In this lecture, we describe an approximation algorithm to the Closest Vector Problem (CVP). This algorithm, known as the **Nearest Plane Algorithm**, was developed by L. Babai in 1986. It obtains a $2(\frac{2}{\sqrt{3}})^n$ approximation ratio, where n is the rank of the lattice. In many applications, this algorithm is applied for a constant n; in such cases, we obtain a constant approximation factor.

One can define approximate-CVP as a search problem, as an optimization problem, or as a decision problem (where the latter is often known as a *gap* problem). In the following definitions, $\gamma \geq 1$ is the approximation factor. By setting $\gamma = 1$ we obtain the exact version of the problems.

DEFINITION 1 (CVP_γ, SEARCH) *Given a basis $B \in \mathbb{Z}^{m \times n}$ and a point $t \in \mathbb{Z}^m$, find a point $x \in \mathcal{L}(B)$ such that $\forall y \in \mathcal{L}(B), \|x - t\| \leq \gamma \|y - t\|$.*

DEFINITION 2 (CVP_γ, OPTIMIZATION) *Given a basis $B \in \mathbb{Z}^{m \times n}$ and a point $t \in \mathbb{Z}^m$, find $r \in \mathbb{Q}$ such that $\mathrm{dist}(t, \mathcal{L}(B)) \leq r \leq \gamma \cdot \mathrm{dist}(t, \mathcal{L}(B))$.*

DEFINITION 3 (CVP_γ, DECISION) *Given a basis $B \in \mathbb{Z}^{m \times n}$, a point $t \in \mathbb{Z}^m$ and $r \in \mathbb{Q}$, decide if $\mathrm{dist}(t, \mathcal{L}(B)) \leq r$ or $\mathrm{dist}(t, \mathcal{L}(B)) > \gamma \cdot r$.*

Babai's nearest plane algorithm solves the search variant of CVP_γ for $\gamma = 2(\frac{2}{\sqrt{3}})^n$. It is easy to see that this implies a solution to the other two variants of CVP_γ as they are not harder than the search version. For simplicity, the algorithm we present here achieves $\gamma = 2^{\frac{n}{2}}$. It is possible to achieve $\gamma = 2(\frac{2}{\sqrt{3}})^n$ by a straightforward modification of the parameters.

1 The Nearest Plane Algorithm

The algorithm has two main steps. First, it applies the LLL reduction to the input lattice. It then looks for an integer combination of the basis vectors that is close to the target vector t. This step is essentially the same as one inner loop in the reduction step of the LLL algorithm.

INPUT: Basis $B \in \mathbb{Z}^{m \times n}$, $t \in \mathbb{Z}^m$
OUTPUT: A vector $x \in \mathcal{L}(B)$ such that $\|x - t\| \leq 2^{\frac{n}{2}} \mathrm{dist}(t, \mathcal{L}(B))$

1. Run δ-LLL on B with $\delta = \frac{3}{4}$
2. $b \leftarrow t$
   for j = n to 1 **do**
      $b \leftarrow b - c_j b_j$ where $c_j = \lceil \langle b, \tilde{b}_j \rangle / \langle \tilde{b}_j, \tilde{b}_j \rangle \rfloor$
   Output $t - b$

It can be seen that this algorithm runs in polynomial time in the input size; indeed, the LLL procedure runs in polynomial time and the reduction step was already analyzed in the previous class. Notice that unlike our description of the LLL algorithm, here we consider the algorithm for arbitrary lattices that are not necessarily full-rank. This will, in fact, make our analysis slightly easier.

A useful way to imagine the second step of the algorithm is the following. Consider the orthonormal set given by $\tilde{b}_1/\|\tilde{b}_1\|, \ldots, \tilde{b}_n/\|\tilde{b}_n\|$. For full-rank lattices (i.e., $m = n$) this is a basis but

in general, we need to extend it with $m - n$ additional vectors to make it an orthonormal basis of $\mathbb{R}^m$. Using such a basis, we can now write the matrix B and the vector t as follows.

$$\begin{pmatrix} \|\tilde{b}_1\| & * & \cdots & * \\ 0 & \|\tilde{b}_2\| & \cdots & * \\ \vdots & & \ddots & \vdots \\ & & & * \\ 0 & \cdots & & \|\tilde{b}_n\| \\ 0 & \cdots & & 0 \\ \vdots & \cdots & & \vdots \\ 0 & \cdots & & 0 \end{pmatrix} \begin{pmatrix} * \\ * \\ \vdots \\ * \\ * \\ * \\ \vdots \\ * \end{pmatrix}$$

The algorithm looks for an integer combination of the columns for which each coordinate $i = 1, \ldots, n$ is within $\pm\frac{1}{2}\|\tilde{b}_i\|$ of the ith coordinate of t. So our algorithm first finds a multiple of the nth matrix column that brings the nth coordinate to within $\pm\frac{1}{2}\|\tilde{b}_n\|$. It then continues to the $n-1$st column and so on. Notice that in case the lattice is not full-rank, the last $m - n$ dimensions correspond to the space orthogonal to the span of the lattice.

We now consider another equivalent description of the second step. This description is recursive and will be the most convenient for our analysis. It emphasizes the geometric nature of the nearest plane algorithm and also explains its name. See Figure 1 for an illustration.

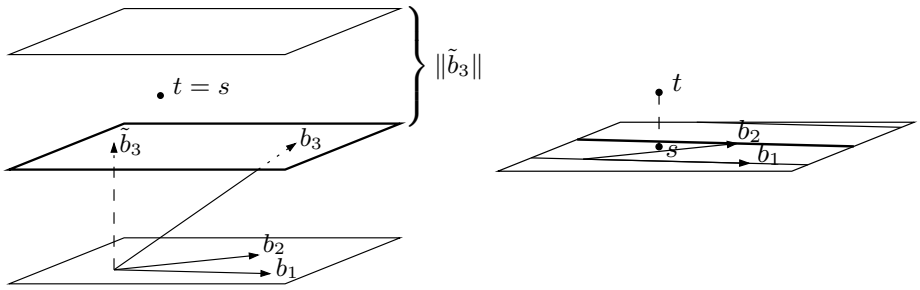

Figure 1: The nearest plane algorithm for a rank 3 lattice and the resulting rank 2 instance. The chosen hyperplanes are thicker.

1. Let s be the projection of t on $\mathrm{span}(b_1, \ldots, b_n)$.
2. Find c such that the hyperplane $c\tilde{b}_n + \mathrm{span}(b_1, \ldots, b_{n-1})$ is as close as possible to s.
3. Let $s' = s - cb_n$. Call recursively with s' and $\mathcal{L}(b_1, \ldots, b_{n-1})$. Let x' be the answer.
4. Return $x = x' + cb_n$.

It is easy to verify that the above is indeed equivalent to the second step of the algorithm. Our first step is to project t on $\mathrm{span}(b_1, \ldots, b_n)$. Some thought reveals that the closest lattice vector to s is the same as the closest lattice vector to t and hence this step makes sense. In Step 2 we identify one translate of the lattice $\mathcal{L}(b_1, \ldots, b_{n-1})$ where we suspect that the closest vector to s resides. It is on this translate that we recurse in Step 3. More precisely, in Steps 3 and 4 we compute a close

vector to s in $cb_n + \mathcal{L}(b_1, \ldots, b_{n-1})$. Since the latter set is not a lattice (it does not contain the zero vector), we shift it (together with s) by $-cb_n$. Then, when we obtain the answer x', we shift it back by cb_n. Hence, the answer x is indeed a close vector to s in $cb_n + \mathcal{L}(b_1, \ldots, b_{n-1})$.

2 Correctness of the Algorithm

We first notice that the algorithm never returns points that are too far away from the input point (for the case where $t \in \mathrm{span}(B)$). This follows easily from the matrix description above since each coordinate i of the output is within $\pm\frac{1}{2}\|\tilde{b_i}\|$ of that of t.

CLAIM 4 *For any $t \in \mathrm{span}(B)$, the output x of the algorithm is such that $\|x-t\|^2 \leq \frac{1}{4}\sum_{i=1}^{n}\|\tilde{b_i}\|^2$.*

Since B is LLL-reduced, we obtain the following.

CLAIM 5 *For any $t \in \mathrm{span}(B)$, the output x of the algorithm is such that $\|x - t\| \leq \frac{1}{2}2^{n/2}\|\tilde{b_n}\|$.*

PROOF: By properties of an LLL reduced basis, we have that

$$\forall 1 \leq i \leq n, \;\; \|\tilde{b_i}\| \leq 2^{\frac{n-i}{2}}\|\tilde{b_n}\|$$

and hence

$$\|x - t\|^2 \leq \frac{1}{4}\sum_{i=1}^{n}\|\tilde{b_i}\|^2 \leq \frac{1}{4}\sum_{i=1}^{n}2^{n-i}\|\tilde{b_n}\|^2 \leq \frac{1}{4}2^n\|\tilde{b_n}\|^2.$$

□

The above claim shows that when $\mathrm{dist}(t, \mathcal{L}(B)) \geq \frac{1}{2}\|\tilde{b_n}\|$, the output of the algorithm is a $2^{n/2}$ approximation to CVP. However, we still need to handle the case where t is very close to the lattice (and also the case where $t \notin \mathrm{span}(B)$). This is done in the following lemma, which completes the proof of correctness.

LEMMA 6 *For any $t \in \mathbb{Z}^m$, let $y \in \mathcal{L}(B)$ be the closest lattice point to t. Then the algorithm described above finds a point $x \in \mathcal{L}(B)$ such that $\|x - t\| \leq 2^{\frac{n}{2}}\|y - t\|$.*

PROOF: We prove by induction on the rank n that our algorithm finds a point $x \in \mathcal{L}(B)$ such that $\|x - s\| \leq 2^{\frac{n}{2}}\|y - s\|$. This yields the claim, since

$$\begin{aligned}\|x - t\|^2 &= \|s - t\|^2 + \|s - x\|^2 \\ &\leq \|s - t\|^2 + 2^n\|y - s\|^2 \\ &\leq 2^n(\|s - t\|^2 + \|y - s\|^2) = 2^n\|y - t\|^2\end{aligned}$$

where the first equality follows since $s - t$ and $s - x$ are orthogonal and the last equality follows similarly.

Now distinguish two cases. If $\|s - y\| < \frac{\|\tilde{b_n}\|}{2}$, then $y \in c\tilde{b_n} + \mathrm{span}(b_1, \ldots, b_{n-1})$, because all other hyperplanes are of distance at least $\frac{\|\tilde{b_n}\|}{2}$. Therefore, $y \in cb_n + \mathcal{L}(b_1, \ldots, b_{n-1})$. Intuitively,

this means that we identified the correct translate in Step 2. So we obtain that $y' = y - cb_n \in \mathcal{L}(b_1, \ldots, b_{n-1})$ is the closest point to s'. Hence, by our inductive assumption,

$$\begin{aligned}\|x - s\| &= \|x' - s'\| \\ &\leq 2^{\frac{n-1}{2}} \|y' - s'\| \\ &= 2^{\frac{n-1}{2}} \|y - s\| \\ &\leq 2^{\frac{n}{2}} \|y - s\|.\end{aligned}$$

Otherwise, we must have that $\|s - y\| \geq \frac{\|\tilde{b_n}\|}{2}$. In this case, it is possible that we identify the wrong translate in Step 2. However, by Claim 5, we have that

$$\|s - x\| \leq \frac{1}{2} 2^{n/2} \|\tilde{b_n}\| \leq 2^{n/2} \|s - y\|.$$

□

Finally, let us mention the following possible extension to Babai's algorithm. Instead of identifying only one translate in Step 2, we take (say) two translates and recurse on both. Such an extension (slightly) improves the approximation ratio but unfortunately runs in time exponential in the rank.