(n) \cdot \text{dist}(\mathbf{t}, \mathcal{L})$ . Equivalently, find an element of the lattice coset  $\mathbf{t} + \mathcal{L}$  having norm at most  $\gamma(n) \cdot \lambda(\mathbf{t} + \mathcal{L})$ , where  $\lambda(\mathbf{t} + \mathcal{L}) := \min_{\mathbf{x} \in \mathbf{t} + \mathcal{L}} \|\mathbf{x}\| = \text{dist}(\mathbf{t}, \mathcal{L})$ .

Above we have used the fact that  $\text{dist}(\mathbf{t}, \mathcal{L}) = \min_{\mathbf{v} \in \mathcal{L}} \|\mathbf{t} - \mathbf{v}\| = \min_{\mathbf{x} \in \mathbf{t} + \mathcal{L}} \|\mathbf{x}\|$ , because  $\mathcal{L} = -\mathcal{L}$ . The two versions of CVP are equivalent by associating each  $\mathbf{v} \in \mathcal{L}$  with  $\mathbf{t} - \mathbf{v} \in \mathbf{t} + \mathcal{L}$ , and vice versa. Although the former version of the problem is the more “obvious” formulation, the latter version is often more convenient in algorithmic settings, so we will use it throughout what follows.

We first show that  $\text{SVP}_\gamma$  is no harder than  $\text{CVP}_\gamma$ ; more specifically, given an oracle for  $\text{CVP}_\gamma$  we can solve  $\text{SVP}_\gamma$  efficiently. Several other natural lattice problems also reduce to  $\text{CVP}_\gamma$  [Mic08], making it a central algorithmic target of interest.

**Theorem 1.3.** For any  $\gamma \geq 1$ , we have  $\text{SVP}_\gamma \leq \text{CVP}_\gamma$  via a Cook reduction.

*Proof.* Consider the following algorithm that, given a lattice basis  $\mathbf{B} = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ , and CVP oracle  $\mathcal{O}$ , outputs some  $\mathbf{v} \in \mathcal{L} = \mathcal{L}(\mathbf{B})$ :

- For each  $i = 1, \dots, n$ , compute basis  $\mathbf{B}_i = (\mathbf{b}_1, \dots, \mathbf{b}_{i-1}, 2\mathbf{b}_i, \mathbf{b}_{i+1}, \dots, \mathbf{b}_n)$  and let  $\mathbf{v}_i = \mathcal{O}(\mathbf{B}_i, \mathbf{b}_i)$ .
- Output any one of the  $\mathbf{v}_i$  that has minimal length  $\|\mathbf{v}_i\|$ .

We claim that this algorithm solves  $\text{SVP}_\gamma$ , i.e., it returns some nonzero lattice vector of length at most  $\gamma \cdot \lambda_1(\mathcal{L})$ .

First observe that for each  $i$ , the lattice  $\mathcal{L}_i = \mathcal{L}(\mathbf{B}_i) \subset \mathcal{L}$  consists of all those vectors in  $\mathcal{L}$  whose  $\mathbf{b}_i$ -coordinate is even, whereas the coset  $\mathbf{b}_i + \mathcal{L}_i \subset \mathcal{L}$  consists of all those whose  $\mathbf{b}_i$ -coordinate is odd. Therefore,  $\mathbf{0} \notin \mathbf{b}_i + \mathcal{L}_i$  for all  $i$ , so  $\lambda(\mathbf{b}_i + \mathcal{L}_i) \geq \lambda_1(\mathcal{L})$ . Moreover, if  $\mathbf{v} \in \mathcal{L}$  is any shortest nonzero lattice vector, then at least one of its coefficients with respect to  $\mathbf{B}$  must be odd, otherwise  $\mathbf{v}/2 \in \mathcal{L}$  would be a shorter nonzero lattice vector. Therefore,  $\mathbf{v} \in \mathbf{b}_i + \mathcal{L}_i$  for at least one  $i$ , and so  $\lambda(\mathbf{b}_i + \mathcal{L}_i) = \lambda_1(\mathcal{L})$  for such  $i$ .

Now by hypothesis on  $\mathcal{O}$ , for every  $i$  we have  $\mathbf{v}_i \in \mathbf{b}_i + \mathcal{L}_i \subset \mathcal{L} \setminus \{\mathbf{0}\}$  (so the reduction outputs a nonzero lattice vector) and  $\|\mathbf{v}_i\| \leq \gamma \cdot \lambda(\mathbf{b}_i + \mathcal{L}_i)$ . Since  $\lambda(\mathbf{b}_i + \mathcal{L}_i) = \lambda_1(\mathcal{L})$  for at least one  $i$ , some  $\|\mathbf{v}_i\| \leq \gamma \cdot \lambda_1(\mathcal{L})$ , and correctness follows.  $\square$

We note that essentially the same reduction works for the *decisional* variants  $\text{GapSVP}_\gamma$  and  $\text{GapCVP}_\gamma$  of the problems, where instead of returning vectors  $\mathbf{v}_i$ , the oracle  $\mathcal{O}$  returns yes/no answers, and the reduction outputs the logical OR of all the answers. It is unknown if there exists a reverse reduction, i.e., if  $\text{CVP}_\gamma \leq \text{SVP}_\gamma$ , even for the exact regime  $\gamma = 1$ , or for (nontrivial) different approximation factors, or allowing randomization.

## 1.1 Algorithms for SVP and CVP

The following are some historical milestones in algorithms for SVP and CVP (for simplicity, we ignore polynomial factors in the dimension  $n$  and bit length of the input basis):

- Using the LLL algorithm [LLL82] and brute-force search over coefficient vectors, one can get an algorithm that solves SVP and CVP in  $2^{O(n^2)}$  time and  $\text{poly}(n)$  space.
- In 1983, Kannan [Kan83] gave deterministic algorithms that solve SVP and CVP in  $n^{O(n)} = 2^{O(n \log n)}$  time and  $\text{poly}(n)$  space.
- In 2001, Ajtai, Kumar, and Sivakumar (AKS) [AKS01] gave *randomized* “sieve” algorithms that solve SVP and CVP $_{1+\epsilon}$  (for any constant  $\epsilon > 0$ ) in singly exponential  $2^{O(n)}$  time *and* space. (For CVP $_{1+\epsilon}$ , the exponent in the running time depends inversely on  $\epsilon$ .)
- In 2010, Micciancio and Voulgaris (MV) [MV10] gave a *deterministic* algorithm that solves CVP (and hence SVP and other problems) in  $2^{2n}$  time and  $2^n$  space.
- In 2015, Aggarwal, Dadush, Regev, and Stephens-Davidowitz [ADRS15] gave a randomized algorithm that solves SVP in  $2^n$  time and space (note that the constant factor in the exponent here is exactly one). A follow-up work by Aggarwal, Dadush, and Stephens-Davidowitz [ADS15] obtained a similar result for CVP.

It is an important open question whether there exists a singly exponential-time (or better) algorithm that uses only polynomial space, or even subexponential space.

## 2 The Micciancio-Voulgaris Algorithm for CVP

The MV algorithm solves CVP on any  $n$ -dimensional lattice in  $2^{O(n)}$  time (again, ignoring polynomial factors in the input length). It is based around the (closed) *Voronoi* cell of the lattice, which, to recall, is the set of all points in  $\mathbb{R}^n$  that are at least as close to the origin as to any other lattice point:

$$\bar{\mathcal{V}}(\mathcal{L}) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| \leq \|\mathbf{x} - \mathbf{v}\| \ \forall \mathbf{v} \in \mathcal{L} \setminus \{\mathbf{0}\}\}. \quad (2.1)$$

We often omit the argument  $\mathcal{L}$  when it is clear from context. From the definition it can be seen that for any coset  $\mathbf{t} + \mathcal{L}$ , the set  $(\mathbf{t} + \mathcal{L}) \cap \bar{\mathcal{V}}$  consists of all the shortest elements of  $\mathbf{t} + \mathcal{L}$  (i.e., all the correct CVP solutions for  $\mathcal{L}, \mathbf{t}$ ). For any nonzero lattice vector  $\mathbf{v}$ , define the halfspace

$$H_{\mathbf{v}} = \{\mathbf{x} : \|\mathbf{x}\| \leq \|\mathbf{x} - \mathbf{v}\|\} \quad (2.2)$$

$$= \{\mathbf{x} : 2\langle \mathbf{x}, \mathbf{v} \rangle \leq \langle \mathbf{v}, \mathbf{v} \rangle\}. \quad (2.3)$$

It is easy to see that  $\bar{\mathcal{V}}$  is the intersection of the  $H_{\mathbf{v}}$  over all  $\mathbf{v} \in \mathcal{L} \setminus \{\mathbf{0}\}$ . However, some of these  $H_{\mathbf{v}}$  can be omitted without affecting the resulting intersection. The minimal set  $V$  of lattice vectors such that  $\bar{\mathcal{V}} = \bigcap_{\mathbf{v} \in V} H_{\mathbf{v}}$  is called the set of (*Voronoi*) *relevant* vectors. The following characterizes the relevant vectors of a lattice:

**Fact 2.1 (Voronoi).** *A nonzero lattice vector  $\mathbf{v} \in \mathcal{L} \setminus \{\mathbf{0}\}$  is relevant if and only if  $\pm \mathbf{v}$  are the only shortest vectors in the coset  $\mathbf{v} + 2\mathcal{L}$ .*

**Corollary 2.2.** *An  $n$ -dimensional lattice has at most  $2(2^n - 1) \leq 2^{n+1}$  relevant vectors.*

*Proof.* Every relevant vector belongs to some nonzero coset of  $\mathcal{L}/2\mathcal{L}$ , of which there exactly  $2^n - 1$ . By the above, there are at most two relevant vectors in each such coset.  $\square$

The MV algorithm has the following high-level structure. Given a basis  $\mathbf{B}$  of a lattice  $\mathcal{L} = \mathcal{L}(\mathbf{B})$  and a target point  $\mathbf{t}$  defining a coset  $\mathbf{t} + \mathcal{L}$ , the algorithm does the following:

1. Compute (a description of) the Voronoi cell  $\bar{\mathcal{V}}(\mathcal{L})$ , as a list of all the relevant vectors of the lattice, of which there are at most  $2^{n+1} = 2^{O(n)}$ . (The possibly exponential number of relevant vectors is the sole reason for the algorithm's exponential space complexity.)

Note that this “preprocessing” phase depends only on the lattice  $\mathcal{L}$ , not the target  $\mathbf{t}$ , so it can be performed before the target is known, and the result can be reused for additional targets.

2. Use the relevant vectors to “walk,” starting from  $\mathbf{t}$ , through a sequence of vectors in the coset  $\mathbf{t} + \mathcal{L}$  by adding appropriately chosen lattice vectors to the target. The walk finally terminates at some element of  $(\mathbf{t} + \mathcal{L}) \cap \bar{\mathcal{V}}$ , which is a solution to the CVP instance.

The walk proceeds in phases, where each phase starts with some  $\mathbf{t}_k \in (\mathbf{t} + \mathcal{L}) \cap 2^k \cdot \bar{\mathcal{V}}$ , and outputs some  $\mathbf{t}_{k-1} \in (\mathbf{t} + \mathcal{L}) \cap 2^{k-1} \cdot \bar{\mathcal{V}}$ . We show below that each phase takes  $2^{O(n)}$  time, and by using LLL we can ensure that the initial target point  $\mathbf{t}$  is in  $2^{O(n)} \cdot \bar{\mathcal{V}}$ , so the total number of phases is only  $O(n)$ . Therefore, the overall runtime of this step is  $2^{O(n)}$ .

## 2.1 The Walk

We first describe how [Item 2](#), the “walk,” is performed. The first observation is that by a scaling argument, it suffices to show how to perform the final phase of the walk, which takes some  $\mathbf{t}_2 \in (\mathbf{t} + \mathcal{L}) \cap 2\bar{\mathcal{V}}$  and outputs some  $\mathbf{t}_1 \in (\mathbf{t} + \mathcal{L}) \cap \bar{\mathcal{V}}$ . All the prior phases can be performed using this procedure with a suitable scaling of the lattice: since  $2^k \cdot \bar{\mathcal{V}}(\mathcal{L}) = 2 \cdot \bar{\mathcal{V}}(2^{k-1}\mathcal{L})$ , we can use the procedure on the lattice  $2^{k-1}\mathcal{L}$  (whose relevant vectors are just  $2^{k-1}$ -factor scalings of  $\mathcal{L}$ 's relevant vectors) to go from some  $\mathbf{t}_k \in (\mathbf{t} + \mathcal{L}) \cap 2^k \cdot \bar{\mathcal{V}}(\mathcal{L})$  to some  $\mathbf{t}_{k-1} \in (\mathbf{t} + \mathcal{L}) \cap 2^{k-1} \cdot \bar{\mathcal{V}}(\mathcal{L})$ .

The walk from  $2\bar{\mathcal{V}}$  to  $\bar{\mathcal{V}}$  works as follows: first, check if our current target  $\mathbf{t} \in \bar{\mathcal{V}}$ , by testing if  $\mathbf{t} \in H_{\mathbf{v}}$  for all  $\mathbf{v} \in V$ ; if so, then simply output  $\mathbf{t}$ . Otherwise, subtract an appropriately chosen relevant vector  $\mathbf{v} \in V$  from  $\mathbf{t}$  and loop. The main question is, which relevant vector should we choose to ensure that we make progress, and terminate within  $2^{O(n)}$  iterations?

Recall that if  $\mathbf{t} \notin \bar{\mathcal{V}}(\mathcal{L})$ , then it lies outside the halfspace  $H_{\mathbf{v}}$ , i.e., it violates the inequality  $2\langle \mathbf{t}, \mathbf{v} \rangle \leq \langle \mathbf{v}, \mathbf{v} \rangle$ , for some relevant vector  $\mathbf{v}$ . The MV algorithm greedily chooses a relevant vector  $\mathbf{v}$  whose inequality is “most violated,” i.e., it maximizes the ratio  $2\langle \mathbf{t}, \mathbf{v} \rangle / \langle \mathbf{v}, \mathbf{v} \rangle$ . Observe that for such  $\mathbf{v}$ , if we define  $\alpha = 2\langle \mathbf{t}, \mathbf{v} \rangle / \langle \mathbf{v}, \mathbf{v} \rangle$ , then  $\alpha$  is the smallest (positive) real scaling factor of the Voronoi cell such that  $\mathbf{t} \in \alpha\bar{\mathcal{V}}(\mathcal{L})$ . By assumption, we have that  $1 < \alpha \leq 2$ .

**Lemma 2.3.** *The walk from  $(\mathbf{t} + \mathcal{L}) \cap 2\bar{\mathcal{V}}$  to  $(\mathbf{t} + \mathcal{L}) \cap \bar{\mathcal{V}}$  terminates within at most  $2^n$  iterations.*

The above lemma follows by combining the following two claims. The first shows that subtracting the chosen  $\mathbf{v}$  brings  $\mathbf{t}$  closer to the origin, while staying within the same multiple of the Voronoi cell.

**Claim 2.4.** *For any  $\mathbf{t} \notin \bar{\mathcal{V}}$ , if  $\mathbf{v} \in \mathcal{L}$  is a relevant vector maximizing  $\alpha = 2\langle \mathbf{t}, \mathbf{v} \rangle / \langle \mathbf{v}, \mathbf{v} \rangle$ , then  $\mathbf{t} - \mathbf{v} \in \alpha\bar{\mathcal{V}}$  and  $\|\mathbf{t} - \mathbf{v}\| < \|\mathbf{t}\|$ .*

*Proof.* Below we show that  $\mathbf{t}$  and  $\mathbf{t} - \alpha\mathbf{v}$  have equal norms, and that both lie in  $\alpha\bar{\mathcal{V}}$ . Then because  $\alpha > 1$ , it follows that  $\mathbf{t} - \mathbf{v}$  is on the interior of the line segment between  $\mathbf{t}$  and  $\mathbf{t} - \alpha\mathbf{v}$ , so  $\|\mathbf{t} - \mathbf{v}\| < \|\mathbf{t}\|$  and  $\mathbf{t} - \mathbf{v} \in \alpha\bar{\mathcal{V}}$  by convexity of  $\bar{\mathcal{V}}$ , as claimed.

Because  $\alpha = 2\langle \mathbf{t}, \mathbf{v} \rangle / \langle \mathbf{v}, \mathbf{v} \rangle$ , we have

$$\|\mathbf{t}\|^2 = \langle \mathbf{t}, \mathbf{t} \rangle = \langle \mathbf{t}, \mathbf{t} \rangle - 2\alpha\langle \mathbf{t}, \mathbf{v} \rangle + \alpha^2\langle \mathbf{v}, \mathbf{v} \rangle = \|\mathbf{t} - \alpha\mathbf{v}\|^2. \quad (2.4)$$

Now because  $\mathbf{t} \in \alpha\bar{\mathcal{V}}(\mathcal{L}) = \bar{\mathcal{V}}(\alpha\mathcal{L})$ , it must be that  $\mathbf{t}$  is a shortest element of  $\mathbf{t} + \alpha\mathcal{L}$ . Since  $\alpha\mathbf{v} \in \alpha\mathcal{L}$ , we also have  $\mathbf{t} - \alpha\mathbf{v} \in \mathbf{t} + \alpha\mathcal{L}$ . Then because  $\|\mathbf{t} - \alpha\mathbf{v}\| = \|\mathbf{t}\|$ , we conclude that  $\mathbf{t} - \alpha\mathbf{v}$  is also a shortest element in  $\mathbf{t} + \alpha\mathcal{L}$ , hence  $\mathbf{t} - \alpha\mathbf{v} \in \alpha\bar{\mathcal{V}}(\mathcal{L})$ .

Finally, because  $\alpha > 1$  and  $\mathbf{v} \neq \mathbf{0}$  we have

$$\|\mathbf{t} - \mathbf{v}\|^2 = \|\mathbf{t}\|^2 + \|\mathbf{v}\|^2 - 2\langle \mathbf{t}, \mathbf{v} \rangle = \|\mathbf{t}\|^2 - (\alpha - 1)\|\mathbf{v}\|^2 < \|\mathbf{t}\|^2. \quad \square$$

The second claim bounds the number of distinct lengths our intermediate target vectors can have.

**Claim 2.5.** *For any  $\mathbf{t}$ , let  $U_{\mathbf{t}} = (\mathbf{t} + \mathcal{L}) \cap 2\bar{\mathcal{V}}(\mathcal{L})$ . Then  $|\{\|\mathbf{u}\| : \mathbf{u} \in U_{\mathbf{t}}\}| \leq 2^n$ .*

*Proof.* For any  $\mathbf{t}' \in \mathbb{R}^n$ , the elements of  $(\mathbf{t}' + 2\mathcal{L}) \cap 2\bar{\mathcal{V}}(\mathcal{L})$  are the shortest vectors in the coset  $\mathbf{t}' + 2\mathcal{L}$ , and therefore all have the same length. Because  $\mathcal{L}$  is the union of  $2^n$  distinct cosets of  $2\mathcal{L}$ , we see that  $\mathbf{t} + \mathcal{L}$  is also the union of  $2^n$  cosets of  $2\mathcal{L}$ . By partitioning the elements of  $U$  according to these cosets, we conclude that the elements of  $U$  have at most  $2^n$  distinct lengths.  $\square$

We can now prove [Lemma 2.3](#): since each step strictly decreases the length of the target while keeping it in  $2\bar{\mathcal{V}}$ , and the intermediate targets can take on at most  $2^n$  distinct lengths overall, the walk must terminate within  $2^n$  steps (and when it terminates, the target is in  $\bar{\mathcal{V}}$ ). This completes the analysis of the “walk” step.

## 2.2 Computing the Voronoi Cell

We only summarize the main ideas behind the computation of the relevant vectors of the lattice  $\mathcal{L} = \mathcal{L}(\mathbf{B})$ . The basic idea is to compute them in a “bottom-up” fashion, by iteratively computing the relevant vectors of the lower-rank lattices  $\mathcal{L}_1 = \mathcal{L}(\mathbf{b}_1)$ ,  $\mathcal{L}_2 = \mathcal{L}(\mathbf{b}_1, \mathbf{b}_2)$ ,  $\mathcal{L}_3 = \mathcal{L}(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$ , etc. Clearly, the relevant vectors of  $\mathcal{L}_1 = \mathcal{L}(\mathbf{b}_1)$  are just  $\{\pm\mathbf{b}_1\}$ .

To iteratively compute the relevant vectors of  $\mathcal{L}_i$ , we use a CVP oracle for  $\mathcal{L}_{i-1}$ , which we can implement using the “walk” step with the already-computed relevant vectors of  $\mathcal{L}_{i-1}$ . Here we use [Fact 2.1](#), which says that every relevant vector  $\mathbf{v} \in \mathcal{L}_i$  is the (unique, up to sign) shortest vector of some coset  $\mathbf{t} + 2\mathcal{L}_i$  for  $\mathbf{t} \in \mathcal{L}_i$ . So if we find a shortest vector of each of the  $2^n$  such cosets  $\mathbf{t} + 2\mathcal{L}_i$ , we will find every relevant vector. (We might find other non-relevant vectors as well, but these do not interfere with the “walk” step, so they can safely be retained. In fact, there is a way to check for relevance if desired.)

For each coset  $\mathbf{t} + 2\mathcal{L}_i$ , we find a shortest element using our CVP oracle for  $\mathcal{L}_{i-1}$ . This is done by partitioning the coset  $\mathbf{t} + 2\mathcal{L}_i$  according to the  $\mathbf{b}_i$  coefficient, which yields several “slices” that each correspond to some coset of  $2\mathcal{L}_{i-1}$ . By using an LLL-reduced basis  $\mathbf{B}$  we can ensure that the number of slices we need to inspect is only  $2^{O(n)}$ . For each of the slices we find a shortest element in the corresponding coset, and then take a shortest one overall to get a shortest element of  $\mathbf{t} + 2\mathcal{L}_i$ .

Overall, to find the relevant vectors of  $\mathcal{L}_i$  given those of  $\mathcal{L}_{i-1}$ , we need to solve CVP on  $2^n$  cosets of  $2\mathcal{L}_i$ , each of which reduces to solving CVP on  $2^{O(n)}$  cosets of  $2\mathcal{L}_{i-1}$ , each of which takes  $2^{O(n)}$  time using the “walk” step. So the relevant vectors of  $\mathcal{L}_i$  can be computed in  $2^{O(n)}$  time overall, and hence so can the relevant vectors of  $\mathcal{L} = \mathcal{L}(\mathbf{B})$ .

## References

- [ADRS15] D. Aggarwal, D. Dadush, O. Regev, and N. Stephens-Davidowitz. Solving the shortest vector problem in  $2^n$  time using discrete Gaussian sampling. In *STOC*, pages 733–742. 2015. Page 2.
- [ADS15] D. Aggarwal, D. Dadush, and N. Stephens-Davidowitz. Solving the closest vector problem in  $2^n$  time - the discrete Gaussian strikes again! In *FOCS*, pages 563–582. 2015. Page 2.
- [AKS01] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610. 2001. Page 2.
- [Kan83] R. Kannan. Improved algorithms for integer programming and related lattice problems. In *STOC*, pages 193–206. 1983. Page 2.
- [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 2.
- [Mic08] D. Micciancio. Efficient reductions among lattice problems. In *SODA*, pages 84–93. 2008. Page 1.
- [MV10] D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *STOC*, pages 351–358. 2010. Page 2.