Sieving for shortest vectors in lattices using angular locality-sensitive hashing

Thijs Laarhoven

mail@thijs.com
http://www.thijs.com/

EiPSI seminar, Eindhoven, The Netherlands
(December 1, 2014)

Outline

Lattices

Enumeration algorithms
- Fincke-Pohst enumeration
- Kannan enumeration
- Pruning the enumeration tree

The Voronoi cell algorithm

Sieving algorithms
- Nguyen-Vidick sieve
- Multiple levels
- GaussSieve

Sieving using angular locality-sensitive hashing
- Nguyen-Vidick sieve
- GaussSieve

Outline

Lattices

Enumeration algorithms
- Fincke-Pohst enumeration
- Kannan enumeration
- Pruning the enumeration tree

The Voronoi cell algorithm

Sieving algorithms
- Nguyen-Vidick sieve
- Multiple levels
- GaussSieve

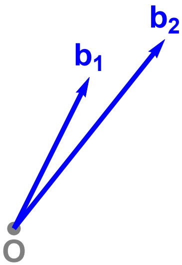
Sieving using angular locality-sensitive hashing
- Nguyen-Vidick sieve
- GaussSieve

Lattices

What is a lattice?

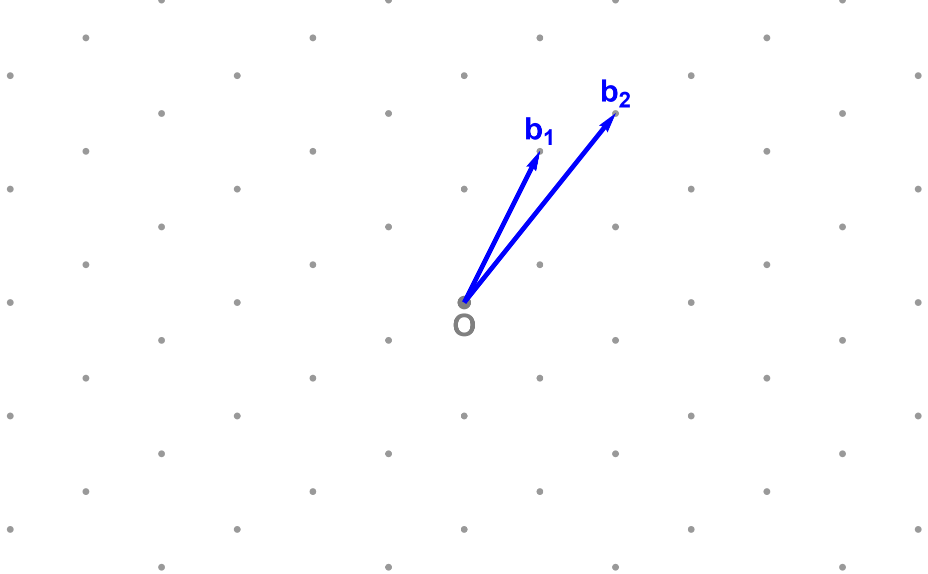
Lattices

What is a lattice?



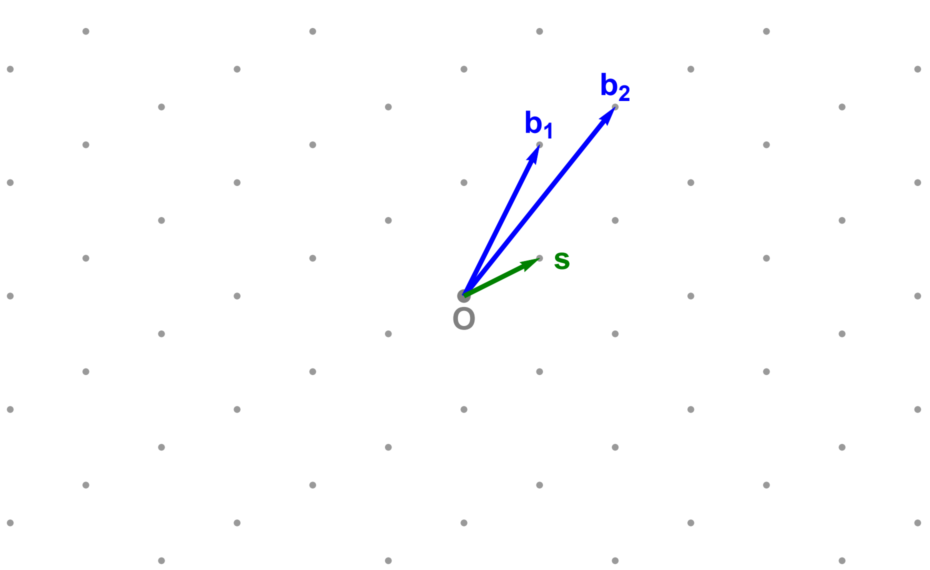
Lattices

What is a lattice?



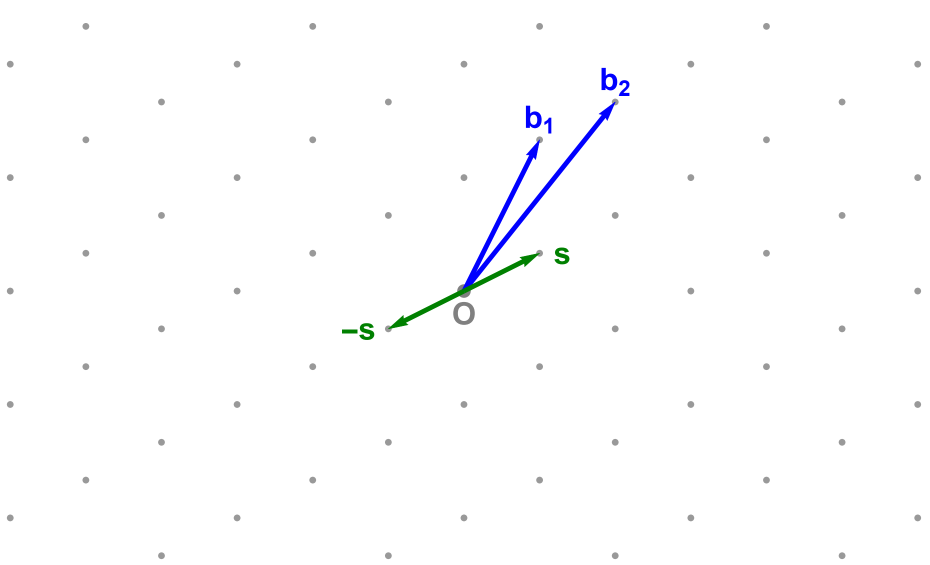
Lattices

Shortest Vector Problem (SVP)



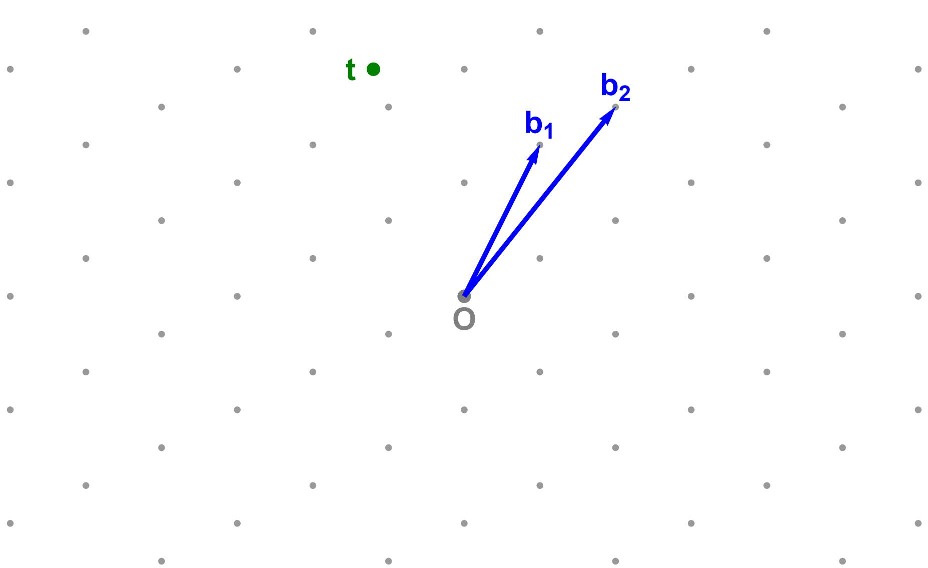
Lattices

Shortest Vector Problem (SVP)



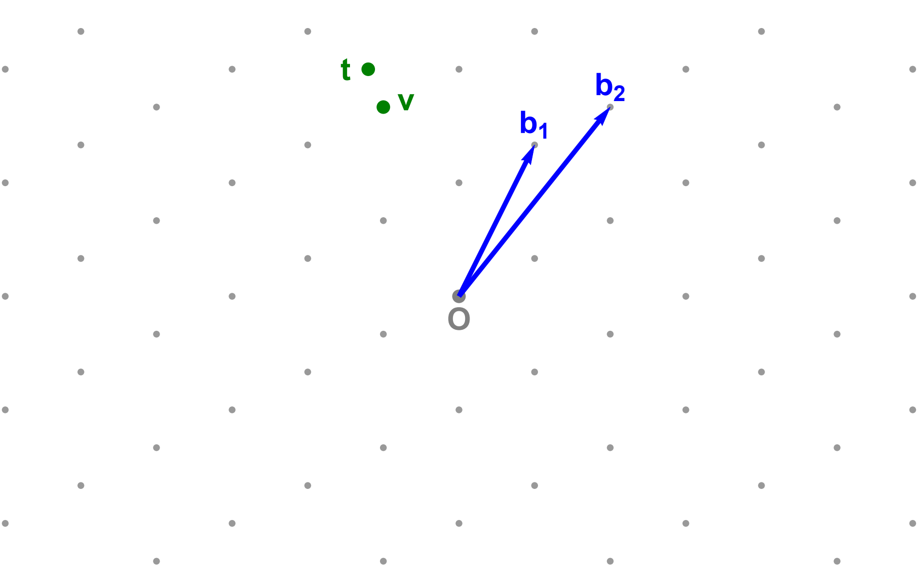
Lattices

Closest Vector Problem (CVP)



Lattices

Closest Vector Problem (CVP)



Lattices

Applications

- "Constructive cryptography": Lattice-based cryptosystems
  - Based on hard lattice problems (SVP, CVP)
  - NTRU cryptosystem [HPS98]
  - Fully Homomorphic Encryption [Gen09]
  - Worst-case to average-case reductions [Ajt96]
  - Candidate for post-quantum cryptography

Lattices

Applications

- "Constructive cryptography": Lattice-based cryptosystems
  - Based on hard lattice problems (SVP, CVP)
  - NTRU cryptosystem [HPS98]
  - Fully Homomorphic Encryption [Gen09]
  - Worst-case to average-case reductions [Ajt96]
  - Candidate for post-quantum cryptography
- "Destructive cryptography": Lattice cryptanalysis
  - Attack knapsack-based cryptosystems [Sha82, LO85]
  - Attack RSA with Coppersmith's method [Cop97]
  - Attack DSA and ECDSA [NS02, NS03]
  - Attack lattice-based cryptosystems [Ngu99, JJ00]

Lattices

Applications

- "Constructive cryptography": Lattice-based cryptosystems
  - Based on hard lattice problems (SVP, CVP)
  - NTRU cryptosystem [HPS98]
  - Fully Homomorphic Encryption [Gen09]
  - Worst-case to average-case reductions [Ajt96]
  - Candidate for post-quantum cryptography
- "Destructive cryptography": Lattice cryptanalysis
  - Attack knapsack-based cryptosystems [Sha82, LO85]
  - Attack RSA with Coppersmith's method [Cop97]
  - Attack DSA and ECDSA [NS02, NS03]
  - Attack lattice-based cryptosystems [Ngu99, JJ00]

How hard are hard lattice problems such as SVP?

Outline

Lattices

Enumeration algorithms

- Fincke-Pohst enumeration
- Kannan enumeration
- Pruning the enumeration tree

The Voronoi cell algorithm

Sieving algorithms

- Nguyen-Vidick sieve
- Multiple levels
- GaussSieve

Sieving using angular locality-sensitive hashing

- Nguyen-Vidick sieve
- GaussSieve

Enumeration

Overview

Studied since the '80s [Poh81, Kan83, FP85, ..., GNR10, MW14]

Procedure:

1. Determine possible coefficients of $\boldsymbol{b}_n$

Enumeration

Overview

Studied since the '80s [Poh81, Kan83, FP85, ..., GNR10, MW14]

Procedure:

1. Determine possible coefficients of $\boldsymbol{b}_n$
2. Find short vectors for each coefficient of $\boldsymbol{b}_n$ (recursively)

Enumeration

Overview

Studied since the ’80s [Poh81, Kan83, FP85, . . . , GNR10, MW14]

Procedure:

1. Determine possible coefficients of $\boldsymbol{b}_n$
2. Find short vectors for each coefficient of $\boldsymbol{b}_n$ (recursively)
3. Find a shortest vector among all found vectors

Enumeration

Overview

Studied since the ’80s [Poh81, Kan83, FP85, ..., GNR10, MW14]

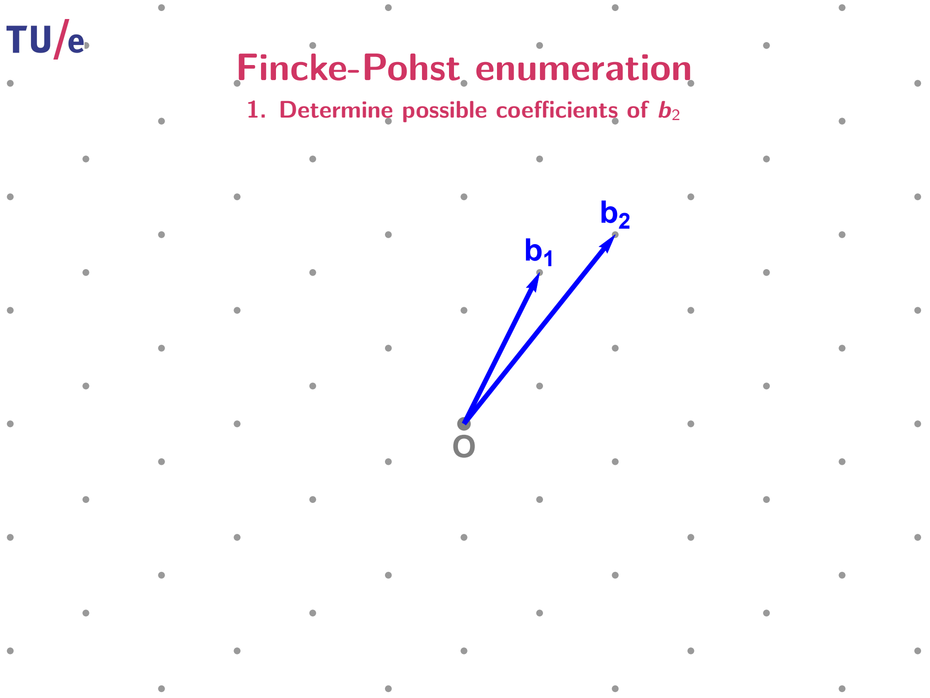
Procedure:

1. Determine possible coefficients of $\boldsymbol{b}_n$
2. Find short vectors for each coefficient of $\boldsymbol{b}_n$ (recursively)
3. Find a shortest vector among all found vectors

Recursive: Reduces SVP_n (CVP_n) to several instances of CVP_{n-1}

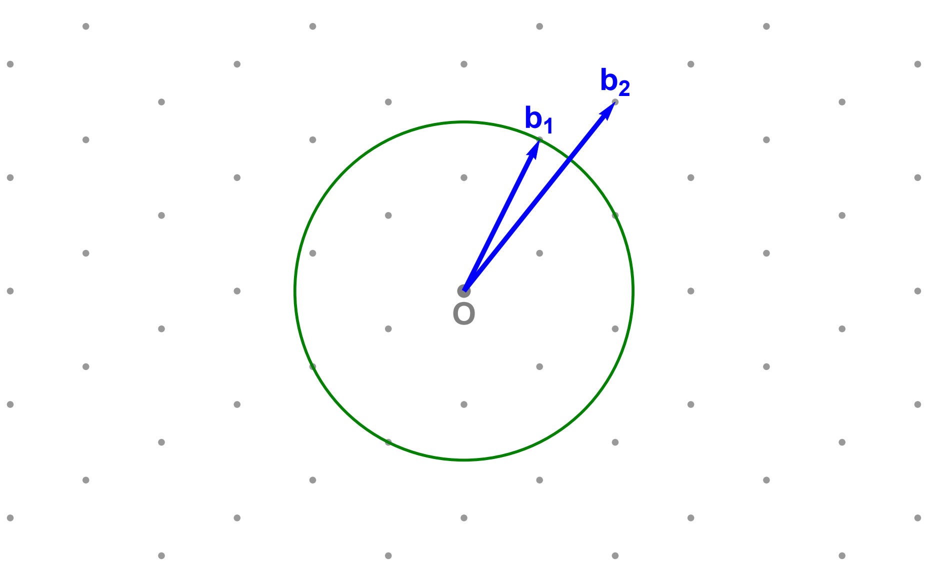
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



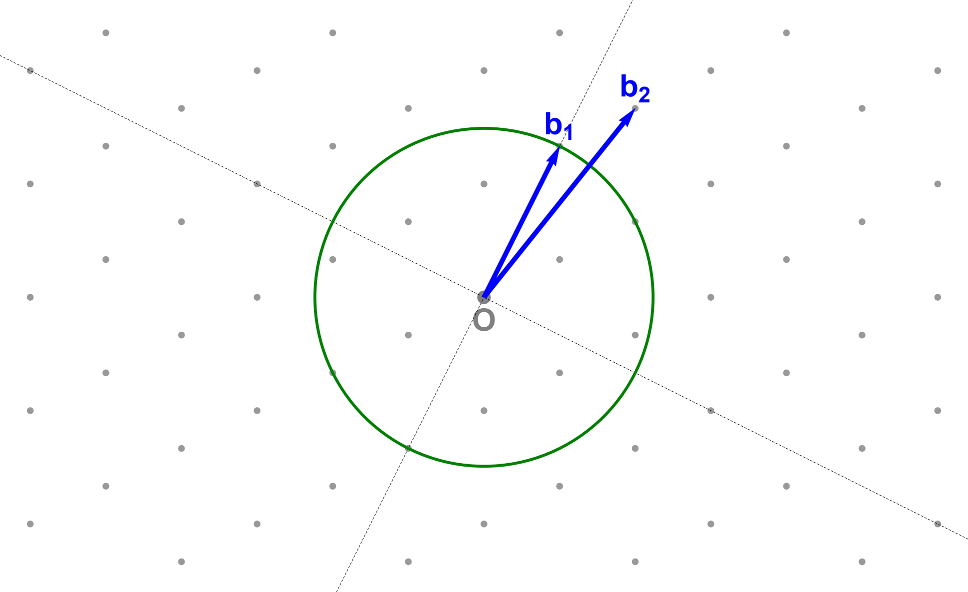
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



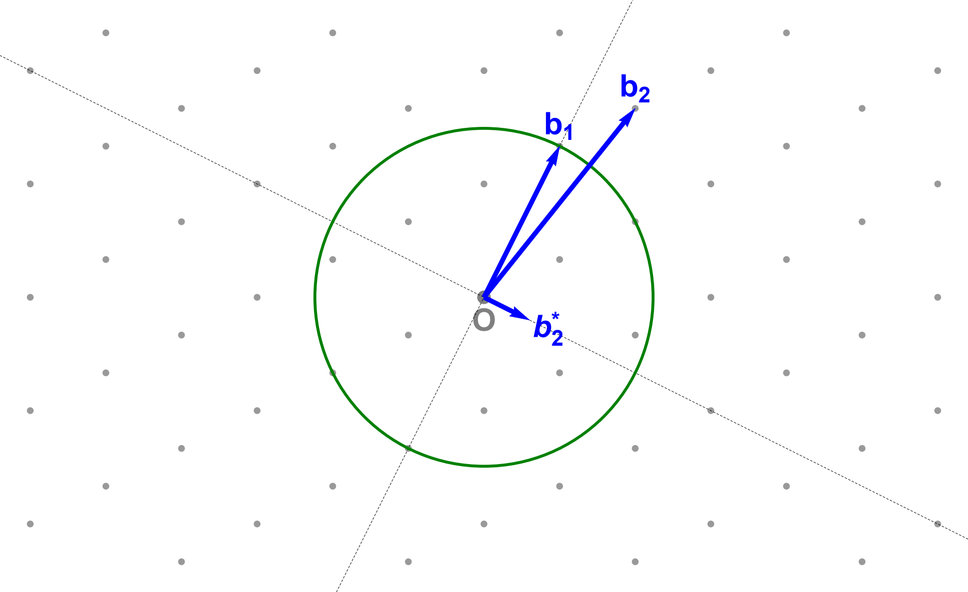
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



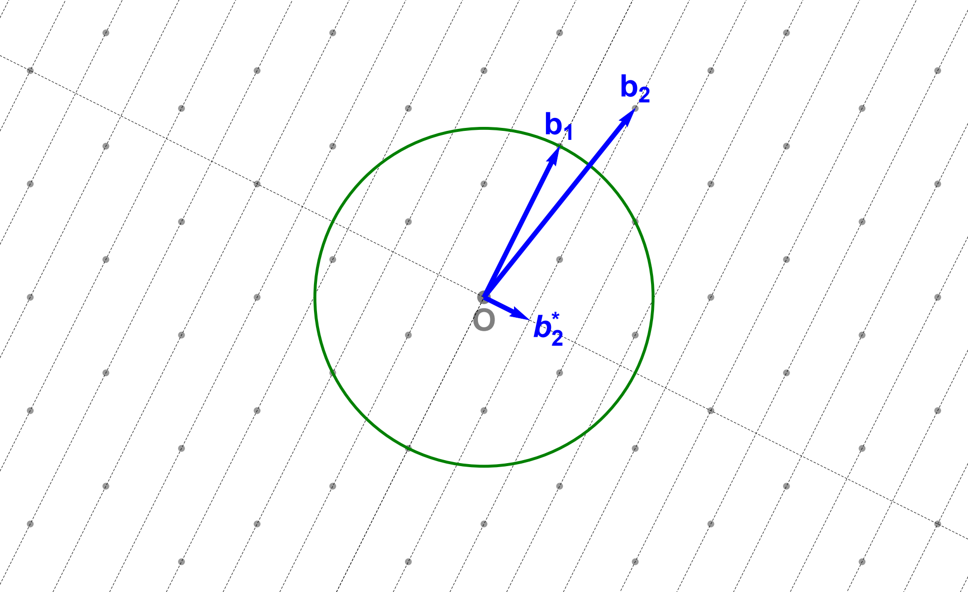
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



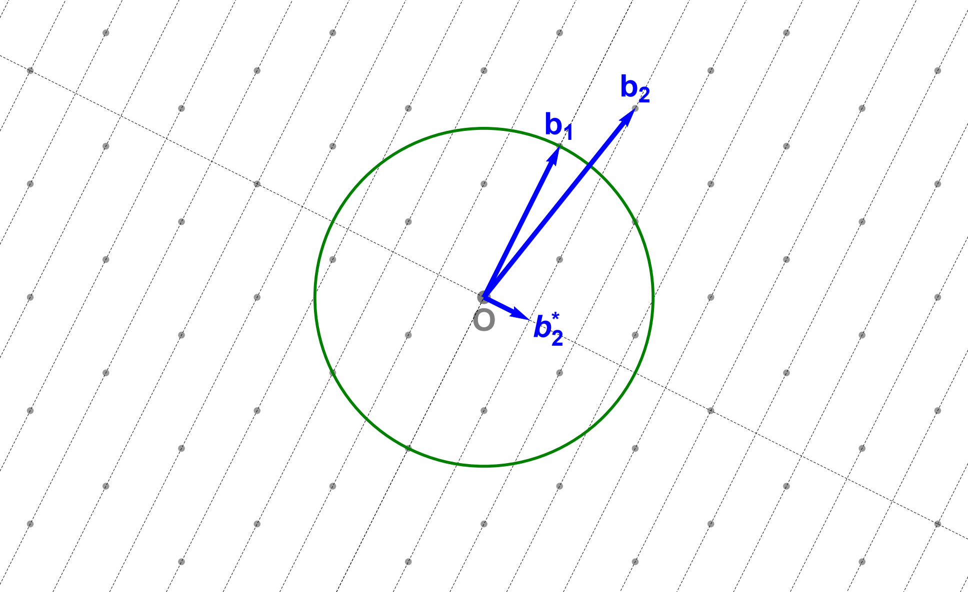
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



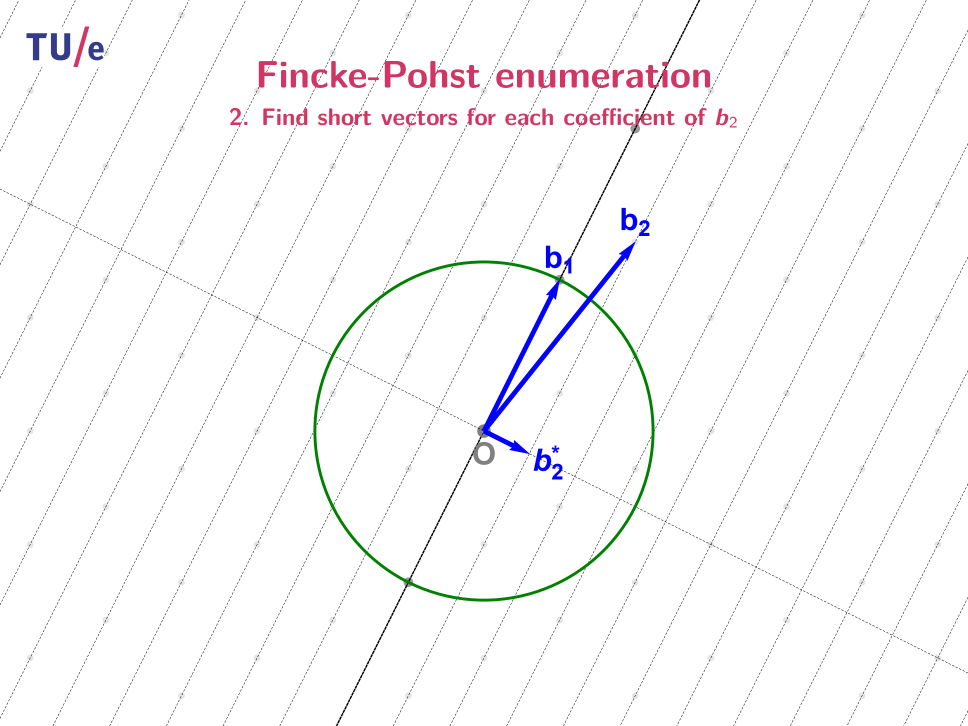
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



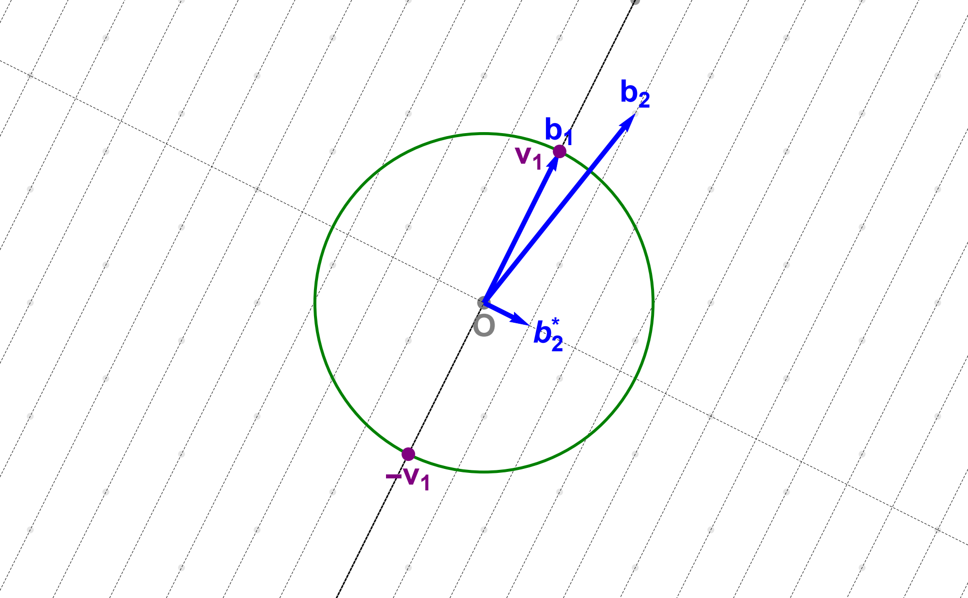
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



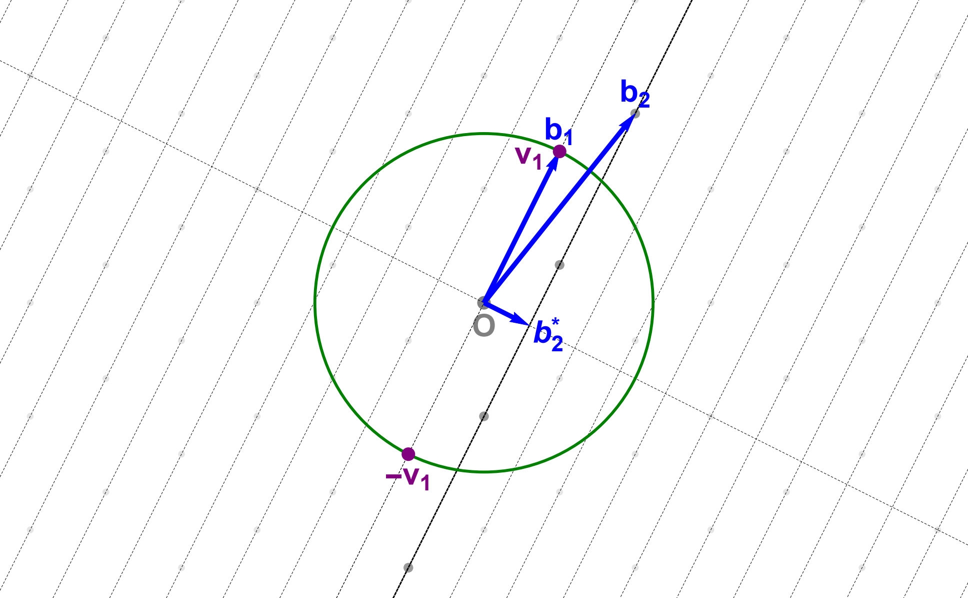
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



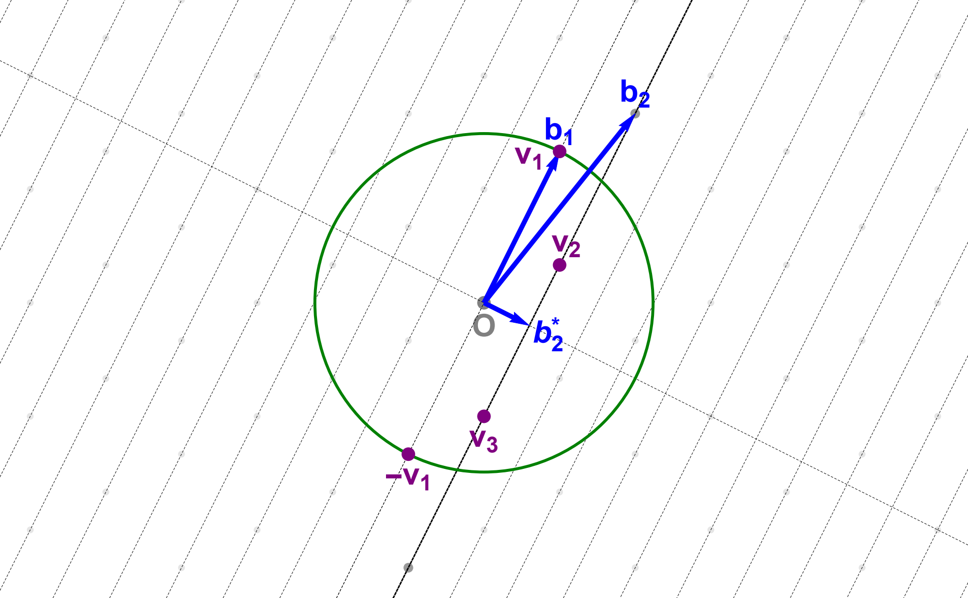
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



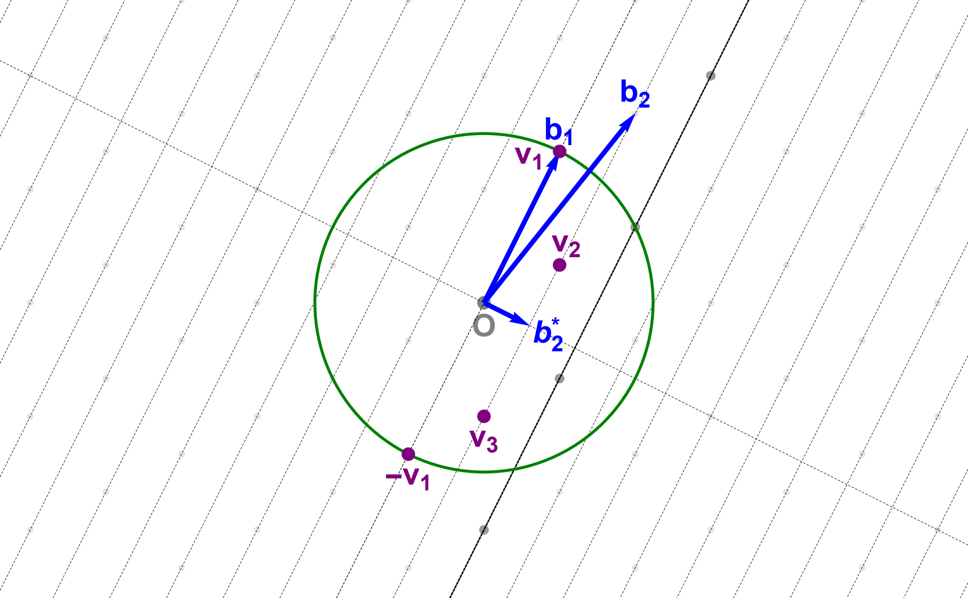
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



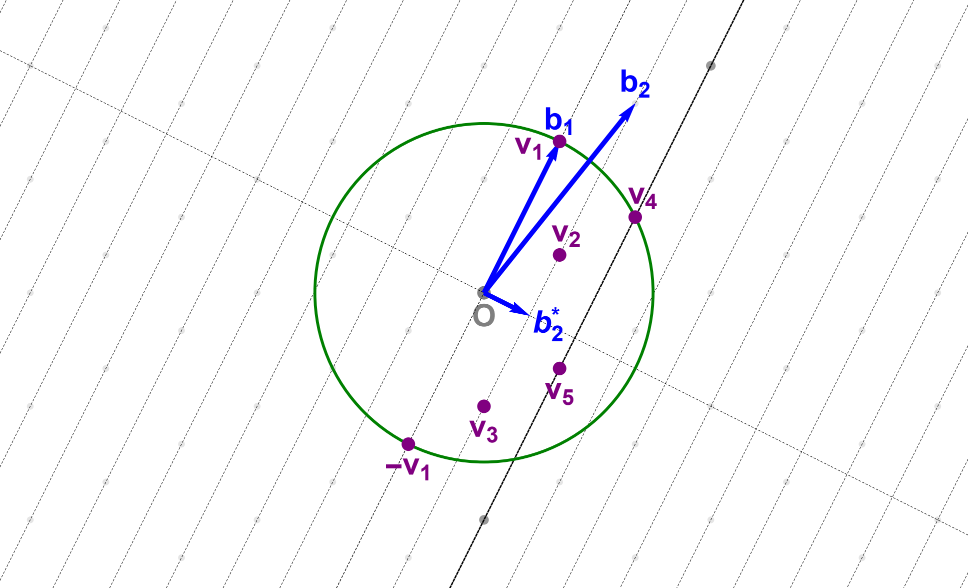
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



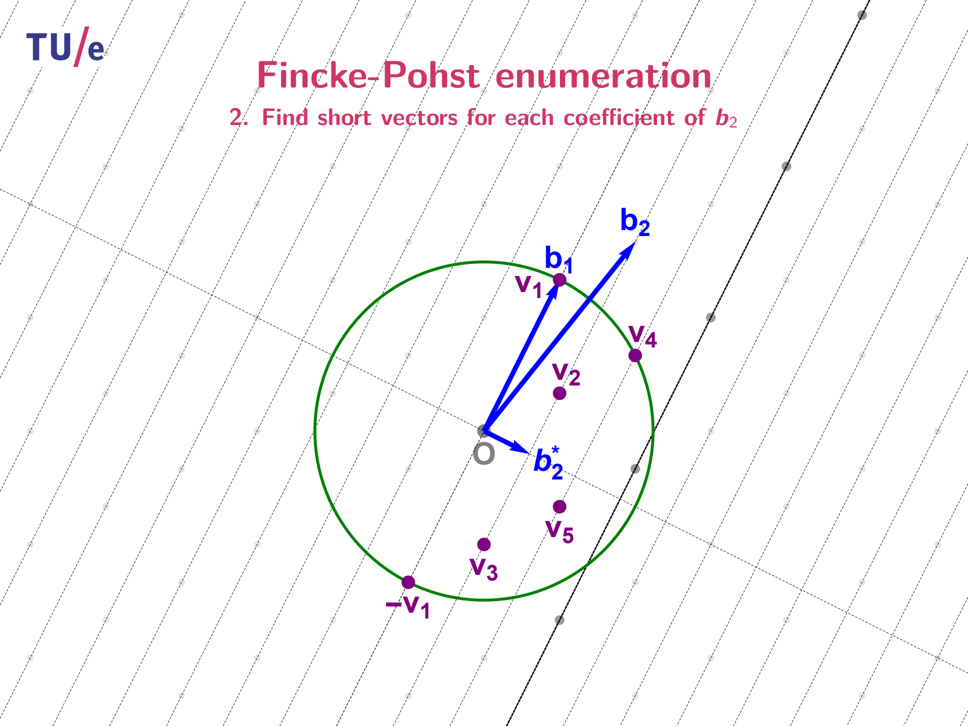
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



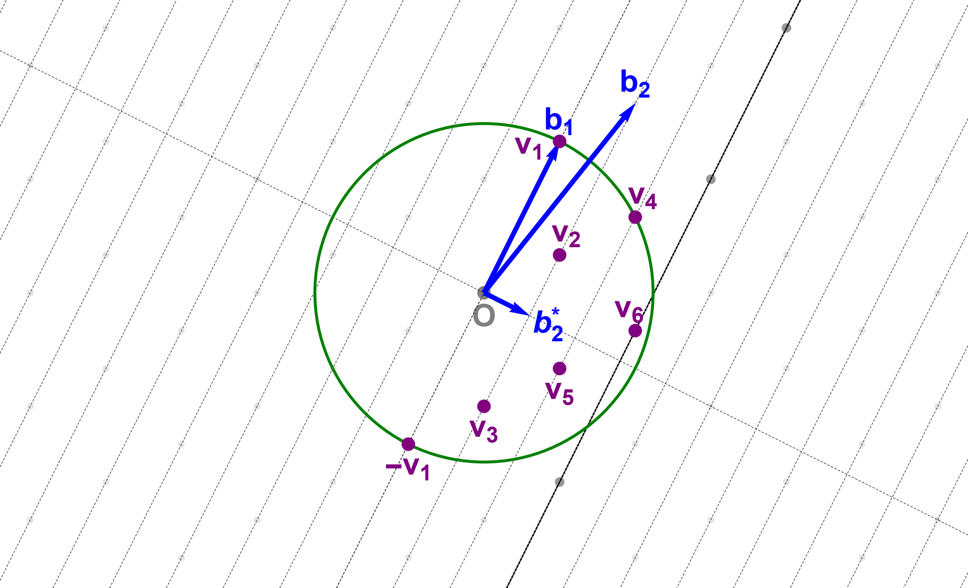
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



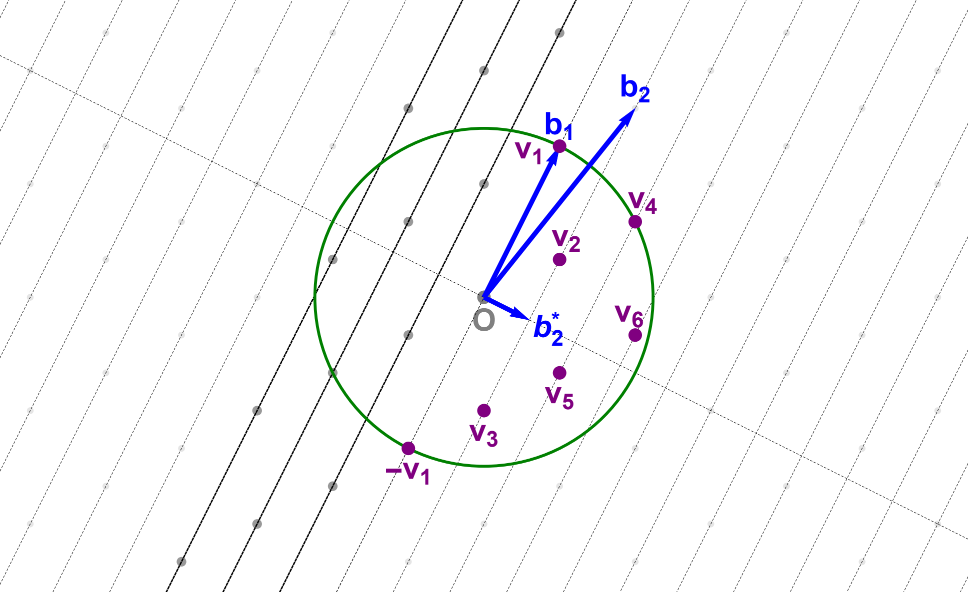
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



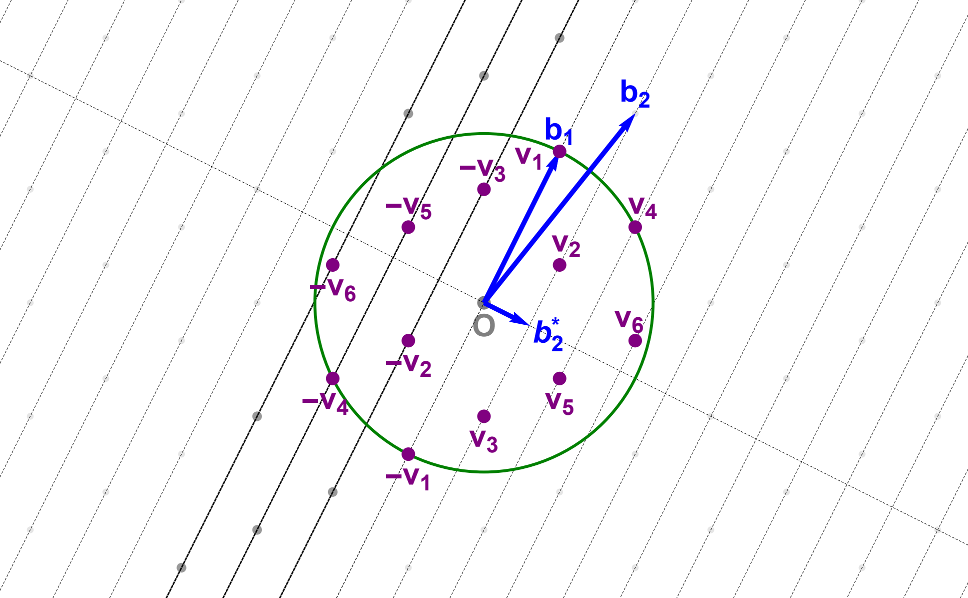
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



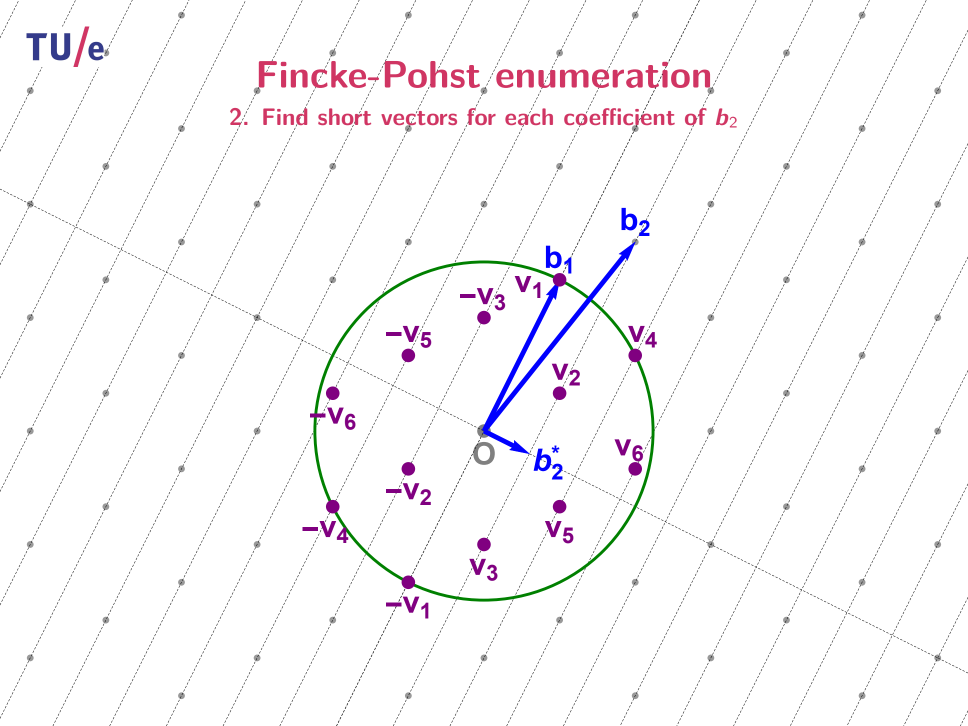
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



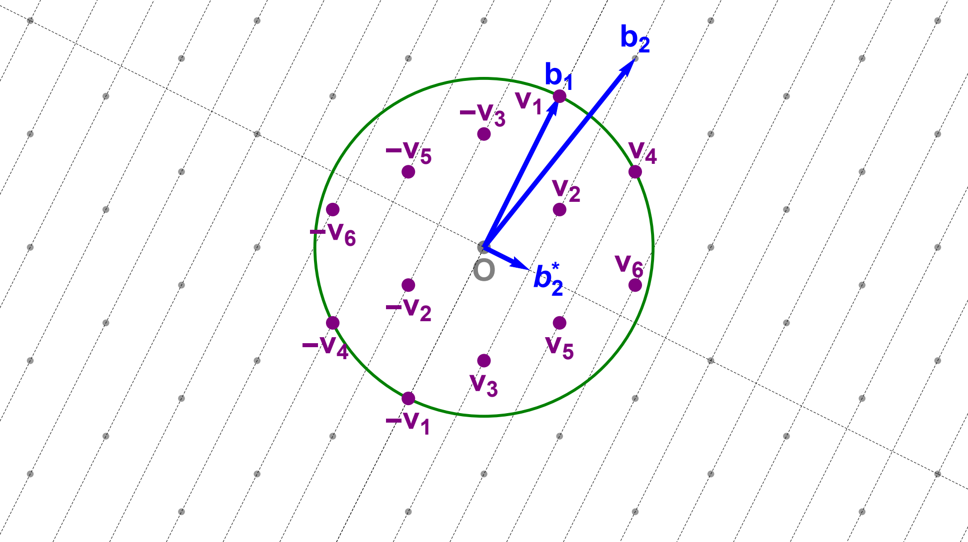
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



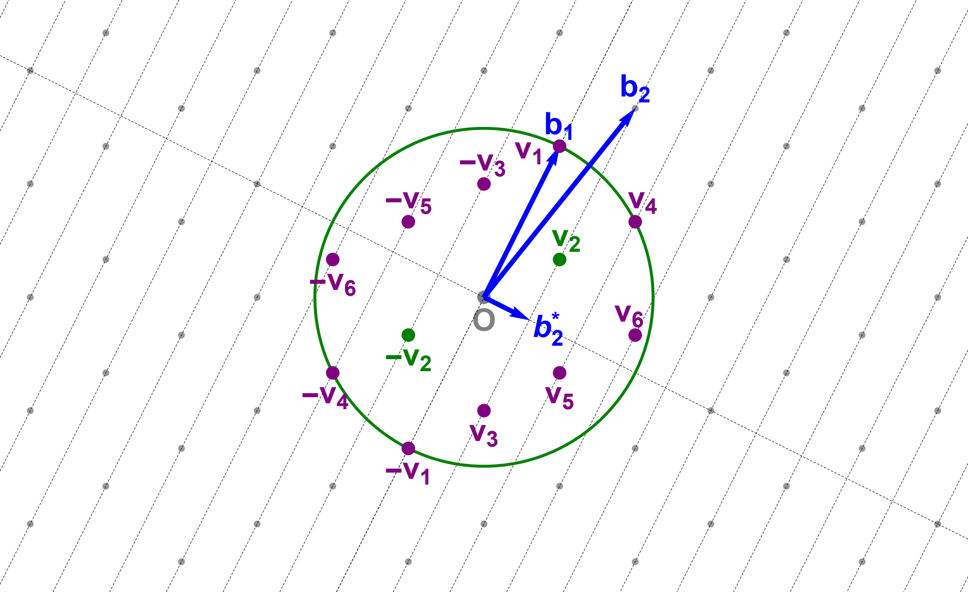
Fincke-Pohst enumeration

3. Find a shortest vector among all found vectors



Fincke-Pohst enumeration

3. Find a shortest vector among all found vectors

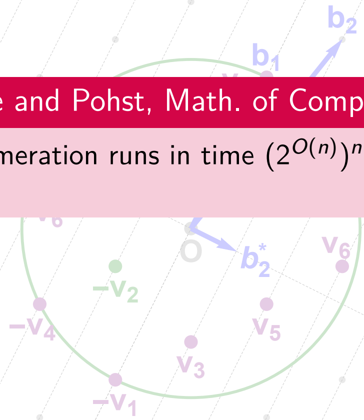


Fincke-Pohst enumeration

Overview

Fincke-Pohst enumeration

Overview



Theorem (Fincke and Pohst, Math. of Comp. '85)

Fincke-Pohst enumeration runs in time $(2^{O(n)})^n = 2^{O(n^2)}$ and space poly(n).

Fincke-Pohst enumeration

Overview

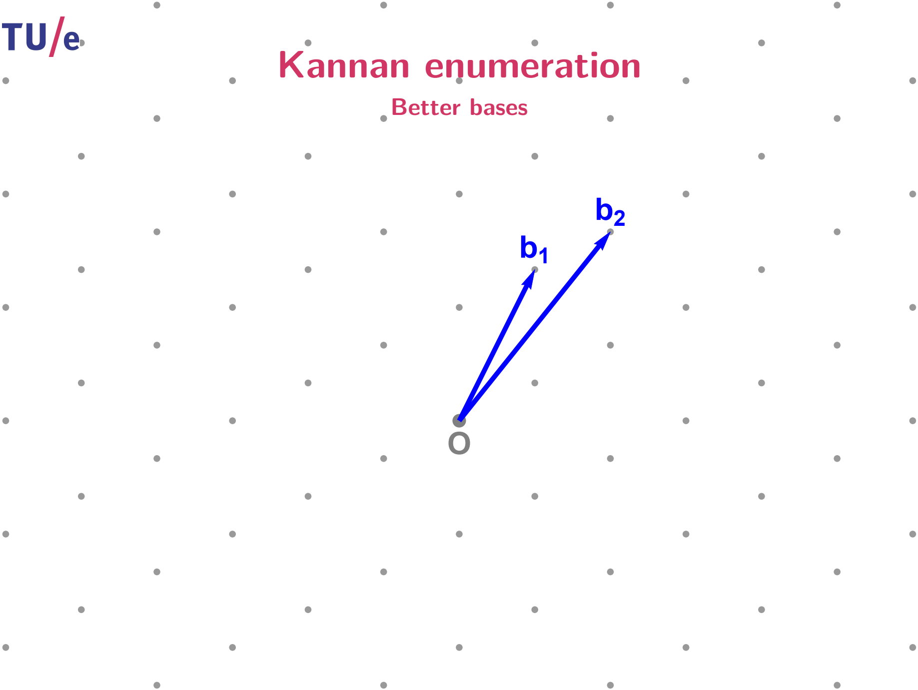
Theorem (Fincke and Pohst, Math. of Comp. '85)

Fincke-Pohst enumeration runs in time $(2^{O(n)})^n = 2^{O(n^2)}$ and space poly(n).

Essentially reduces SVP_n (CVP_n) to $2^{O(n)}$ instances of CVP_{n-1}

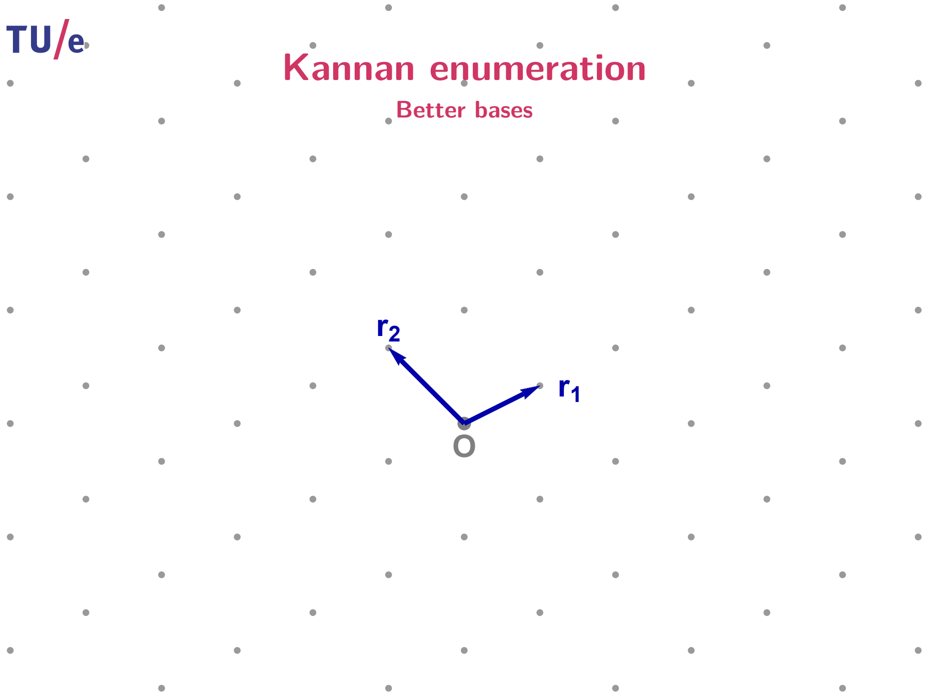
Kannan enumeration

Better bases



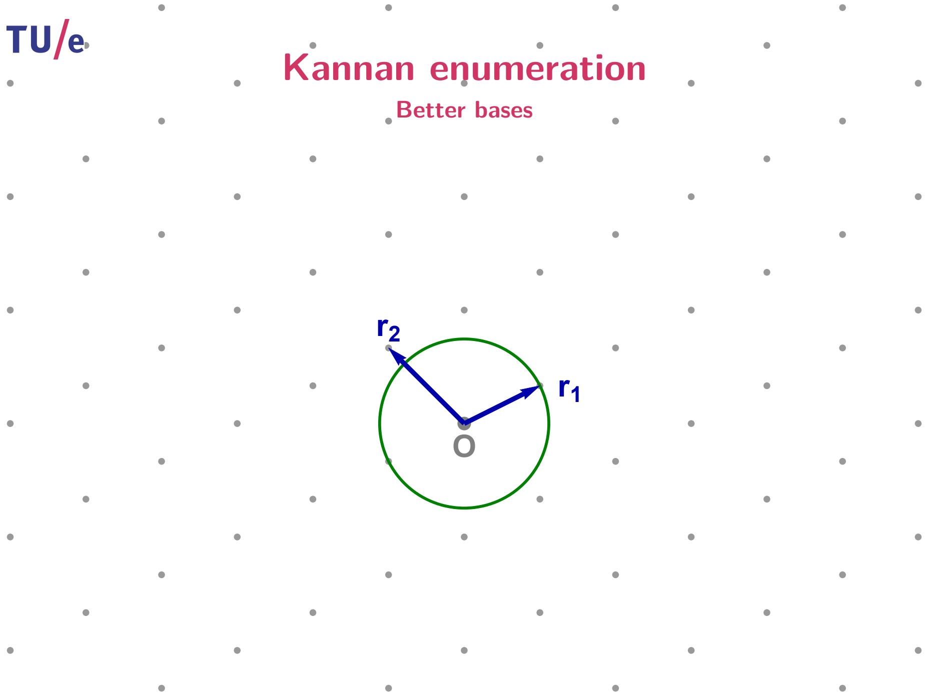
Kannan enumeration

Better bases



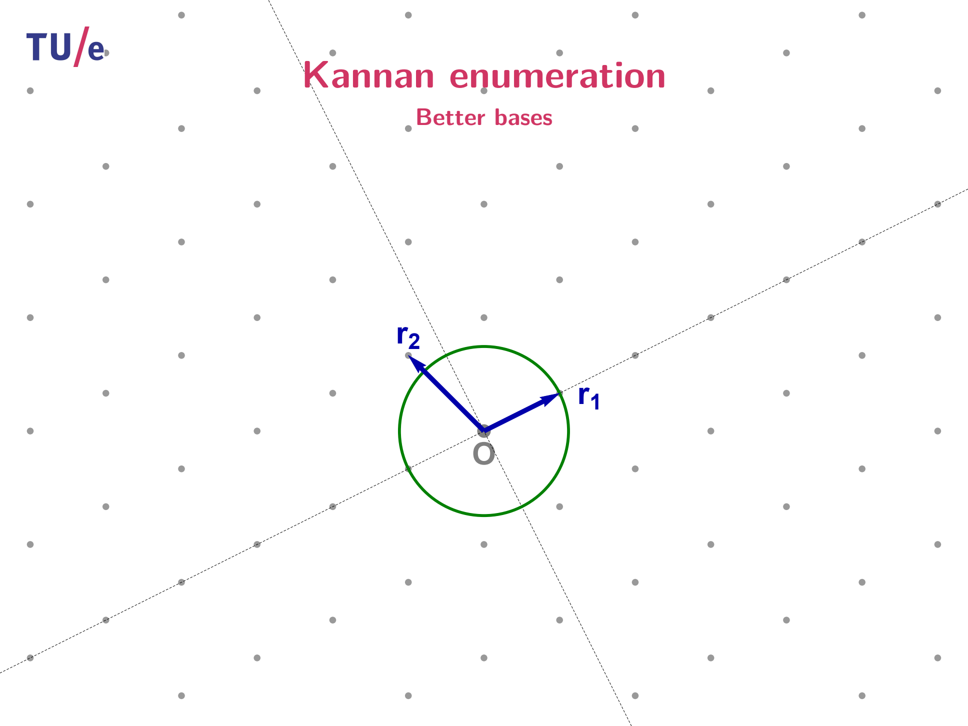
Kannan enumeration

Better bases



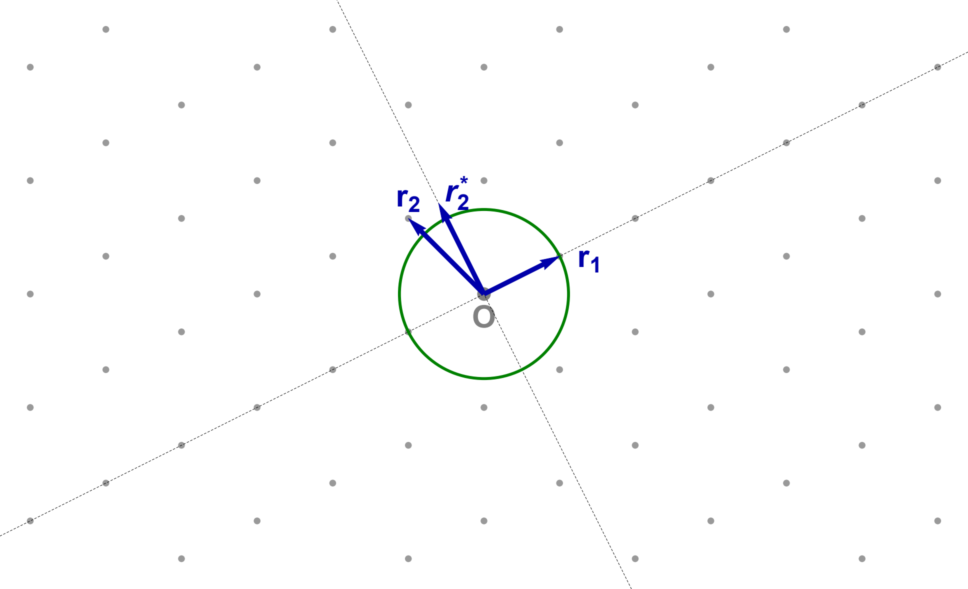
Kannan enumeration

Better bases



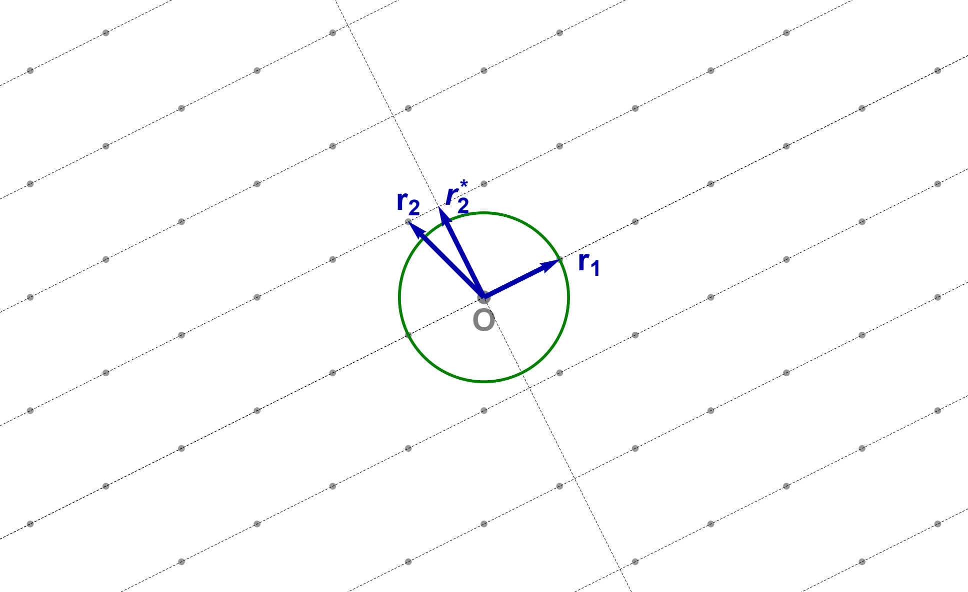
Kannan enumeration

Better bases



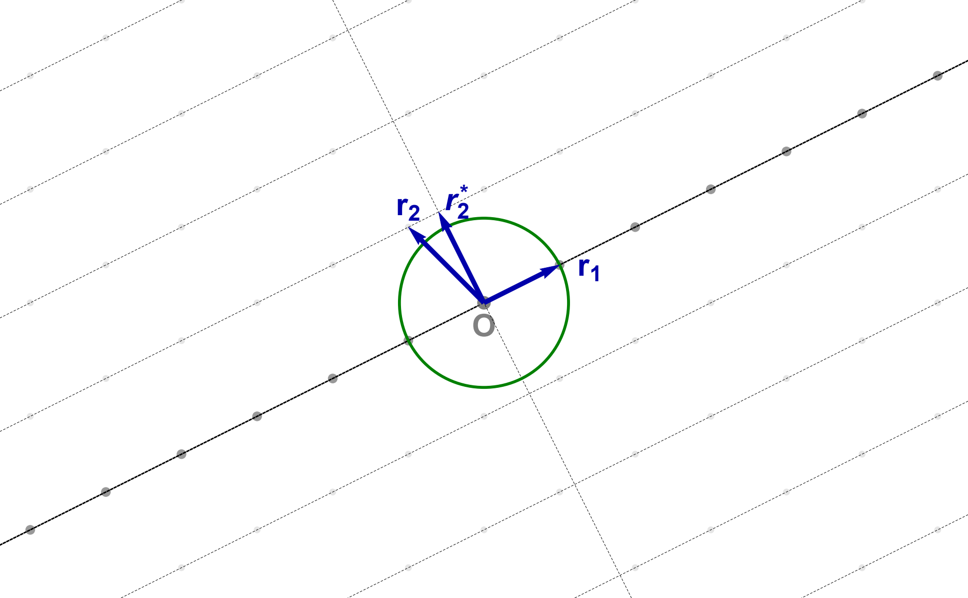
Kannan enumeration

Better bases



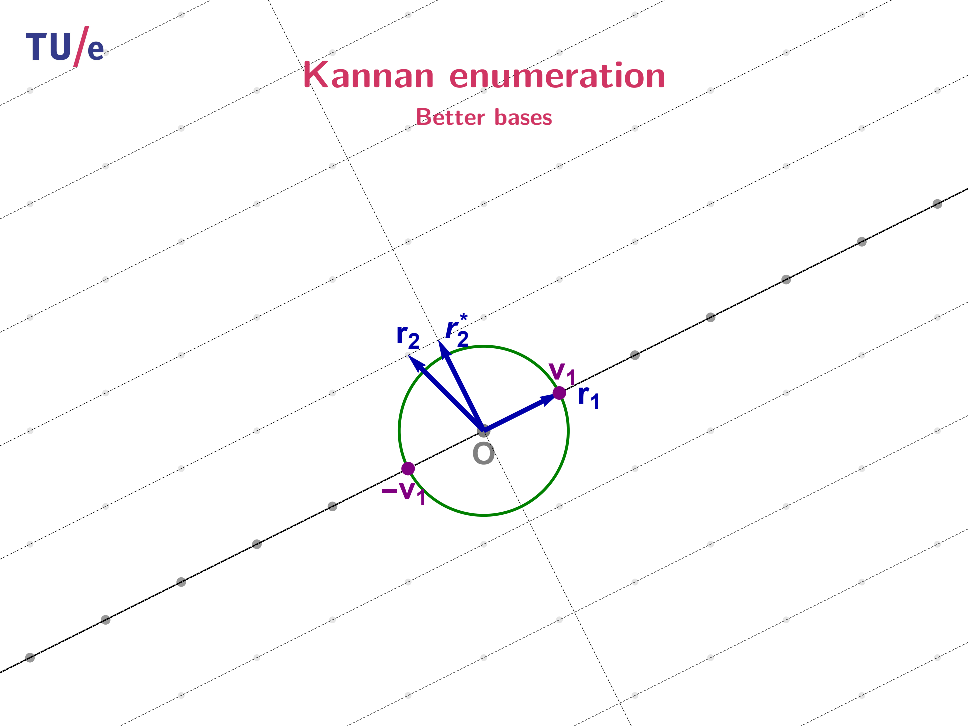
Kannan enumeration

Better bases



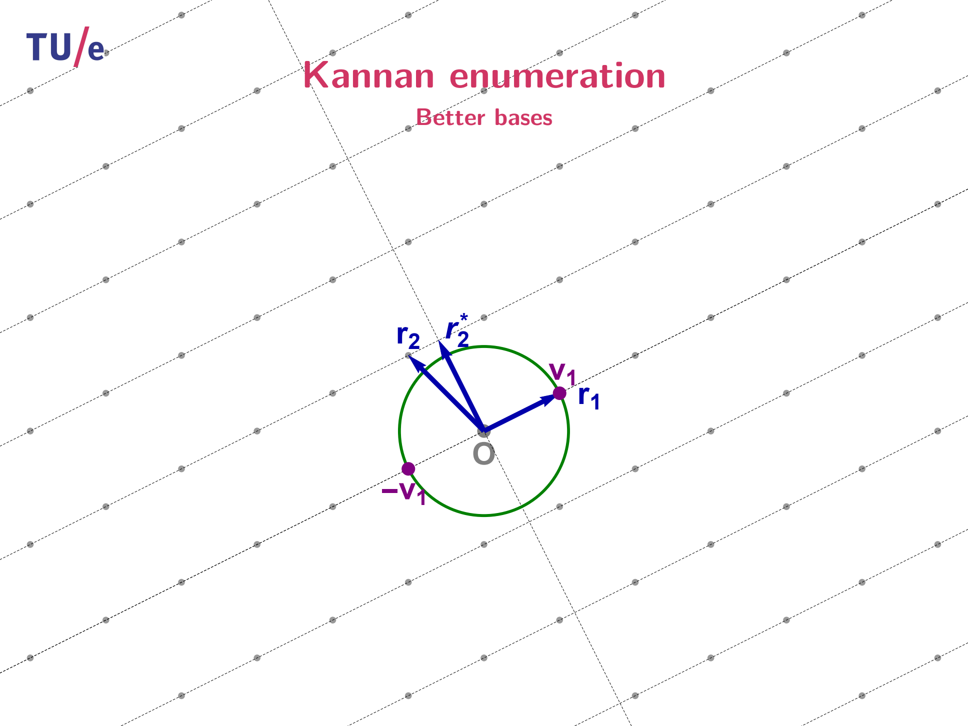
Kannan enumeration

Better bases



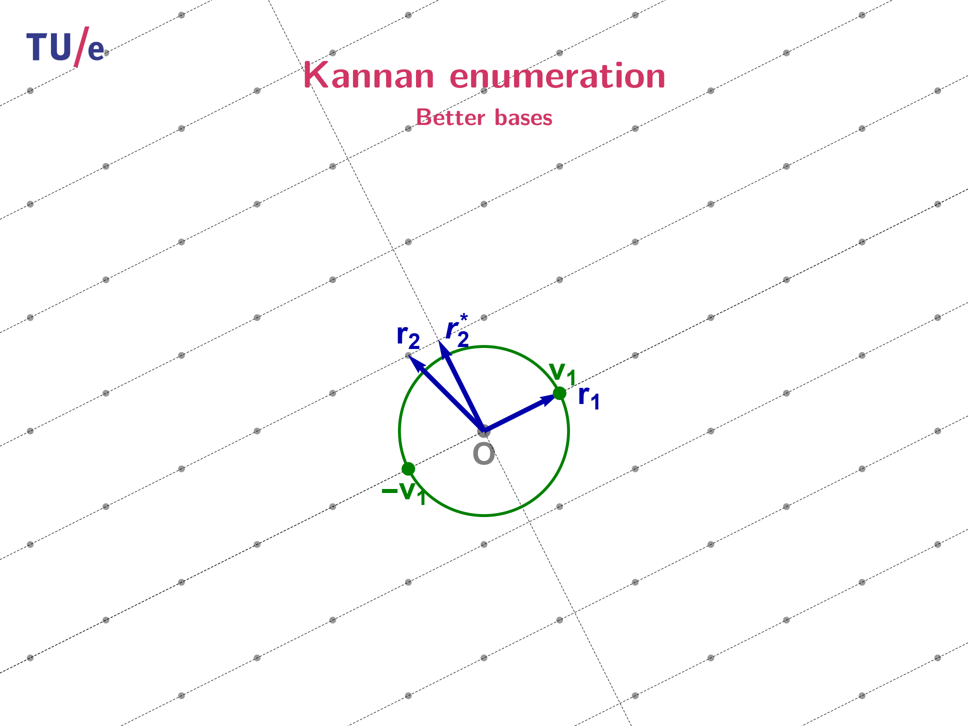
Kannan enumeration

Better bases



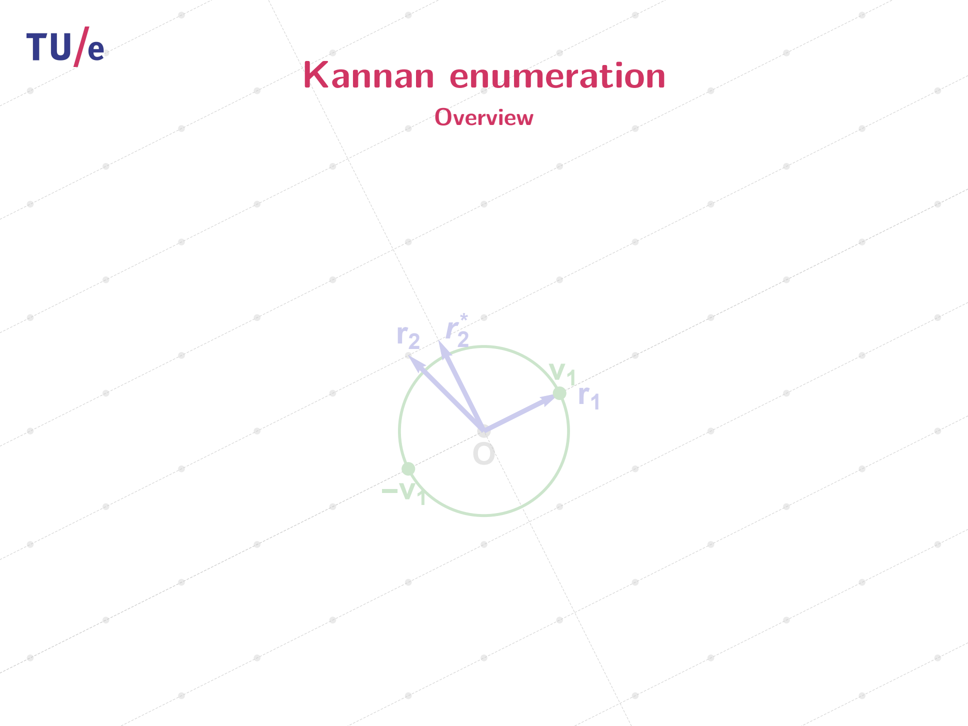
Kannan enumeration

Better bases



Kannan enumeration

Overview



Kannan enumeration

Overview

Theorem (Kannan, STOC'83)

Kannan enumeration runs in time $2^{O(n \log n)}$ and space $\mathrm{poly}(n)$.

Kannan enumeration

Overview

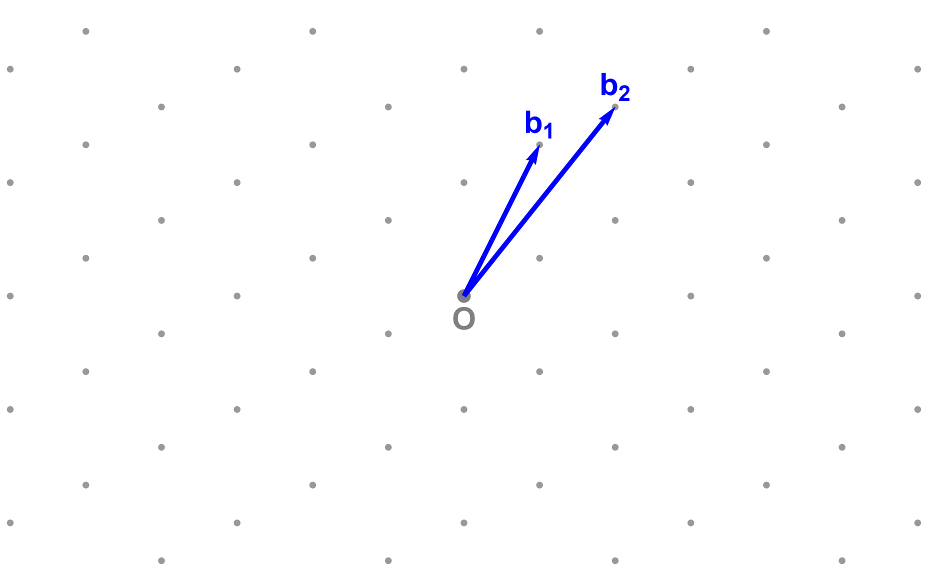
Theorem (Kannan, STOC'83)

Kannan enumeration runs in time $2^{O(n \log n)}$ and space poly(n).

> *"Our algorithm reduces an n-dimensional problem to polynomially many (instead of $2^{O(n)}$) $(n-1)$-dimensional problems. [...] The algorithm we propose, first finds a more orthogonal basis for a lattice in time $2^{O(n \log n)}$."*
>
> – Kannan, *STOC'83*

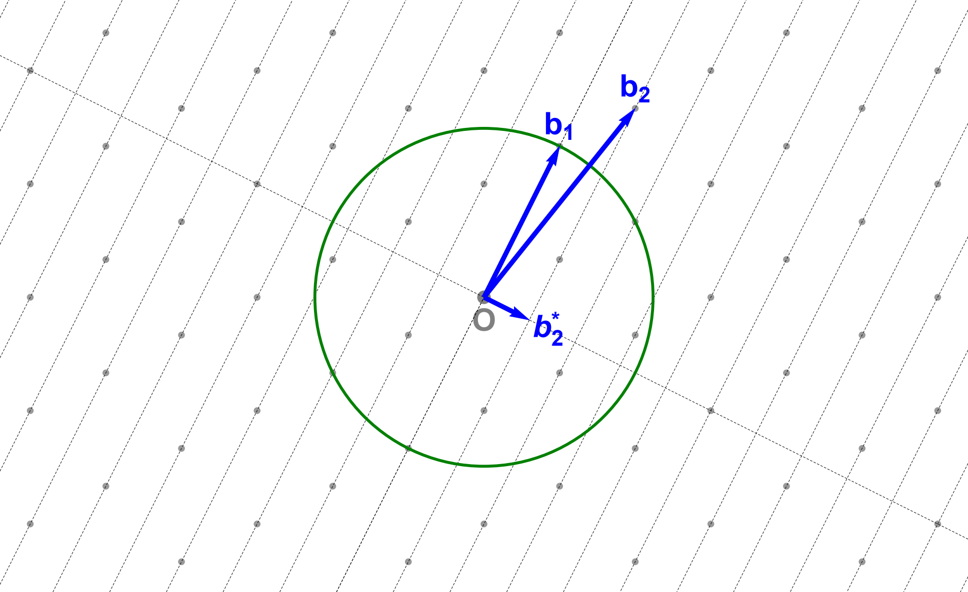
Pruned enumeration

Reducing the search space



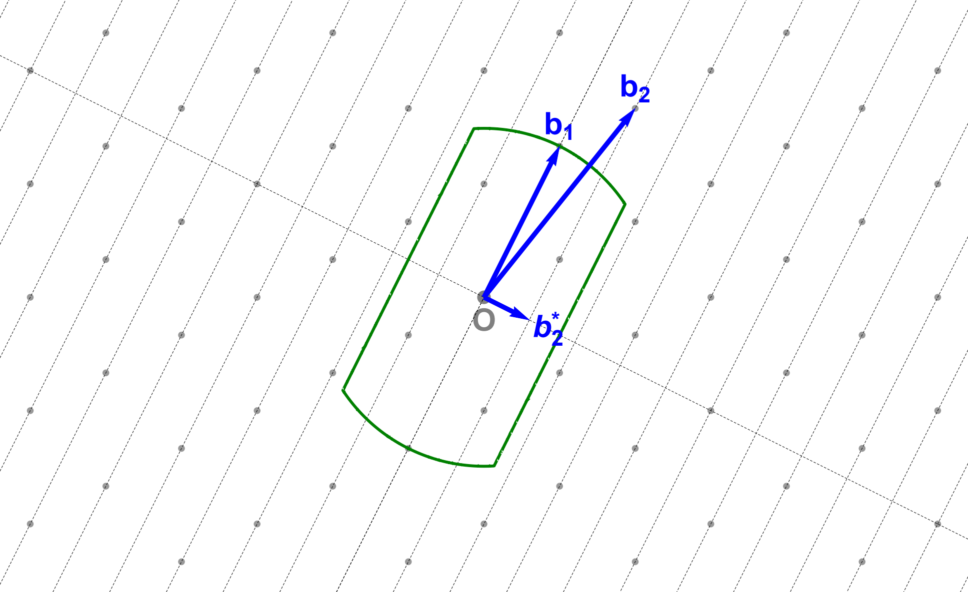
Pruned enumeration

Reducing the search space



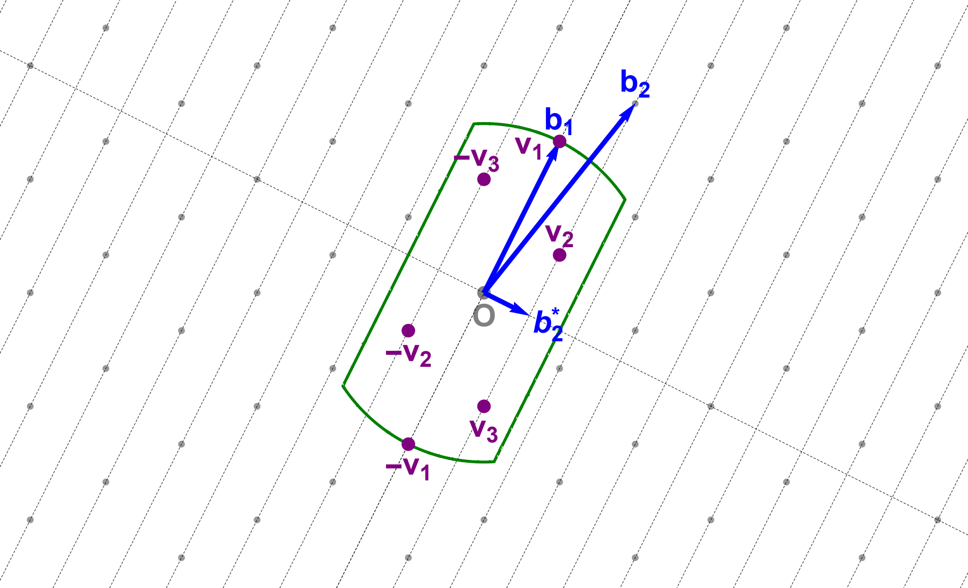
Pruned enumeration

Reducing the search space



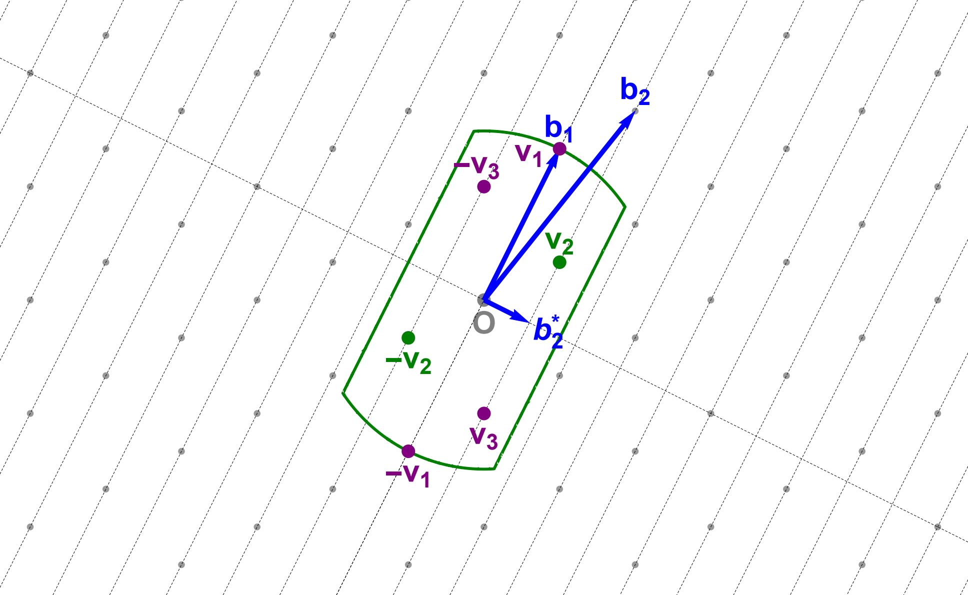
Pruned enumeration

Reducing the search space



Pruned enumeration

Reducing the search space



Pruned enumeration

Overview

Pruned enumeration

Overview

"Well-chosen bounding functions lead asymptotically to an exponential speedup of about $2^{n/4}$ over basic enumeration, maintaining a success probability $\geq$ 95%."

– Gama et al., *EUROCRYPT'10*

Pruned enumeration

Overview

"Well-chosen bounding functions lead asymptotically to an exponential speedup of about $2^{n/4}$ *over basic enumeration, maintaining a success probability* $\geq 95\%$*."*

– Gama et al., *EUROCRYPT'10*

"With extreme pruning, the probability of finding the desired vector is actually rather low (say, 0.1%*), but surprisingly, the running time of the enumeration is reduced by a much more significant factor (say, much more than* 1000*)."*

– Gama et al., *EUROCRYPT'10*

Outline

Lattices

Enumeration algorithms
- Fincke-Pohst enumeration
- Kannan enumeration
- Pruning the enumeration tree

The Voronoi cell algorithm

Sieving algorithms
- Nguyen-Vidick sieve
- Multiple levels
- GaussSieve

Sieving using angular locality-sensitive hashing
- Nguyen-Vidick sieve
- GaussSieve

The Voronoi cell algorithm

Very recent algorithm [MV10]

Procedure:

1. Construct the Voronoi cell of the lattice $\mathcal{L}$

The Voronoi cell algorithm

Very recent algorithm [MV10]
Procedure:

1. Construct the Voronoi cell of the lattice $\mathcal{L}$
   - Consider all vectors $\boldsymbol{t} \in \{0, 1\}^n \cdot B$

The Voronoi cell algorithm

Very recent algorithm [MV10]
Procedure:

1. Construct the Voronoi cell of the lattice $\mathcal{L}$
   - Consider all vectors $\boldsymbol{t} \in \{0,1\}^n \cdot B$
   - For each $\boldsymbol{t}$, find the closest vector $\boldsymbol{v} \in 2\mathcal{L}$ using the $(n-1)$-dimensional Voronoi cell (recursively)

The Voronoi cell algorithm

Very recent algorithm [MV10]

Procedure:

1. Construct the Voronoi cell of the lattice $\mathcal{L}$
   - Consider all vectors $\boldsymbol{t} \in \{0,1\}^n \cdot B$
   - For each $\boldsymbol{t}$, find the closest vector $\boldsymbol{v} \in 2\mathcal{L}$ using the $(n-1)$-dimensional Voronoi cell (recursively)
   - Find the Voronoi cell as the intersection of half spaces

The Voronoi cell algorithm

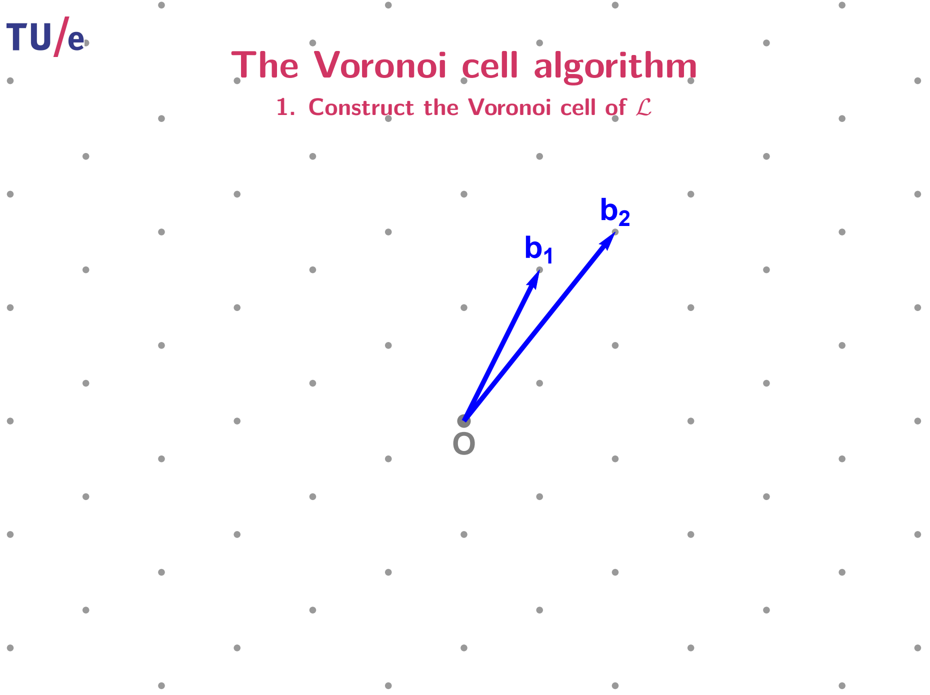
Very recent algorithm [MV10]

Procedure:

1. Construct the Voronoi cell of the lattice $\mathcal{L}$
   - Consider all vectors $\boldsymbol{t} \in \{0, 1\}^n \cdot B$
   - For each $\boldsymbol{t}$, find the closest vector $\boldsymbol{v} \in 2\mathcal{L}$ using the $(n-1)$-dimensional Voronoi cell (recursively)
   - Find the Voronoi cell as the intersection of half spaces
2. Find a shortest vector among the relevant vectors

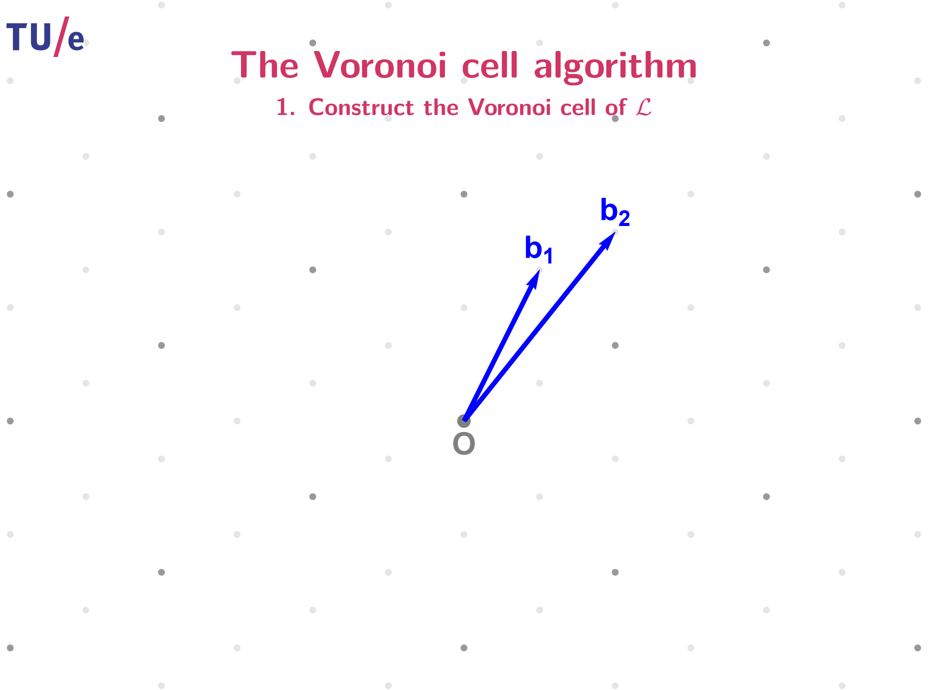
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



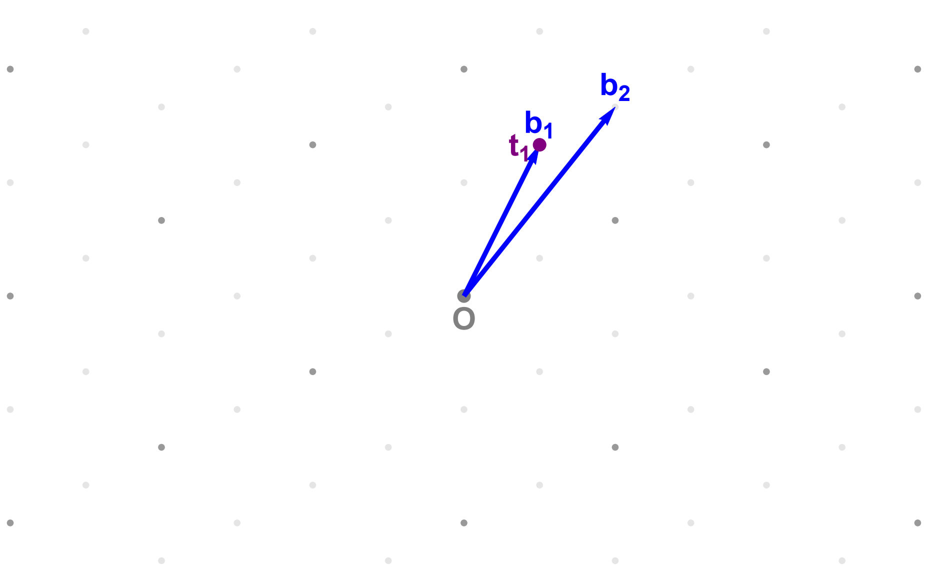
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



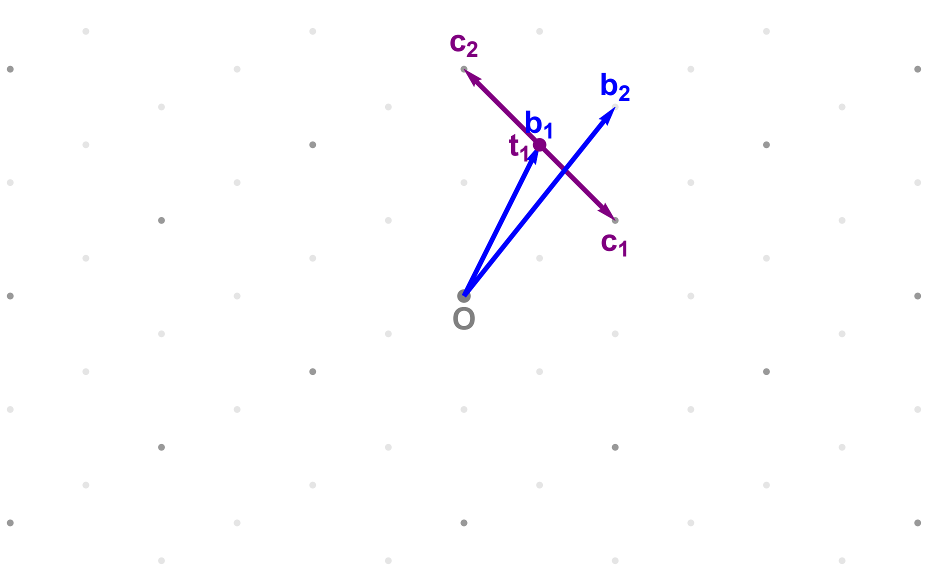
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



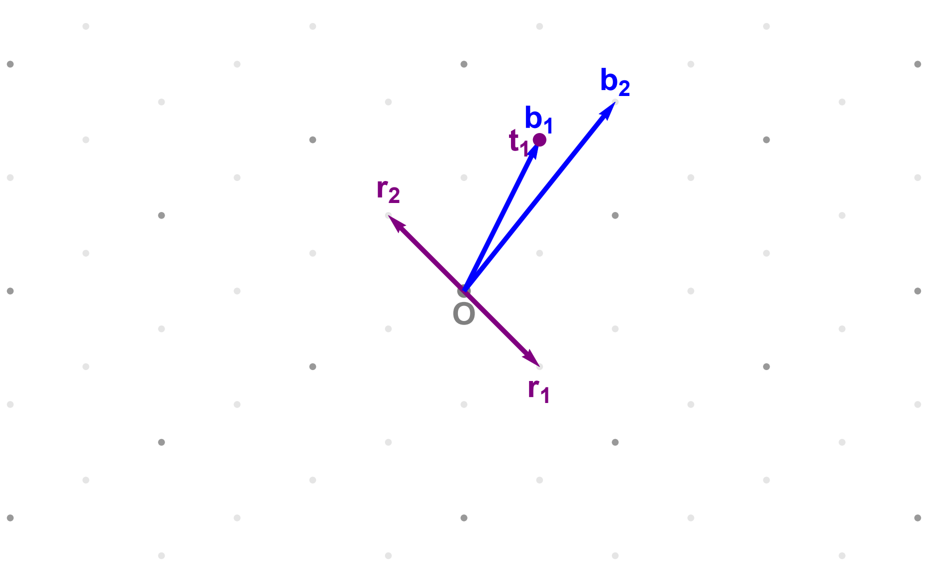
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



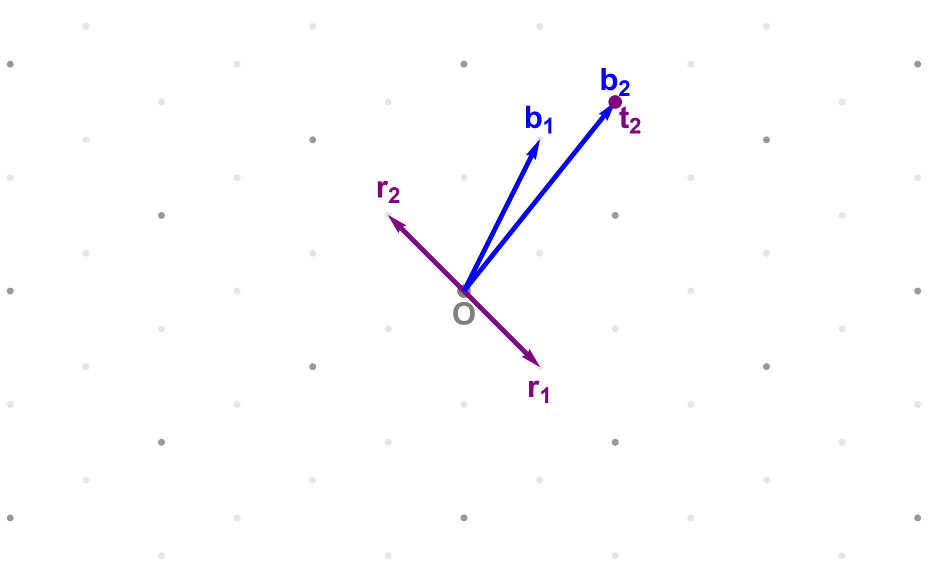
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



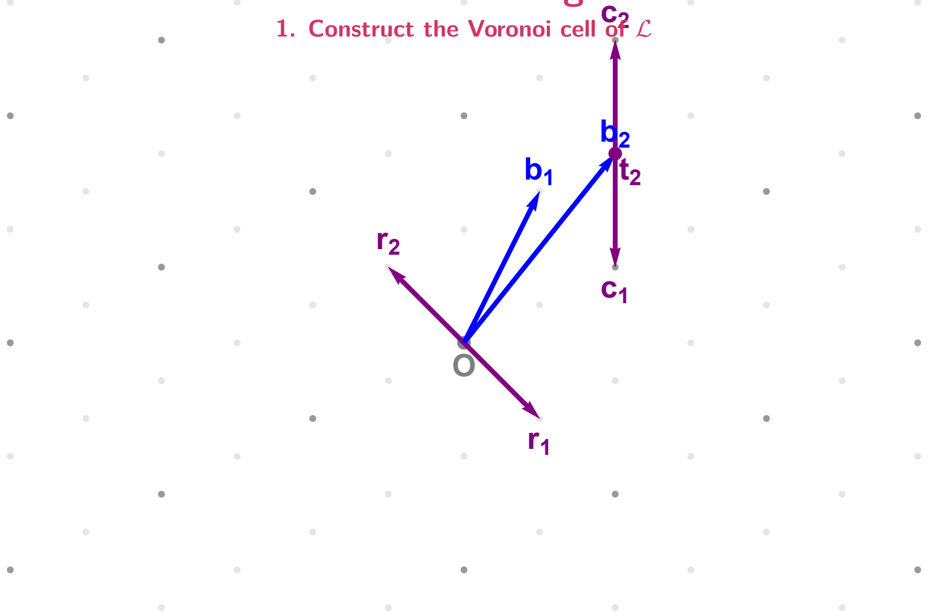
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



The Voronoi cell algorithm

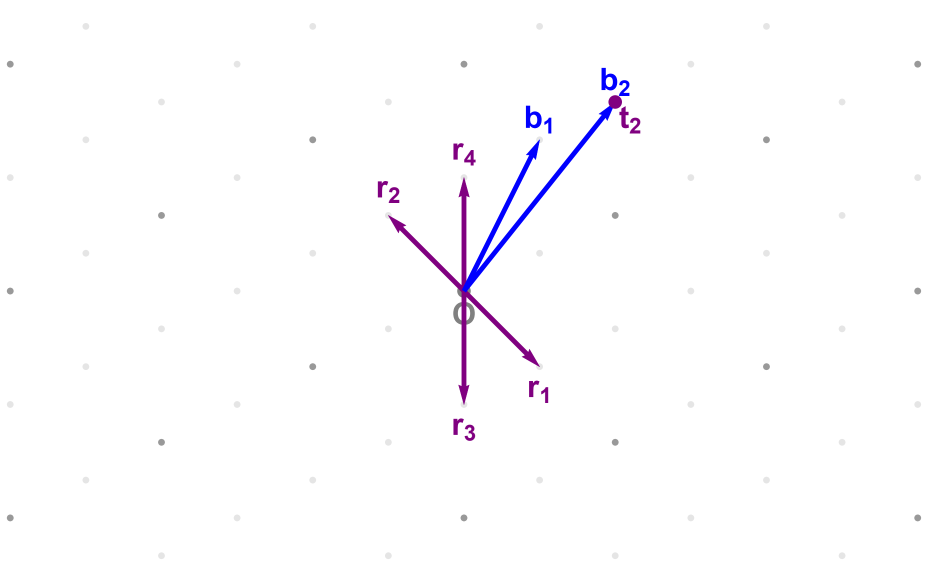
1. Construct the Voronoi cell of $\mathcal{L}$



TU/e

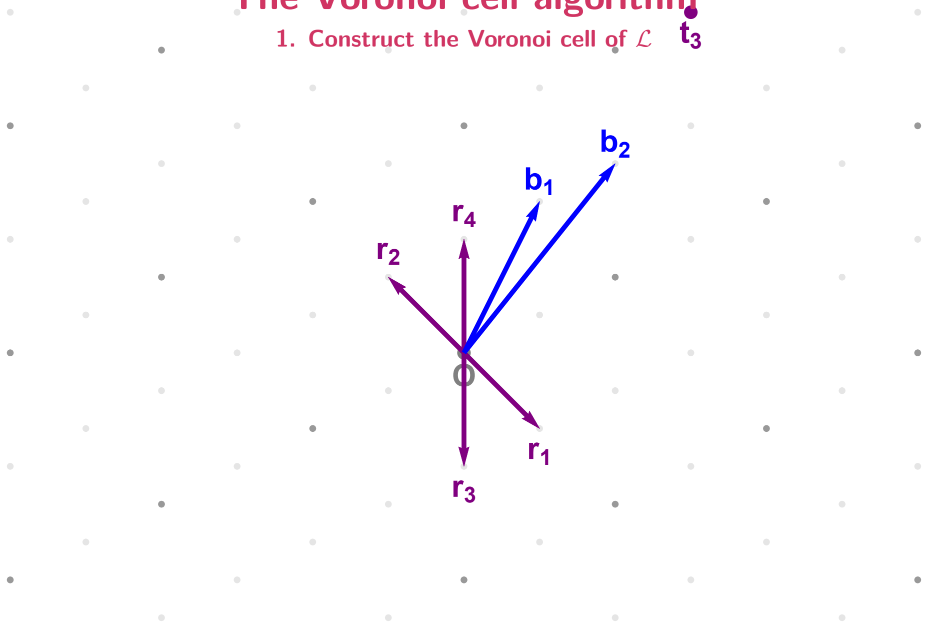
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



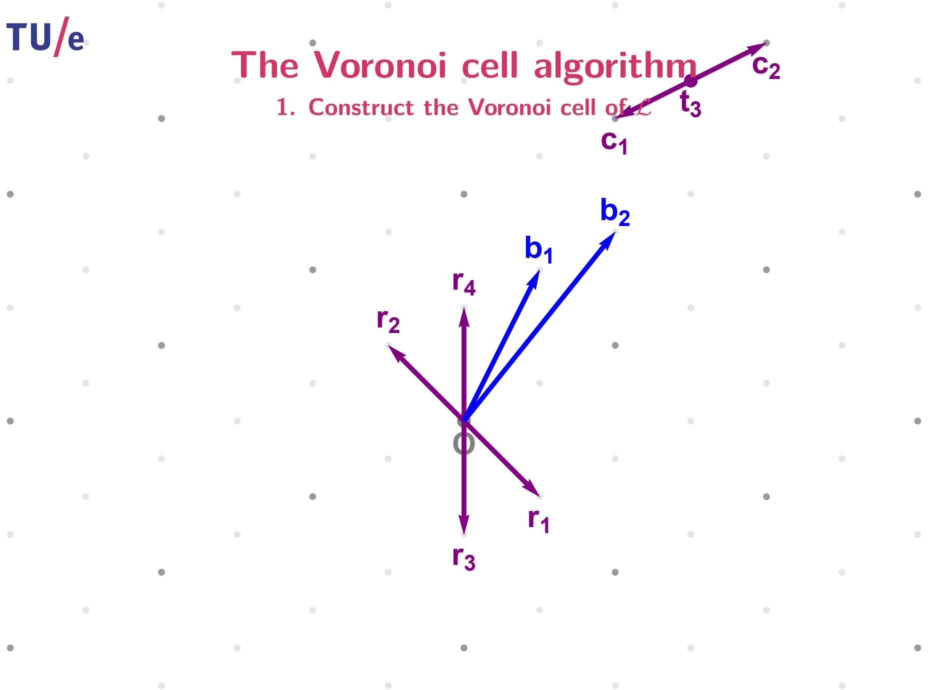
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



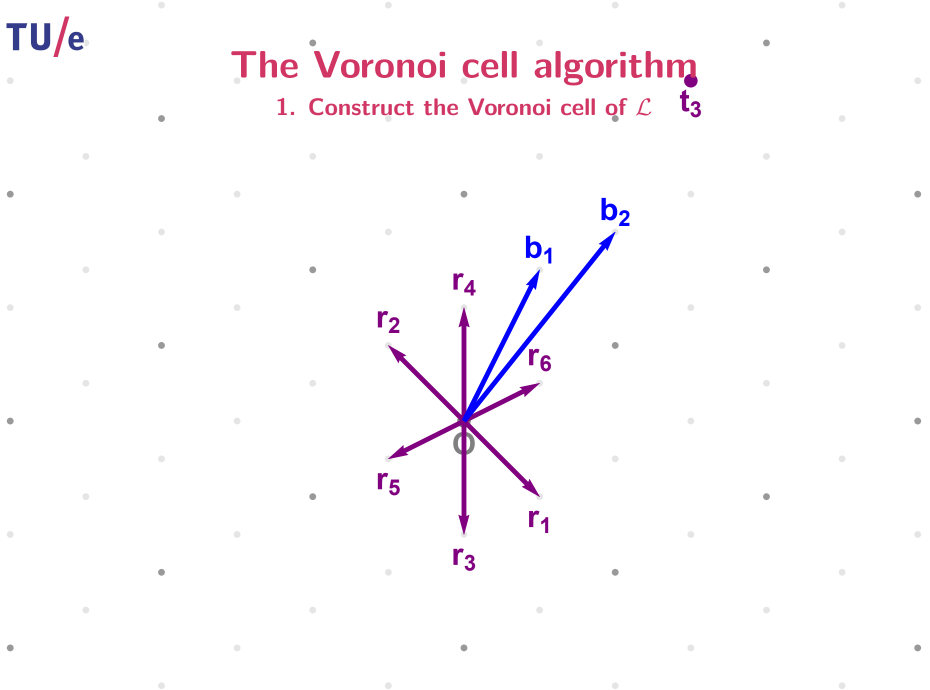
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



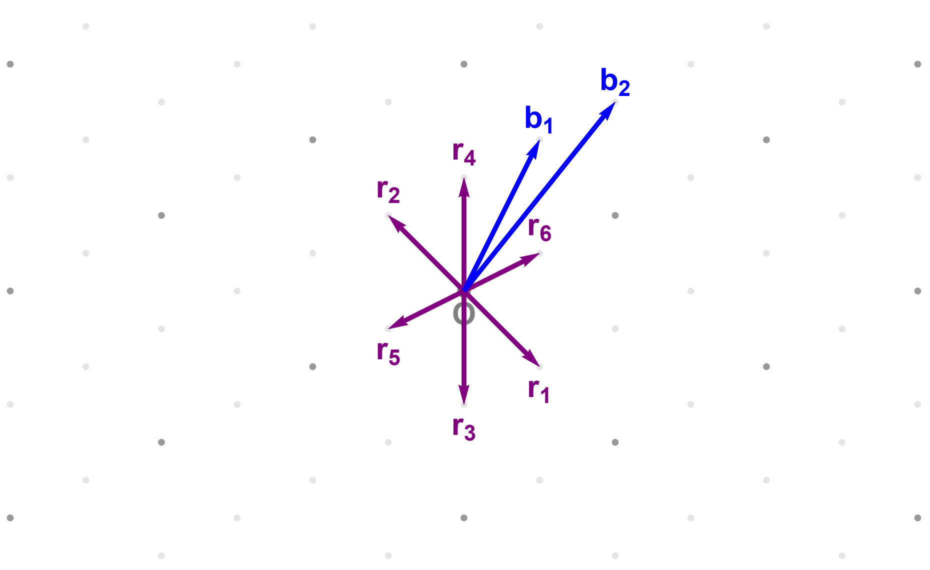
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



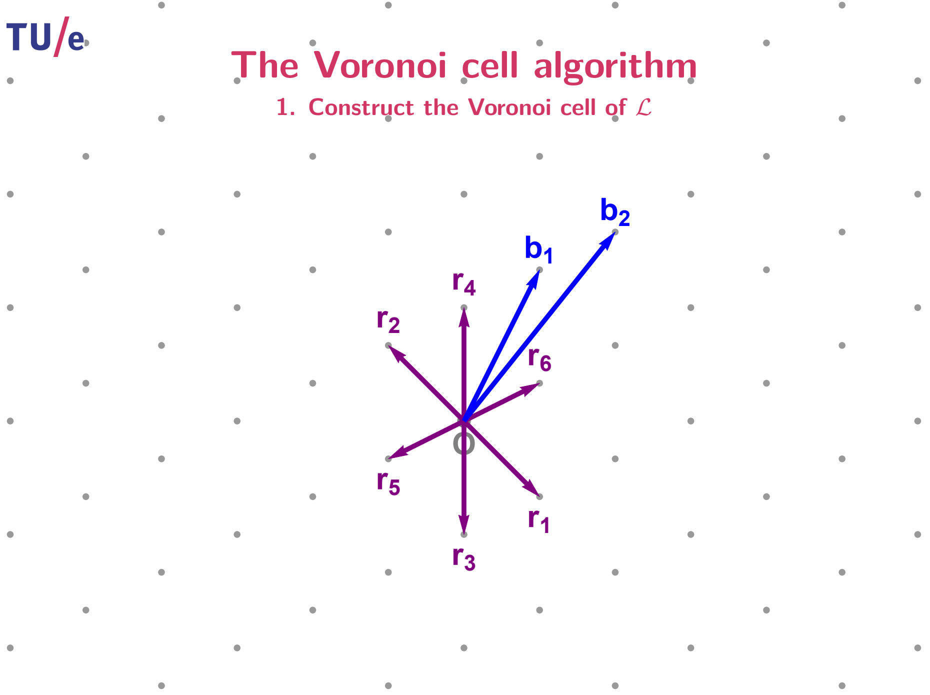
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



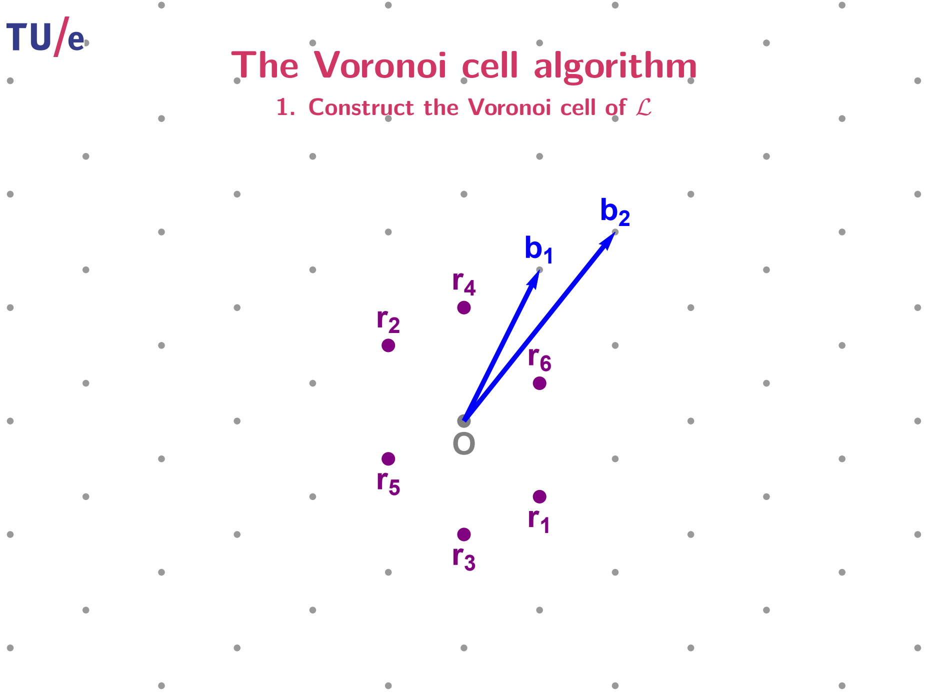
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



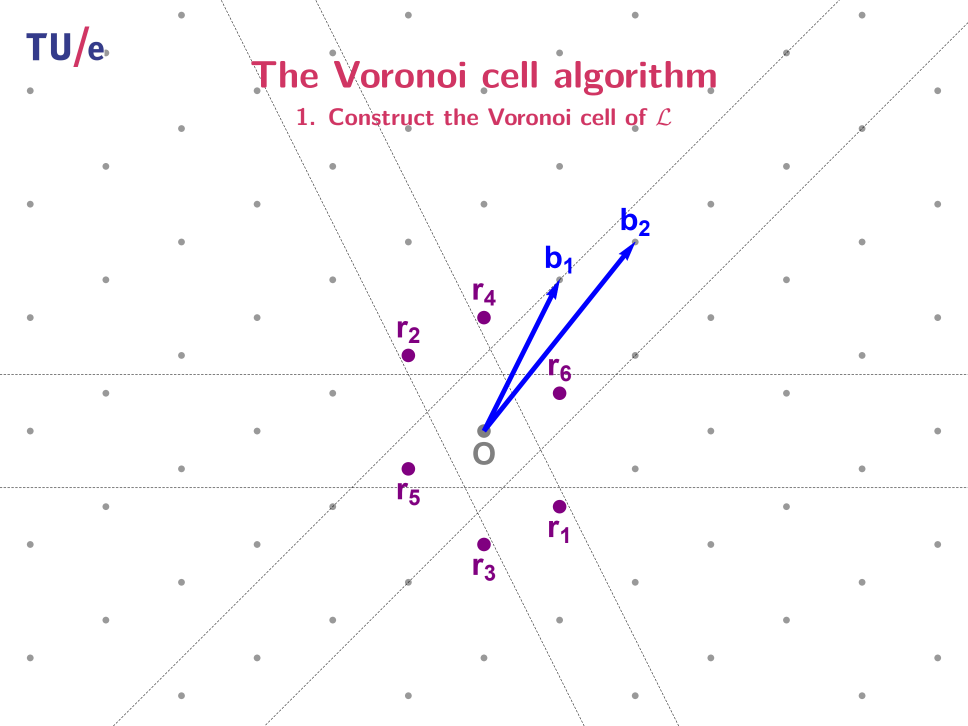
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



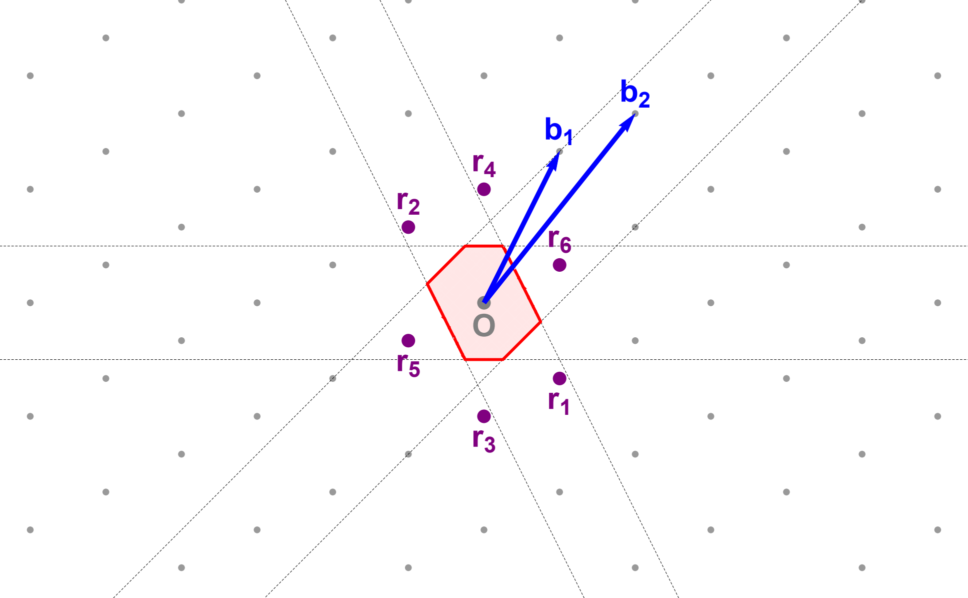
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



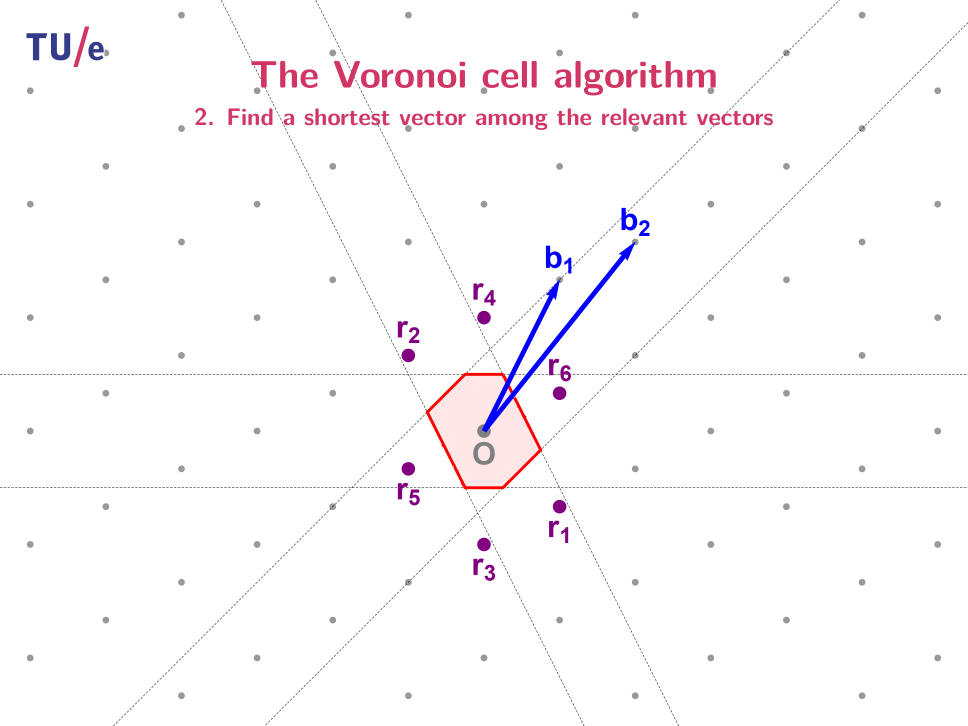
The Voronoi cell algorithm

1. Construct the Voronoi cell of $\mathcal{L}$



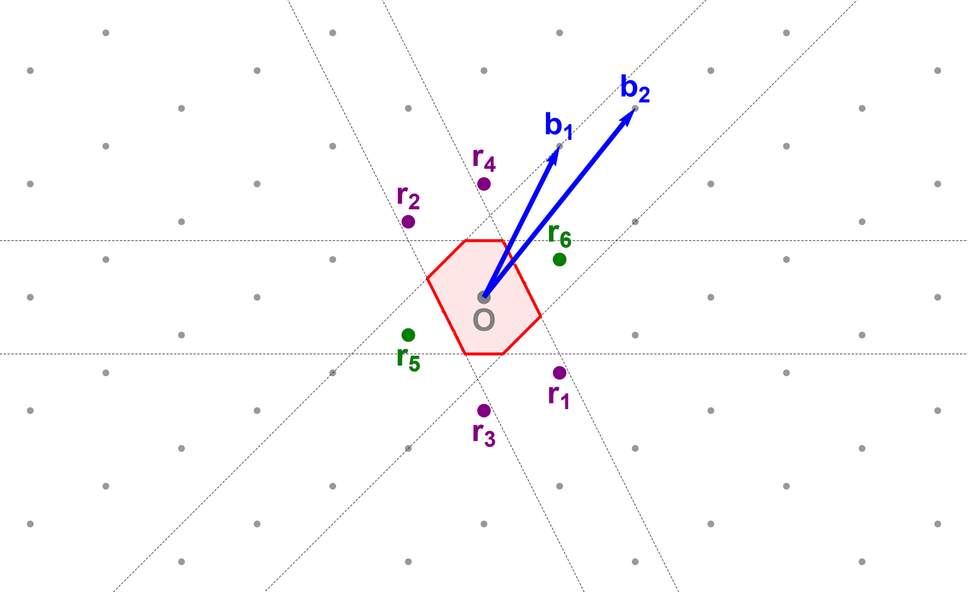
The Voronoi cell algorithm

2. Find a shortest vector among the relevant vectors



The Voronoi cell algorithm

2. Find a shortest vector among the relevant vectors



The Voronoi cell algorithm

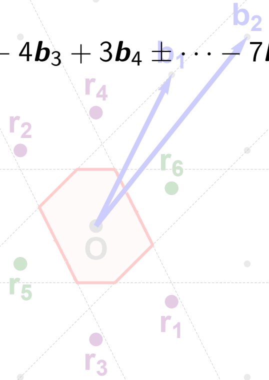
Analysis

The Voronoi cell algorithm

Analysis

Suppose the shortest vector $\boldsymbol{s}$ satisfies:

$$\boldsymbol{s} = 5\boldsymbol{b}_1 + 2\boldsymbol{b}_2 - 4\boldsymbol{b}_3 + 3\boldsymbol{b}_4 \pm \cdots - 7\boldsymbol{b}_n$$



The Voronoi cell algorithm

Analysis

Suppose the shortest vector $\boldsymbol{s}$ satisfies:

$$\boldsymbol{s} = 5\boldsymbol{b}_1 + 2\boldsymbol{b}_2 - 4\boldsymbol{b}_3 + 3\boldsymbol{b}_4 \pm \cdots - 7\boldsymbol{b}_n$$

Choose the vector $\boldsymbol{t} \in \{0,1\}^n \cdot B$ such that $\boldsymbol{s} + \boldsymbol{t} \in 2\mathcal{L}$:

$$\boldsymbol{t} = 1\boldsymbol{b}_1 + 0\boldsymbol{b}_2 + 0\boldsymbol{b}_3 + 1\boldsymbol{b}_4 \pm \cdots + 1\boldsymbol{b}_n$$

$$\boldsymbol{s} + \boldsymbol{t} = 6\boldsymbol{b}_1 + 2\boldsymbol{b}_2 - 4\boldsymbol{b}_3 + 4\boldsymbol{b}_4 \pm \cdots - 6\boldsymbol{b}_n \qquad (\in 2\mathcal{L})$$

The Voronoi cell algorithm

Analysis

Suppose the shortest vector $\boldsymbol{s}$ satisfies:

$$\boldsymbol{s} = 5\boldsymbol{b}_1 + 2\boldsymbol{b}_2 - 4\boldsymbol{b}_3 + 3\boldsymbol{b}_4 \pm \cdots - 7\boldsymbol{b}_n$$

Choose the vector $\boldsymbol{t} \in \{0,1\}^n \cdot B$ such that $\boldsymbol{s} + \boldsymbol{t} \in 2\mathcal{L}$:

$$\begin{aligned} \boldsymbol{t} &= 1\boldsymbol{b}_1 + 0\boldsymbol{b}_2 + 0\boldsymbol{b}_3 + 1\boldsymbol{b}_4 \pm \cdots + 1\boldsymbol{b}_n \\ \boldsymbol{s} + \boldsymbol{t} &= 6\boldsymbol{b}_1 + 2\boldsymbol{b}_2 - 4\boldsymbol{b}_3 + 4\boldsymbol{b}_4 \pm \cdots - 6\boldsymbol{b}_n \qquad (\in 2\mathcal{L}) \end{aligned}$$

The vectors closest to $\boldsymbol{t}$ in the lattice $2\mathcal{L}$ are $\boldsymbol{t} \pm \boldsymbol{s}$.

The Voronoi cell algorithm

Analysis

Suppose the shortest vector $\boldsymbol{s}$ satisfies:

$$\boldsymbol{s} = 5\boldsymbol{b}_1 + 2\boldsymbol{b}_2 - 4\boldsymbol{b}_3 + 3\boldsymbol{b}_4 \pm \cdots - 7\boldsymbol{b}_n$$

Choose the vector $\boldsymbol{t} \in \{0, 1\}^n \cdot B$ such that $\boldsymbol{s} + \boldsymbol{t} \in 2\mathcal{L}$:

$$\begin{aligned} \boldsymbol{t} &= 1\boldsymbol{b}_1 + 0\boldsymbol{b}_2 + 0\boldsymbol{b}_3 + 1\boldsymbol{b}_4 \pm \cdots + 1\boldsymbol{b}_n \\ \boldsymbol{s} + \boldsymbol{t} &= 6\boldsymbol{b}_1 + 2\boldsymbol{b}_2 - 4\boldsymbol{b}_3 + 4\boldsymbol{b}_4 \pm \cdots - 6\boldsymbol{b}_n \qquad (\in 2\mathcal{L}) \end{aligned}$$

The vectors closest to $\boldsymbol{t}$ in the lattice $2\mathcal{L}$ are $\boldsymbol{t} \pm \boldsymbol{s}$.

Theorem (Micciancio and Voulgaris, SODA'10)

The Voronoi cell algorithm runs in time $2^{2n+o(n)}$ and space $2^{n+o(n)}$.

Outline

Lattices

Enumeration algorithms
- Fincke-Pohst enumeration
- Kannan enumeration
- Pruning the enumeration tree

The Voronoi cell algorithm

Sieving algorithms
- Nguyen-Vidick sieve
- Multiple levels
- GaussSieve

Sieving using angular locality-sensitive hashing
- Nguyen-Vidick sieve
- GaussSieve

Sieving

Studied since 2001 [AKS01, Reg04, NV08, . . ., ZPH13, BGJ14]

1. Generate a long list L of random lattice vectors

Sieving

Studied since 2001 [AKS01, Reg04, NV08, . . ., ZPH13, BGJ14]

1. Generate a long list L of random lattice vectors
2. Split L into two sets C (centers) and R (rest):
   - Set $C = \emptyset$ and $R = \emptyset$
   - For each $\boldsymbol{v} \in L$, find the closest $\boldsymbol{c} \in C$
     - If $\|\boldsymbol{v} - \boldsymbol{c}\|$ is “large”, add $\boldsymbol{v}$ to C
     - If $\|\boldsymbol{v} - \boldsymbol{c}\|$ is “small”, add $\boldsymbol{v} - \boldsymbol{c}$ to R

Sieving

Studied since 2001 [AKS01, Reg04, NV08, ..., ZPH13, BGJ14]

1. Generate a long list L of random lattice vectors
2. Split L into two sets C (centers) and R (rest):
   - Set $C = \emptyset$ and $R = \emptyset$
   - For each $\boldsymbol{v} \in L$, find the closest $\boldsymbol{c} \in C$
     - If $\|\boldsymbol{v} - \boldsymbol{c}\|$ is "large", add $\boldsymbol{v}$ to C
     - If $\|\boldsymbol{v} - \boldsymbol{c}\|$ is "small", add $\boldsymbol{v} - \boldsymbol{c}$ to R
3. Set $L \leftarrow R$ and repeat until L contains a shortest vector

Nguyen-Vidick sieve

1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve

1. Sample a list L of random lattice vectors

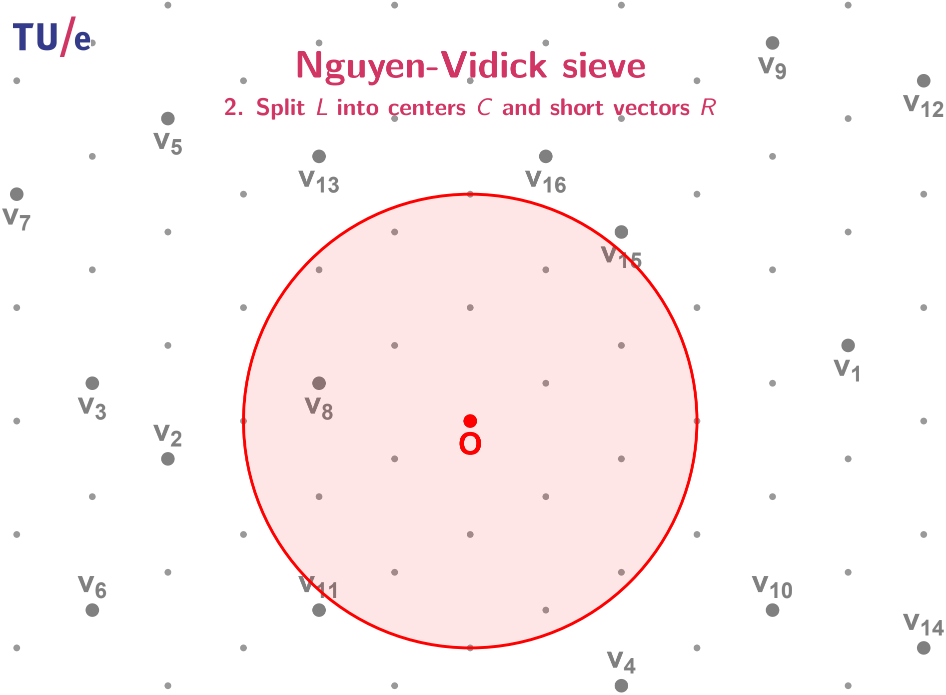
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R

TU/e

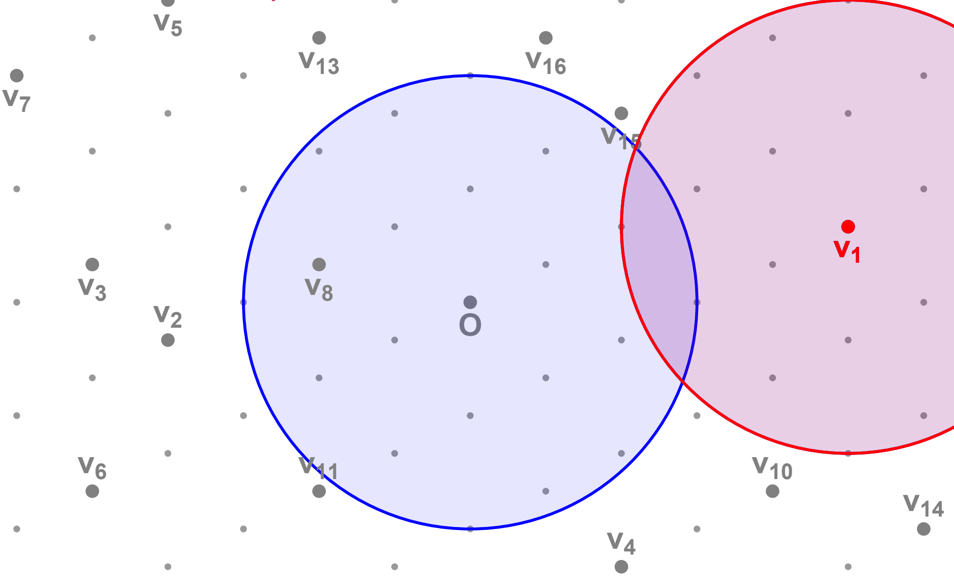
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



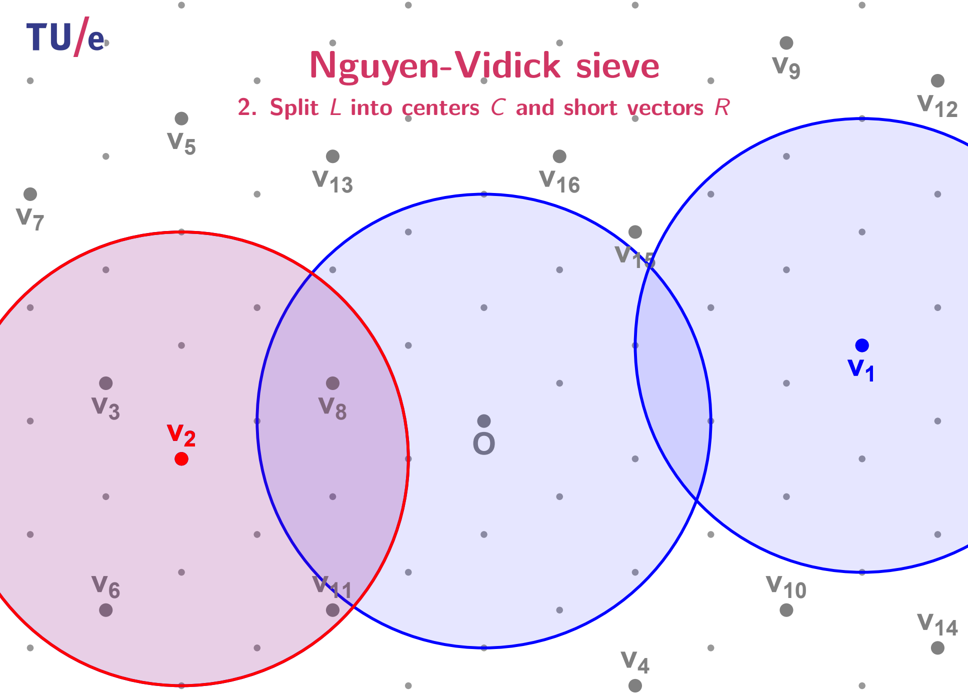
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



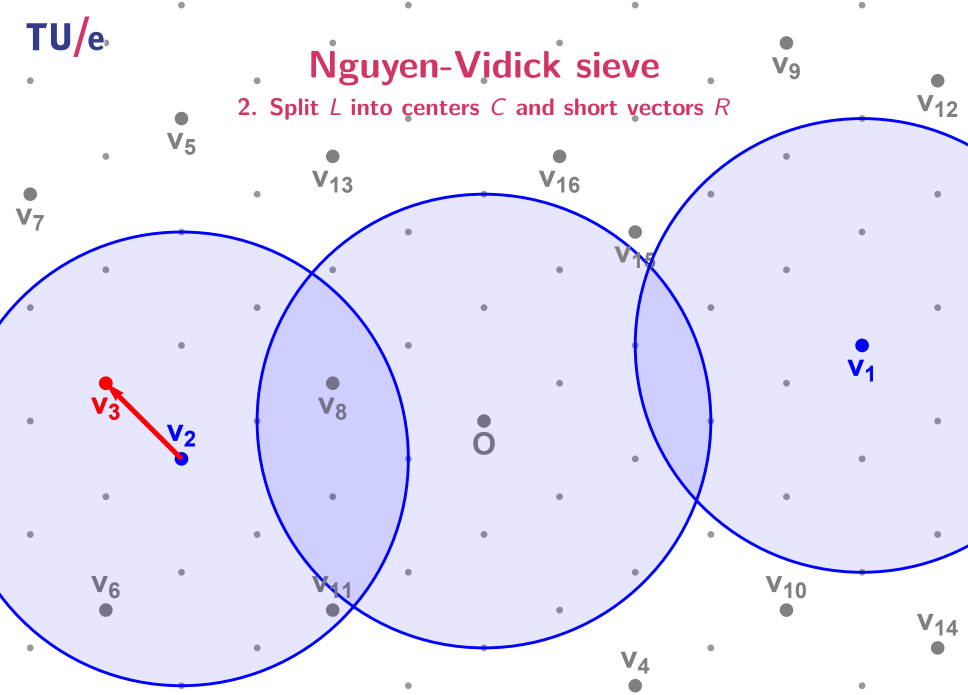
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



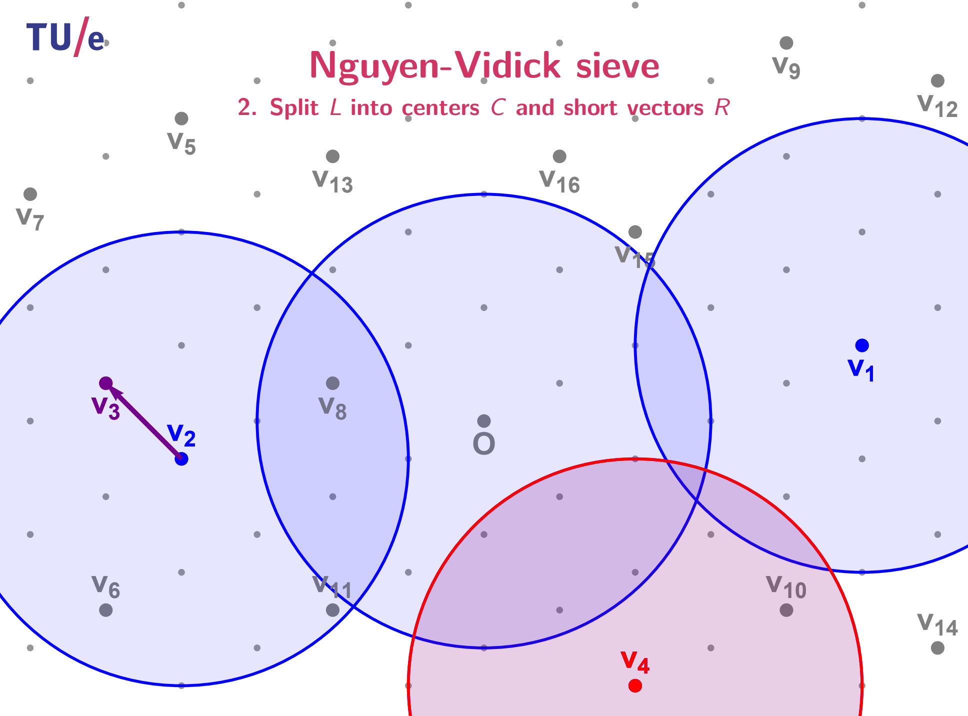
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



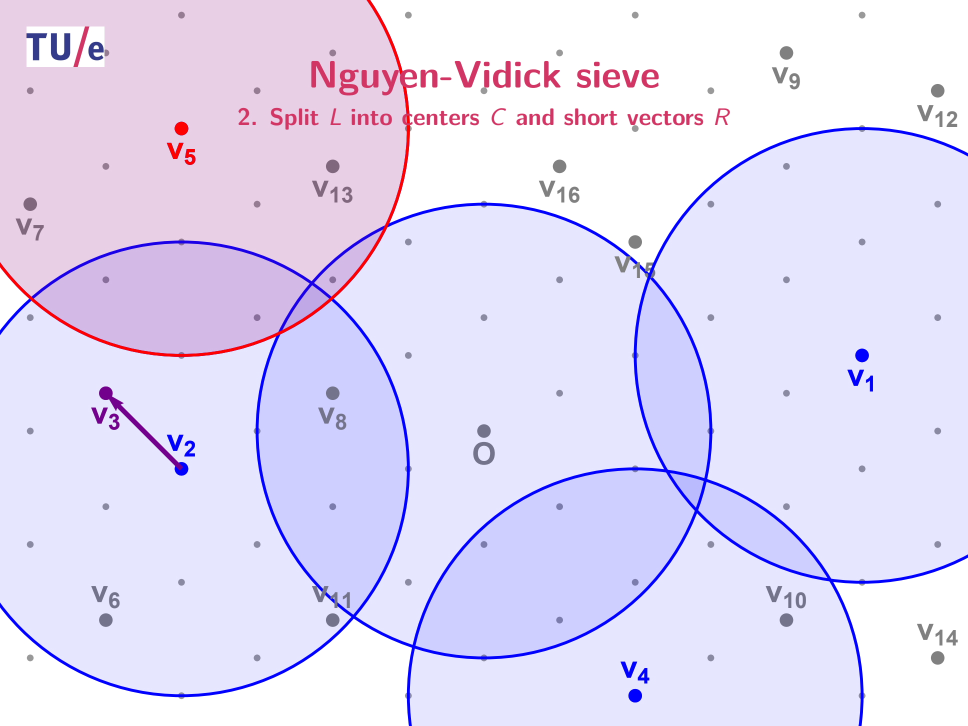
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



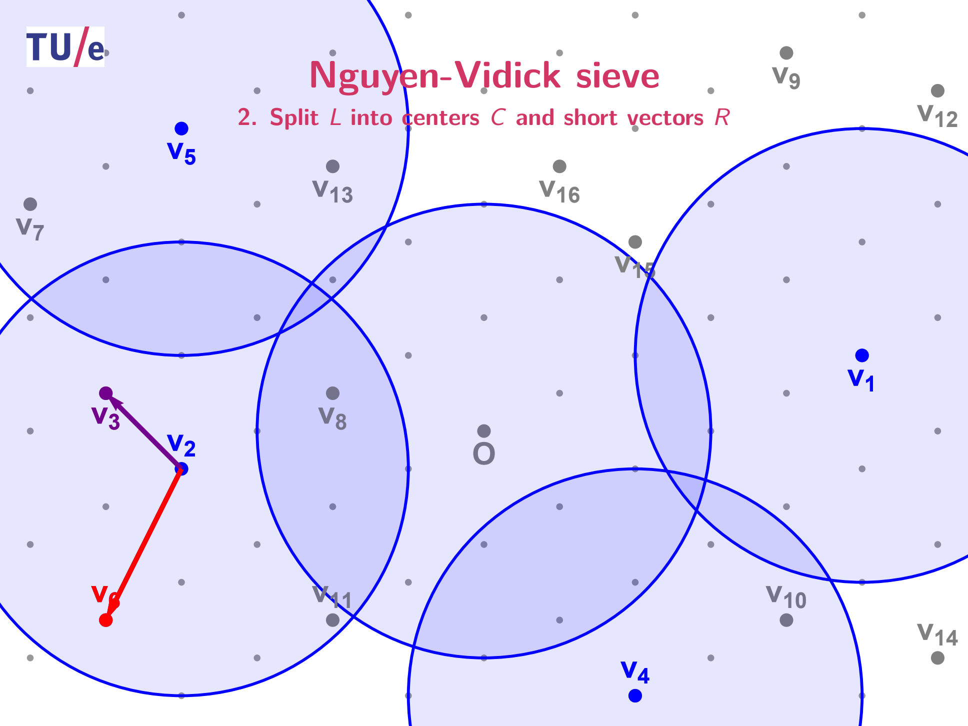
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



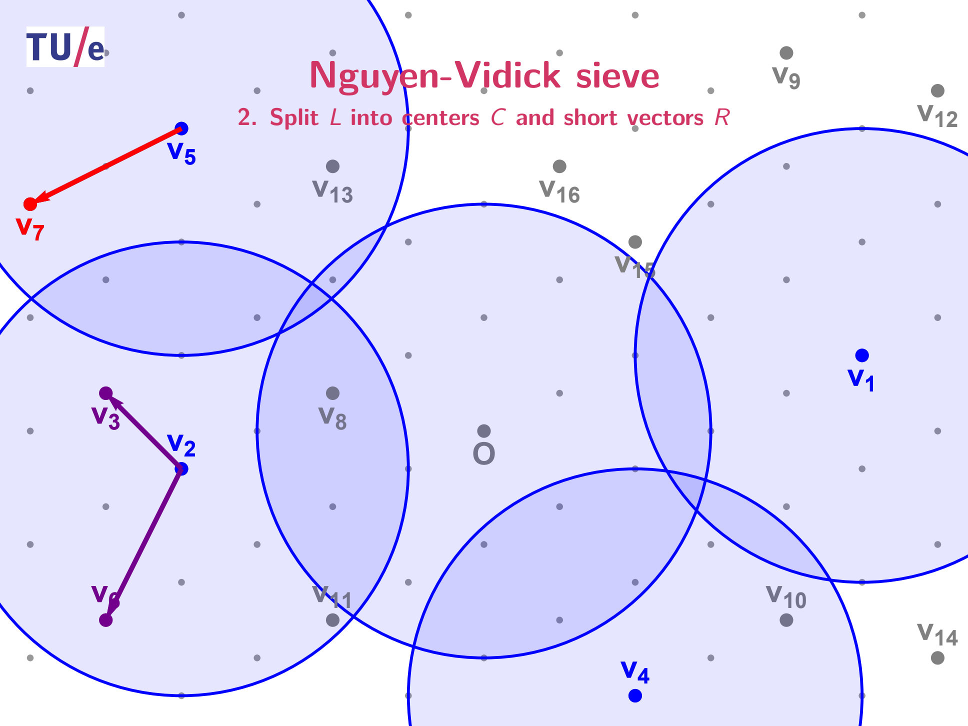
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



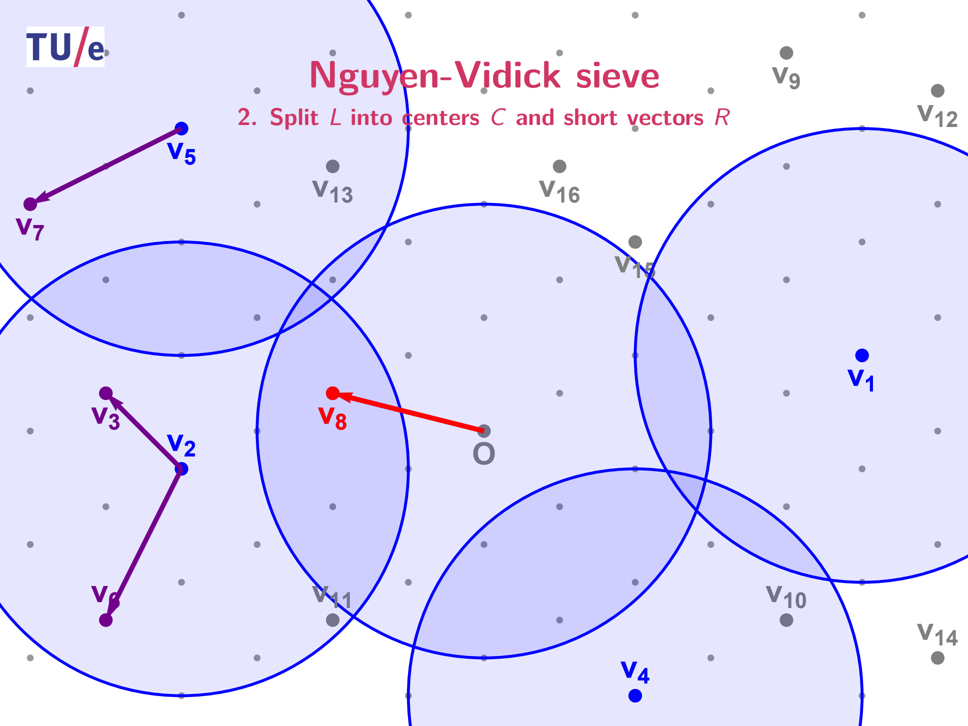
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



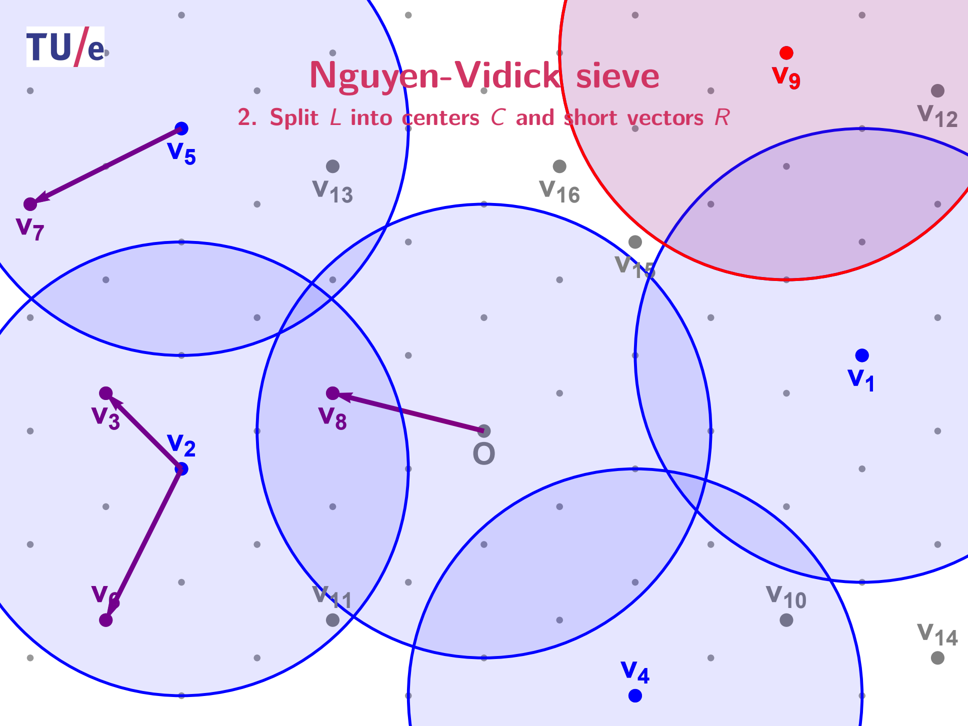
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



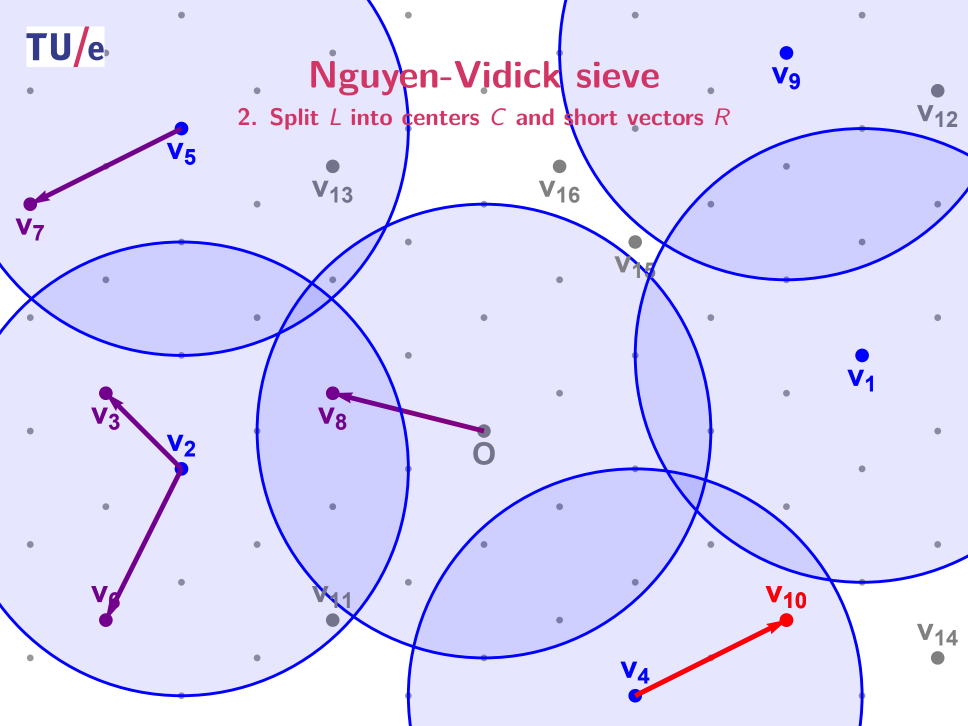
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



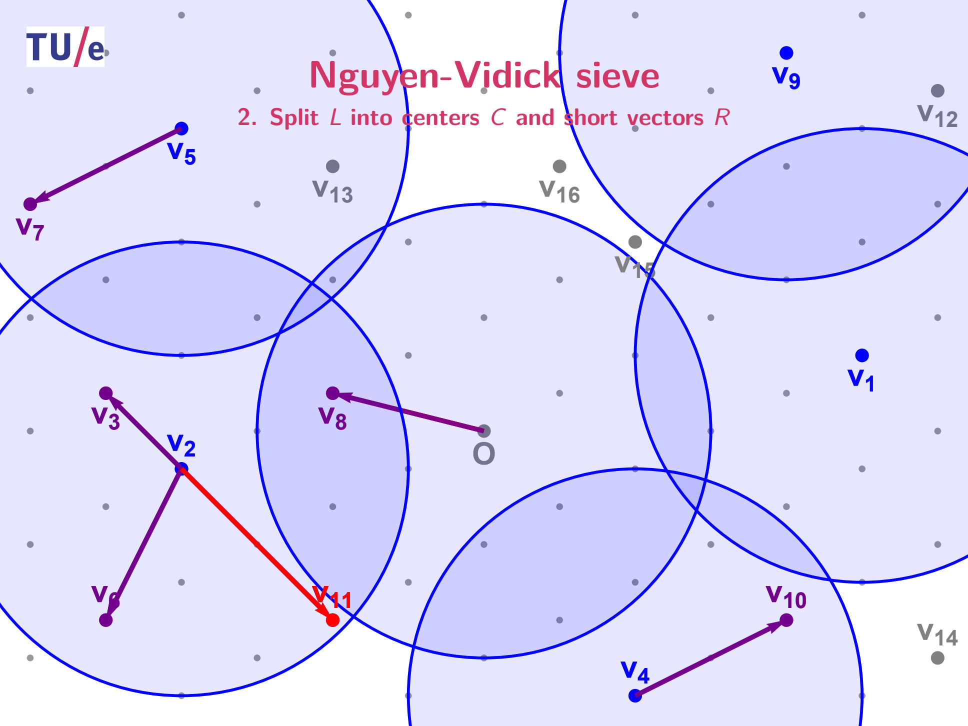
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



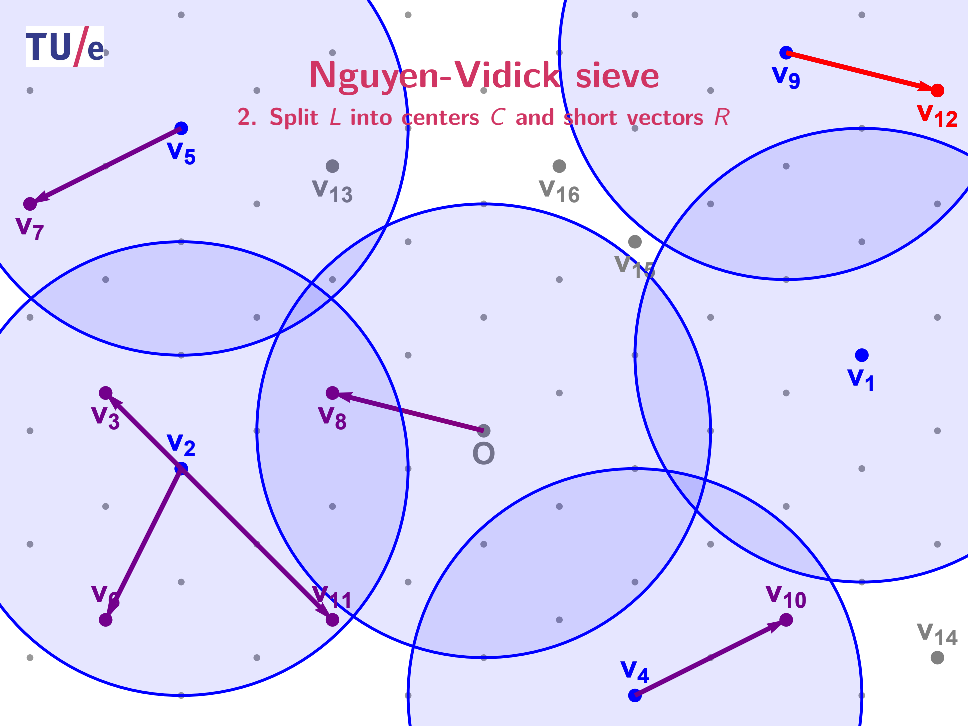
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



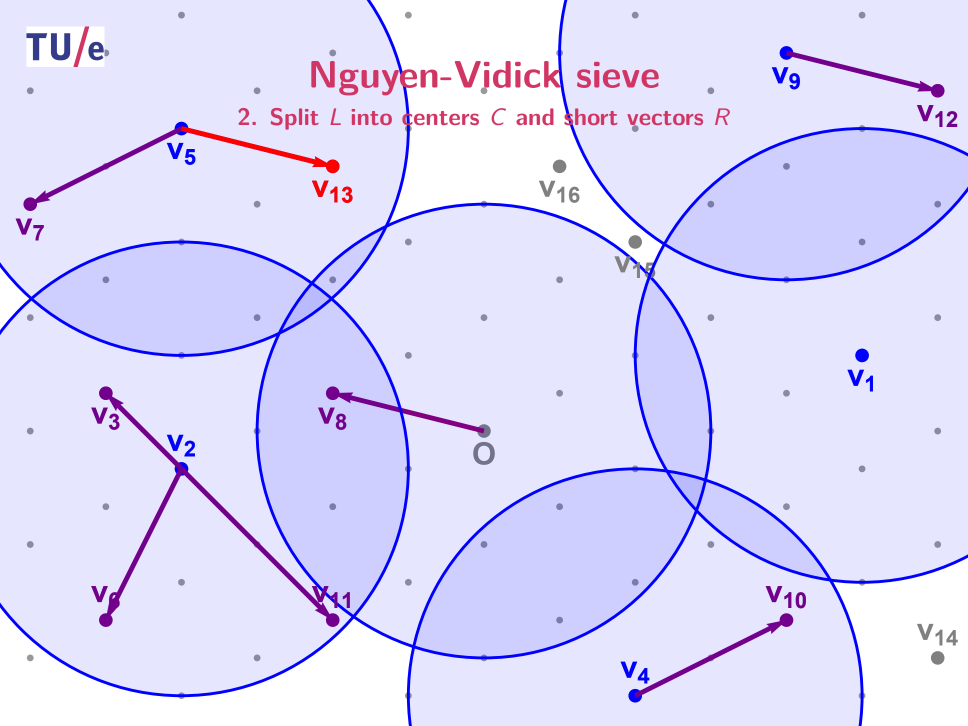
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



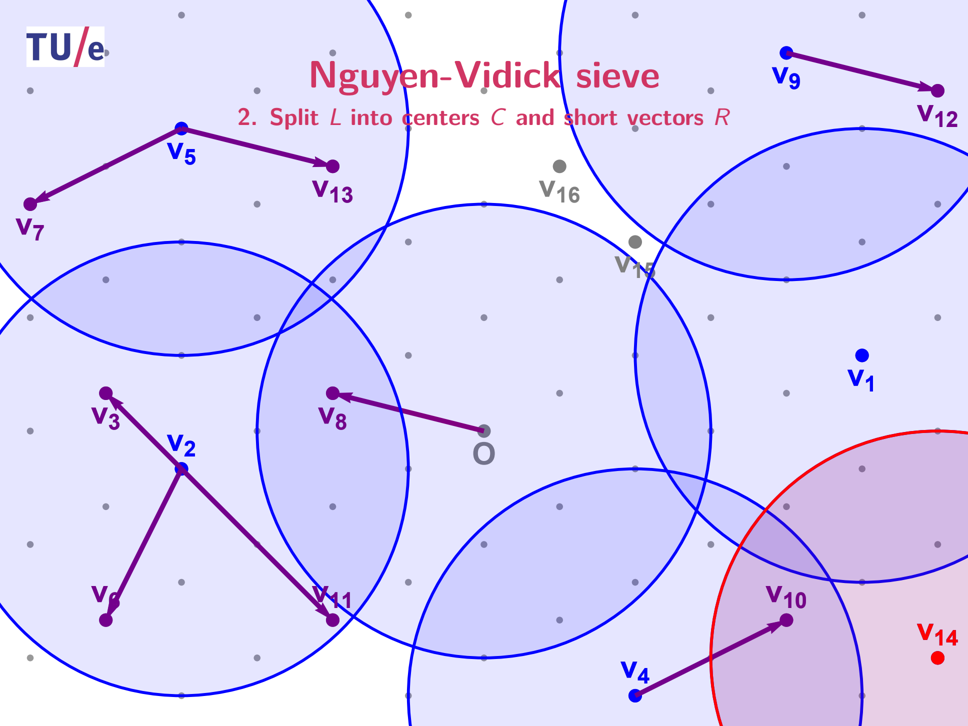
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



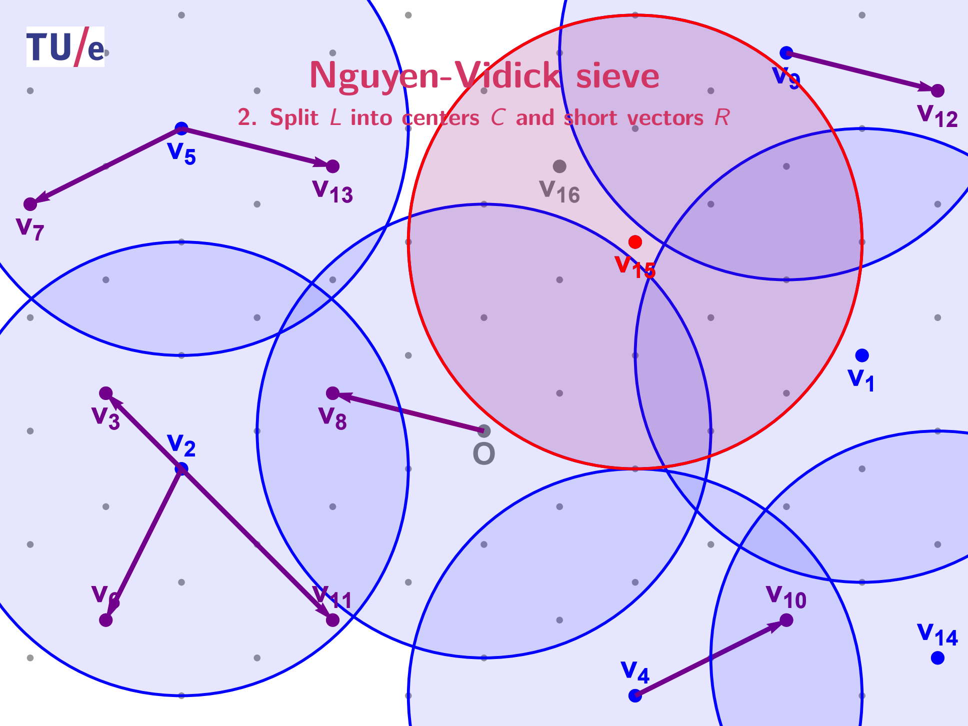
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



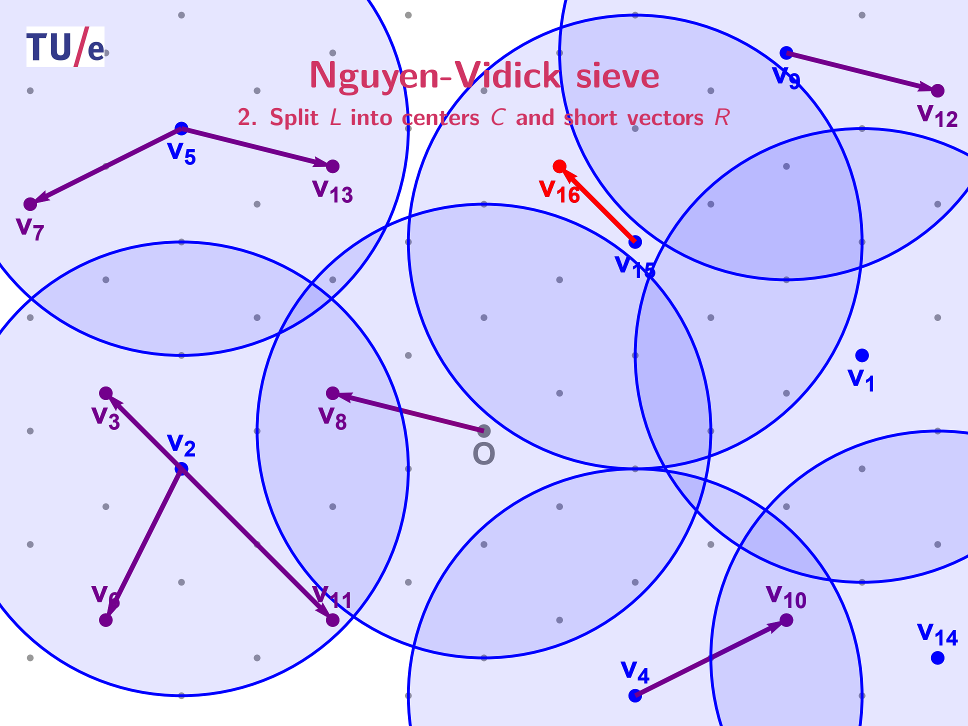
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



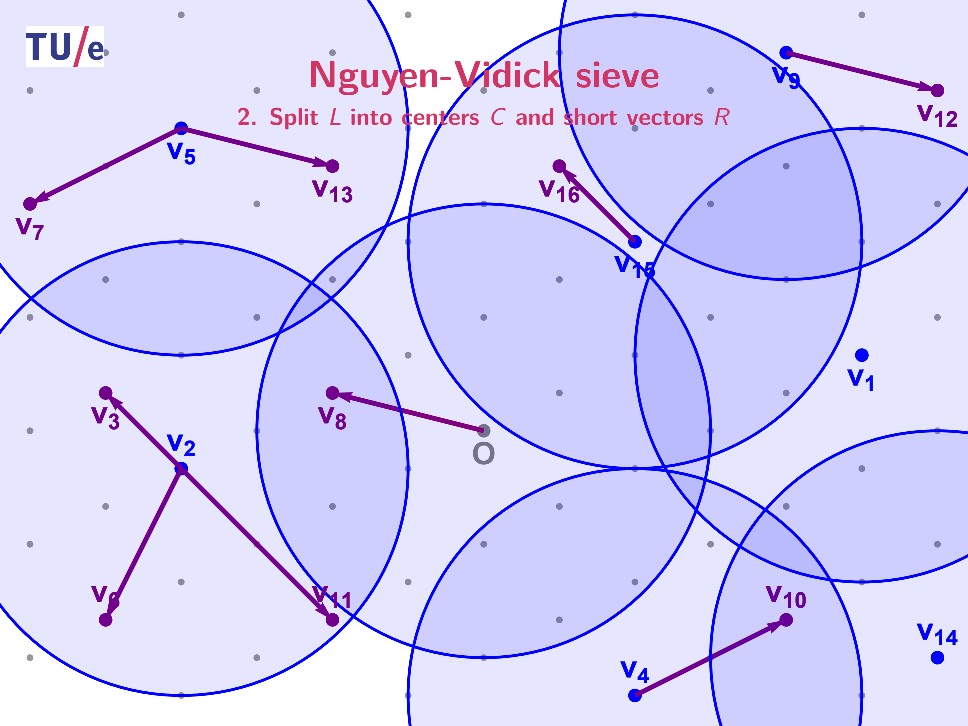
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



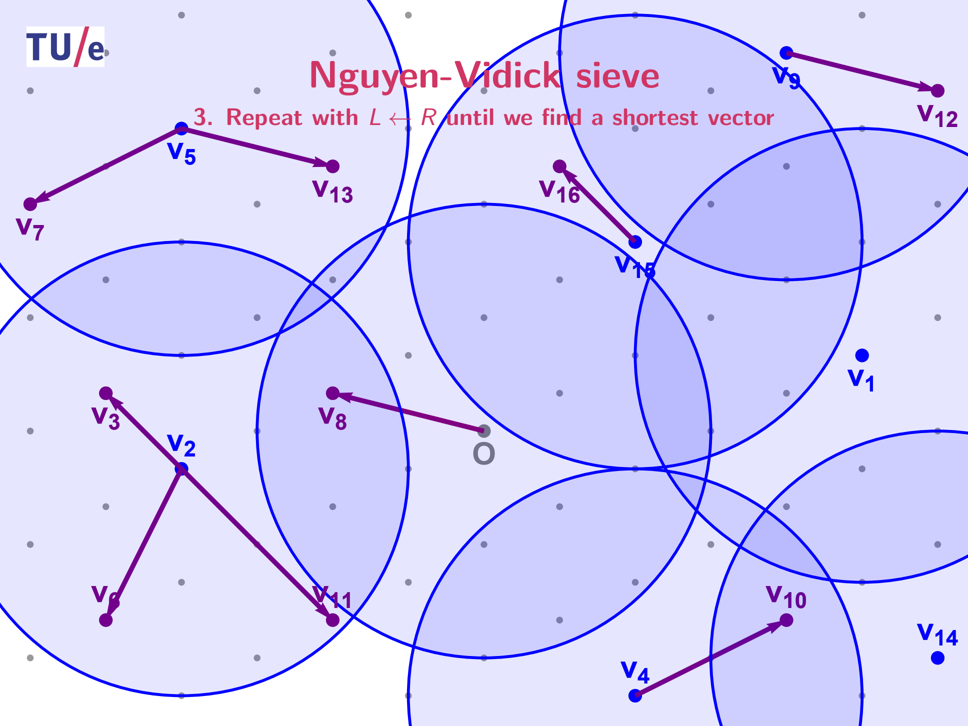
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



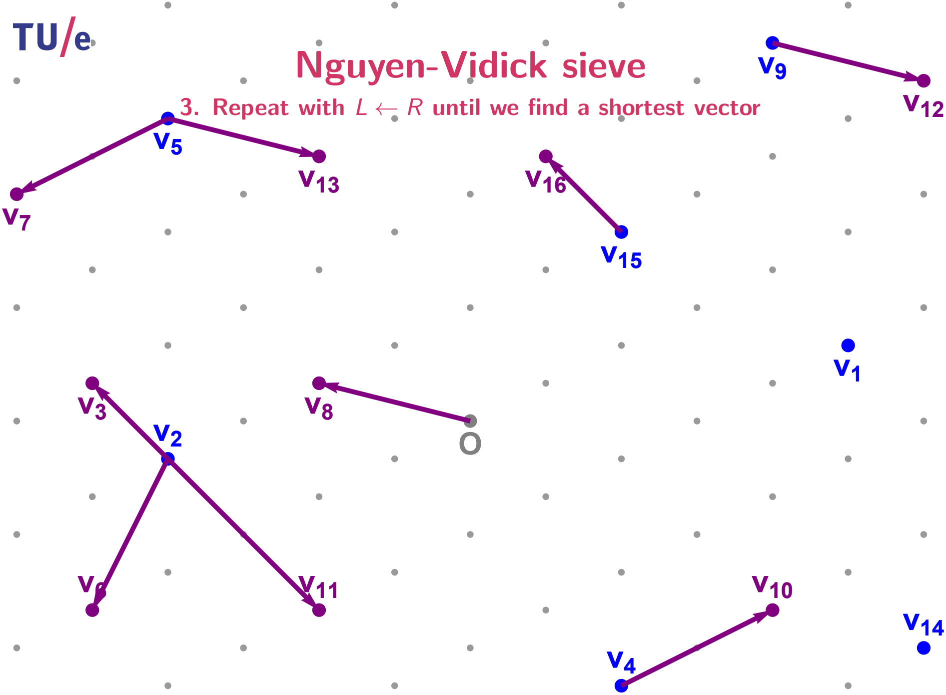
Nguyen-Vidick sieve

3. Repeat with $L \leftarrow R$ until we find a shortest vector



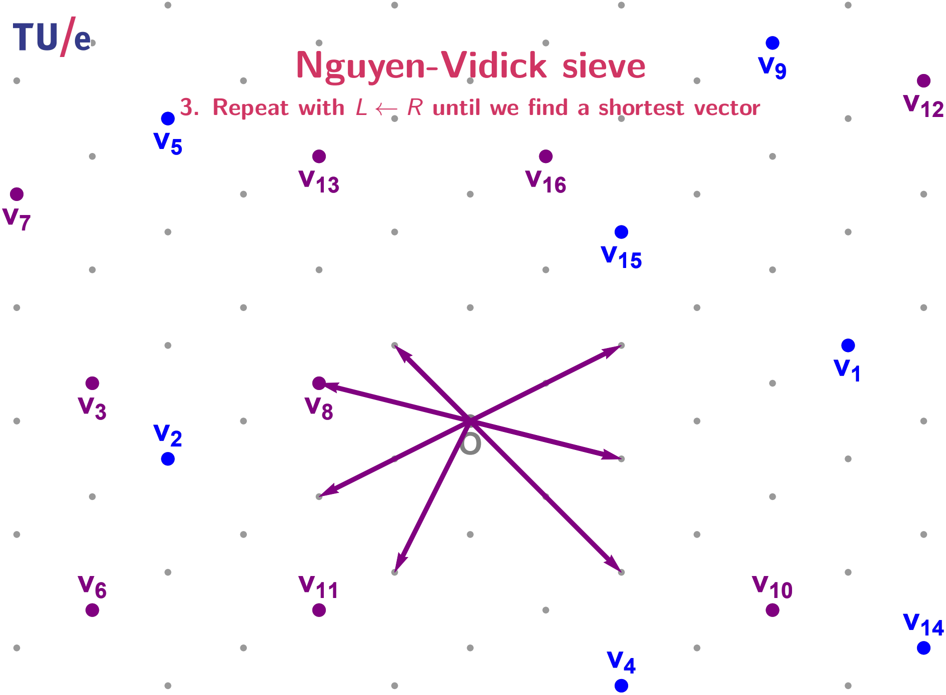
Nguyen-Vidick sieve

3. Repeat with $L \leftarrow R$ until we find a shortest vector



Nguyen-Vidick sieve

3. Repeat with $L \leftarrow R$ until we find a shortest vector



Nguyen-Vidick sieve

3. Repeat with $L \leftarrow R$ until we find a shortest vector

Nguyen-Vidick sieve

Overview

Nguyen-Vidick sieve

Overview

Heuristic (Nguyen and Vidick, J. Math. Crypt. '08)

The Nguyen-Vidick sieve runs in time $(4/3)^n$ and space $\sqrt{4/3}^n$.

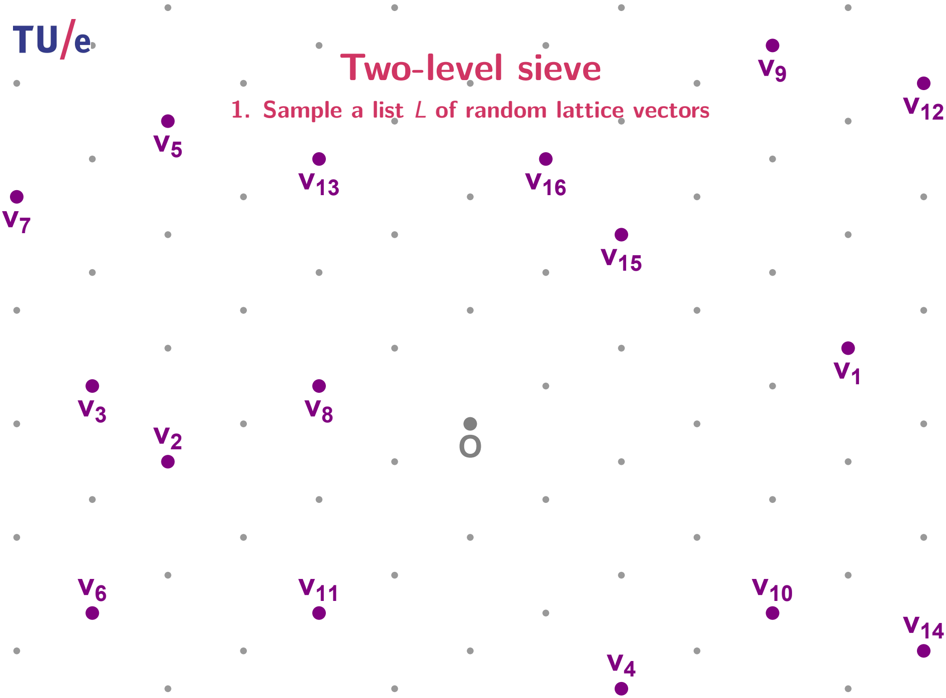
Two-level sieve

1. Sample a list L of random lattice vectors

TU/e

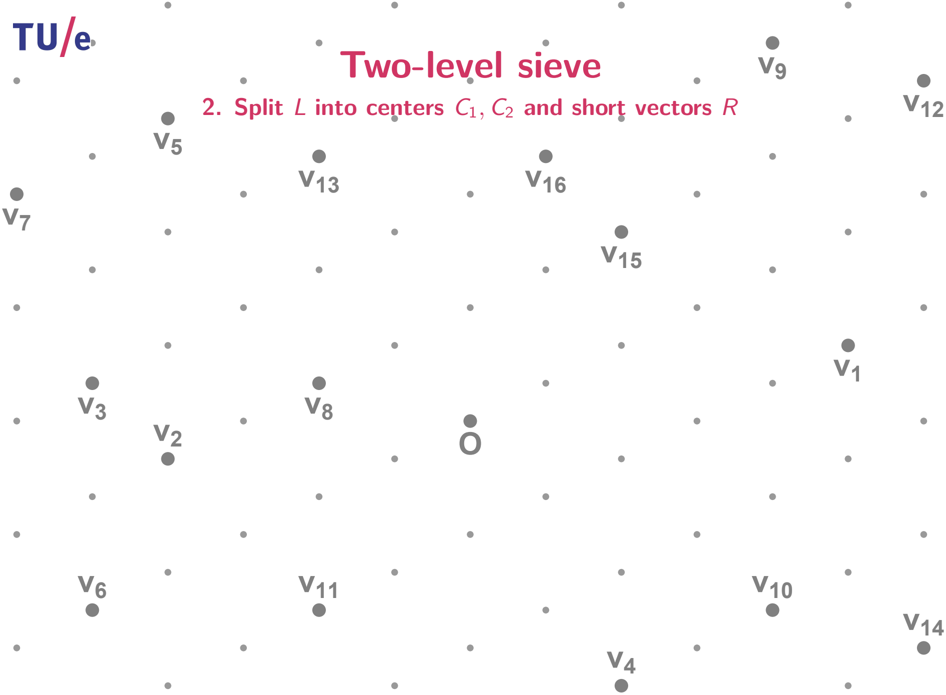
Two-level sieve

1. Sample a list L of random lattice vectors



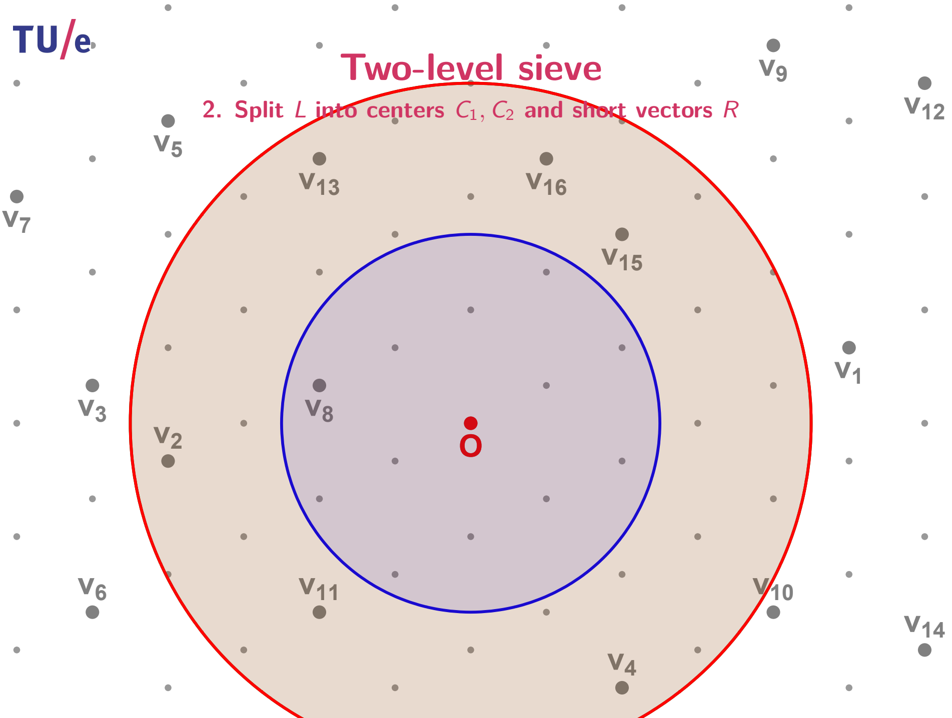
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



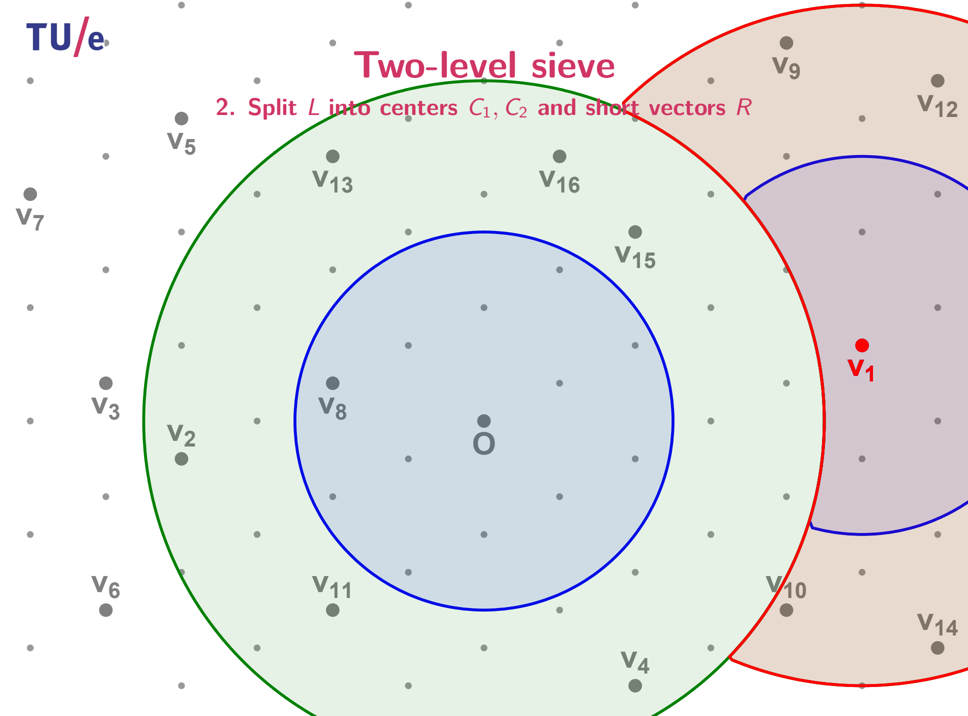
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



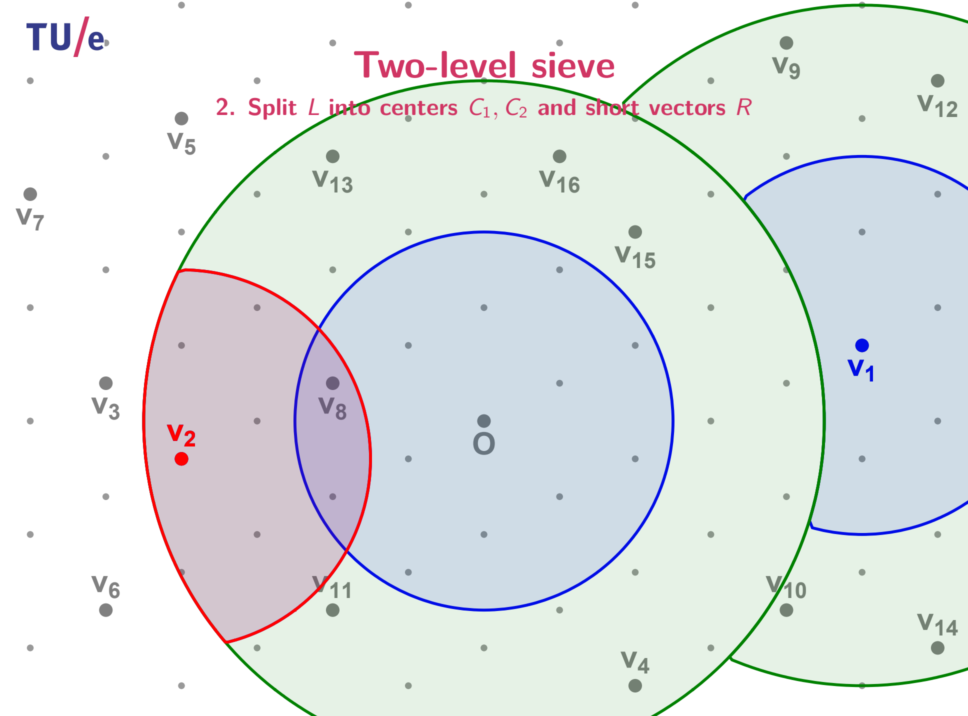
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



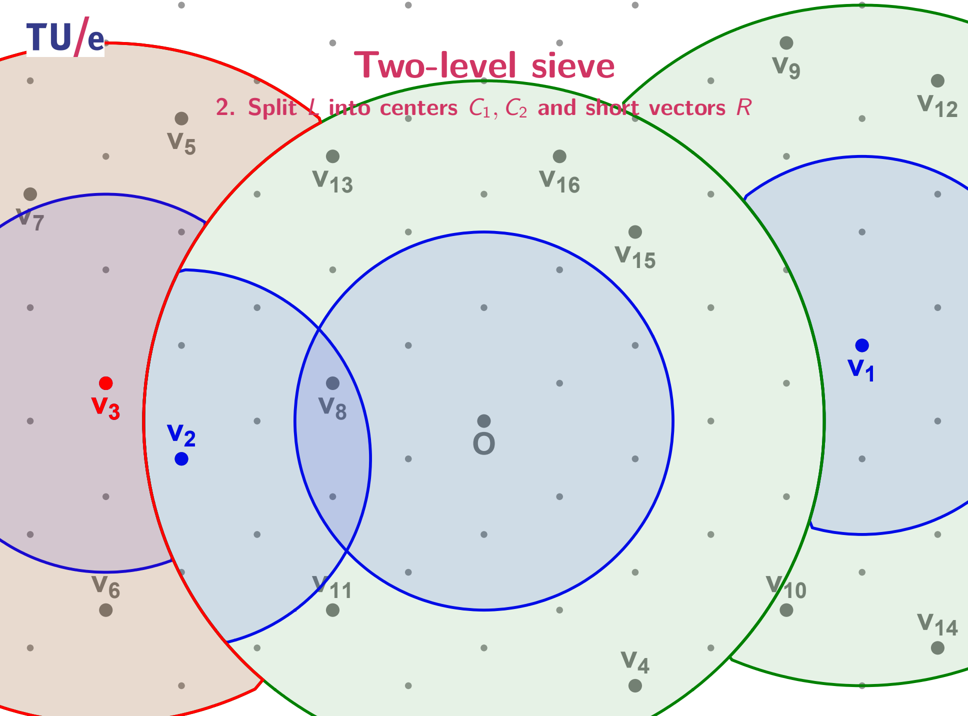
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



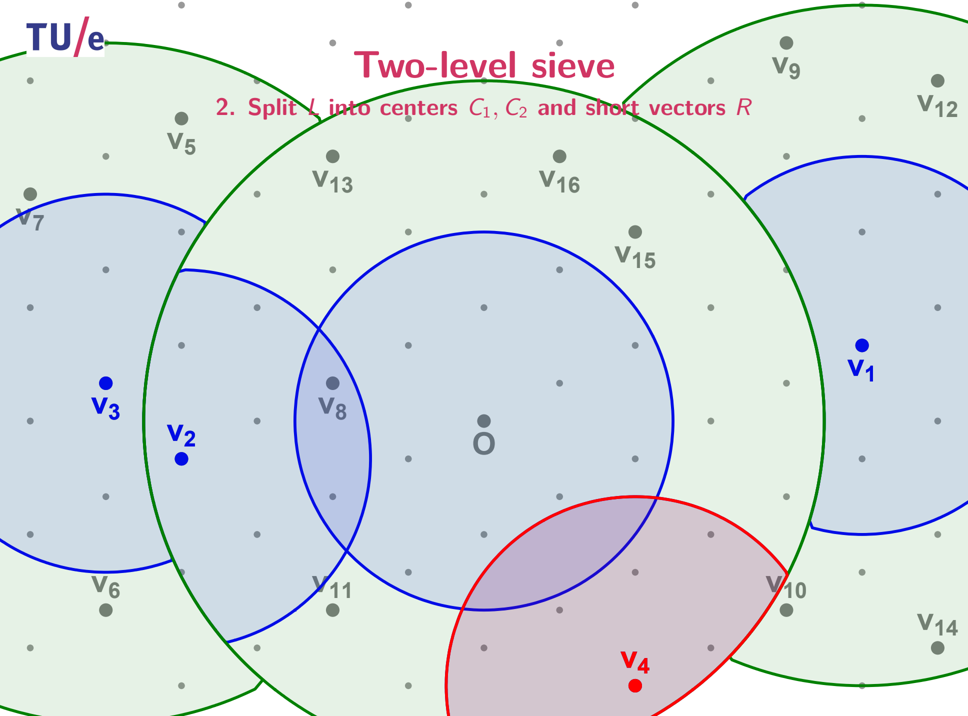
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



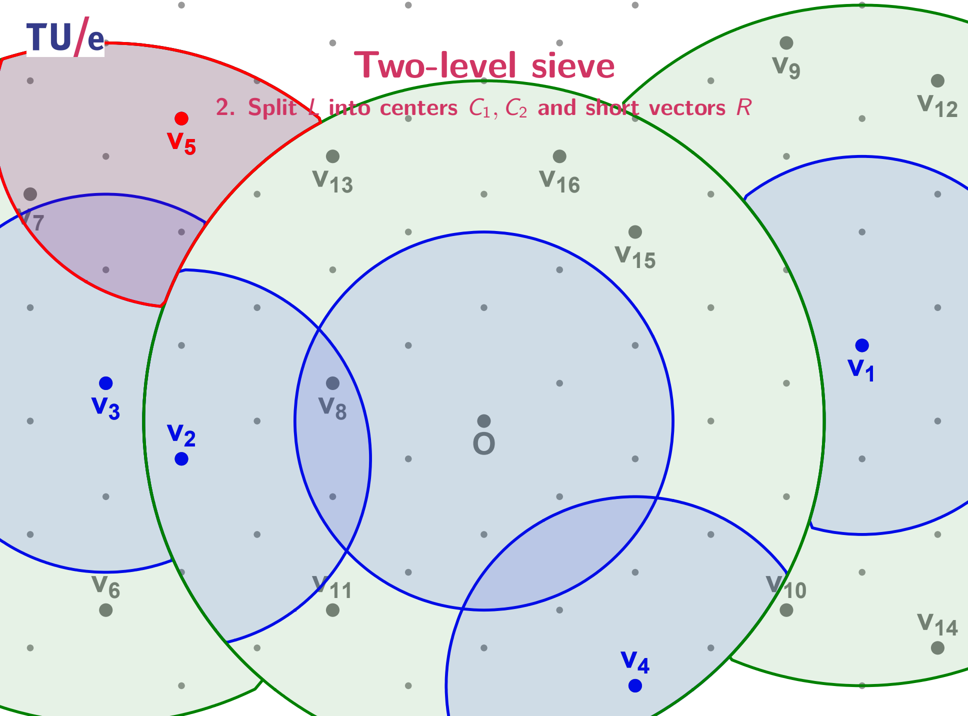
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



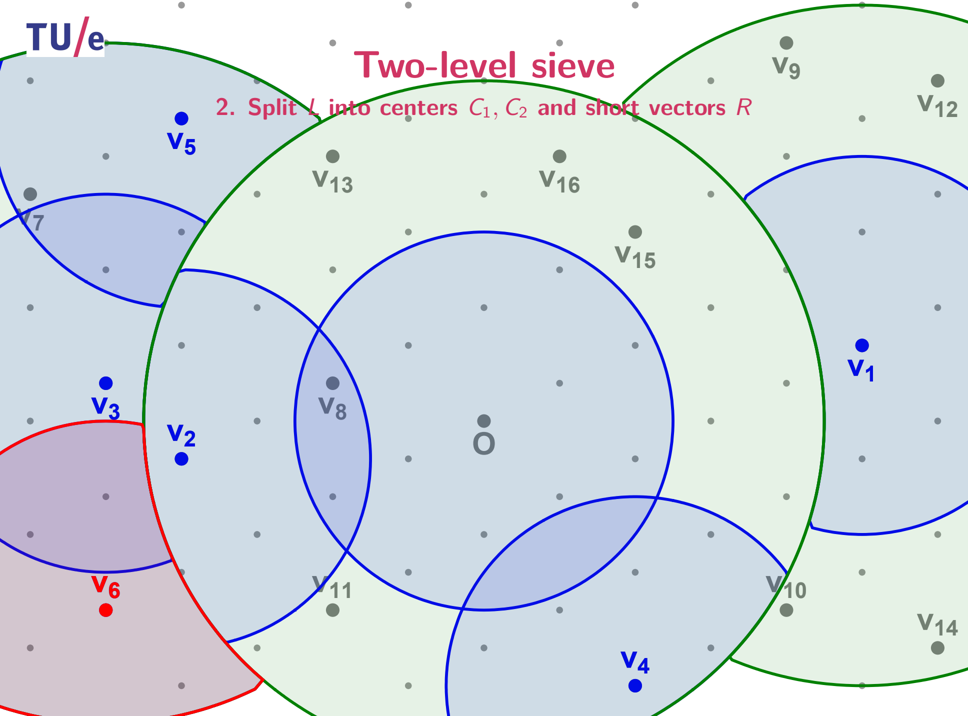
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



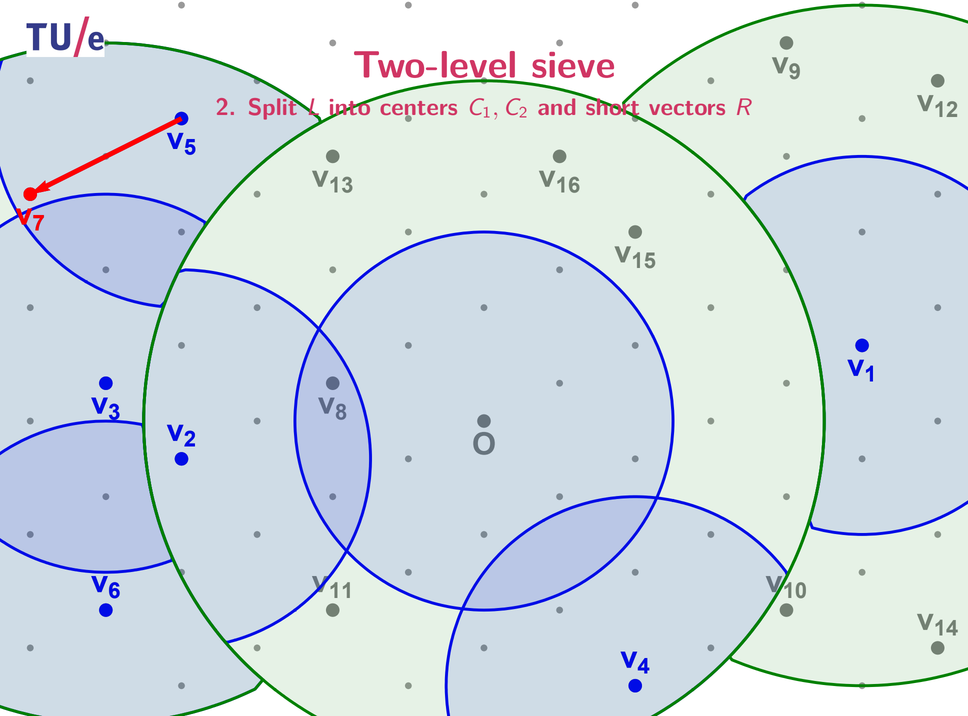
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



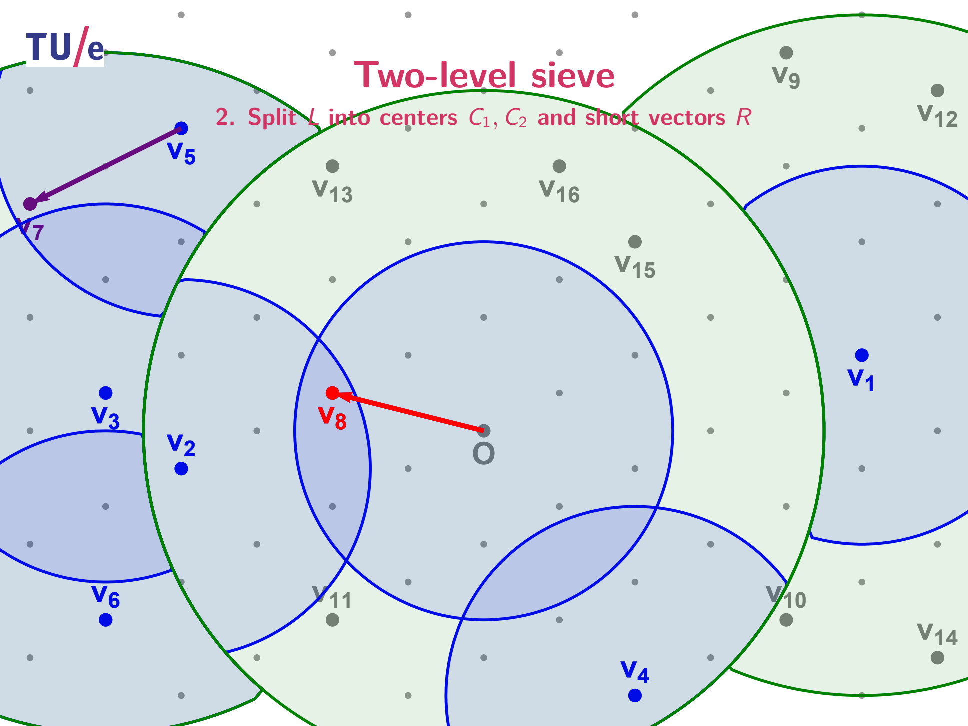
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



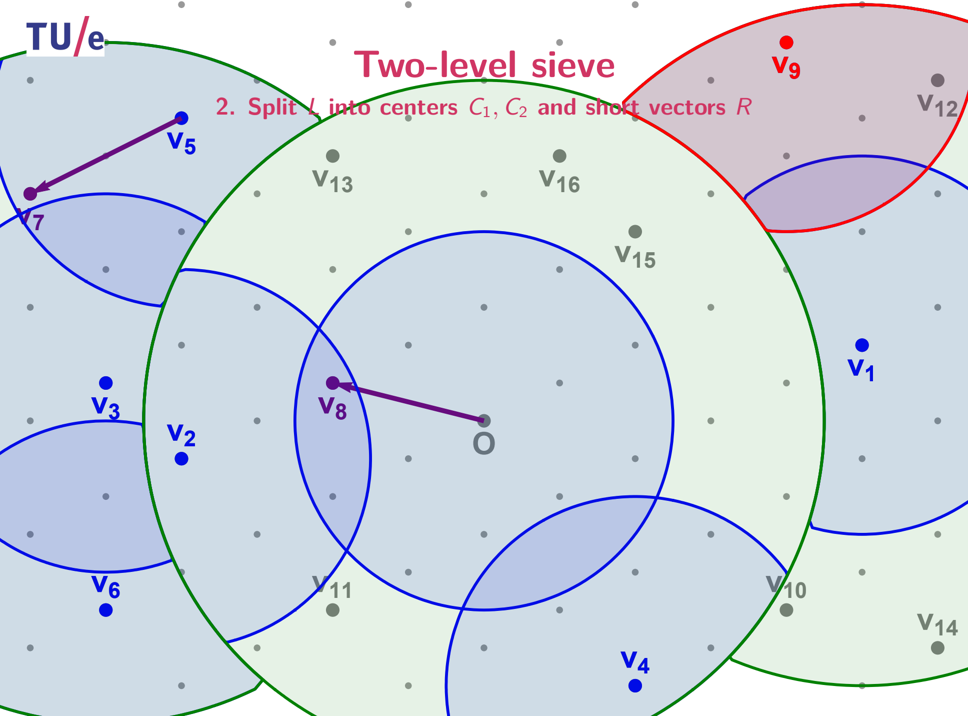
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



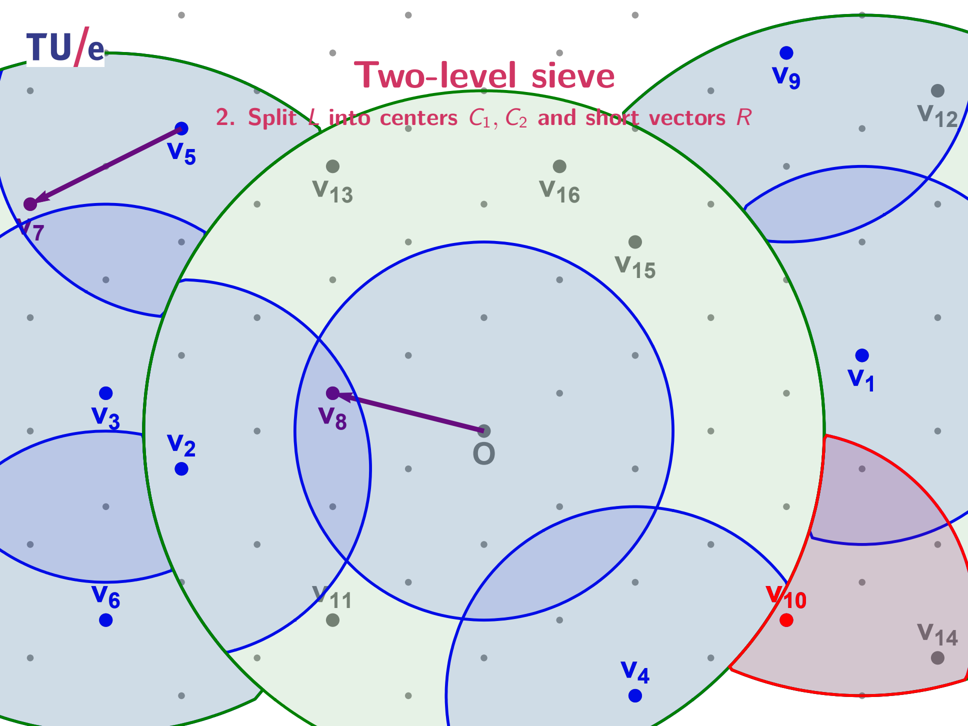
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



Two-level sieve

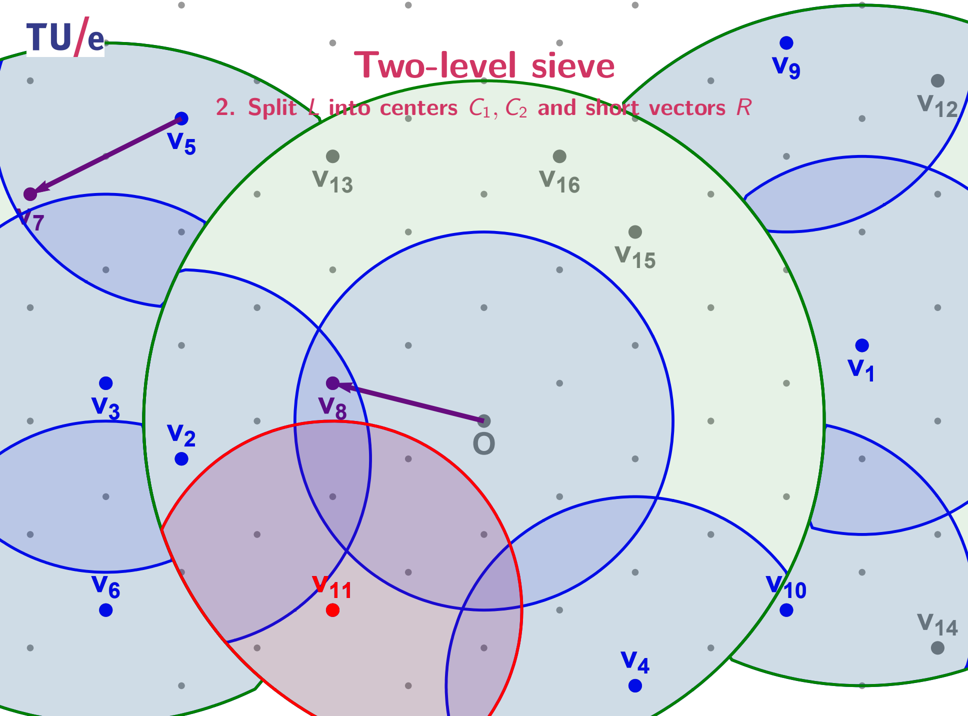
2. Split L into centers C_1, C_2 and short vectors R



TU/e

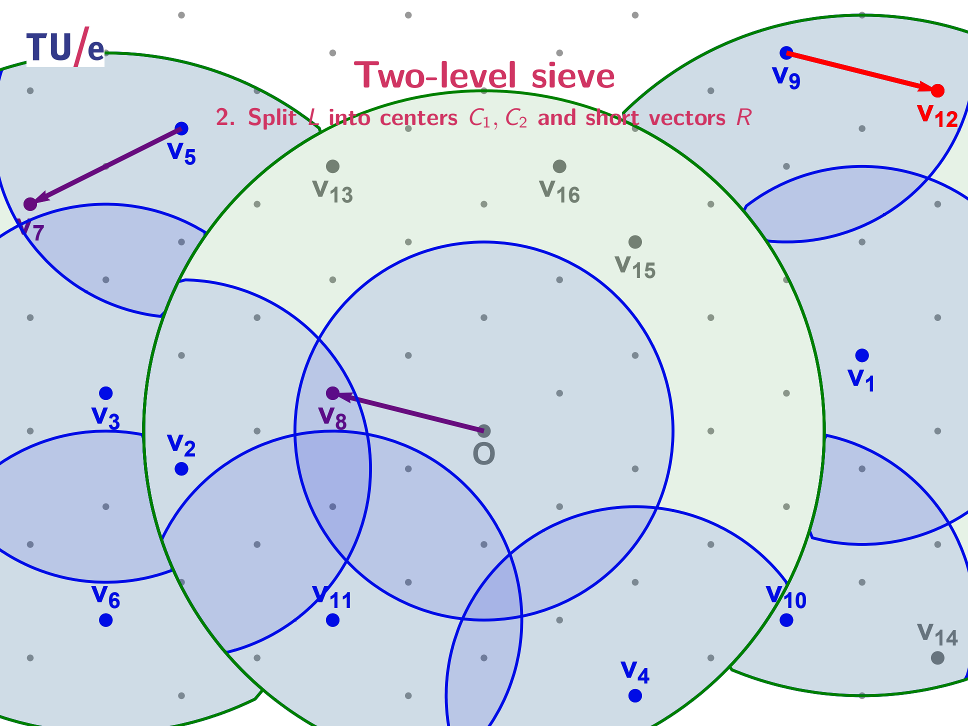
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



Two-level sieve

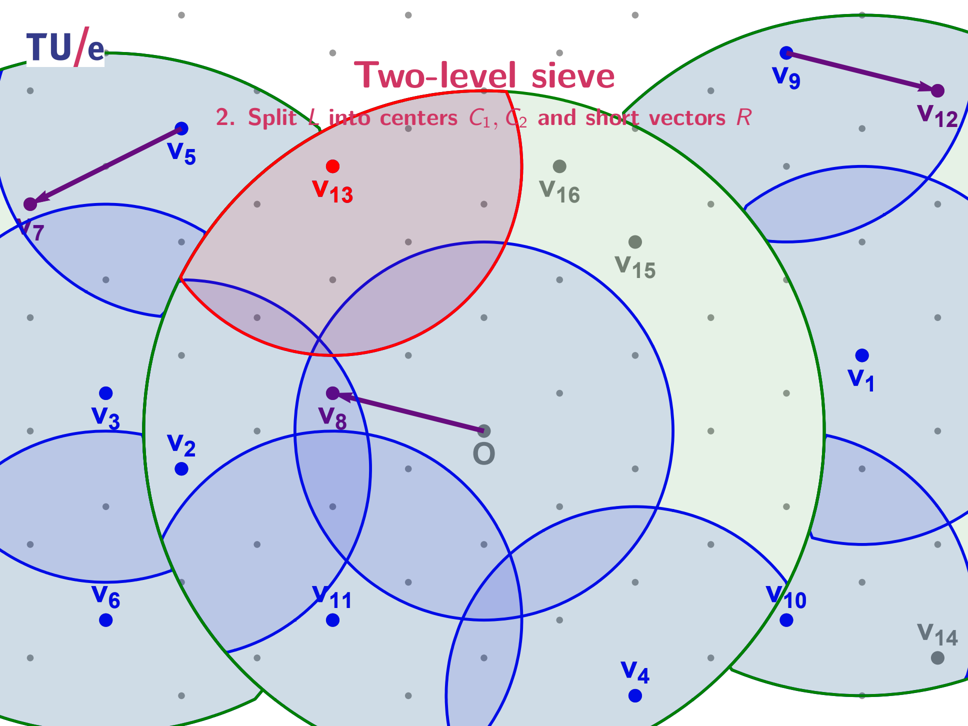
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

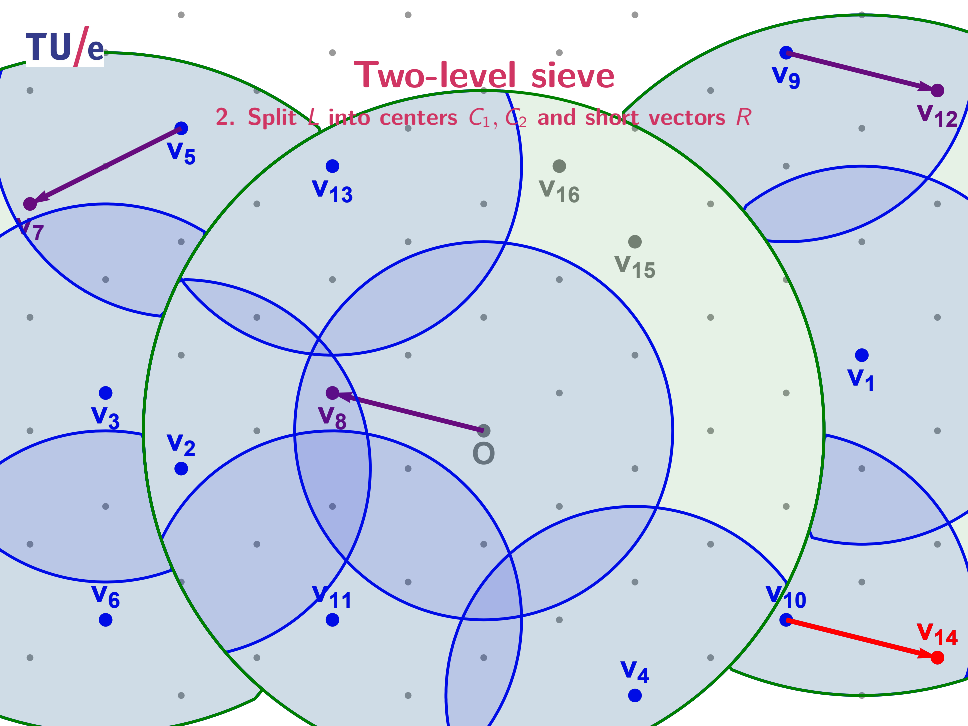
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

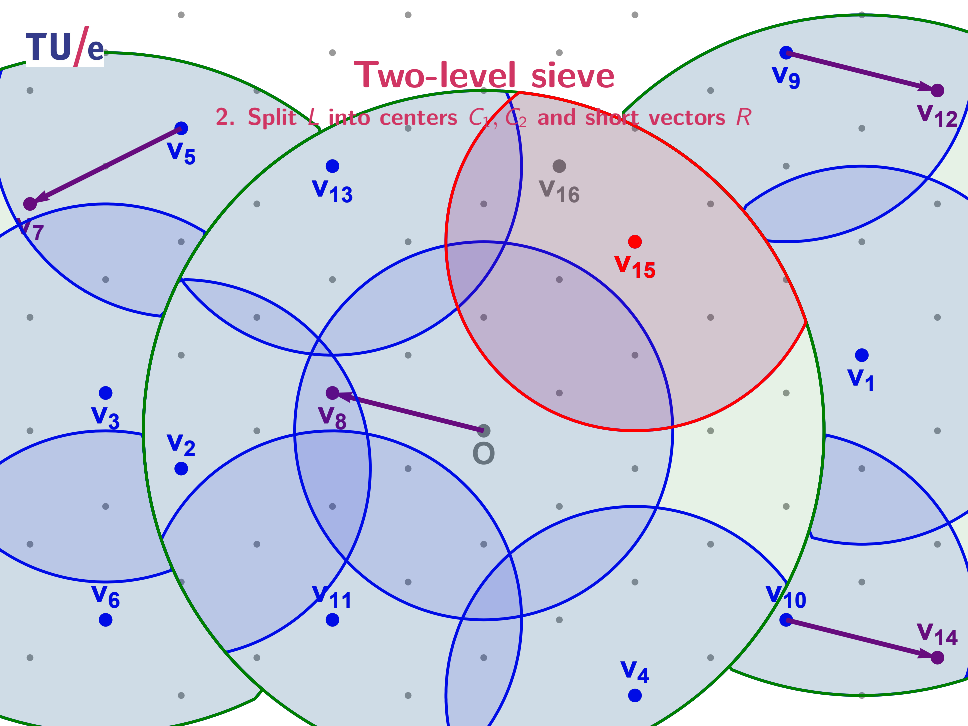
2. Split L into centers C_1, C_2 and short vectors R



TU/e

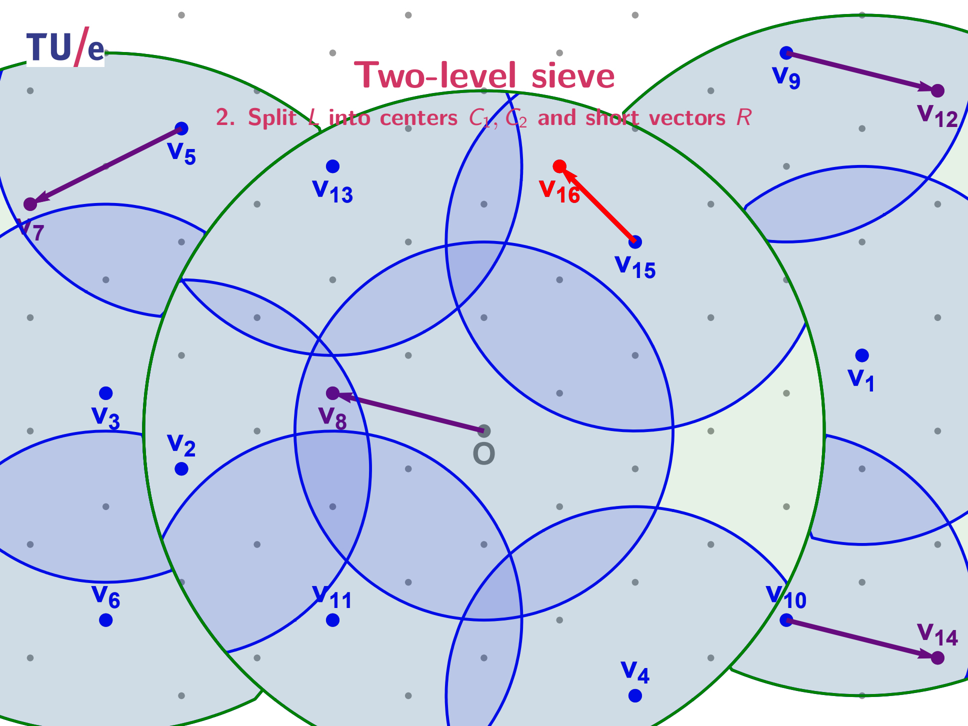
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



Two-level sieve

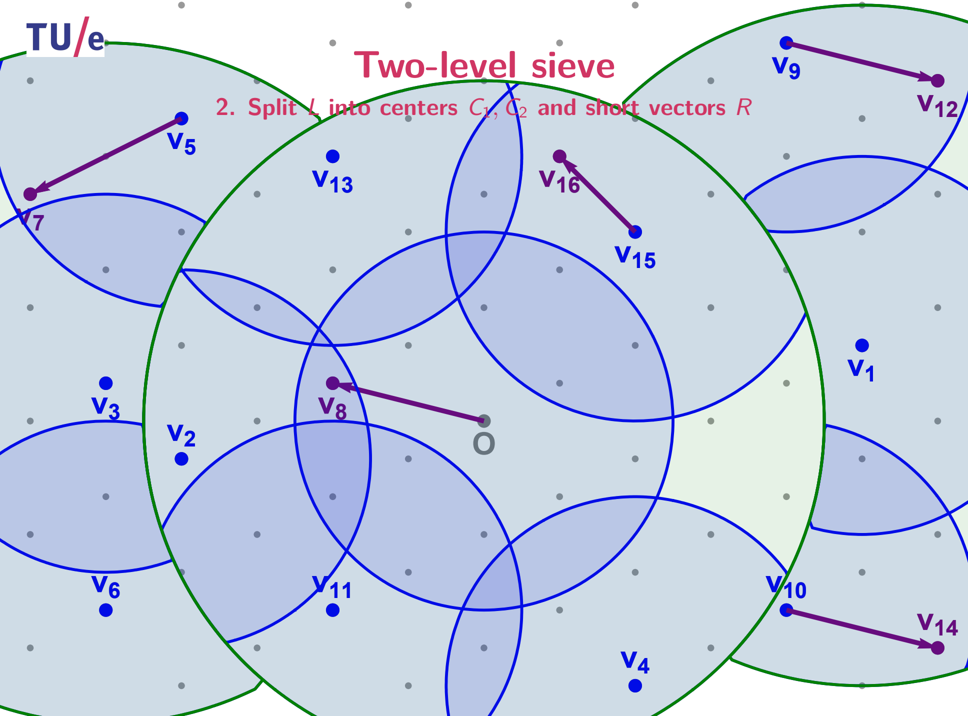
2. Split L into centers C_1, C_2 and short vectors R



TU/e

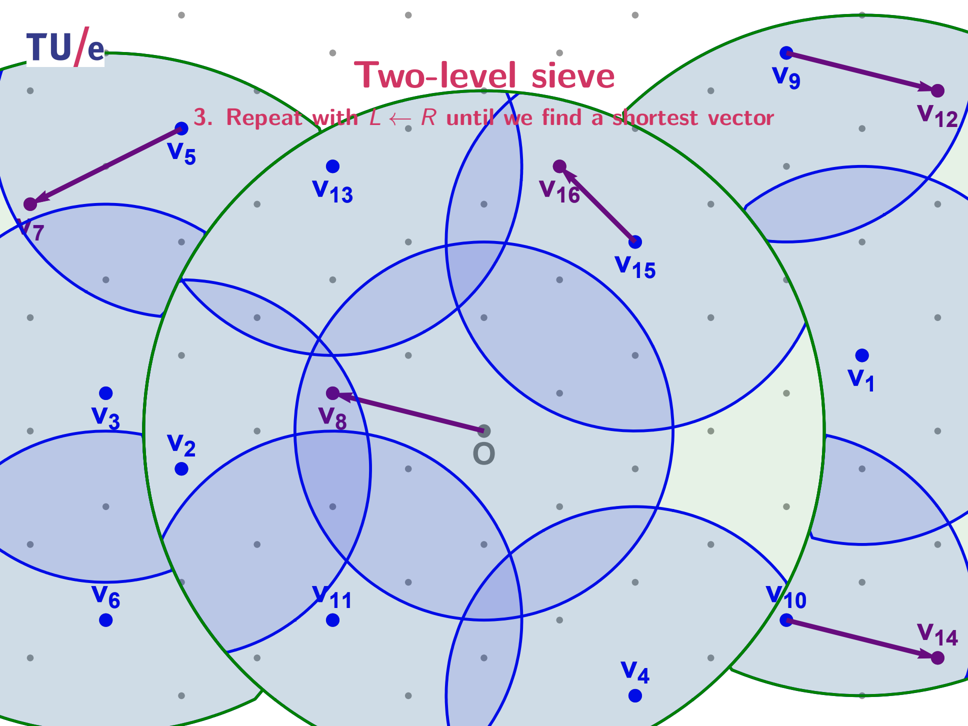
Two-level sieve

2. Split L into centers C_1, C_2 and short vectors R



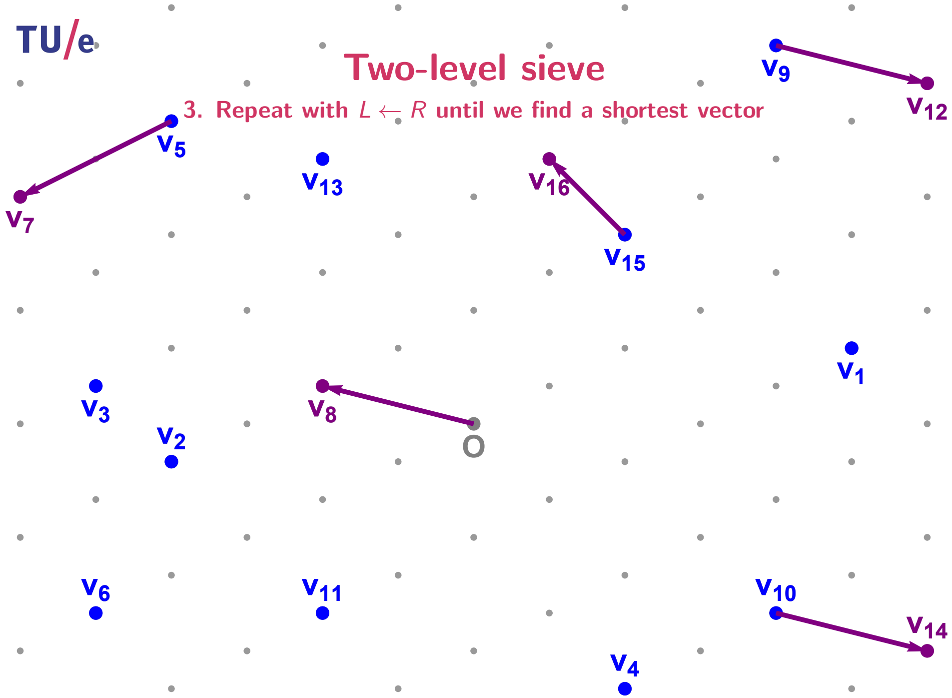
Two-level sieve

3. Repeat with $L \leftarrow R$ until we find a shortest vector



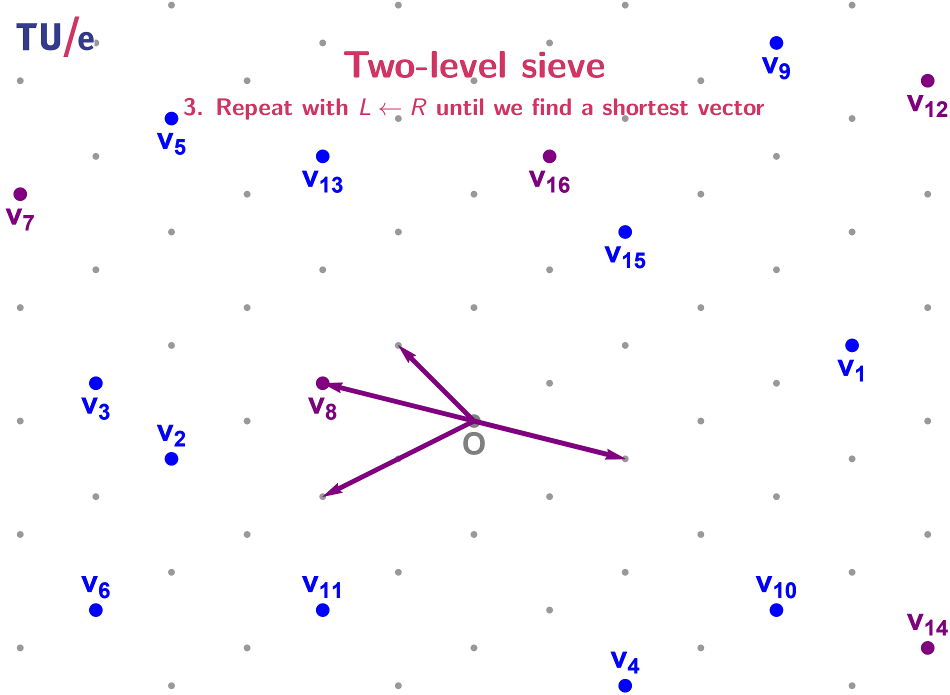
Two-level sieve

3. Repeat with $L \leftarrow R$ until we find a shortest vector



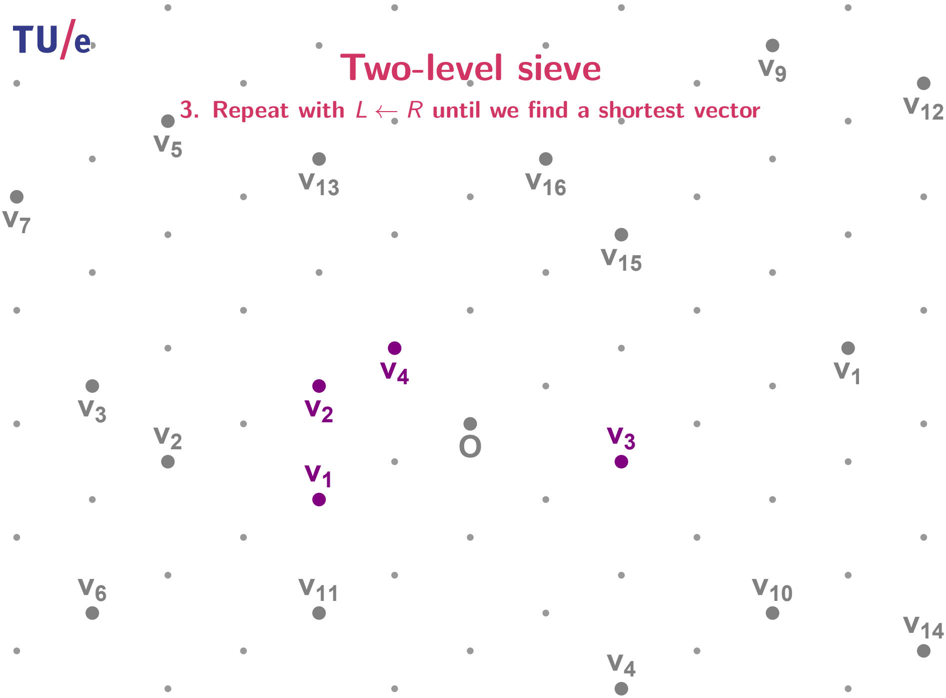
Two-level sieve

3. Repeat with $L \leftarrow R$ until we find a shortest vector



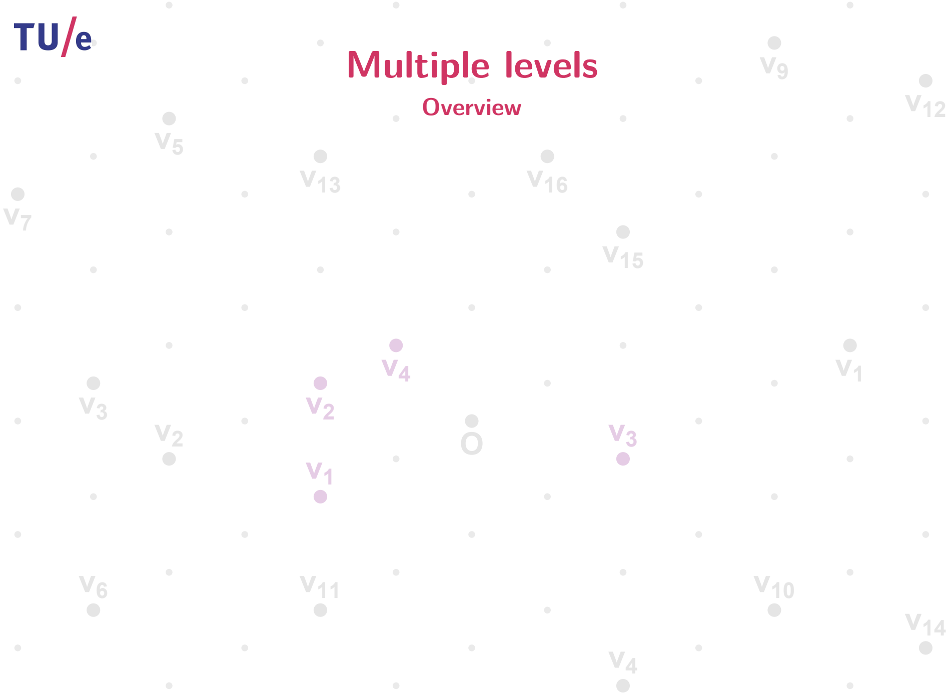
Two-level sieve

3. Repeat with $L \leftarrow R$ until we find a shortest vector



Multiple levels

Overview



Multiple levels

Overview

Heuristic (Nguyen and Vidick, J. Math. Crypt. '08)

The one-level sieve runs in time $2^{0.4150n}$ and space $2^{0.2075n}$.

Multiple levels

Overview

Heuristic (Nguyen and Vidick, J. Math. Crypt. '08)

The one-level sieve runs in time $2^{0.4150n}$ and space $2^{0.2075n}$.

Heuristic (Wang et al., ASIACCS'11)

The two-level sieve runs in time $2^{0.3836n}$ and space $2^{0.2557n}$.

Multiple levels

Overview

Heuristic (Nguyen and Vidick, J. Math. Crypt. '08)

The one-level sieve runs in time $2^{0.4150n}$ and space $2^{0.2075n}$.

Heuristic (Wang et al., ASIACCS'11)

The two-level sieve runs in time $2^{0.3836n}$ and space $2^{0.2557n}$.

Heuristic (Zhang et al., SAC'13)

The three-level sieve runs in time $2^{0.3778n}$ and space $2^{0.2833n}$.

GaussSieve

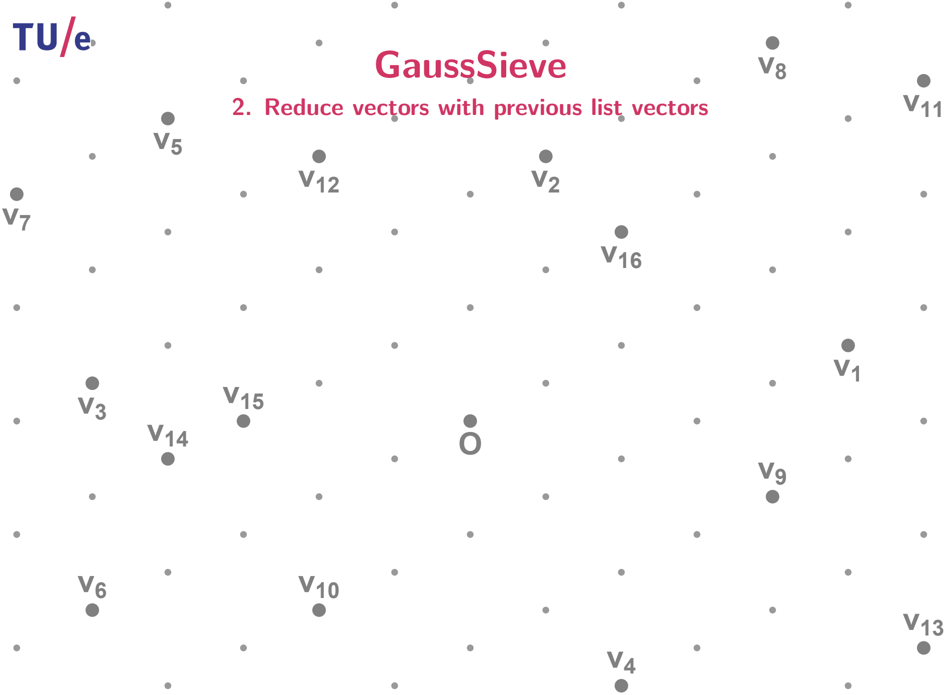
1. Generate random lattice vectors

GaussSieve

1. Generate random lattice vectors

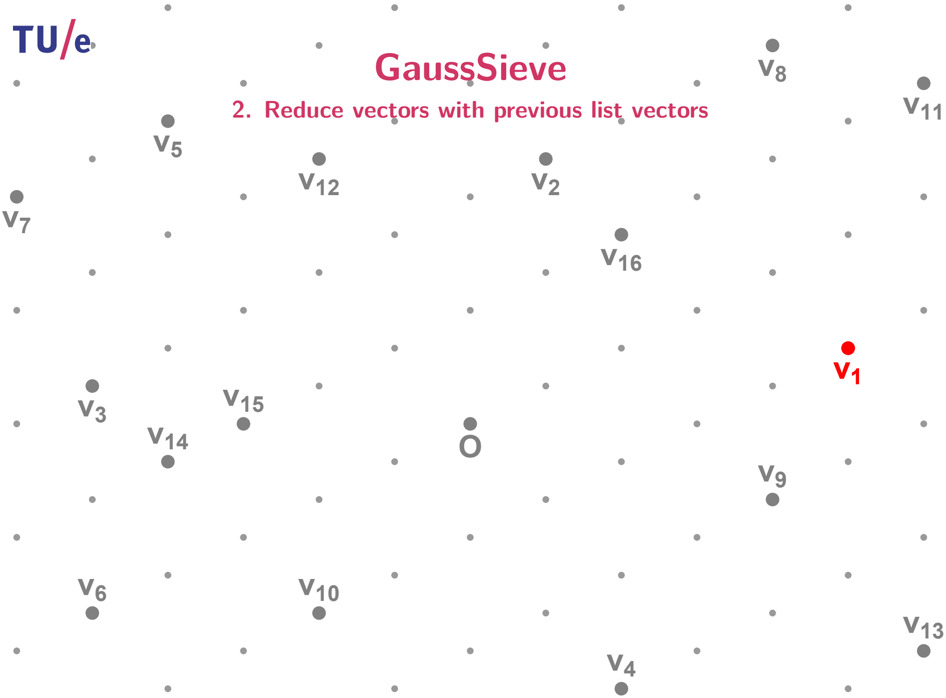
GaussSieve

2. Reduce vectors with previous list vectors



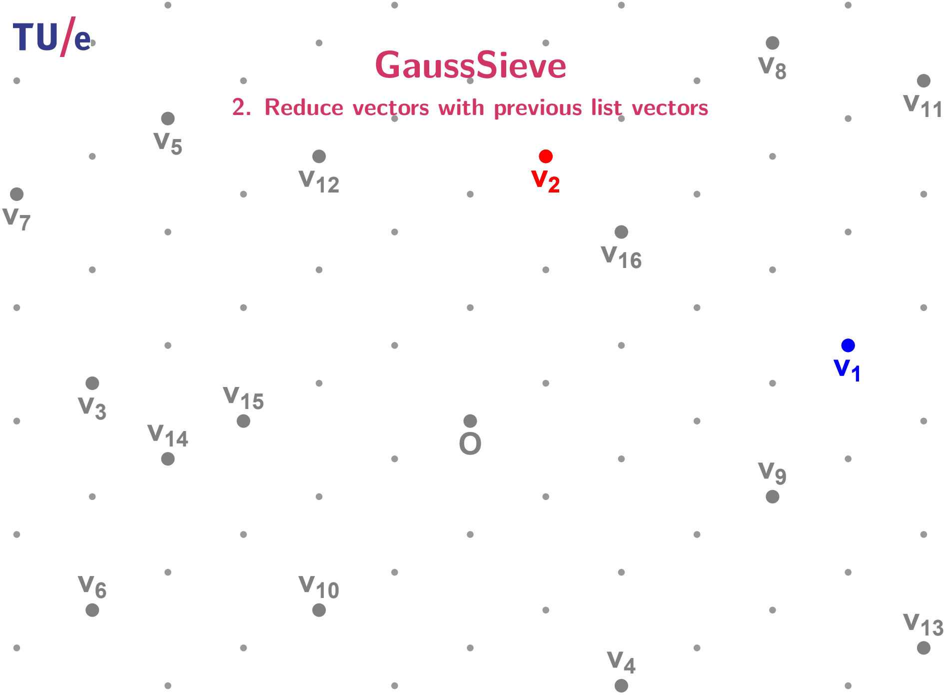
GaussSieve

2. Reduce vectors with previous list vectors



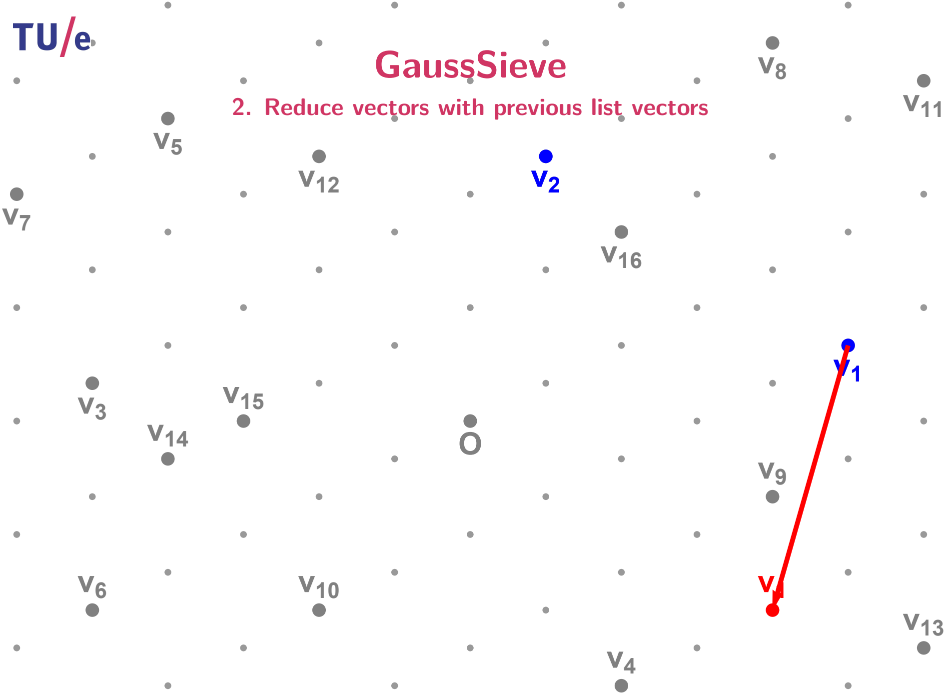
GaussSieve

2. Reduce vectors with previous list vectors



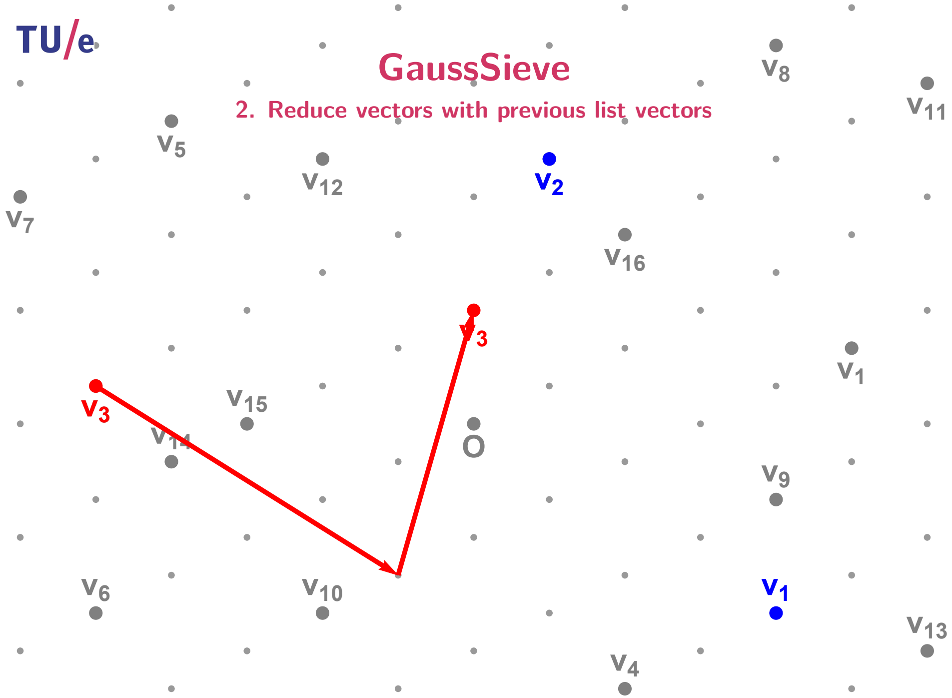
GaussSieve

2. Reduce vectors with previous list vectors



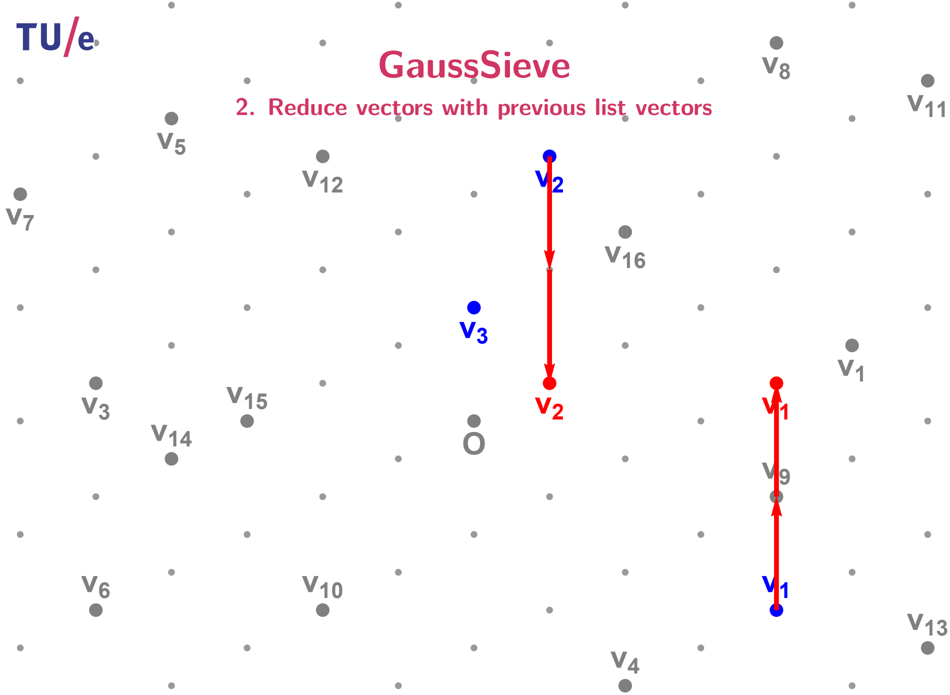
GaussSieve

2. Reduce vectors with previous list vectors



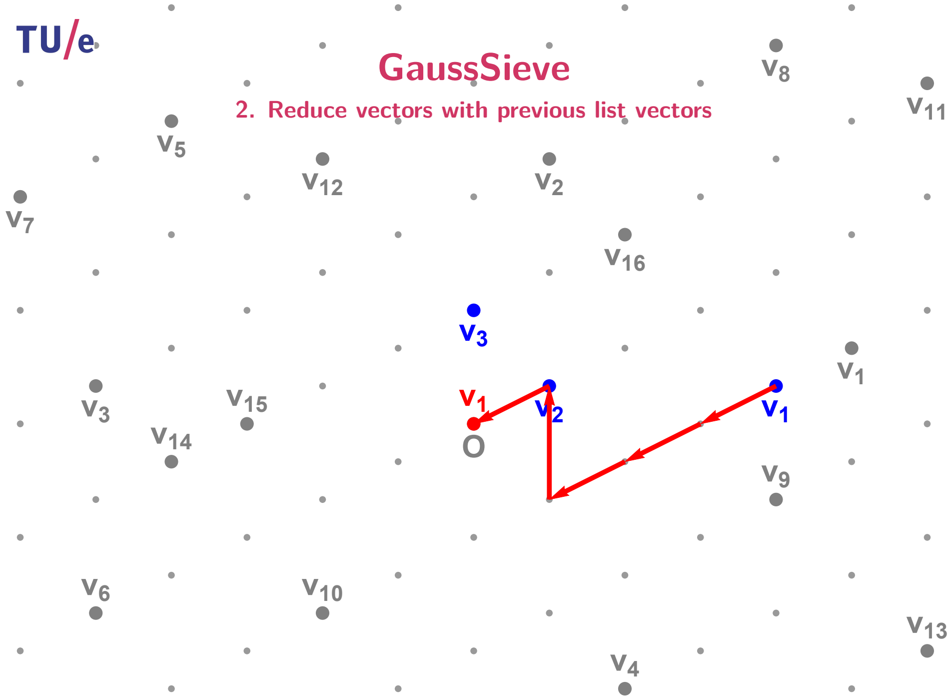
GaussSieve

2. Reduce vectors with previous list vectors



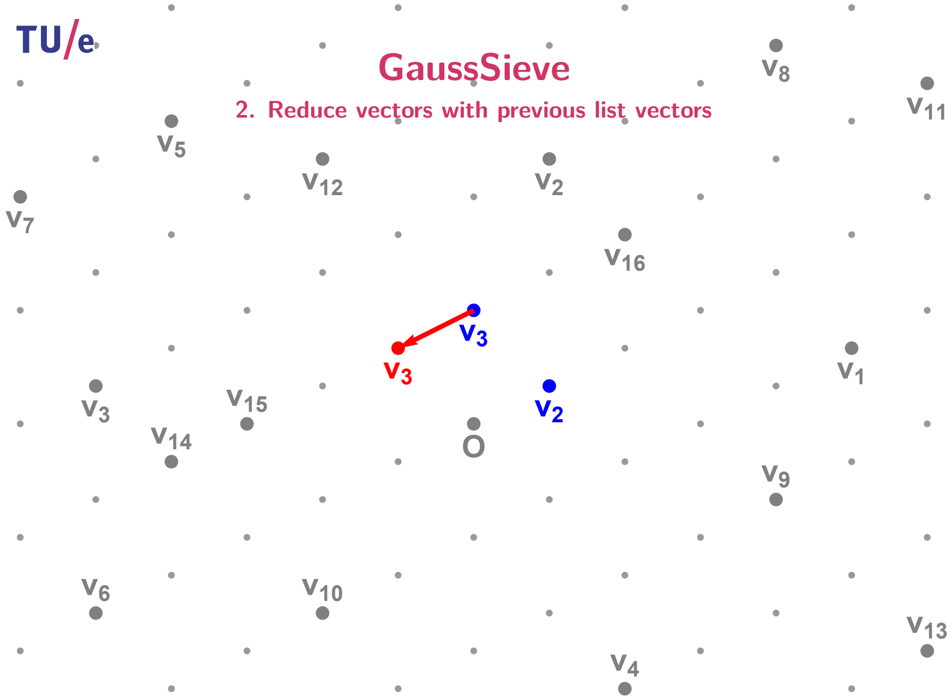
GaussSieve

2. Reduce vectors with previous list vectors



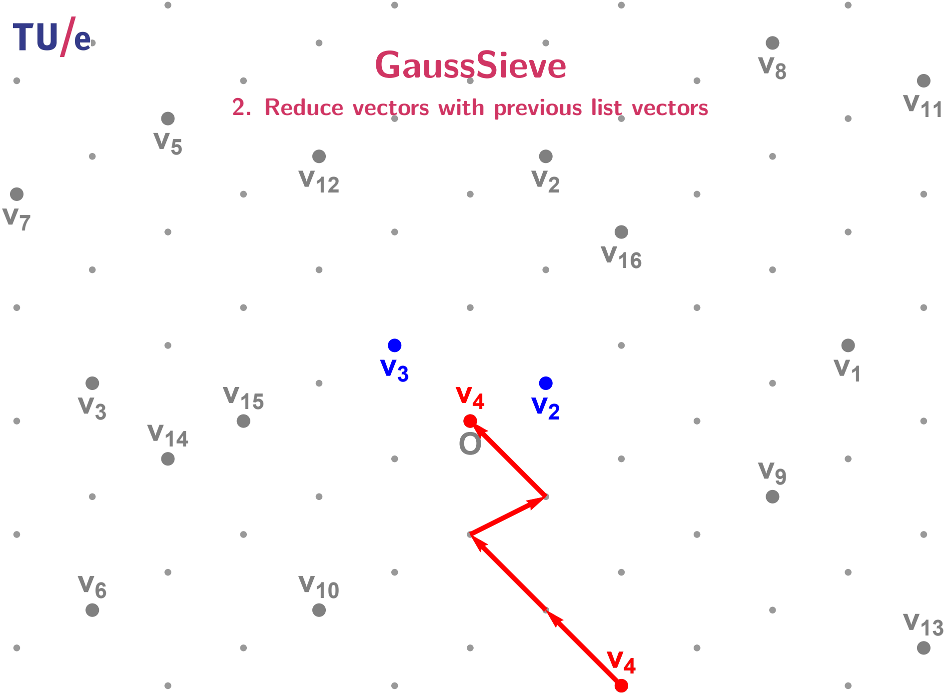
GaussSieve

2. Reduce vectors with previous list vectors



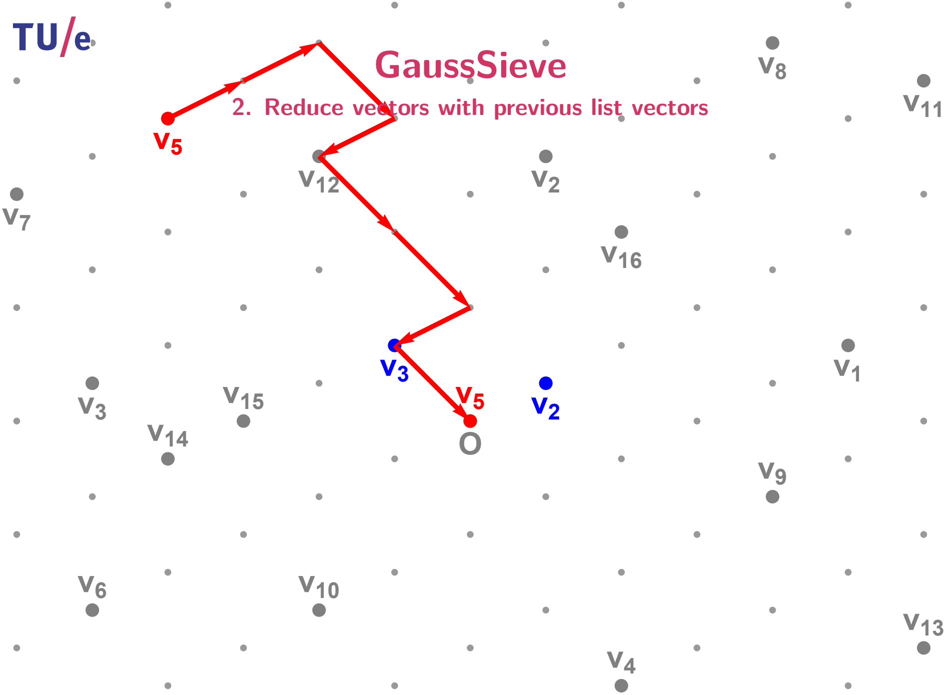
GaussSieve

2. Reduce vectors with previous list vectors



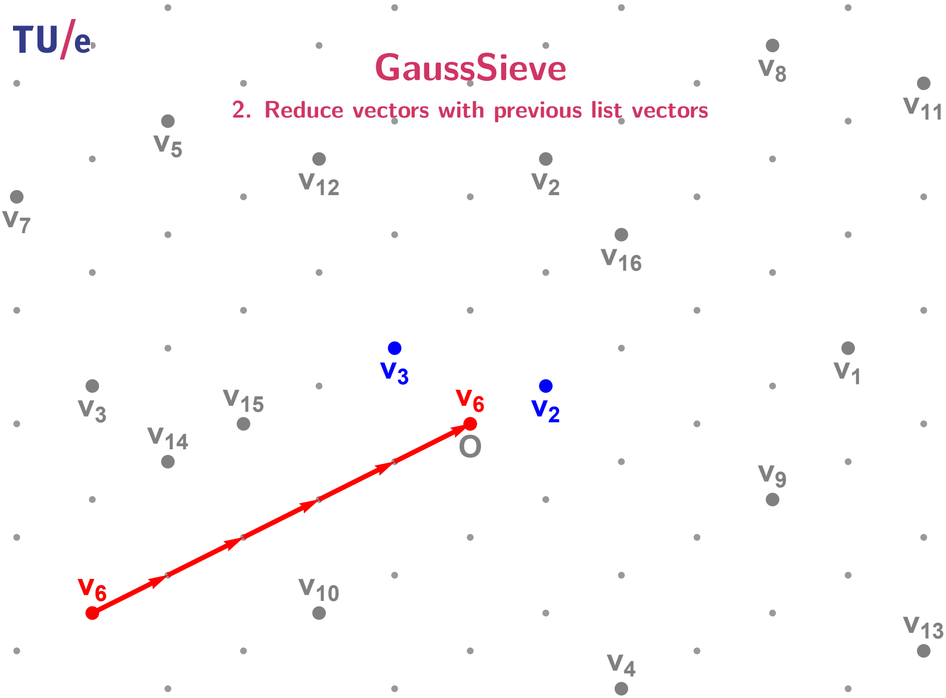
GaussSieve

2. Reduce vectors with previous list vectors



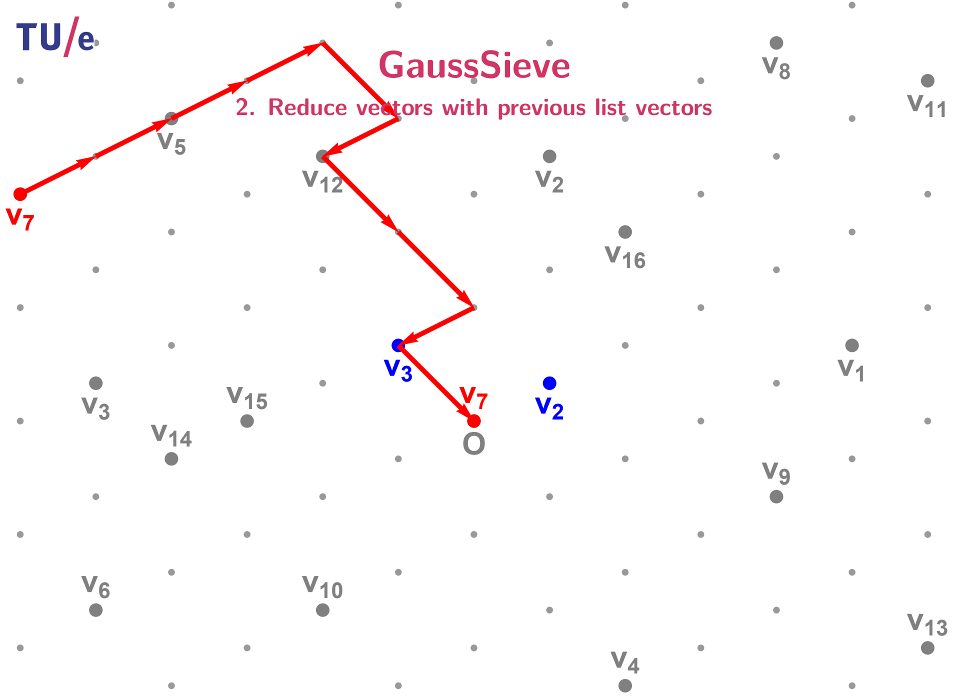
GaussSieve

2. Reduce vectors with previous list vectors



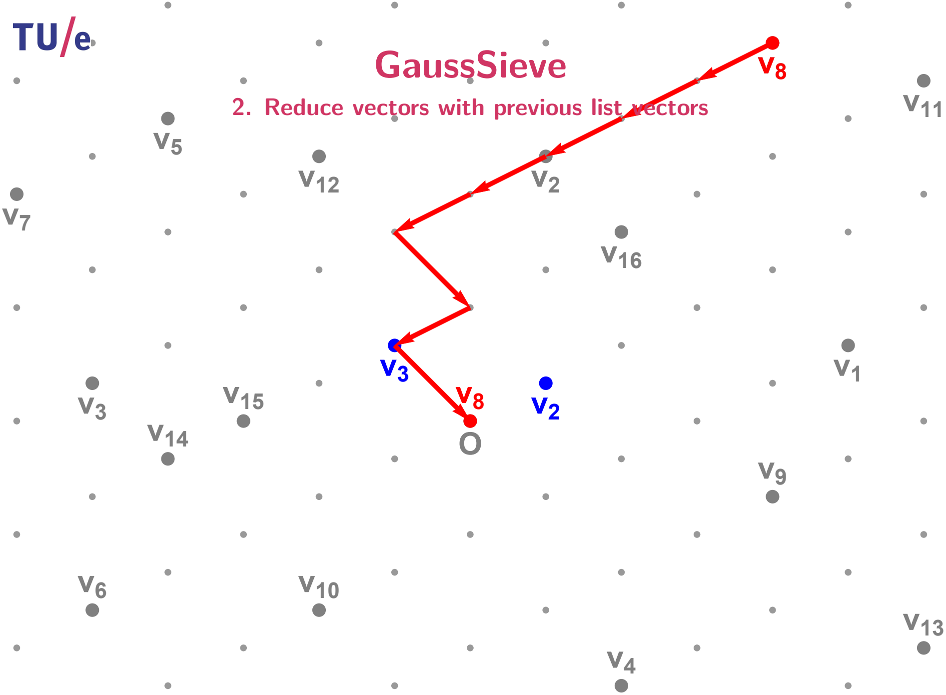
GaussSieve

2. Reduce vectors with previous list vectors



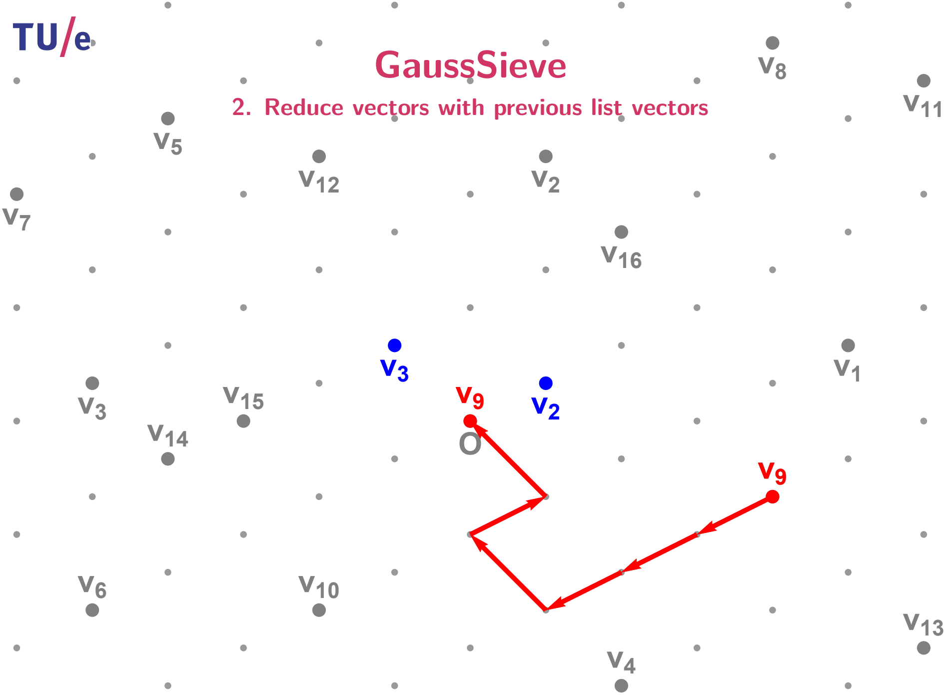
GaussSieve

2. Reduce vectors with previous list vectors



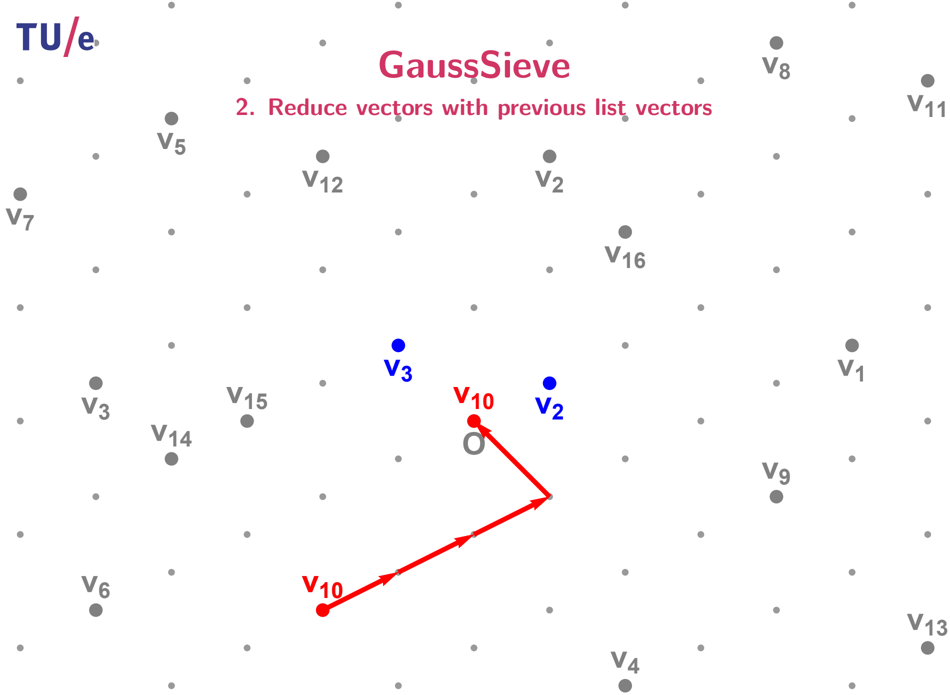
GaussSieve

2. Reduce vectors with previous list vectors



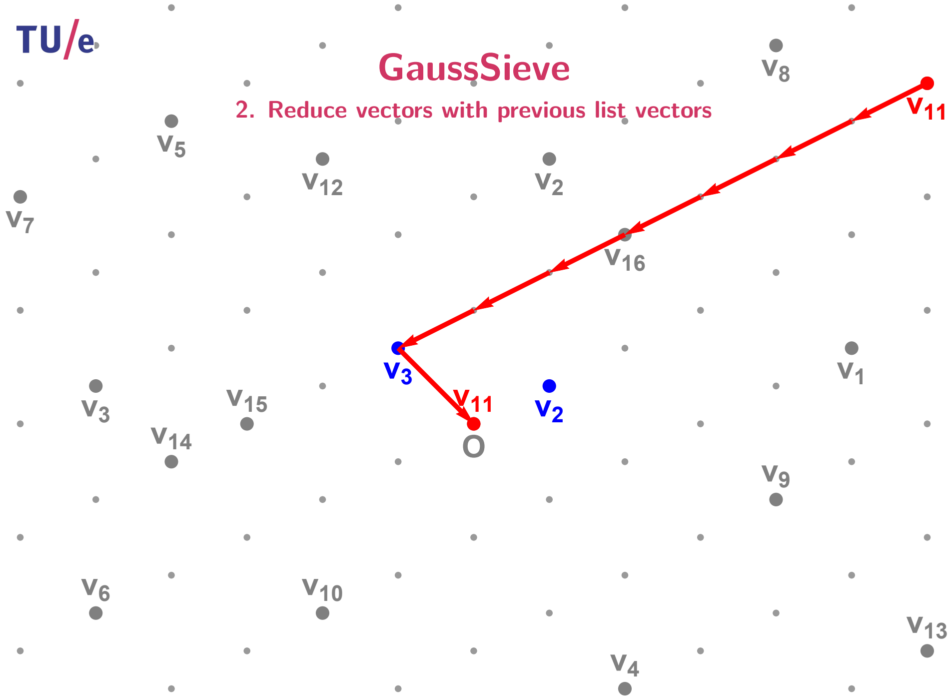
GaussSieve

2. Reduce vectors with previous list vectors



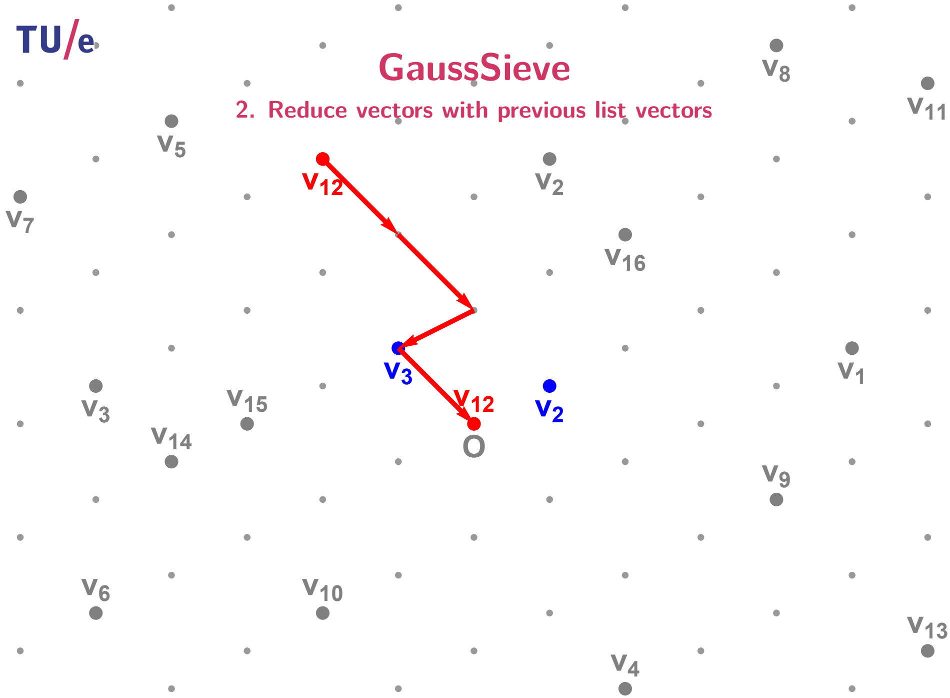
GaussSieve

2. Reduce vectors with previous list vectors



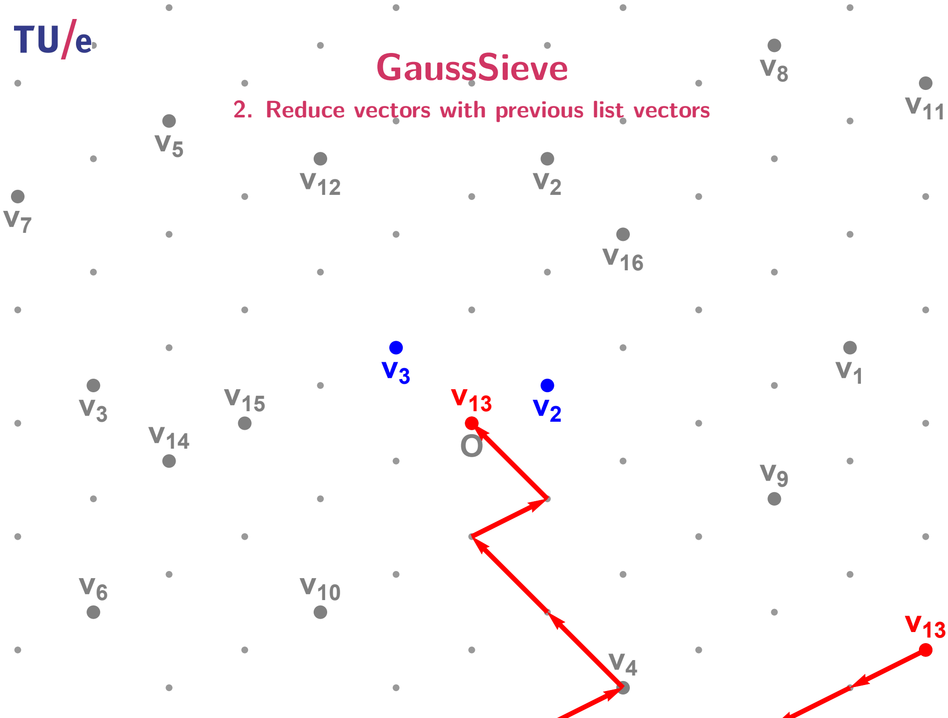
GaussSieve

2. Reduce vectors with previous list vectors



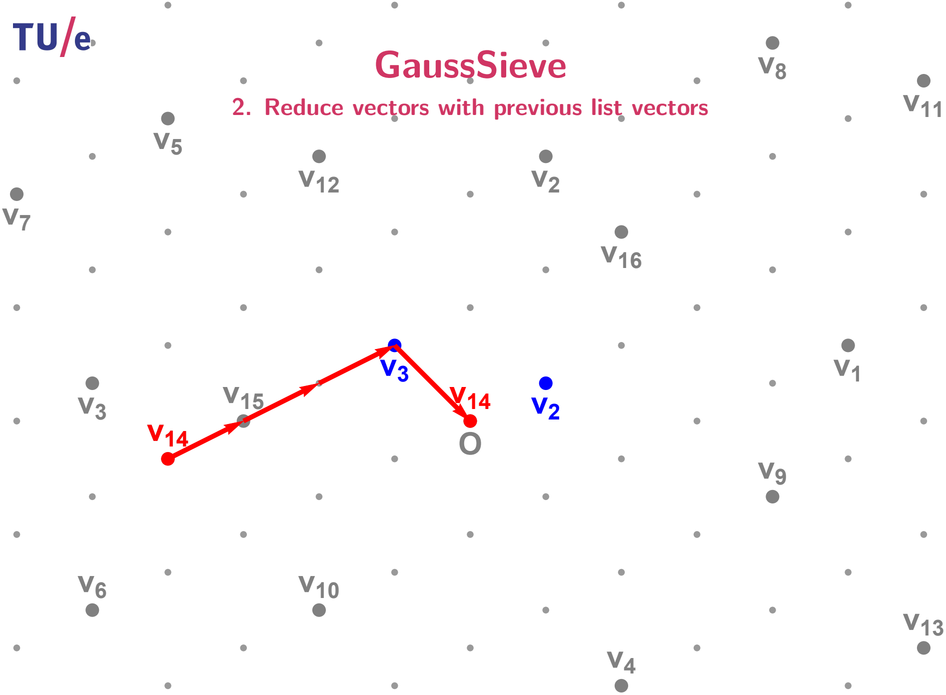
GaussSieve

2. Reduce vectors with previous list vectors



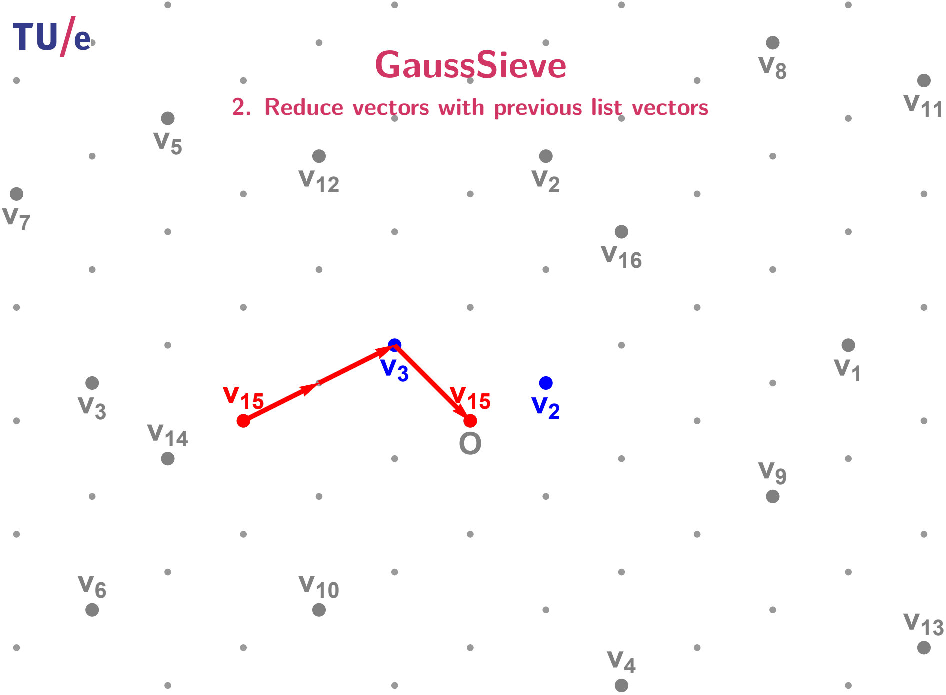
GaussSieve

2. Reduce vectors with previous list vectors



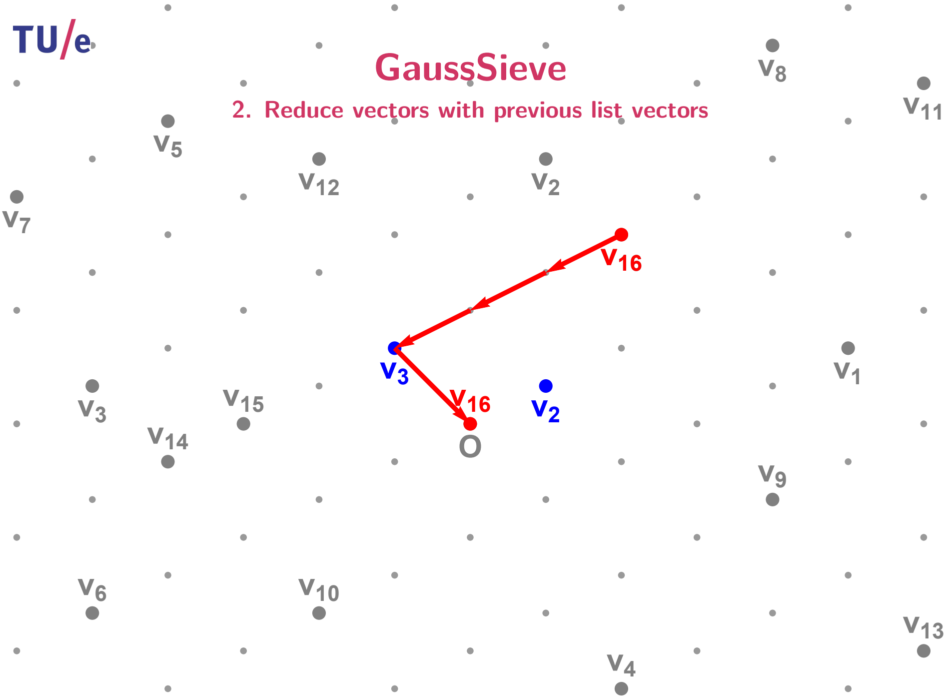
GaussSieve

2. Reduce vectors with previous list vectors



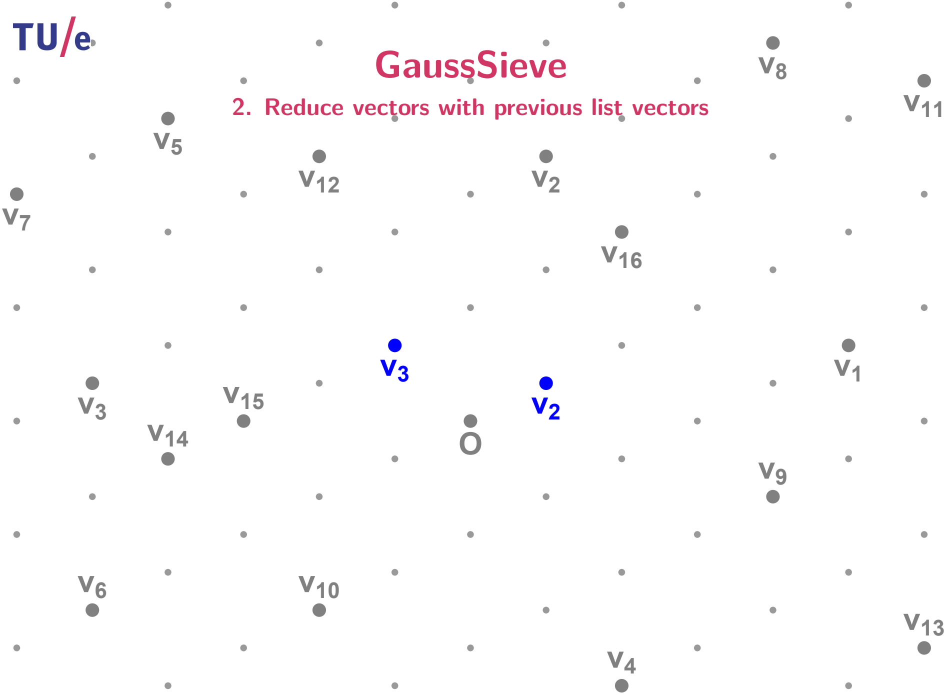
GaussSieve

2. Reduce vectors with previous list vectors



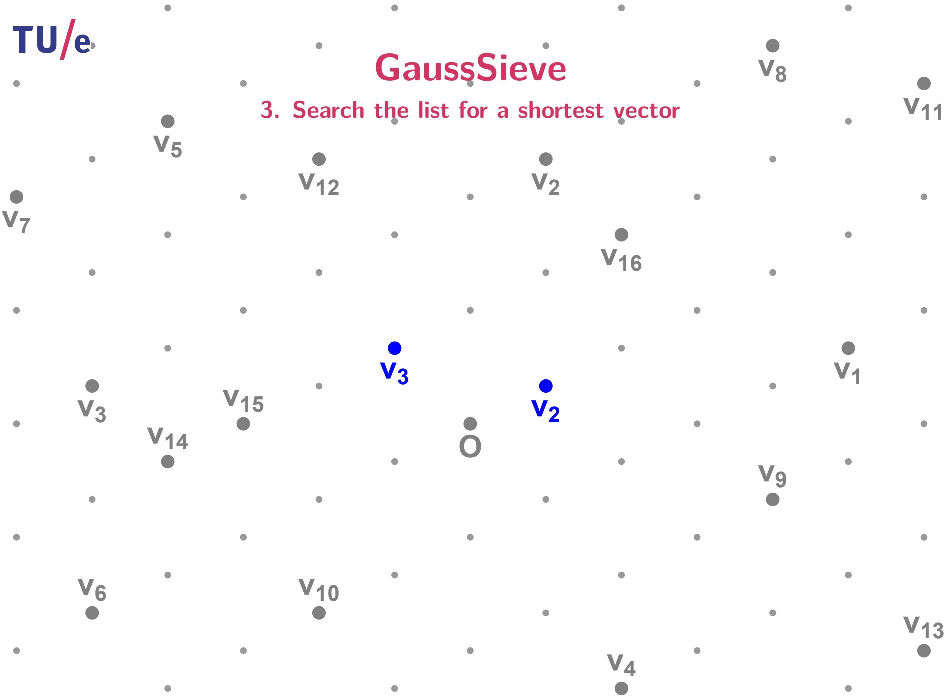
GaussSieve

2. Reduce vectors with previous list vectors



GaussSieve

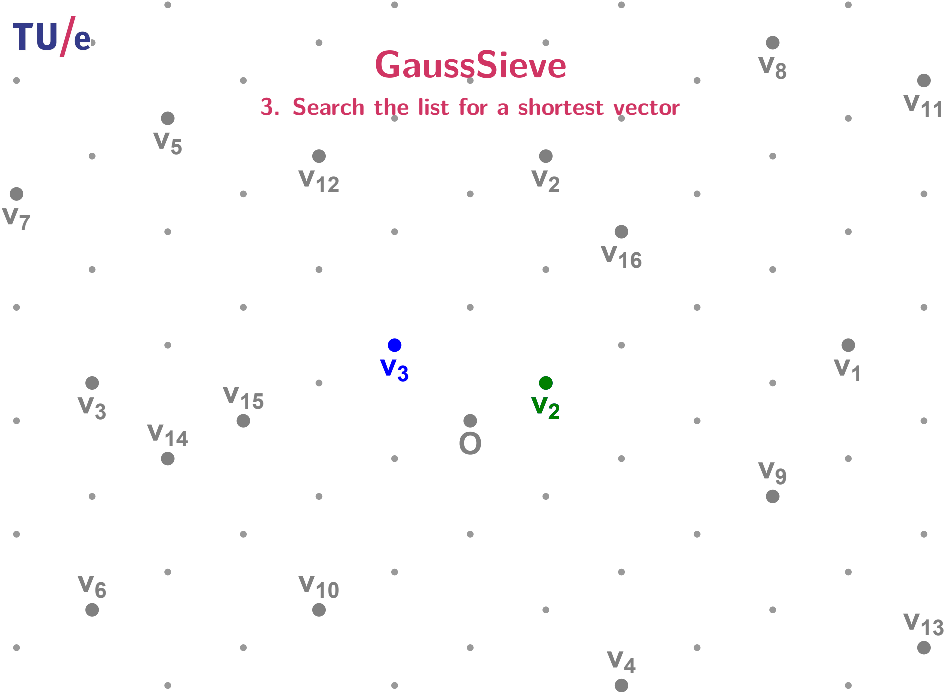
3. Search the list for a shortest vector



TU/e

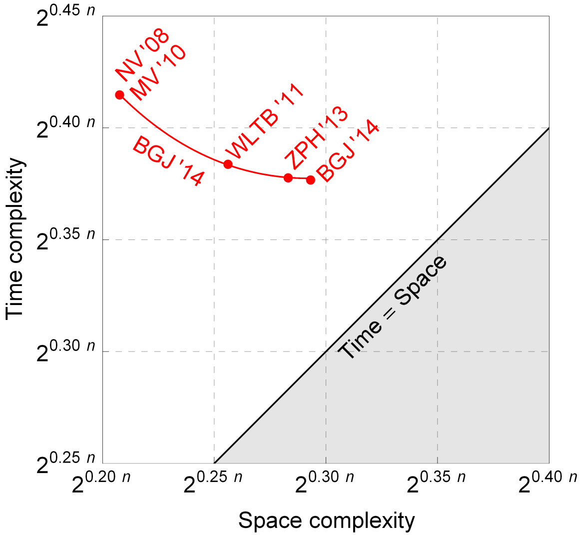
GaussSieve

3. Search the list for a shortest vector



Sieving

Space/time trade-off



Outline

Lattices

Enumeration algorithms
- Fincke-Pohst enumeration
- Kannan enumeration
- Pruning the enumeration tree

The Voronoi cell algorithm

Sieving algorithms
- Nguyen-Vidick sieve
- Multiple levels
- GaussSieve

Sieving using angular locality-sensitive hashing
- Nguyen-Vidick sieve
- GaussSieve

Locality-sensitive hashing

Introduction

"The key idea is to use hash functions such that the probability of collision is much higher for objects that are close to each other than for those that are far apart."

– Indyk and Motwani, *STOC'98*

Locality-sensitive hashing

Introduction

"The key idea is to use hash functions such that the probability of collision is much higher for objects that are close to each other than for those that are far apart."

– Indyk and Motwani, *STOC'98*

"Unfortunately, the estimated improvement is too small to outweigh the increase of the constants for dimensions of practical interest. [...] We implemented the LSH algorithm of Datar et al. and found that it performed only marginally better than the naive algorithm for dimensions higher than 50*."*

– Nguyen and Vidick, *J. Math. Crypt. '08*

Nguyen-Vidick sieve with LSH

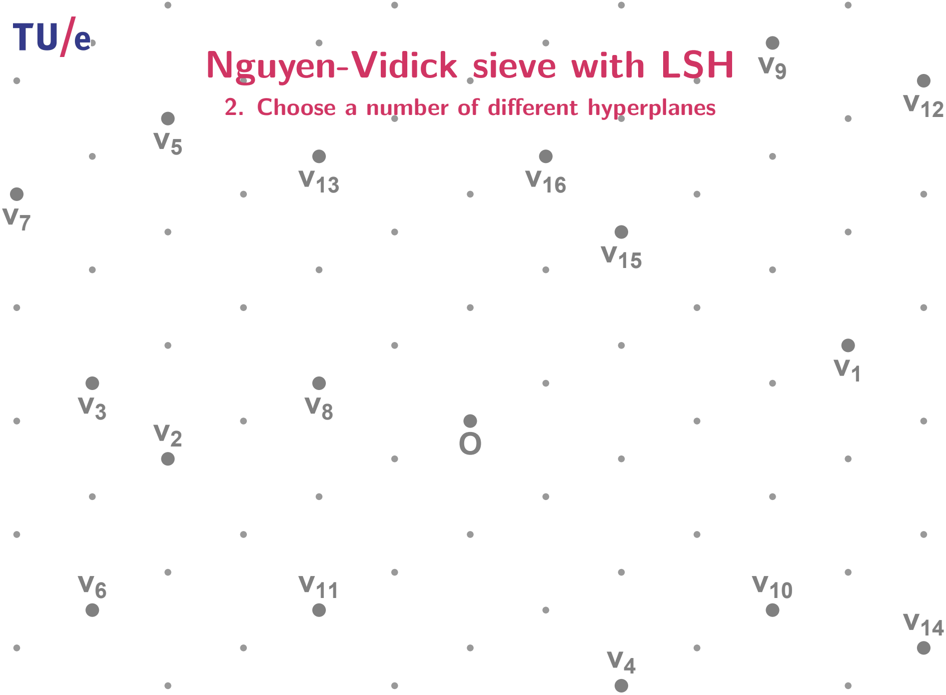
1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve with LSH

1. Sample a list L of random lattice vectors

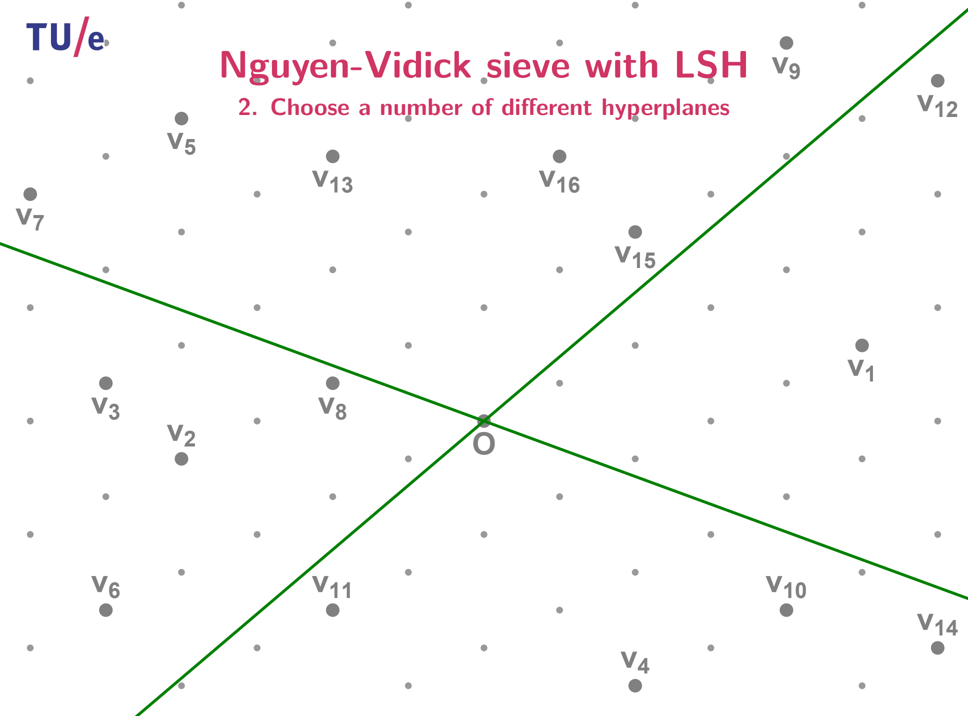
Nguyen-Vidick sieve with LSH

2. Choose a number of different hyperplanes



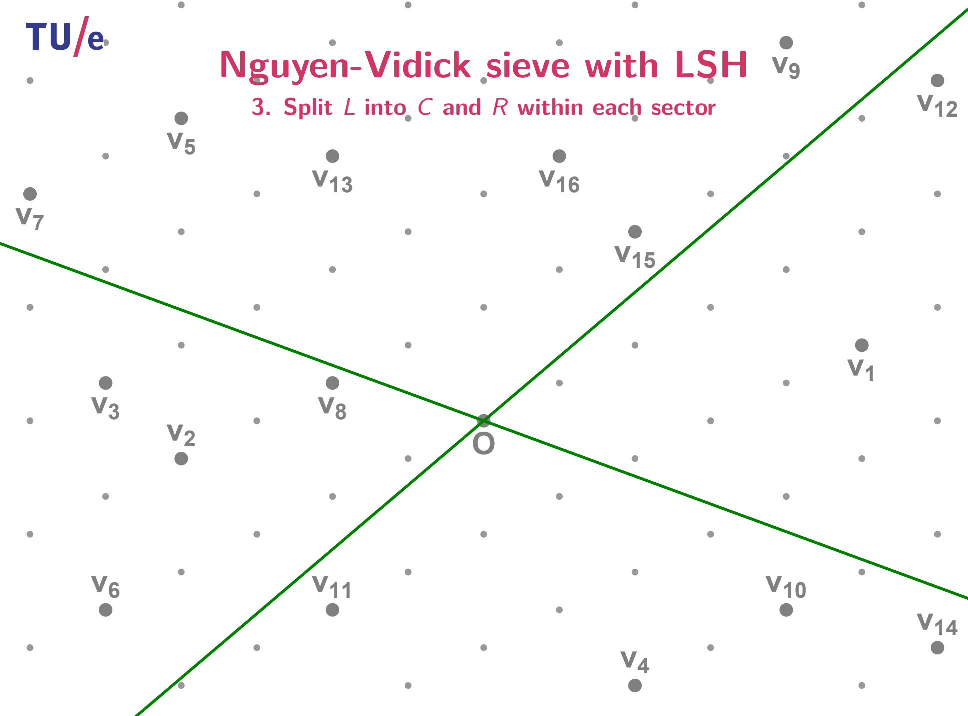
Nguyen-Vidick sieve with LSH

2. Choose a number of different hyperplanes



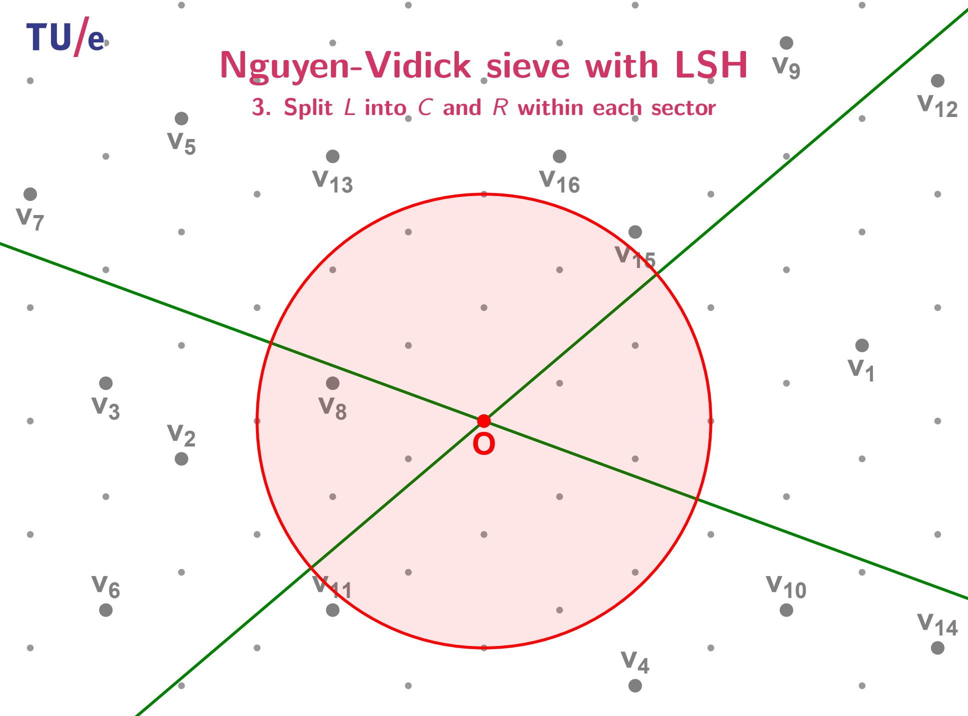
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



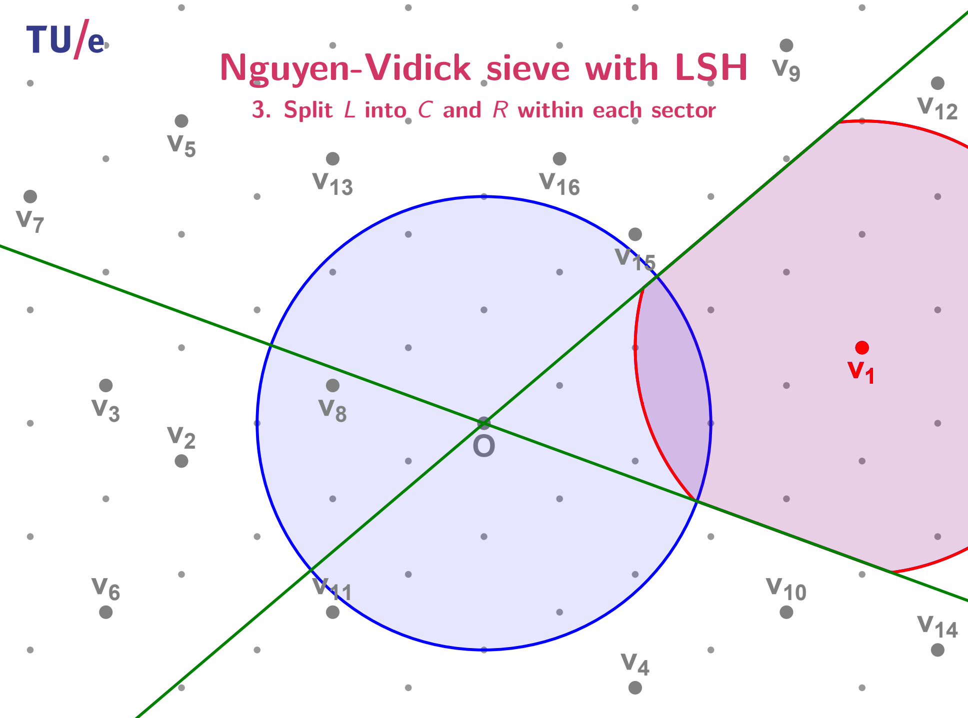
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



Nguyen-Vidick sieve with LSH

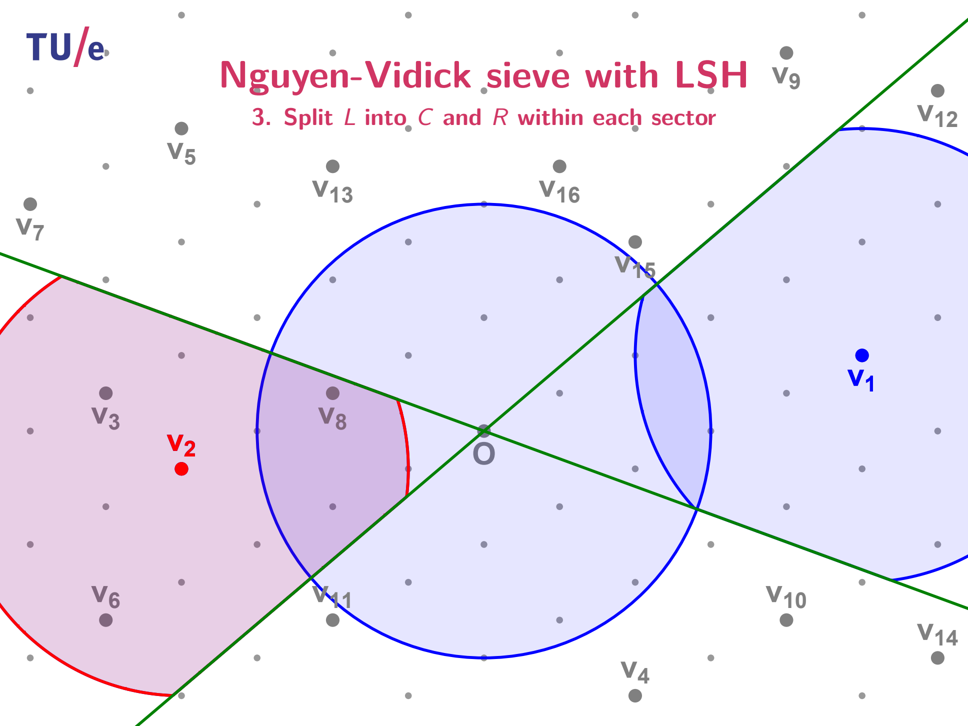
3. Split L into C and R within each sector



TU/e

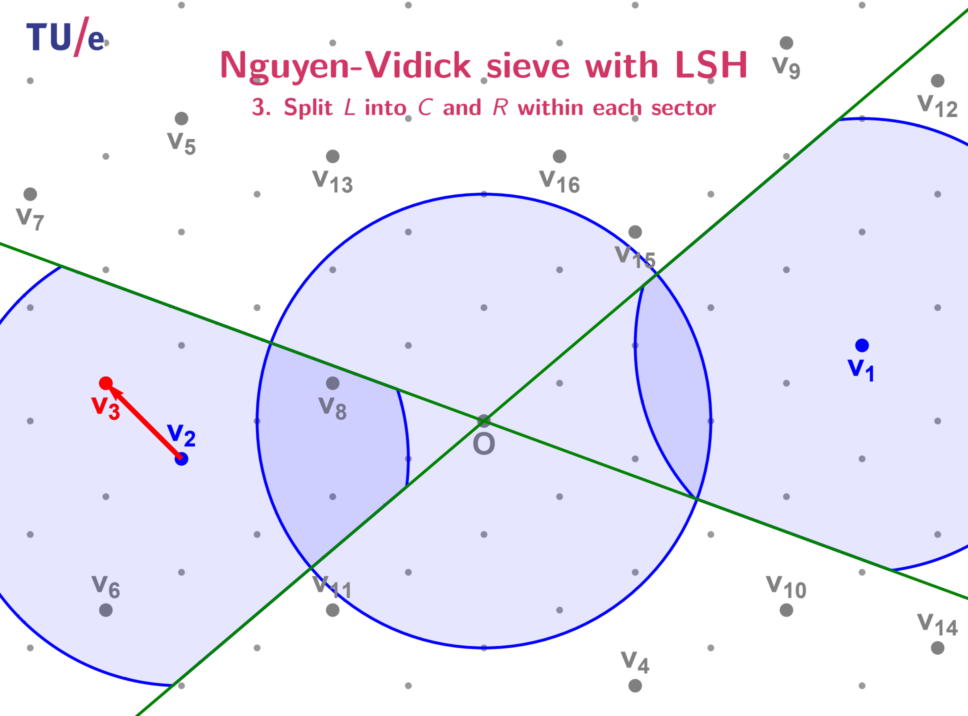
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



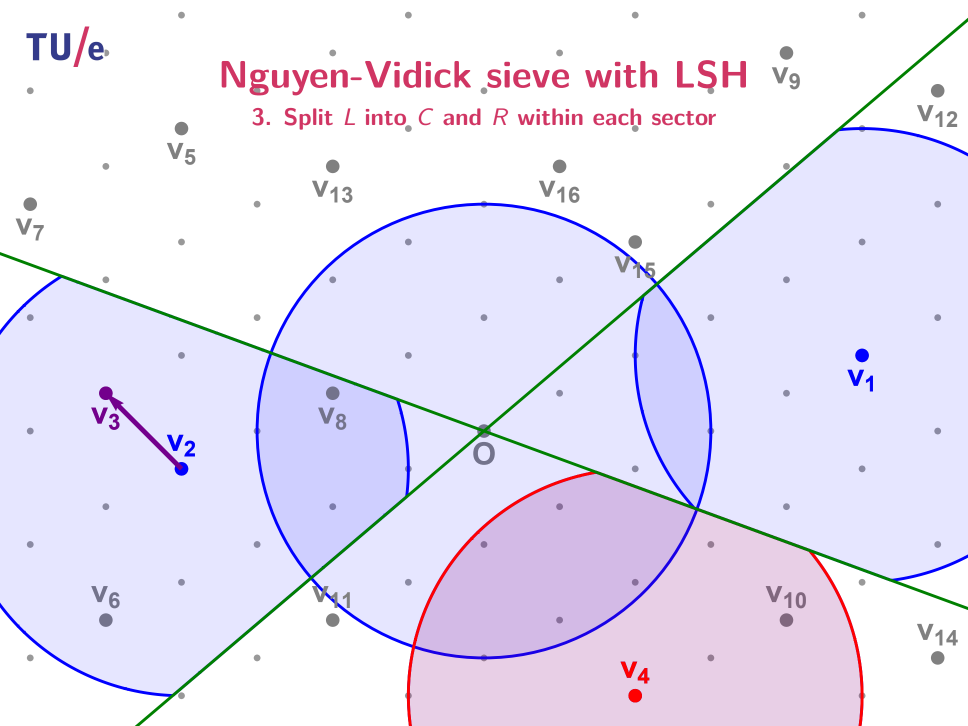
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



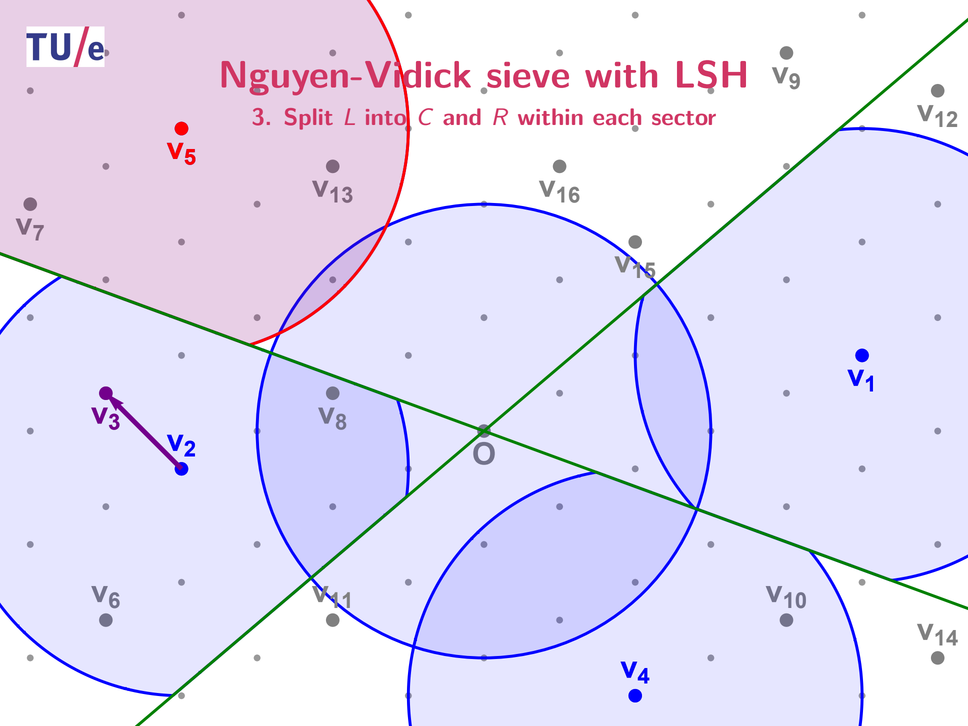
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



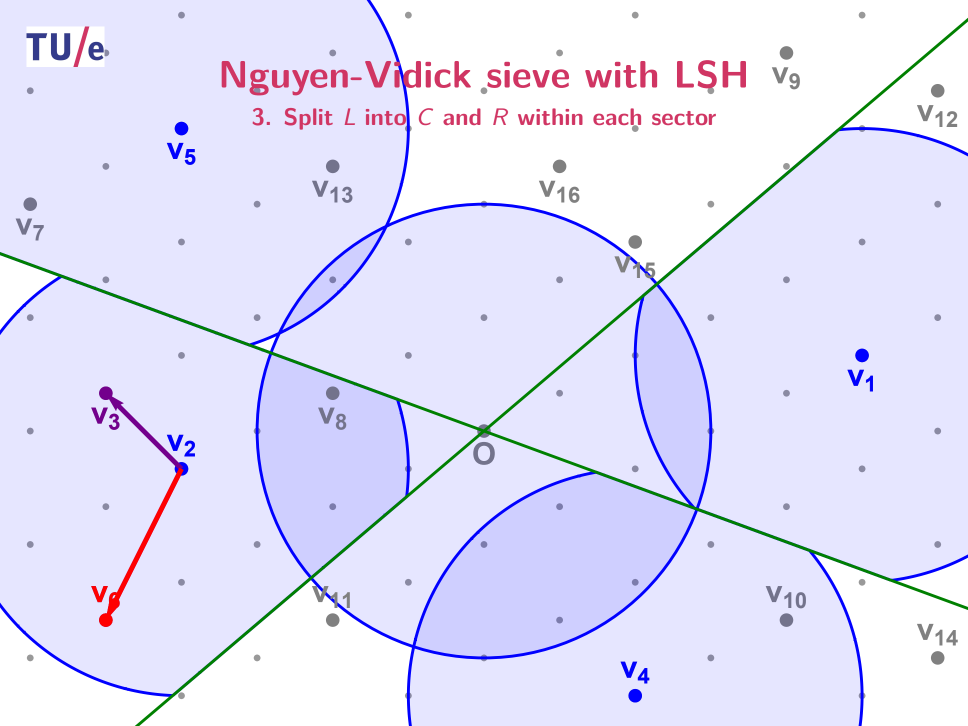
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



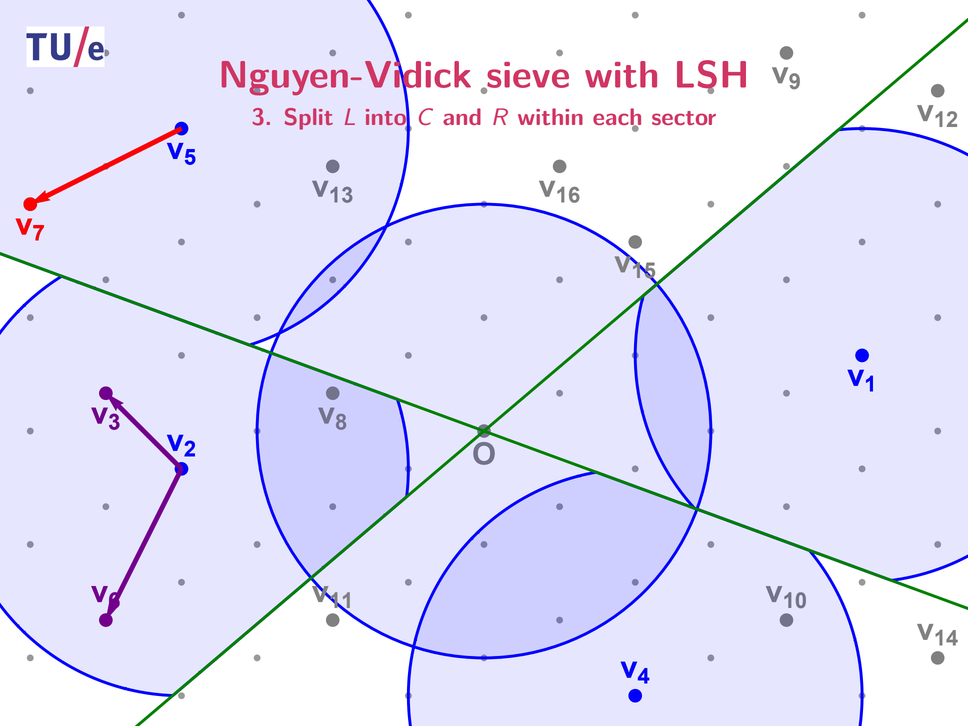
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



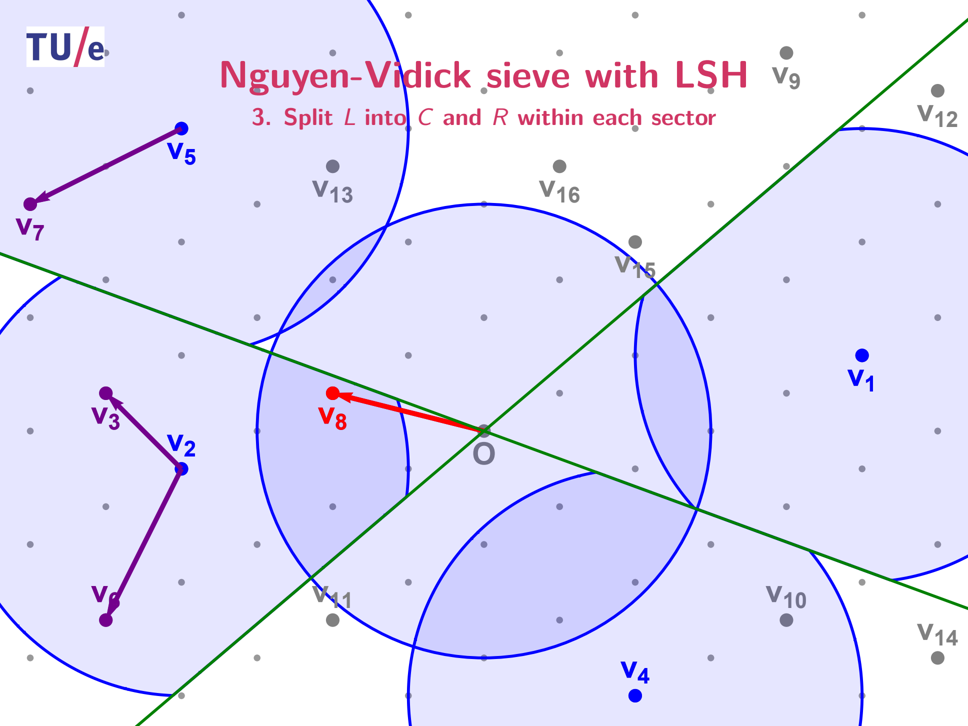
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



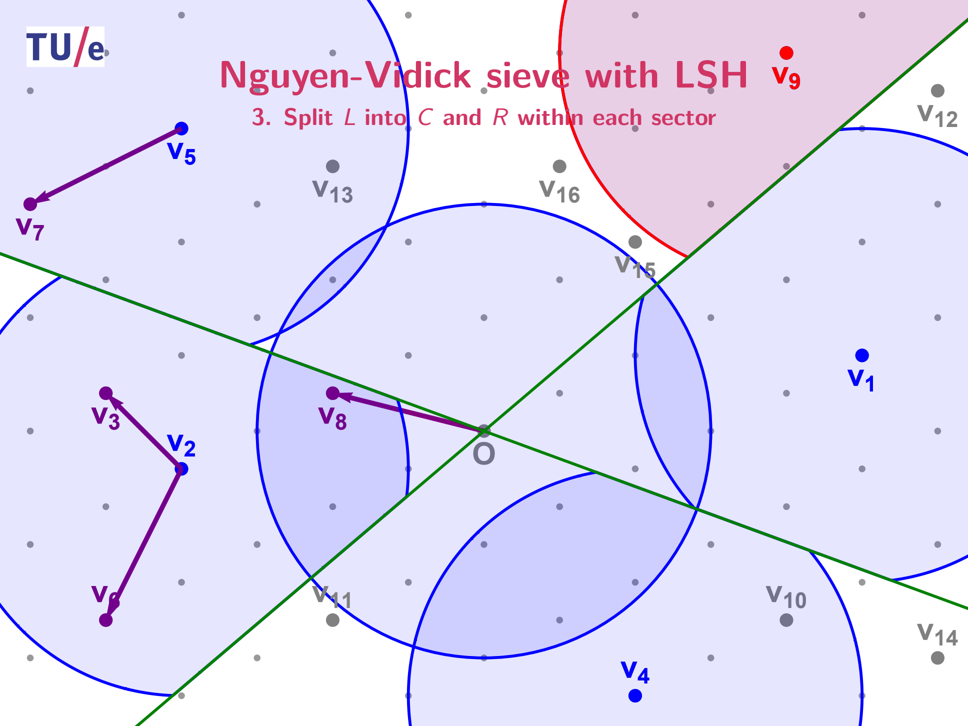
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



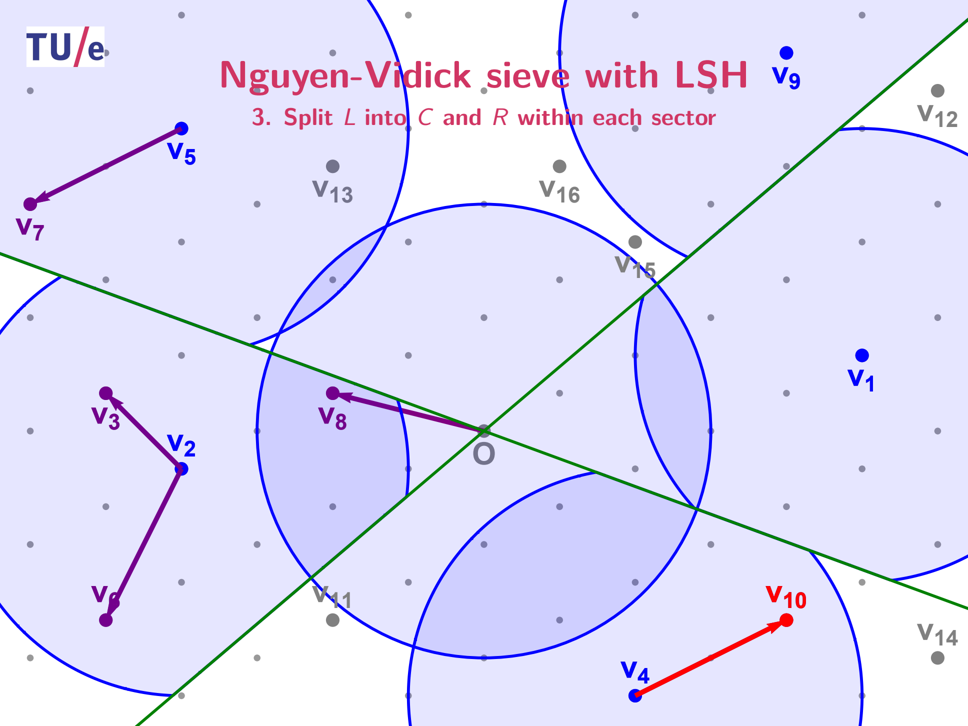
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



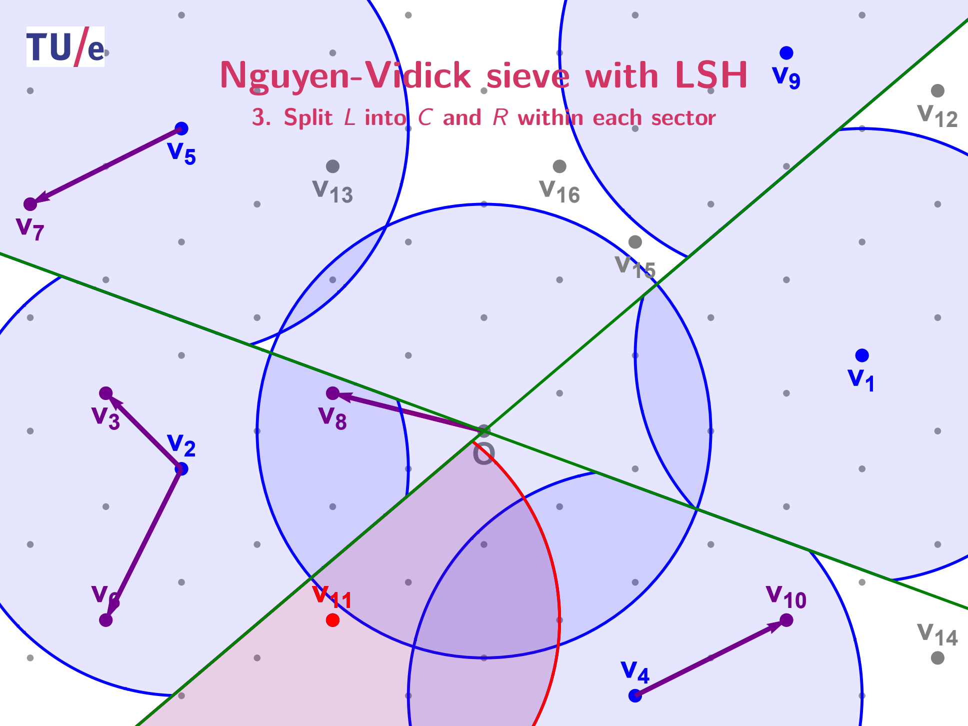
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



Nguyen-Vidick sieve with LSH

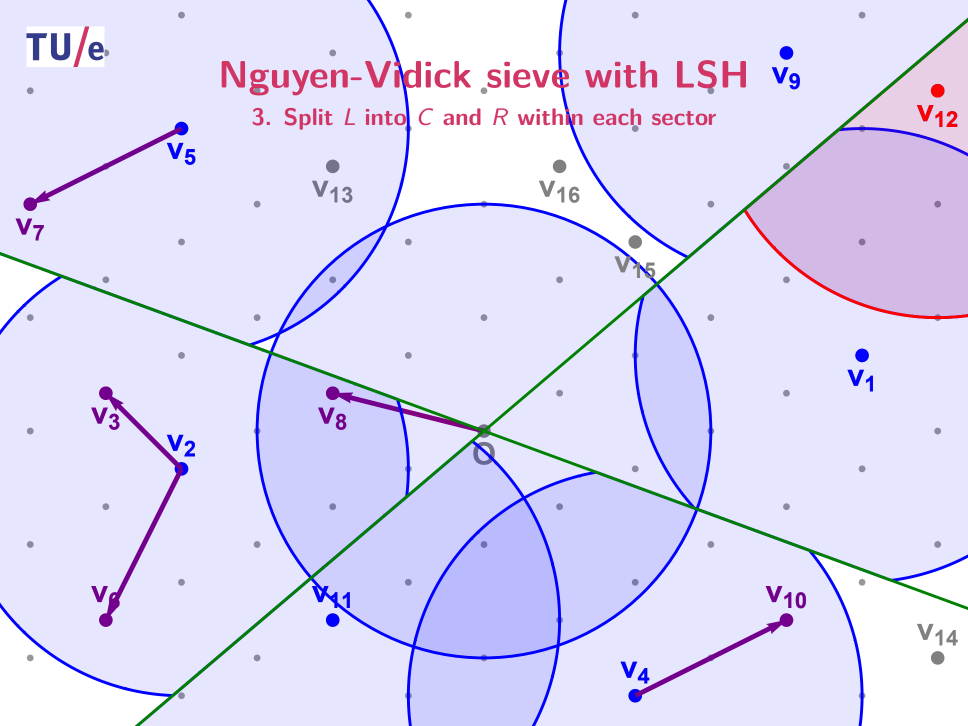
3. Split L into C and R within each sector



TU/e

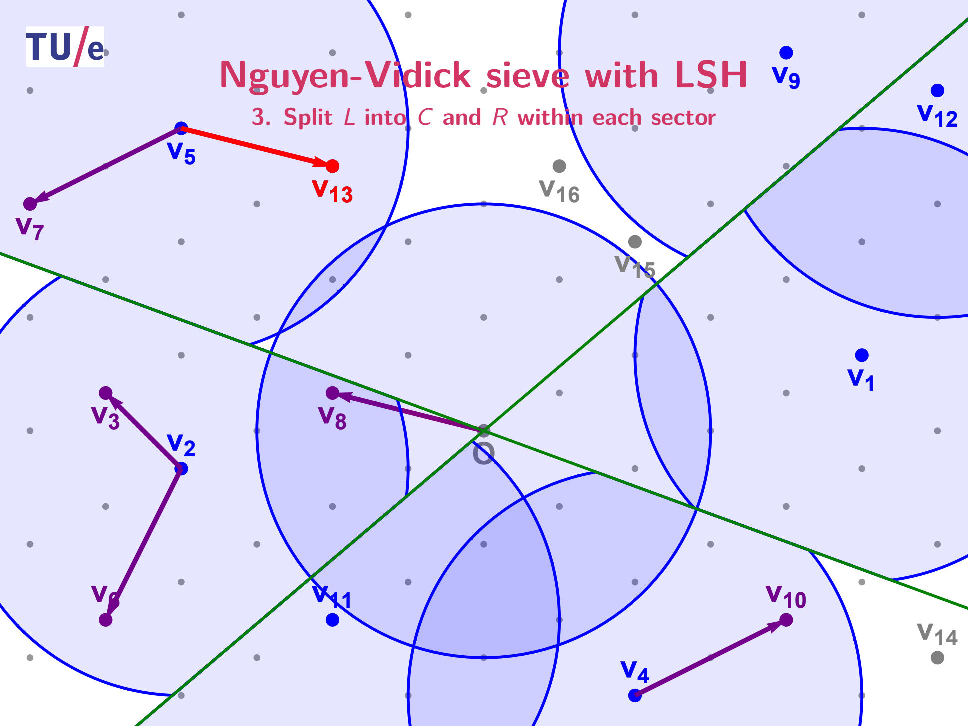
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



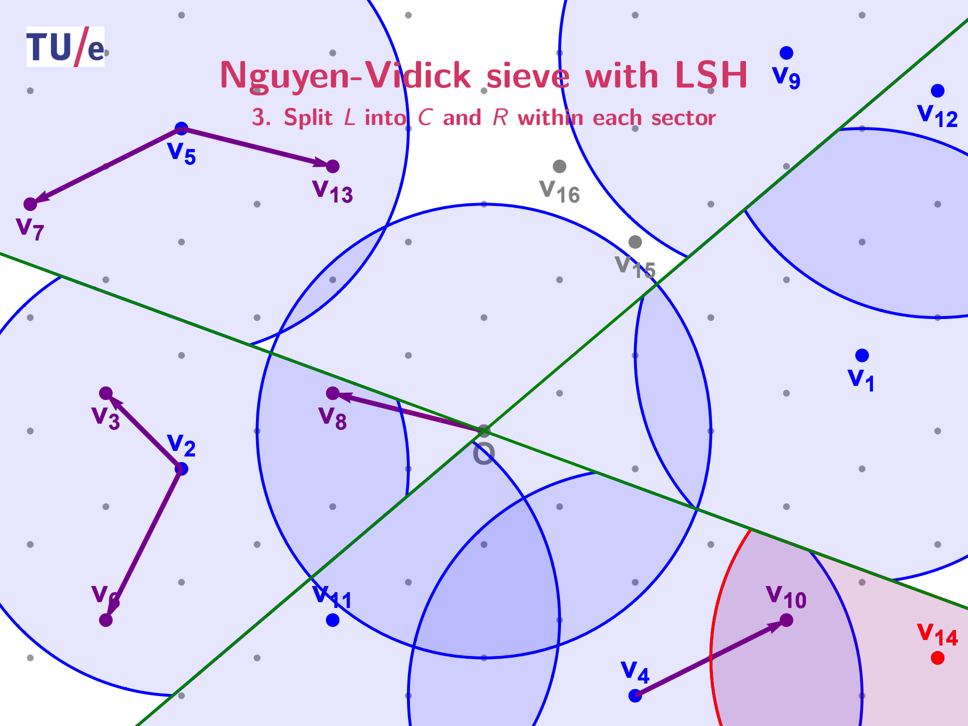
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



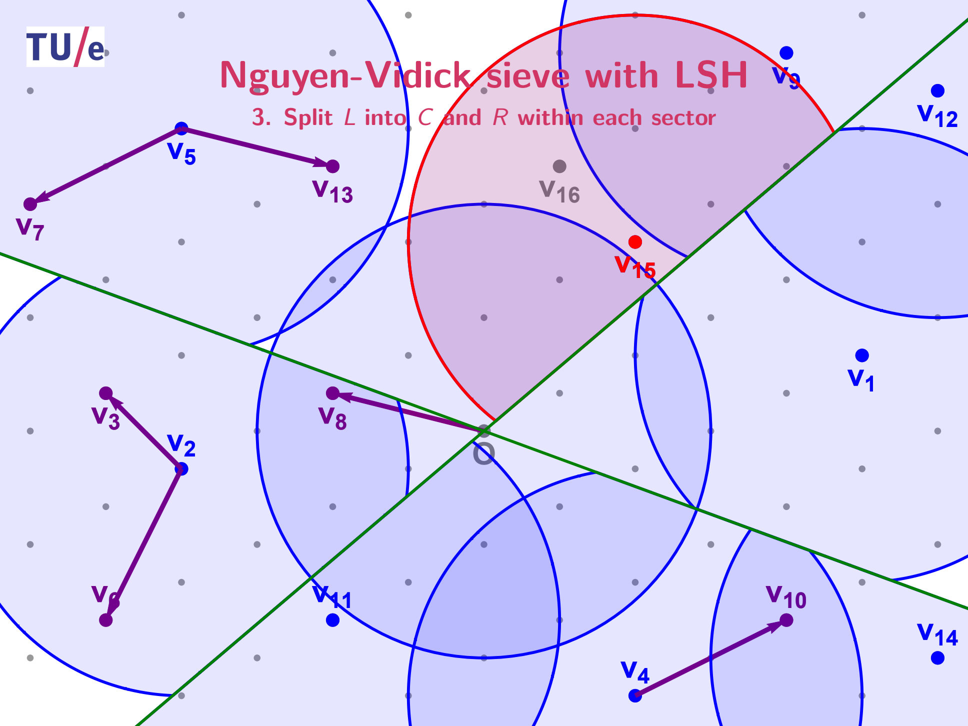
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



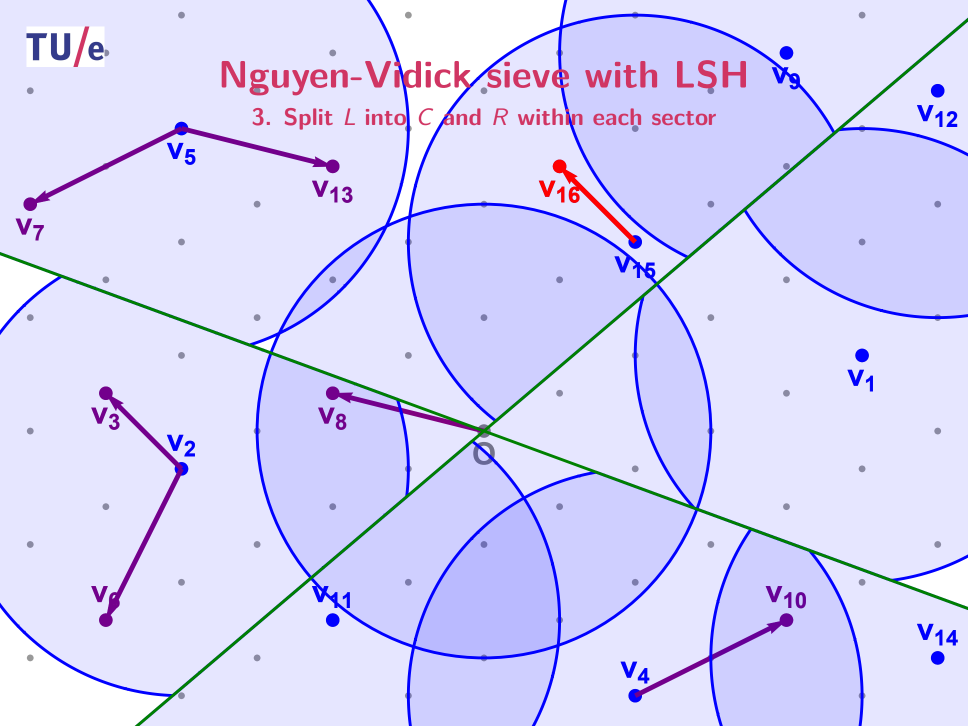
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



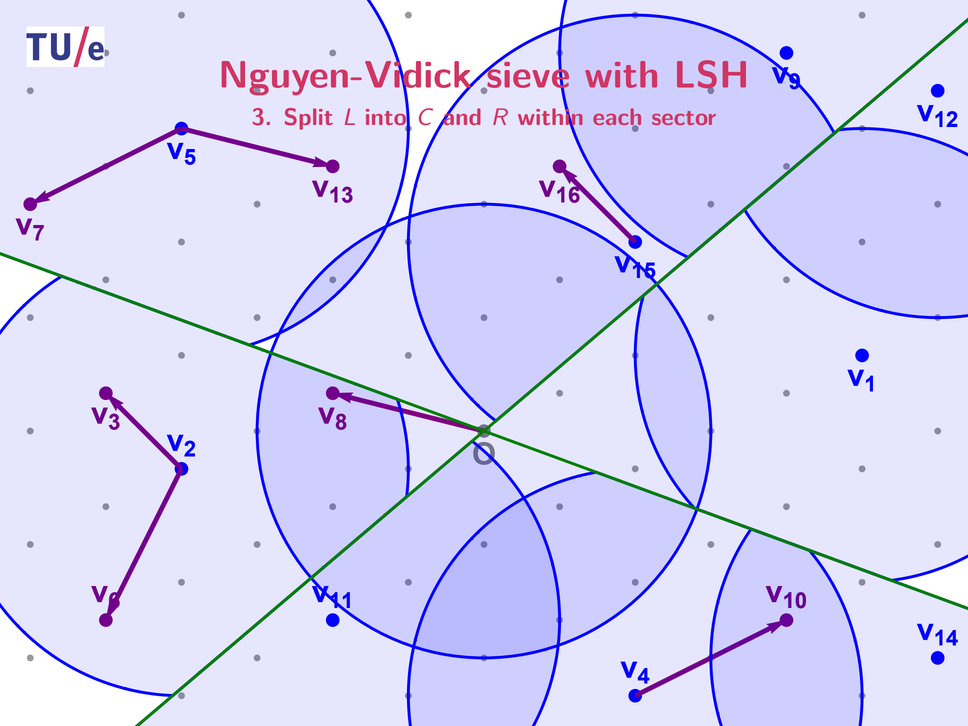
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



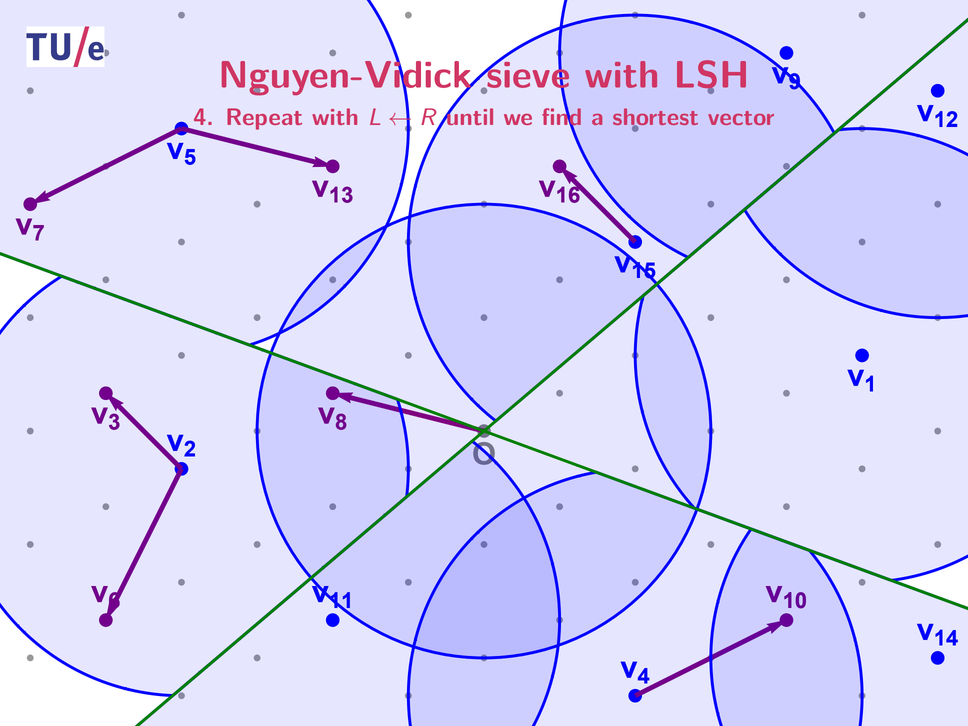
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



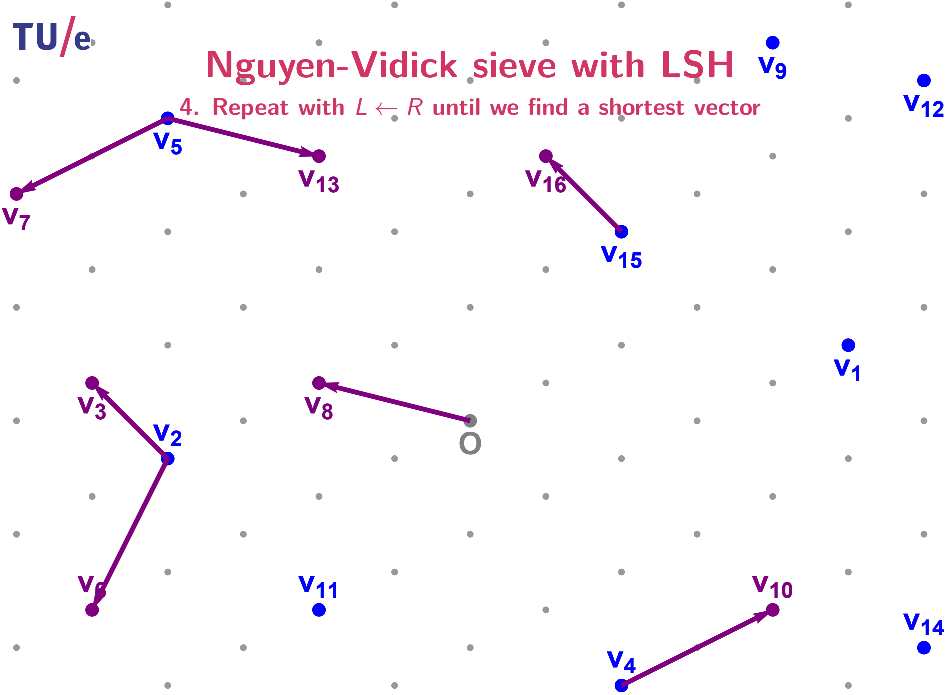
Nguyen-Vidick sieve with LSH

4. Repeat with $L \leftarrow R$ until we find a shortest vector



Nguyen-Vidick sieve with LSH

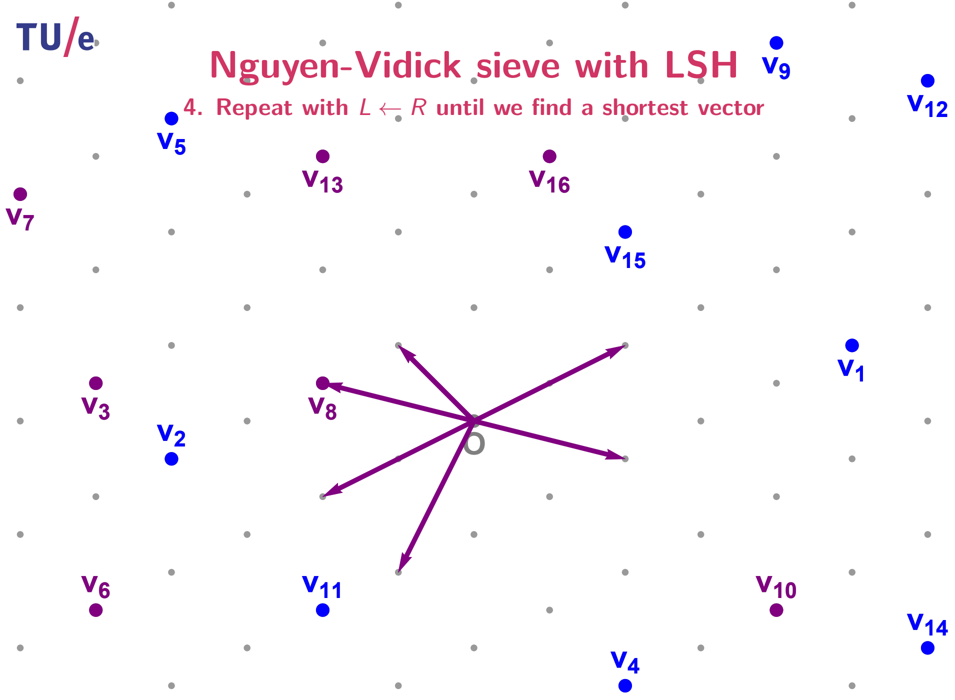
4. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

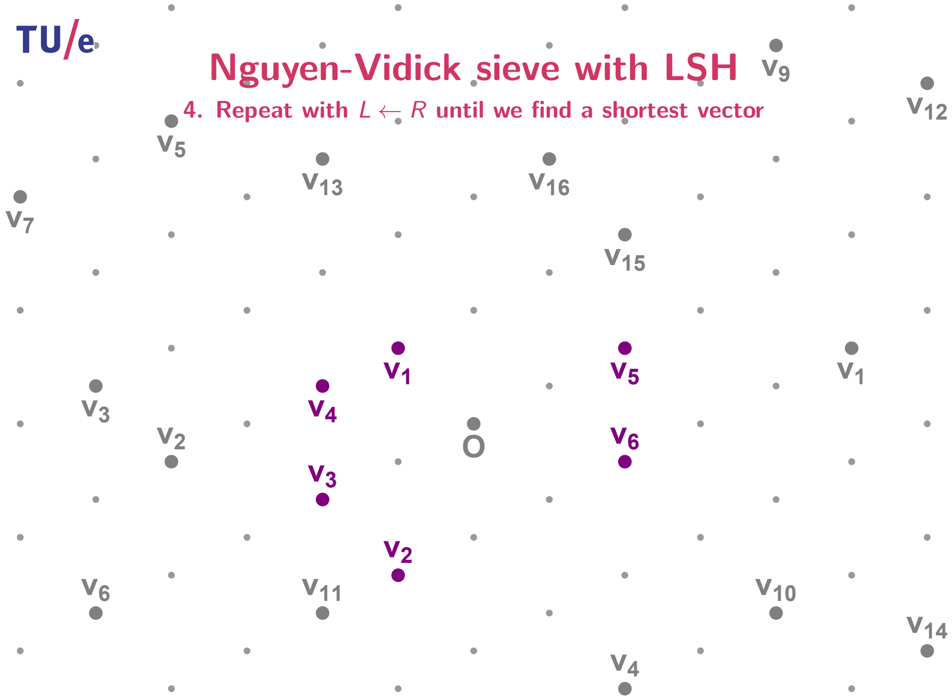
Nguyen-Vidick sieve with LSH

4. Repeat with $L \leftarrow R$ until we find a shortest vector



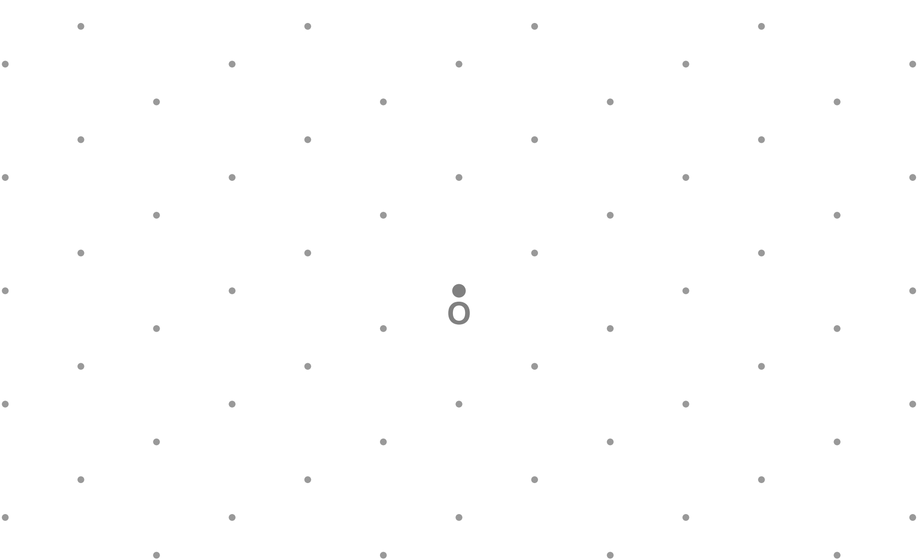
Nguyen-Vidick sieve with LSH

4. Repeat with $L \leftarrow R$ until we find a shortest vector



GaussSieve

1. Generate random lattice vectors

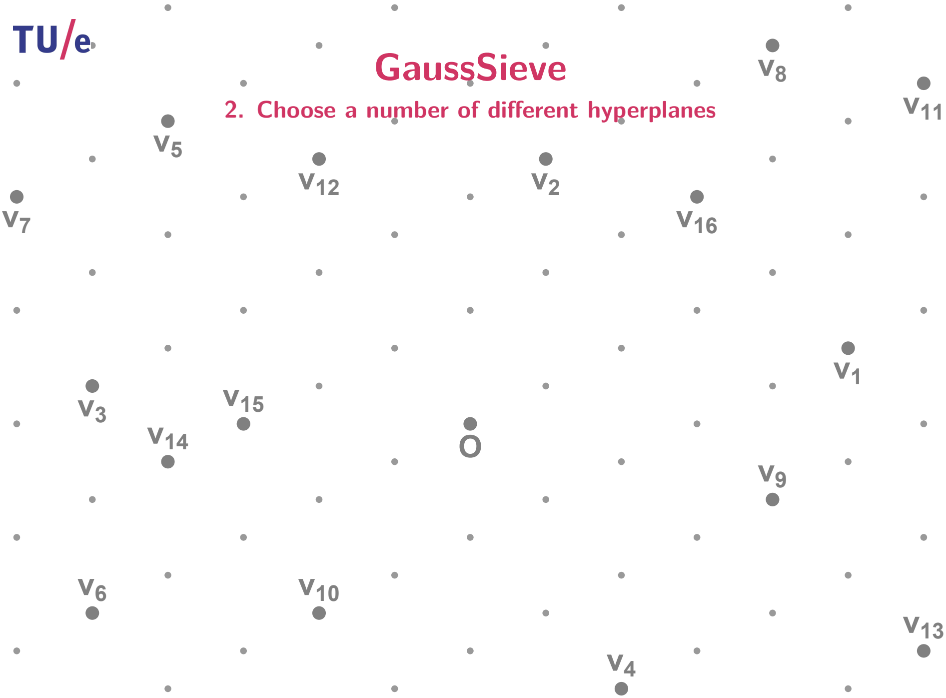


GaussSieve

1. Generate random lattice vectors

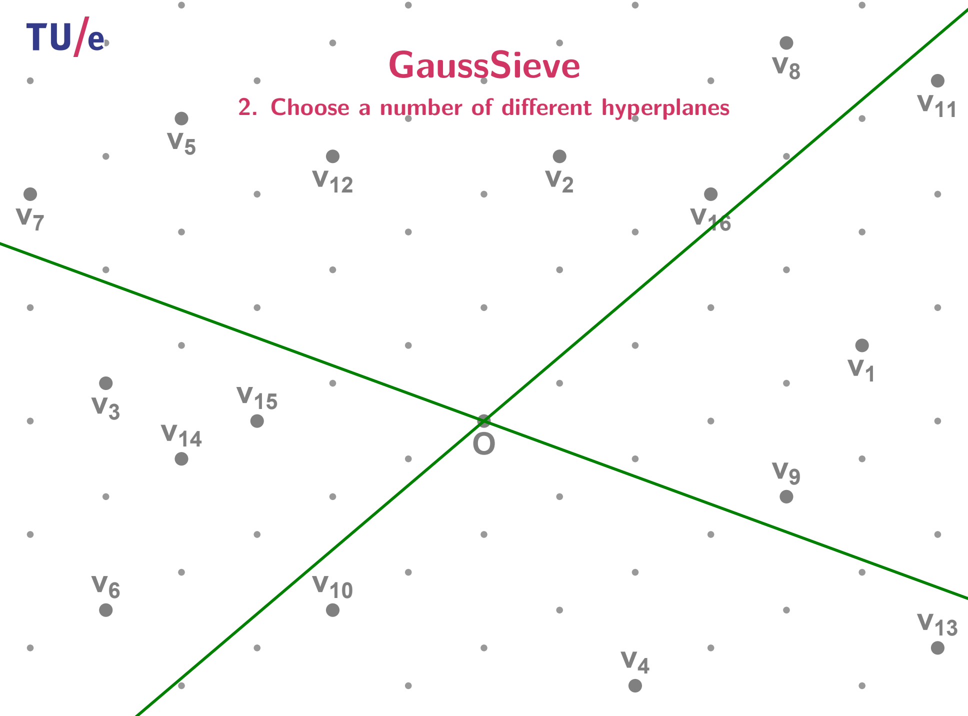
GaussSieve

2. Choose a number of different hyperplanes



GaussSieve

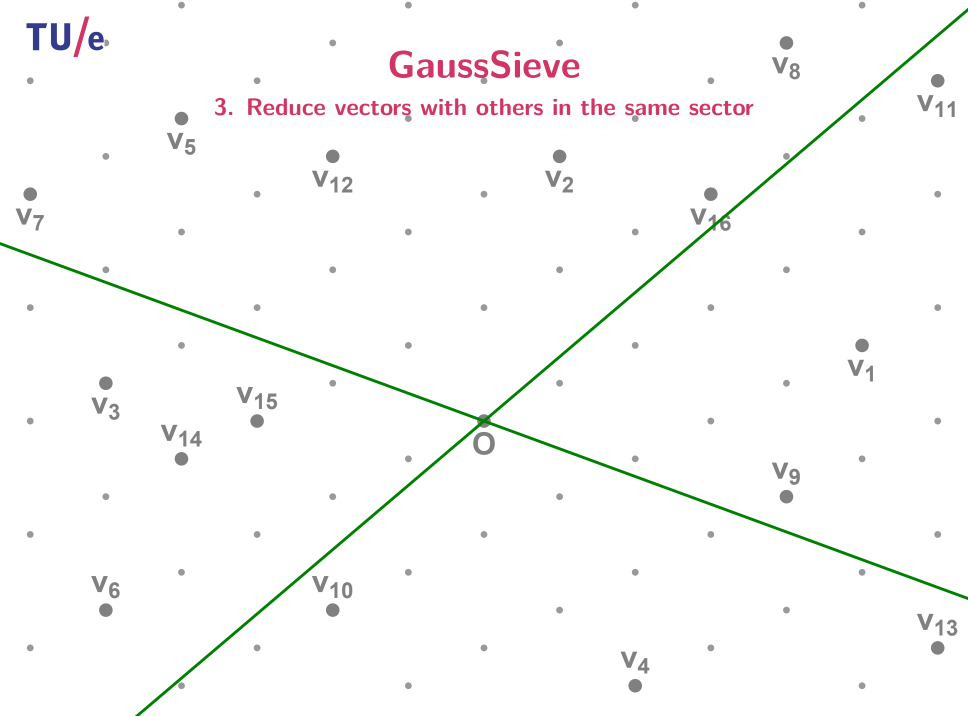
2. Choose a number of different hyperplanes



TU/e

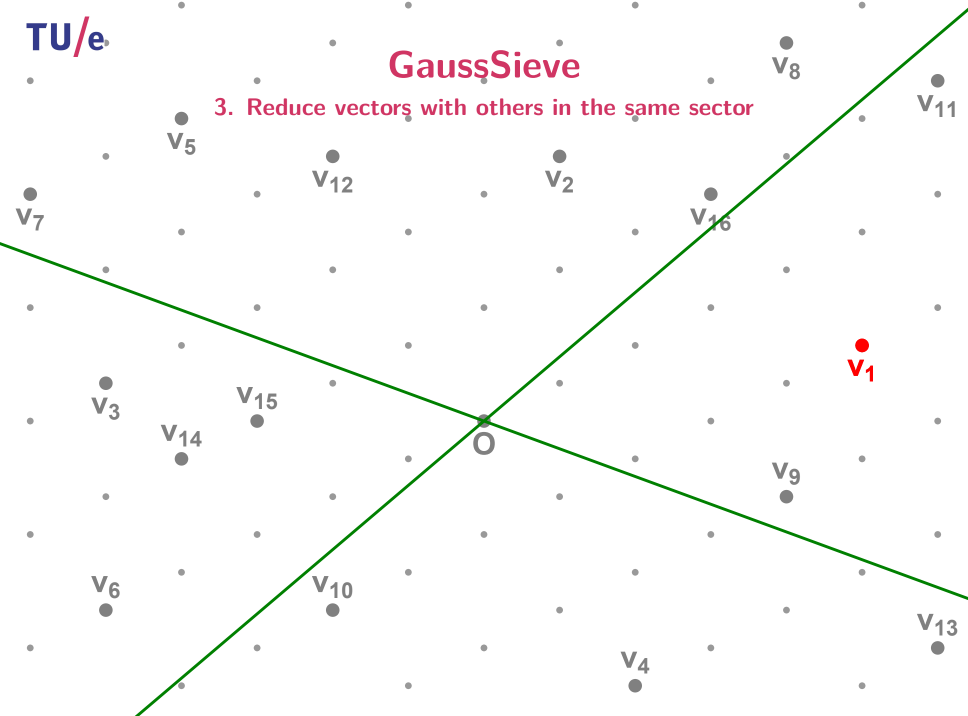
GaussSieve

3. Reduce vectors with others in the same sector



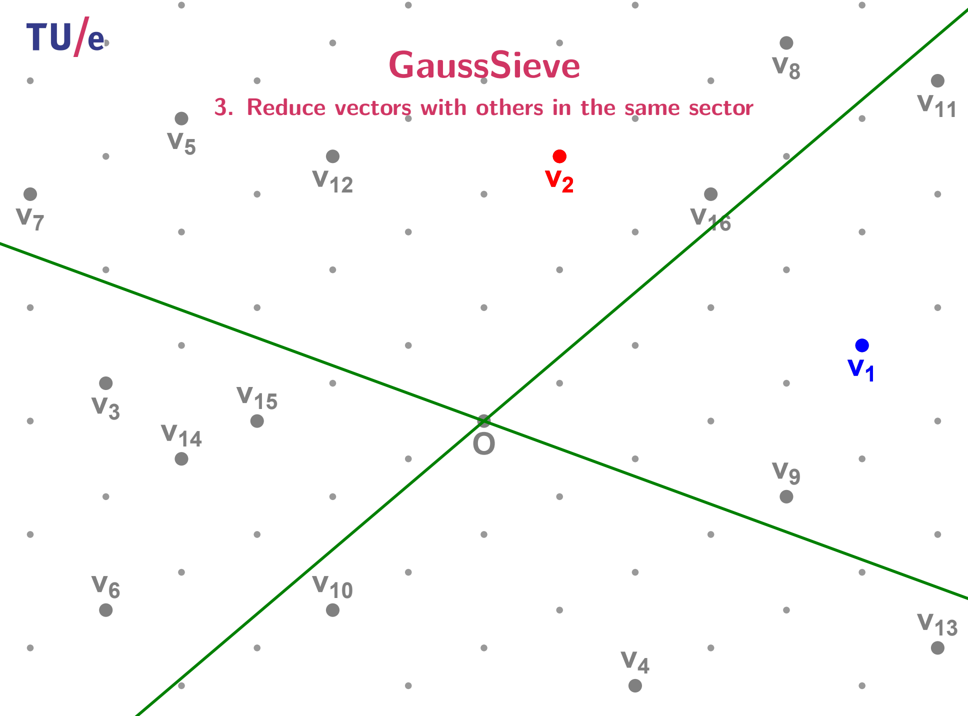
GaussSieve

3. Reduce vectors with others in the same sector



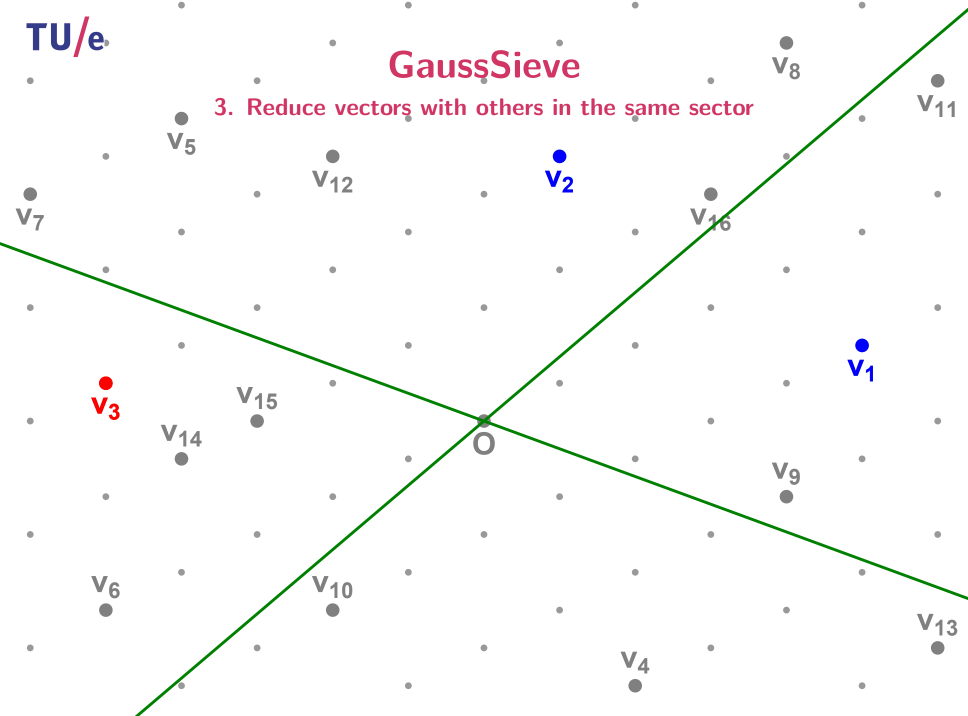
GaussSieve

3. Reduce vectors with others in the same sector



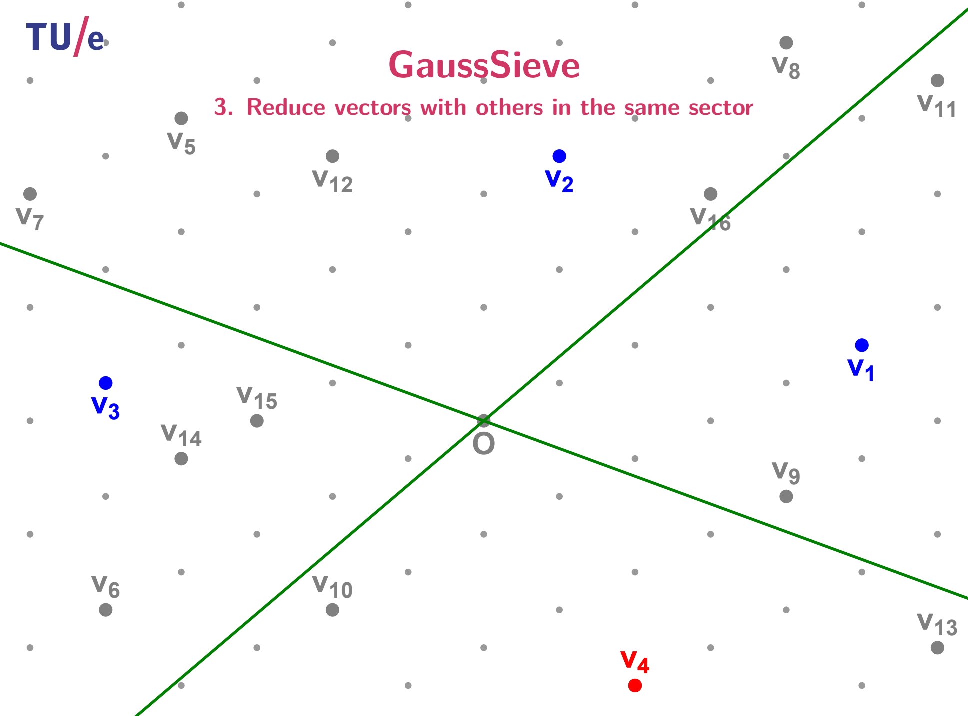
GaussSieve

3. Reduce vectors with others in the same sector



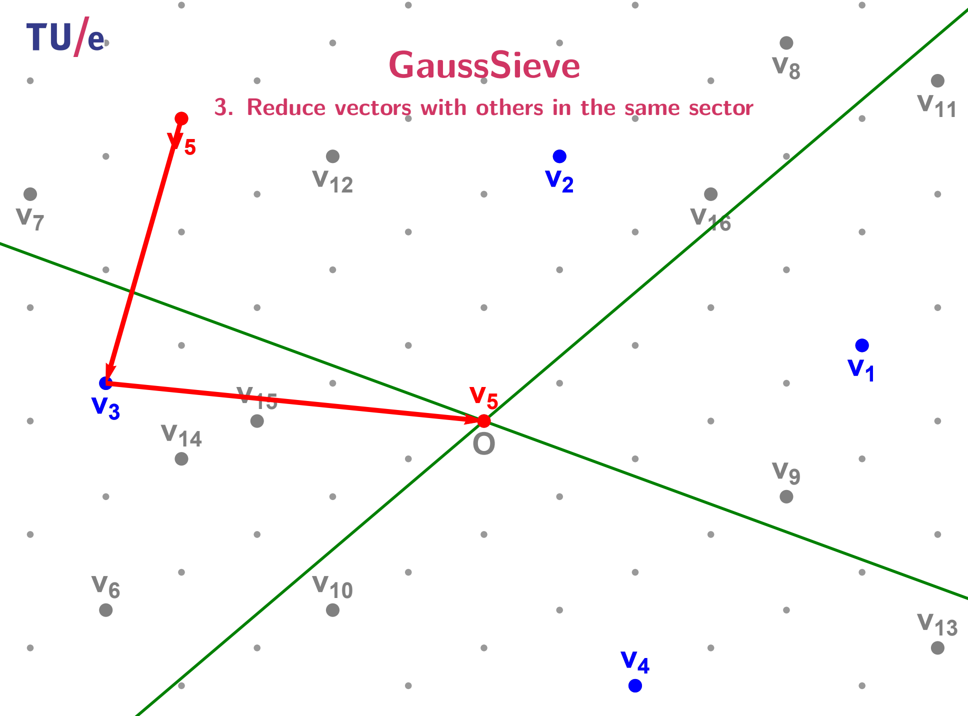
GaussSieve

3. Reduce vectors with others in the same sector



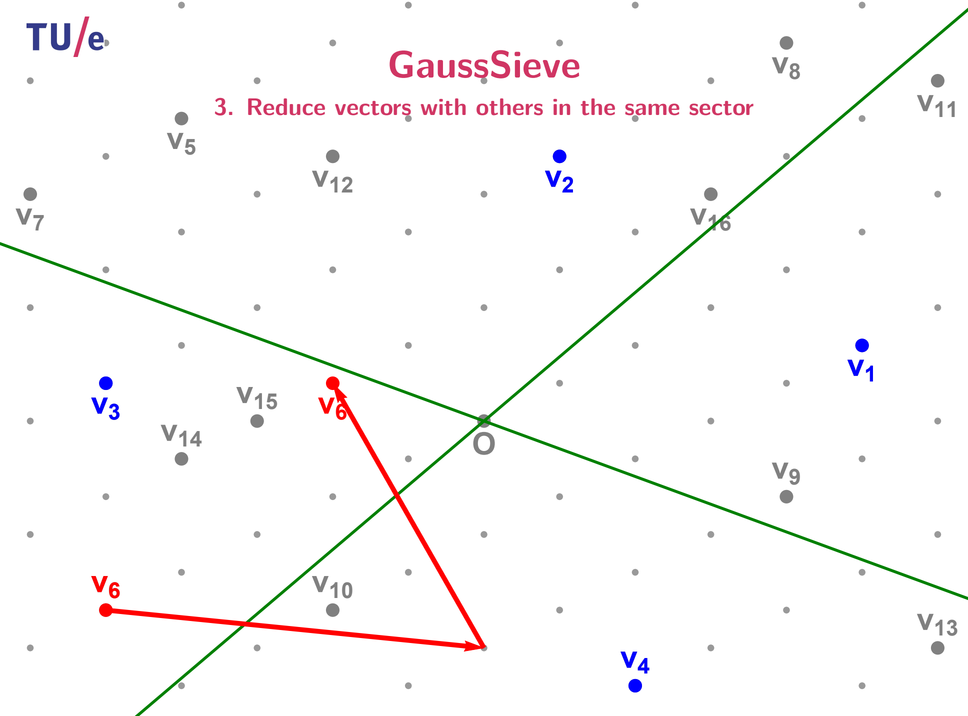
GaussSieve

3. Reduce vectors with others in the same sector



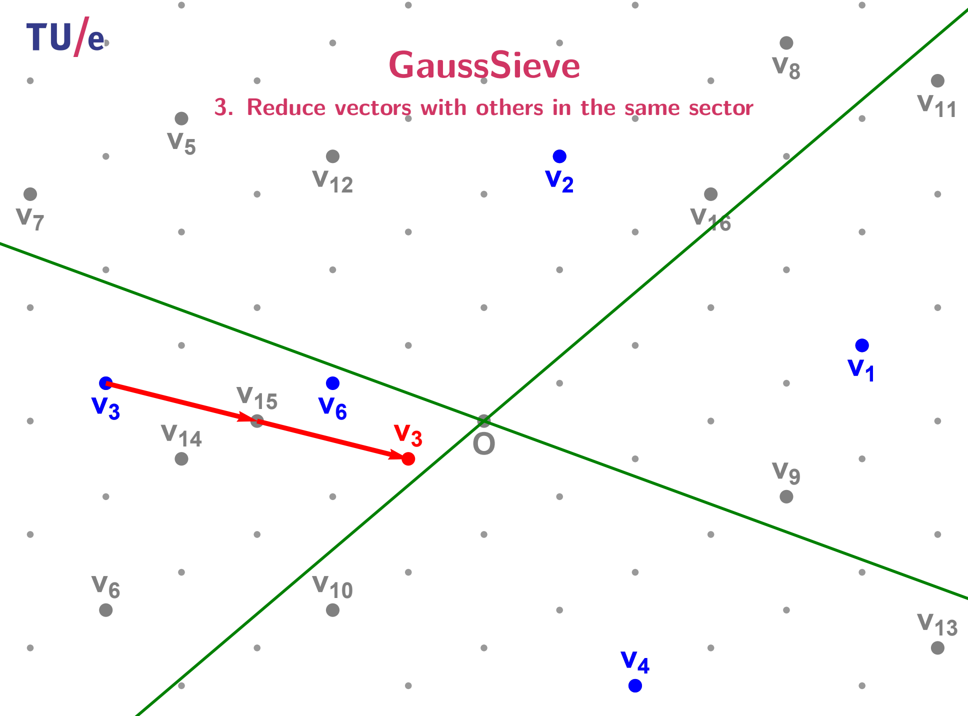
GaussSieve

3. Reduce vectors with others in the same sector



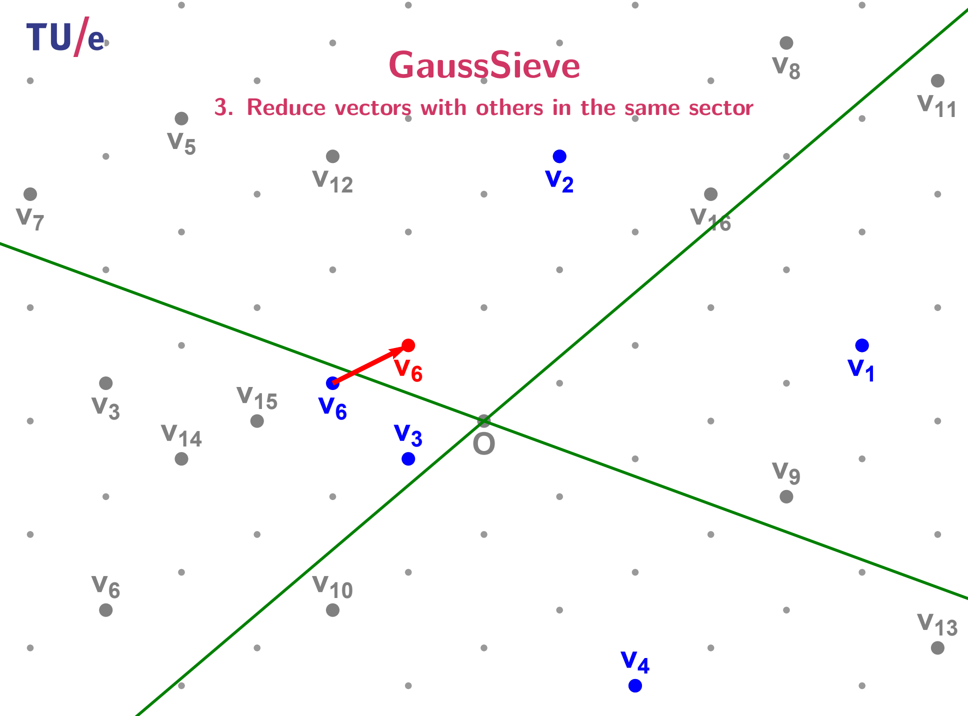
GaussSieve

3. Reduce vectors with others in the same sector



GaussSieve

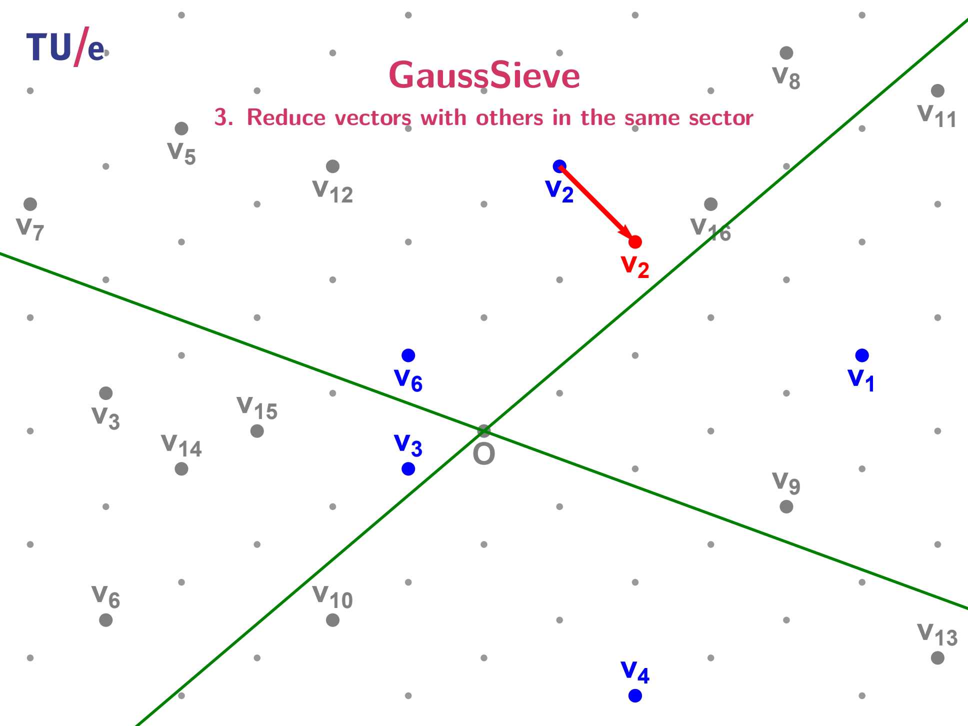
3. Reduce vectors with others in the same sector



TU/e

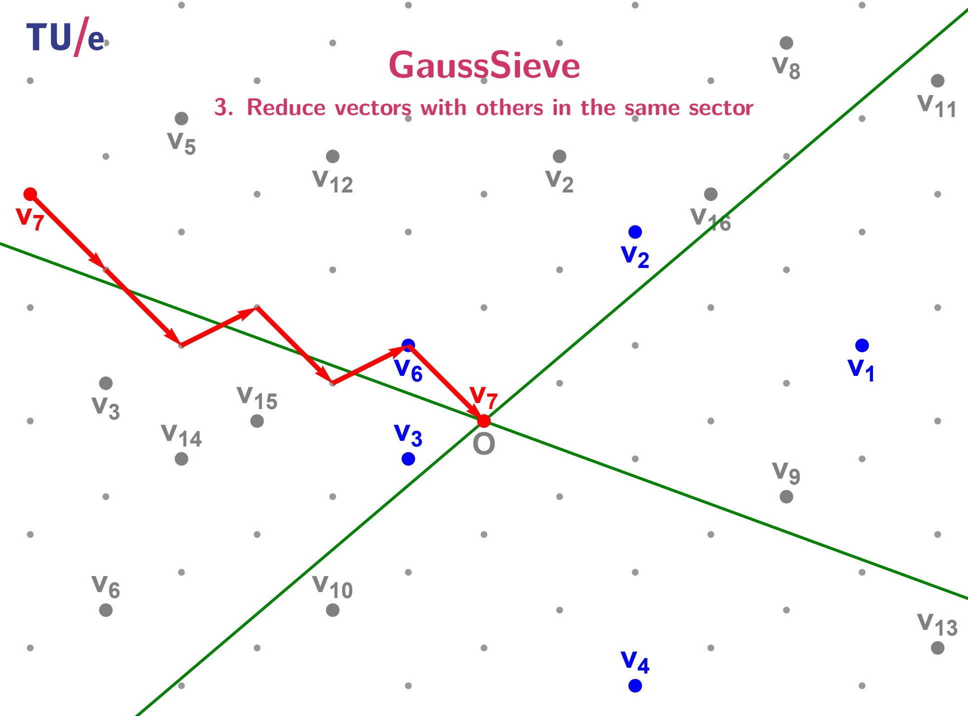
GaussSieve

3. Reduce vectors with others in the same sector



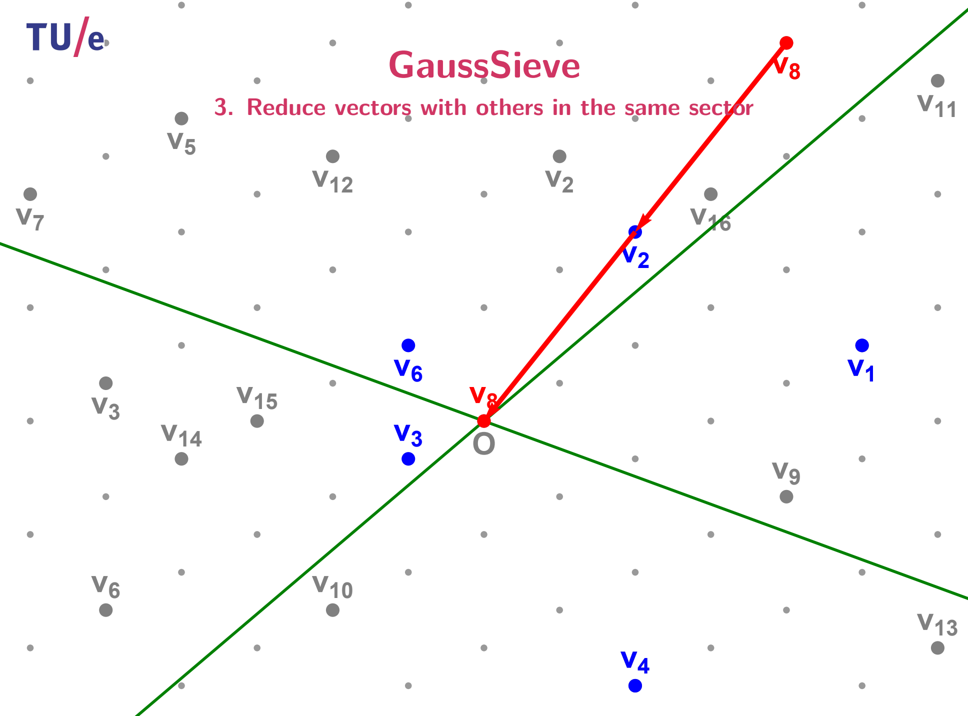
GaussSieve

3. Reduce vectors with others in the same sector



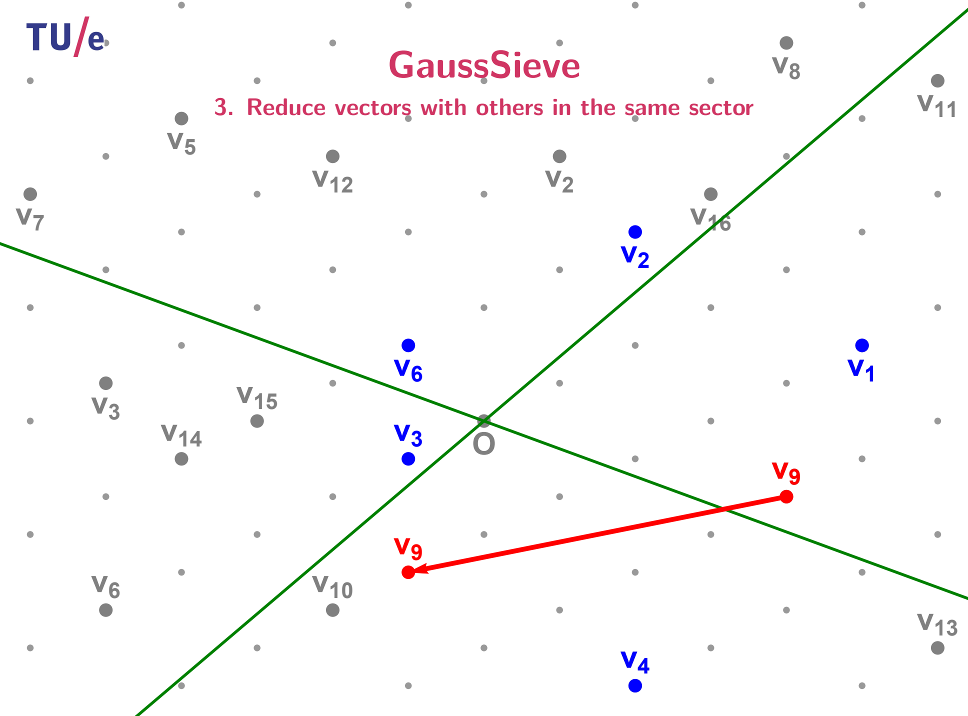
GaussSieve

3. Reduce vectors with others in the same sector



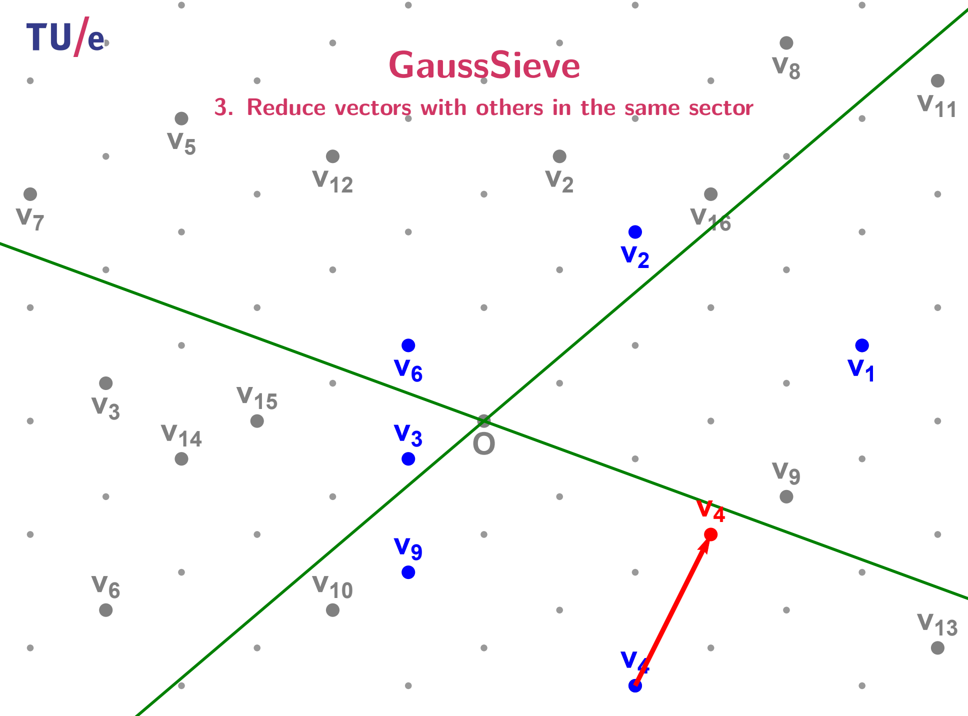
GaussSieve

3. Reduce vectors with others in the same sector



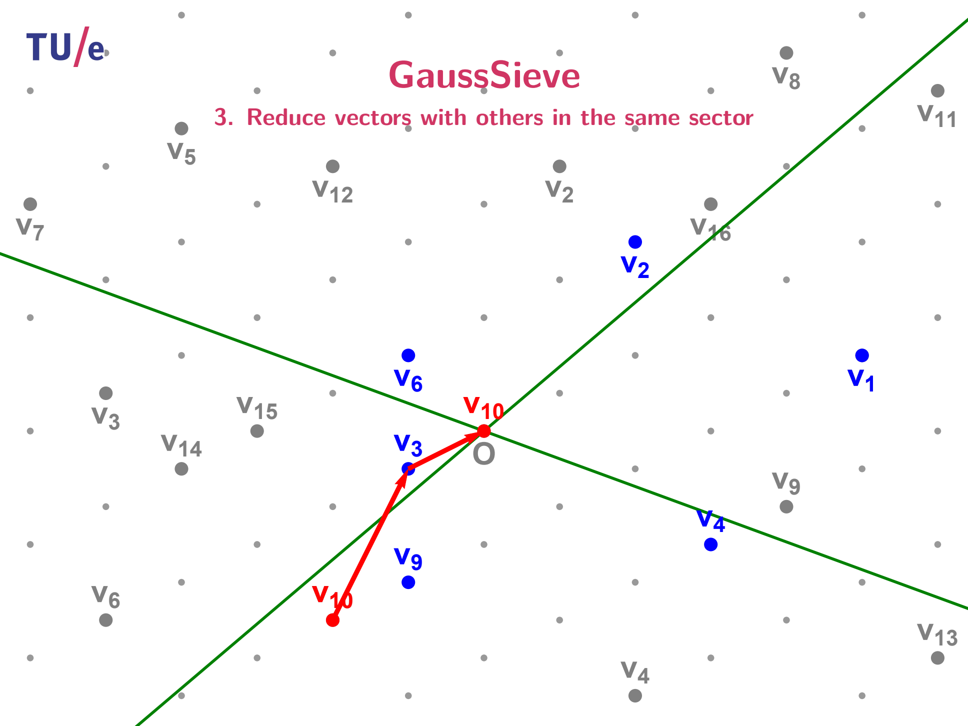
GaussSieve

3. Reduce vectors with others in the same sector



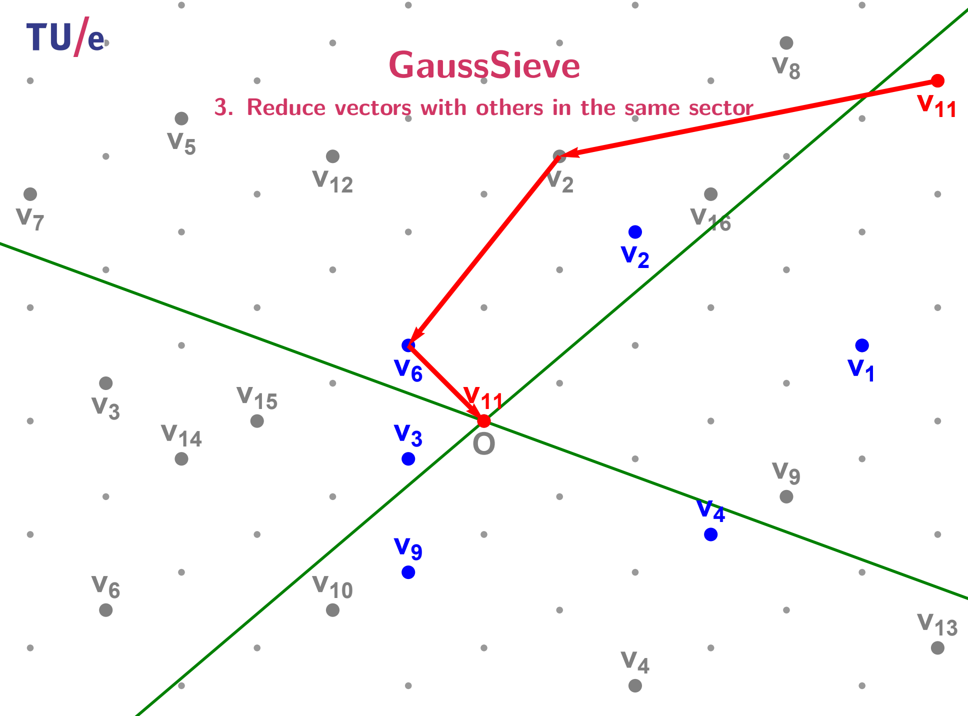
GaussSieve

3. Reduce vectors with others in the same sector



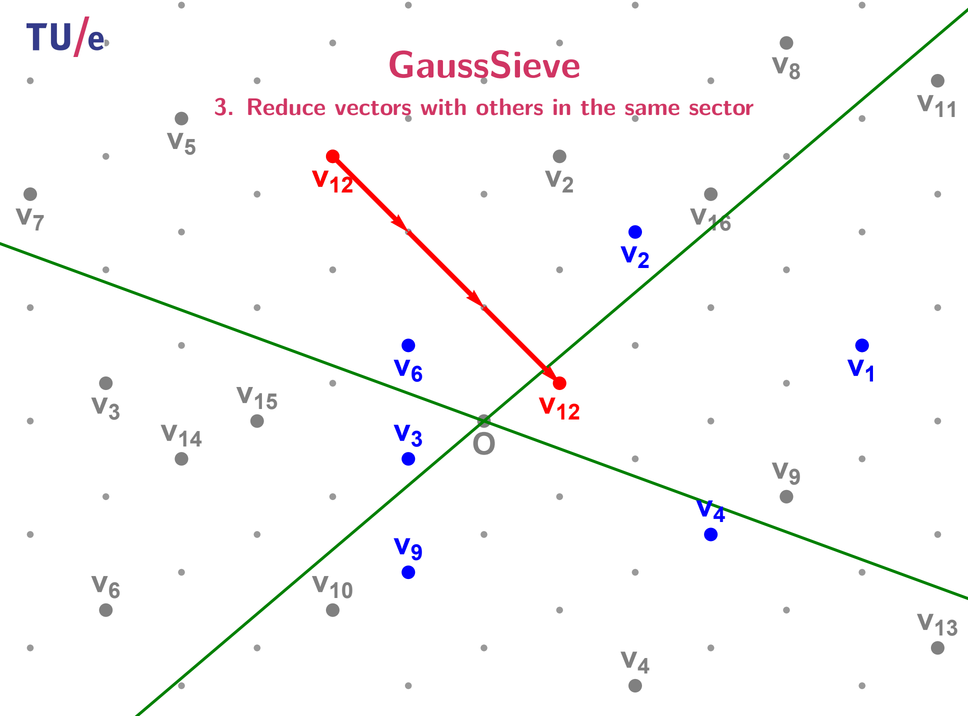
GaussSieve

3. Reduce vectors with others in the same sector



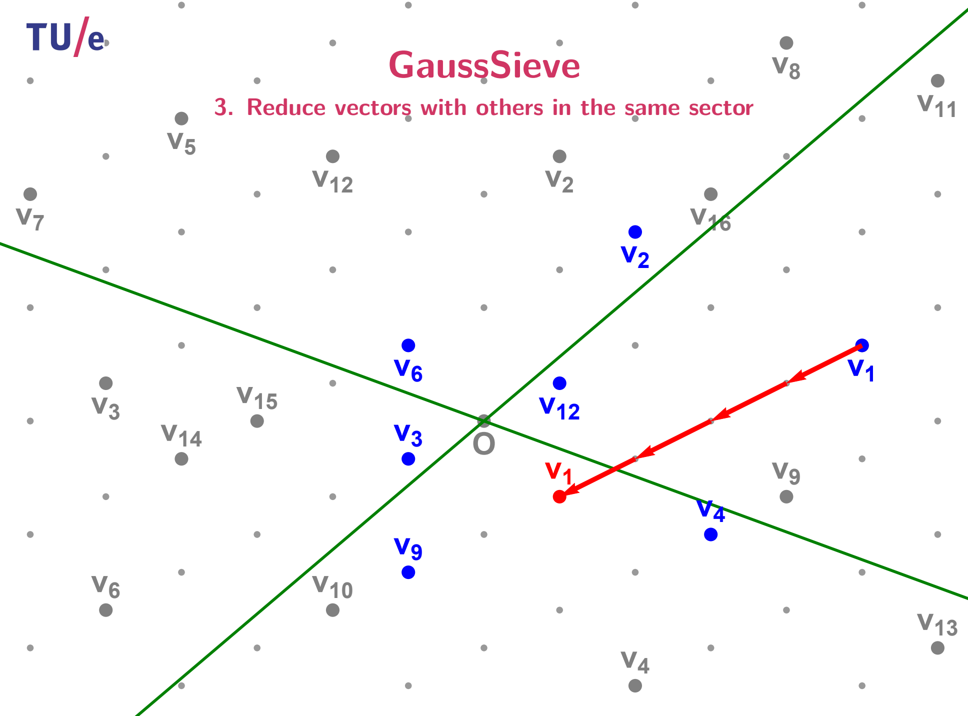
GaussSieve

3. Reduce vectors with others in the same sector



GaussSieve

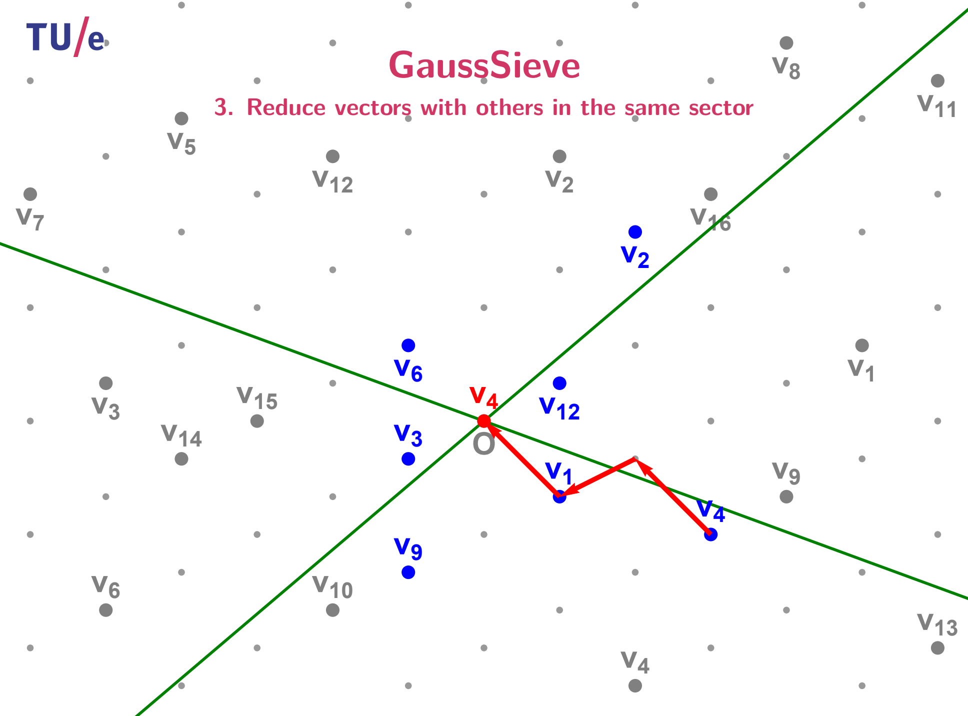
3. Reduce vectors with others in the same sector



TU/e

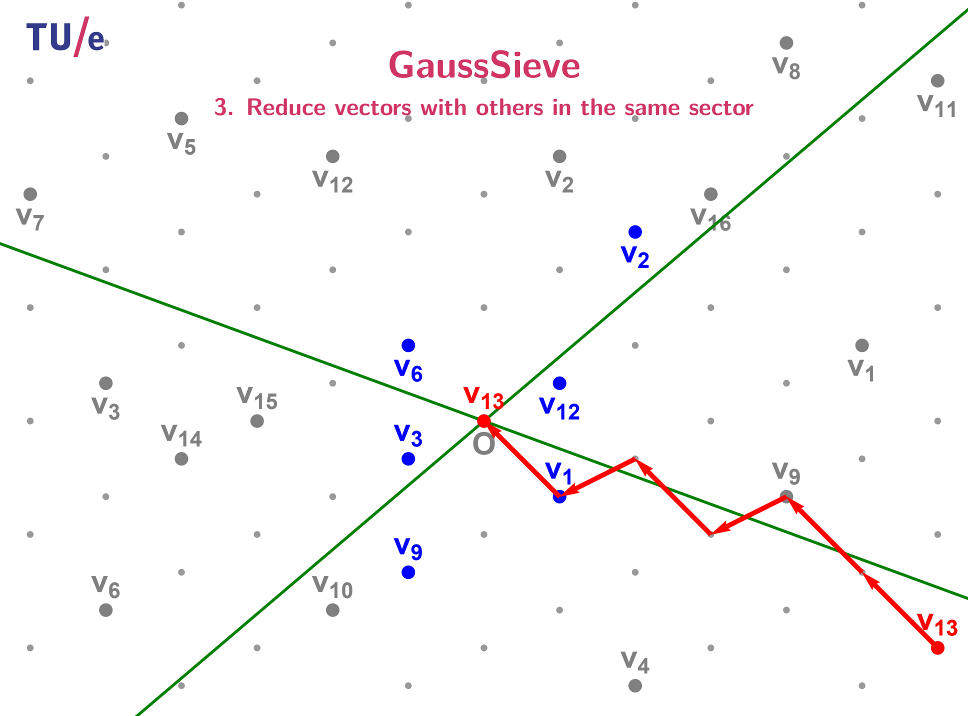
GaussSieve

3. Reduce vectors with others in the same sector



GaussSieve

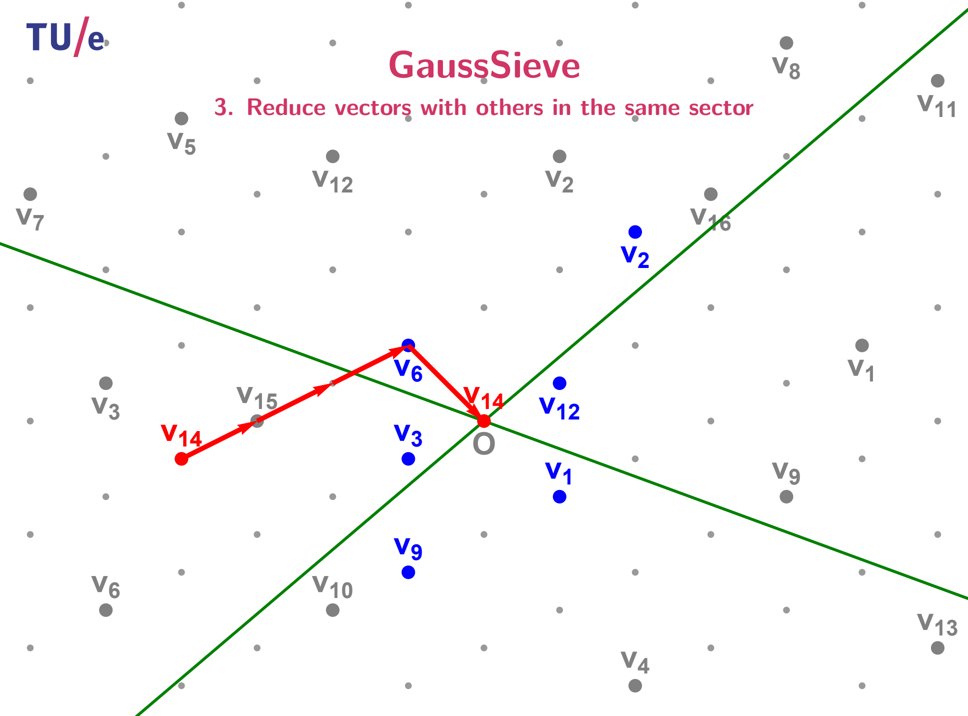
3. Reduce vectors with others in the same sector



TU/e

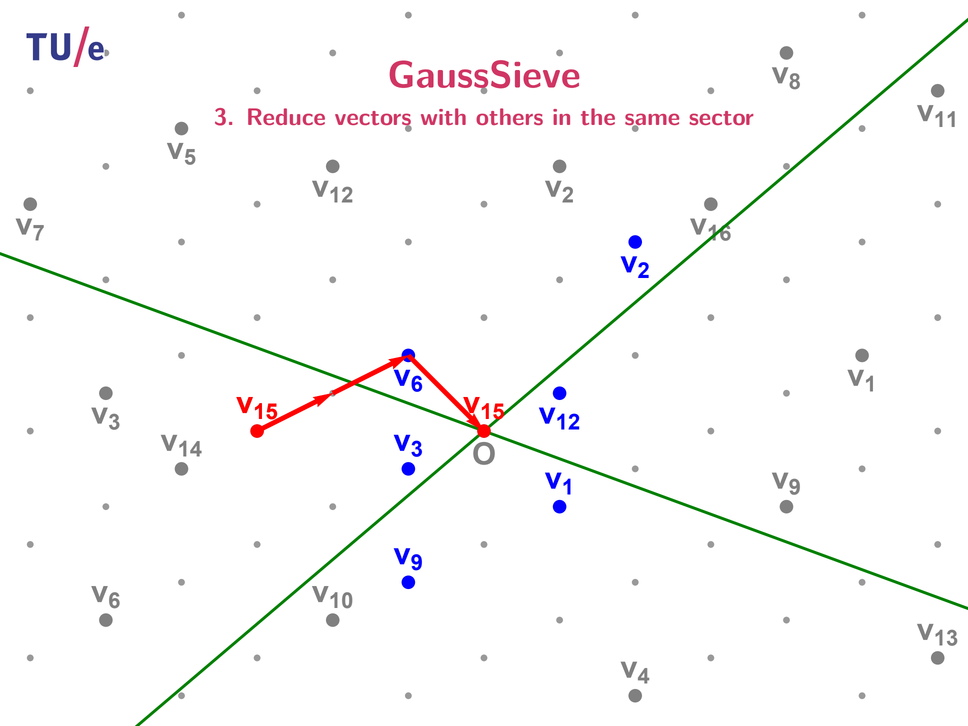
GaussSieve

3. Reduce vectors with others in the same sector



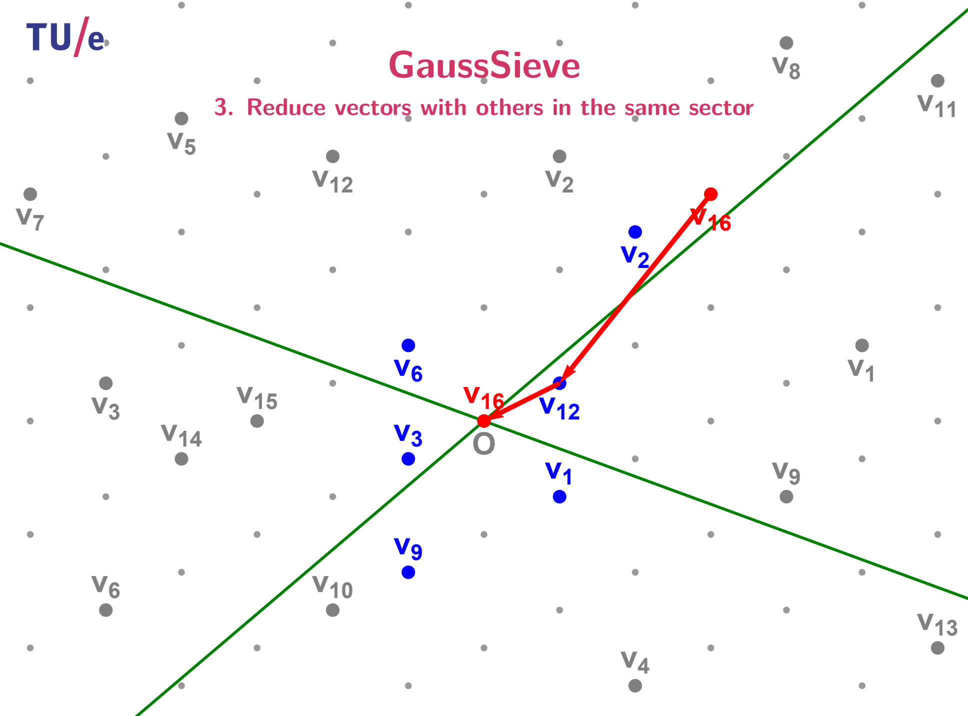
GaussSieve

3. Reduce vectors with others in the same sector



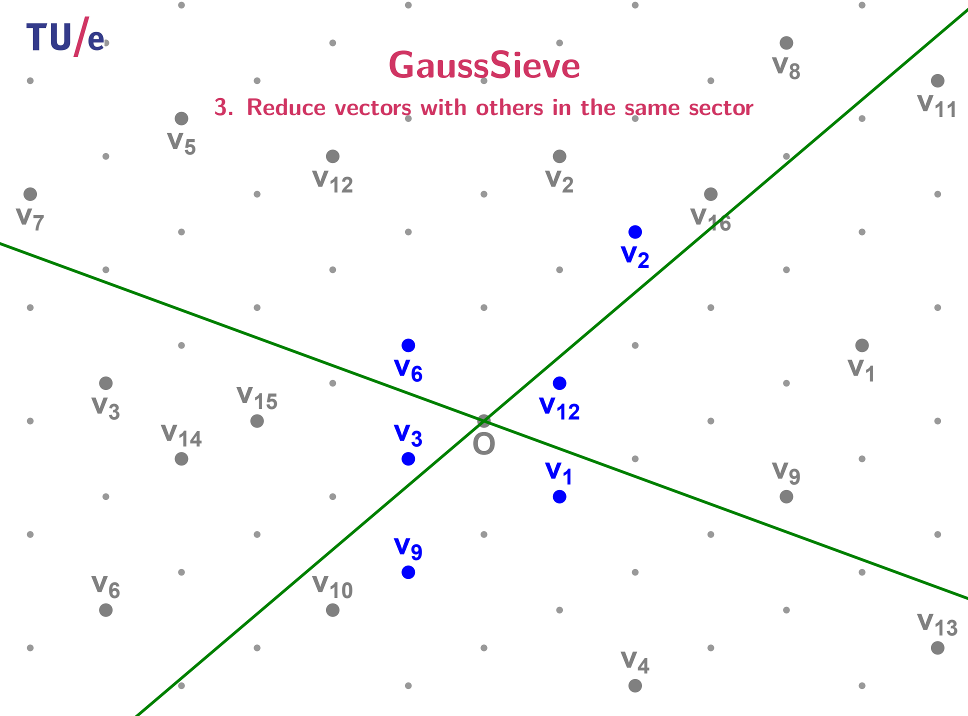
GaussSieve

3. Reduce vectors with others in the same sector



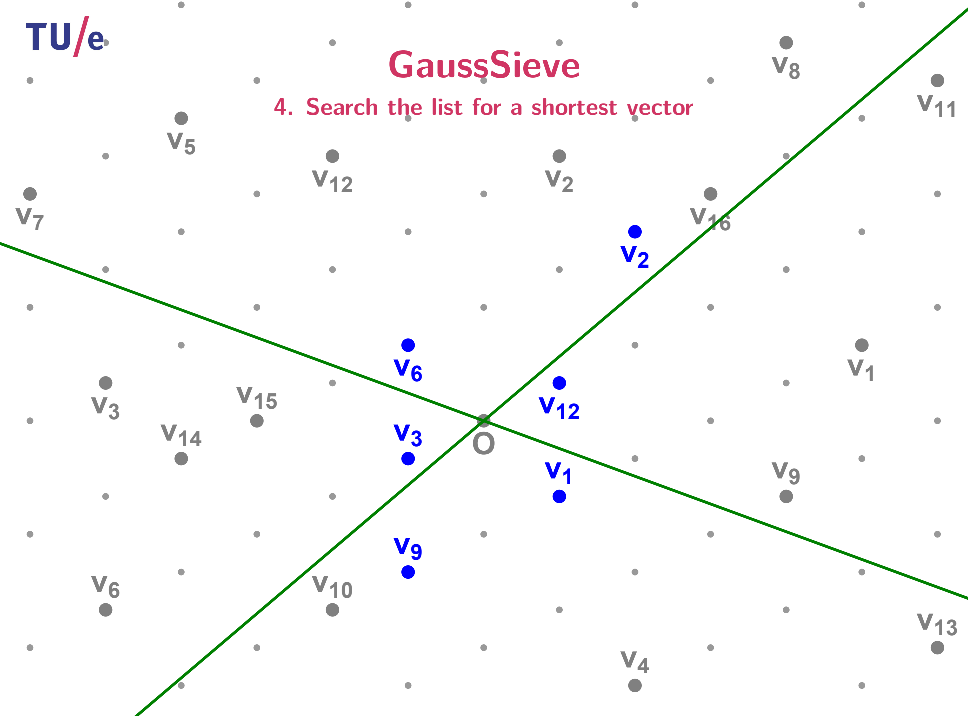
GaussSieve

3. Reduce vectors with others in the same sector



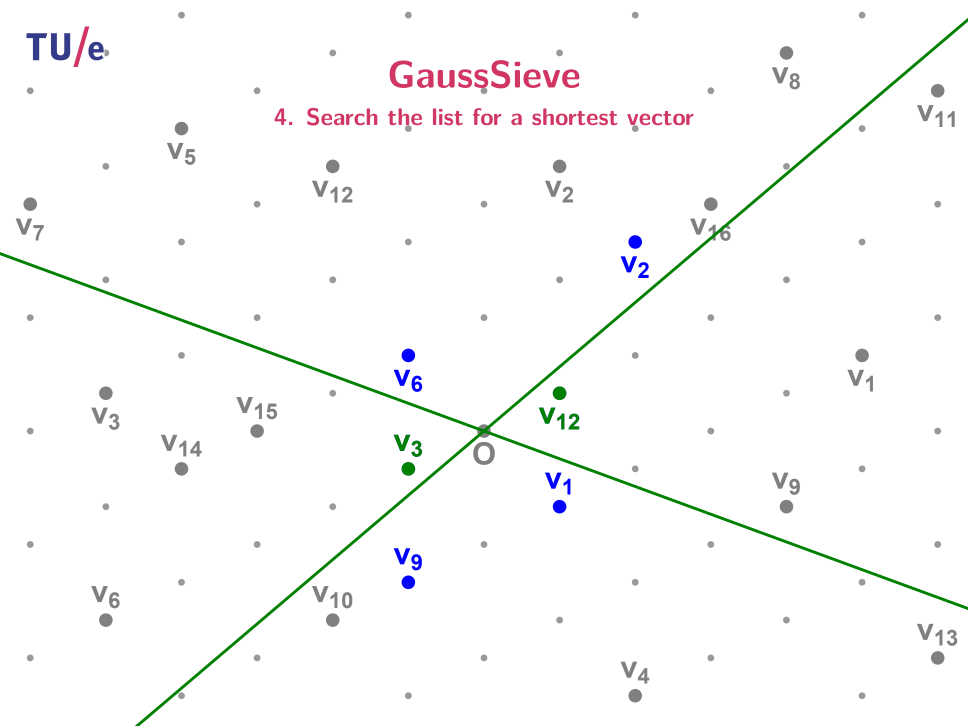
GaussSieve

4. Search the list for a shortest vector



GaussSieve

4. Search the list for a shortest vector



Sieving with angular LSH

Overview

TU/e

Sieving with angular LSH

Overview

- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent "hash tables"

Sieving with angular LSH

Overview

- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent “hash tables”
- Space complexity: $2^{0.34n+o(n)}$
  - Store $2^{0.13n}$ hash tables, each containing all $2^{0.21n}$ vectors

Sieving with angular LSH

Overview

- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent "hash tables"
- Space complexity: $2^{0.34n+o(n)}$
  - Store $2^{0.13n}$ hash tables, each containing all $2^{0.21n}$ vectors
- Time complexity: $2^{0.34n+o(n)}$
  - Compute $2^{0.13n}$ hashes, and go through $2^{0.13n}$ vectors
  - Repeat this for each of $2^{0.21n}$ vectors

Sieving with angular LSH

Overview

- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent "hash tables"
- Space complexity: $2^{0.34n+o(n)}$
  - Store $2^{0.13n}$ hash tables, each containing all $2^{0.21n}$ vectors
- Time complexity: $2^{0.34n+o(n)}$
  - Compute $2^{0.13n}$ hashes, and go through $2^{0.13n}$ vectors
  - Repeat this for each of $2^{0.21n}$ vectors

Heuristic

Sieving with LSH runs in time $2^{0.34n+o(n)}$ and space $2^{0.34n+o(n)}$.

Sieving with angular LSH

Overview

- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent "hash tables"
- Space complexity: $2^{0.34n+o(n)}$
  - Store $2^{0.13n}$ hash tables, each containing all $2^{0.21n}$ vectors
- Time complexity: $2^{0.34n+o(n)}$
  - Compute $2^{0.13n}$ hashes, and go through $2^{0.13n}$ vectors
  - Repeat this for each of $2^{0.21n}$ vectors

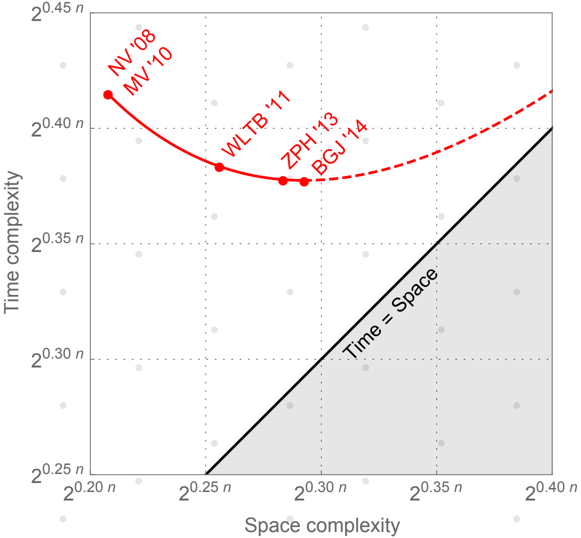
Heuristic

Sieving with LSH runs in time $2^{0.34n+o(n)}$ and space $2^{0.34n+o(n)}$.

Full details online at `http://eprint.iacr.org/2014/744`

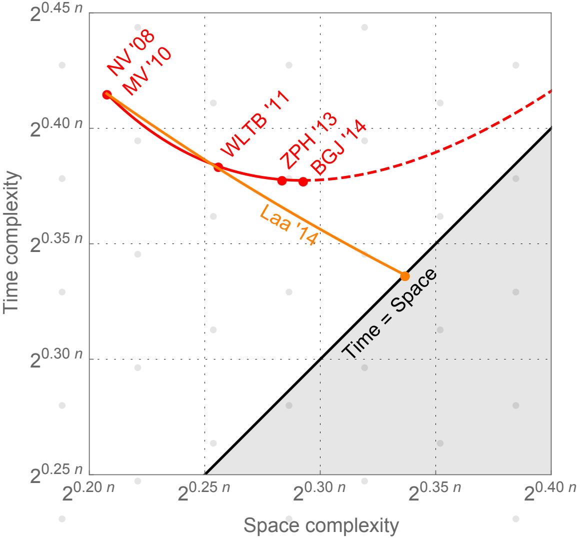
Results

Space/time trade-off



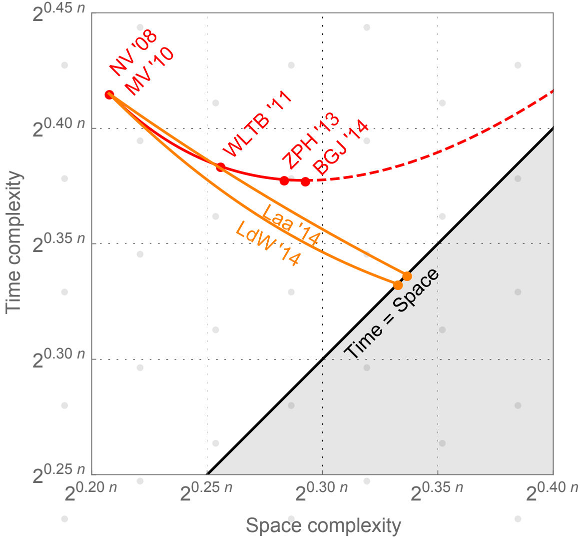
Results

Space/time trade-off



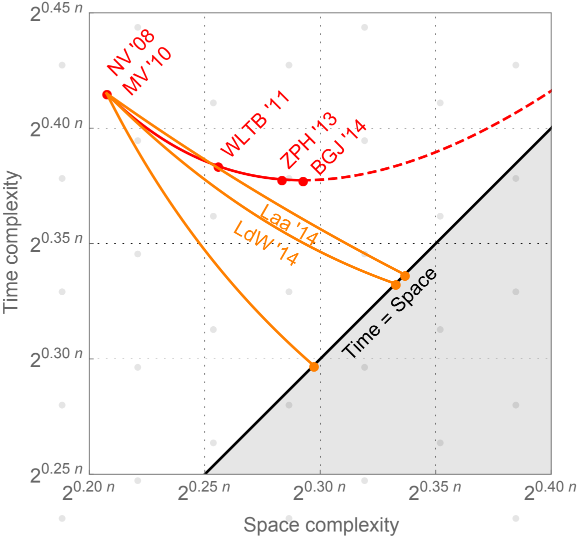
Results

Space/time trade-off



Results

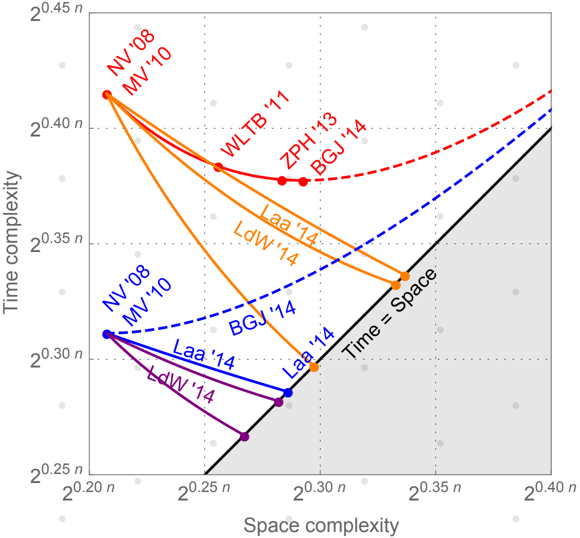
Space/time trade-off



TU/e

Results

Space/time trade-off



Questions