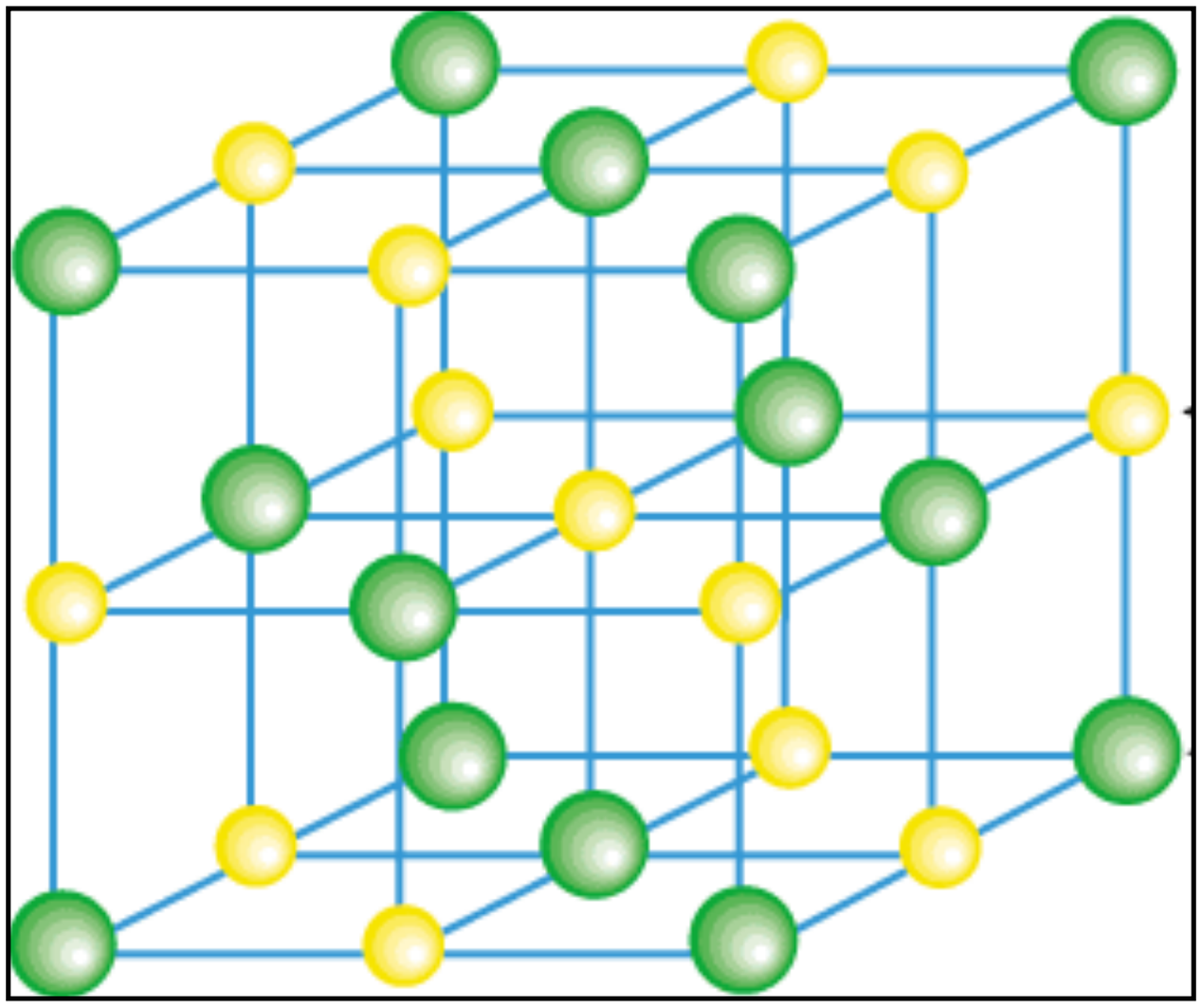
SHORTEST LATTICE VECTOR PROBLEM

- by Naman Verma
Mentors: Prof. SK Mehta, Mahesh Sreekumar Rajasree

Lattices

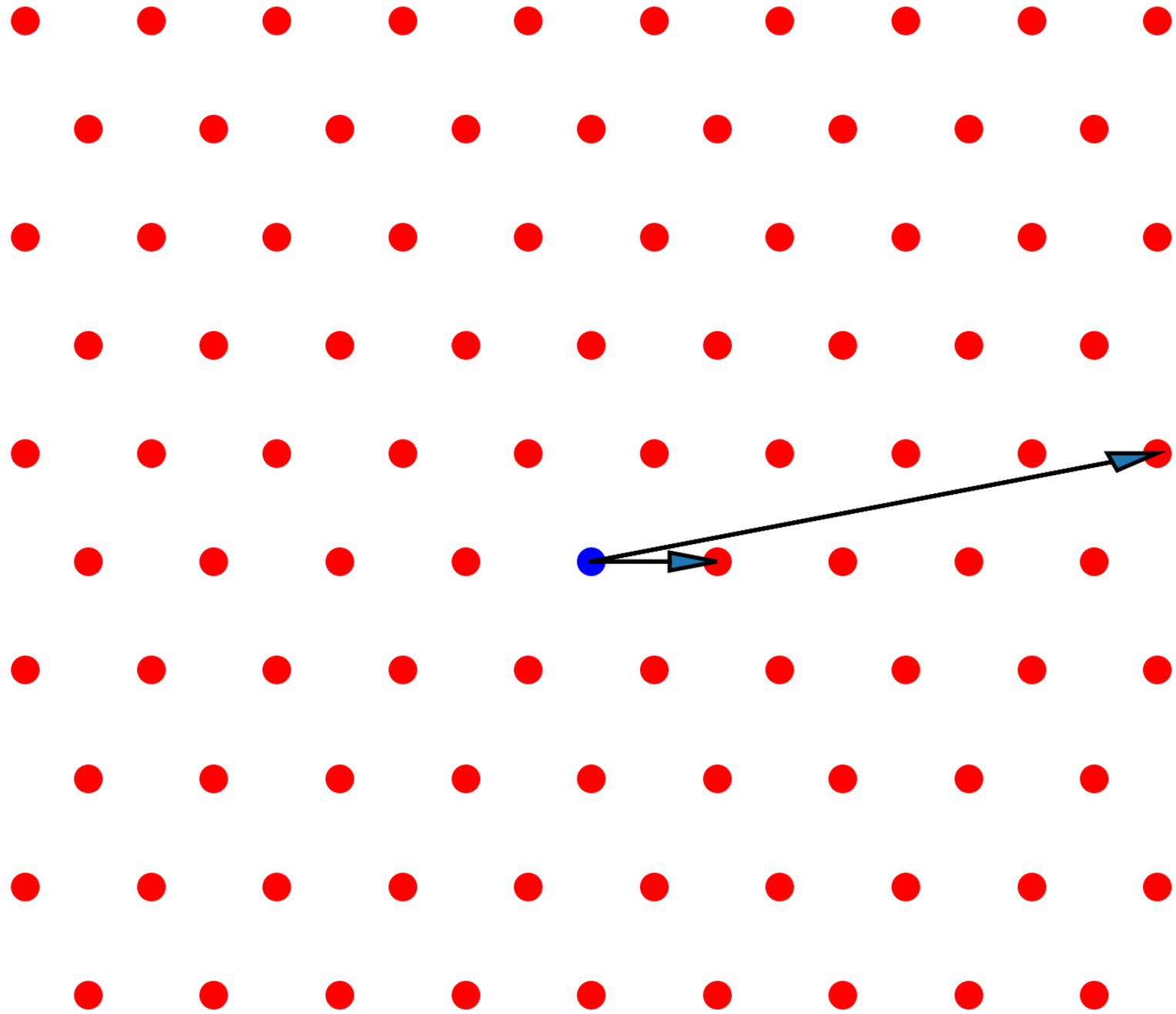


Definition of Lattices

Let $B = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_K\}, \boldsymbol{b}_i \in \mathbb{R}^n,$ **be a set of vectors.**

$$\mathscr{L} = \left\{ \sum_{i=1}^{K} a_i \boldsymbol{b}_i \,|\, a_i \in \mathbb{Z} \right\}$$

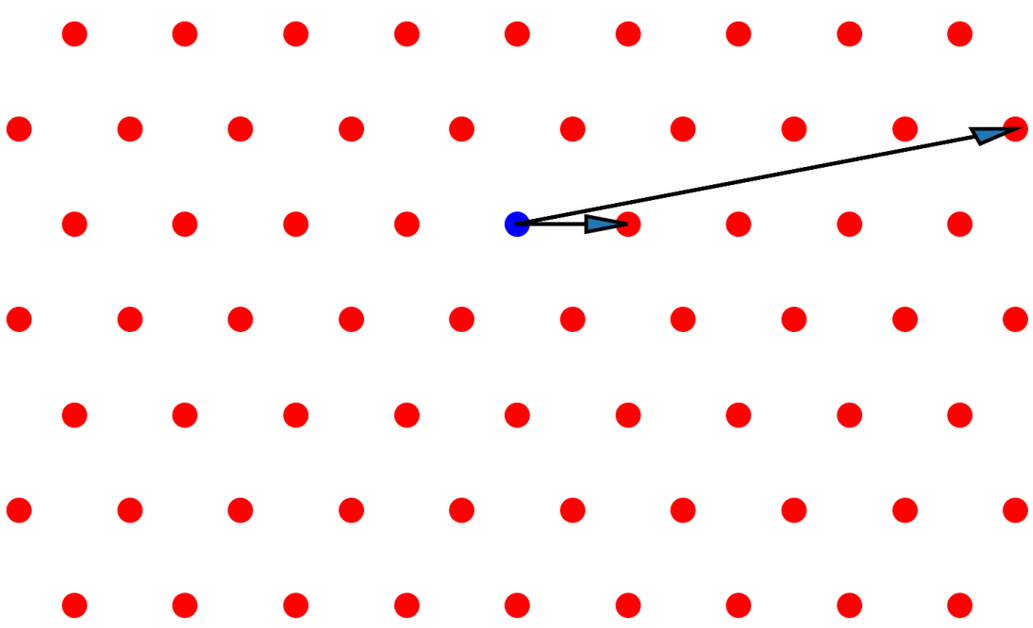
Definition of Lattices

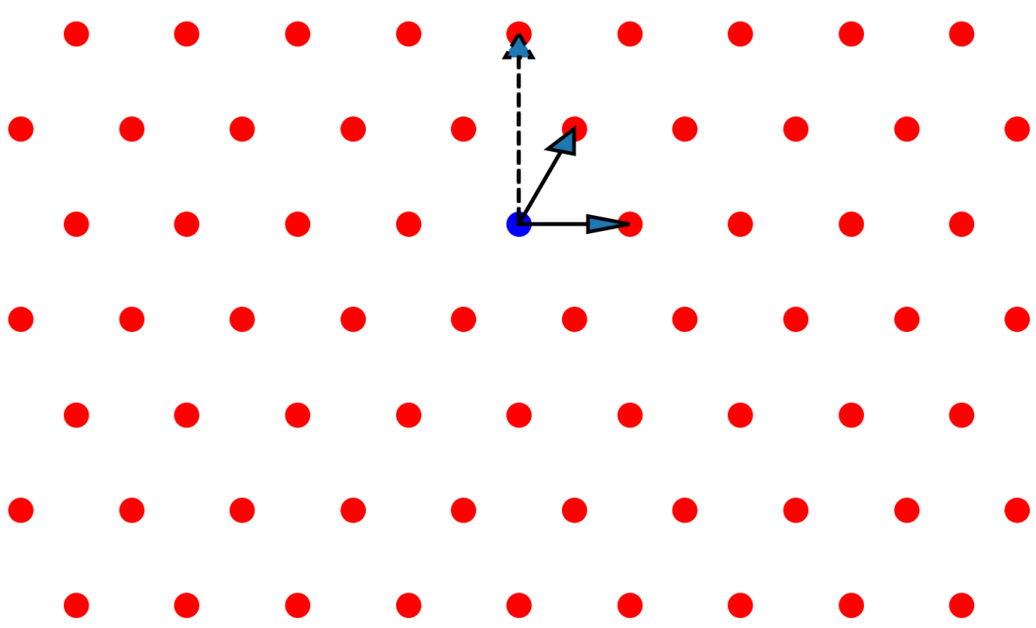


Definition of Lattices

- If **B** is a set of independent vectors, then **B** is called the **Basis of the Lattice.**
- An infinite number of Bases can generate the same Lattice.

Definition of Lattices





Shortest Lattice Vector Problem(SVP)

Problem Statement:
Given a Lattice, find the smallest non-zero vector that belongs to that lattice.

An NP-Hard Problem.

γ-Approximate SVP Problem

Approximation Variant of SVP:

Given a Lattice, find the smallest non-zero vector that has a norm smaller than γ times the shortest vector in the lattice.

γ-Approximate SVP Problem

Find a vector $\boldsymbol{v}$ in the Lattice such that:

$$0 < ||\boldsymbol{v}|| \leq \gamma ||\boldsymbol{\lambda}||$$

Here, $\boldsymbol{\lambda}$ is the smallest vector in the Lattice.

LLL Algorithm

- Full Name: **Lenstra-Lenstra-Lovász Algorithm.**
- The LLL-algorithm gives a polynomial time solution for $2^{n/2}$ approximate SVP.

LLL Algorithm

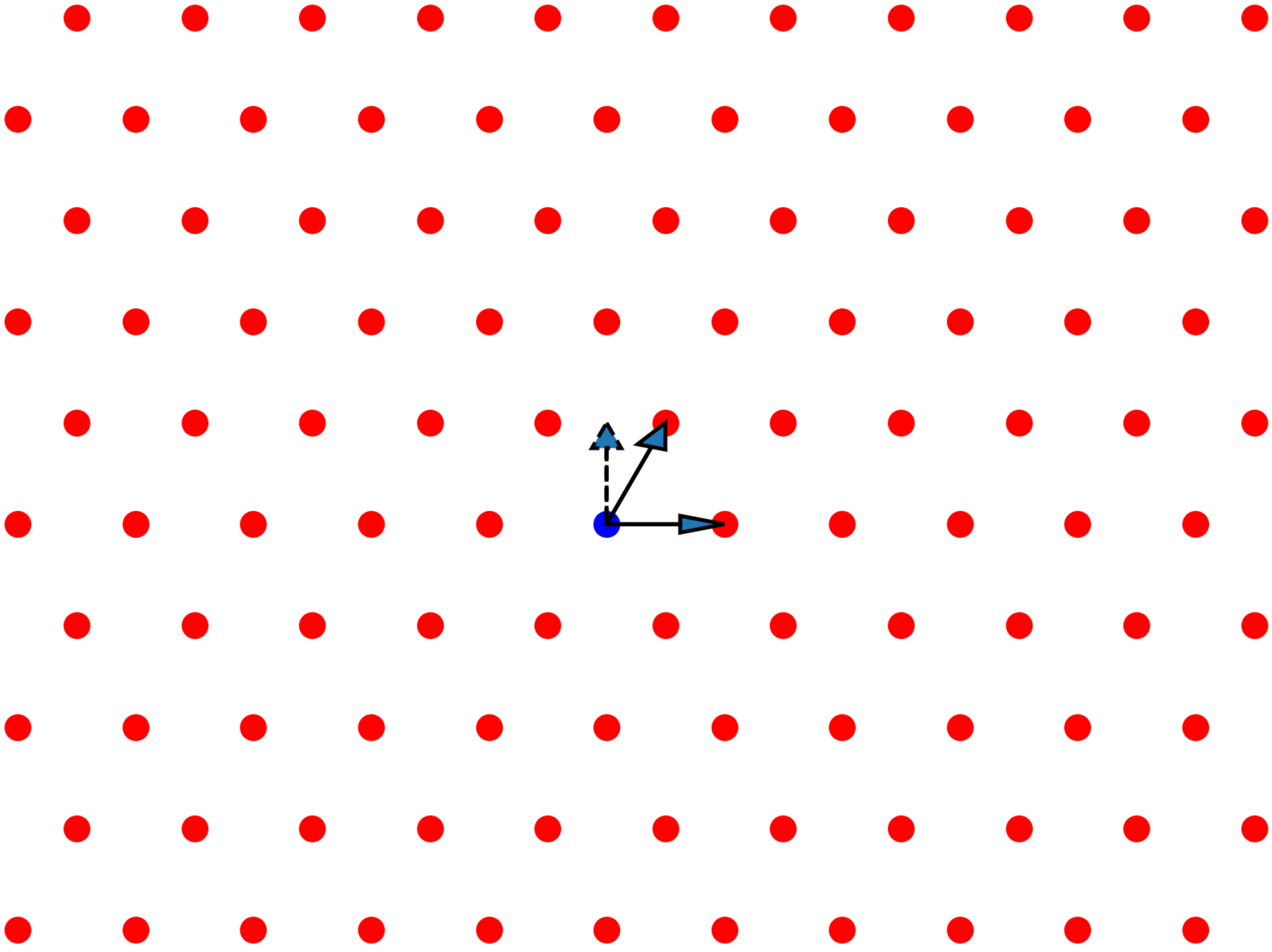
- It takes an input basis **B** that generates a particular Lattice, and gives another basis **B’** as output.

- **B’** satisfies certain conditions (described soon) and generates the same lattice.

LLL-Reduced Basis

- Original Basis: $\boldsymbol{B} = [\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}]$
- GSO Basis: $\boldsymbol{B}^* = [\boldsymbol{b}_0^*, \boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_{n-1}^*]$
- The GSO Basis is defined as:

$$\boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \boldsymbol{b}_j, \text{ where } \mu_{i,j} = \frac{\langle \boldsymbol{b}_i, \boldsymbol{b}_j^* \rangle}{\langle \boldsymbol{b}_j^*, \boldsymbol{b}_j^* \rangle}$$



LLL Reduced Basis

- Reduction Conditions:
  1. $\mu_{i,j} \leq 0.5$
  2. $\delta ||\boldsymbol{b}_{k-1}^*||^2 \leq ||\boldsymbol{b}_k^* + \mu_{k,k-1}^2 \boldsymbol{b}_{k-1}^*||^2$, $\delta \in (0.25,1]$

- Using the first condition, the second condition can be rewritten as:

$$||\boldsymbol{b}_k^*||^2 \geq \left(\delta - \frac{1}{4}\right) ||\boldsymbol{b}_{k-1}^*||^2$$

- $\boldsymbol{b_{k-1}}$ is not much shorter than $\boldsymbol{b_k}$.

LLL-Reduced Basis

- It can be shown that:

$$||\boldsymbol{b}_0|| \leq \left(\frac{2}{\sqrt{4\delta - 1}}\right)^{n-1} ||\boldsymbol{\lambda}||$$

$$\gamma = \left(\frac{2}{\sqrt{4\delta - 1}}\right)^{n-1}$$

LLL Algorithm

- Higher the value of δ, smaller is the resultant smallest vector.
- However, time taken by LLL Algorithm increases as δ increases.

$$\textbf{Time} \propto \frac{1}{\log(\frac{1}{\delta})}$$

Suggested Improvements to LLL Algorithm

‘Dis-balancing’ a LLL-Reduced Basis

- The Basis returned by a run of the LLL algorithm, when once again sent through another LLL run, doesn't change at all.

- This is because inputting a LLL-reduced basis to the LLL Algorithm doesn't do anything.

‘Dis-balancing’ a LLL-Reduced Basis

- If we add some changes to the output basis, and input this changed basis for another run of the LLL Algorithm, this new run gives a new basis which has an even shorter smallest vector present in it.

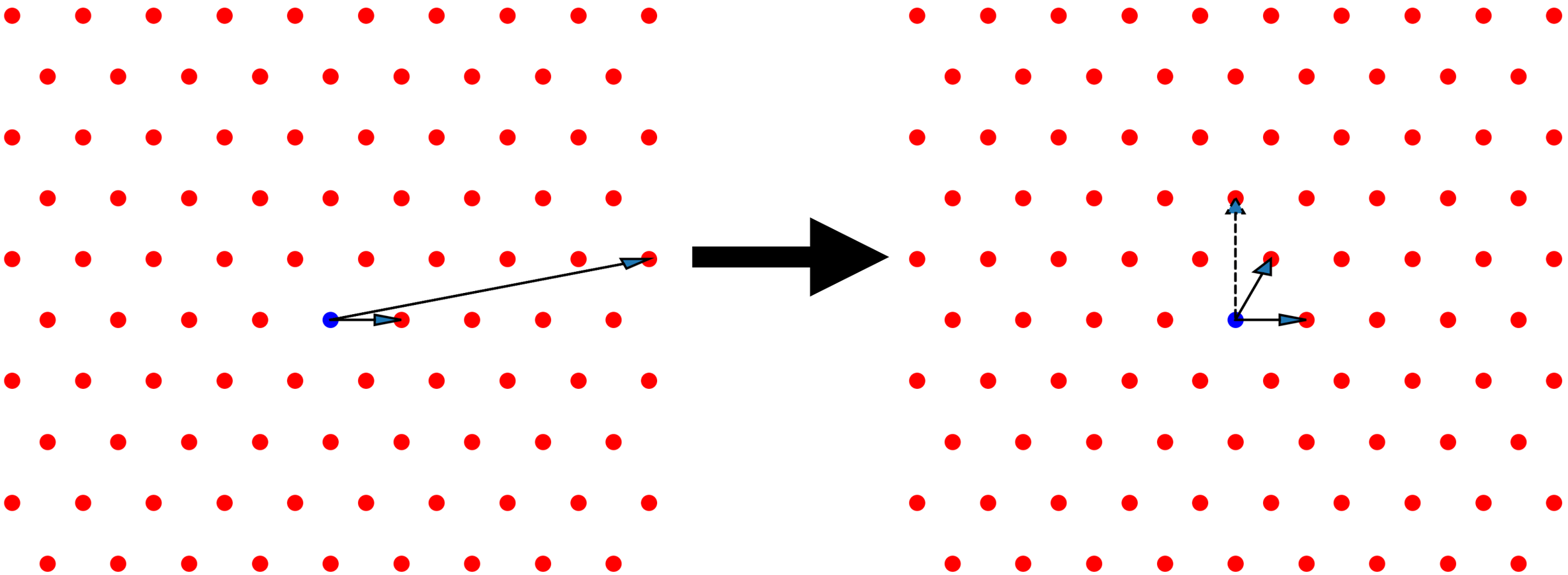
- In the proceeding slides, we describe the best dis-balancing procedure we found in our testings.

Maximum Separation Plane (MSP)

- $B = \{b_0, b_1, b_2, \ldots, b_{n-1}\}$

- Let b_{n-1} be fixed. Now, we can change the remaining vectors in such a way that the Lattice doesn't change, but b_{n-1} forms a smaller angle with the normal to the sub-plane formed by the remaining vectors.

Maximum Separation Plane



Maximum Separation Plane

With $\boldsymbol{b_{n-1}}$ fixed, the way we find a plane at a higher separation is as follows: For all i = 0, 1, . . . , n – 2, do the following:

- We have a fixed vector $\boldsymbol{b_{n-1}}$ and a currently chosen vector, say $\boldsymbol{b_i}$. The remaining (n–2) vectors form a vector space *S*.

- Find out the perpendicular component vectors of $\boldsymbol{b_{n-1}}$ and $\boldsymbol{b_i}$ on S, and call these $\boldsymbol{g_{n-1}}$ and $\boldsymbol{g_i}$ respectively.

Maximum Separation Plane

With $\boldsymbol{b_{n-1}}$ fixed, the way we find a plane at a higher separation is as follows: For all i = 0, 1, . . . , n – 2, do the following:

- We have a fixed vector $\boldsymbol{b_{n-1}}$ and a currently chosen vector, say $\boldsymbol{b_i}$. The remaining (n–2) vectors form a vector space *S*.

- Find out the perpendicular component vectors of $\boldsymbol{b_{n-1}}$ and $\boldsymbol{b_i}$ on S, and call these $\boldsymbol{g_{n-1}}$ and $\boldsymbol{g_i}$ respectively.

- To get the MSP, we use the following update equation:

$$\boldsymbol{b}_i = \boldsymbol{b}_i - \left\lfloor \frac{\langle \boldsymbol{g}_{n-1}, \boldsymbol{g}_i \rangle}{\langle \boldsymbol{g}_{n-1}, \boldsymbol{g}_{n-1} \rangle} \right\rceil \times \boldsymbol{b}_{n-1}$$

Maximum Separation Plane

- The update equation is:

$$\boldsymbol{b}_i = \boldsymbol{b}_i - \left\lfloor \frac{\langle \boldsymbol{g}_{n-1}, \boldsymbol{g}_i \rangle}{\langle \boldsymbol{g}_{n-1}, \boldsymbol{g}_{n-1} \rangle} \right\rceil \times \boldsymbol{b}_{n-1}$$

- What's happening here is the following: we find the projection-ratio of $\boldsymbol{g_i}$ on $\boldsymbol{g_{n-1}}$, round it off to the nearest integer to get an integer β, multiply the rounded off value to $\boldsymbol{b_{n-1}}$ to get the vector $\beta\boldsymbol{b_{n-1}}$, and subtract this from $\boldsymbol{b_i}$.

Using Max-Separation Plane to 'Open Up' the Lattice

- First, we fixed one vector and changed the sub-plane formed by the remaining *n-1* vectors.

- Now, consider the sub-lattice formed by these *n-1* vectors.

- Fix any one vector, and change the remaining *n-2* vectors to get the MSP for the fixed vector in this sub-lattice.

Using Max-Separation Plane to ‘Open Up’ the Lattice

- Keep on going deeper and deeper into the sub-lattices.

- We call this routine the `REDUCE` function.

The REDUCE Function

Algorithm 2 PERP-REDUCE

Input: Lattice Basis $\mathbf{B} = [\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{n-1}]$

1: **for** i `in` $0 \ldots (n-2)$ **do**
2: $S = \text{PLANE}(\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_{i+1}, \ldots \mathbf{b}_{n-2})$ `//Sub-plane formed by the input vectors`
3: $\mathbf{g}_i = \text{PERP-COMP}(\mathbf{b}_i, S)$
4: $\mathbf{g}_{n-1} = \text{PERP-COMP}(\mathbf{b}_{n-1}, S)$
5: $\eta = \text{PROJECTION}(\mathbf{g}_i, \mathbf{g}_{n-1})/\text{NORM}(\mathbf{g}_{n-1})$
6: $\mathbf{b}_i = \mathbf{b}_i - \eta \times \mathbf{b}_{n-1}$
7: **end for**

Algorithm 3 REDUCE function

Input: Lattice Basis $\mathbf{B} = [\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{n-1}]$

1: $m \leftarrow n$
2: **while** $m \geq 0$ **do**
3: $\mathbf{B}' \leftarrow \mathbf{B}[0 : m]$
4: $\mathbf{B}' \leftarrow \text{PERP-REDUCE}(\mathbf{B}')$. `//See Algorithm 2`
5: $m \leftarrow m - 1$.
6: **end while**

The Full Pseudo-Code

Algorithm 1 Modified LLL

Input: Lattice Basis $\mathbf{B} = [\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{n-1}]$

1: $\mathbf{B}' \leftarrow \mathrm{LLL}(\mathbf{B})$
2: **while NOT** $\textsc{Converged}(\mathbf{B}', \mathbf{B})$ **do**
3: $\quad \mathbf{B} \leftarrow \mathbf{B}'$
4: $\quad \mathbf{B}' \leftarrow \textsc{reduce}(\mathbf{B})$
5: $\quad \mathbf{B}' \leftarrow \mathrm{LLL}(\mathbf{B}')$.
6: **end while**

Convergence

- When no better results are seen after an iteration, we break from the while loop.

- Need to check changes in the lattice basis after each iteration without adding any major overheads.

- One simple way: check the sum of norms of all vectors and check the norm of the smallest vectors.

- If both these quantities don't change, break from the loop.

Experiments

- **Original LLL vs Modified LLL with same δ = 0.251**: Both the Algorithms run till termination.

- **Original LLL with δ = 0.9 vs Modified LLL with δ = 0.251**: The former runs till termination, while the latter either stops when it finds the smallest vector found by the former, or if it converges before finding that vector.

Expected Results in Both the Experiments

- Given a Lattice Basis, we compare the performance of original and modified conditions in both the experiments.

- In the first one, the primary concern is that the modified LLL should show improvement in the resultant smallest vector.

- In the latter, the concern is to get the same smallest vector with the modified version taking lesser time.

Basis used for Testing

$$\begin{bmatrix} x_0 & 0 & 0 & 0 & \dots & 0 \\ x_1 & 1 & 0 & 0 & \dots & 0 \\ x_2 & 0 & 1 & 0 & \dots & 0 \\ x_3 & 0 & 0 & 1 & \dots & 0 \\ & \dots\dots\dots & & \ddots & & \\ x_{n-1} & 0 & 0 & 0 & \dots & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_0^T \\ \mathbf{b}_1^T \\ \mathbf{b}_2^T \\ \mathbf{b}_3^T \\ \cdot \\ \cdot \\ \mathbf{b}_{n-1}^T \end{bmatrix}$$

- For all *i*, x_i has approximately 240 digits.

Basis used for Testing

- These tests were run on 500 different Lattices.
- Of these 500 Lattices, there were 100 Lattices each of dimensions 20, 22, 24, 26, & 28.
- All these Lattices had basis of the form mentioned above.

Results of Experiment 1

Dimensions	Average Improvement	Average Speed-down
20	17.602	0.300
22	30.100	0.262
24	50.245	0.222
26	73.363	0.188
28	99.461	0.172

Results of Experiment 2

Dimensions	Success Percentage	Average Speedup	Average Speedup in Success Cases	Average Ratio of Shortest Norms in Failures
20	90%	2.590	2.620	1.065
22	80%	2.530	2.545	1.263
24	74%	2.399	2.392	2.200
26	59%	2.233	2.247	2.059
28	46%	2.099	1.947	9.431

Inferences

- The modifications greatly increase the power of lower δ-valued LLL algorithm.

- While increasing the power, the time taken is less as compared to higher δ-valued pure LLL algorithm.

- We also observe that as the number of Basis vectors increase (that is, as the dimensions of the Lattice increase), the modified LLL Algorithm performs even more better than the original LLL algorithm for the same δ value.

Thank You.