

Solving All Lattice Problems in Deterministic Single Exponential Time

Daniele Micciancio
(Joint work with P. Voulgaris, STOC 2010)

UCSD

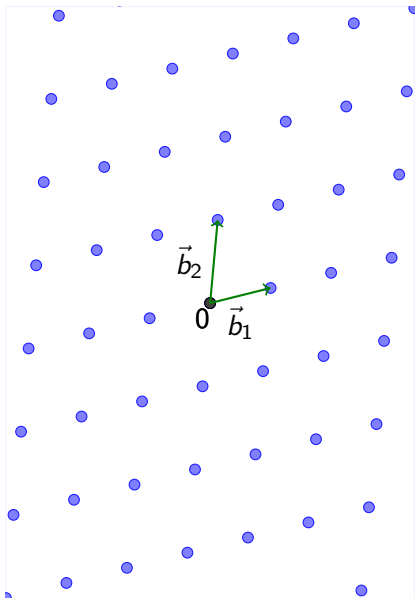
March 22, 2011

- Traditional area of mathematics
 - Bridge between number theory and geometry
 - Studied by Lagrange, Gauss, ..., Minkowski, ...
- Key to many algorithmic applications
 - Cryptanalysis, Coding Theory, Integer Programming
- Foundation of Lattice based Cryptography
 - Exponentially hard to break, even by quantum adversary
 - Asymptotically fast and easily parallelizable cryptographic functions
 - Secure based on conjectured hardness of *worst-case* problems
 - Extremely versatile: CPA/CCA encryption, digital signature, ... group and ring signatures, threshold cryptography, IBE, ..., HIBE, ..., FHE, ...

- 1 Introduction Lattices
 - Lattice Problems
 - Algorithmic Techniques
- 2 New Algorithm
 - Overview
 - Voronoi Cell
 - CVPP Algorithm
- 3 Final Remarks and Open Problems

- 1 Introduction Lattices
 - Lattice Problems
 - Algorithmic Techniques
- 2 New Algorithm
 - Overview
 - Voronoi Cell
 - CVPP Algorithm
- 3 Final Remarks and Open Problems

Point Lattices

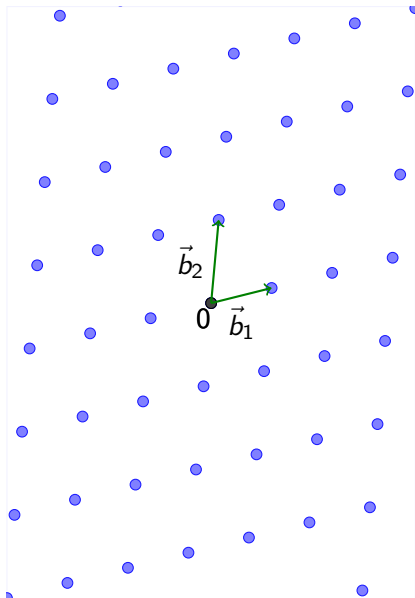


A lattice is the set of all integer linear combinations of (linearly independent) basis vectors

$$\mathbf{B} = \{\vec{b}_1, \dots, \vec{b}_n\} \subset \mathbb{R}^n:$$

$$\Lambda = \sum_{i=1}^n \vec{b}_i \cdot \mathbb{Z}$$

Point Lattices

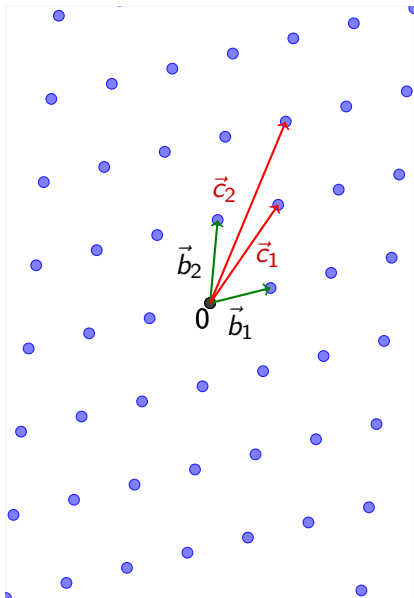


A lattice is the set of all integer linear combinations of (linearly independent) basis vectors

$$\mathbf{B} = \{\vec{b}_1, \dots, \vec{b}_n\} \subset \mathbb{R}^n:$$

$$\Lambda = \sum_{i=1}^n \vec{b}_i \cdot \mathbb{Z} = \{\mathbf{B}\vec{x} : \vec{x} \in \mathbb{Z}^n\}$$

Point Lattices



A lattice is the set of all integer linear combinations of (linearly independent) basis vectors

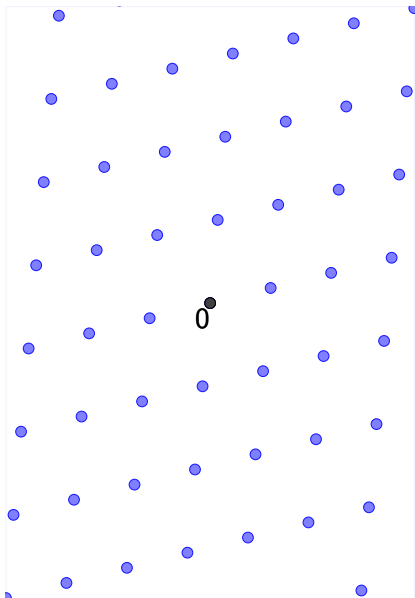
$$\mathbf{B} = \{\vec{b}_1, \dots, \vec{b}_n\} \subset \mathbb{R}^n:$$

$$\Lambda = \sum_{i=1}^n \vec{b}_i \cdot \mathbb{Z} = \{\mathbf{B}\vec{x} : \vec{x} \in \mathbb{Z}^n\}$$

The same lattice has many bases

$$\Lambda = \sum_{i=1}^n \vec{c}_i \cdot \mathbb{Z}$$

Point Lattices



A lattice is the set of all integer linear combinations of (linearly independent) basis vectors

$$\mathbf{B} = \{\vec{b}_1, \dots, \vec{b}_n\} \subset \mathbb{R}^n:$$

$$\Lambda = \sum_{i=1}^n \vec{b}_i \cdot \mathbb{Z} = \{\mathbf{B}\vec{x} : \vec{x} \in \mathbb{Z}^n\}$$

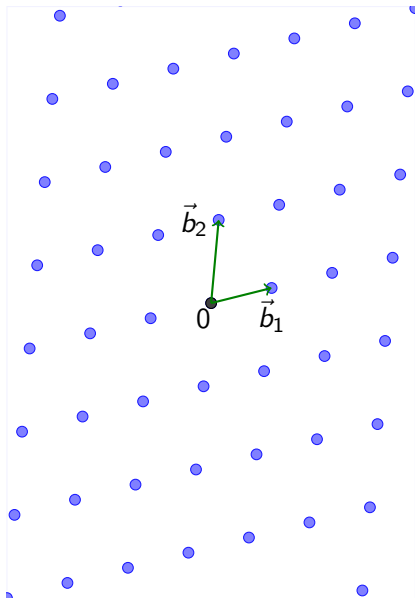
The same lattice has many bases

$$\Lambda = \sum_{i=1}^n \vec{c}_i \cdot \mathbb{Z}$$

Definition (Lattice)

Discrete additive subgroup of $\mathbb{R}^n$

Shortest Vector Problem (SVP)

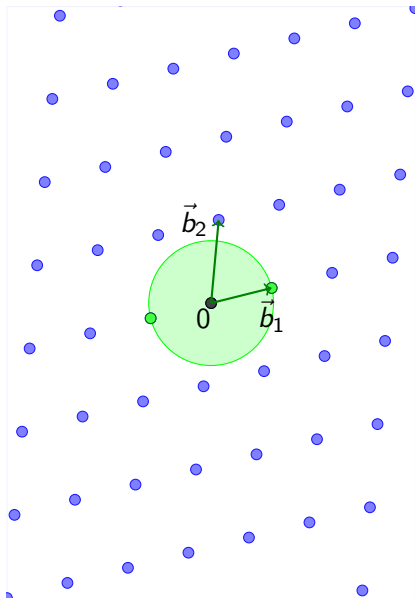


Definition (SVP)

Given a lattice $\mathcal{L}(\mathbf{B})$, find a (nonzero) lattice vector $\mathbf{B}\vec{x}$ (with $\vec{x} \in \mathbb{Z}^k$) of minimal length $\|\mathbf{B}\vec{x}\|$

- Input: A lattice basis $\mathbf{B}$

Shortest Vector Problem (SVP)

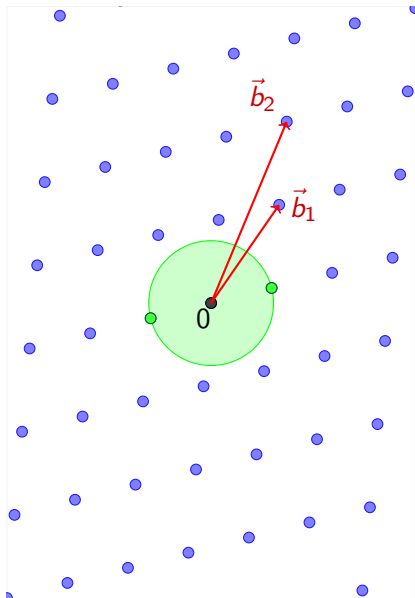


Definition (SVP)

Given a lattice $\mathcal{L}(\mathbf{B})$, find a (nonzero) lattice vector $\mathbf{B}\vec{x}$ (with $\vec{x} \in \mathbb{Z}^k$) of minimal length $\|\mathbf{B}\vec{x}\|$

- Input: A lattice basis $\mathbf{B}$
- Output: A shortest nonzero vector $\vec{s} \in \Lambda$

Shortest Vector Problem (SVP)

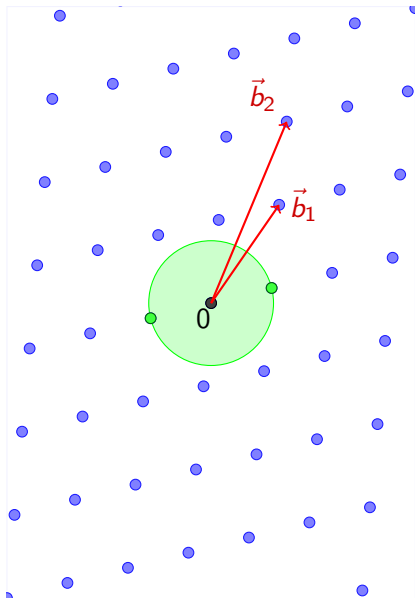


Definition (SVP)

Given a lattice $\mathcal{L}(\mathbf{B})$, find a (nonzero) lattice vector $\mathbf{B}\vec{x}$ (with $\vec{x} \in \mathbb{Z}^k$) of minimal length $\|\mathbf{B}\vec{x}\|$

- Input: A lattice basis $\mathbf{B}$
- Output: A shortest nonzero vector $\vec{s} \in \Lambda$
- The problem is hard when dimension n is high and basis is skewed

Shortest Vector Problem (SVP)

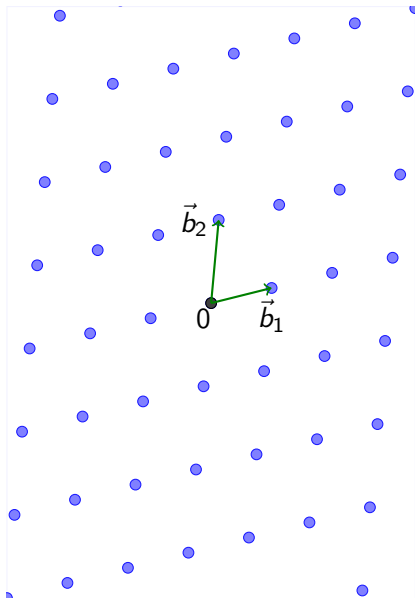


Definition (SVP)

Given a lattice $\mathcal{L}(\mathbf{B})$, find a (nonzero) lattice vector $\mathbf{B}\vec{x}$ (with $\vec{x} \in \mathbb{Z}^k$) of minimal length $\|\mathbf{B}\vec{x}\|$

- Input: A lattice basis $\mathbf{B}$
- Output: A shortest nonzero vector $\vec{s} \in \Lambda$
- The problem is hard when dimension n is high and basis is skewed
- Shortest vector can be much shorter than basis vectors

Shortest Independent Vectors Problem (SIVP)

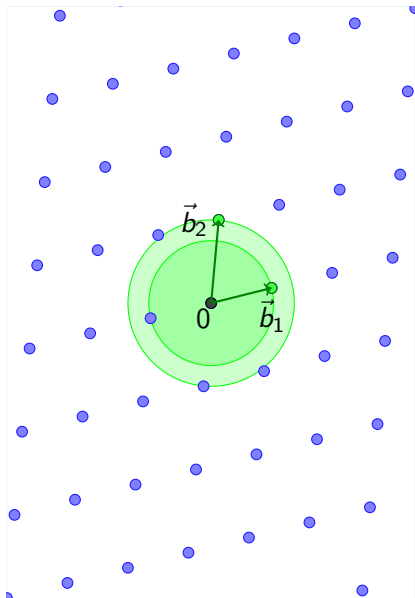


Definition (SIVP)

Given a lattice $\mathcal{L}(\mathbf{B})$, find n linearly independent lattice vectors $\vec{s}_1, \dots, \vec{s}_n$ of minimal length $\max_i \|\vec{s}_i\|$

- Input: A lattice basis $\mathbf{B}$

Shortest Independent Vectors Problem (SIVP)

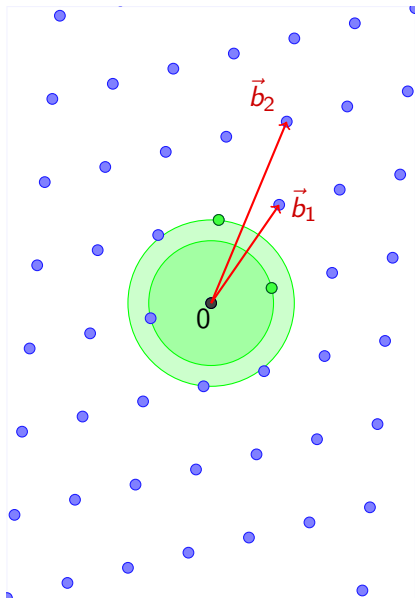


Definition (SIVP)

Given a lattice $\mathcal{L}(\mathbf{B})$, find n linearly independent lattice vectors $\vec{s}_1, \dots, \vec{s}_n$ of minimal length $\max_i \|\vec{s}_i\|$

- Input: A lattice basis $\mathbf{B}$
- Output: n shortest linearly independent lattice vectors $\vec{s}_1, \dots, \vec{s}_n \in \Lambda$

Shortest Independent Vectors Problem (SIVP)

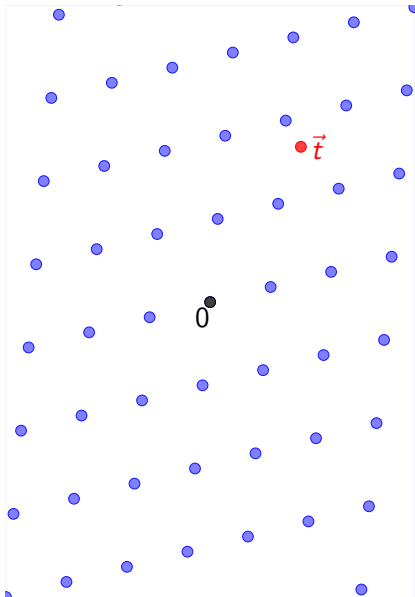


Definition (SIVP)

Given a lattice $\mathcal{L}(\mathbf{B})$, find n linearly independent lattice vectors $\vec{s}_1, \dots, \vec{s}_n$ of minimal length $\max_i \|\vec{s}_i\|$

- Input: A lattice basis $\mathbf{B}$
- Output: n shortest linearly independent lattice vectors $\vec{s}_1, \dots, \vec{s}_n \in \Lambda$
- The problem is hard when dimension n is high and basis is skewed

Closest Vector Point (CVP)



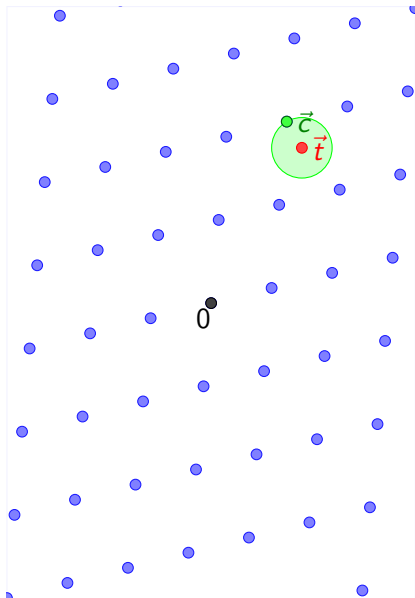
Inhomogeneous version of SVP

Definition (CVP)

Given a lattice $\mathcal{L}(\mathbf{B})$ and a target point $\vec{t}$, find a lattice vector $\mathbf{B}\vec{x}$ which minimizes the distance $\|\mathbf{B}\vec{x} - \vec{t}\|$

- Input: A lattice $\Lambda(\mathbf{B})$, and a target vector $\vec{t}$

Closest Vector Point (CVP)



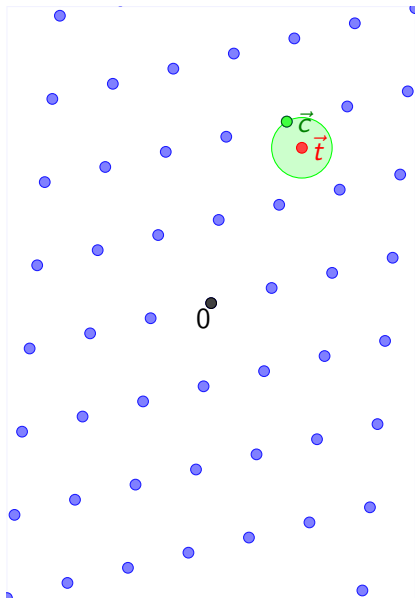
Inhomogeneous version of SVP

Definition (CVP)

Given a lattice $\mathcal{L}(\mathbf{B})$ and a target point $\vec{t}$, find a lattice vector $\mathbf{B}\vec{x}$ which minimizes the distance $\|\mathbf{B}\vec{x} - \vec{t}\|$

- Input: A lattice $\Lambda(\mathbf{B})$, and a target vector $\vec{t}$
- Output: A closest lattice point $\vec{c} \in \Lambda$

Closest Vector Point (CVP)



Inhomogeneous version of SVP

Definition (CVP)

Given a lattice $\mathcal{L}(\mathbf{B})$ and a target point $\vec{t}$, find a lattice vector $\mathbf{B}\vec{x}$ which minimizes the distance $\|\mathbf{B}\vec{x} - \vec{t}\|$

- Input: A lattice $\Lambda(\mathbf{B})$, and a target vector $\vec{t}$
- Output: A closest lattice point $\vec{c} \in \Lambda$
- NP-hard [vEB'81], even for fixed lattice [M'01]

Approximating SVP, SIVP, CVP

- Best known polynomial time algorithm only find poor $(2^{\omega(n/\log n)})$ approximations
- Lattice based cryptography is based on the conjectured hardness of finding good $(n^{O(1)})$ approximate solutions

Approximating SVP, SIVP, CVP

- Best known polynomial time algorithm only find poor $(2^{\omega(n/\log n)})$ approximations
- Lattice based cryptography is based on the conjectured hardness of finding good $(n^{O(1)})$ approximate solutions

Solving SVP, SIVP, CVP **exactly**

- **NP-hard**: no subexponential time solution is expected
- Best known exact algorithms run in exponential time $2^{\Omega(n)}$

Approximating SVP, SIVP, CVP

- Best known polynomial time algorithm only find poor ($2^{\omega(n/\log n)}$) approximations
- Lattice based cryptography is based on the conjectured hardness of finding good ($n^{O(1)}$) approximate solutions

Solving SVP, SIVP, CVP **exactly**

- **NP-hard**: no subexponential time solution is expected
- Best known exact algorithms run in exponential time $2^{\Omega(n)}$

Applications of exact SVP, SIVP, CVP

- Some applications involve low dimensional lattices
- Efficient approximation algorithms are based on exact solution of small dimensional subproblems

How fast we we solve SVP, SIVP, CVP? (E.g., $2^{n/2} < 2^{100 \cdot n} < n^n$)

Complexity of SVP, SIVP, CVP

- Efficient (dimension preserving) reductions
 - $\text{SVP}, \text{SIVP} \leq \text{CVP}$ [GMSS'99, M'08]
- Fastest previous algorithm
 - SVP, SIVP, CVP : [Kannan'87] runs in $n^{O(n)}$ time
 - **SVP**: [AKS'01] runs in randomized $2^{O(n)}$ time and space
 - Algorithms work in any ℓ_p norm [BN'07]

Complexity of SVP, SIVP, CVP

- Efficient (dimension preserving) reductions
 - $\text{SVP}, \text{SIVP} \leq \text{CVP}$ [GMSS'99, M'08]
- Fastest previous algorithm
 - $\text{SVP}, \text{SIVP}, \text{CVP}$: [Kannan'87] runs in $n^{O(n)}$ time
 - **SVP**: [AKS'01] runs in randomized $2^{O(n)}$ time and space
 - Algorithms work in any ℓ_p norm [BN'07]
- Questions
 - Can **CVP, SIVP** also be solved in $2^{c \cdot n}$ time?
 - What is the smallest constant c ? [NV'09, MP'10, PS'10]:
 $c < 2.5$ for SVP in ℓ_2 .
 - Is **randomization** and **exponential space** useful/necessary?

Complexity of SVP, SIVP, CVP

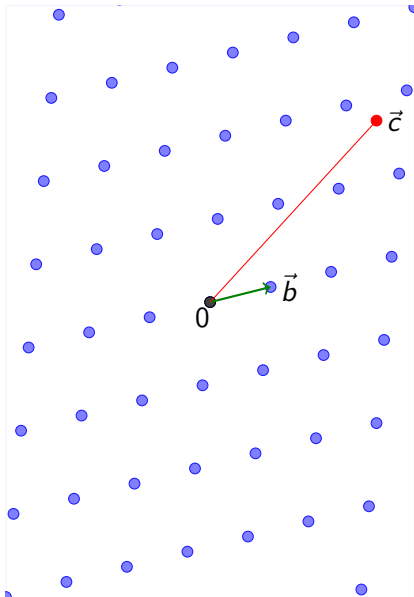
- Efficient (dimension preserving) reductions
 - $\text{SVP}, \text{SIVP} \leq \text{CVP}$ [GMSS'99, M'08]
- Fastest previous algorithm
 - $\text{SVP}, \text{SIVP}, \text{CVP}$: [Kannan'87] runs in $n^{O(n)}$ time
 - SVP : [AKS'01] runs in randomized $2^{O(n)}$ time and space
 - Algorithms work in any ℓ_p norm [BN'07]
- Questions
 - Can CVP, SIVP also be solved in $2^{c \cdot n}$ time? Yes! (for ℓ_2)
 - What is the smallest constant c ? [NV'09, MP'10, PS'10]:
 $c < 2.5$ for SVP in ℓ_2 . $c \leq 2$ for $\text{SVP}, \text{SIVP}, \text{CVP}$!
 - Is randomization and exponential space useful/necessary?
Randomization is not!

Complexity of SVP, SIVP, CVP

- Efficient (dimension preserving) reductions
 - $\text{SVP}, \text{SIVP} \leq \text{CVP}$ [GMSS'99, M'08]
- Fastest previous algorithm
 - $\text{SVP}, \text{SIVP}, \text{CVP}$, **IP**: [Kannan'87] runs in $n^{O(n)}$ time
 - SVP : [AKS'01] runs in randomized $2^{O(n)}$ time and space
 - Algorithms work in any ℓ_p norm [BN'07]
- Questions
 - Can CVP , SIVP also be solved in $2^{c \cdot n}$ time? **Yes!** (for ℓ_2)
 - What is the smallest constant **c** ? [NV'09, MP'10, PS'10]:
 $c < 2.5$ for SVP in ℓ_2 . **$c \leq 2$ for $\text{SVP}, \text{SIVP}, \text{CVP}$!**
 - Is randomization and **exponential space** useful/necessary?
Randomization is not!
 - What about **other norms** and **Integer Programming (IP)**?

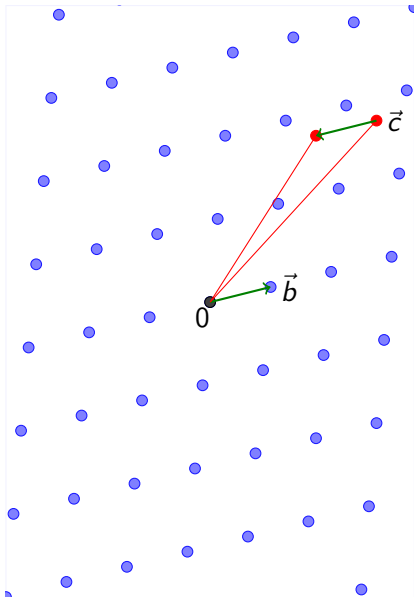
- 1 Introduction Lattices
 - Lattice Problems
 - Algorithmic Techniques
- 2 New Algorithm
 - Overview
 - Voronoi Cell
 - CVPP Algorithm
- 3 Final Remarks and Open Problems

Size Reduction



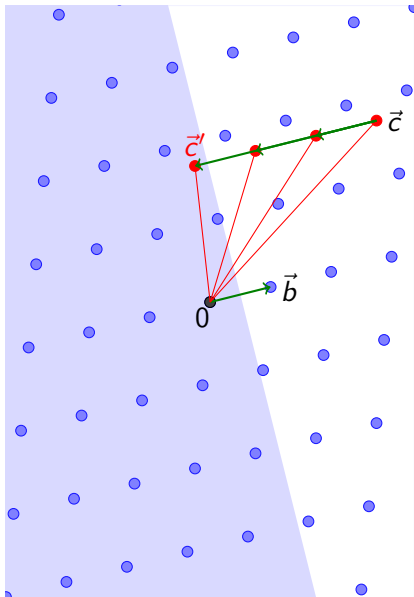
- $\vec{b}$: (short) lattice vector
- $\vec{c}$: arbitrary point

Size Reduction



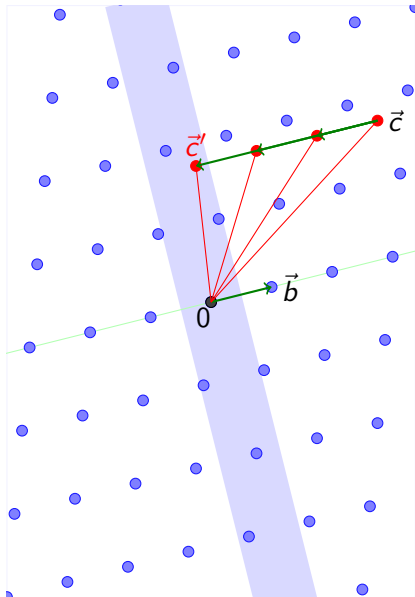
- $\vec{b}$: (short) lattice vector
- $\vec{c}$: arbitrary point
- Can make $\vec{c}$ shorter by subtracting $\vec{b}$ from it

Size Reduction



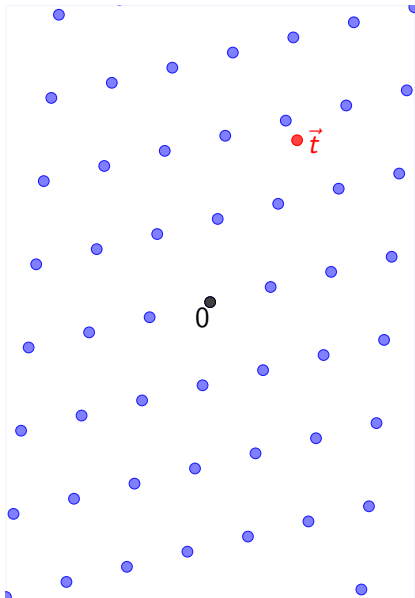
- $\vec{b}$: (short) lattice vector
- $\vec{c}$: arbitrary point
- Can make $\vec{c}$ shorter by subtracting $\vec{b}$ from it
- Repeat until $\vec{c}$ closer to $\vec{0}$ than to $\vec{b}$

Size Reduction



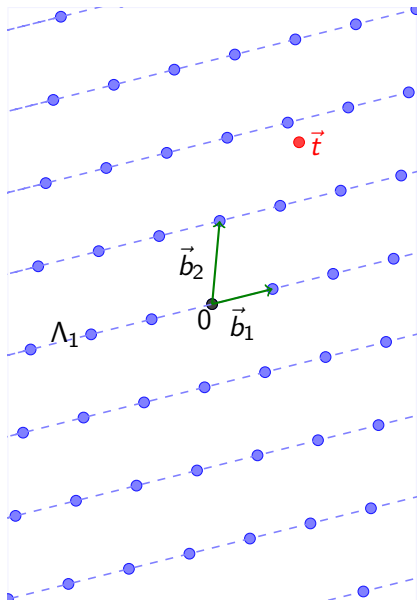
- $\vec{b}$: (short) lattice vector
- $\vec{c}$: arbitrary point
- Can make $\vec{c}$ shorter by subtracting $\vec{b}$ from it
- Repeat until $\vec{c}$ closer to $\vec{0}$ than to $\vec{b}$ or $-\vec{b}$
- Remarks
 - $\vec{c} - \vec{c}' \in \Lambda$
 - Key step in [LLL'82] basis reduction algorithm
 - Technique is used in most other lattice algorithms

Rank reduction



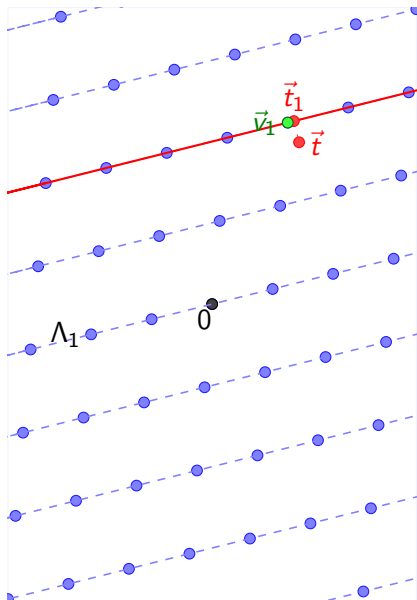
- Goal: Solve $CVP(\Lambda_n, \vec{t})$

Rank reduction



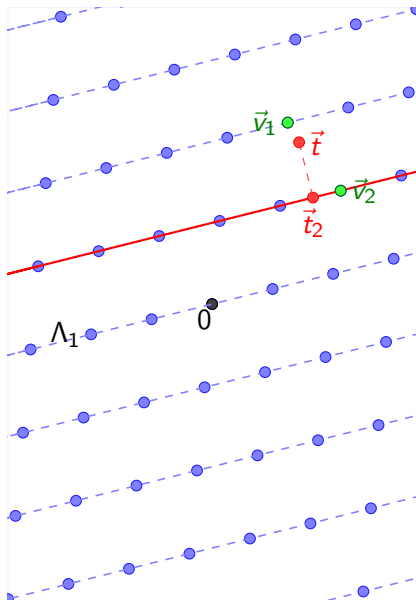
- Goal: Solve $CVP(\Lambda_n, \vec{t})$
- Partition Λ_n into layers of the form: $\Lambda_{n-1} + c\vec{b}_n$, $c = 2, 1, 3, 0, \dots$

Rank reduction



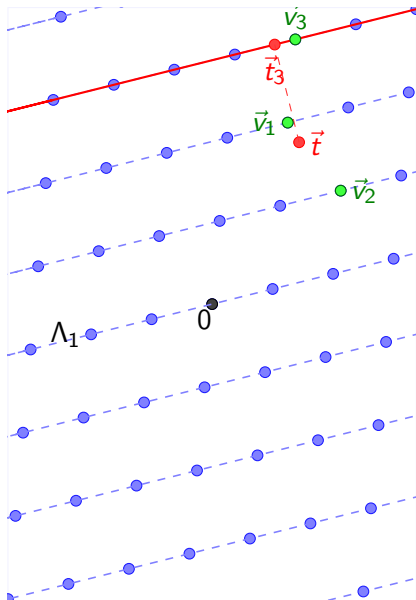
- Goal: Solve $CVP(\Lambda_n, \vec{t})$
- Partition Λ_n into **layers** of the form: $\Lambda_{n-1} + c\vec{b}_n$, $c = 2, 1, 3, 0, \dots$
- Find lattice point $\vec{v}_i$ in each layer closest to (the projection of) $\vec{t}$

Rank reduction



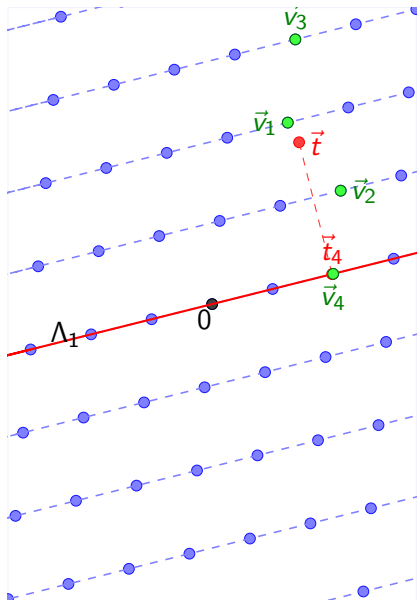
- Goal: Solve $CVP(\Lambda_n, \vec{t})$
- Partition Λ_n into **layers** of the form: $\Lambda_{n-1} + c\vec{b}_n$, $c = 2, 1, 3, 0, \dots$
- Find lattice point $\vec{v}_i$ in each layer closest to (the projection of) $\vec{t}$

Rank reduction



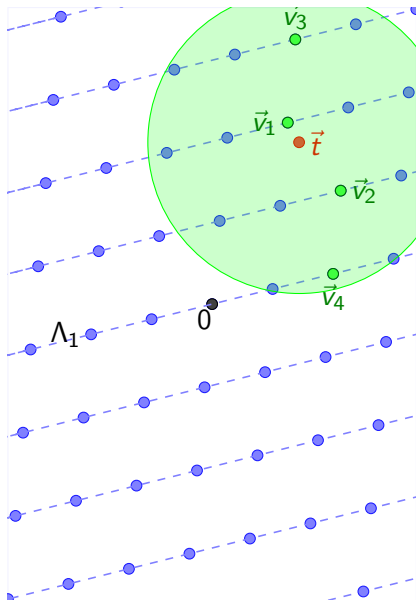
- Goal: Solve $CVP(\Lambda_n, \vec{t})$
- Partition Λ_n into **layers** of the form: $\Lambda_{n-1} + c\vec{b}_n$, $c = 2, 1, 3, 0, \dots$
- Find lattice point $\vec{v}_i$ in each layer closest to (the projection of) $\vec{t}$

Rank reduction



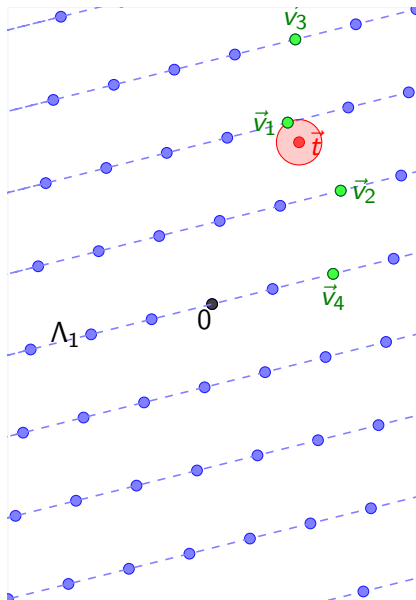
- Goal: Solve $CVP(\Lambda_n, \vec{t})$
- Partition Λ_n into **layers** of the form: $\Lambda_{n-1} + c\vec{b}_n$, $c = 2, 1, 3, 0, \dots$
- Find lattice point $\vec{v}_i$ in each layer closest to (the projection of) $\vec{t}$

Rank reduction



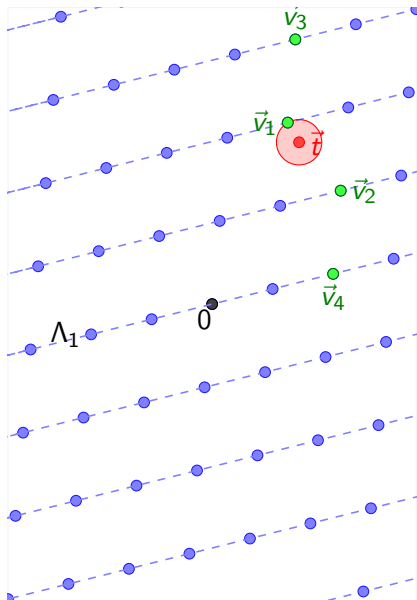
- Goal: Solve $CVP(\Lambda_n, \vec{t})$
- Partition Λ_n into **layers** of the form: $\Lambda_{n-1} + c\vec{b}_n$, $c = 2, 1, 3, 0, \dots$
- Find lattice point $\vec{v}_i$ in each layer closest to (the projection of) $\vec{t}$
- Only need to consider nearby layers
 - Dual LLL: 2^n layers
 - Dual SVP: n layers

Rank reduction



- Goal: Solve $CVP(\Lambda_n, \vec{t})$
- Partition Λ_n into **layers** of the form: $\Lambda_{n-1} + c\vec{b}_n$, $c = 2, 1, 3, 0, \dots$
- Find lattice point $\vec{v}_i$ in each layer closest to (the projection of) $\vec{t}$
- Only need to consider nearby layers
 - Dual LLL: 2^n layers
 - Dual SVP: n layers
- Select the best solution $\vec{v}_1$

Rank reduction: $CVP(\Lambda_n) \leq 2^n \cdot CVP(\Lambda_{n-1})$



- Goal: Solve $CVP(\Lambda_n, \vec{t})$
- Partition Λ_n into **layers** of the form: $\Lambda_{n-1} + c\vec{b}_n$, $c = 2, 1, 3, 0, \dots$
- Find lattice point $\vec{v}_i$ in each layer closest to (the projection of) $\vec{t}$
- Only need to consider nearby layers
 - Dual LLL: 2^n layers
 - Dual SVP: n layers
- Select the best solution $\vec{v}_1$
- Notice: All layers contain same lattice Λ_{n-1}

- 1 Introduction Lattices
 - Lattice Problems
 - Algorithmic Techniques
- 2 New Algorithm
 - Overview
 - Voronoi Cell
 - CVPP Algorithm
- 3 Final Remarks and Open Problems

Solving CVP by rank reduction

- Rank reduction $CVP(\Lambda_n) \leq k \cdot CVP(\Lambda_{n-1})$
 - LLL: $k = 2^n$,
 - SVP: $k = n$,

Solving CVP by rank reduction

- Rank reduction $CVP(\Lambda_n) \leq k \cdot CVP(\Lambda_{n-1})$
 - LLL: $k = 2^n$, $T = 2^{n^2}$
 - SVP: $k = n$, $T = n^n$
- Iterate: $CVP(\Lambda_n) \leq k \cdot CVP(\Lambda_{n-1}) \leq \dots \leq k^n CVP(\Lambda_1) = k^n$

Solving CVP by rank reduction

- Rank reduction $CVP(\Lambda_n) \leq k \cdot CVP(\Lambda_{n-1})$
 - LLL: $k = 2^n$, $T = 2^{n^2}$
 - SVP: $k = n$, $T = n^n$
- Iterate: $CVP(\Lambda_n) \leq k \cdot CVP(\Lambda_{n-1}) \leq \dots \leq k^n CVP(\Lambda_1) = k^n$
- Our approach
 - Exploit the fact that recursive calls use the same lower dimensional sublattices
 - Preprocess the lattice to speed up the solution of many CVP instances

CVP with Preprocessing (CVPP)

Problem (CVPP)

Find a function π and an efficient algorithm CVPP such that $CVPP(\pi(\Lambda), \vec{t}) = CVP(\Lambda, \vec{t})$

- Only the running time of CVPP counts. The function π is arbitrary.

CVP with Preprocessing (CVPP)

Problem (CVPP)

Find a function π and an efficient algorithm CVPP such that $CVPP(\pi(\Lambda), \vec{t}) = CVP(\Lambda, \vec{t})$

- Only the running time of CVPP counts. The function π is arbitrary.
- Complexity
 - Still NP-hard [M'01]!
 - [LLS'93,AR'04] approximates within $n^{O(1)}$ in polynomial time
 - Polynomial time solutions require $|\pi(\Lambda)| \leq n^{O(1)}$

CVP with Preprocessing (CVPP)

Problem (CVPP)

Find a function π and an efficient algorithm CVPP such that $CVPP(\pi(\Lambda), \vec{t}) = CVP(\Lambda, \vec{t})$

- Only the running time of CVPP counts. The function π is arbitrary.
- Complexity
 - Still NP-hard [M'01]!
 - [LLS'93,AR'04] approximates within $n^{O(1)}$ in polynomial time
 - Polynomial time solutions require $|\pi(\Lambda)| \leq n^{O(1)}$
- Our work:
 - $CVPP(\pi(\Lambda), \vec{t})$ runs in $2^{O(n)}$ time

CVP with Preprocessing (CVPP)

Problem (CVPP)

Find a function π and an efficient algorithm CVPP such that $CVPP(\pi(\Lambda), \vec{t}) = CVP(\Lambda, \vec{t})$

- Only the running time of CVPP counts. The function π is arbitrary.
- Complexity
 - Still NP-hard [M'01]!
 - [LLS'93,AR'04] approximates within $n^{O(1)}$ in polynomial time
 - Polynomial time solutions require $|\pi(\Lambda)| \leq n^{O(1)}$
- Our work:
 - $CVPP(\pi(\Lambda), \vec{t})$ runs in $2^{O(n)}$ time
 - $\pi(\Lambda)$ has size $2^{O(n)}$

CVP with Preprocessing (CVPP)

Problem (CVPP)

Find a function π and an efficient algorithm CVPP such that $CVPP(\pi(\Lambda), \vec{t}) = CVP(\Lambda, \vec{t})$

- Only the running time of CVPP counts. The function π is arbitrary.
- Complexity
 - Still NP-hard [M'01]!
 - [LLS'93,AR'04] approximates within $n^{O(1)}$ in polynomial time
 - Polynomial time solutions require $|\pi(\Lambda)| \leq n^{O(1)}$
- Our work:
 - $CVPP(\pi(\Lambda), \vec{t})$ runs in $2^{O(n)}$ time
 - $\pi(\Lambda)$ has size $2^{O(n)}$
 - $\pi(\Lambda)$ can also be computed in time $2^{O(n)}$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$
- Voronoi cell computation $\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$
- Voronoi cell computation $\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$
- Dimension reduction $CVP(\Lambda_n) \leq 2^{O(n)} \cdot CVP(\Lambda_{n-1})$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$
- Voronoi cell computation $\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$
- Dimension reduction $CVP(\Lambda_n) \leq 2^{O(n)} \cdot CVP(\Lambda_{n-1})$

Computing the Voronoi cell of a lattice:

$$\mathcal{V}(\Lambda_n)$$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$
- **Voronoi cell computation** $\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$
- Dimension reduction $CVP(\Lambda_n) \leq 2^{O(n)} \cdot CVP(\Lambda_{n-1})$

Computing the Voronoi cell of a lattice:

$$\mathcal{V}(\Lambda_n)$$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$
- Voronoi cell computation $\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$
- Dimension reduction $CVP(\Lambda_n) \leq 2^{O(n)} \cdot CVP(\Lambda_{n-1})$

Computing the Voronoi cell of a lattice:

$$\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$
- Voronoi cell computation $\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$
- **Dimension reduction** $CVP(\Lambda_n) \leq 2^{O(n)} \cdot CVP(\Lambda_{n-1})$

Computing the Voronoi cell of a lattice:

$$\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$
- Voronoi cell computation $\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$
- Dimension reduction $CVP(\Lambda_n) \leq 2^{O(n)} \cdot CVP(\Lambda_{n-1})$

Computing the Voronoi cell of a lattice:

$$\begin{aligned}\mathcal{V}(\Lambda_n) &\leq 2^{O(n)} CVP(\Lambda_n) \\ &\leq 2^{O(n)} \cdot 2^{O(n)} \cdot CVP(\Lambda_{n-1})\end{aligned}$$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- **Our approach:** $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$
- Voronoi cell computation $\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$
- Dimension reduction $CVP(\Lambda_n) \leq 2^{O(n)} \cdot CVP(\Lambda_{n-1})$

Computing the Voronoi cell of a lattice:

$$\begin{aligned}\mathcal{V}(\Lambda_n) &\leq 2^{O(n)} CVP(\Lambda_n) \\ &\leq 2^{O(n)} \cdot 2^{O(n)} \cdot CVP(\Lambda_{n-1})\end{aligned}$$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$
- Voronoi cell computation $\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$
- Dimension reduction $CVP(\Lambda_n) \leq 2^{O(n)} \cdot CVP(\Lambda_{n-1})$

Computing the Voronoi cell of a lattice:

$$\begin{aligned}\mathcal{V}(\Lambda_n) &\leq 2^{O(n)} CVP(\Lambda_n) \\ &\leq 2^{O(n)} \cdot 2^{O(n)} \cdot CVP(\Lambda_{n-1}) \\ &\leq 2^{O(n)} \cdot 2^{O(n)} \cdot CVPP(\mathcal{V}(\Lambda_{n-1})) + \mathcal{V}(\Lambda_{n-1})\end{aligned}$$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$
- Voronoi cell computation $\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$
- Dimension reduction $CVP(\Lambda_n) \leq 2^{O(n)} \cdot CVP(\Lambda_{n-1})$

Computing the Voronoi cell of a lattice:

$$\begin{aligned}\mathcal{V}(\Lambda_n) &\leq 2^{O(n)} CVP(\Lambda_n) \\ &\leq 2^{O(n)} \cdot 2^{O(n)} \cdot CVP(\Lambda_{n-1}) \\ &\leq 2^{O(n)} \cdot 2^{O(n)} \cdot CVPP(\mathcal{V}(\Lambda_{n-1})) + \mathcal{V}(\Lambda_{n-1})\end{aligned}$$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$
- Voronoi cell computation $\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$
- Dimension reduction $CVP(\Lambda_n) \leq 2^{O(n)} \cdot CVP(\Lambda_{n-1})$

Computing the Voronoi cell of a lattice:

$$\begin{aligned}\mathcal{V}(\Lambda_n) &\leq 2^{O(n)} CVP(\Lambda_n) \\ &\leq 2^{O(n)} \cdot 2^{O(n)} \cdot CVP(\Lambda_{n-1}) \\ &\leq 2^{O(n)} \cdot 2^{O(n)} \cdot CVPP(\mathcal{V}(\Lambda_{n-1})) + \mathcal{V}(\Lambda_{n-1}) \\ &\leq 2^{O(n)} 2^{O(n)} 2^{O(n)} + \mathcal{V}(\Lambda_{n-1})\end{aligned}$$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$
- Voronoi cell computation $\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$
- Dimension reduction $CVP(\Lambda_n) \leq 2^{O(n)} \cdot CVP(\Lambda_{n-1})$

Computing the Voronoi cell of a lattice:

$$\begin{aligned}\mathcal{V}(\Lambda_n) &\leq 2^{O(n)} CVP(\Lambda_n) \\ &\leq 2^{O(n)} \cdot 2^{O(n)} \cdot CVP(\Lambda_{n-1}) \\ &\leq 2^{O(n)} \cdot 2^{O(n)} \cdot CVPP(\mathcal{V}(\Lambda_{n-1})) + \mathcal{V}(\Lambda_{n-1}) \\ &\leq 2^{O(n)} 2^{O(n)} 2^{O(n)} + \mathcal{V}(\Lambda_{n-1}) \\ &= 2^{O(n)} + \mathcal{V}(\Lambda_{n-1})\end{aligned}$$

Overview of CVP algorithm

Building blocks:

- $\pi(\Lambda) = \mathcal{V}(\Lambda)$: Voronoi cell of the lattice
- Our approach: $CVP(\Lambda_n) \leq CVPP(\mathcal{V}(\Lambda_n)) + \mathcal{V}(\Lambda_n)$
- $CVPP(\mathcal{V}(\Lambda_n))$ algorithm with running time $2^{O(n)}$
- Voronoi cell computation $\mathcal{V}(\Lambda_n) \leq 2^{O(n)} CVP(\Lambda_n)$
- Dimension reduction $CVP(\Lambda_n) \leq 2^{O(n)} \cdot CVP(\Lambda_{n-1})$

Computing the Voronoi cell of a lattice:

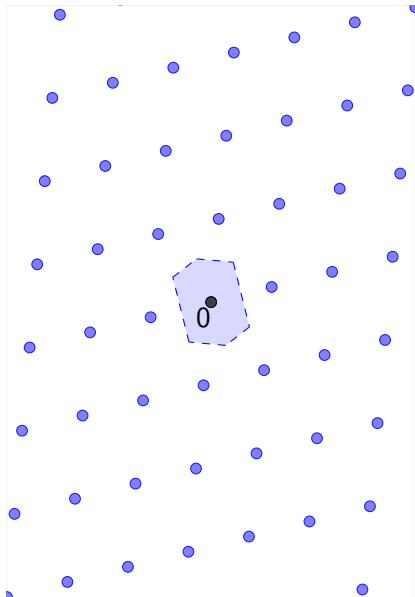
$$\begin{aligned}\mathcal{V}(\Lambda_n) &\leq 2^{O(n)} CVP(\Lambda_n) \\ &\leq 2^{O(n)} \cdot 2^{O(n)} \cdot CVP(\Lambda_{n-1}) \\ &\leq 2^{O(n)} \cdot 2^{O(n)} \cdot CVPP(\mathcal{V}(\Lambda_{n-1})) + \mathcal{V}(\Lambda_{n-1}) \\ &\leq 2^{O(n)} 2^{O(n)} 2^{O(n)} + \mathcal{V}(\Lambda_{n-1}) \\ &= 2^{O(n)} + \mathcal{V}(\Lambda_{n-1}) \\ &\leq 2^{O(n)} + 2^{O(n)} + \mathcal{V}(\Lambda_{n-2}) \leq \dots \leq 2^{O(n)}\end{aligned}$$

- 1 Introduction Lattices
 - Lattice Problems
 - Algorithmic Techniques

- 2 New Algorithm
 - Overview
 - Voronoi Cell
 - CVPP Algorithm

- 3 Final Remarks and Open Problems

Voronoi Cell

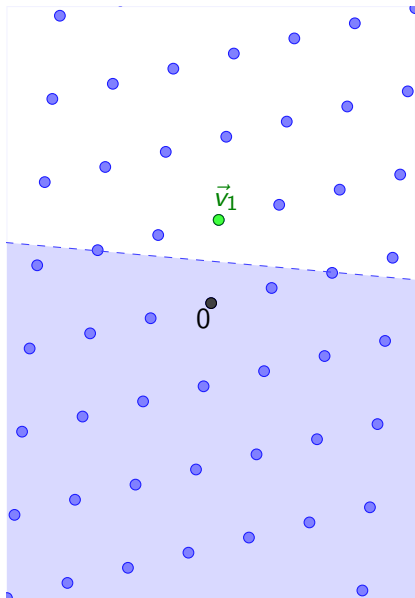


Definition (Voronoi Cell)

Set of points in $\mathbb{R}^n$ closer to 0 than to any other lattice point

$$\mathcal{V}(\Lambda) = \{\vec{x}: \forall \vec{v} \in \Lambda, \|\vec{x}\| \leq \|\vec{x} - \vec{v}\|\}$$

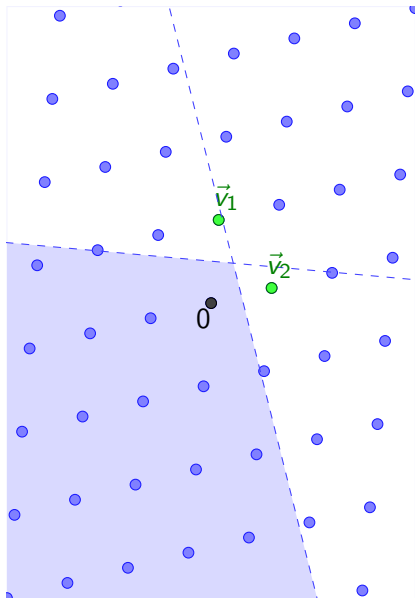
Representing the Voronoi cell



- Each $\vec{v} \in \Lambda$ defines

$$\mathcal{H}_{\vec{v}} = \{\vec{x}: \|\vec{x}\| \leq \|\vec{x} - \vec{v}\|\}$$

Representing the Voronoi cell



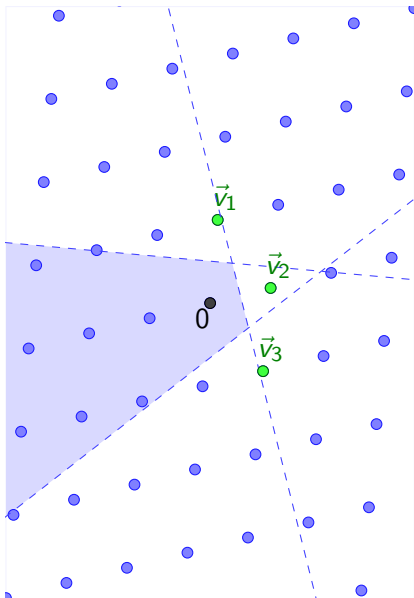
- Each $\vec{v} \in \Lambda$ defines

$$\mathcal{H}_{\vec{v}} = \{\vec{x}: \|\vec{x}\| \leq \|\vec{x} - \vec{v}\|\}$$

- $\mathcal{V}$ is the intersection

$$\mathcal{V} = \bigcap_{\vec{v} \in \Lambda} \mathcal{H}_{\vec{v}}$$

Representing the Voronoi cell



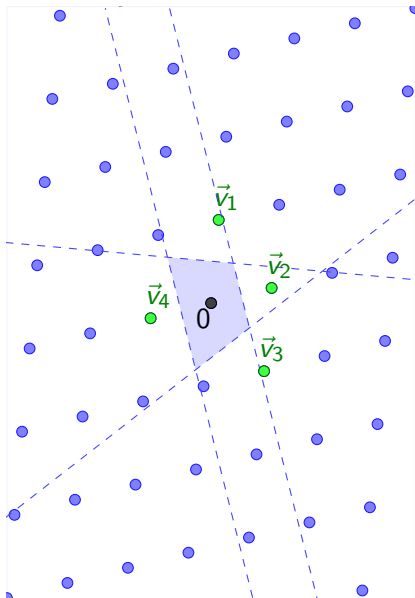
- Each $\vec{v} \in \Lambda$ defines

$$\mathcal{H}_{\vec{v}} = \{\vec{x}: \|\vec{x}\| \leq \|\vec{x} - \vec{v}\|\}$$

- $\mathcal{V}$ is the intersection

$$\mathcal{V} = \bigcap_{\vec{v} \in \Lambda} \mathcal{H}_{\vec{v}}$$

Representing the Voronoi cell



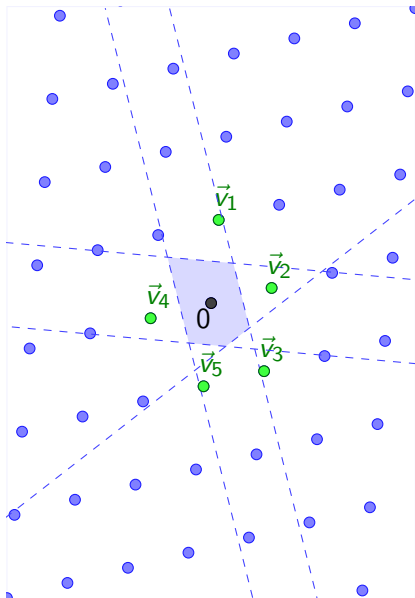
- Each $\vec{v} \in \Lambda$ defines

$$\mathcal{H}_{\vec{v}} = \{\vec{x}: \|\vec{x}\| \leq \|\vec{x} - \vec{v}\|\}$$

- $\mathcal{V}$ is the intersection

$$\mathcal{V} = \bigcap_{\vec{v} \in \Lambda} \mathcal{H}_{\vec{v}}$$

Representing the Voronoi cell



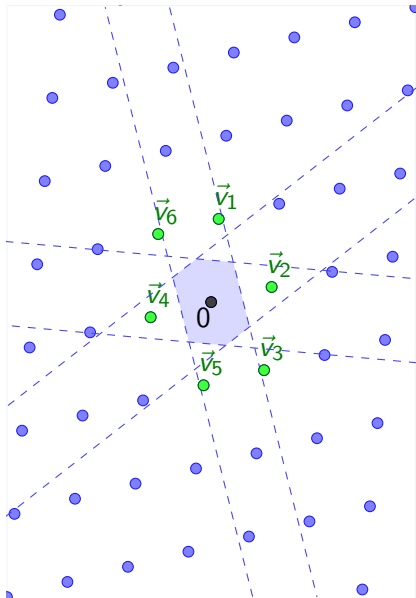
- Each $\vec{v} \in \Lambda$ defines

$$\mathcal{H}_{\vec{v}} = \{\vec{x}: \|\vec{x}\| \leq \|\vec{x} - \vec{v}\|\}$$

- $\mathcal{V}$ is the intersection

$$\mathcal{V} = \bigcap_{\vec{v} \in \Lambda} \mathcal{H}_{\vec{v}}$$

Representing the Voronoi cell



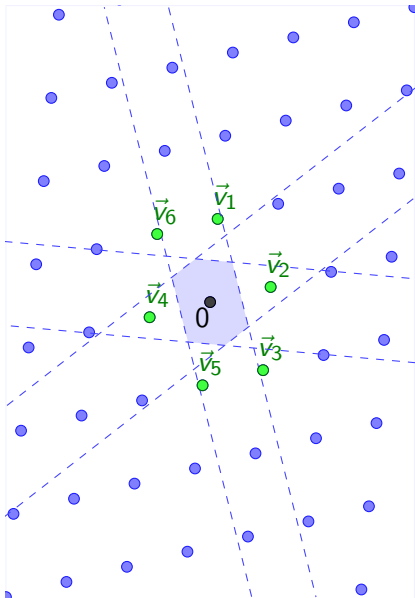
- Each $\vec{v} \in \Lambda$ defines

$$\mathcal{H}_{\vec{v}} = \{\vec{x}: \|\vec{x}\| \leq \|\vec{x} - \vec{v}\|\}$$

- $\mathcal{V}$ is the intersection

$$\mathcal{V} = \bigcap_{\vec{v} \in \Lambda} \mathcal{H}_{\vec{v}}$$

Representing the Voronoi cell



- Each $\vec{v} \in \Lambda$ defines

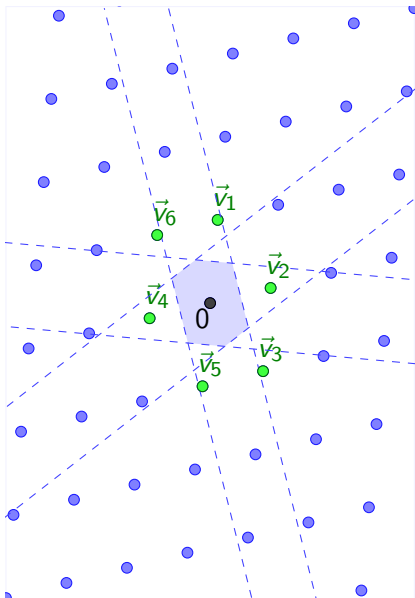
$$\mathcal{H}_{\vec{v}} = \{\vec{x}: \|\vec{x}\| \leq \|\vec{x} - \vec{v}\|\}$$

- $\mathcal{V}$ is the intersection

$$\mathcal{V} = \bigcap_{\vec{v} \in R} \mathcal{H}_{\vec{v}}, R \subset \Lambda$$

- Not all $\vec{v} \in \Lambda$ are needed

Representing the Voronoi cell



- Each $\vec{v} \in \Lambda$ defines

$$\mathcal{H}_{\vec{v}} = \{\vec{x}: \|\vec{x}\| \leq \|\vec{x} - \vec{v}\|\}$$

- $\mathcal{V}$ is the intersection

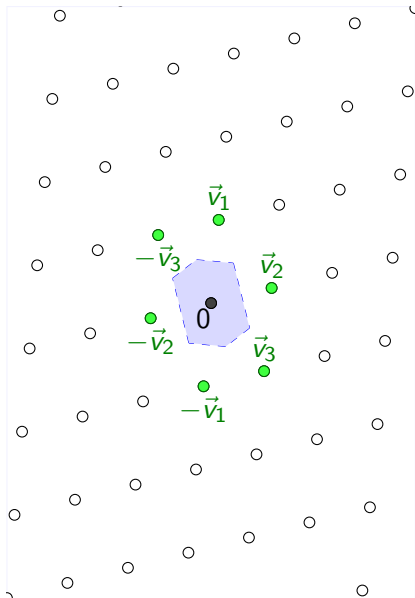
$$\mathcal{V} = \bigcap_{\vec{v} \in R} \mathcal{H}_{\vec{v}}, R \subset \Lambda$$

- Not all $\vec{v} \in \Lambda$ are needed

Theorem (Voronoi)

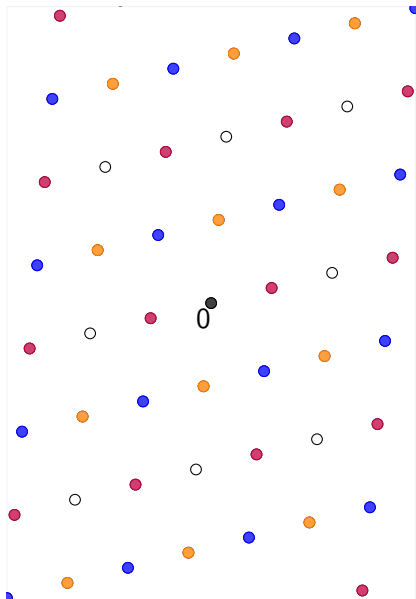
The number of relevant points is at most $|R| \leq 2 \cdot (2^n - 1)$

Computing $\mathcal{V}(\Lambda_n)$



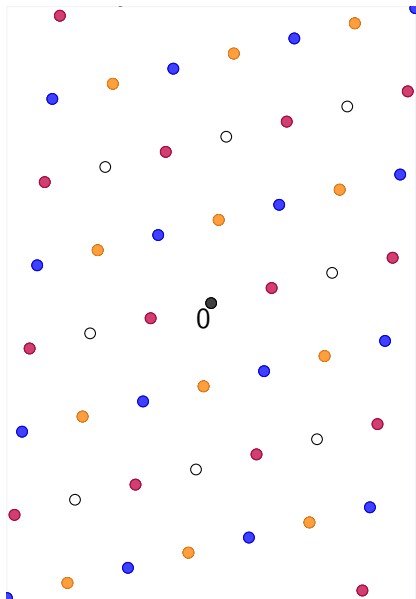
- Why is $|R| \leq 2 \cdot (2^n - 1)$?

Computing $\mathcal{V}(\Lambda_n)$



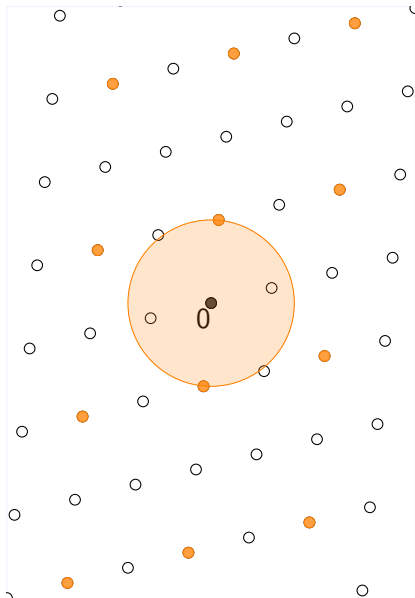
- Why is $|R| \leq 2 \cdot (2^n - 1)$?
- Partition Λ into cosets modulo 2Λ

Computing $\mathcal{V}(\Lambda_n)$



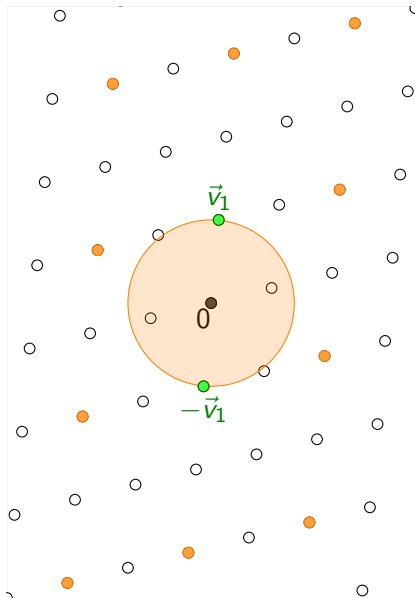
- Why is $|R| \leq 2 \cdot (2^n - 1)$?
- Partition Λ into cosets modulo 2Λ
- There are $2^n - 1$ nonzero cosets

Computing $\mathcal{V}(\Lambda_n)$



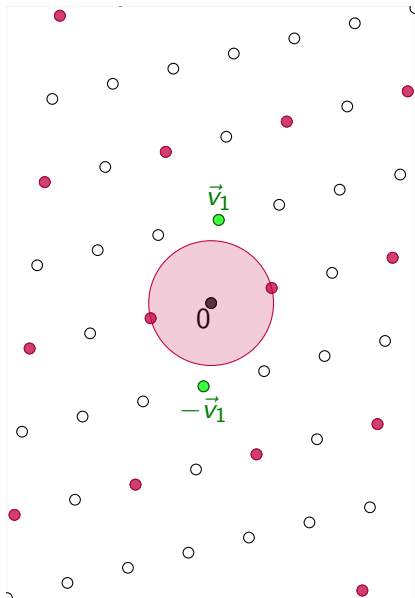
- Why is $|R| \leq 2 \cdot (2^n - 1)$?
- Partition Λ into cosets modulo 2Λ
- There are $2^n - 1$ nonzero cosets
- From each coset, select the pair $\vec{v}, -\vec{v}$ closest to $\vec{0}$

Computing $\mathcal{V}(\Lambda_n)$



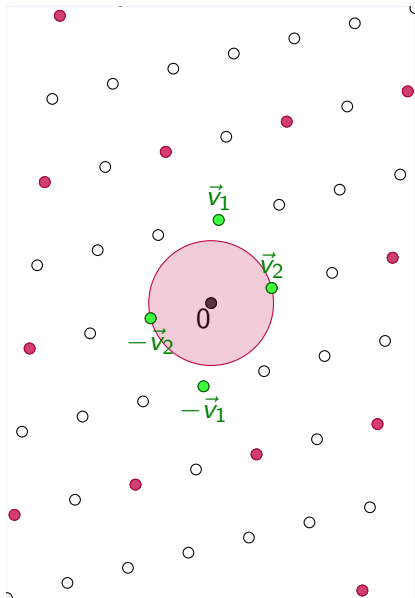
- Why is $|R| \leq 2 \cdot (2^n - 1)$?
- Partition Λ into cosets modulo 2Λ
- There are $2^n - 1$ nonzero cosets
- From each coset, select the pair $\vec{v}, -\vec{v}$ closest to $\vec{0}$

Computing $\mathcal{V}(\Lambda_n)$



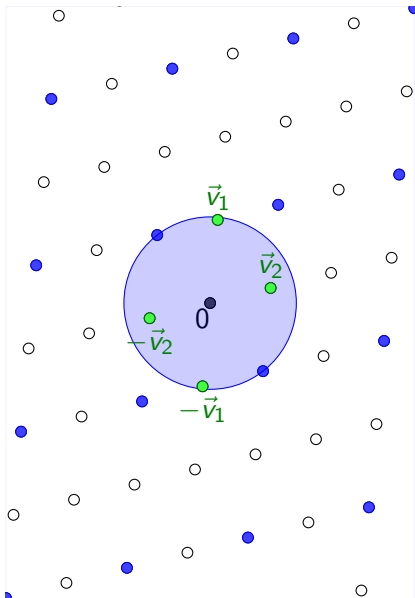
- Why is $|R| \leq 2 \cdot (2^n - 1)$?
- Partition Λ into cosets modulo 2Λ
- There are $2^n - 1$ nonzero cosets
- From each coset, select the pair $\vec{v}, -\vec{v}$ closest to $\vec{0}$

Computing $\mathcal{V}(\Lambda_n)$



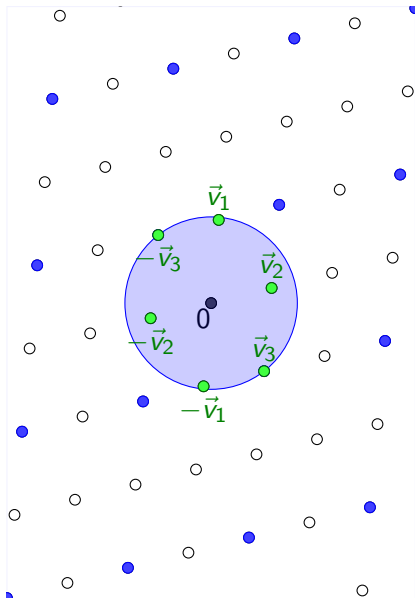
- Why is $|R| \leq 2 \cdot (2^n - 1)$?
- Partition Λ into cosets modulo 2Λ
- There are $2^n - 1$ nonzero cosets
- From each coset, select the pair $\vec{v}, -\vec{v}$ closest to $\vec{0}$

Computing $\mathcal{V}(\Lambda_n)$



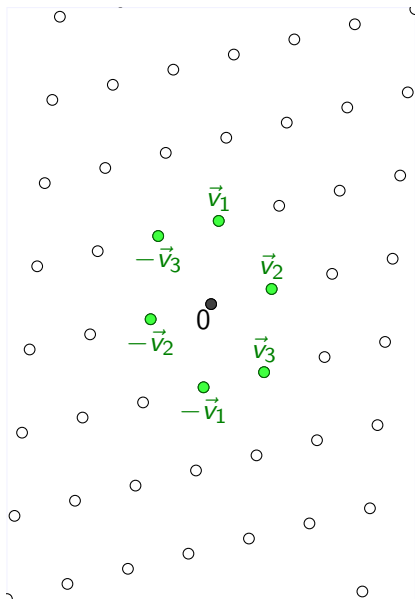
- Why is $|R| \leq 2 \cdot (2^n - 1)$?
- Partition Λ into cosets modulo 2Λ
- There are $2^n - 1$ nonzero cosets
- From each coset, select the pair $\vec{v}, -\vec{v}$ closest to $\vec{0}$

Computing $\mathcal{V}(\Lambda_n)$



- Why is $|R| \leq 2 \cdot (2^n - 1)$?
- Partition Λ into cosets modulo 2Λ
- There are $2^n - 1$ nonzero cosets
- From each coset, select the pair $\vec{v}, -\vec{v}$ closest to $\vec{0}$

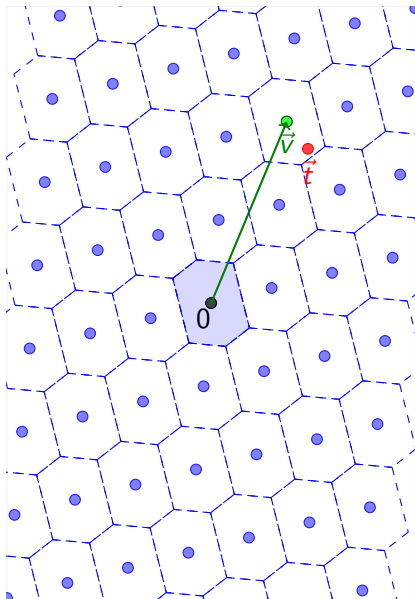
Computing $\mathcal{V}(\Lambda_n) \leq 2^n \text{CVP}(\Lambda_n)$



- Why is $|R| \leq 2 \cdot (2^n - 1)$?
- Partition Λ into cosets modulo 2Λ
- There are $2^n - 1$ nonzero cosets
- From each coset, select the pair $\vec{v}, -\vec{v}$ closest to $\vec{0}$
- R is the set of all such pairs
- Each pair is found by a CVP computation in lattice 2Λ
- $\text{CVP}(2\Lambda)$ is equivalent to $\text{CVP}(\Lambda)$

- 1 Introduction Lattices
 - Lattice Problems
 - Algorithmic Techniques
- 2 New Algorithm
 - Overview
 - Voronoi Cell
 - CVPP Algorithm
- 3 Final Remarks and Open Problems

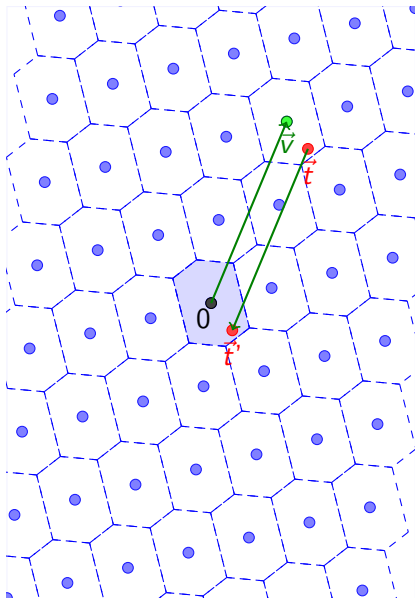
CVP and Voronoi cell



Definition (CVP)

Given Λ and $\vec{t}$, find $\vec{v} \in \Lambda$ such that $\vec{t} \in \vec{v} + \mathcal{V}$

CVP and Voronoi cell

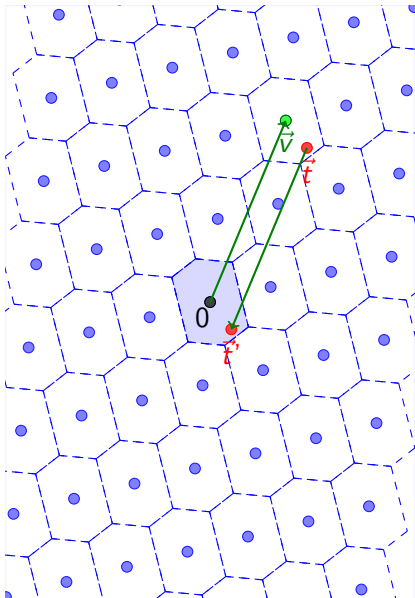


Definition (CVP)

Given Λ and $\vec{t}$, find $\vec{v} \in \Lambda$ such that $\vec{t} \in \vec{v} + \mathcal{V}$

- $\vec{t} \in \vec{v} + \mathcal{V} \equiv \vec{t} - \vec{v} \in \mathcal{V}$

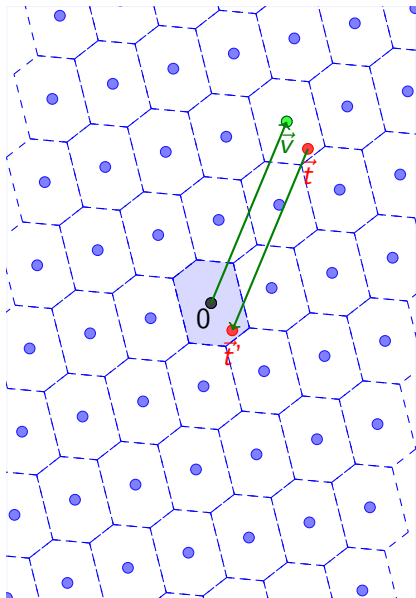
CVP and Voronoi cell



Definition (CVP)

Given Λ and $\vec{t}$, find $\vec{v} \in \Lambda$ such that $\vec{t} \in \vec{v} + \mathcal{V}$

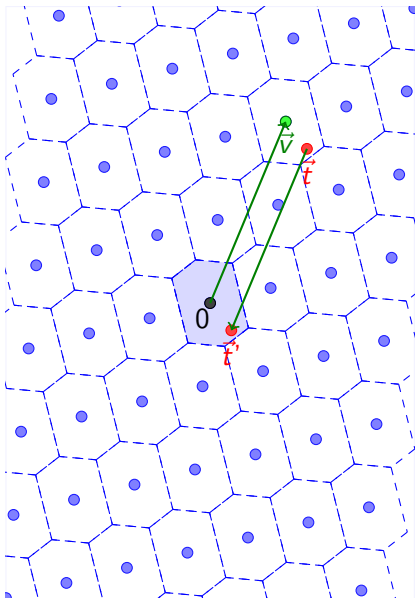
- $\vec{t} \in \vec{v} + \mathcal{V} \equiv \vec{t} - \vec{v} \in \mathcal{V}$
- CVP goal: bring $\vec{t}$ inside $\mathcal{V}$ by shifting it by $\vec{v} \in \Lambda$



Definition (CVP)

Given Λ and $\vec{t}$, find $\vec{v} \in \Lambda$ such that $\vec{t} \in \vec{v} + \mathcal{V}$

- $\vec{t} \in \vec{v} + \mathcal{V} \equiv \vec{t} - \vec{v} \in \mathcal{V}$
- CVP goal: bring $\vec{t}$ inside $\mathcal{V}$ by shifting it by $\vec{v} \in \Lambda$
- Algorithm [SFS'09]:
 - While $\vec{t} \notin \mathcal{V}$:
 - Select $\vec{v} \in R$. $\vec{t} \notin \mathcal{H}_{\vec{v}}$
 - size reduce $\vec{t}$ using $\vec{v}$

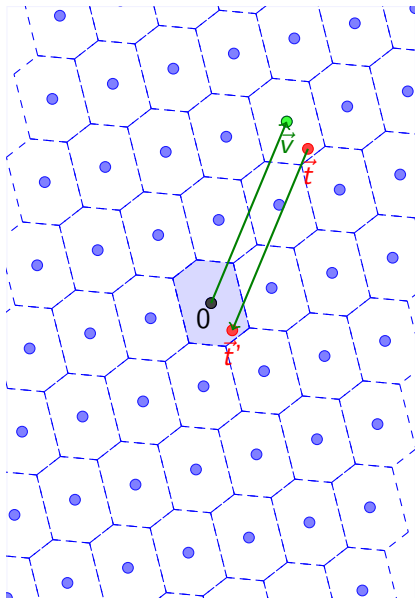


Definition (CVP)

Given Λ and $\vec{t}$, find $\vec{v} \in \Lambda$ such that $\vec{t} \in \vec{v} + \mathcal{V}$

- $\vec{t} \in \vec{v} + \mathcal{V} \equiv \vec{t} - \vec{v} \in \mathcal{V}$
- CVP goal: bring $\vec{t}$ inside $\mathcal{V}$ by shifting it by $\vec{v} \in \Lambda$
- Algorithm [SFS'09]:
 - While $\vec{t} \notin \mathcal{V}$:
 - Select $\vec{v} \in R$. $\vec{t} \notin \mathcal{H}_{\vec{v}}$
 - size reduce $\vec{t}$ using $\vec{v}$

[SFS'09] only proves termination



Definition (CVP)

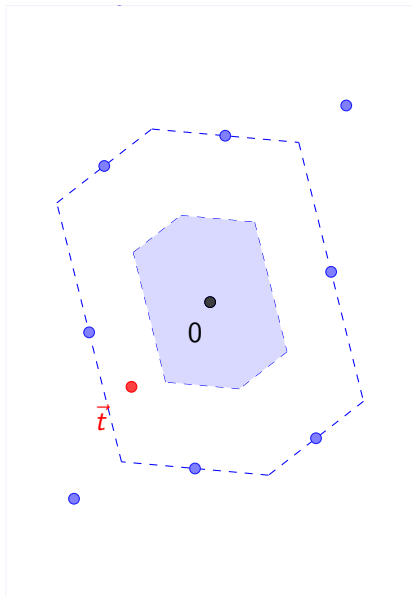
Given Λ and $\vec{t}$, find $\vec{v} \in \Lambda$ such that $\vec{t} \in \vec{v} + \mathcal{V}$

- $\vec{t} \in \vec{v} + \mathcal{V} \equiv \vec{t} - \vec{v} \in \mathcal{V}$
- CVP goal: bring $\vec{t}$ inside $\mathcal{V}$ by shifting it by $\vec{v} \in \Lambda$
- Algorithm [SFS'09]:
 - While $\vec{t} \notin \mathcal{V}$:
 - **Select** $\vec{v} \in R$. $\vec{t} \notin \mathcal{H}_{\vec{v}}$
 - size reduce $\vec{t}$ using $\vec{v}$

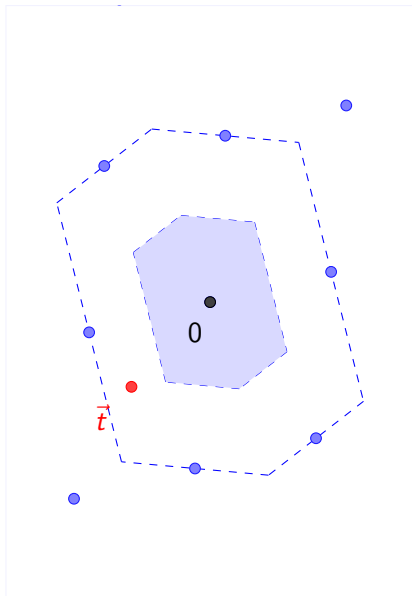
[SFS'09] only proves termination
Question: What is a good selection strategy for $\vec{v} \in R$?

Our selection strategy

- Assume $\vec{t} \in 2\mathcal{V}$

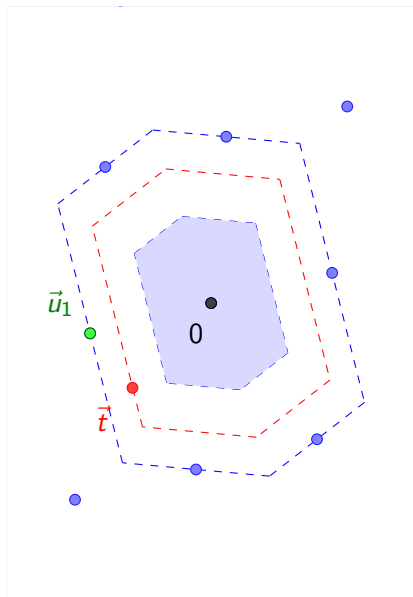


Our selection strategy



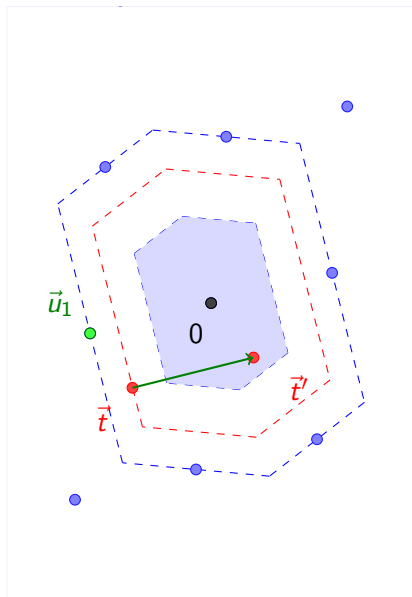
- Assume $\vec{t} \in 2\mathcal{V}$
- Goal: find $\vec{t}' \in \vec{t} - \Lambda \cap \mathcal{V}$:

Our selection strategy



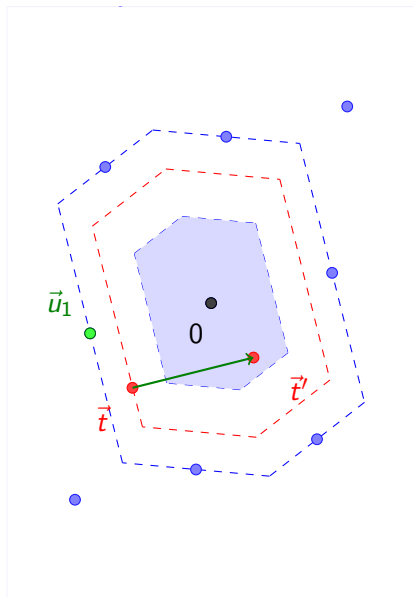
- Assume $\vec{t} \in 2\mathcal{V}$
- Goal: find $\vec{t}' \in \vec{t} - \Lambda \cap \mathcal{V}$:
- Strategy:
 - Compute smallest $k \in \mathbb{R}$ such that $\vec{t} \in k\mathcal{V}$

Our selection strategy



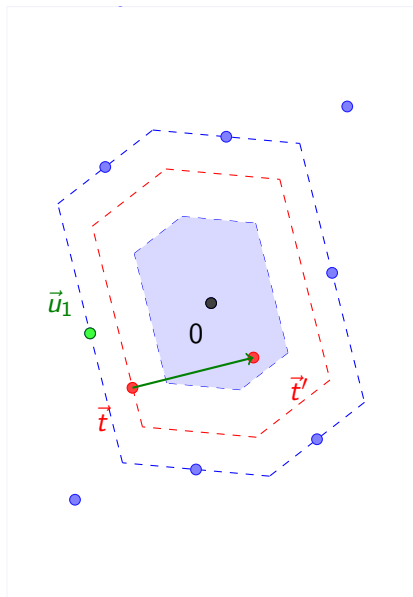
- Assume $\vec{t} \in 2\mathcal{V}$
- Goal: find $\vec{t}' \in \vec{t} - \Lambda \cap \mathcal{V}$:
- Strategy:
 - Compute smallest $k \in \mathbb{R}$ such that $\vec{t} \in k\mathcal{V}$
 - Subtract the relevant vector associated to corresponding facet

Our selection strategy



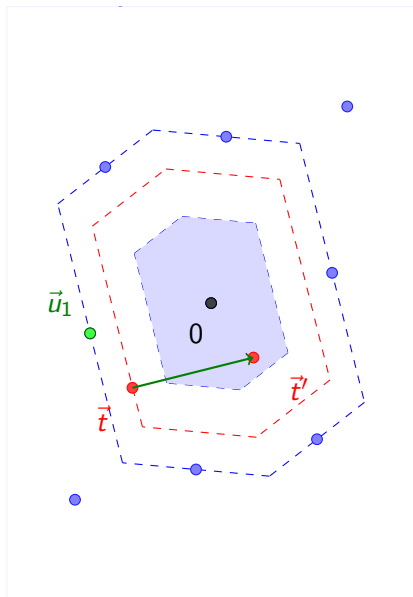
- Assume $\vec{t} \in 2\mathcal{V}$
- Goal: find $\vec{t}' \in \vec{t} - \Lambda \cap \mathcal{V}$:
- Strategy:
 - Compute smallest $k \in \mathbb{R}$ such that $\vec{t} \in k\mathcal{V}$
 - Subtract the relevant vector associated to corresponding facet
- Why does it work?

Our selection strategy



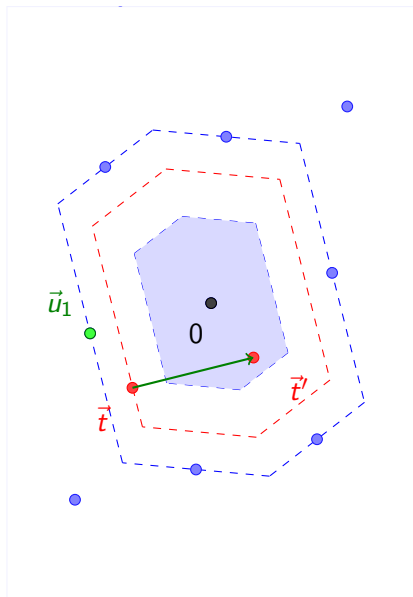
- Assume $\vec{t} \in 2\mathcal{V}$
- Goal: find $\vec{t}' \in \vec{t} - \Lambda \cap \mathcal{V}$:
- Strategy:
 - Compute smallest $k \in \mathbb{R}$ such that $\vec{t} \in k\mathcal{V}$
 - Subtract the relevant vector associated to corresponding facet
- Why does it work?
 - The new vector $\vec{t}'$ is shorter than $\vec{t}$

Our selection strategy



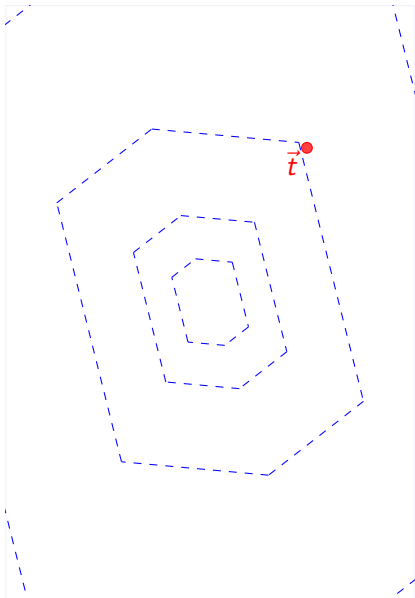
- Assume $\vec{t} \in 2\mathcal{V}$
- Goal: find $\vec{t}' \in \vec{t} - \Lambda \cap \mathcal{V}$:
- Strategy:
 - Compute smallest $k \in \mathbb{R}$ such that $\vec{t} \in k\mathcal{V}$
 - Subtract the relevant vector associated to corresponding facet
- Why does it work?
 - The new vector $\vec{t}'$ is shorter than $\vec{t}$
 - still $\vec{t}' \in 2\mathcal{V}$

Our selection strategy



- Assume $\vec{t} \in 2\mathcal{V}$
- Goal: find $\vec{t}' \in \vec{t} - \Lambda \cap \mathcal{V}$:
- Strategy:
 - Compute smallest $k \in \mathbb{R}$ such that $\vec{t} \in k\mathcal{V}$
 - Subtract the relevant vector associated to corresponding facet
- Why does it work?
 - The new vector $\vec{t}'$ is shorter than $\vec{t}$
 - still $\vec{t}' \in 2\mathcal{V}$
 - $|(\vec{t} - \Lambda) \cap 2\mathcal{V}| \leq 2^n$

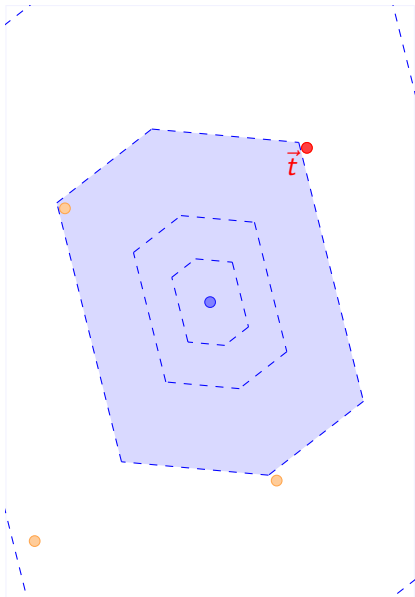
Doubling the Voronoi Cell



Solve CVP for any $\vec{t}$:

- Find $\vec{k} \in \mathbb{Z}$ such that $\vec{t} \in 2^k \mathcal{V}$
- Use $\text{CVP}_{2^k \mathcal{V}}$ to go from $2^k \mathcal{V}$ to $2^{k-1} \mathcal{V}$

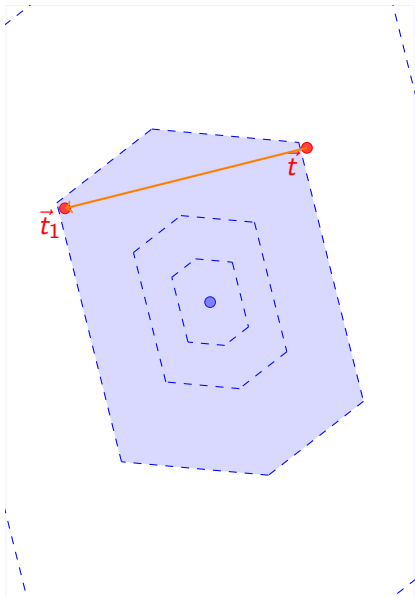
Doubling the Voronoi Cell



Solve CVP for any $\vec{t}$:

- Find $\vec{k} \in \mathbb{Z}$ such that $\vec{t} \in 2^k \mathcal{V}$
- Use $\text{CVP}_{2^k \mathcal{V}}$ to go from $2^k \mathcal{V}$ to $2^{k-1} \mathcal{V}$

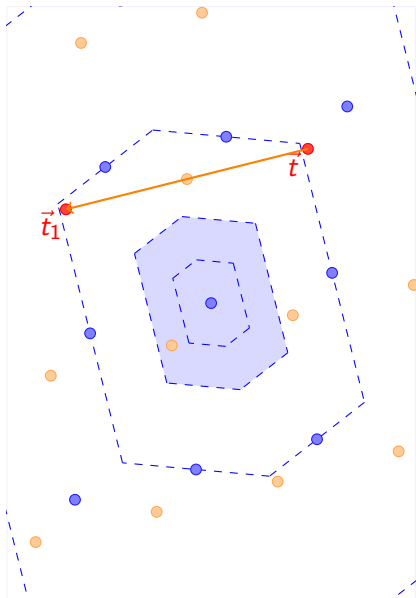
Doubling the Voronoi Cell



Solve CVP for any $\vec{t}$:

- Find $\vec{k} \in \mathbb{Z}$ such that $\vec{t} \in 2^k \mathcal{V}$
- Use $\text{CVP}_{2^k \mathcal{V}}$ to go from $2^k \mathcal{V}$ to $2^{k-1} \mathcal{V}$

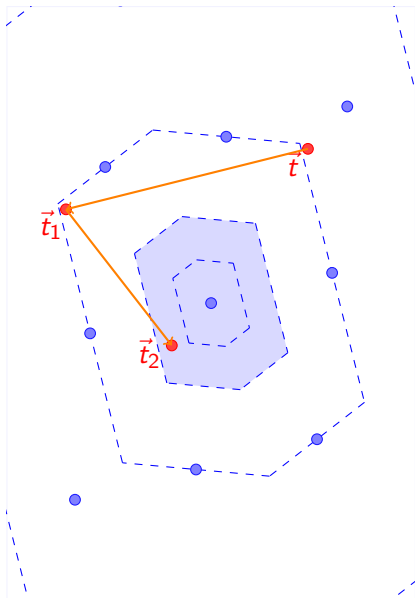
Doubling the Voronoi Cell



Solve CVP for any $\vec{t}$:

- Find $\vec{k} \in \mathbb{Z}$ such that $\vec{t} \in 2^k \mathcal{V}$
- Use $\text{CVP}_{2\mathcal{V}}$ to go from $2^k \mathcal{V}$ to $2^{k-1} \mathcal{V}$

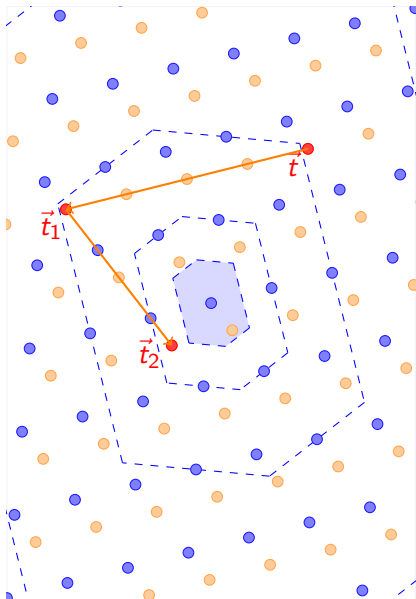
Doubling the Voronoi Cell



Solve CVP for any $\vec{t}$:

- Find $\vec{k} \in \mathbb{Z}$ such that $\vec{t} \in 2^k \mathcal{V}$
- Use $\text{CVP}_{2\mathcal{V}}$ to go from $2^k \mathcal{V}$ to $2^{k-1} \mathcal{V}$

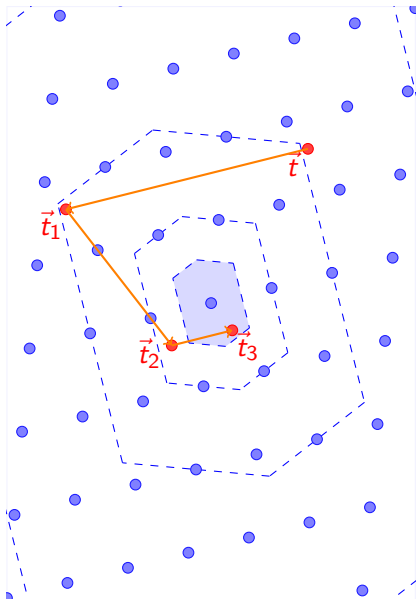
Doubling the Voronoi Cell



Solve CVP for any $\vec{t}$:

- Find $\vec{k} \in \mathbb{Z}$ such that $\vec{t} \in 2^k \mathcal{V}$
- Use $\text{CVP}_{2^k \mathcal{V}}$ to go from $2^k \mathcal{V}$ to $2^{k-1} \mathcal{V}$

Doubling the Voronoi Cell



Solve CVP for any $\vec{t}$:

- Find $\vec{k} \in \mathbb{Z}$ such that $\vec{t} \in 2^k \mathcal{V}$
- Use $\text{CVP}_{2^k \mathcal{V}}$ to go from $2^k \mathcal{V}$ to $2^{k-1} \mathcal{V}$

Summary

- CVP can be solved deterministically in time $2^{c \cdot n}$
- Algorithms for SVP, SIVP and many other problems follow by reduction
- Question: what is the best possible c ?
 - Under ETH, $c = \Omega(1)$
 - In this talk, we didn't optimize c
 - With some more work, we can reduce $c = 2$
- SVP: improves previous $c < 2.5$, deterministically!
- CVP: First $2^{O(n)}$ time algorithm, and first asymptotic improvement since [K'87]

Open Problems

- Reduce space complexity to polynomial
 - Closely related to the problem of **compressing** the description of the Voronoi cell of a lattice

Open Problems

- Reduce space complexity to polynomial
 - Closely related to the problem of **compressing** the description of the Voronoi cell of a lattice
- Faster CVPP solutions
 - Can the number of iterations in our algorithm be bounded by $O(n)$?
 - Can CVPP be **approximated** in polynomial time using approximate Voronoi cell?

Open Problems

- Reduce space complexity to polynomial
 - Closely related to the problem of **compressing** the description of the Voronoi cell of a lattice
- Faster CVPP solutions
 - Can the number of iterations in our algorithm be bounded by $O(n)$?
 - Can CVPP be **approximated** in polynomial time using approximate Voronoi cell?
- Extend algorithms to other norms (e.g., ℓ_∞)
 - Useful in cryptanalysis, integer programming, optimization, etc.
 - Is the number of ℓ_∞ -relevant points still bounded by $2^{O(n)}$

Open Problems

- Reduce space complexity to polynomial
 - Closely related to the problem of **compressing** the description of the Voronoi cell of a lattice
- Faster CVPP solutions
 - Can the number of iterations in our algorithm be bounded by $O(n)$?
 - Can CVPP be **approximated** in polynomial time using approximate Voronoi cell?
- Extend algorithms to other norms (e.g., ℓ_∞)
 - Useful in cryptanalysis, integer programming, optimization, etc.
 - Is the number of ℓ_∞ -relevant points still bounded by $2^{O(n)}$
- Better algorithms for special classes lattices (e.g., ideals of the ring of integers of algebraic number fields)
 - Small improvements can be obtained using symmetries
 - No NP-hardness results, so subexponential algorithms may be possible
 - Important for cryptographic applications