5.1 The Closest Vector Problem

5.1.1 Inhomogeneous linear equations

Recall that, in our first lecture, we considered "short" non-zero solutions $\boldsymbol{z}$ to linear equations of the form

$$\langle \boldsymbol{a}, \boldsymbol{z} \rangle \equiv 0 \bmod q \ . \tag{5.1}$$

We showed that the set of solutions

$$\mathcal{L} := \{ \boldsymbol{z} \in \mathbb{Z}^n \ : \ \langle \boldsymbol{a}, \boldsymbol{z} \rangle \equiv 0 \bmod q \}$$

formed a lattice. I.e., $\mathcal{L}$ is a subgroup of $\mathbb{R}^n$, and there exists a basis $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ of linearly independent vectors such that $\mathcal{L}$ is the set of integer linear combinations of the basis vectors,

$$\mathcal{L} = \left\{ \sum c_i \boldsymbol{b}_i \ : \ c_i \in \mathbb{Z} \right\} .$$

We then generalized our discussion to consider short vectors in arbitrary lattices $\mathcal{L} \subset \mathbb{R}^n$—i.e., any subgroup of $\mathbb{R}^n$ that is the set of integer linear combinations of some basis vectors.

It is therefore natural to consider short solutions to the *inhomogeneous* variant of Eq. (5.1),

$$\langle \boldsymbol{a}, \boldsymbol{z} \rangle \equiv c \bmod q \ . \tag{5.2}$$

For $c \not\equiv 0 \bmod q$, the set of solutions to such an equation do *not* form a lattice. In particular $\boldsymbol{0}$ is not a solution (and, if $2c \not\equiv 0 \bmod q$, then the sum of two solutions will never be a solution). Rather, they form a lattice *coset*. I.e., if $\boldsymbol{z} \in \mathbb{Z}^n$ is a solution to Eq. (5.2), then clearly $\boldsymbol{x} \in \mathbb{Z}^n$ is also a solution to Eq. (5.2) if and only if the difference $\boldsymbol{x} - \boldsymbol{z}$ is a solution to the *homogeneous* equation Eq. (5.1). So, with $\mathcal{L}$ defined as before, we see that the set of all solutions to Eq. (5.2) is the lattice coset $\mathcal{L} + \boldsymbol{z}$.

It is trivial to find a solution $\boldsymbol{z} \in \mathbb{Z}^n$ to Eq. (5.2) (though not necessarily a short one!). So, the problem of finding a short solution to Eq. (5.2) is equivalent to finding short vectors in a certain lattice coset $\mathcal{L} + \boldsymbol{z}$, given a description of $\mathcal{L}$ and $\boldsymbol{z}$. And, the problem is equivalent to finding a *lattice* vector $\boldsymbol{y} \in \mathcal{L}$ that is *close* to $\boldsymbol{z}$. Indeed, if $\boldsymbol{y} \in \mathcal{L}$ is close to $\boldsymbol{z}$, then $\boldsymbol{y} - \boldsymbol{z}$ is a short vector in $\mathcal{L} - \boldsymbol{z}$, and if $\boldsymbol{x} \in \mathcal{L} - \boldsymbol{z}$ is a short vector, then $\boldsymbol{x} + \boldsymbol{z}$ is a close lattice vector to $\boldsymbol{z}$.

Just like before, we will study these questions for arbitrary lattices $\mathcal{L}$ and vectors $\boldsymbol{z}$—not just those that come from linear modular equations.

5.1.2 The Closest Vector Problem

Given a target vector $\boldsymbol{t} \in \mathbb{R}^n$ and a lattice $\mathcal{L} \subset \mathbb{R}^n$, we define

$$\mathrm{dist}(\mathcal{L}, \boldsymbol{t}) := \min_{\boldsymbol{x} \in \mathcal{L}} \|\boldsymbol{x} - \boldsymbol{t}\| \ .$$

Definition 5.1. *For any approximation factor $\gamma = \gamma(n) \geq 1$, the γ-Closest Vector Problem (γ-CVP) is defined as follows. The input is a basis for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a target vector $\boldsymbol{t} \in \mathbb{R}^n$. The goal is to output a lattice vector $\boldsymbol{x} \in \mathcal{L}$ with $\|\boldsymbol{x} - \boldsymbol{t}\| \leq \gamma \cdot \mathrm{dist}(\boldsymbol{t}, \mathcal{L})$.*

As we observed above, finding a lattice vector $\boldsymbol{x}$ with $\|\boldsymbol{x} - \boldsymbol{t}\| \leq d$ is trivially equivalent to finding a vector $\boldsymbol{y} \in \mathcal{L} - \boldsymbol{t}$ with $\|\boldsymbol{y}\| \leq d$. So, we define this alternative equivalent problem, and we will freely switch between the two problems. (Note that a shortest vector in $\mathcal{L} - \boldsymbol{t}$ has length exactly $\mathrm{dist}(\boldsymbol{t}, \mathcal{L})$.)

Definition 5.2. *For any approximation factor $\gamma = \gamma(n) \geq 1$, γ-CVP' is defined as follows. The input is a basis for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a* shift *vector $\boldsymbol{t} \in \mathbb{R}^n$. The goal is to output a vector $\boldsymbol{s} \in \mathcal{L} - \boldsymbol{t}$ with $\|\boldsymbol{s}\| \leq \gamma \cdot \mathrm{dist}(\boldsymbol{t}, \mathcal{L})$.*

5.1.3 Hardness of CVP and relationship with SVP

It is straightforward to show that the exact version of CVP (i.e., 1-CVP) is NP-complete. (Try to prove it yourself! The original proof is due to van Emde Boas [vEB81].) So, CVP's hardness was known long before that of SVP. Indeed, γ-CVP is now known to be NP-complete for any $\gamma \leq n^{c/\log\log n}$ [DKRS03]. (Recall that γ-SVP is also known to be "hard" for such parameters, but for a much weaker notion of "hardness.")

Given this history and the definitions of the problems (in which CVP is essentially the inhomogeneous form of SVP), it is natural to guess that one can reduce γ-SVP to γ-CVP. In fact, a totally trivial reduction "almost works." Since CVP is the problem of finding a short vector in a lattice coset $\mathcal{L} - \boldsymbol{t}$, and $\mathcal{L}$ itself is a lattice coset (the zero coset), we would like to solve SVP just by calling our CVP oracle on the "coset" $\mathcal{L}$. Of course, this fails rather spectacularly because the shortest vector in this coset is $\mathbf{0}$, and SVP of course does not allow us to output the zero vector.

Goldreich, Micciancio, Safra, and Seifert managed to get around this annoying issue in order to prove the following result.

Theorem 5.3 ([GMSS99])**.** *For any $\gamma = \gamma(n) \geq 1$, there is an efficient reduction from γ-SVP to γ-CVP.*

Proof. We reduce to γ-CVP'. Let $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ be the input basis for the lattice $\mathcal{L} \subset \mathbb{R}^n$. Let $\boldsymbol{v} \in \mathcal{L}$ be a vector of length $\lambda_1(\mathcal{L})$. The key observations from [GMSS99] are simply that (1) $\boldsymbol{v} = \sum a_i \boldsymbol{b}_i$ must have at least one odd coordinate, $a_j \not\equiv 0 \bmod 2$; and (2) the set of vectors whose jth coordinate is odd is a lattice coset! In particular, let $\mathcal{L}_j$ be the lattice spanned by "the input basis with its jth vector doubled," $(\boldsymbol{b}_1, \ldots, 2\boldsymbol{b}_j, \ldots, \boldsymbol{b}_n)$. I.e., $\mathcal{L}_j$ is the set of all lattice vectors with *even* jth coordinate. Then, $\mathcal{L}_j + \boldsymbol{b}_j$ is the set of all vectors with *odd* jth coordinate. In particular, there exists some j such that $\mathcal{L}_j + \boldsymbol{b}_j$ contains a vector of length $\lambda_1(\mathcal{L})$.

So, if we call our γ-CVP' oracle on $\mathcal{L}_j + \boldsymbol{b}_j$ for all $j = 1, \ldots, n$, one of the results must be a lattice vector of length at most $\gamma \cdot \lambda_1(\mathcal{L})$. All of the output vectors are non-zero, so by returning the shortest vector that we see, we obtain a solution γ-SVP. □

5.2 Babai's Algorithm

5.2.1 A geometric interpretation of the Gram-Schmidt orthogonalization

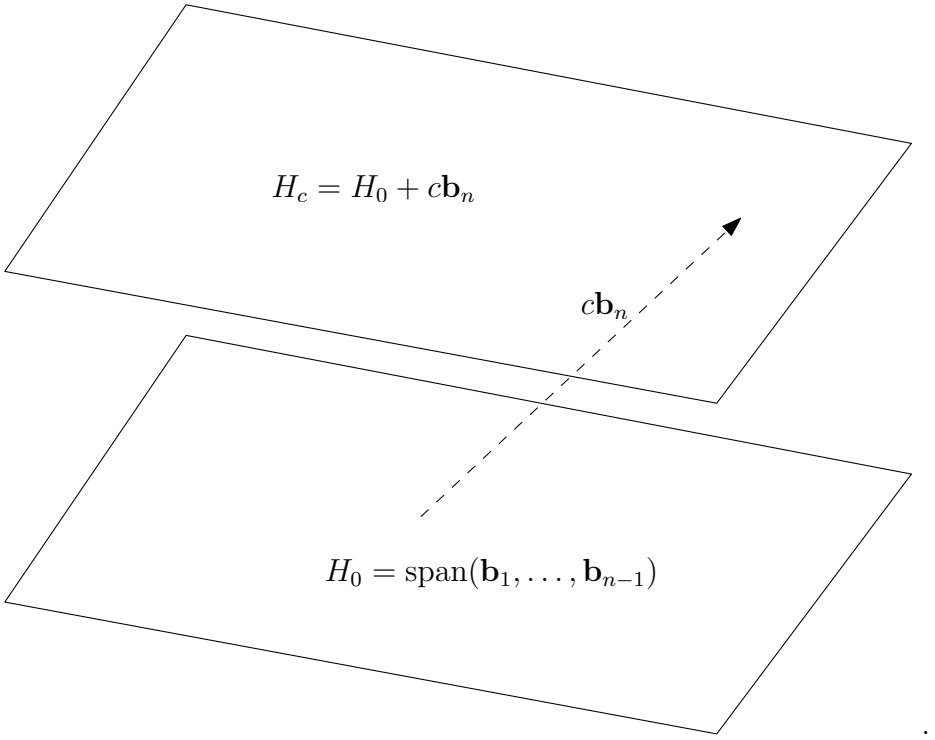
Recall from Lecture 2 that the Gram-Schmidt orthogonalization $(\widetilde{\boldsymbol{b}}_1, \ldots, \widetilde{\boldsymbol{b}}_n)$ of a basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ is defined as

$$\widetilde{\boldsymbol{b}}_i := \pi_{\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}\}^\perp}(\boldsymbol{b}_i) \ .$$

The nth Gram-Schmidt vector, $\widetilde{\boldsymbol{b}}_n$, has a very nice geometric interpretation. In particular, consider the hyperplane (i.e., affine subspace) defined by all vectors $\boldsymbol{x} \in \mathbb{R}^n$ (not necessarily lattice vectors) whose last coordinate is c for some fixed c,

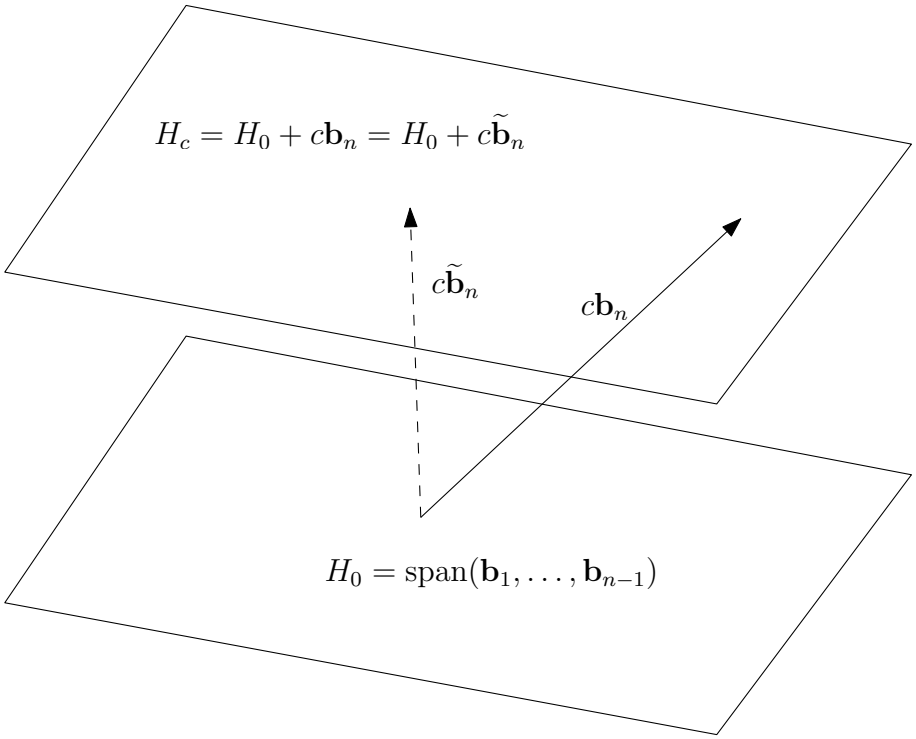
$$H_c := \left\{ \sum_{i=1}^{n-1} a_i \boldsymbol{b}_i + c\boldsymbol{b}_n \right\} .$$

Here is an attempt at a drawing of $(n-1)$-dimensional hyperplanes in IPE:



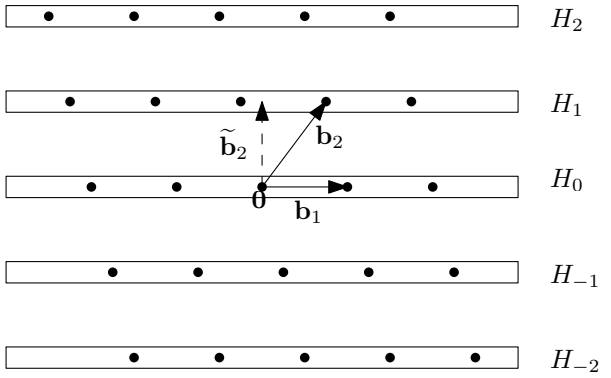
.

Notice in particular that $H_c = H_0 + c\boldsymbol{b}_n$. But, since $\boldsymbol{b}_n$ is typically not orthogonal to H_0, the distance between H_0 and H_c to the origin will typically not be $c\|\boldsymbol{b}_n\|$. Instead, the distance will be the length of the component of $c\boldsymbol{b}_n$ that is orthogonal to H_0. I.e., it will be exactly $c\|\widetilde{\boldsymbol{b}}_n\|$:



.

The last coordinate of a lattice vector in the basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ is an integer (as are all the other coordinates), so each lattice point lies in a hyperplane H_c for integer c. I.e., the lattice can be partitioned into $(n-1)$-dimensional *lattice hyperplanes*, $\{, \ldots, H_{-1}, H_0, H_1, \ldots, \}$, with adjacent hyperplanes separated by $\widetilde{\boldsymbol{b}}_n$. Inside each of these hyperplanes is a copy of the sublattice $\mathcal{L}'$ generated by the first $n-1$ basis vectors. But, in the ith hyperplane, the lattice $\mathcal{L}'$ is shifted by $i\boldsymbol{b}_n$. Here's what this looks like in two dimensions:

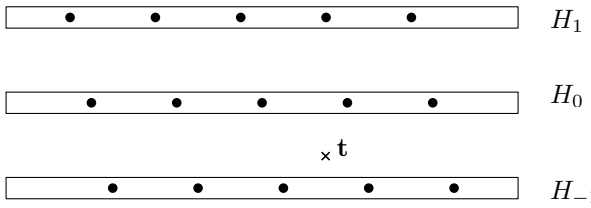


.

Of course, the other Gram-Schmidt vectors can be interpreted similarly. In particular, the sublattice $\mathcal{L}'$ generated by the first $n-1$ basis vectors can itself be partitioned into $(n-2)$-dimensional hyperplanes, each separated by $\widetilde{\boldsymbol{b}}_{n-1}$. Since H_c contains a copy of $\mathcal{L}'$ (shifted by $c\boldsymbol{b}_n$), inside of H_c we will find the lattice hyperplanes of $\mathcal{L}'$, also shifted by $c\boldsymbol{b}_n$.

5.2.2 Babai's nearest hyperplane algorithm

With this picture in mind, the idea behind Babai's algorithm is quite simple. Given a target vector $\boldsymbol{t} \in \mathbb{R}^n$ and a lattice $\mathcal{L} \subset \mathbb{R}^n$ with some basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$, it seems reasonable to look for a lattice vector that is close to $\boldsymbol{t}$ inside the closest lattice hyperplane to $\boldsymbol{t}$. So, Babai's *nearest hyperplane* algorithm [Bab86] works by first identifying the nearest lattice hyperplane H_c to $\boldsymbol{t}$. For example, Babai's algorithm would choose the hyperplane H_{-1} in this example (though the closest vector to $\boldsymbol{t}$ happens to lie in H_0):



Recall that H_c contains a *coset* $\mathcal{L}' + c\boldsymbol{b}_n$ of the sublattice $\mathcal{L}'$ spanned by the first $n-1$ basis vectors. $\mathcal{L}' + c\boldsymbol{b}_n$ can itself be subdivided into $(n-2)$-dimensional lattice hyperplanes (separated by $\widetilde{\boldsymbol{b}}_{n-1}$). So, the algorithm proceeds by picking the closest $(n-2)$-dimensional lattice hyperplane to $\boldsymbol{t}$ amongst these. It then chooses an $(n-3)$-dimensional lattice hyperplane inside of the $(n-2)$-dimensional hyperplane, etc., until eventually it has found a 0-dimensional lattice hyperplane—i.e., a lattice vector.

This is the typical presentation of Babai's algorithm, but in class, Huck Bennett suggested a different view that is perhaps easier to understand (and is also easier to convert into (pseudo)code). It most naturally solves γ-CVP'. I.e., it finds a short vector in the coset $\mathcal{L} - \boldsymbol{t}$. Recall from Lecture 2 that we can rotate space so that our basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ is upper triangular and looks like

$$\begin{pmatrix} \|\widetilde{\boldsymbol{b}}_1\| & \mu_{1,2}\|\widetilde{\boldsymbol{b}}_1\| & \mu_{1,3}\|\widetilde{\boldsymbol{b}}_1\| & \cdots & \mu_{1,n}\|\widetilde{\boldsymbol{b}}_1\| \\ 0 & \|\widetilde{\boldsymbol{b}}_2\| & \mu_{2,3}\|\widetilde{\boldsymbol{b}}_2\| & \cdots & \mu_{2,n}\|\widetilde{\boldsymbol{b}}_2\| \\ 0 & 0 & \|\widetilde{\boldsymbol{b}}_3\| & \cdots & \mu_{3,n}\|\widetilde{\boldsymbol{b}}_3\| \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \|\widetilde{\boldsymbol{b}}_n\| \end{pmatrix} . \tag{5.3}$$

We can of course also write the coordinates of our shift vector in this rotated space

$$\boldsymbol{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{pmatrix} .$$

In this view, Babai's algorithm works by first finding an element in $\mathcal{L} - \boldsymbol{t}$ whose nth coordinate is minimal. I.e., it takes $c := \lfloor t_n / \|\widetilde{\boldsymbol{b}}_n\| \rceil$ to be the integer that minimizes $c\|\widetilde{\boldsymbol{b}}_n\| - t_n$ and sets $\boldsymbol{s} := c\boldsymbol{b}_n - \boldsymbol{t} \in \mathcal{L} - \boldsymbol{t}$. Since the nth coordinate of any vector in $\mathcal{L} - \boldsymbol{t}$ must differ from $-t_n$ by an integer multiple of $\widetilde{\boldsymbol{b}}_n$, this process guarantees that $\boldsymbol{s}$ has the smallest possible nth coordinate. The algorithm then leaves the nth coordinate fixed and does the same for the remaining coordinates. I.e., in the ith step, the algorithm takes $c := \lfloor s_{n-i+1} / \|\widetilde{\boldsymbol{b}}_{n-i+1}\| \rceil$

and updates $\boldsymbol{s}$ to $\boldsymbol{s} - c\boldsymbol{b}_{n-1}$. In the end, we are left with a vector $\boldsymbol{s}$ in the original coset that is relatively short.

For completeness, we now present the pseudocode of Babai's algorithm. Notice that it is quite simple (although perhaps a bit opaque without the above discussion).

1. Set $\boldsymbol{s} := -\boldsymbol{t}$.
2. For $i = 1, \ldots, n$,
   (a) Set $c := \lfloor \langle \boldsymbol{s}, \widetilde{\boldsymbol{b}}_{n-i+1} \rangle / \|\widetilde{\boldsymbol{b}}_{n-i+1}\|^2 \rceil$.
   (b) Update $\boldsymbol{s}$ to $\boldsymbol{s} - c\boldsymbol{b}_{n-i+1}$.
3. Output $\boldsymbol{s}$.

Before moving on, convince yourself that these are really three descriptions of the same algorithm. In what follows, we will use the notation given by the pseudocode.

5.2.3 Analysis

We first note some basic properties of Babai's algorithm.

Claim 5.4. *After Babai's algorithm terminates, the output vector $\boldsymbol{s}$ lies in the (hyper-)rectangle*

$$\left\{ \sum a_i \widetilde{\boldsymbol{b}}_i \ : \ |a_i| \leq 1/2 \right\} .$$

In particular,

$$\|\boldsymbol{s}\|^2 \leq \frac{1}{4} \sum_i \|\widetilde{\boldsymbol{b}}_i\|^2 ,$$

and this bound is tight.

Proof. Since the Gram-Schmidt vectors are orthogonal, any vector $\boldsymbol{x} \in \mathbb{R}^n$ can be written as

$$\boldsymbol{x} = \sum \frac{\langle \widetilde{\boldsymbol{b}}_i, \boldsymbol{x} \rangle}{\|\widetilde{\boldsymbol{b}}_i\|^2} \cdot \widetilde{\boldsymbol{b}}_i .$$

Then, we simply recall that, in the ith step, the algorithm fixes $\langle \widetilde{\boldsymbol{b}}_{n-i+1}, \boldsymbol{s} \rangle / \|\widetilde{\boldsymbol{b}}_{n-i+1}\|^2$ so that its magnitude is at most $1/2$.

To see that the bound is tight, simply start with a point at a "corner" of the rectangle. E.g., take $\boldsymbol{t} := \frac{1}{2} \sum \widetilde{\boldsymbol{b}}_i$. Notice that the output will itself be a corner of the rectangle. (*Which* corner simply depends on how we choose to round $1/2$.) □

Note that the distance bound in Claim 5.4 suggests that we should prefer bases that minimize $\sum \|\widetilde{\boldsymbol{b}}_i\|^2$. Since $\det(\mathcal{L}) = \prod \|\widetilde{\boldsymbol{b}}_i\|$ is fixed, this suggests that we should prefer bases that Gram-Schmidt vectors of fairly consistent length. (Recall that, if $a_1, \ldots, a_n > 0$, then $\sum a_i / \prod a_i^{1/n}$ is minimized when $a_i = a_j$ for all i, j.)

Indeed, at this point it might seem that we are done with our analysis We have shown that Babai's algorithm outputs fairly short vectors; our bound is tight; and we seem to have

the right metric for judging how "good" a basis is for Babai's algorithm—a basis is apparently better if its Gram-Schmidt lengths are consistent, as measured by $\sum \|\widetilde{\boldsymbol{b}}_i\|^2 / \prod \|\widetilde{\boldsymbol{b}}_i\|^{2/n}$.

But, remember the definitions of γ-CVP and γ-CVP'. They do not ask for a vector that is closer/shorter than some *fixed* bound. Instead, they ask for a vector that is within a factor γ of the best possible! Since $\boldsymbol{t}$ can be arbitrarily close to the lattice, the distance bound in Claim 5.4 does not give us any bound at all on the approximation factor γ!

So, we need a bit more analysis. As it happens, just like in Lecture 2, we will find that a nice metric of how "good" a basis is the *decay rate* of the Gram-Schmidt vectors, $\max_{i \geq j} \|\widetilde{\boldsymbol{b}}_j\| / \|\widetilde{\boldsymbol{b}}_i\|$—*not* how consistent the lengths of the Gram-Schmidt vectors are. First, we will notice a nice geometric interpretation of the behavior of Babai's algorithm.

Claim 5.5. *The output vector $\boldsymbol{s}$ depends only on the coset $\mathcal{L} - \boldsymbol{t}$ of the input.*[1] *(I.e., if we replace $\boldsymbol{t}$ by $\boldsymbol{t} + \boldsymbol{x}$ for some lattice vector $\boldsymbol{x} \in \mathcal{L}$, the output is unchanged.)*

Proof. It suffices to note that the output is unchanged when we replace $\boldsymbol{t}$ by $\boldsymbol{t} + \boldsymbol{b}_i$. Let $\boldsymbol{s}_0, \ldots, \boldsymbol{s}_n$ be the sequence of values of $\boldsymbol{s}$ set by the algorithm on input $\boldsymbol{t}$ (e.g., $\boldsymbol{s}_0 = -\boldsymbol{t}$ and $\boldsymbol{s}_n$ is the output of the algorithm) with corresponding sequence $c_1, \ldots, c_n$, and let $\boldsymbol{s}'_0, \ldots, \boldsymbol{s}'_n$ and $c'_1, \ldots, c'_n$ be this sequence on input $\boldsymbol{t} + \boldsymbol{b}_i$. In particular, $\boldsymbol{s}_0 = \boldsymbol{s}'_0 + \boldsymbol{b}_i$. A simple induction argument shows that, for $j \leq n - i$, $c_j = c'_j$ and therefore

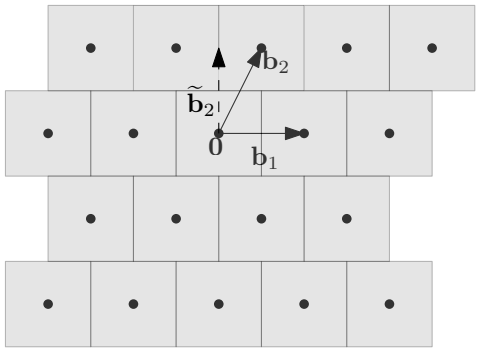
$$\boldsymbol{s}_j = \boldsymbol{s}_{j-1} - c_j \boldsymbol{b}_{n-j+1} = \boldsymbol{s}'_{j-1} + \boldsymbol{b}_i - c'_j \boldsymbol{b}_{n-j+1} = \boldsymbol{s}'_j + \boldsymbol{b}_i \; .$$

For $j = n - i + 1$, we have

$$c_j = \frac{\langle \widetilde{\boldsymbol{b}}_i, \boldsymbol{s}_{j-1} \rangle}{\|\widetilde{\boldsymbol{b}}_i\|^2} = \frac{\langle \widetilde{\boldsymbol{b}}_i, \boldsymbol{s}'_{j-1} + \boldsymbol{b}_i \rangle}{\|\widetilde{\boldsymbol{b}}_i\|^2} = \frac{\langle \widetilde{\boldsymbol{b}}_i, \boldsymbol{s}'_{j-1} \rangle}{\|\widetilde{\boldsymbol{b}}_i\|^2} + 1 = c'_j + 1 \; .$$

Therefore, $\boldsymbol{s}_j = \boldsymbol{s}_{j-1} - c_j \boldsymbol{b}_i = \boldsymbol{s}'_{j-i} - c'_j \boldsymbol{b}_i = \boldsymbol{s}'_j$. Since the behavior of the algorithm is entirely determined by $\boldsymbol{s}_j$, it follows that the output must be identical. □

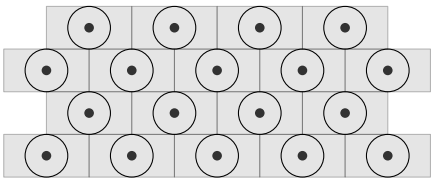
Note that, when taken together, Claims 5.4 and 5.5 completely determine the behavior of Babai's algorithm. In particular, they show us that Babai's algorithm divides space into (hyper-)rectangles according to the Gram-Schmidt vectors:



[1] Equivalently, the rectangle from Claim 5.4 is a fundamental body of the lattice—it contains a unique representative from each lattice coset (ignoring the boundary).

The output of Babai's algorithm depends only on where the input lands modulo this tiling.[2] Equivalently, in terms of CVP as opposed to CVP', Babai's algorithm always outputs the lattice vector at the center of the rectangle containing the target.

From this picture, we see that if $\operatorname{dist}(\boldsymbol{t}, \mathcal{L}) \leq \min \|\widetilde{\boldsymbol{b}}_i\|/2$, then Babai's algorithm must always output an *exact* correct answer—i.e., an exact closest vector to the target or an exact shortest vector in the coset. In particular, we can place around each lattice point a ball of radius $\min \|\widetilde{\boldsymbol{b}}_i\|/2$:



Since these balls are non-overlapping and entirely contained in the rectangle with the same center, Babai's algorithm must output an *exact* correct answer in this case. (Is this the only case when the algorithm outputs an exact answer?)

From this, we see that Babai's algorithm actually solves γ-CVP for

$$\gamma \leq \frac{\left(\sum \|\widetilde{\boldsymbol{b}}_i\|^2\right)^{1/2}}{\min \|\widetilde{\boldsymbol{b}}_i\|} .$$

Unfortunately, there are lattices for which this quantity must be arbitrarily large for *any* basis. (Consider, for example, the lattice generated by $\boldsymbol{e}_1$ and $s\boldsymbol{e}_2$ for arbitrarily large s.) Luckily, we can do better with slightly more careful analysis. In particular, just like in Lecture 2, we will see that it is actually a good thing when the "later" Gram-Schmidt vectors are long.

Theorem 5.6 ([Bab86])**.** *Babai's algorithm solves γ-CVP with*

$$\gamma \leq \left(1 + \max_i \frac{\sum_{j=1}^{i} \|\widetilde{\boldsymbol{b}}_j\|^2}{\|\widetilde{\boldsymbol{b}}_i\|^2}\right)^{1/2} \leq \sqrt{n+1} \cdot \max_{i \geq j} \frac{\|\widetilde{\boldsymbol{b}}_j\|}{\|\widetilde{\boldsymbol{b}}_i\|} .$$

Proof. The key observation here is that, even when $\operatorname{dist}(\boldsymbol{t}, \mathcal{L}) > \|\widetilde{\boldsymbol{b}}_i\|/2$, Babai's algorithm might choose *some* coordinates correctly. For example, if $\operatorname{dist}(\boldsymbol{t}, \mathcal{L}) < \|\widetilde{\boldsymbol{b}}_n\|/2$, then clearly any closest vector to $\boldsymbol{t}$ in the lattice must lie in the nearest lattice hyperplane to $\boldsymbol{t}$. (Since all vectors in other lattice hyperplanes are at distance at least $\|\widetilde{\boldsymbol{b}}_n\|/2 > \operatorname{dist}(\boldsymbol{t}, \mathcal{L})$ away from $\boldsymbol{t}$.) So, Babai's algorithm must choose the correct nth coordinate in this case.

More generally, if $\operatorname{dist}(\boldsymbol{t}, \mathcal{L}) < \|\widetilde{\boldsymbol{b}}_j\|/2$ for all $j \geq i$ for some i, then Babai's algorithm will choose the "correct" coordinate for all $j \geq i$, and the output vector will actually satisfy

$$\|\boldsymbol{s}\|^2 \leq \frac{1}{2} \cdot \sum_{j=1}^{i} \|\widetilde{\boldsymbol{b}}_j\|^2 + \operatorname{dist}(\boldsymbol{t}, \mathcal{L})^2 .$$

[2] In class, we considered an alternative to Babai's nearest hyperplane algorithm in which we simply wrote the shift $\boldsymbol{t} = \sum a_i \boldsymbol{b}_i$ in terms of the lattice basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ and returned the "fractional part" $-\sum(a_i - \lfloor a_i \rceil)\boldsymbol{b}_i$. We observed that this algorithm can be seen as dividing space into parallelepipeds, not necessarily rectangles. Indeed, Babai studied this algorithm as well in his original paper [Bab86].

If $\mathrm{dist}(\boldsymbol{t}, \mathcal{L}) < \|\widetilde{\boldsymbol{b}}_i\|/2$ for all i, then we know that the output will be correct, so we may assume that there is an $i \in \{1, \ldots, n\}$ such that $\mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \geq \|\widetilde{\boldsymbol{b}}_i\|/2$, and we may take i to be maximal satisfying this property. Then,

$$\|\boldsymbol{s}\|^2 \leq \frac{1}{2} \cdot \sum_{j=1}^{i} \|\widetilde{\boldsymbol{b}}_j\|^2 + \mathrm{dist}(\boldsymbol{t}, \mathcal{L})^2 \leq \frac{\mathrm{dist}(\boldsymbol{t}, \mathcal{L})^2}{\|\widetilde{\boldsymbol{b}}_i\|^2} \cdot \sum_{j=1}^{i} \|\widetilde{\boldsymbol{b}}_j\|^2 + \mathrm{dist}(\boldsymbol{t}, \mathcal{L})^2 \ ,$$

as claimed. □

5.2.4 Plugging in a good basis

Do we happen to know any types of lattice bases with bounded "decay rate" $\max_{i \geq j} \|\widetilde{\boldsymbol{b}}_j\|/\|\widetilde{\boldsymbol{b}}_i\|$? Yes!

Corollary 5.7 ([Bab86])**.** *For any $\delta \in (1/4, 1]$, Babai's algorithm with a δ-LLL basis solves γ-CVP for $\gamma \leq \sqrt{n}(\delta - 1/4)^{-n/2}$. In particular, there is an efficient algorithm that solves γ-CVP for $\gamma = 2^{O(n)}$.*

Proof. Simply recall from Lecture 2 that a δ-LLL basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ satisfies $\|\widetilde{\boldsymbol{b}}_{i+1}\|^2 \geq (\delta - 1/4) \cdot \|\widetilde{\boldsymbol{b}}_i\|^2$ for all i. □

Just like in the SVP case, slightly better approximation factors are known using BKZ bases (which generalize LLL bases). In particular, for any constant $C > 0$, there is an efficient algorithm that solves $2^{Cn \log\log n / \log n}$-CVP.

References

[Bab86] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[DKRS03] I. Dinur, G. Kindler, R. Raz, and S. Safra. Approximating CVP to within almost-polynomial factors is NP-hard. *Combinatorica*, 23(2):205–243, 2003.

[GMSS99] Oded Goldreich, Daniele Micciancio, Shmuel Safra, and Jean-Paul Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Information Processing Letters*, 71(2):55 – 61, 1999.

[vEB81] P. van Emde Boas. Another NP-complete problem and the complexity of computing short vectors in a lattice. Technical report, University of Amsterdam, Department of Mathematics, Netherlands, 1981. Technical Report 8104.