Technische Universiteit
Eindhoven
University of Technology

Part 2: Lattice algorithms for solving the shortest (non-zero) vector problem

Thijs Laarhoven

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

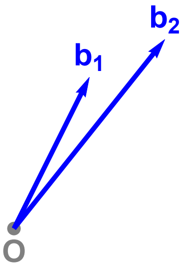
Lecture for Cryptography I
(January 8, 2015)

Lattices

What is a lattice?

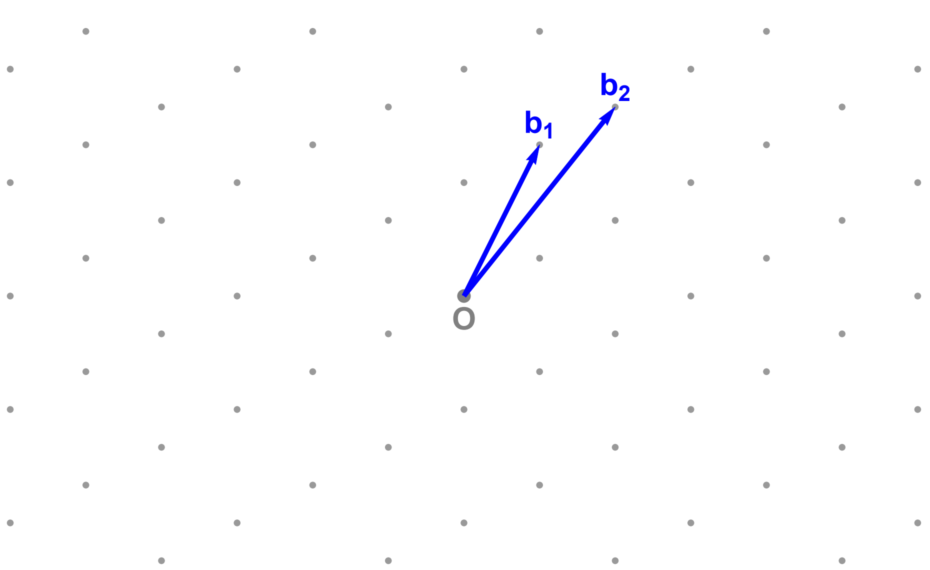
Lattices

What is a lattice?



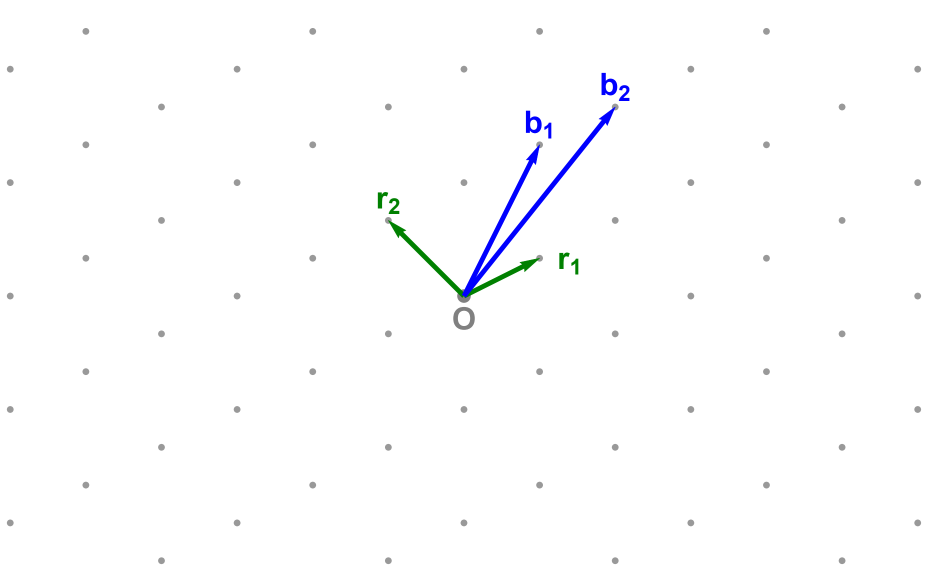
Lattices

What is a lattice?



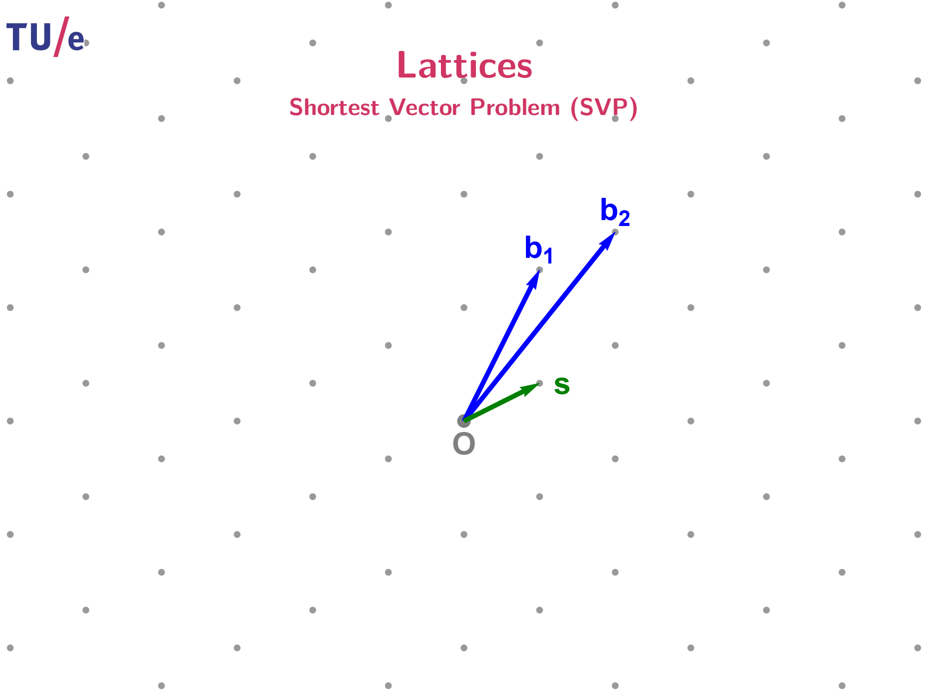
Lattices

Lattice basis reduction



Lattices

Shortest Vector Problem (SVP)



Outline

Enumeration algorithms

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

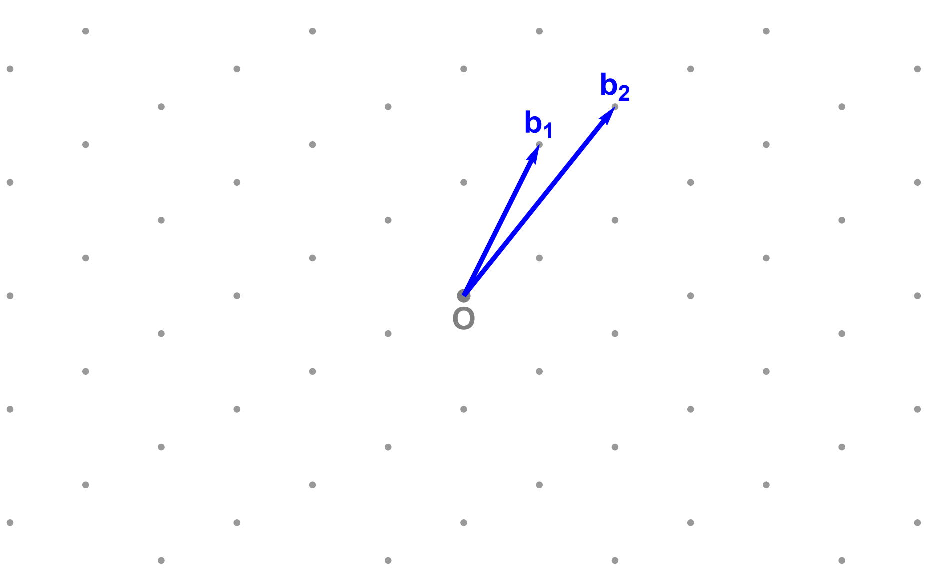
The Voronoi cell algorithm

Sieving algorithms

- Nguyen-Vidick sieve
- Multiple levels
- GaussSieve
- Locality-sensitive hashing

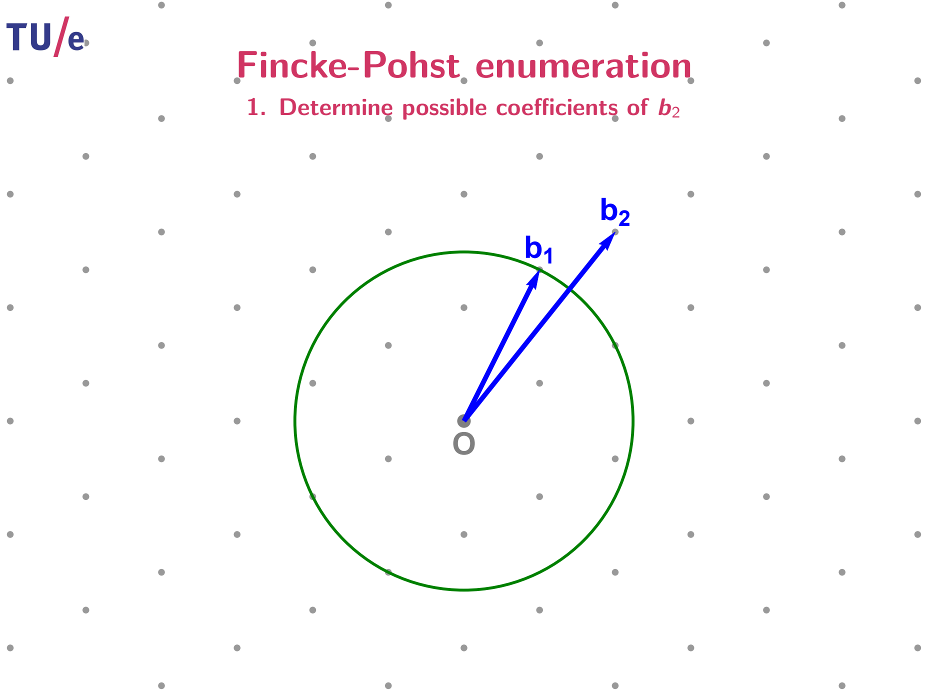
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



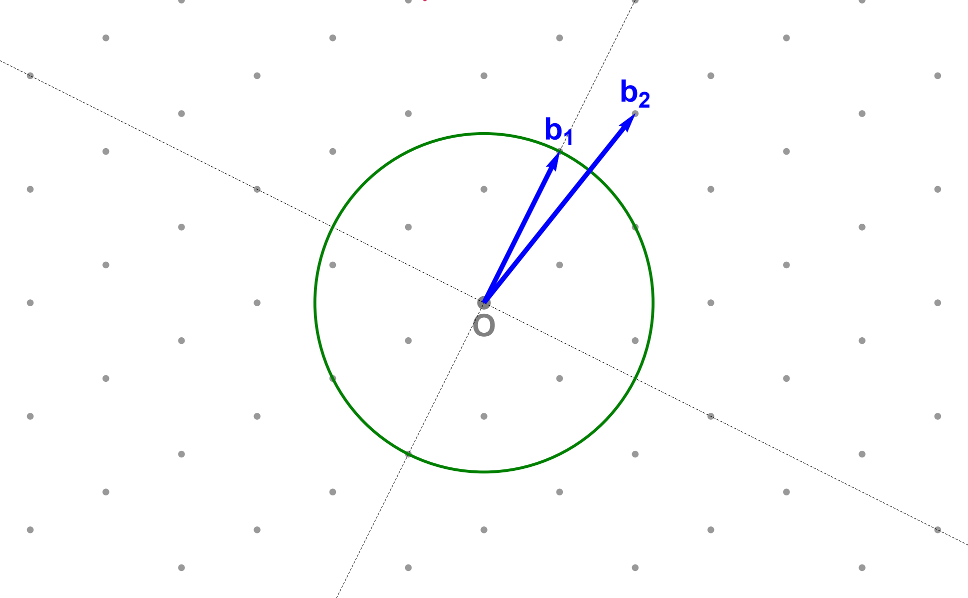
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



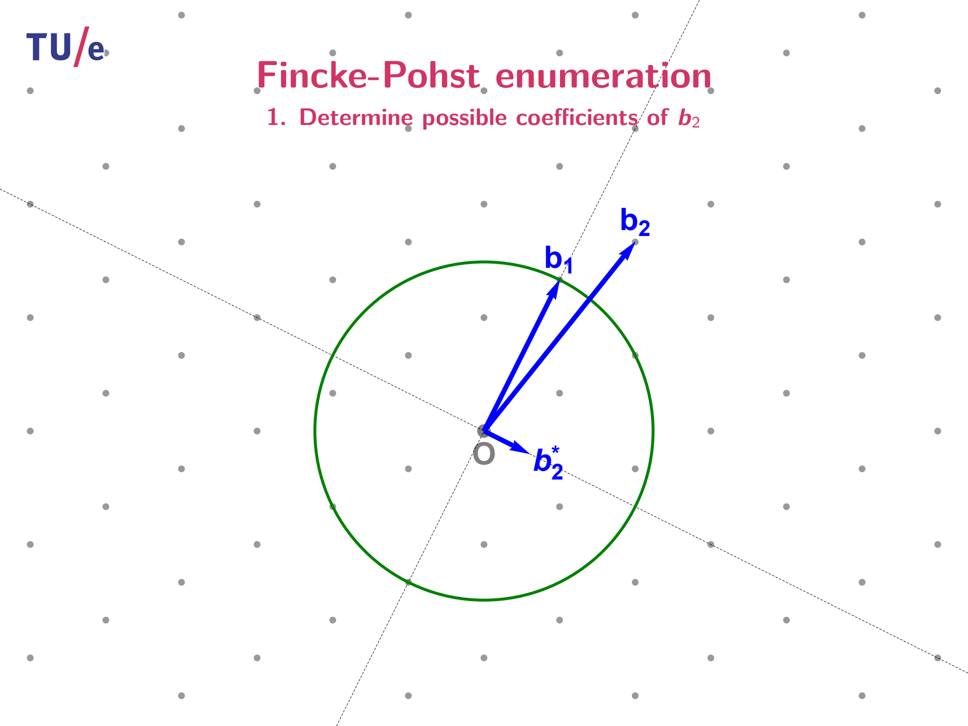
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



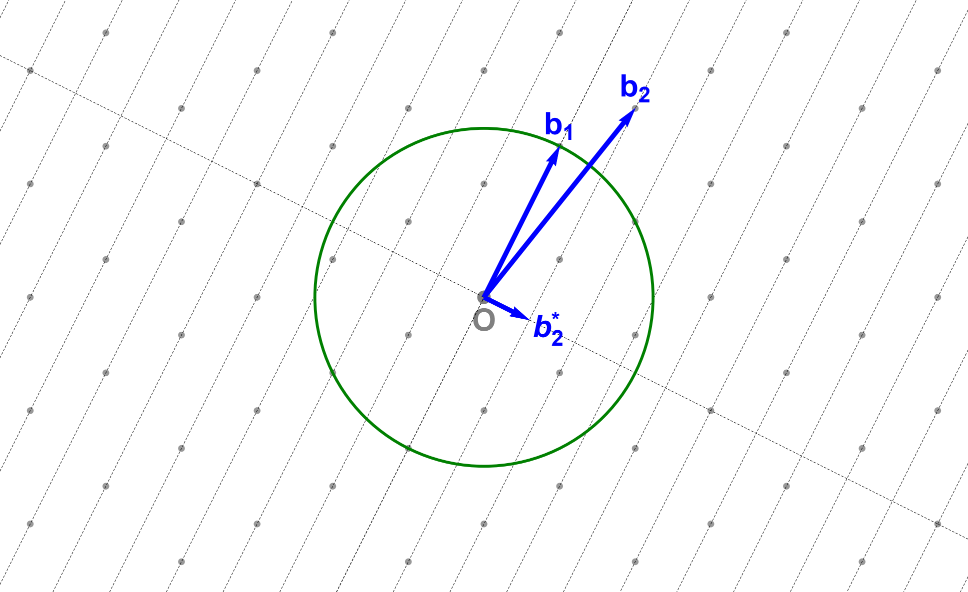
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



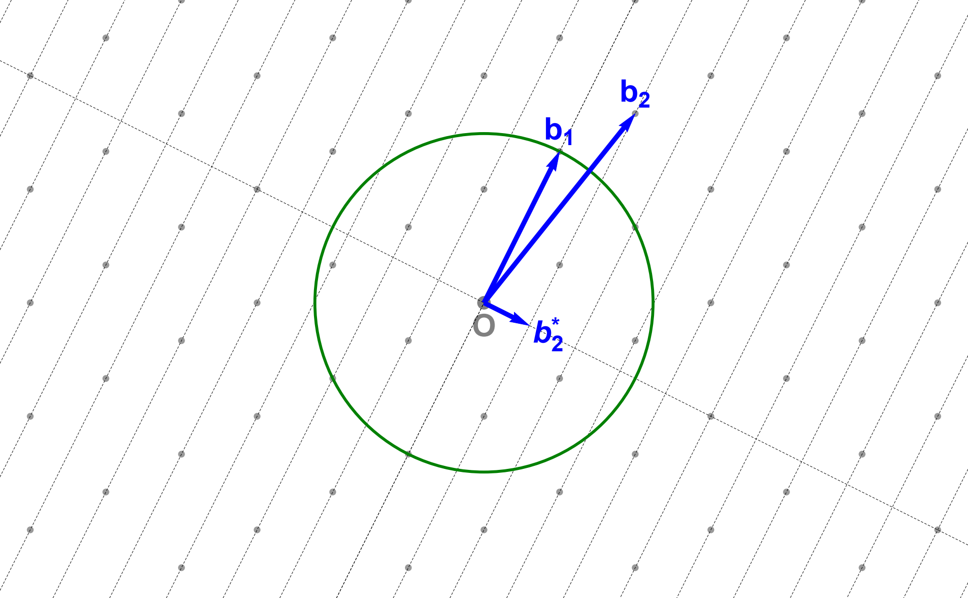
Fincke-Pohst enumeration

1. Determine possible coefficients of b_2



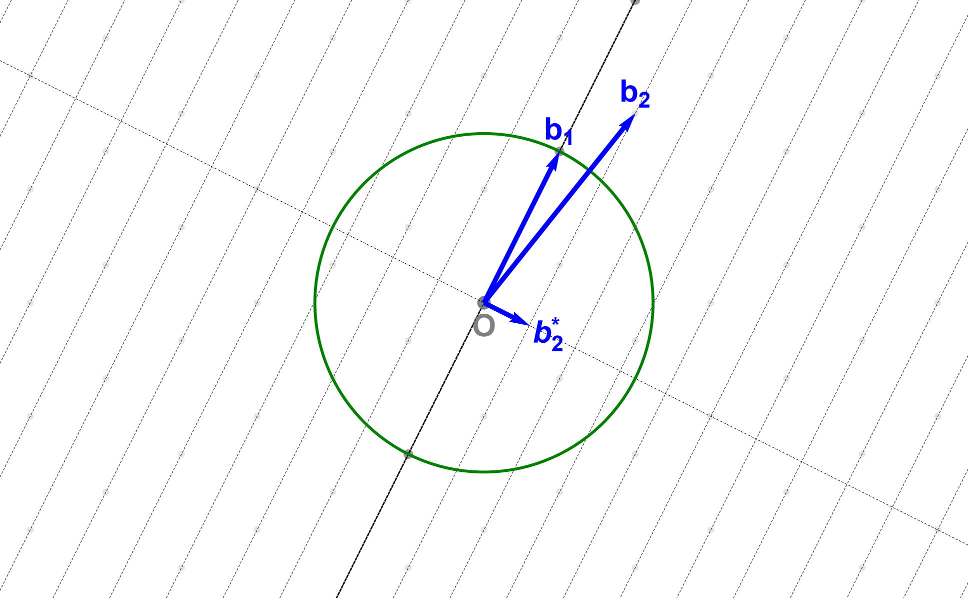
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



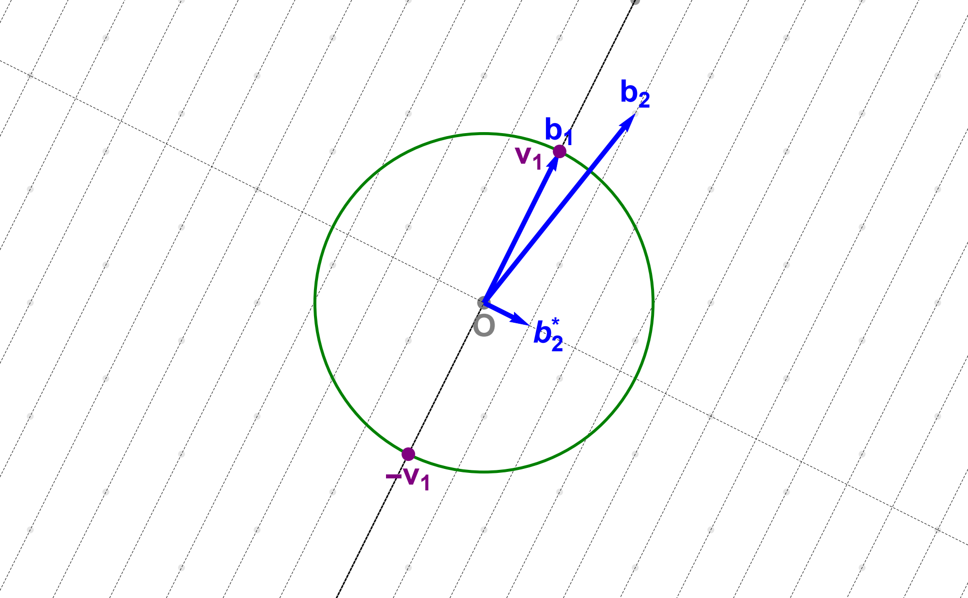
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



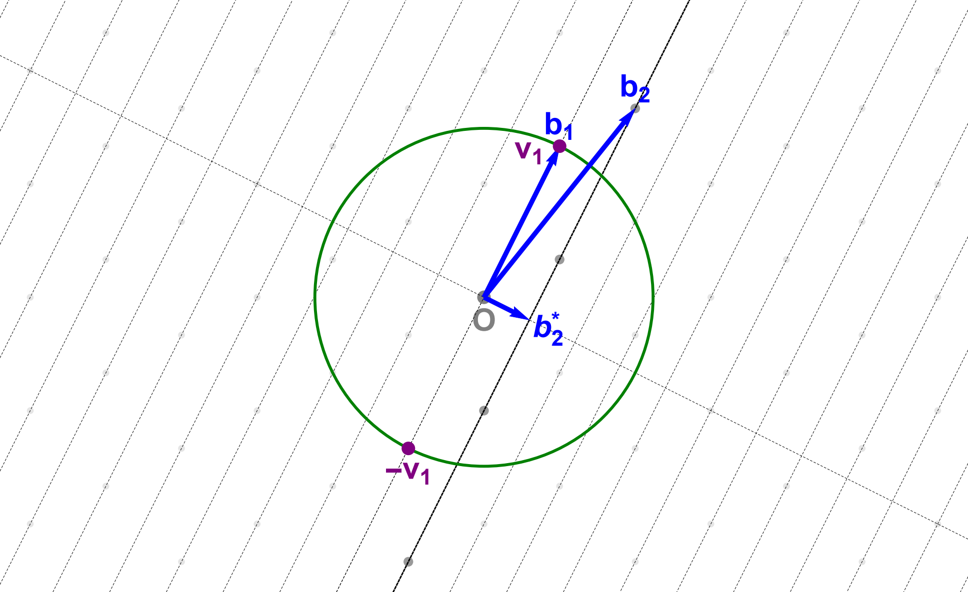
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



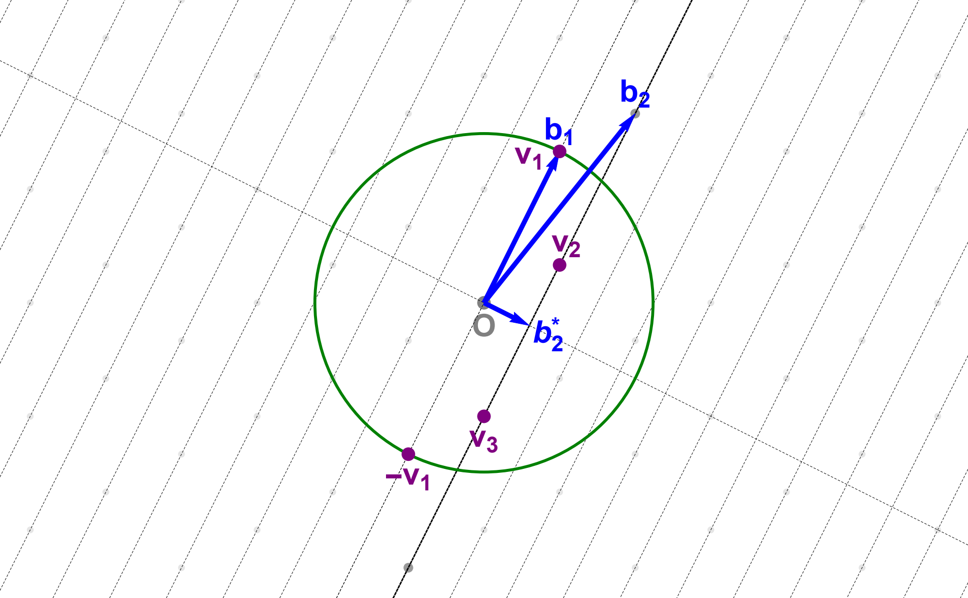
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



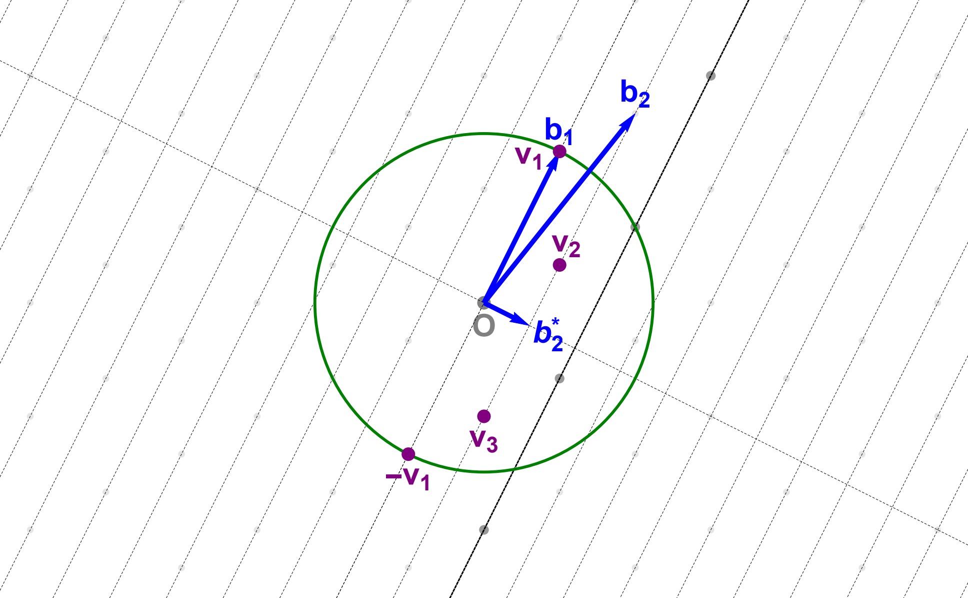
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



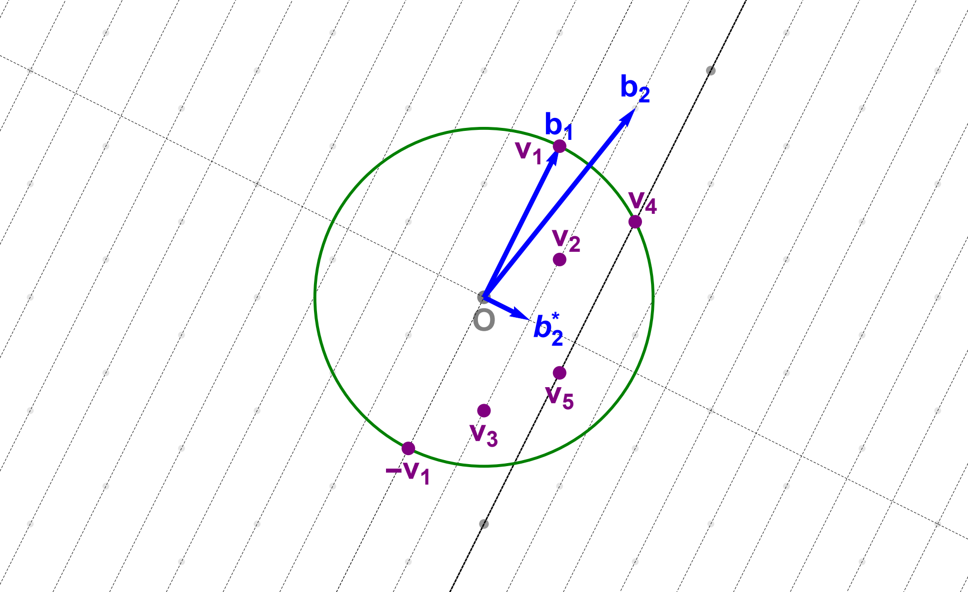
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



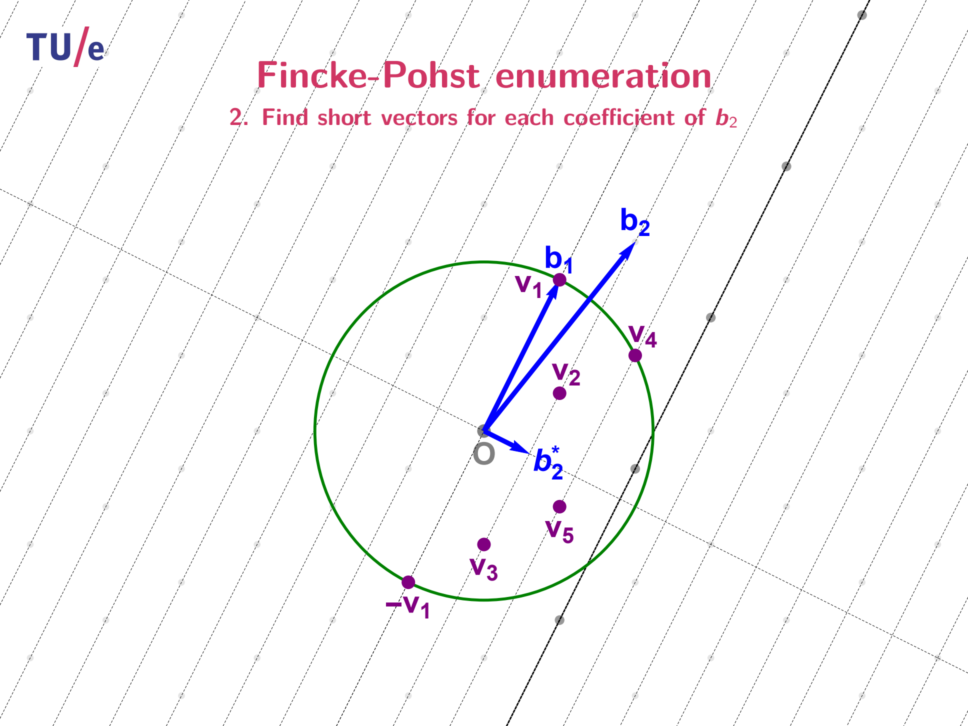
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



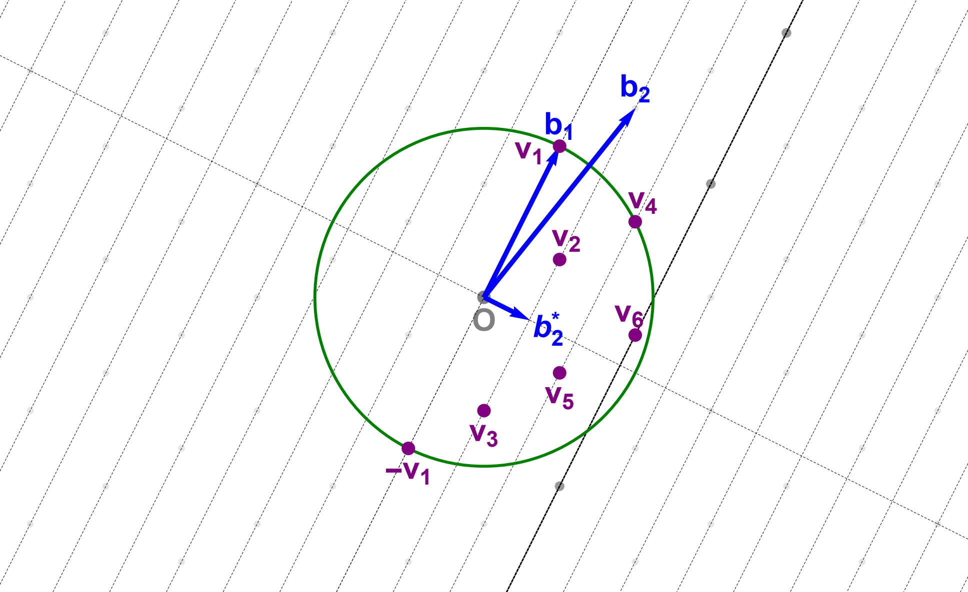
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



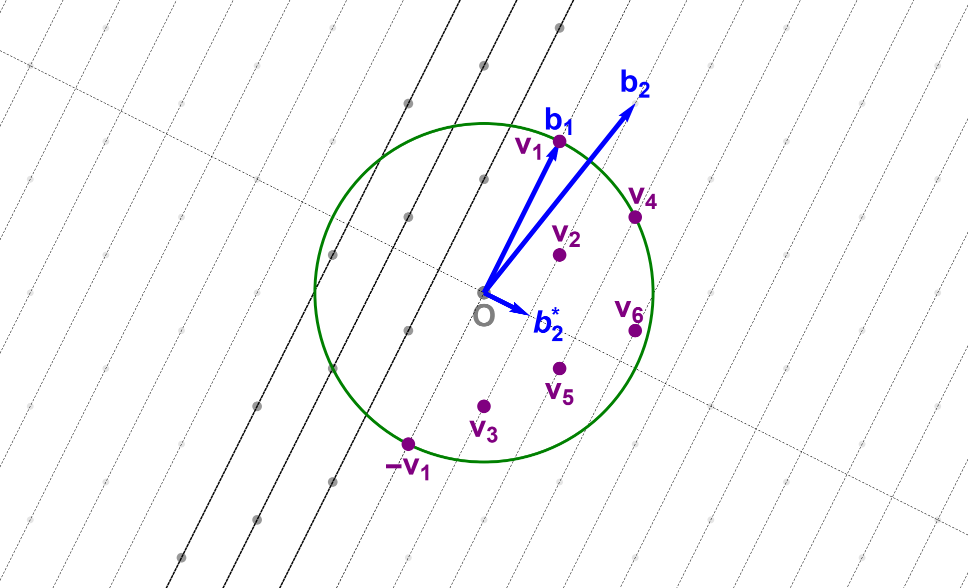
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



Fincke-Pohst enumeration

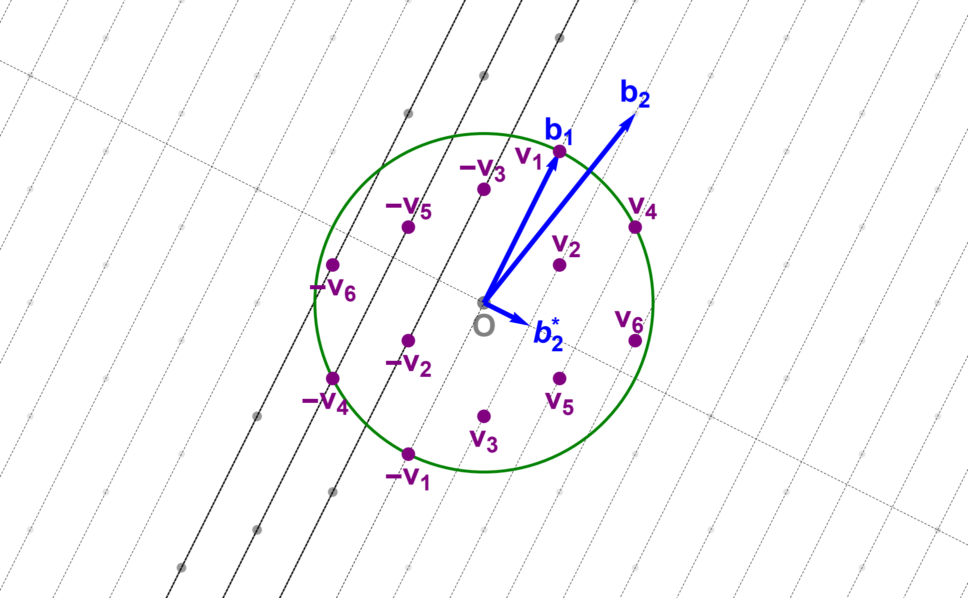
2. Find short vectors for each coefficient of b_2



TU/e

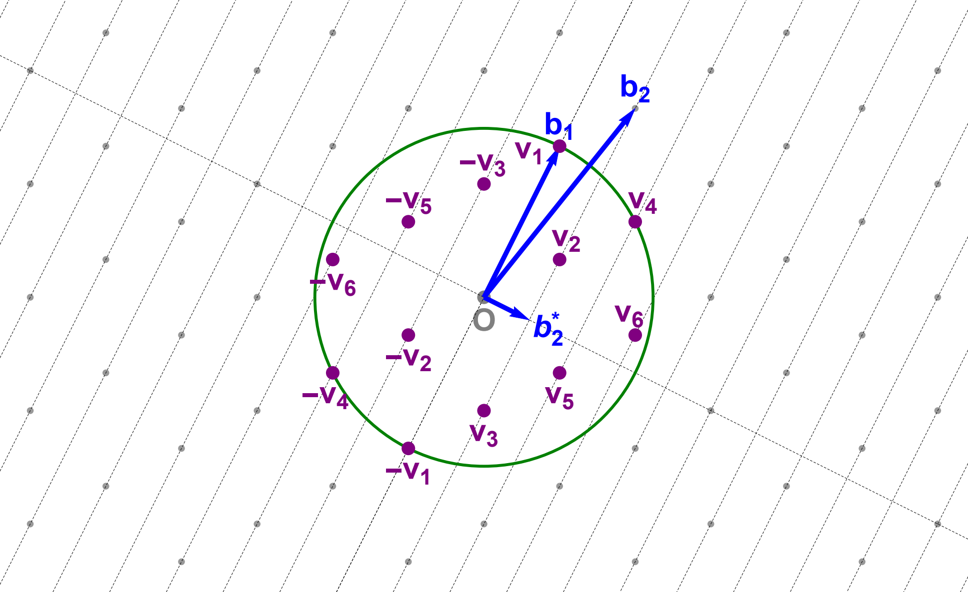
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



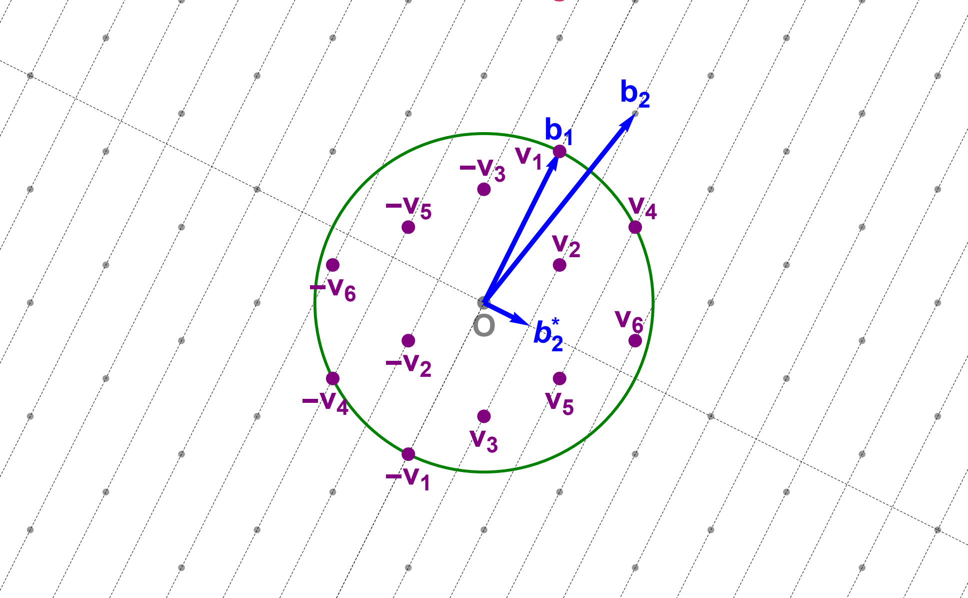
Fincke-Pohst enumeration

2. Find short vectors for each coefficient of b_2



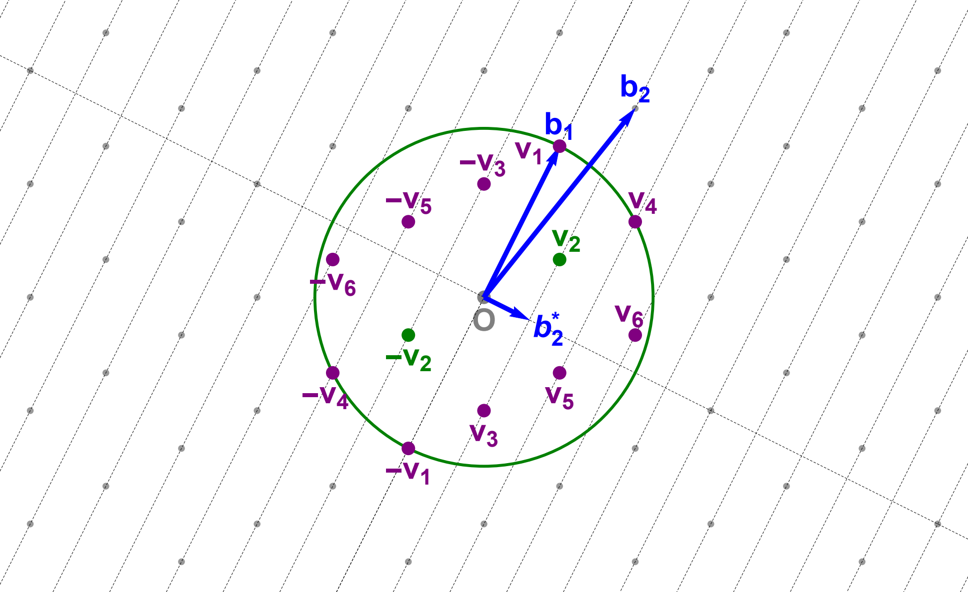
Fincke-Pohst enumeration

3. Find a shortest vector among all found vectors



Fincke-Pohst enumeration

3. Find a shortest vector among all found vectors

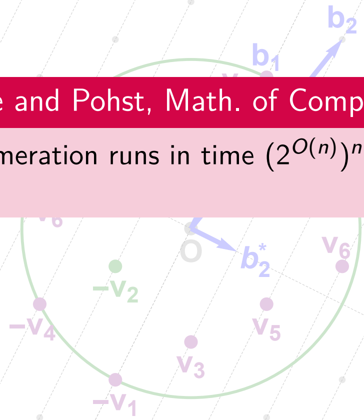


Fincke-Pohst enumeration

Overview

Fincke-Pohst enumeration

Overview

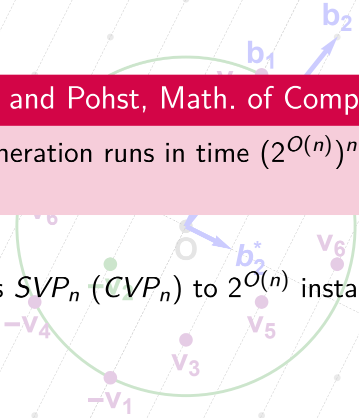


Theorem (Fincke and Pohst, Math. of Comp. '85)

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

Fincke-Pohst enumeration

Overview



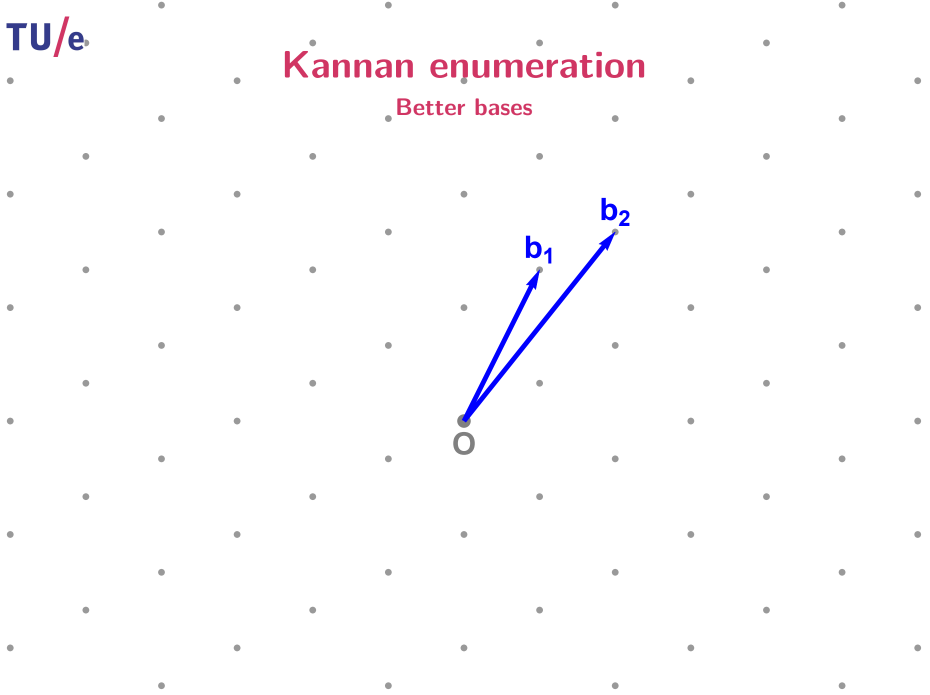
Theorem (Fincke and Pohst, Math. of Comp. '85)

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}

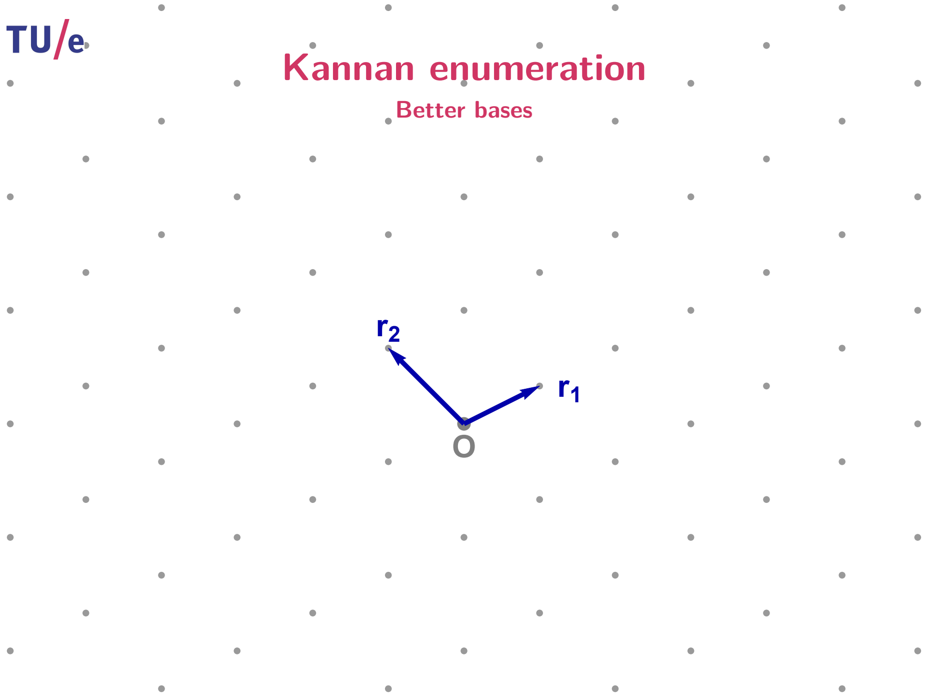
Kannan enumeration

Better bases



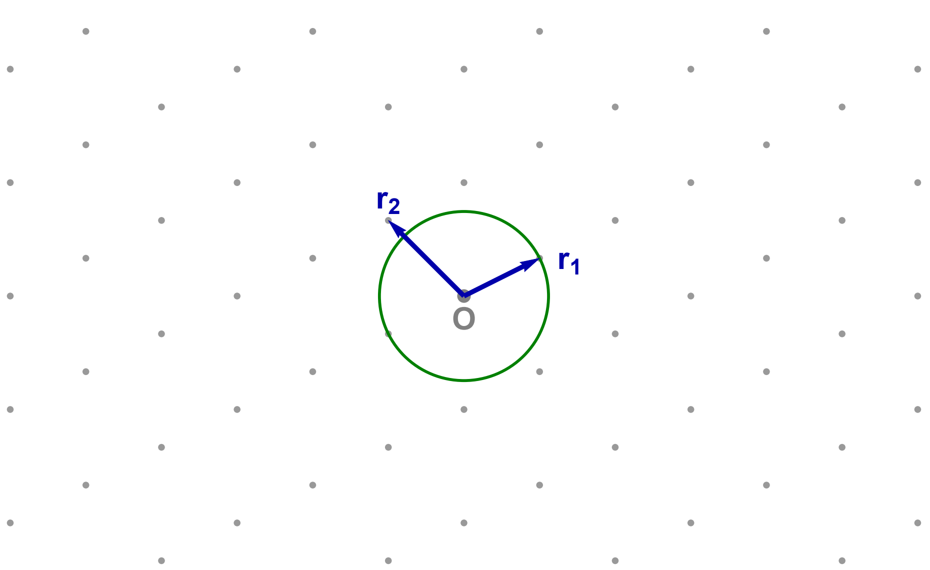
Kannan enumeration

Better bases



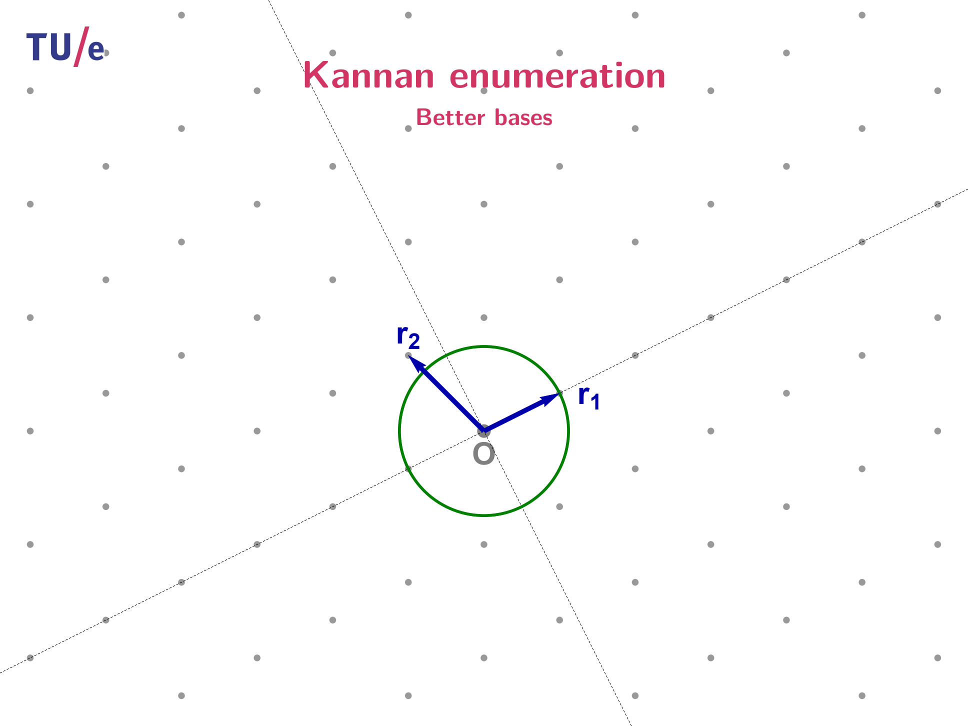
Kannan enumeration

Better bases



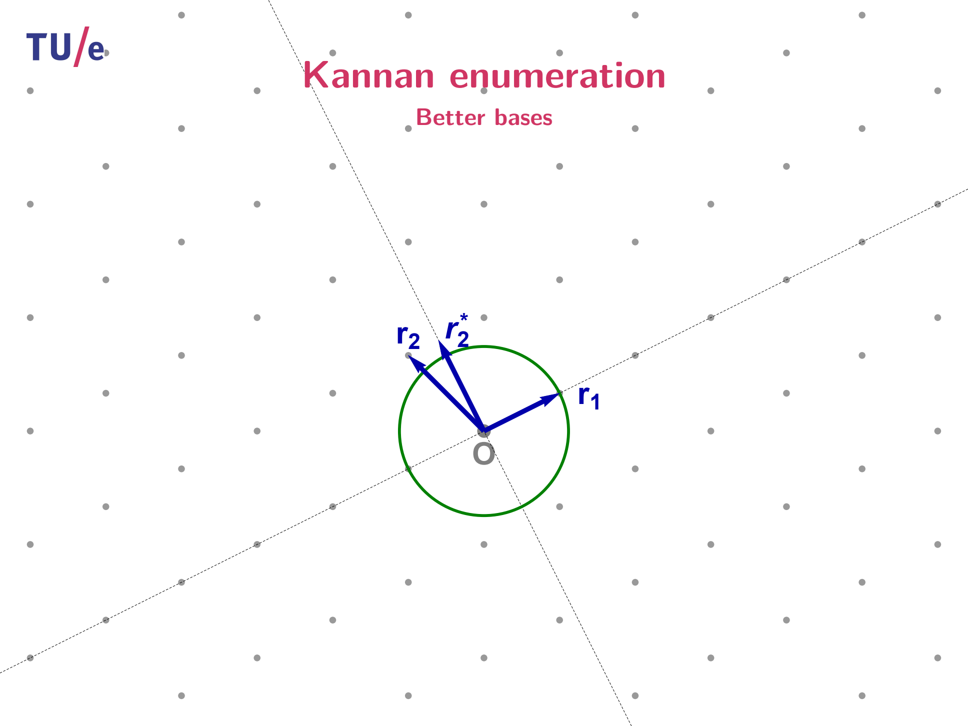
Kannan enumeration

Better bases



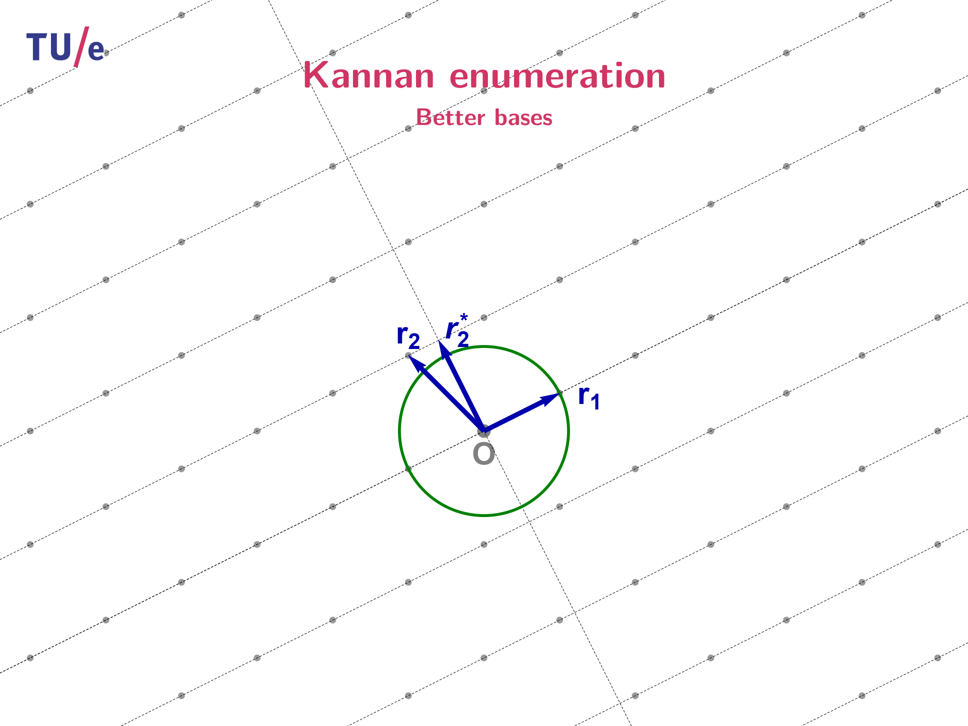
Kannan enumeration

Better bases



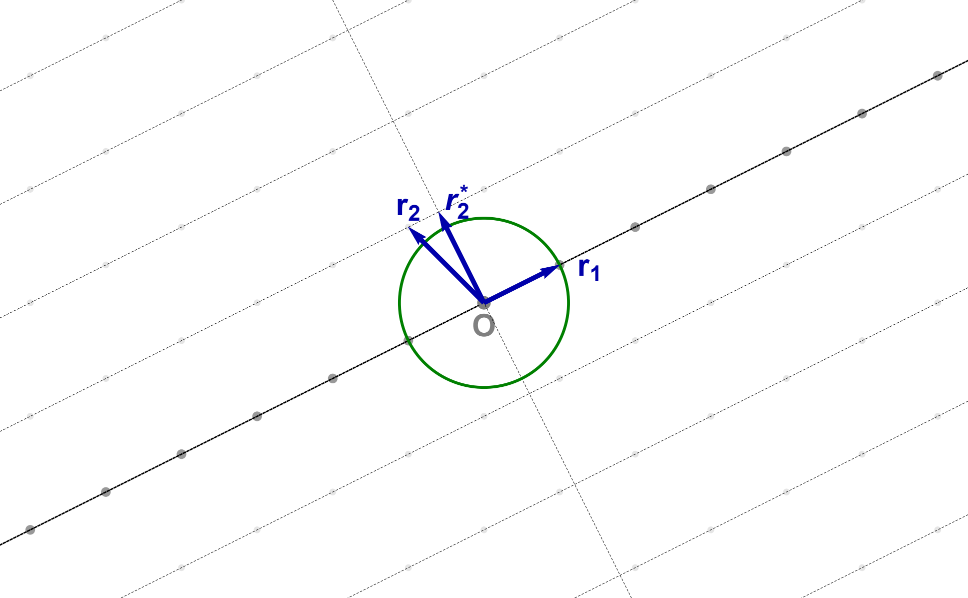
Kannan enumeration

Better bases



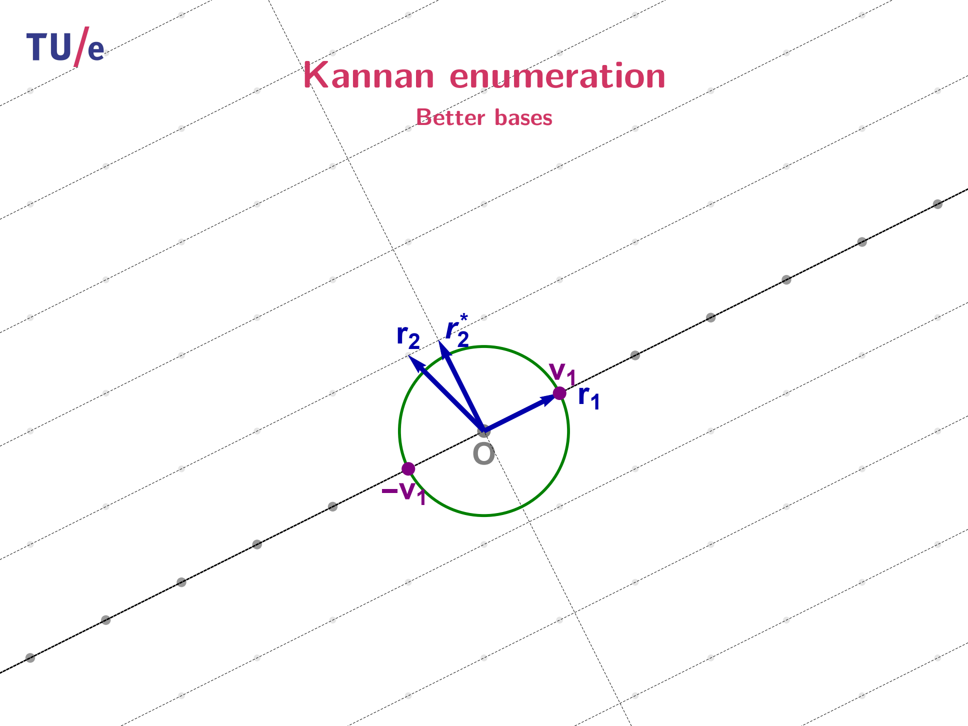
Kannan enumeration

Better bases



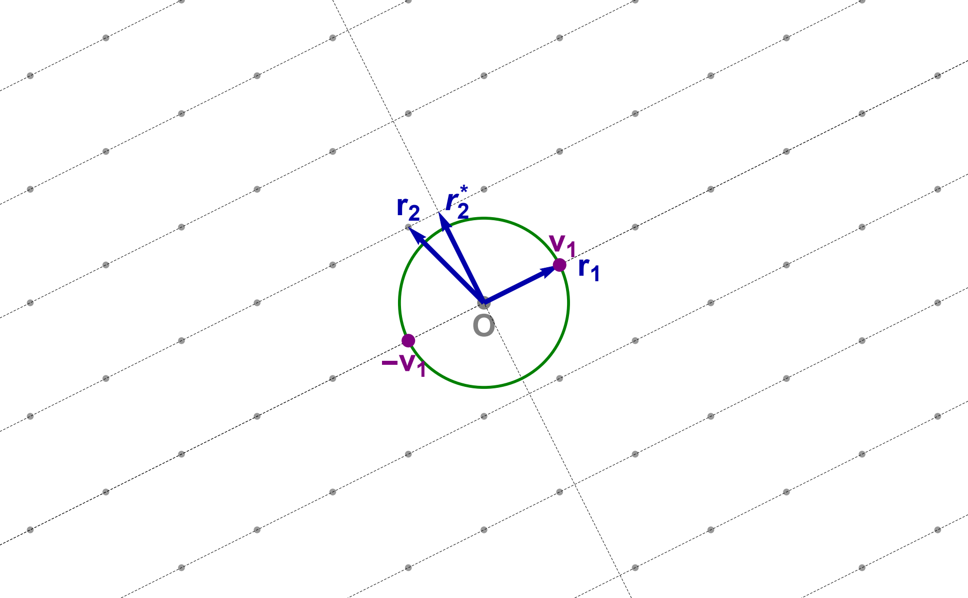
Kannan enumeration

Better bases



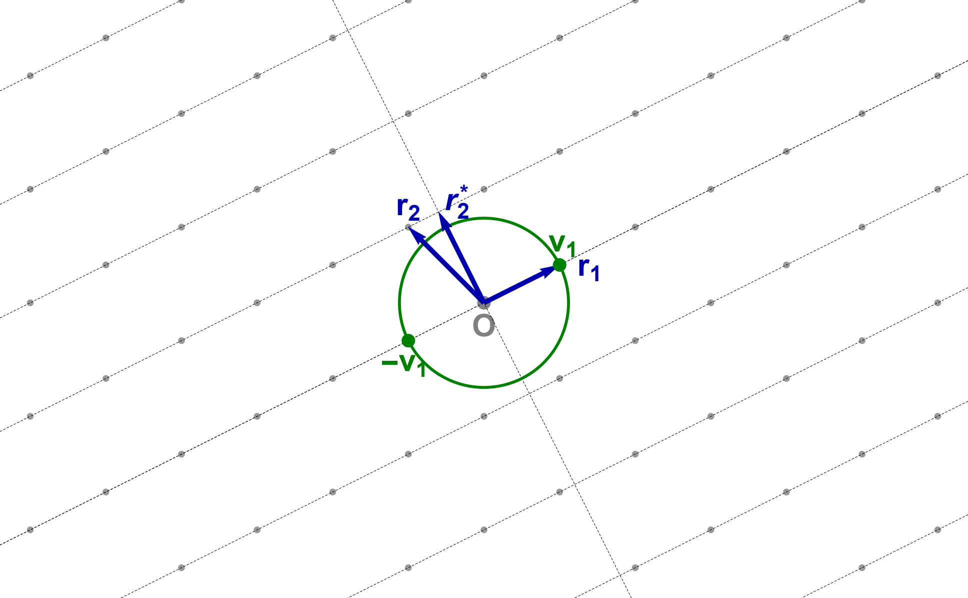
Kannan enumeration

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

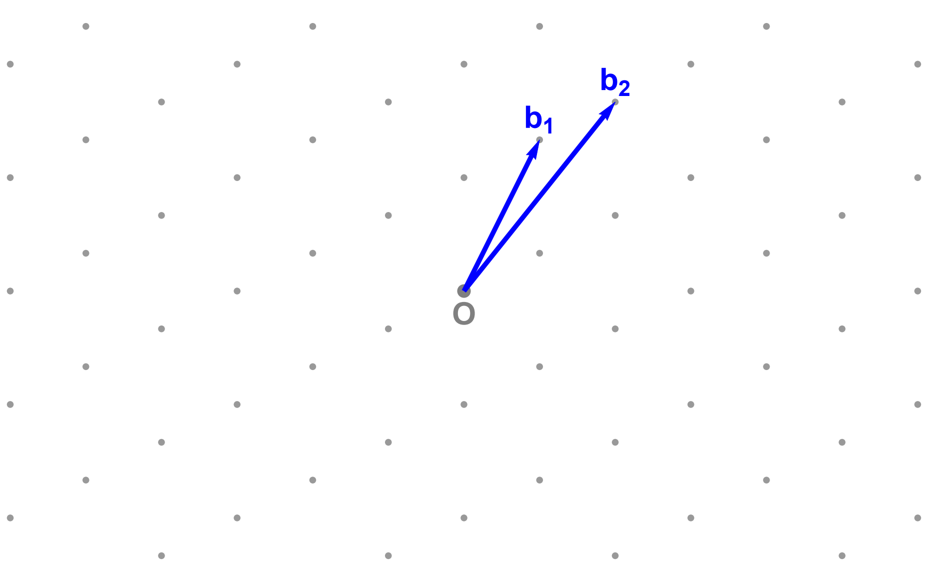
Theorem (Kannan, STOC'83)

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*

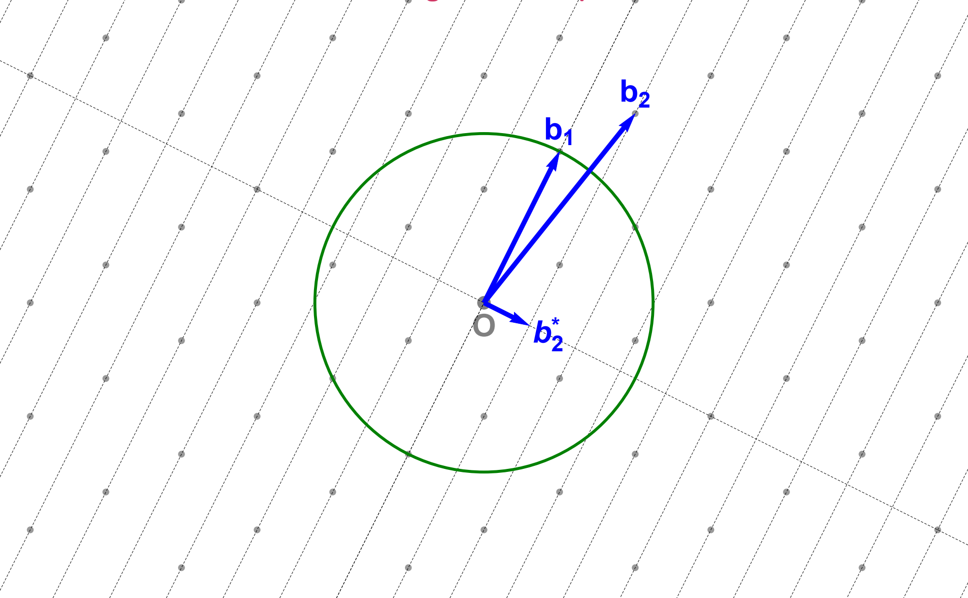
Pruned enumeration

Reducing the search space



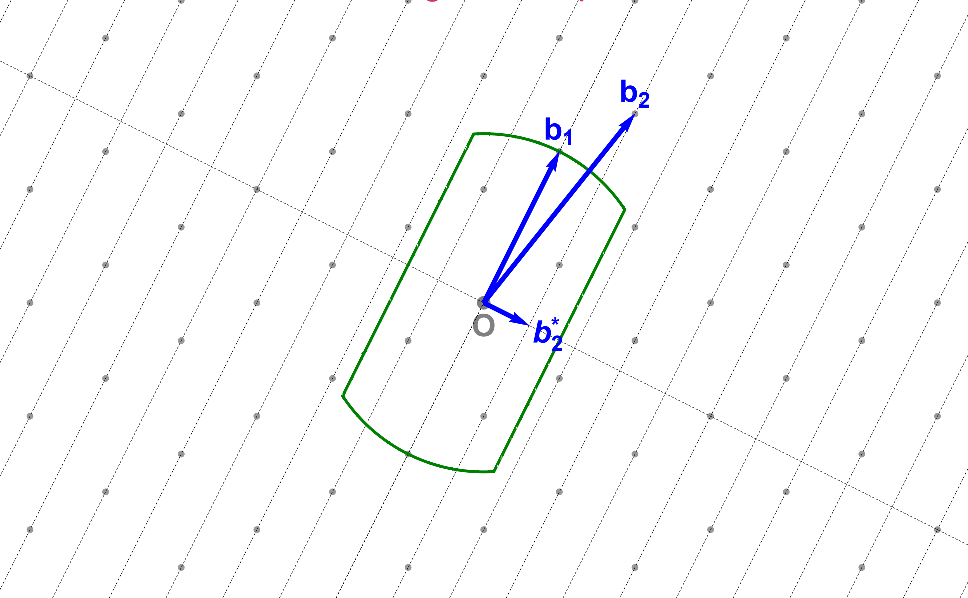
Pruned enumeration

Reducing the search space



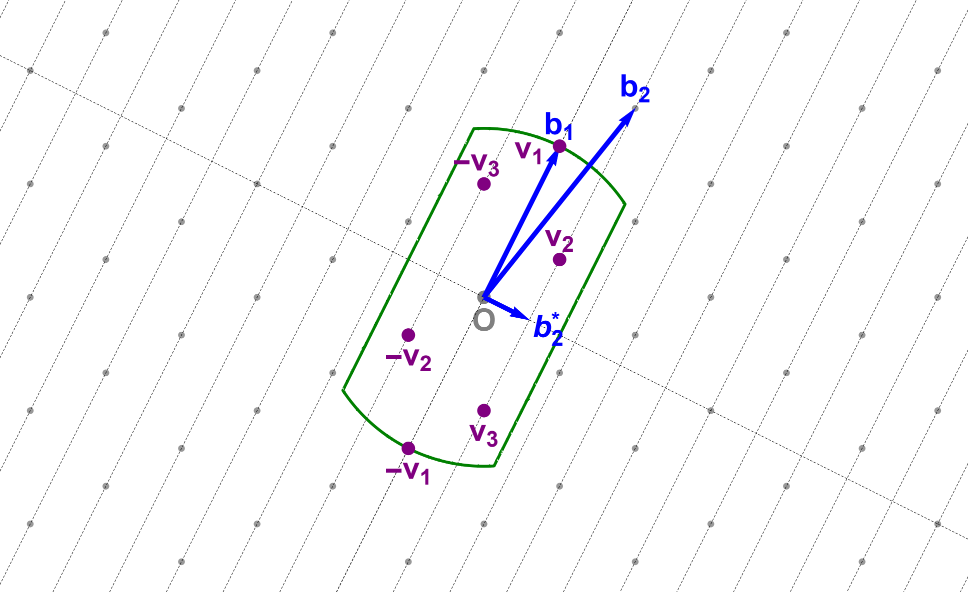
Pruned enumeration

Reducing the search space



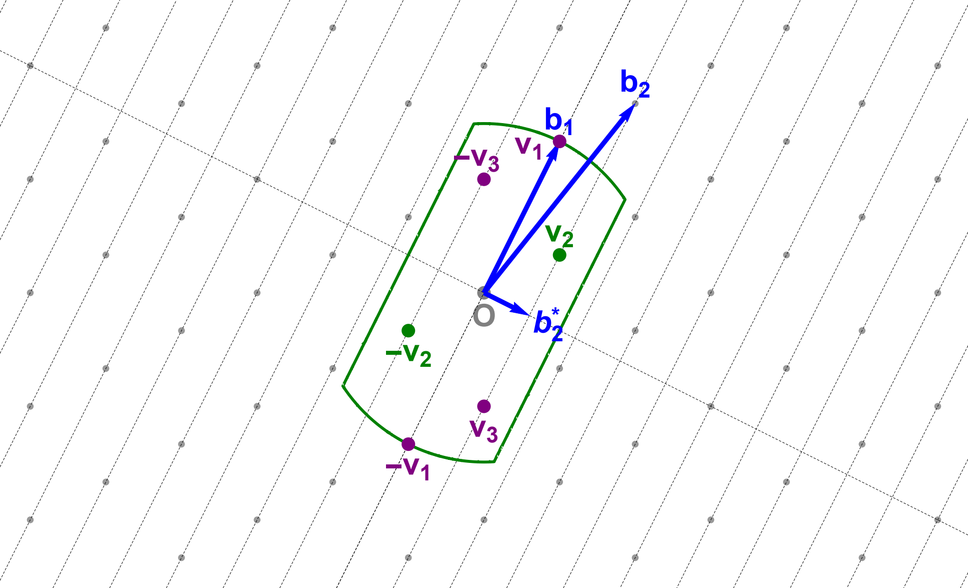
Pruned enumeration

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

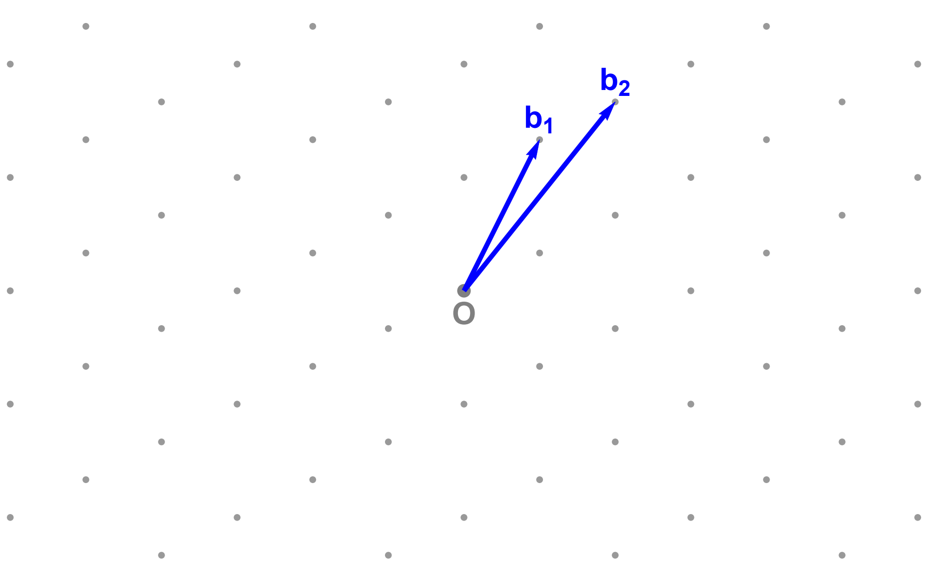
Enumeration algorithms
- Fincke-Pohst enumeration
- Kannan enumeration
- Pruning the enumeration tree

The Voronoi cell algorithm

Sieving algorithms
- Nguyen-Vidick sieve
- Multiple levels
- GaussSieve
- Locality-sensitive hashing

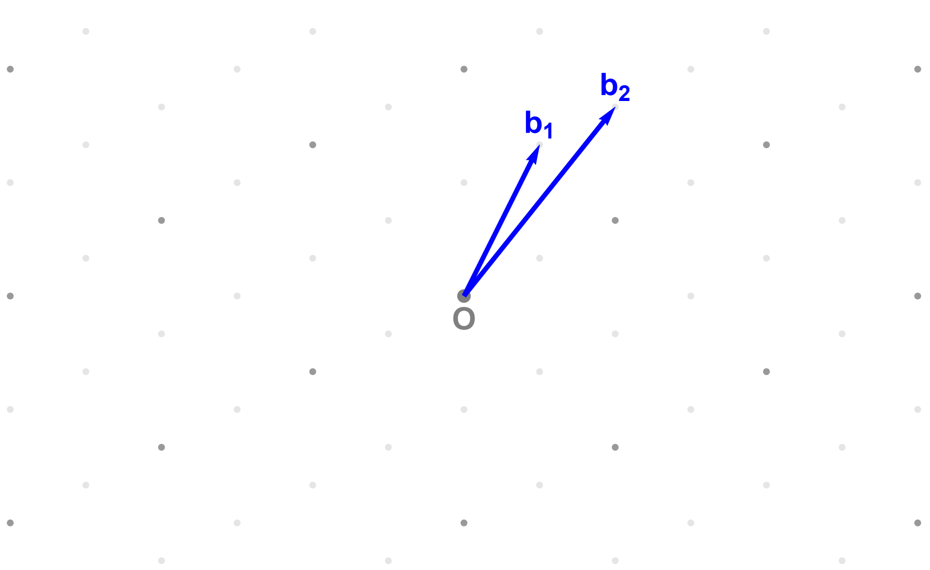
The Voronoi cell algorithm

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



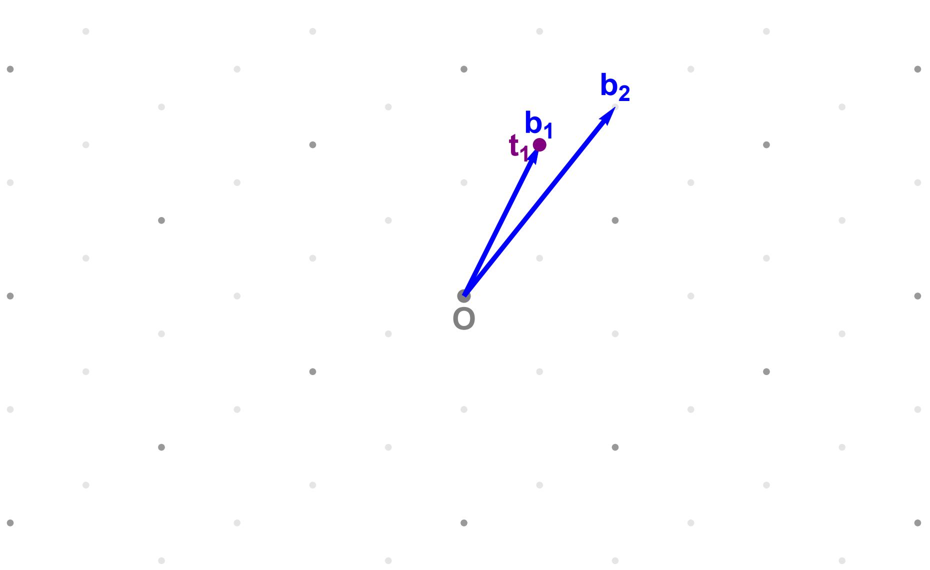
The Voronoi cell algorithm

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



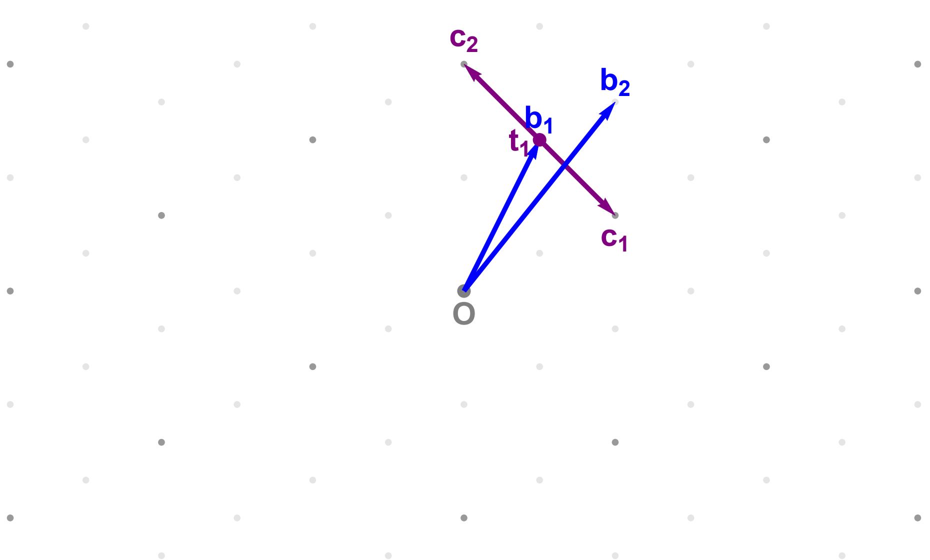
The Voronoi cell algorithm

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



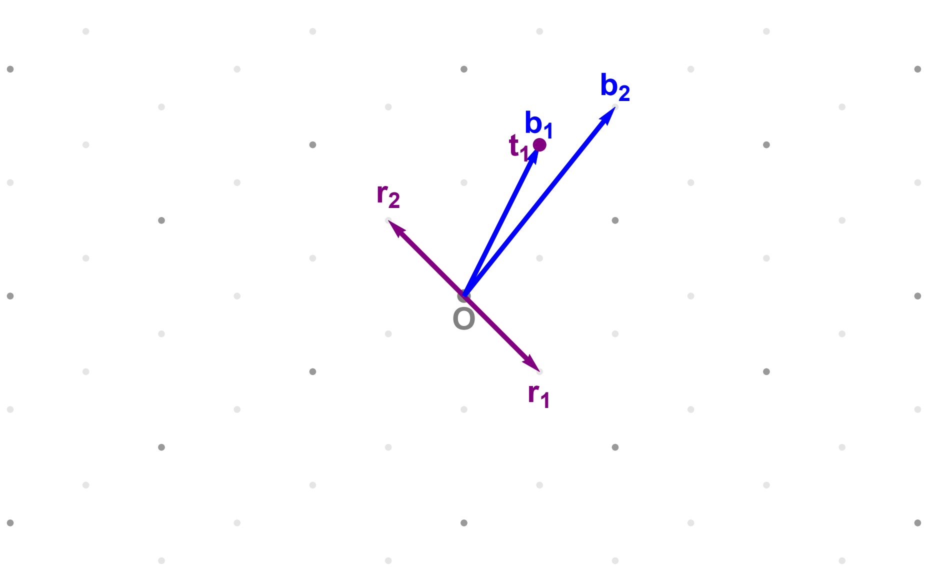
The Voronoi cell algorithm

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



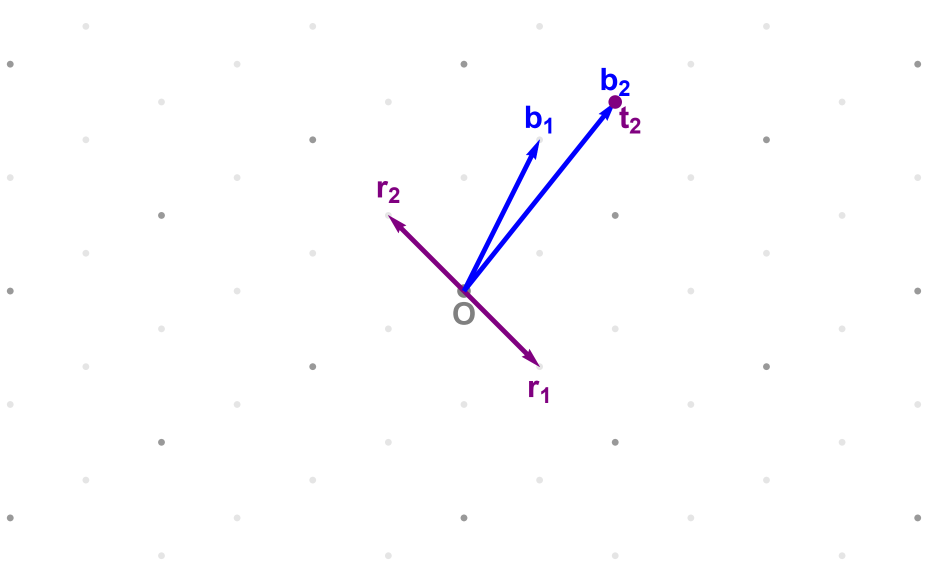
The Voronoi cell algorithm

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



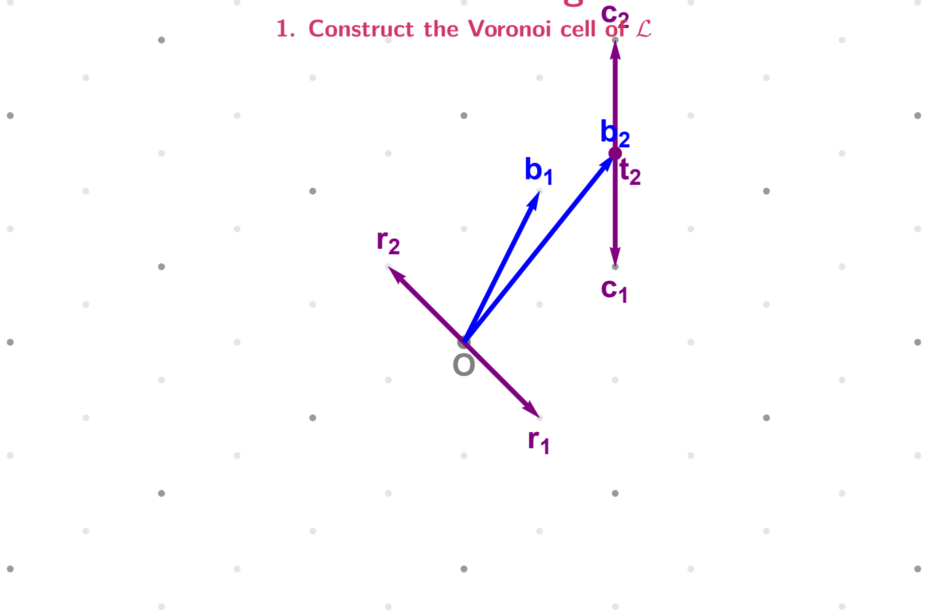
The Voronoi cell algorithm

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



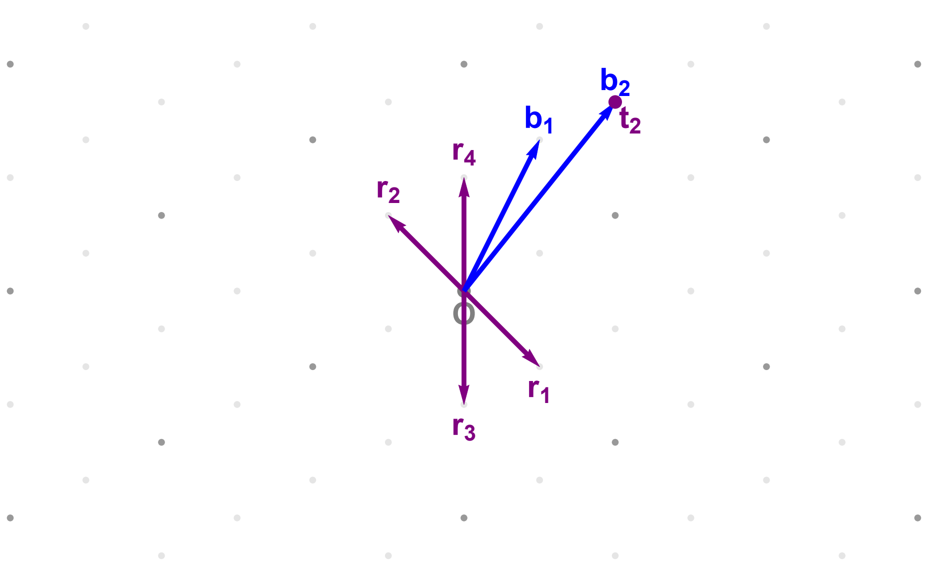
The Voronoi cell algorithm

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



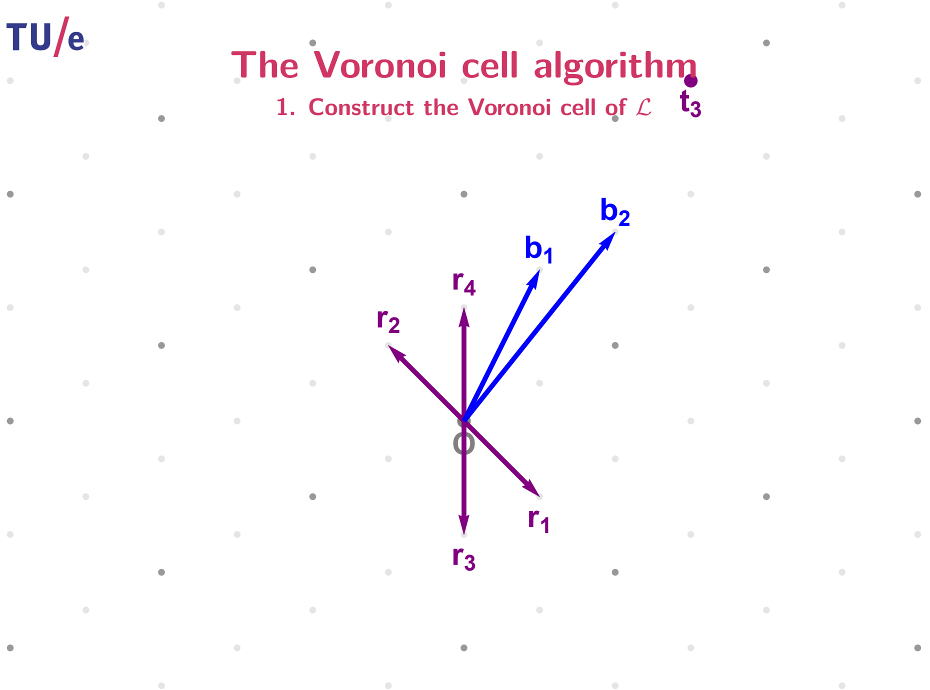
The Voronoi cell algorithm

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



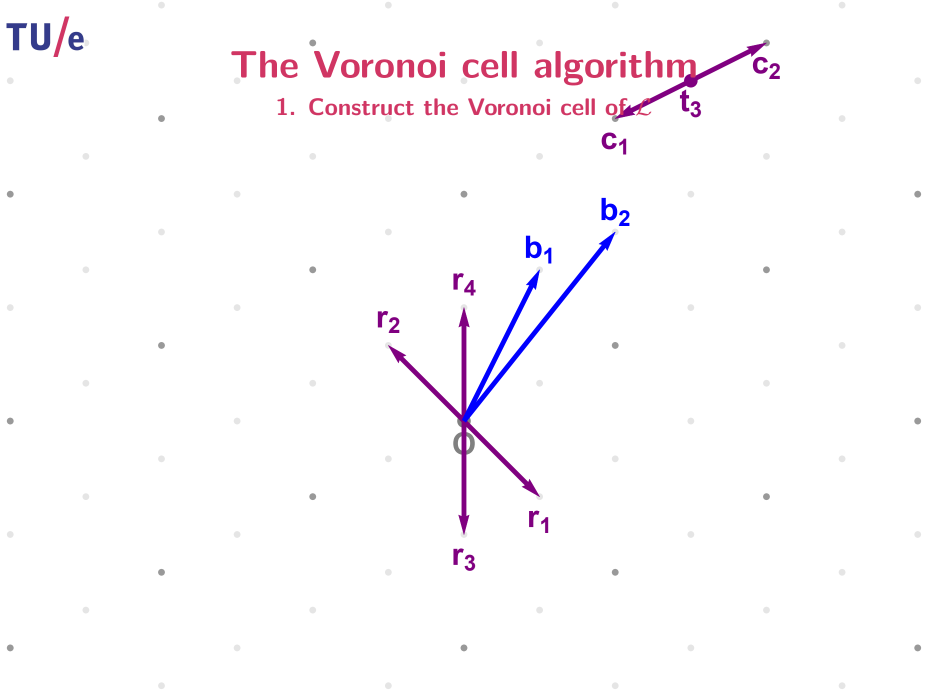
The Voronoi cell algorithm

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



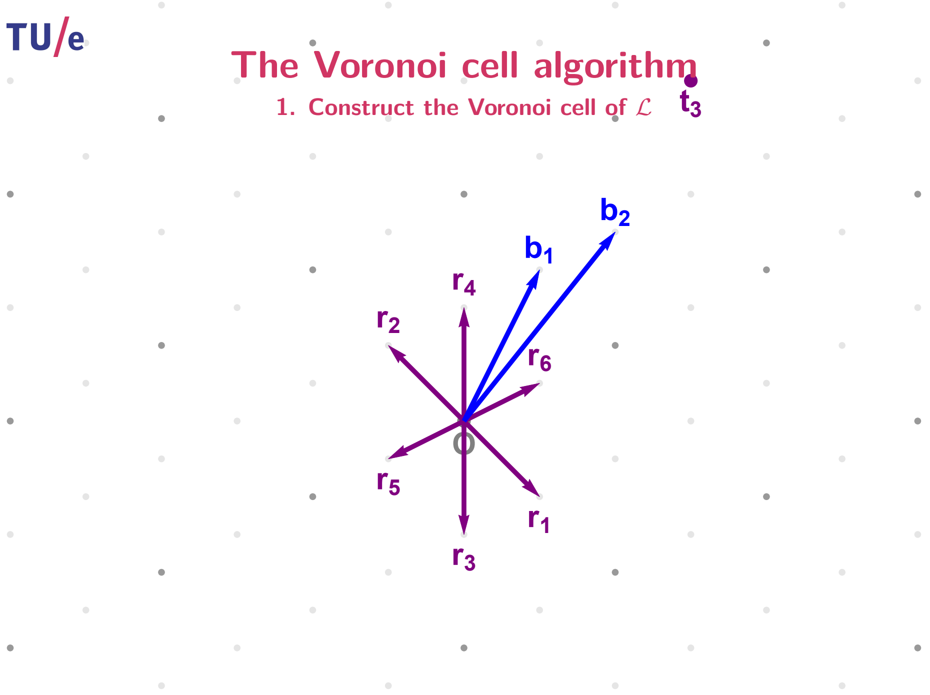
The Voronoi cell algorithm

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



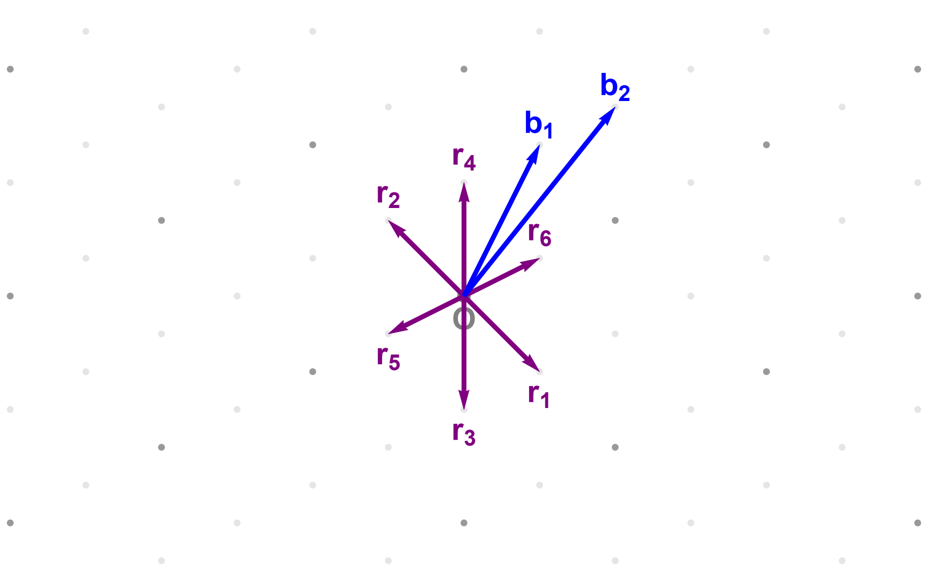
The Voronoi cell algorithm

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



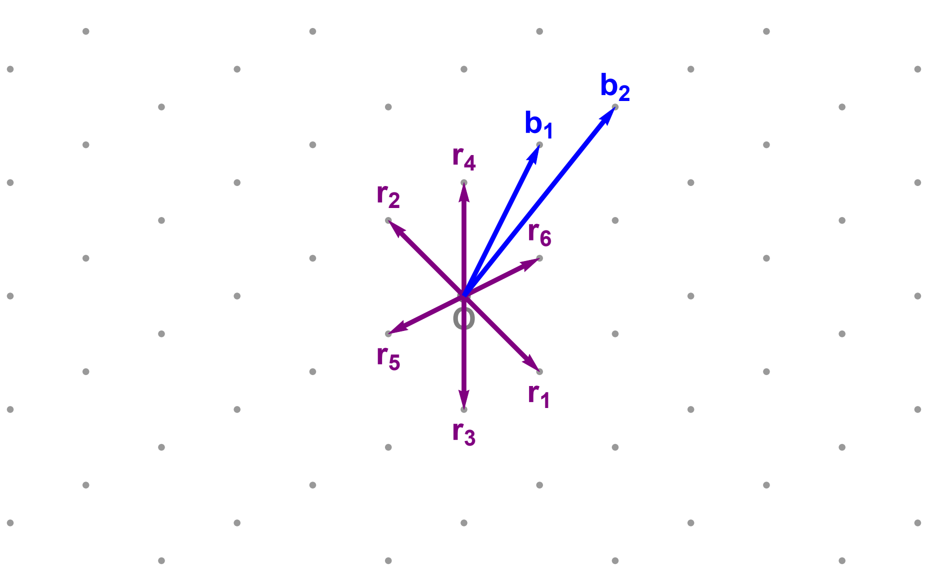
The Voronoi cell algorithm

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



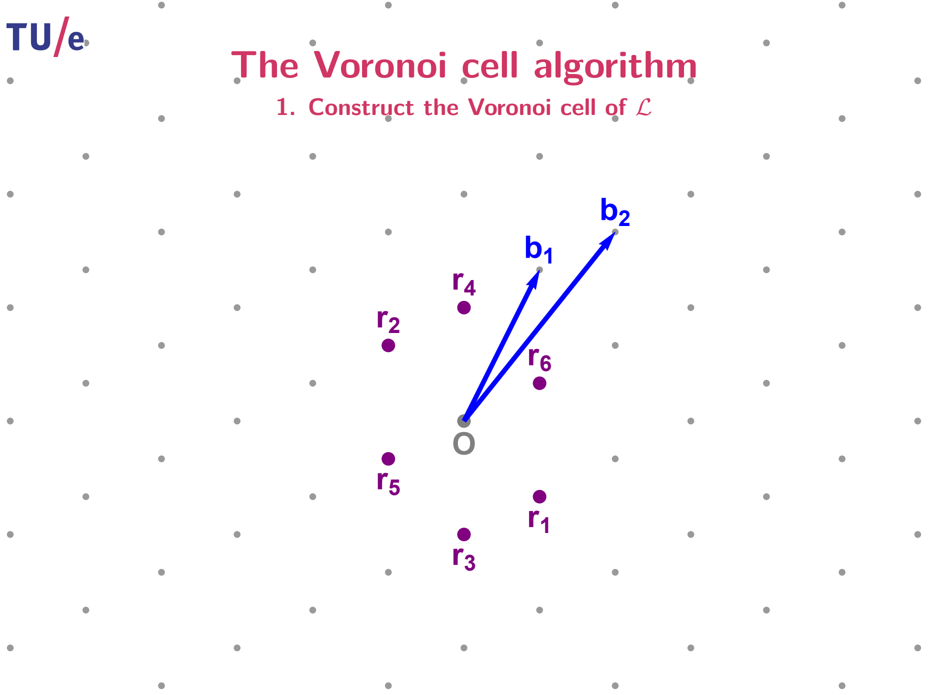
The Voronoi cell algorithm

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



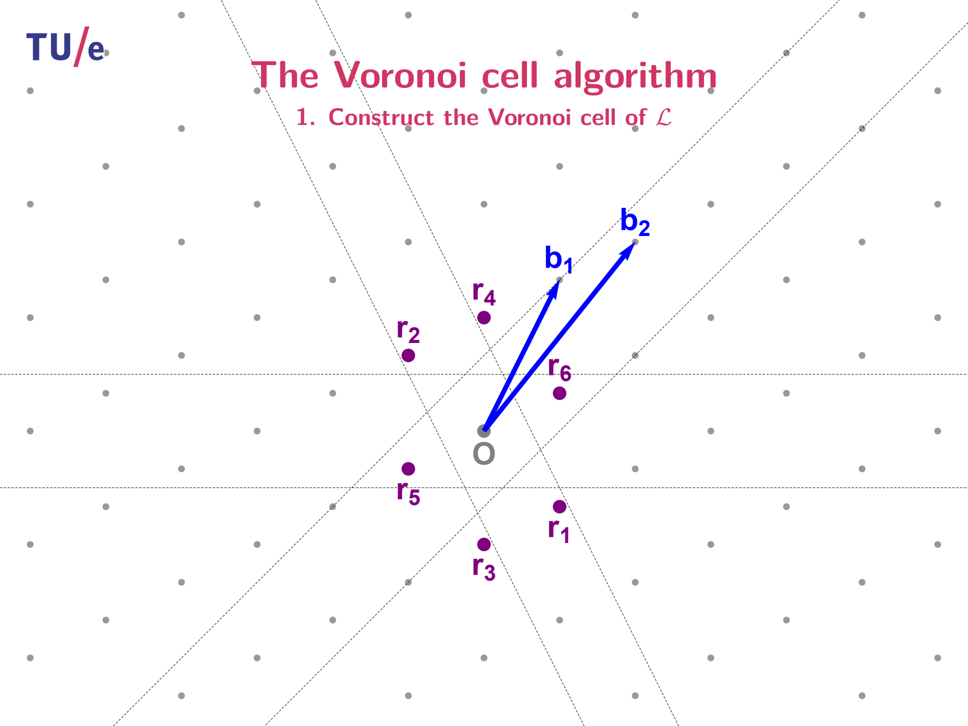
The Voronoi cell algorithm

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



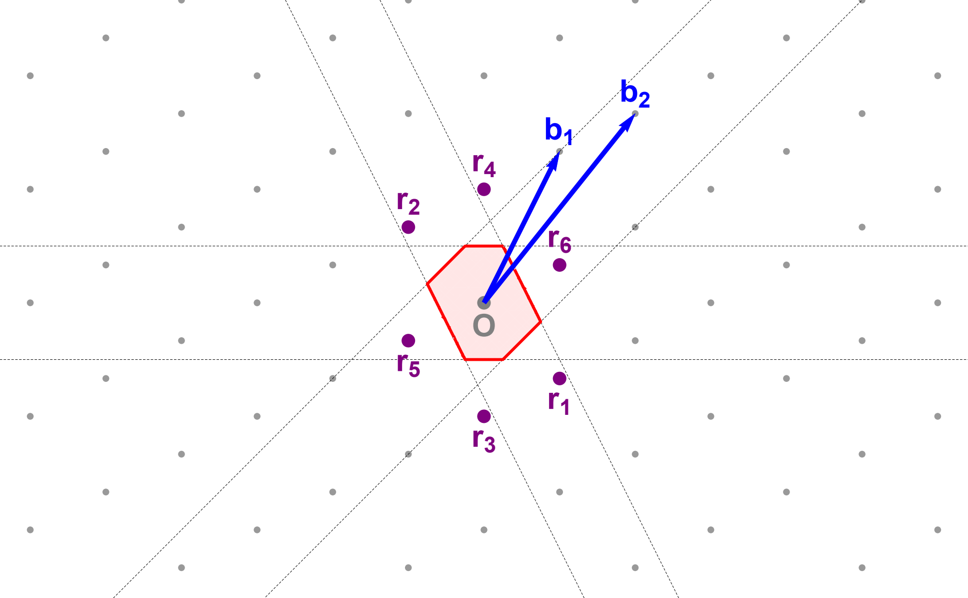
The Voronoi cell algorithm

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



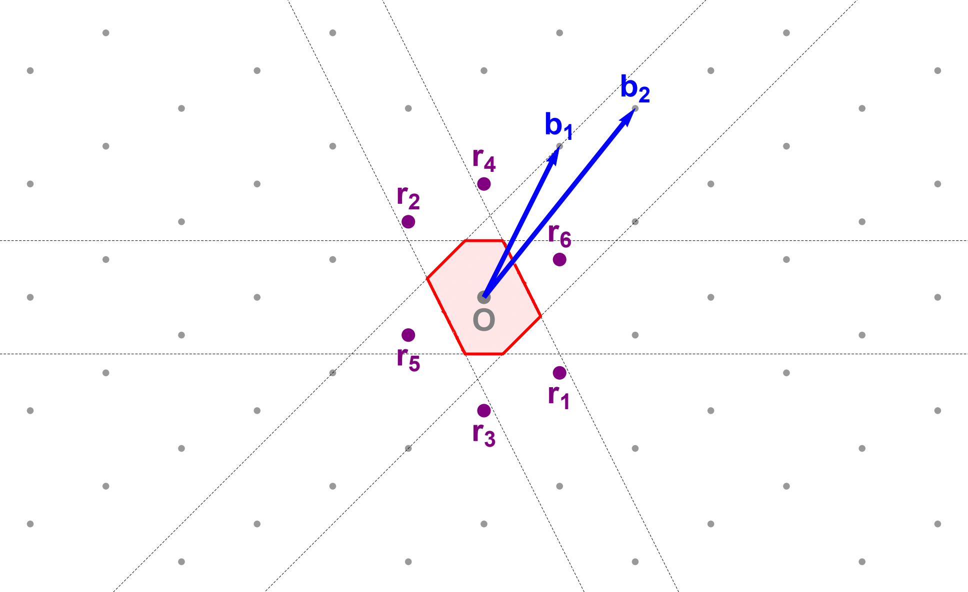
The Voronoi cell algorithm

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



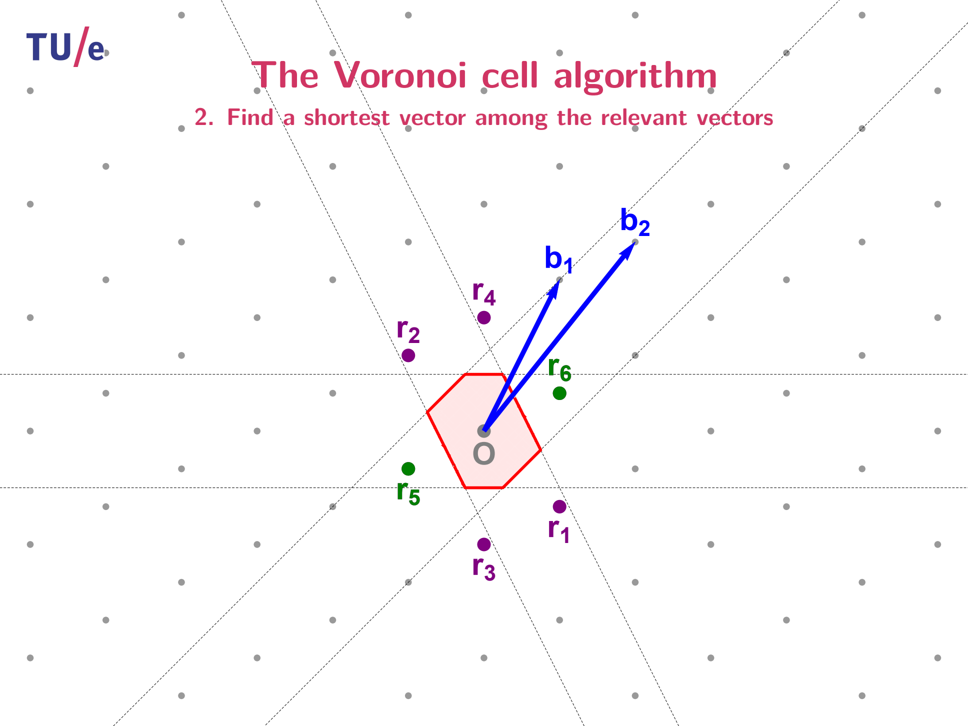
The Voronoi cell algorithm

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

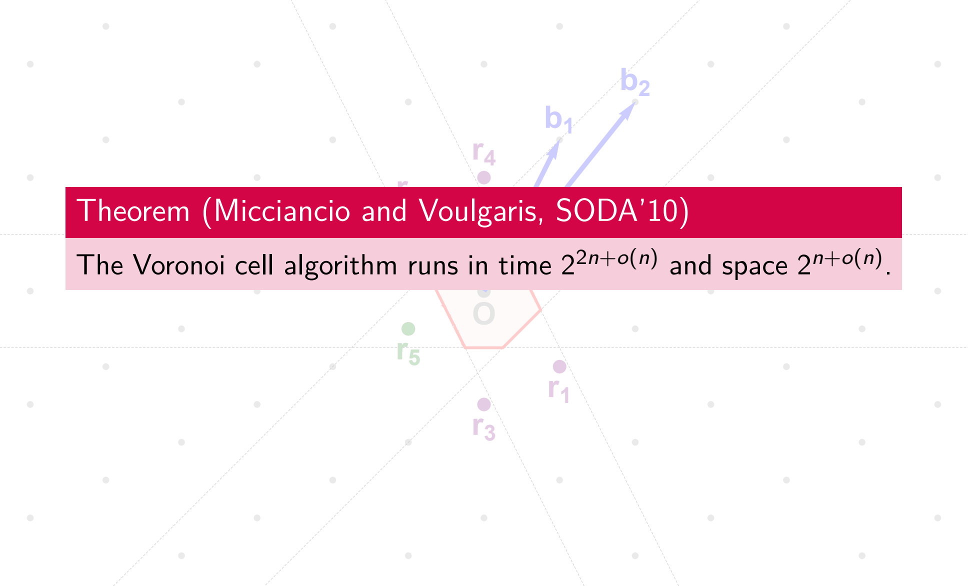
Overview

The Voronoi cell algorithm

Overview

Theorem (Micciancio and Voulgaris, SODA'10)

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



Outline

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

The Voronoi cell algorithm

Sieving algorithms
- Nguyen-Vidick sieve
- Multiple levels
- GaussSieve
- Locality-sensitive hashing

Nguyen-Vidick sieve

1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve

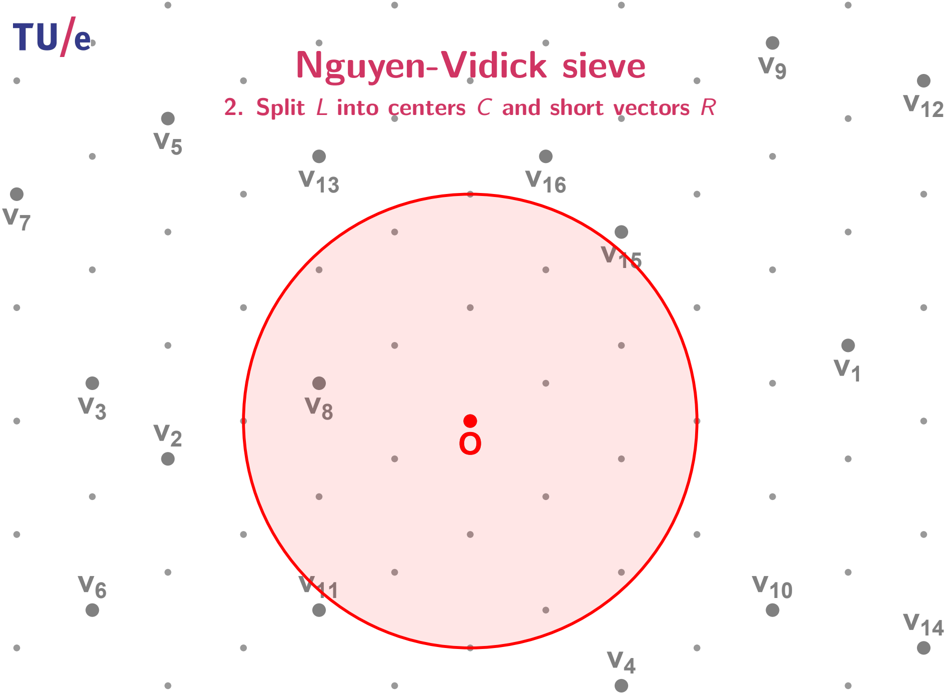
1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve

2. Split L into centers C and short vectors R

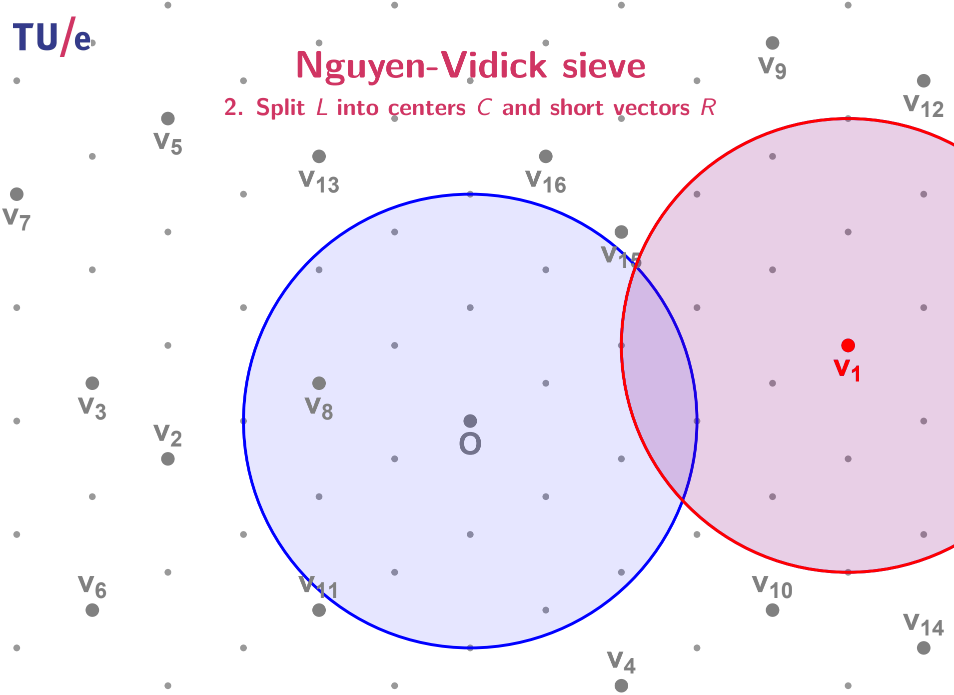
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



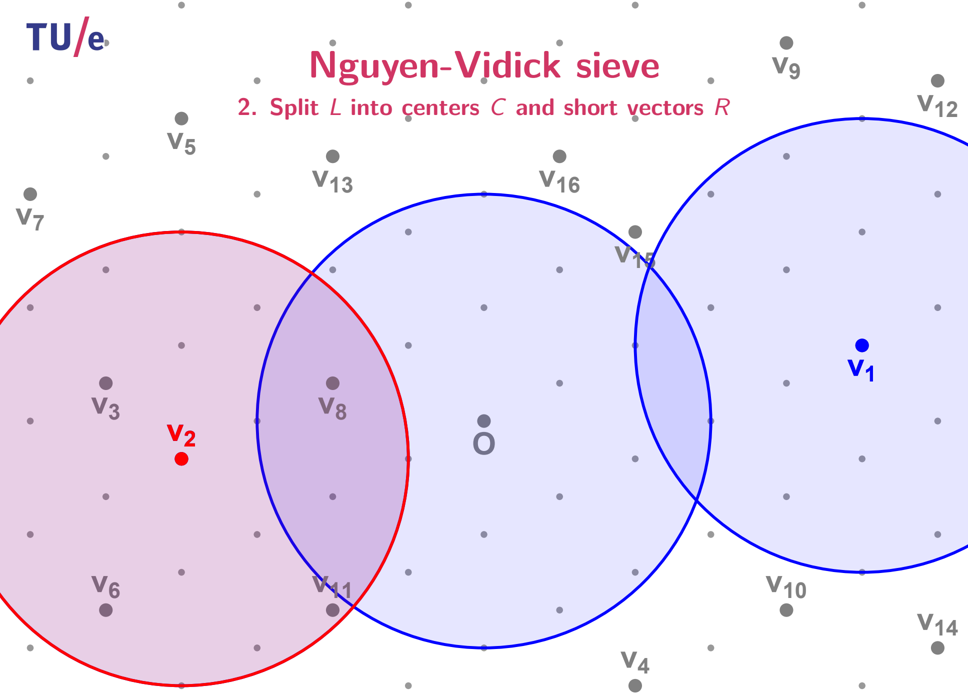
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



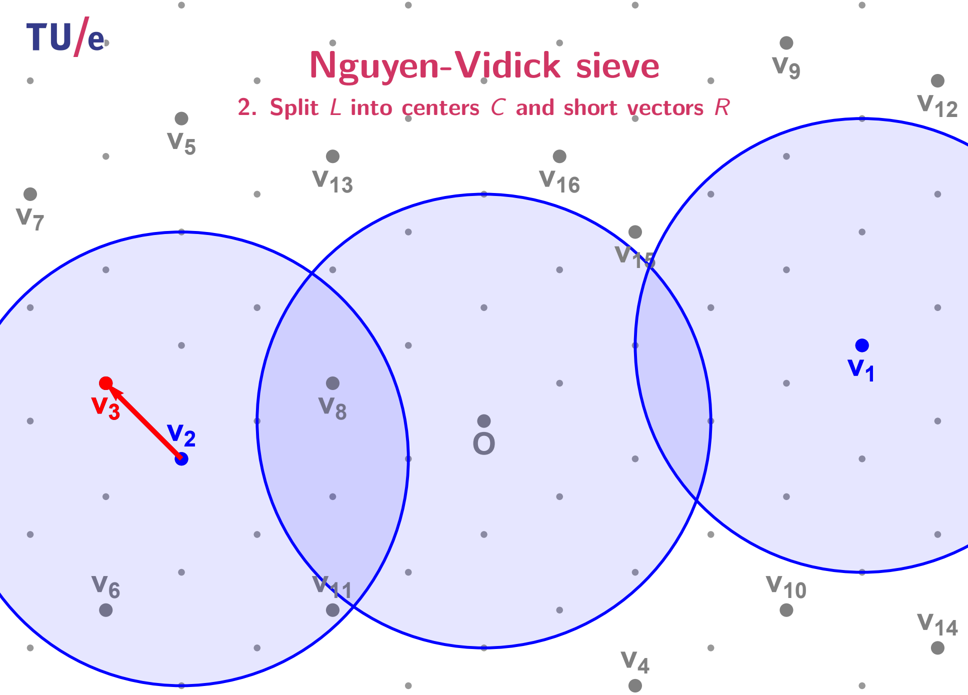
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



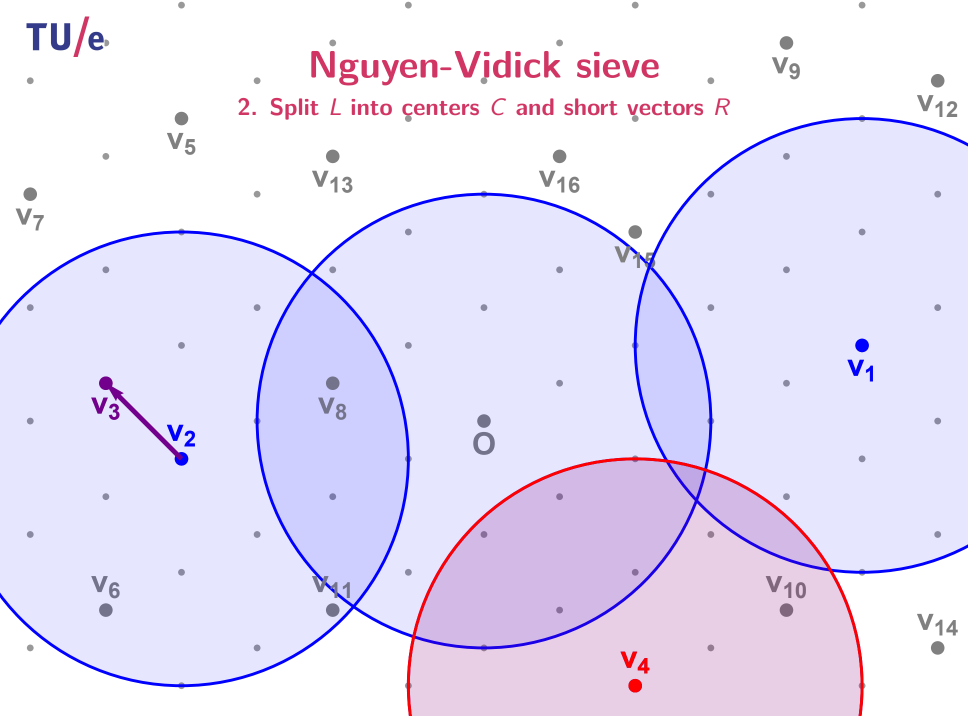
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



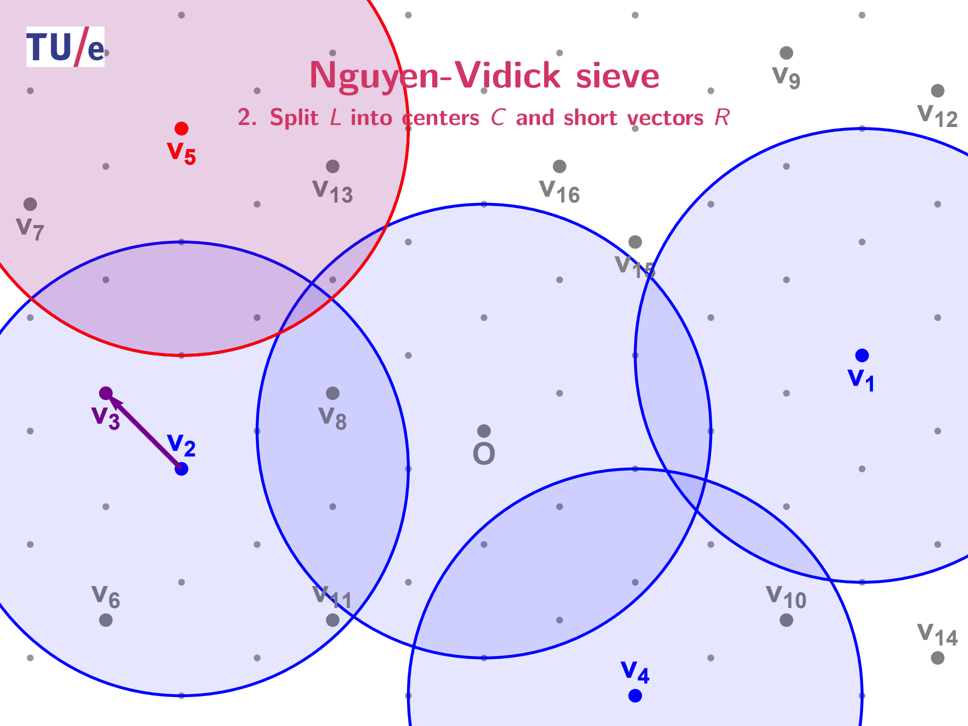
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



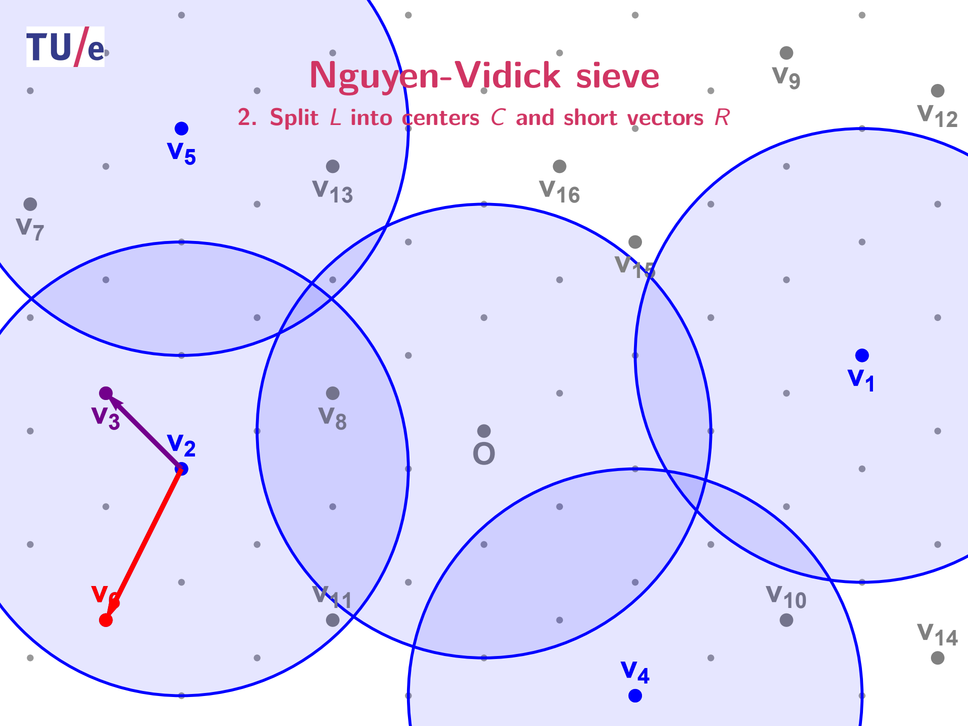
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



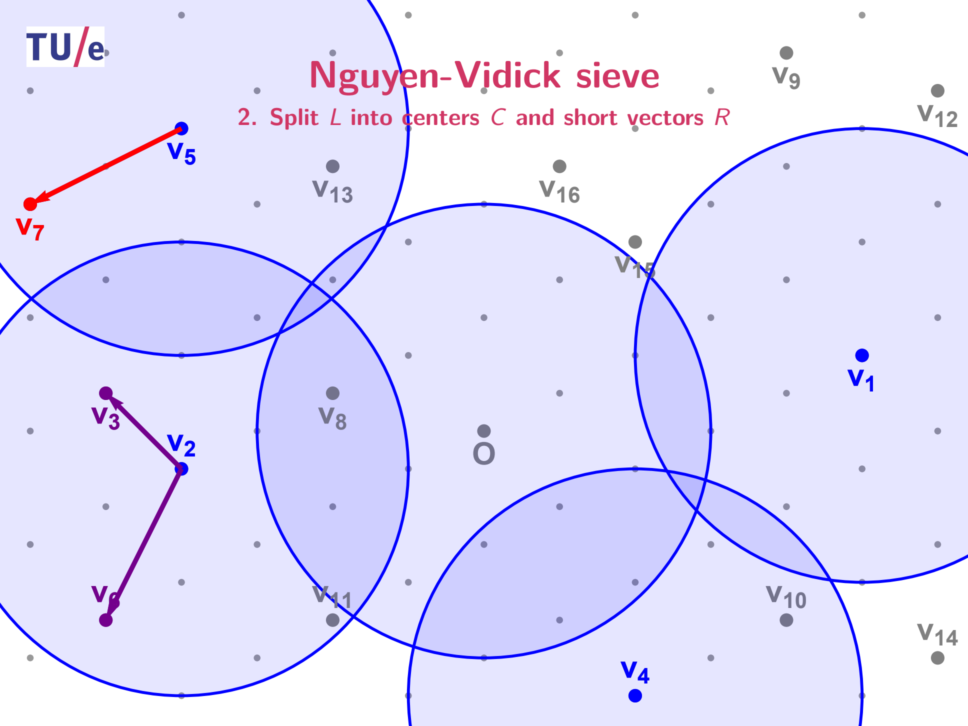
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



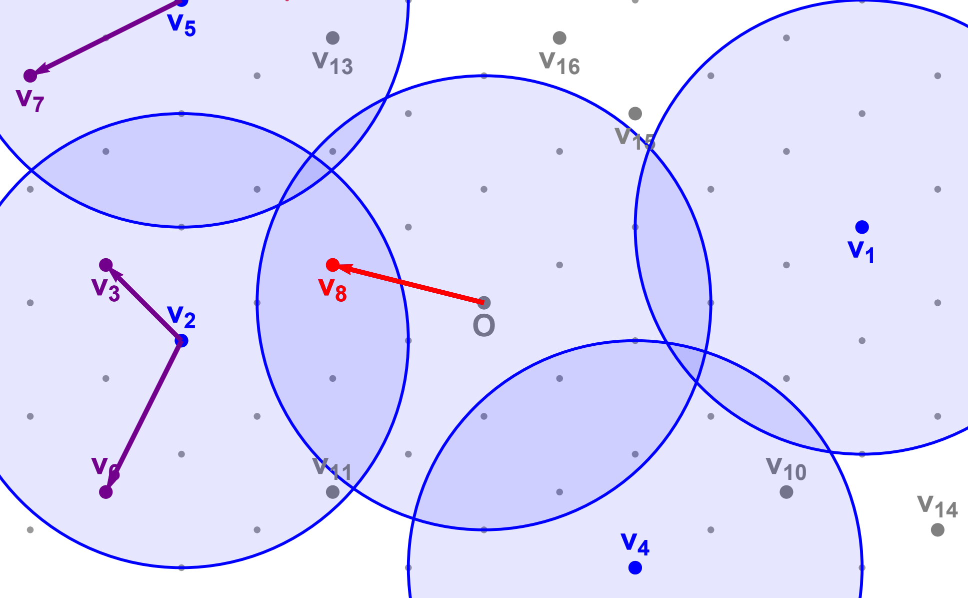
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



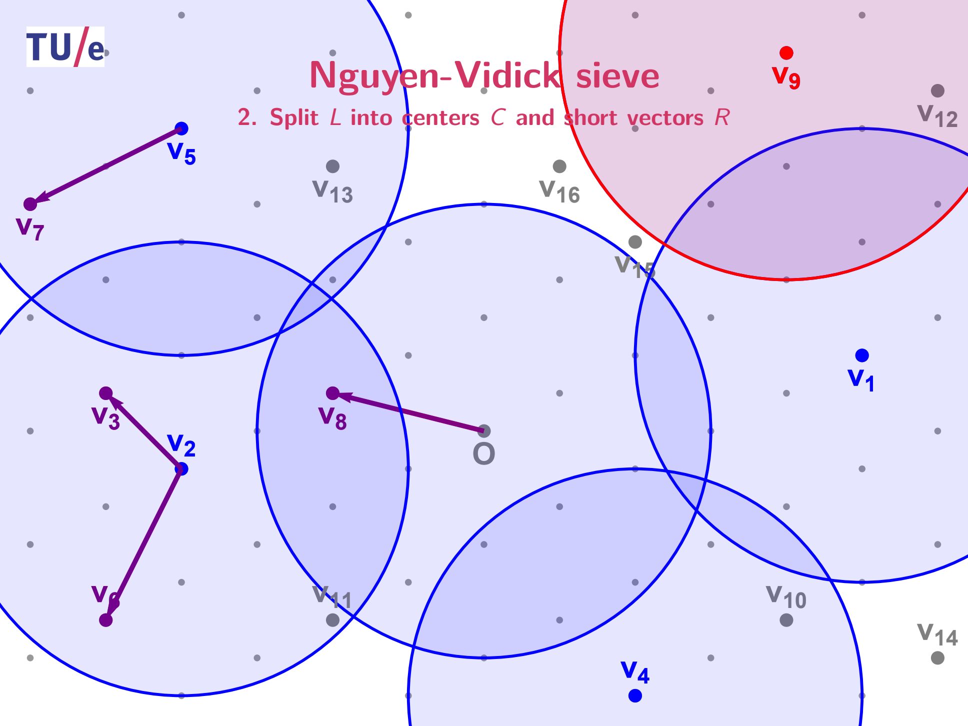
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



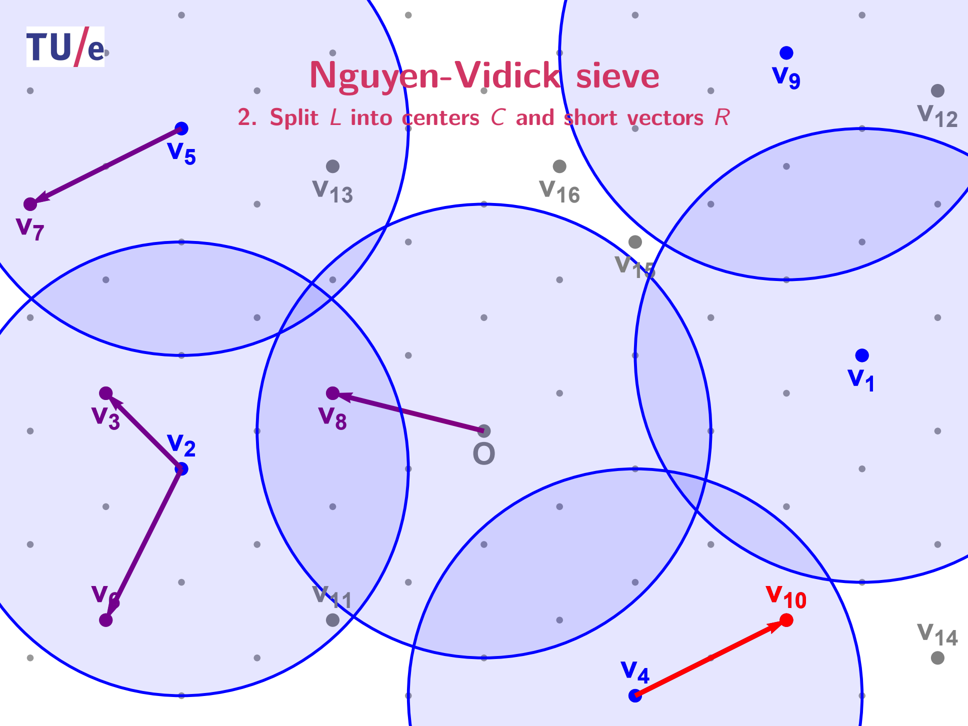
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



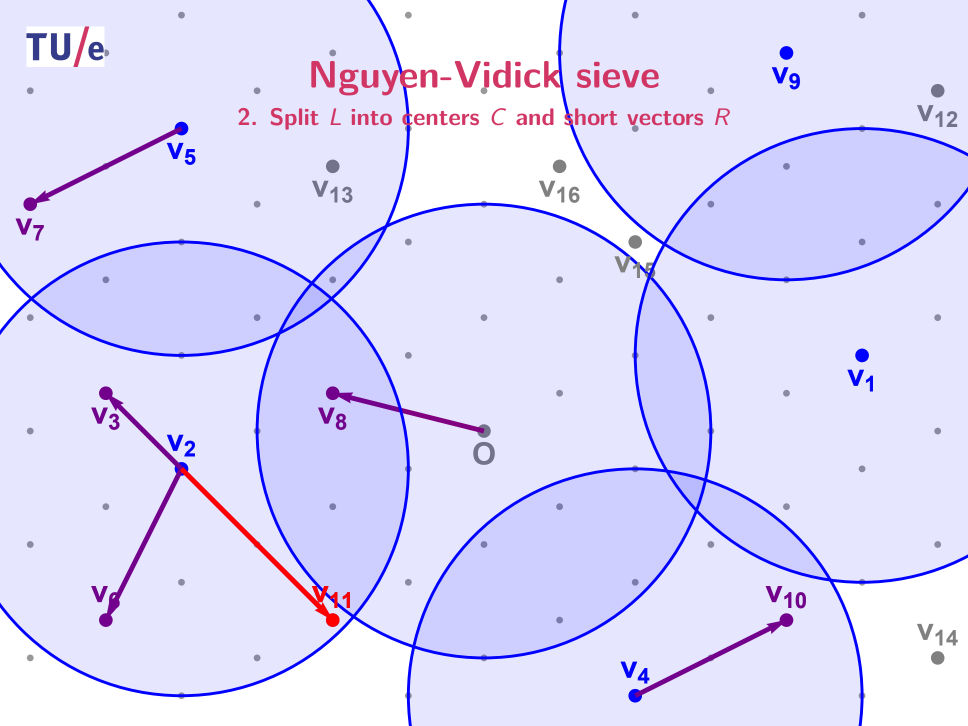
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



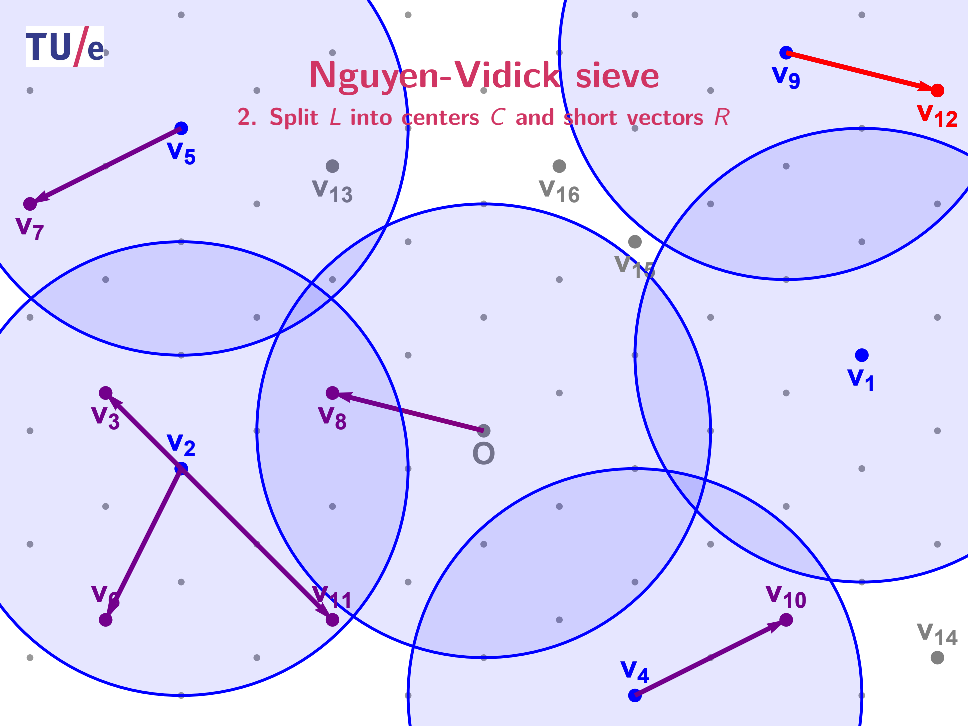
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



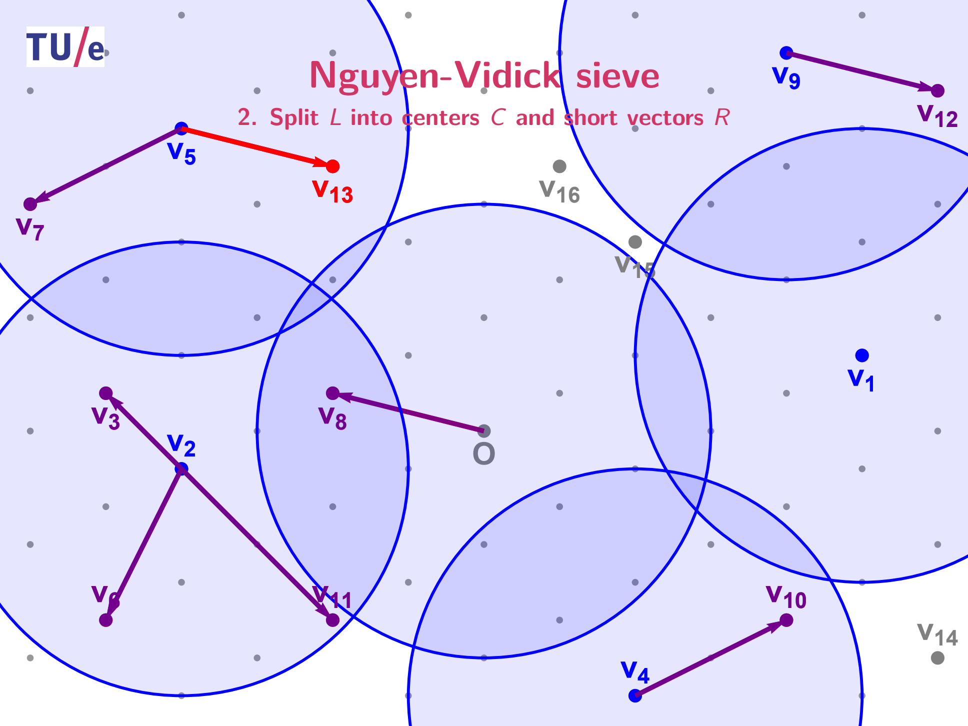
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



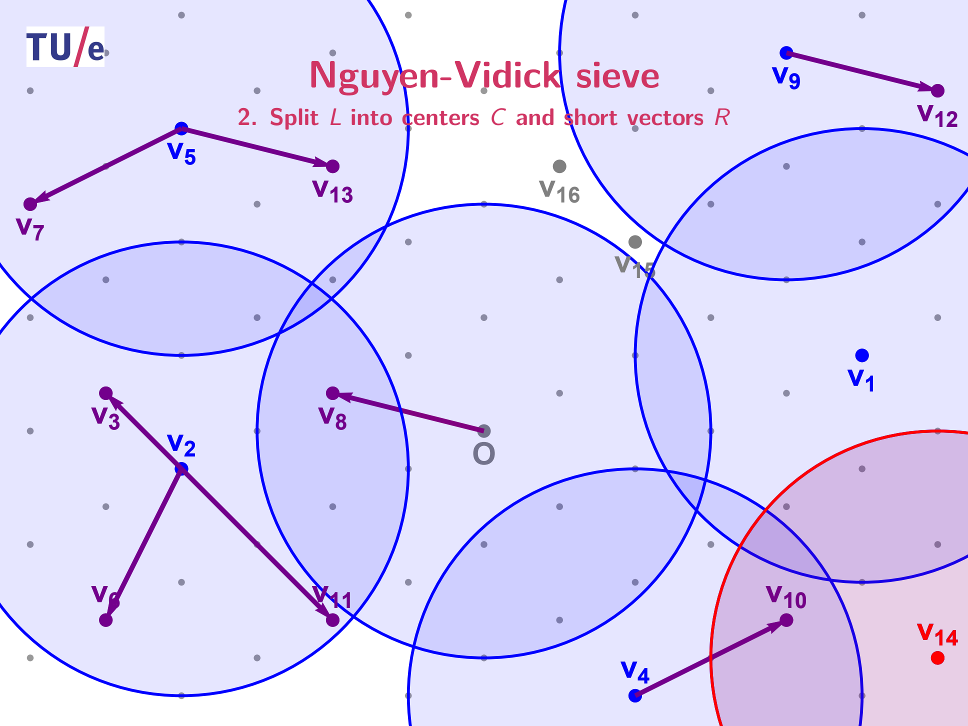
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



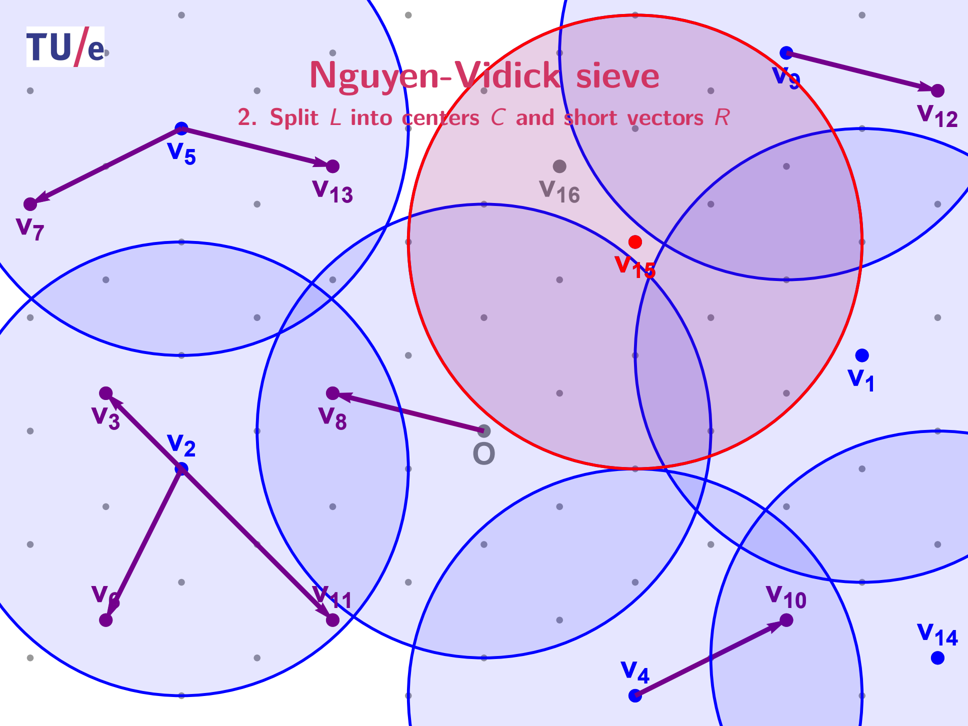
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



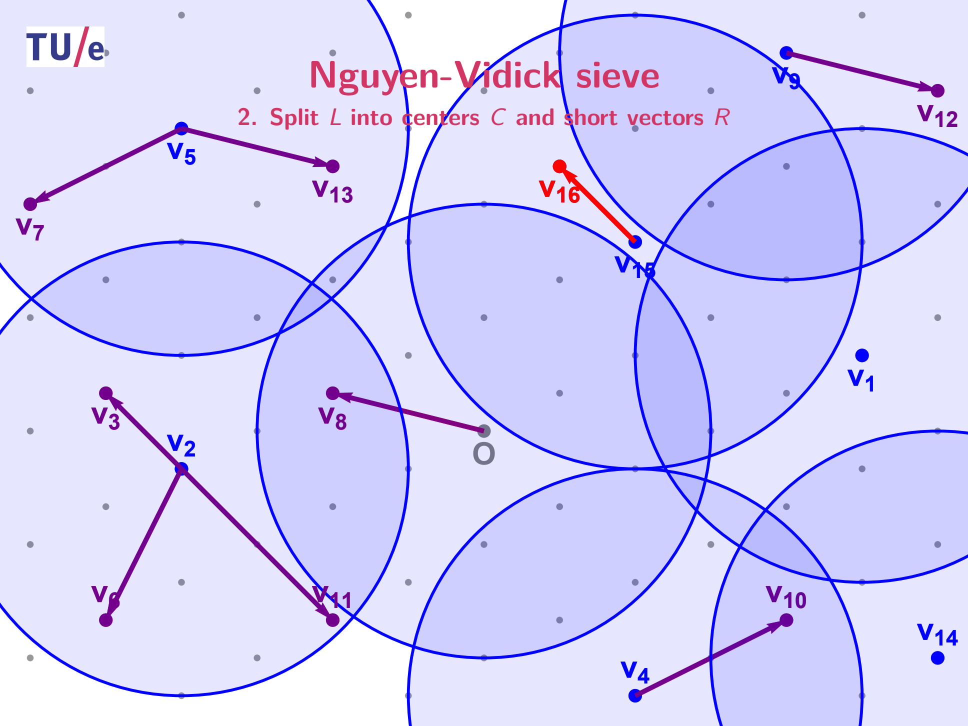
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



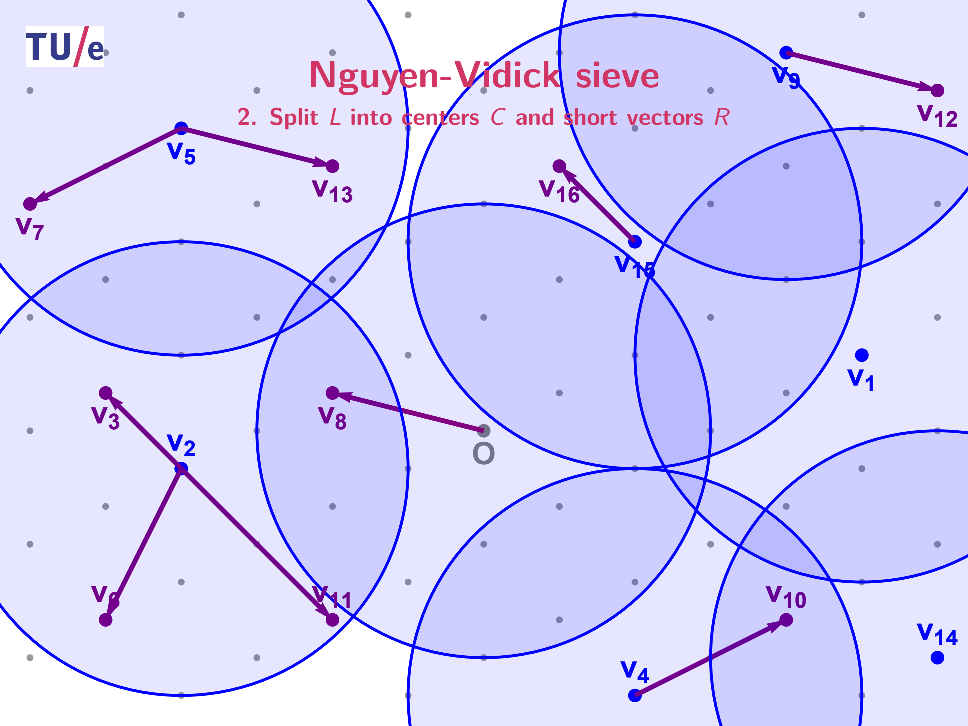
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



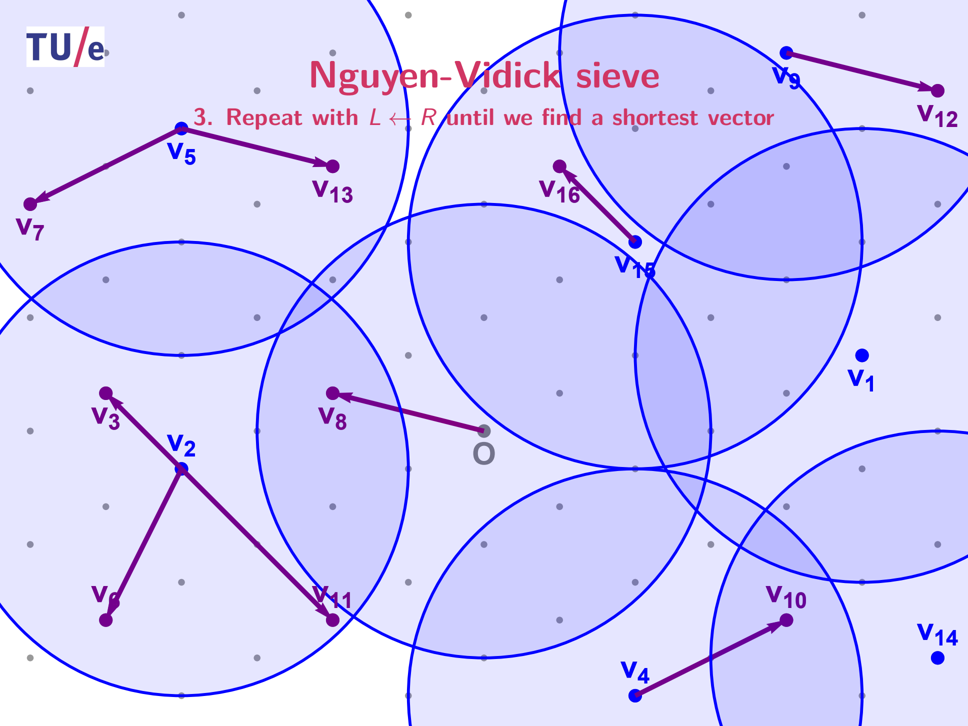
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



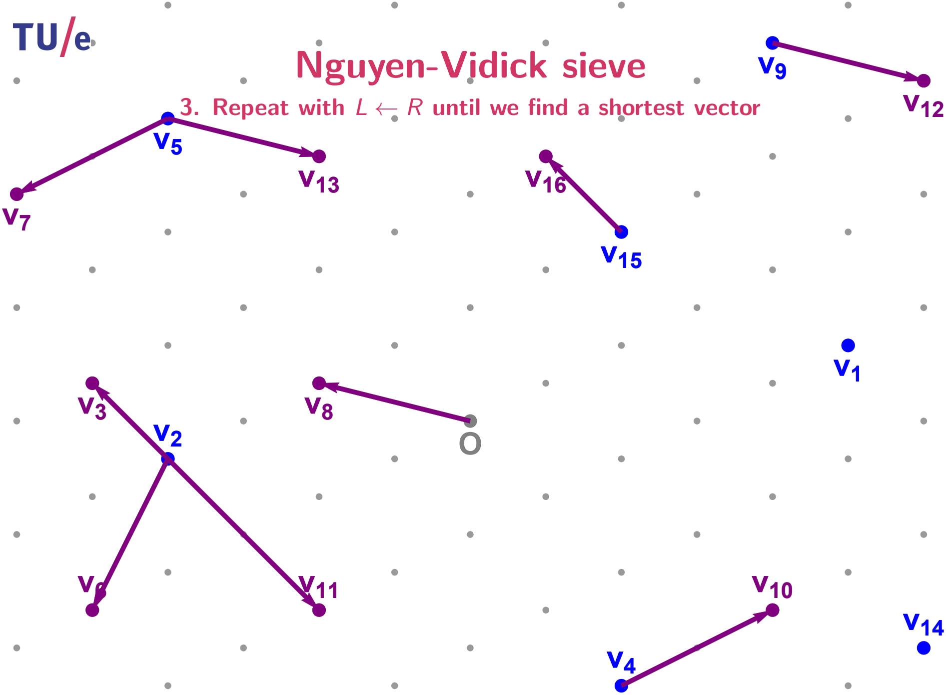
Nguyen-Vidick sieve

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



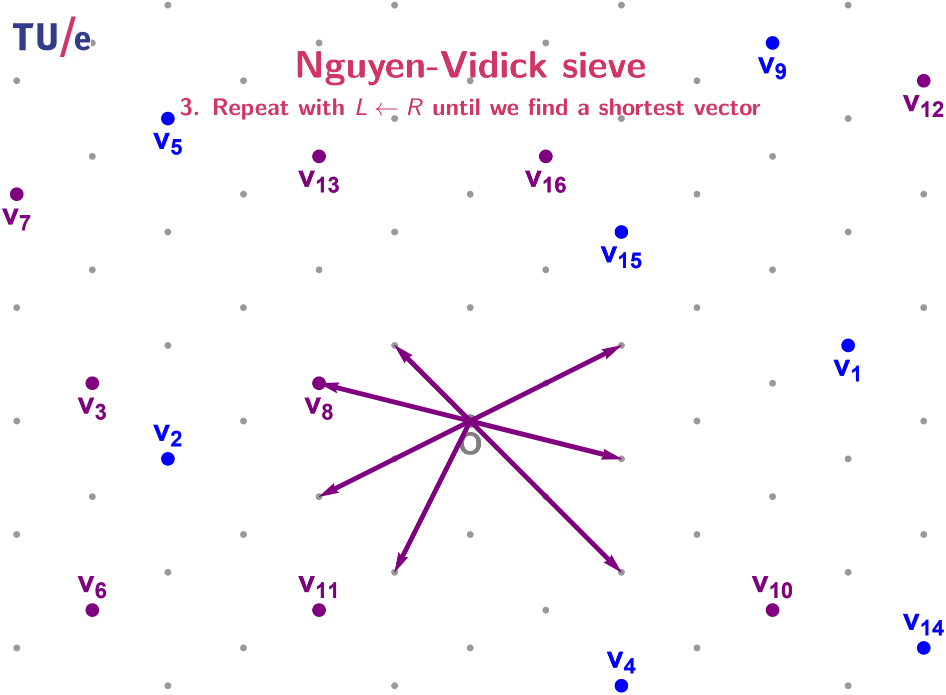
Nguyen-Vidick sieve

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



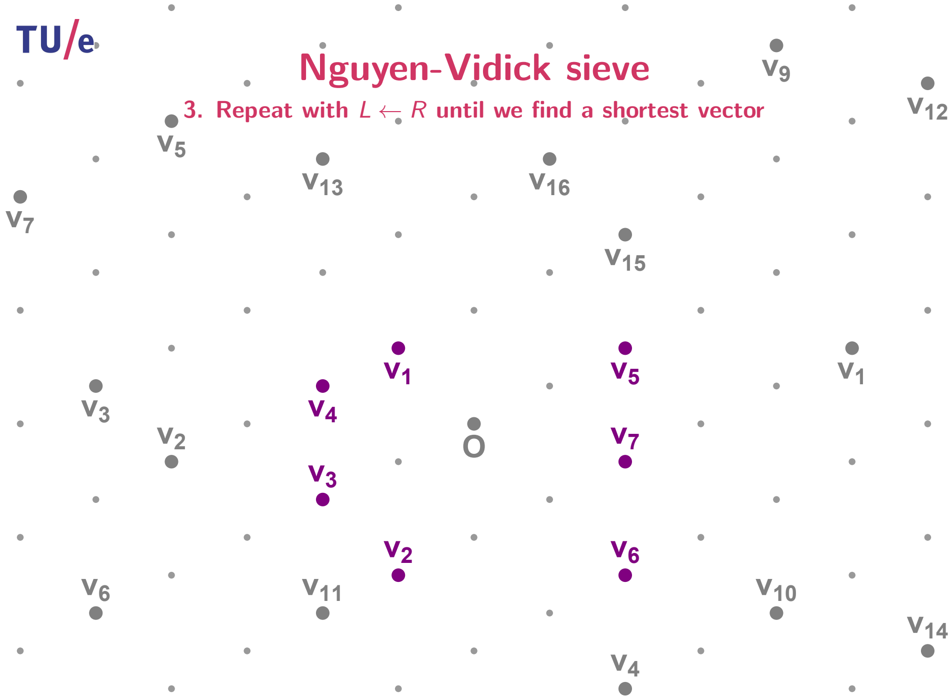
Nguyen-Vidick sieve

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)

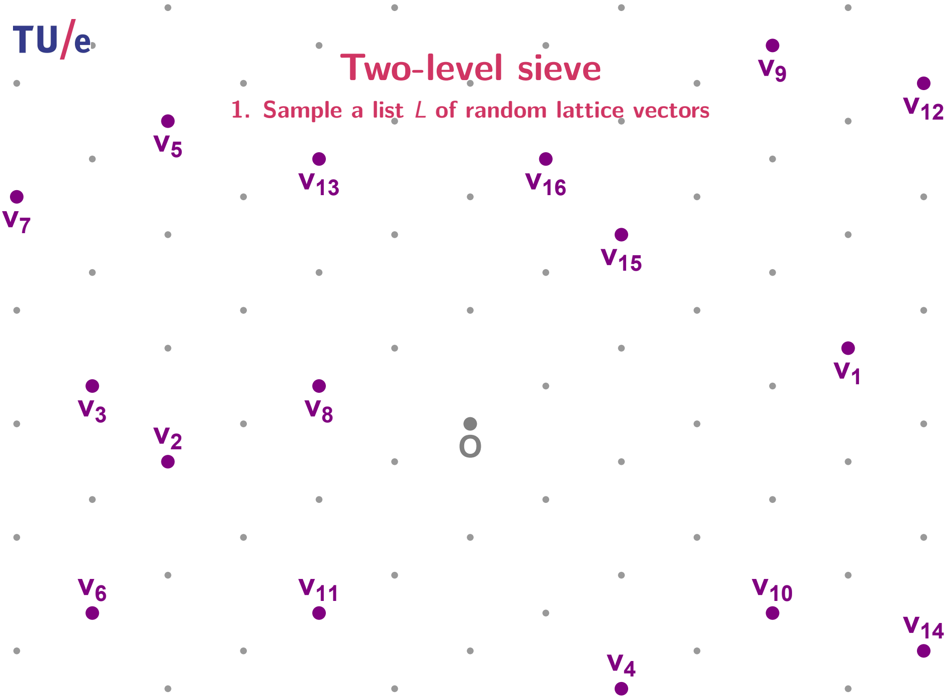
The Nguyen-Vidick sieve runs in time $(4/3)^n$ and space $\sqrt{4/3}^n$.

Two-level sieve

1. Sample a list L of random lattice vectors

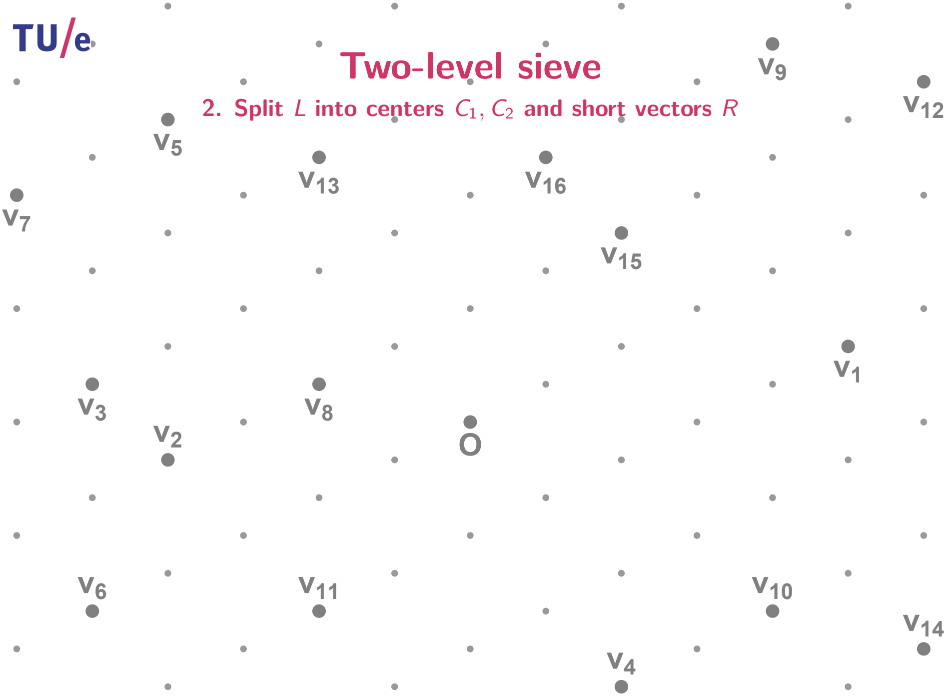
Two-level sieve

1. Sample a list L of random lattice vectors



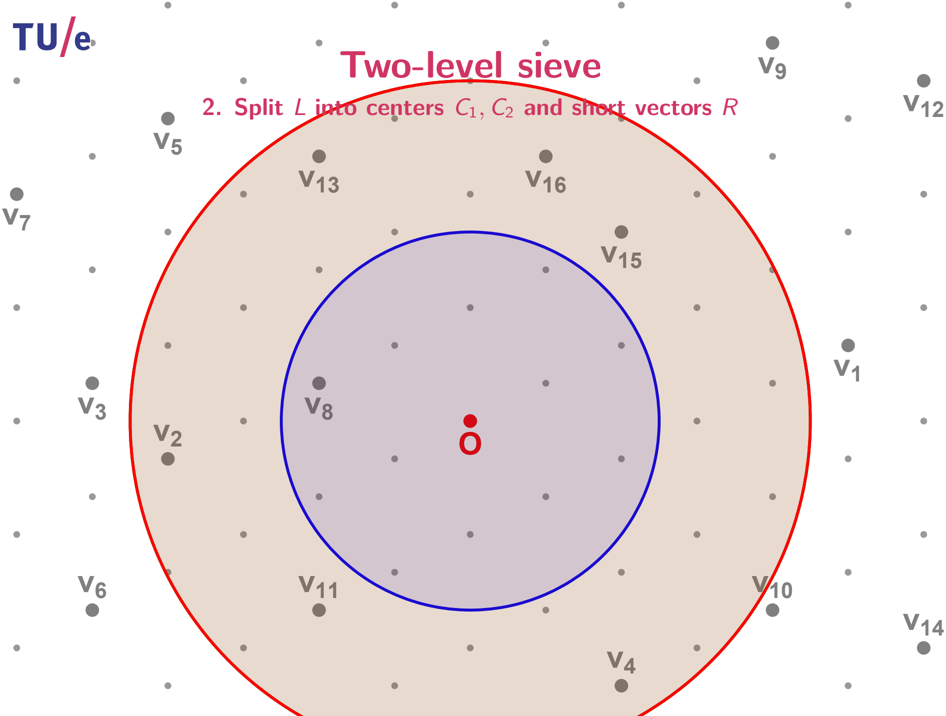
Two-level sieve

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



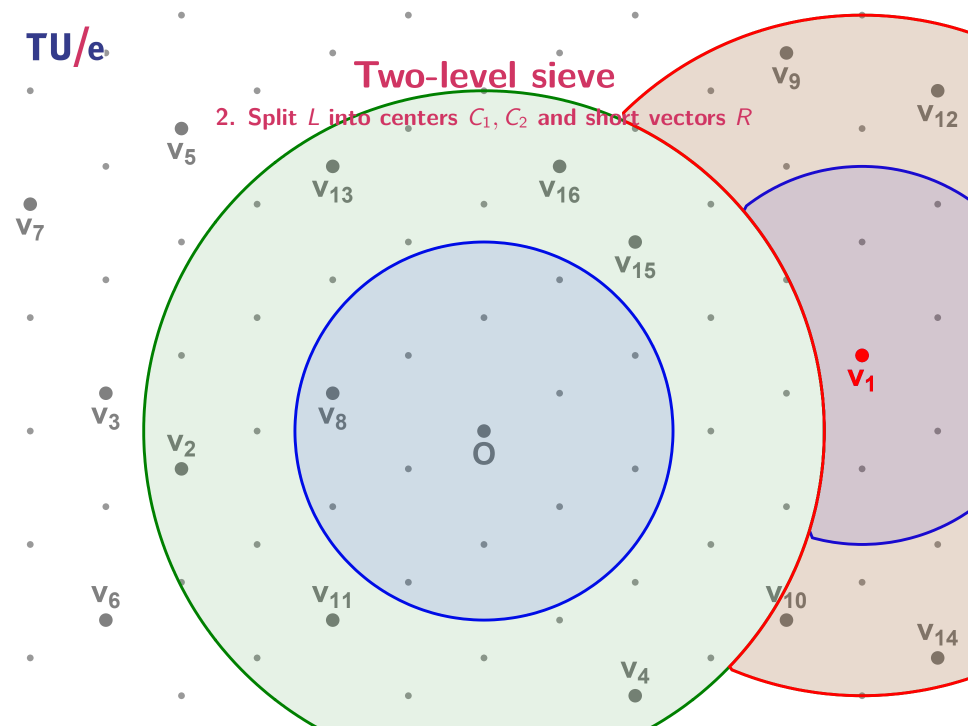
Two-level sieve

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



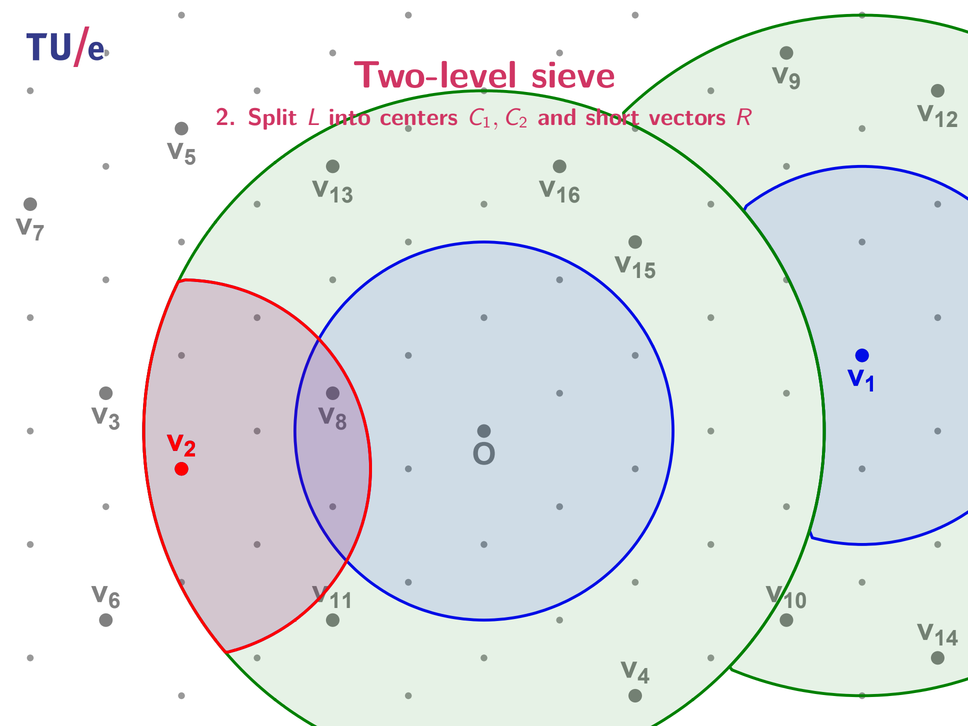
Two-level sieve

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



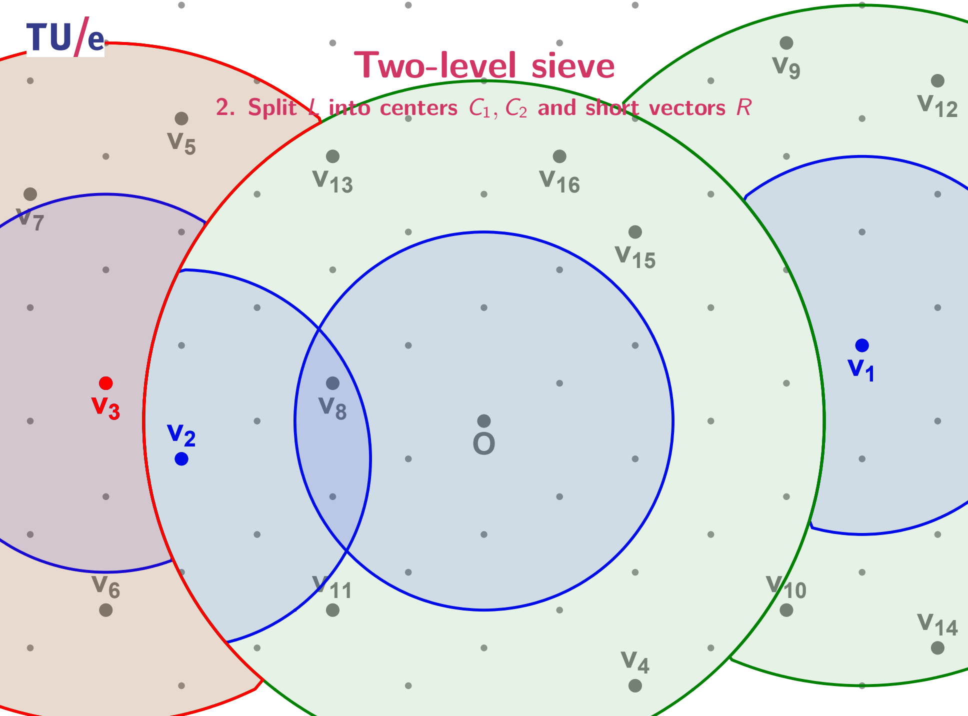
Two-level sieve

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



Two-level sieve

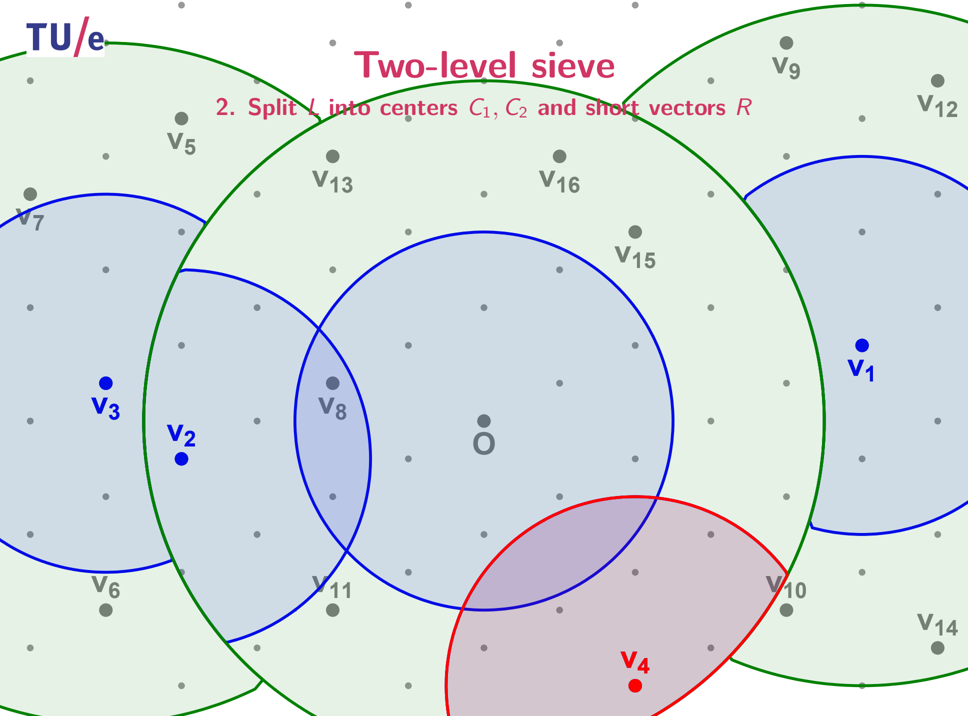
2. Split L into centers C_1, C_2 and short vectors R



TU/e

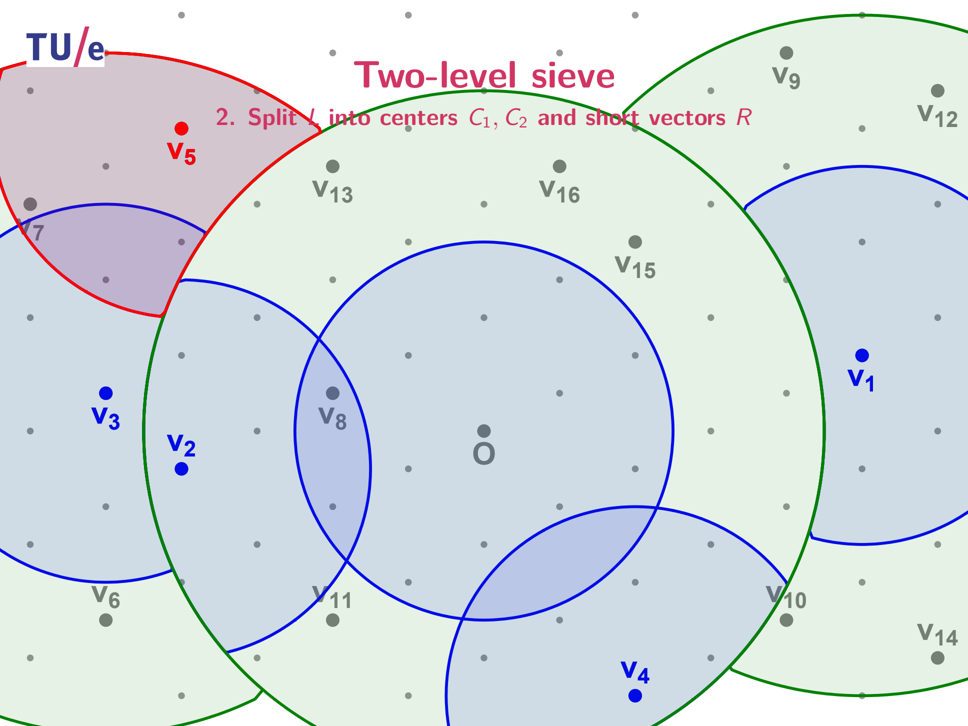
Two-level sieve

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



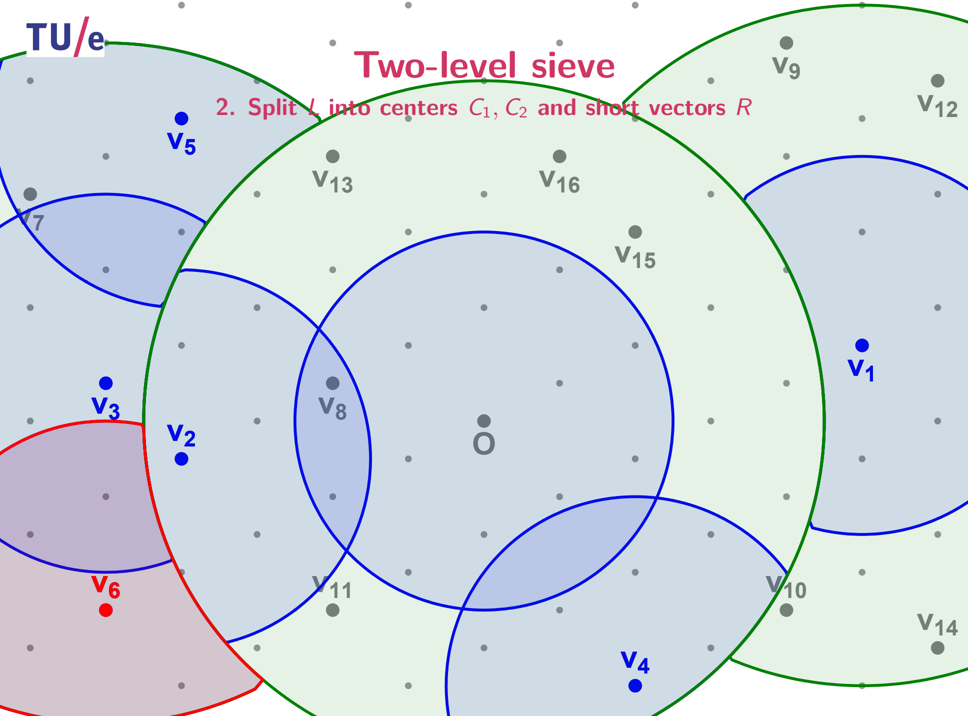
Two-level sieve

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



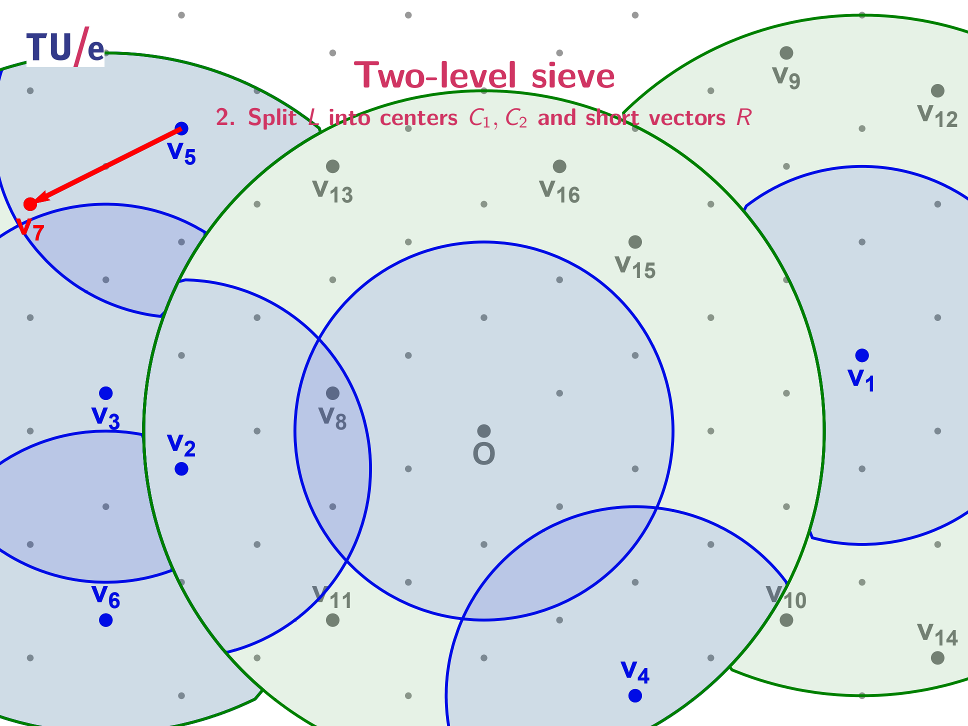
Two-level sieve

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



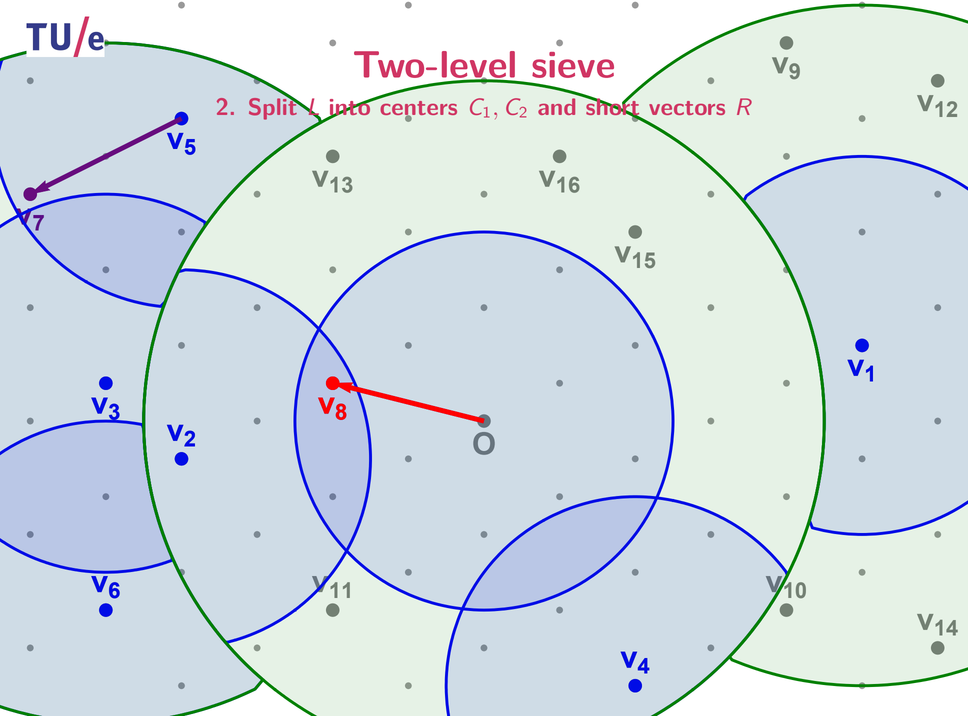
Two-level sieve

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



Two-level sieve

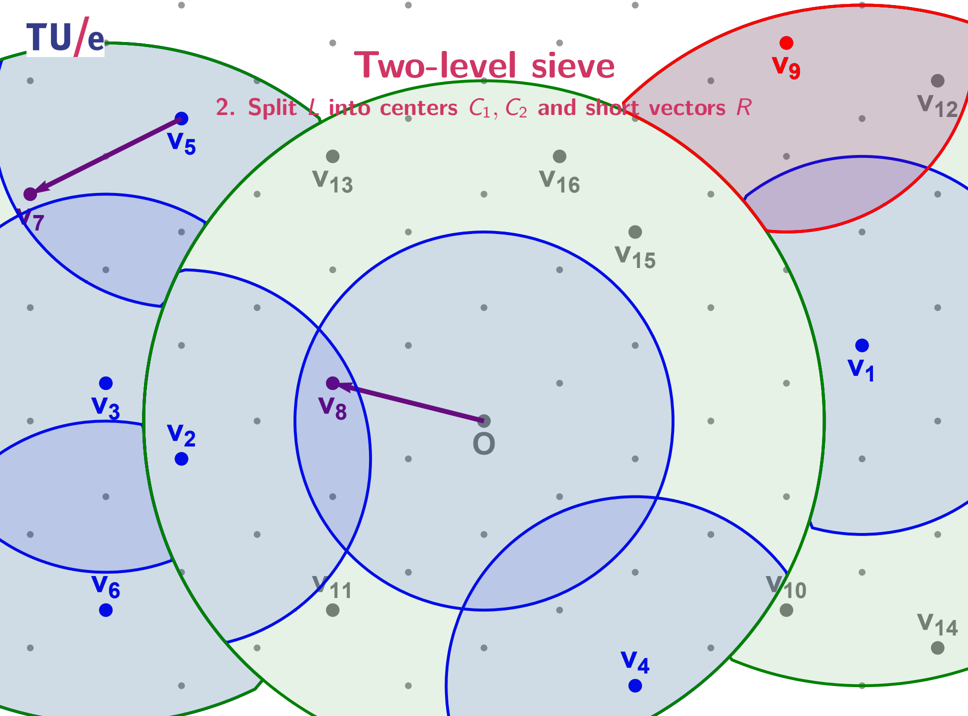
2. Split L into centers C_1, C_2 and short vectors R



TU/e

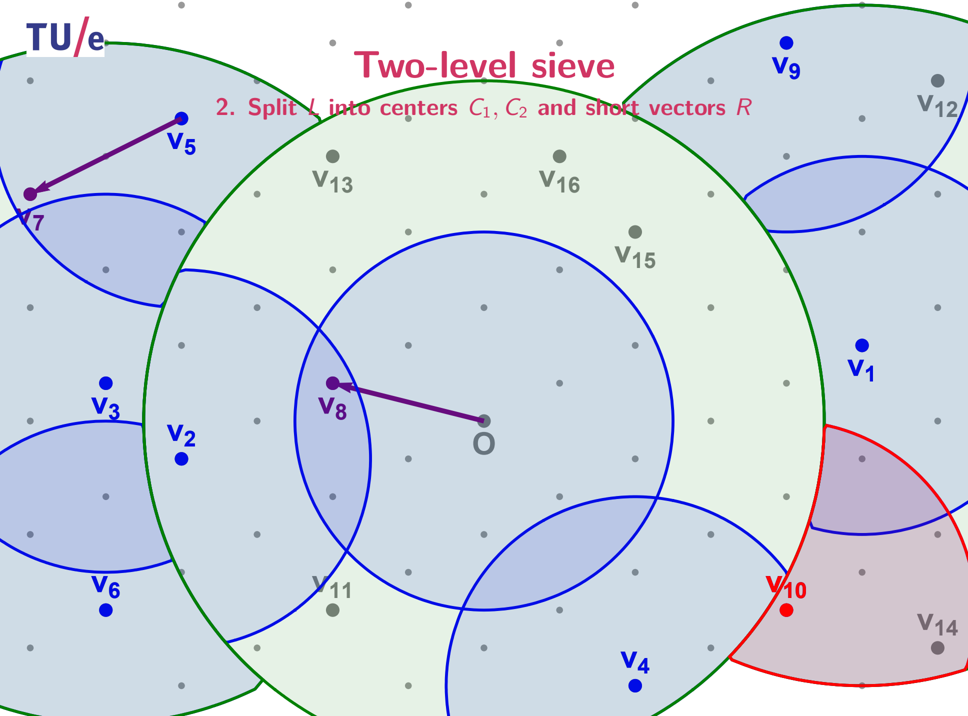
Two-level sieve

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



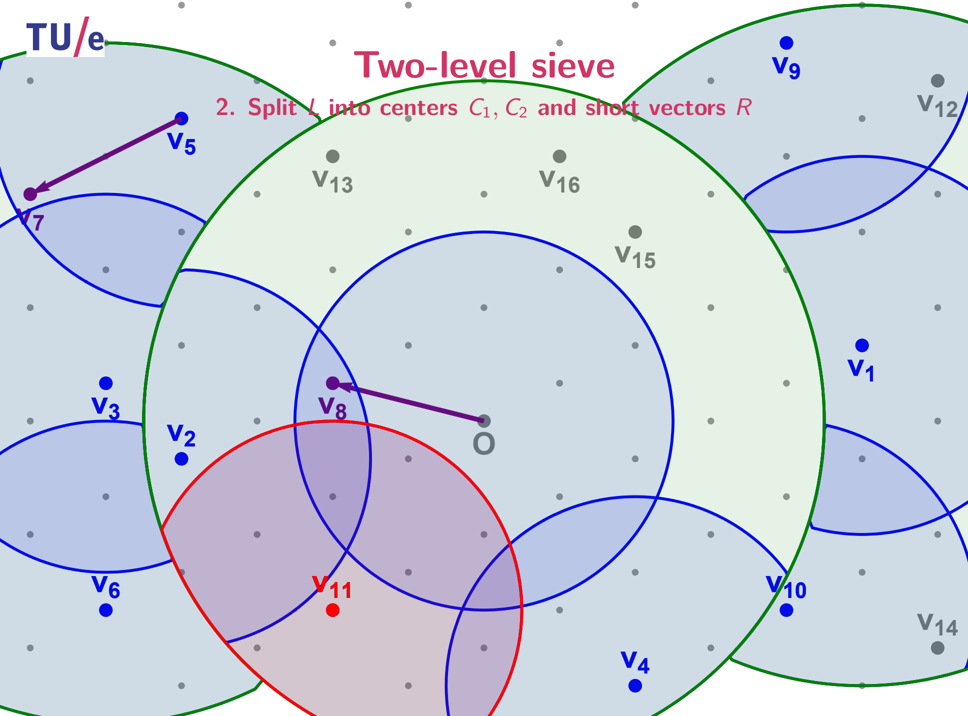
Two-level sieve

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



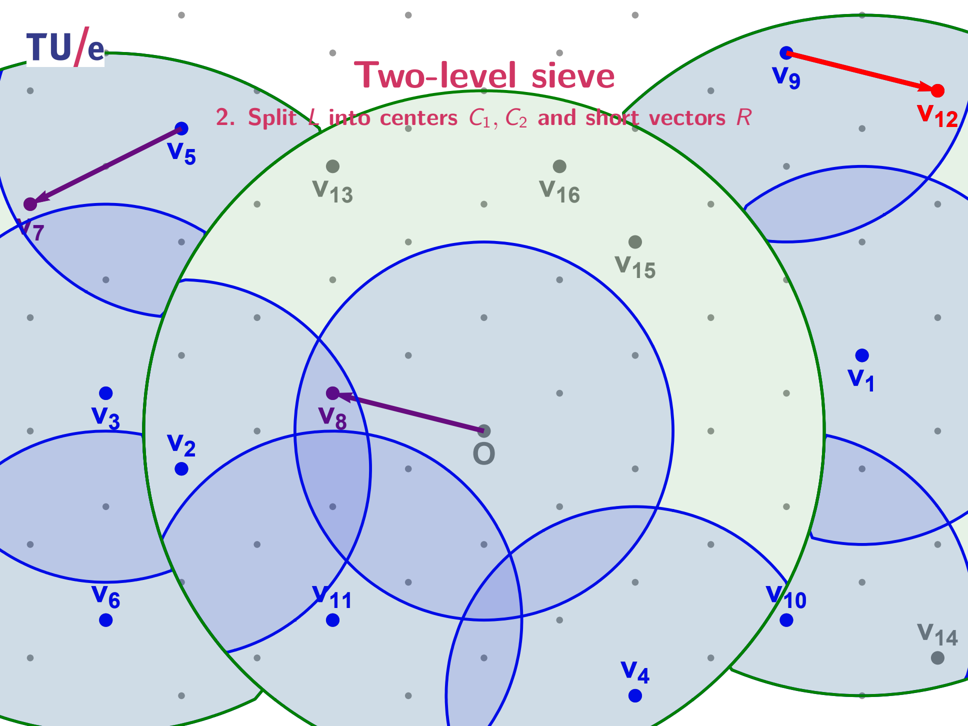
Two-level sieve

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



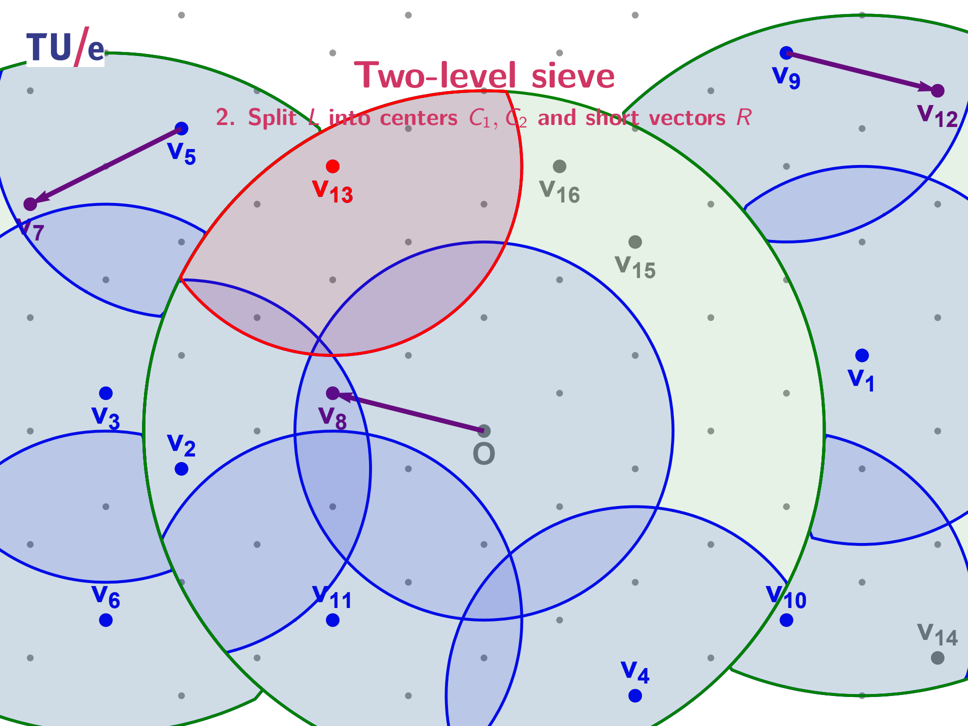
Two-level sieve

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



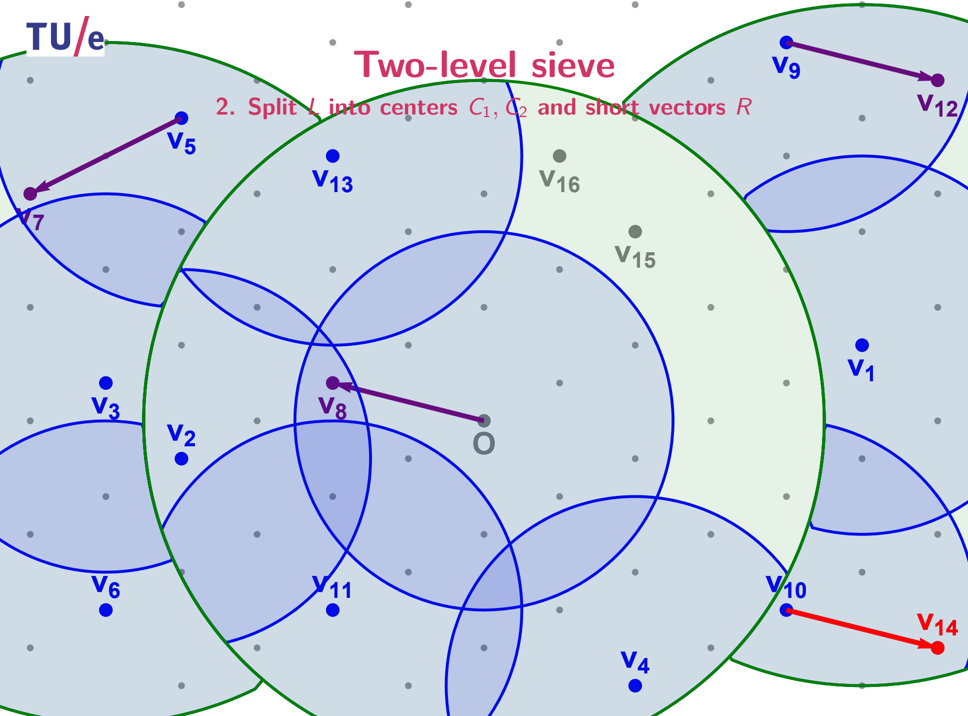
Two-level sieve

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



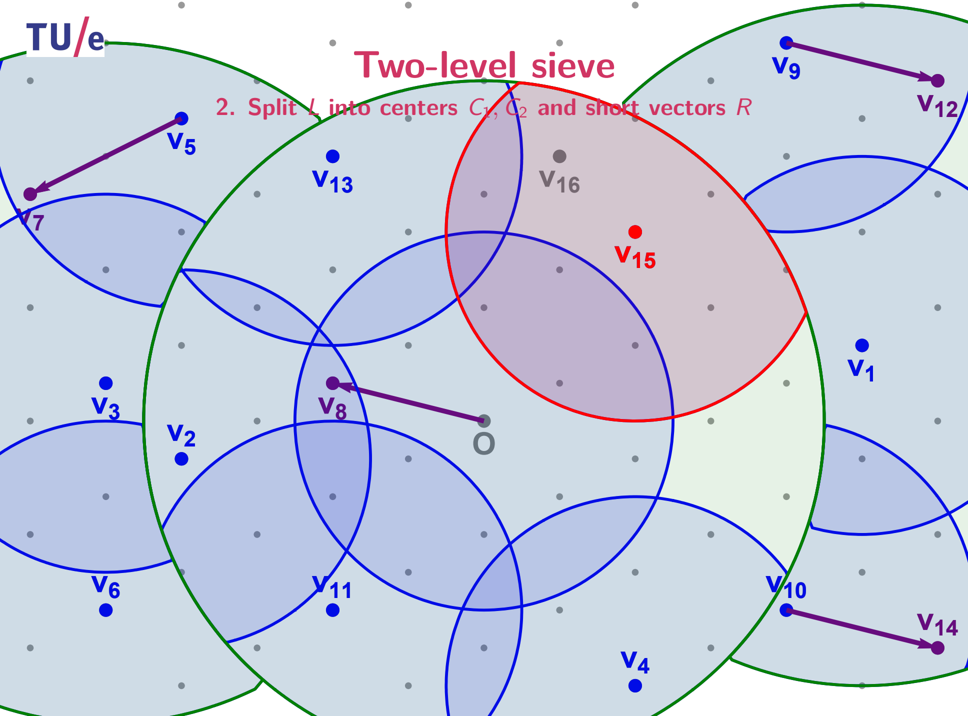
Two-level sieve

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



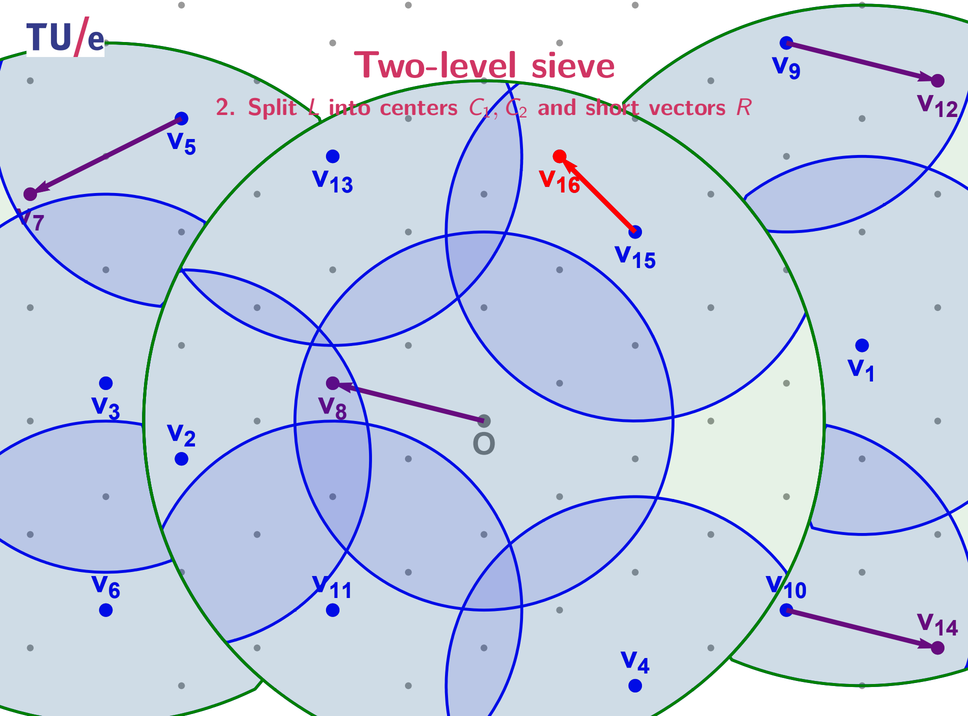
Two-level sieve

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



Two-level sieve

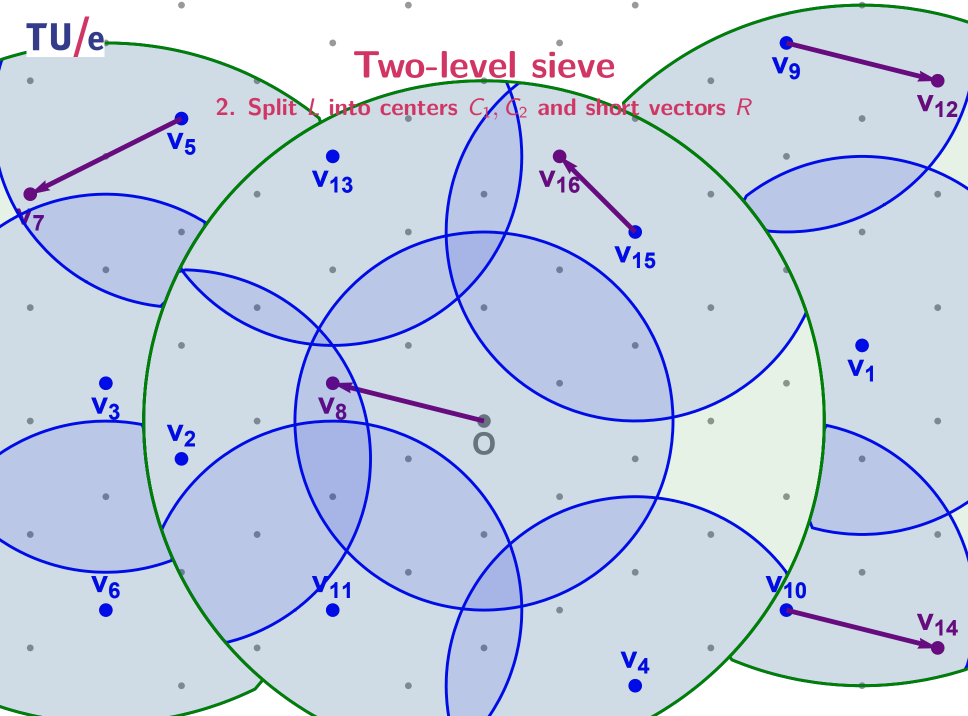
2. Split L into centers C_1, C_2 and short vectors R



TU/e

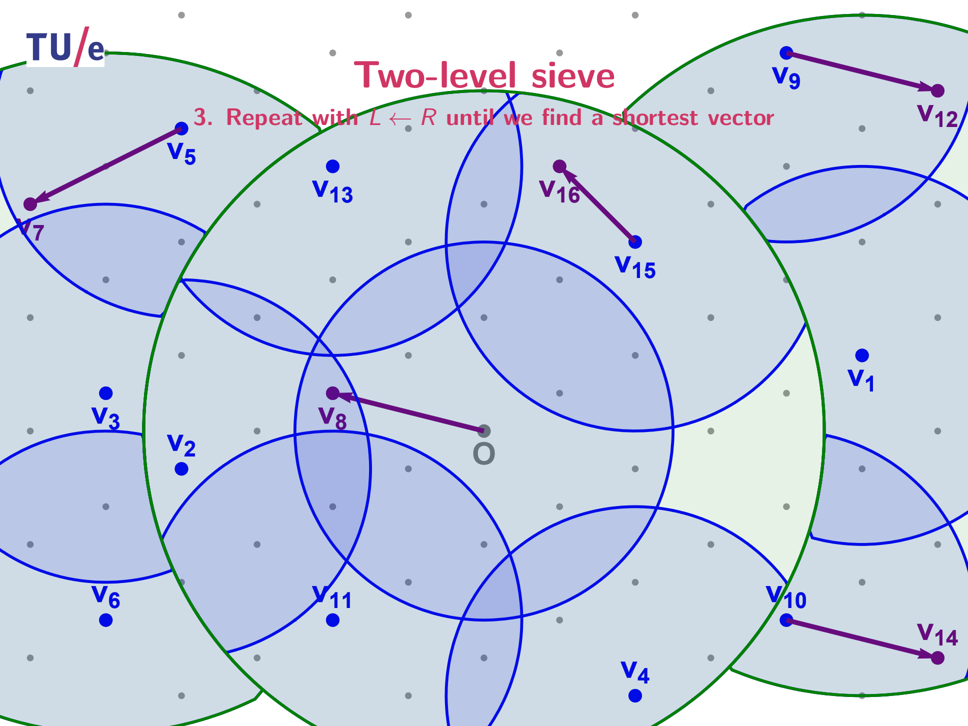
Two-level sieve

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



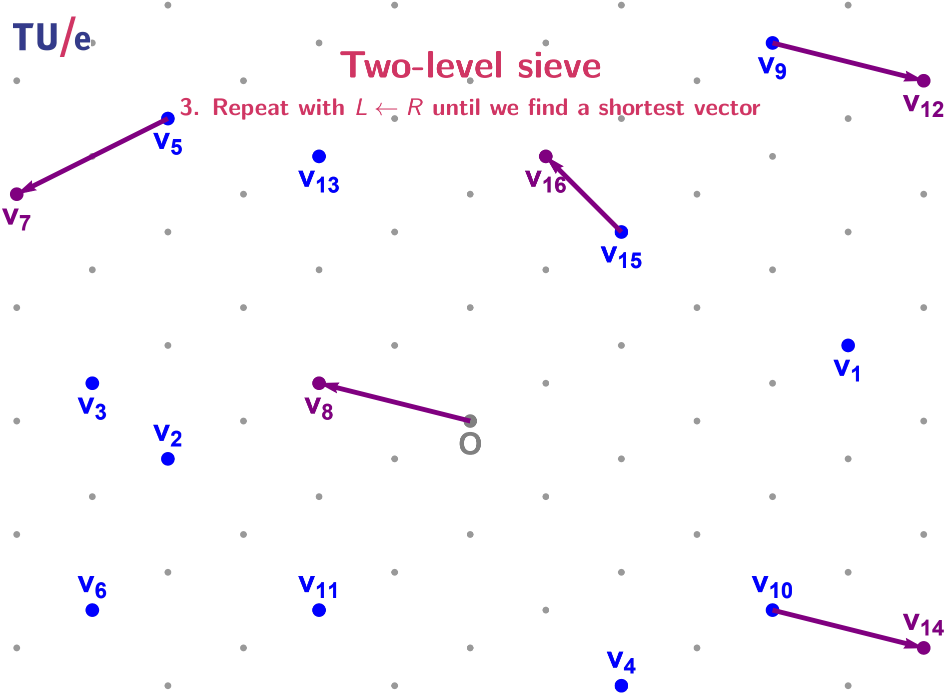
Two-level sieve

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



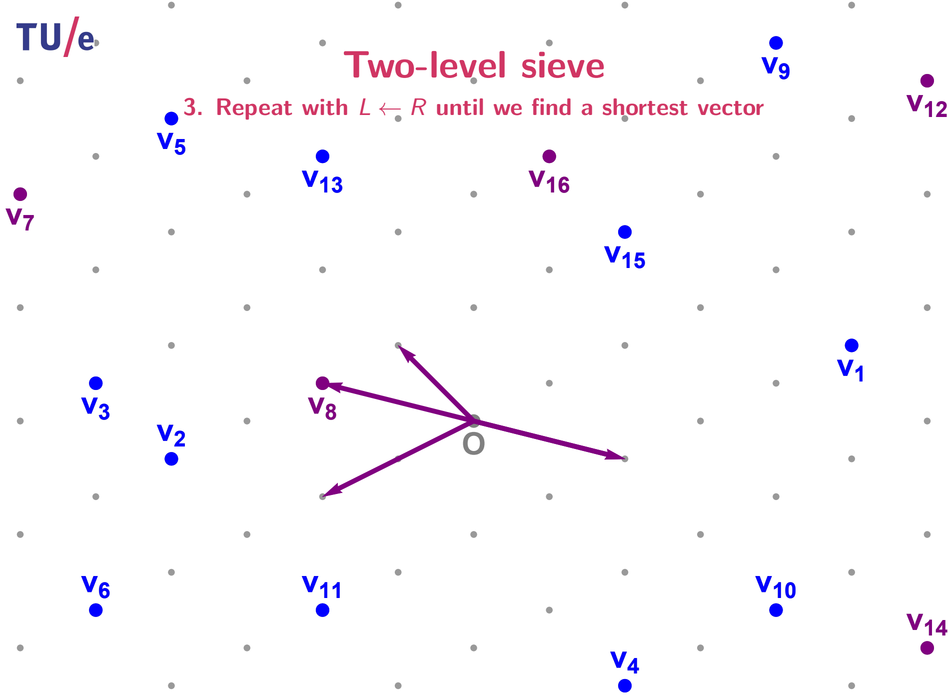
Two-level sieve

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



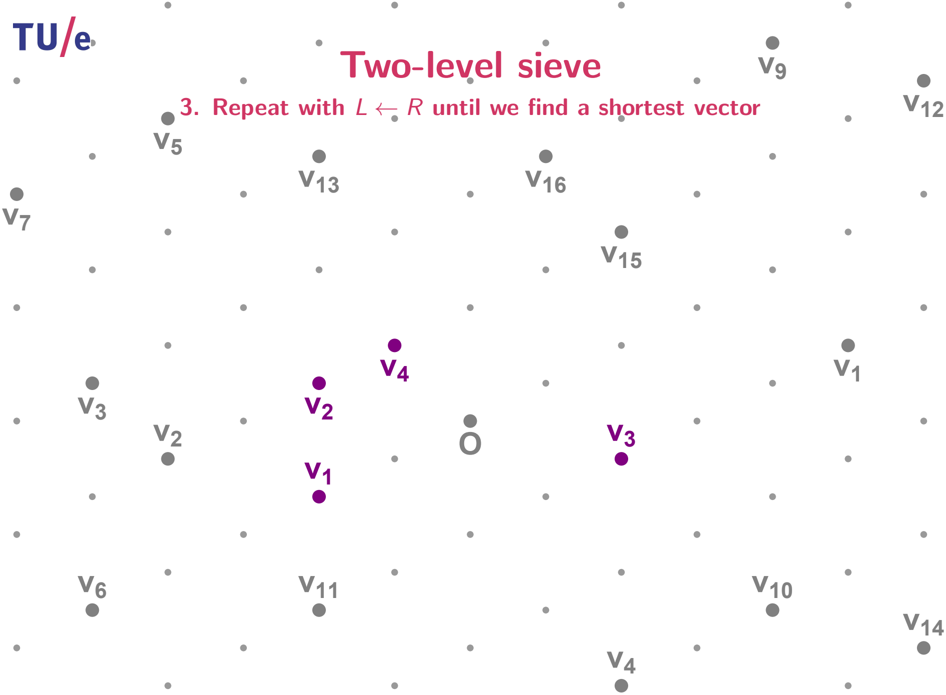
Two-level sieve

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



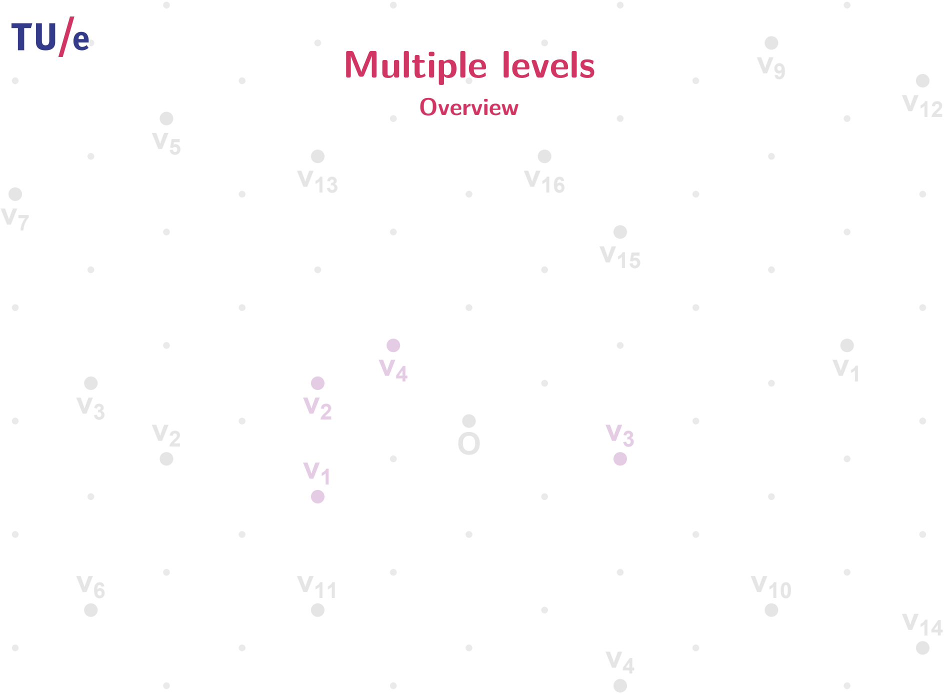
Two-level sieve

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

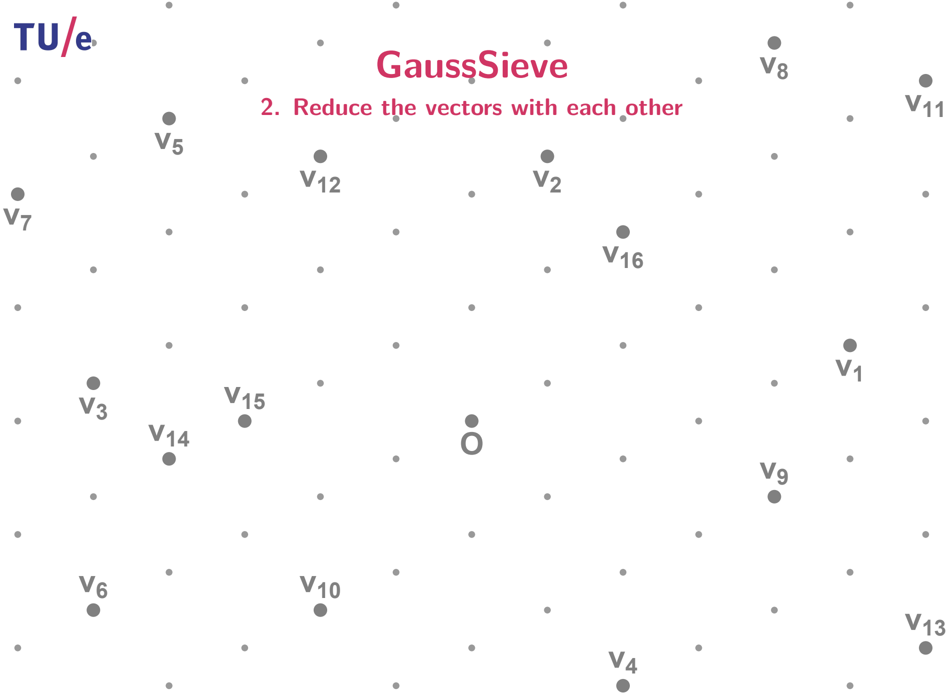
1. Generate random lattice vectors

GaussSieve

1. Generate random lattice vectors

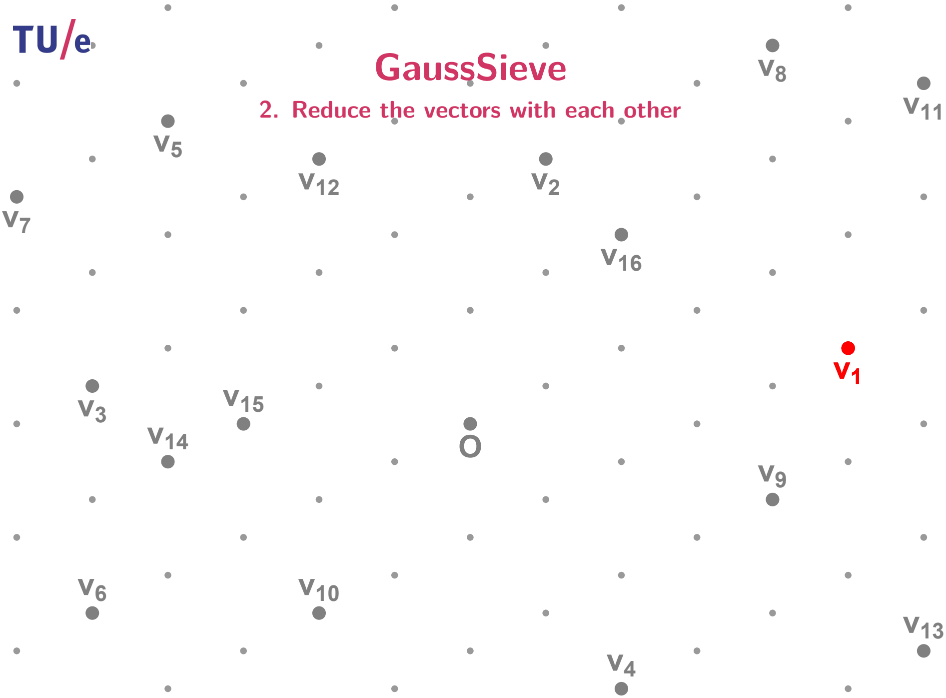
GaussSieve

2. Reduce the vectors with each other



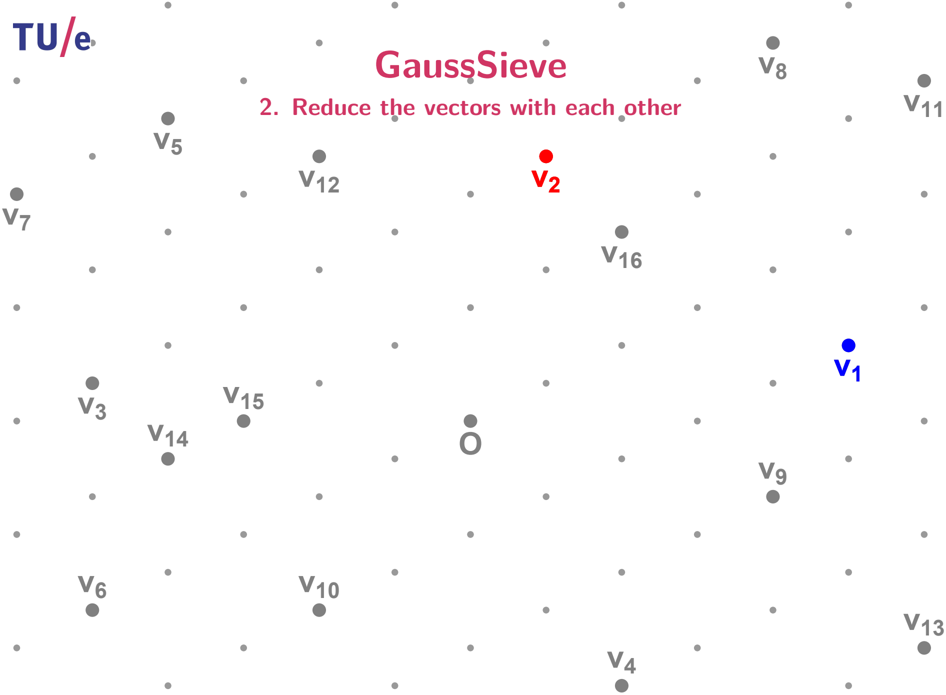
GaussSieve

2. Reduce the vectors with each other



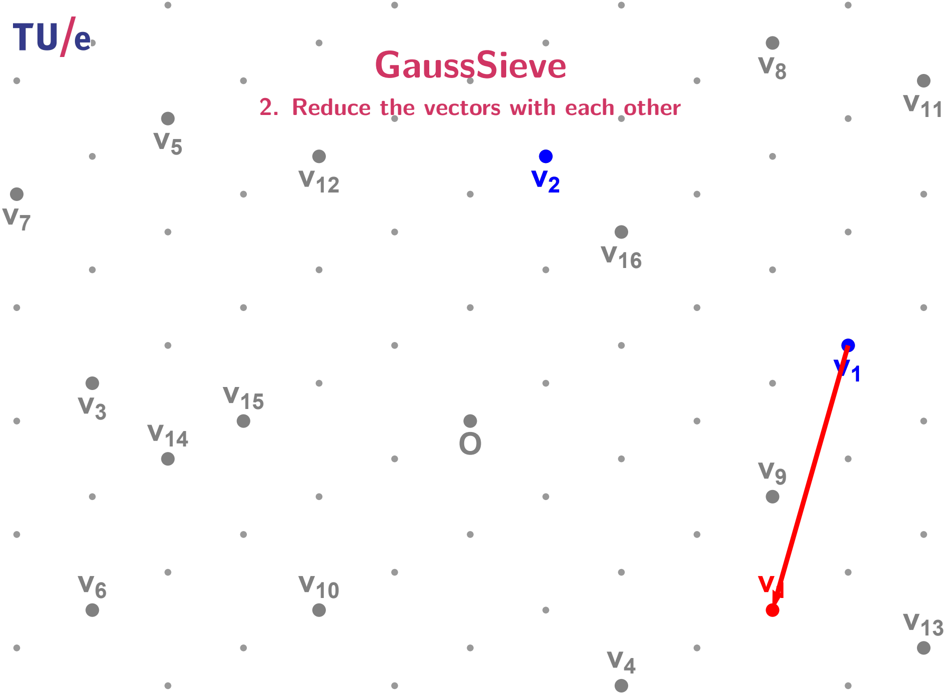
GaussSieve

2. Reduce the vectors with each other



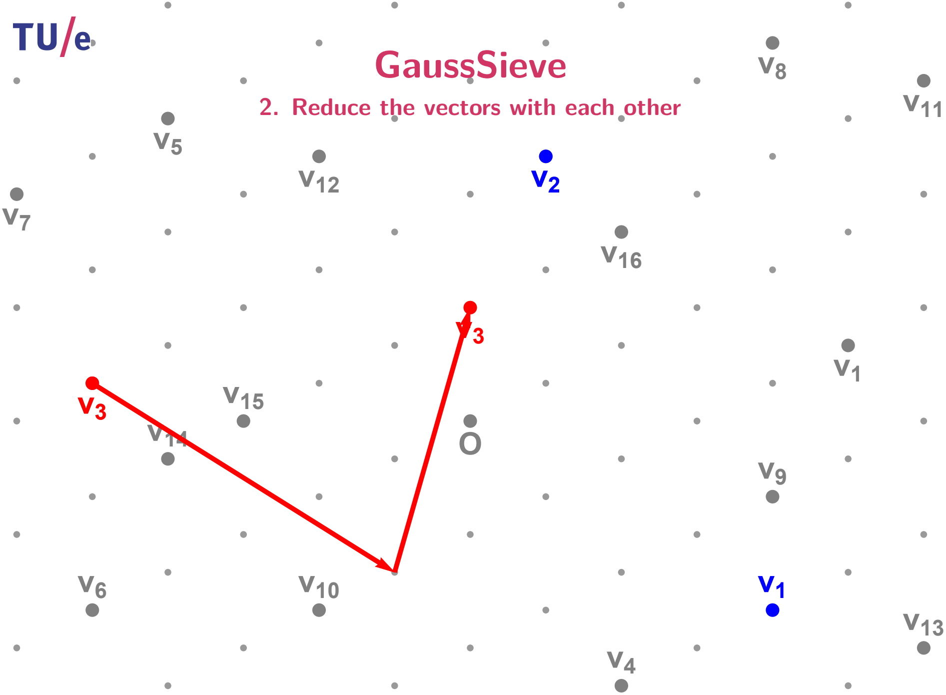
GaussSieve

2. Reduce the vectors with each other



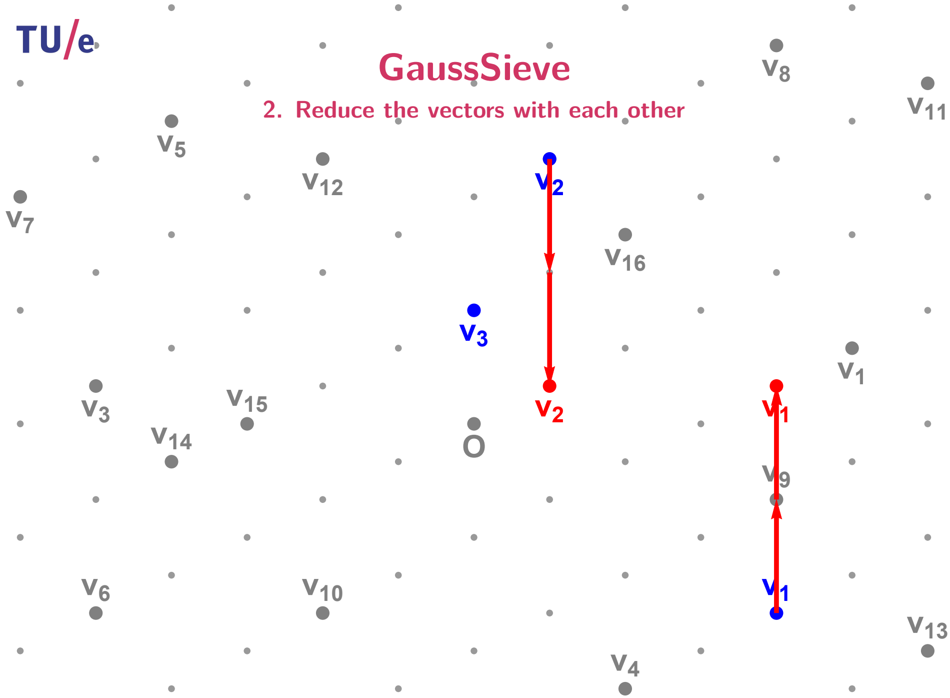
GaussSieve

2. Reduce the vectors with each other



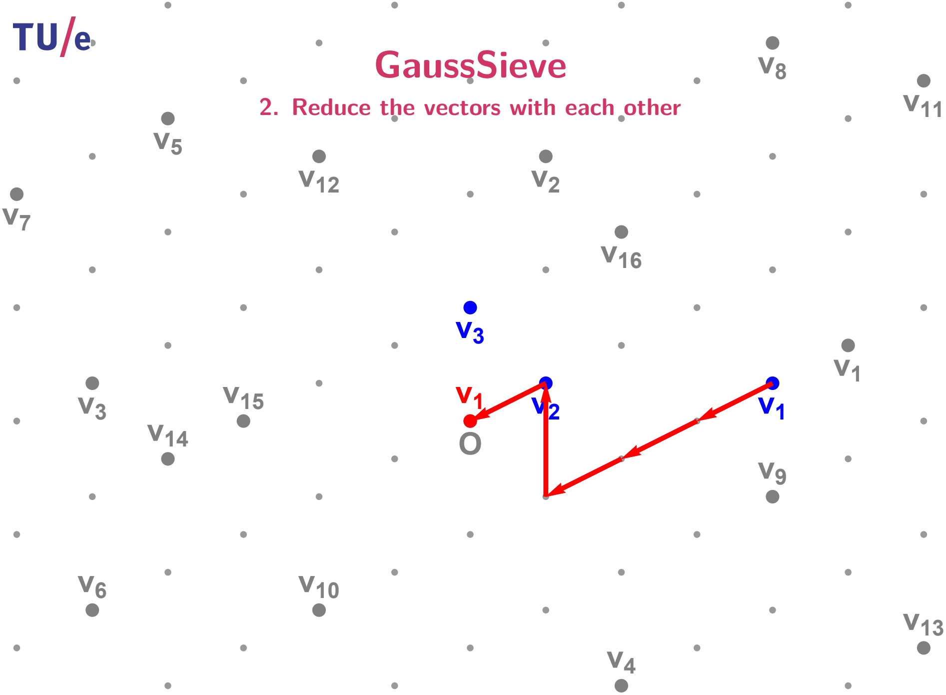
GaussSieve

2. Reduce the vectors with each other



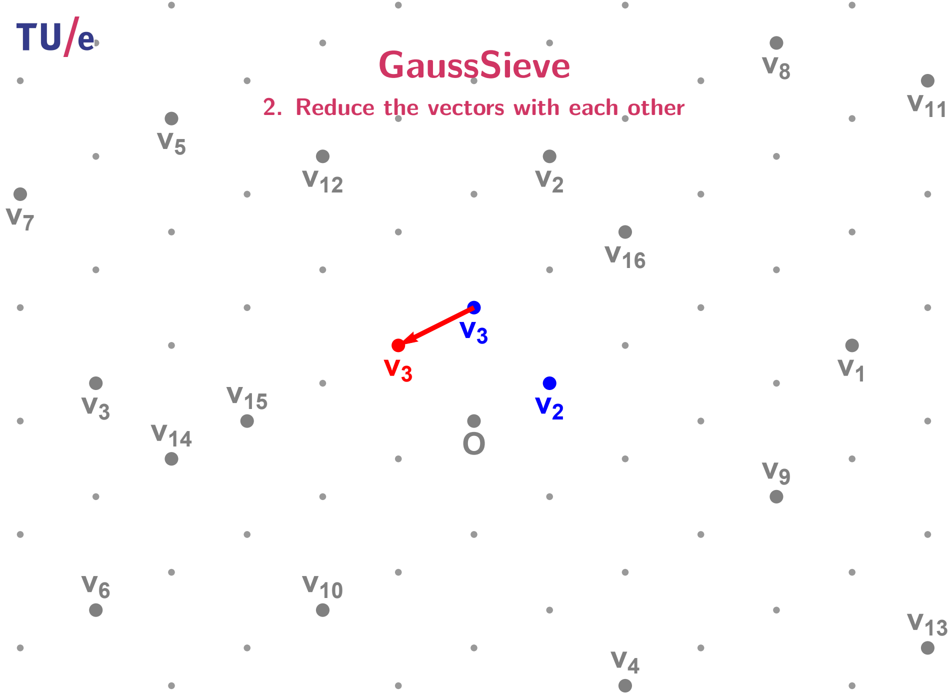
GaussSieve

2. Reduce the vectors with each other



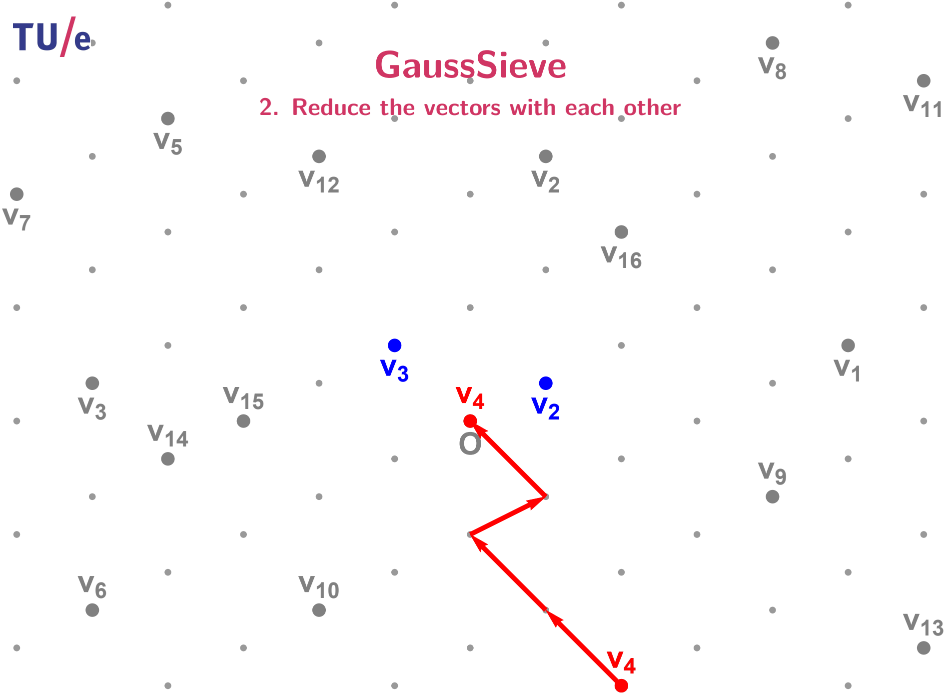
GaussSieve

2. Reduce the vectors with each other



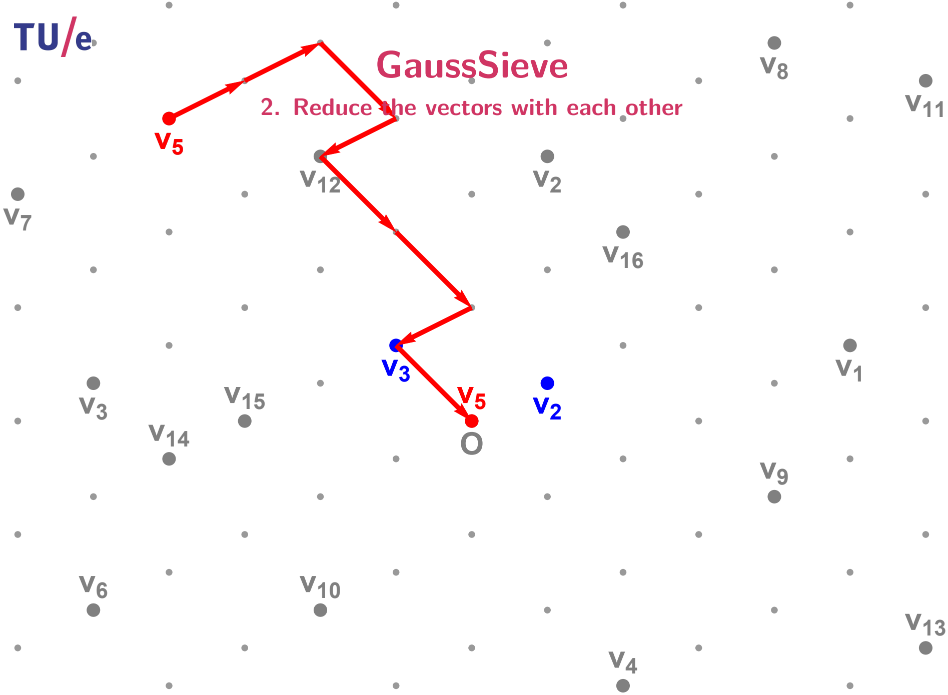
GaussSieve

2. Reduce the vectors with each other



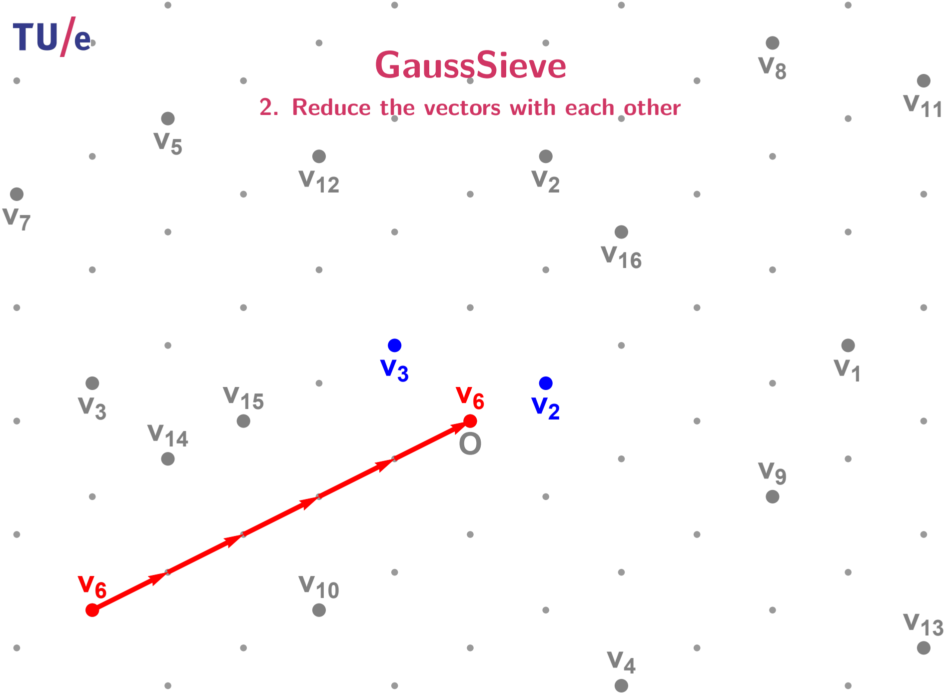
GaussSieve

2. Reduce the vectors with each other



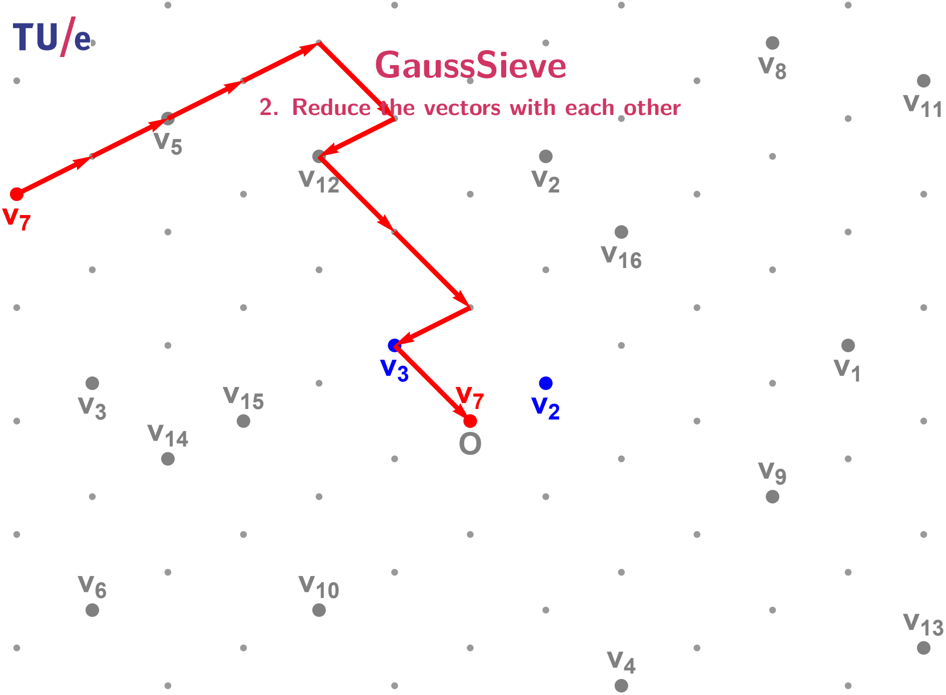
GaussSieve

2. Reduce the vectors with each other



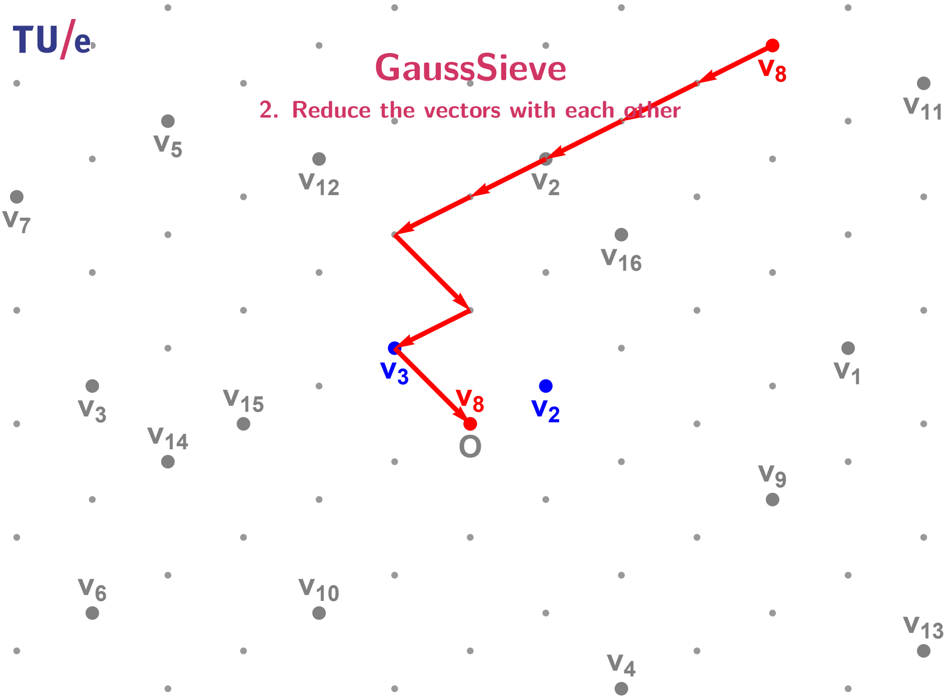
GaussSieve

2. Reduce the vectors with each other



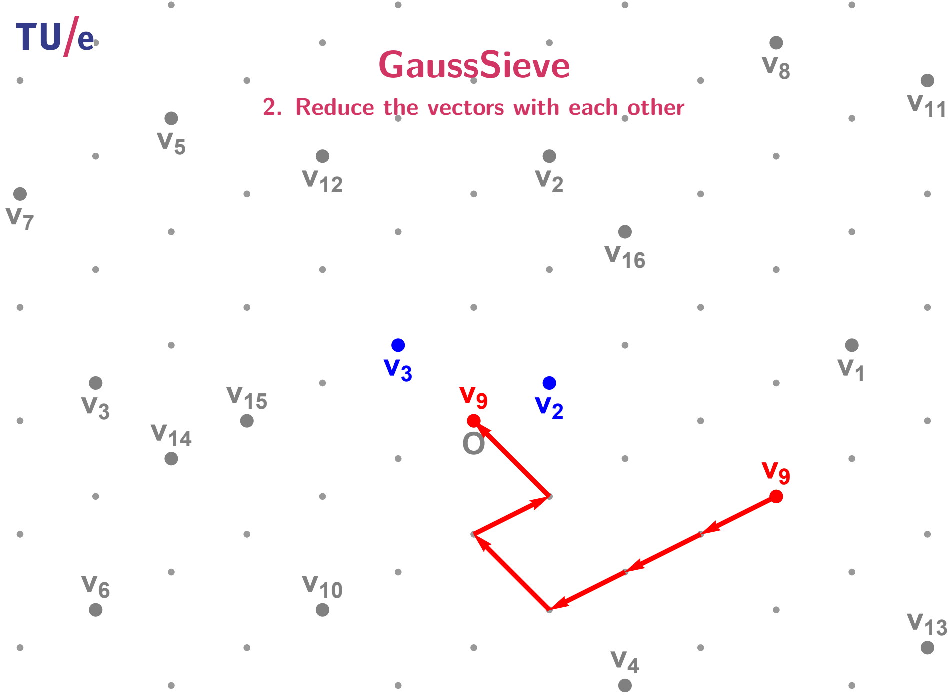
GaussSieve

2. Reduce the vectors with each other



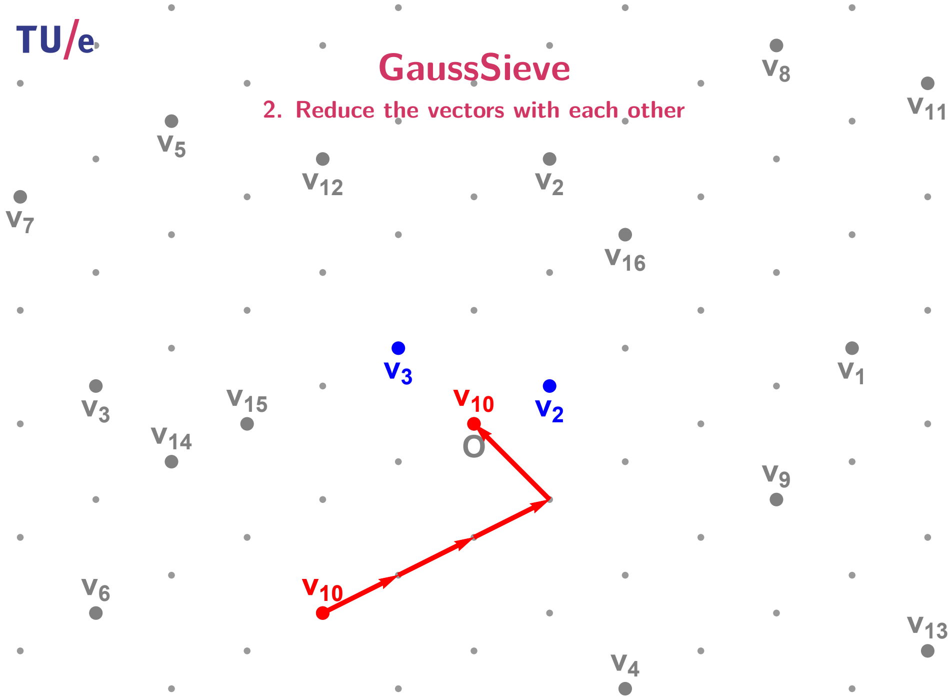
GaussSieve

2. Reduce the vectors with each other



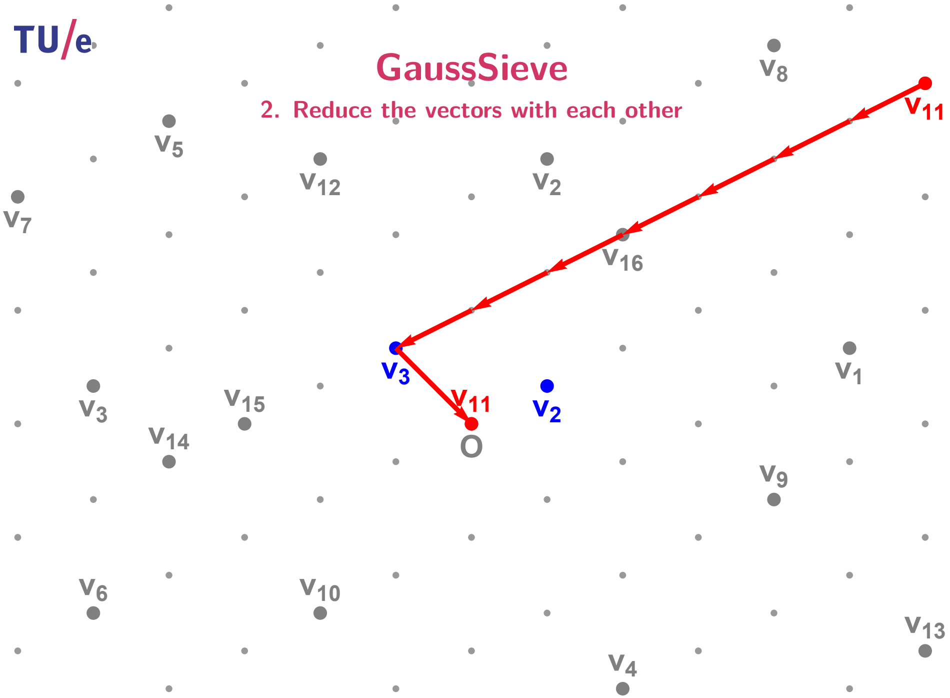
GaussSieve

2. Reduce the vectors with each other



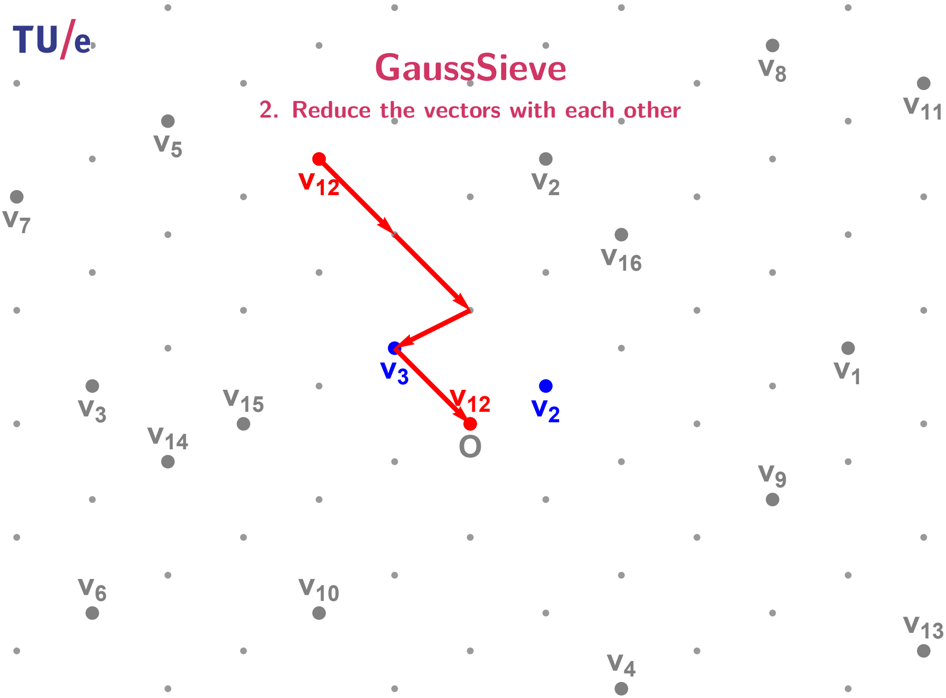
GaussSieve

2. Reduce the vectors with each other



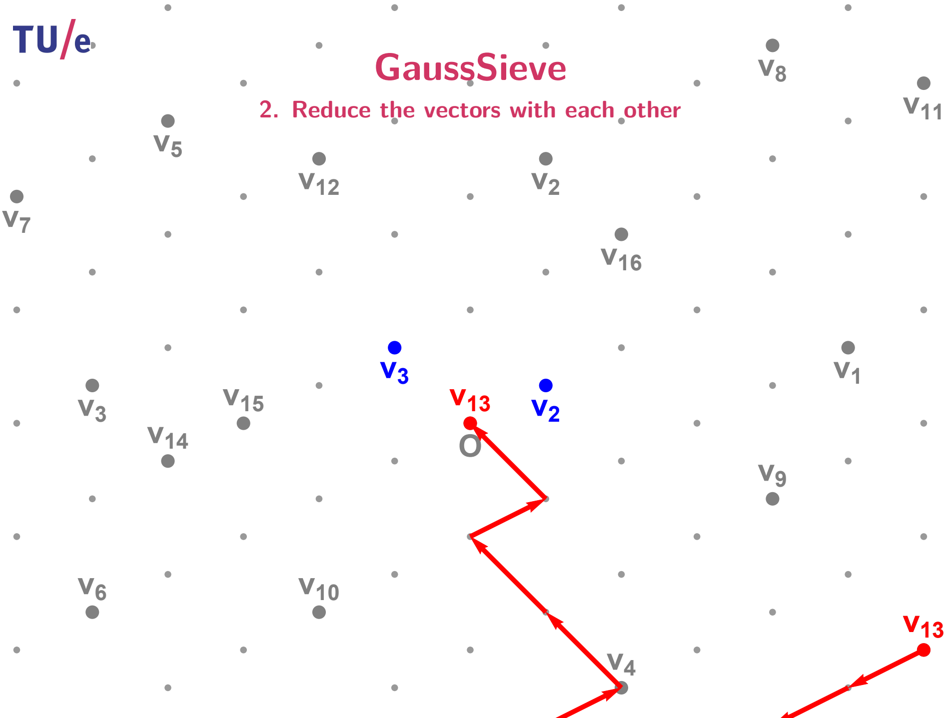
GaussSieve

2. Reduce the vectors with each other



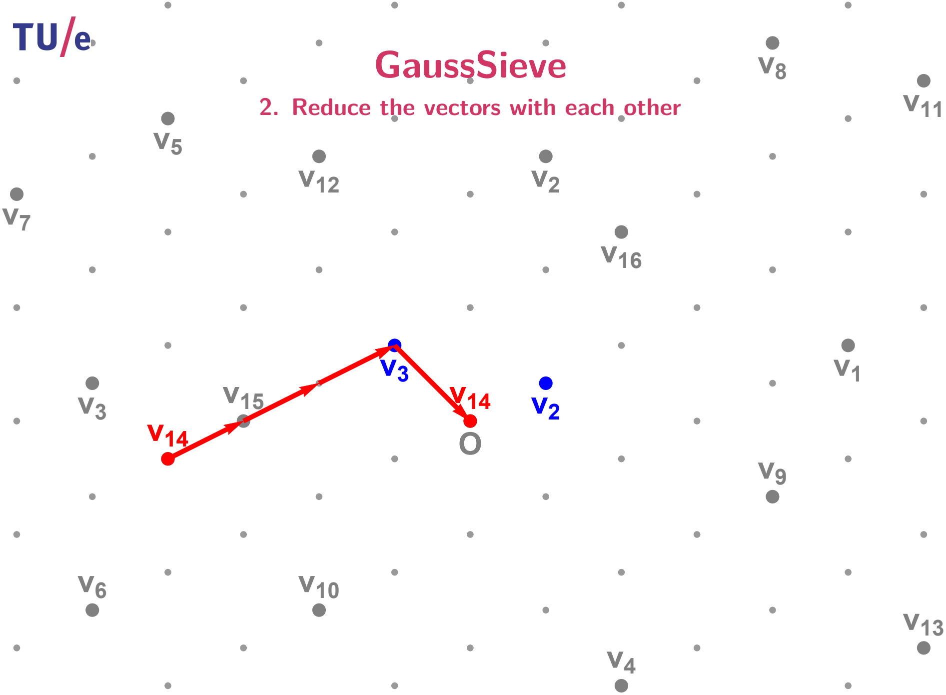
GaussSieve

2. Reduce the vectors with each other



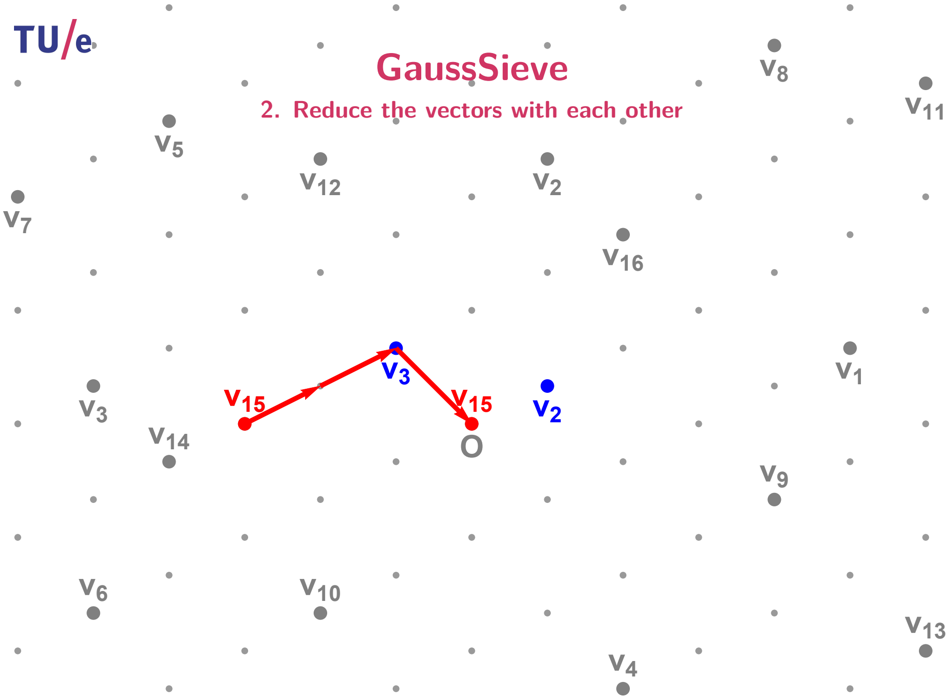
GaussSieve

2. Reduce the vectors with each other



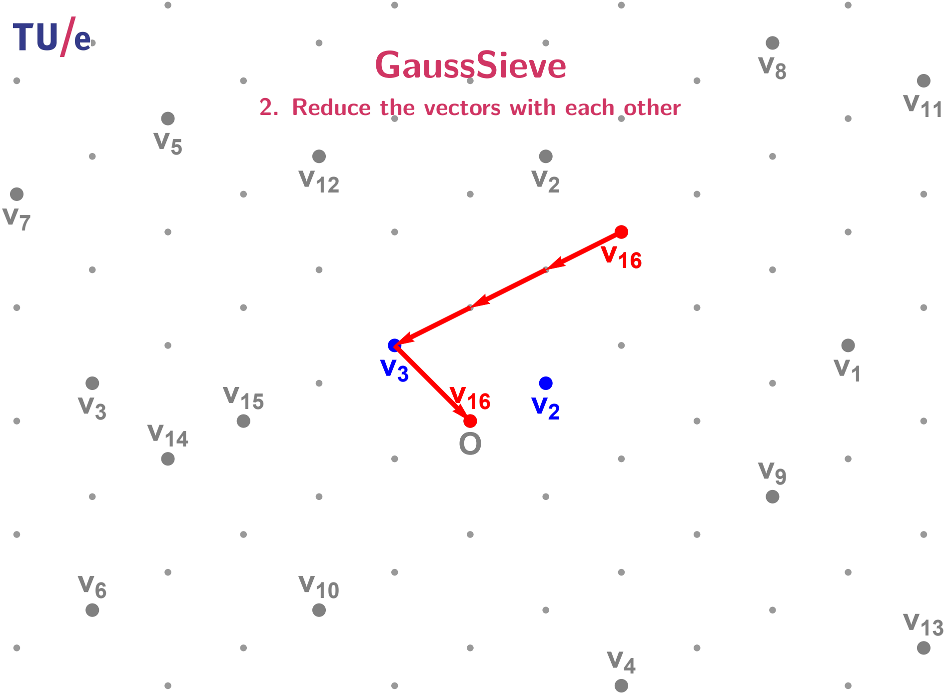
GaussSieve

2. Reduce the vectors with each other



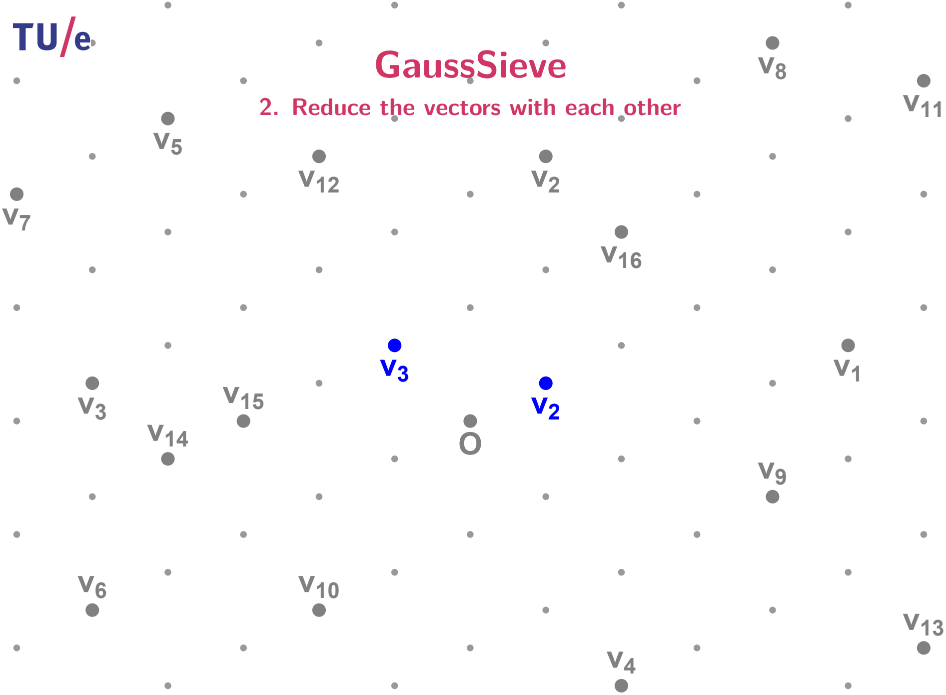
GaussSieve

2. Reduce the vectors with each other



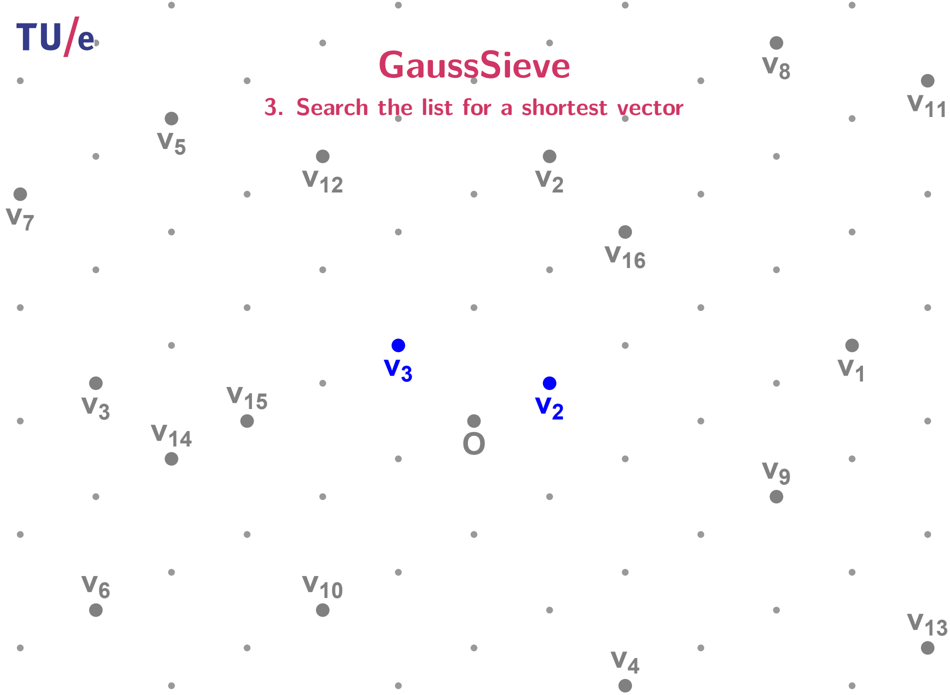
GaussSieve

2. Reduce the vectors with each other



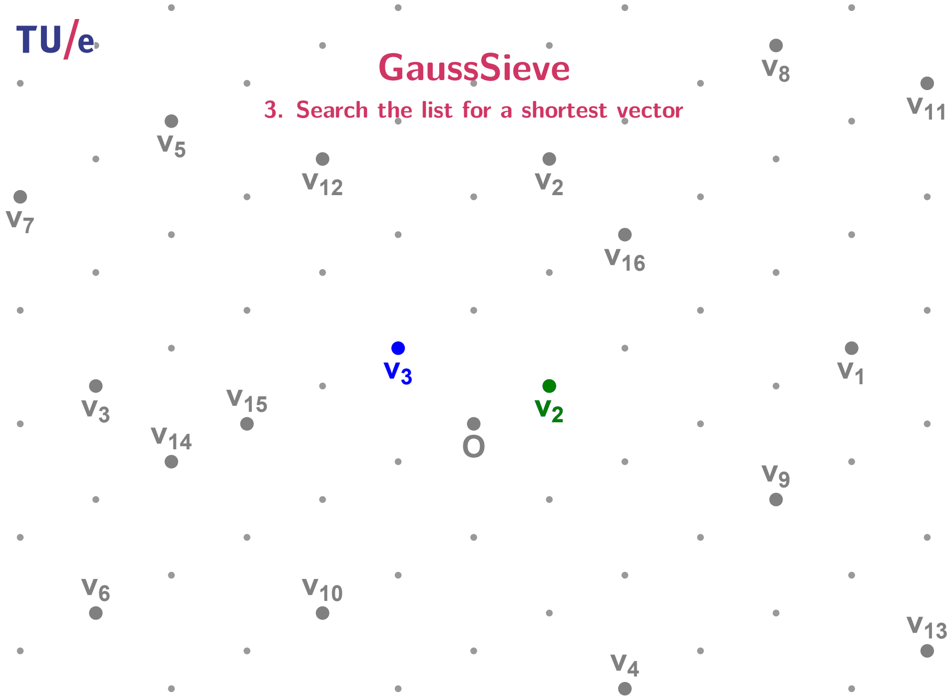
GaussSieve

3. Search the list for a shortest vector



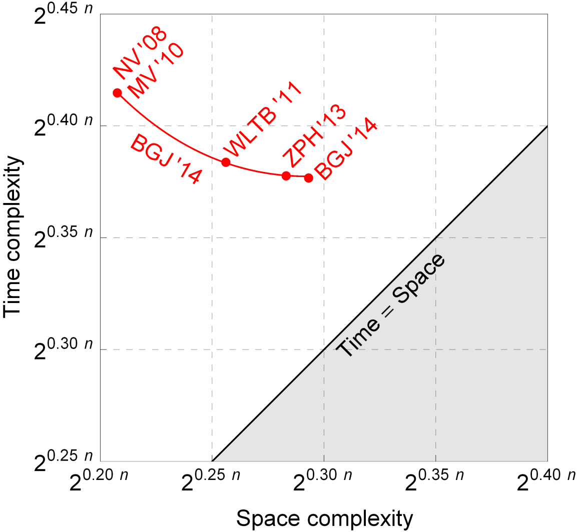
GaussSieve

3. Search the list for a shortest vector



Sieving

Space/time trade-off



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

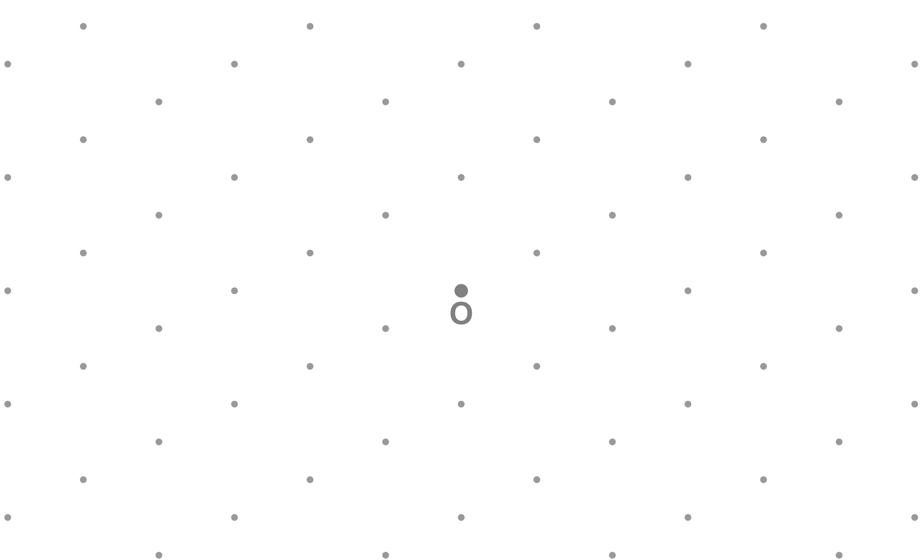
Introduction

> *"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*

(These hash functions are **not** cryptographic hash functions!)

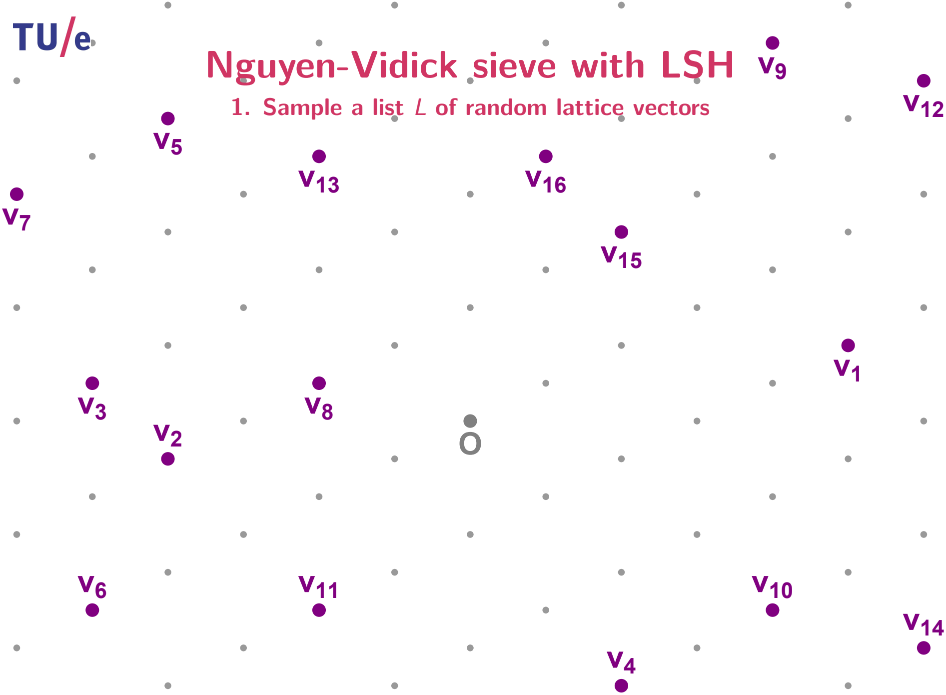
Nguyen-Vidick sieve with LSH

1. Sample a list L of random lattice vectors



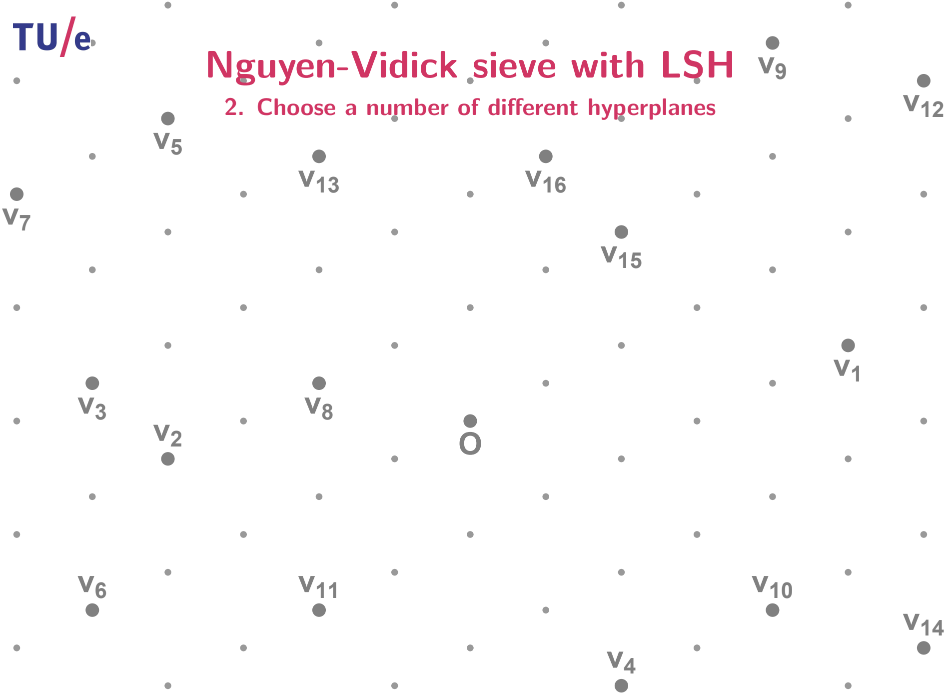
Nguyen-Vidick sieve with LSH

1. Sample a list L of random lattice vectors



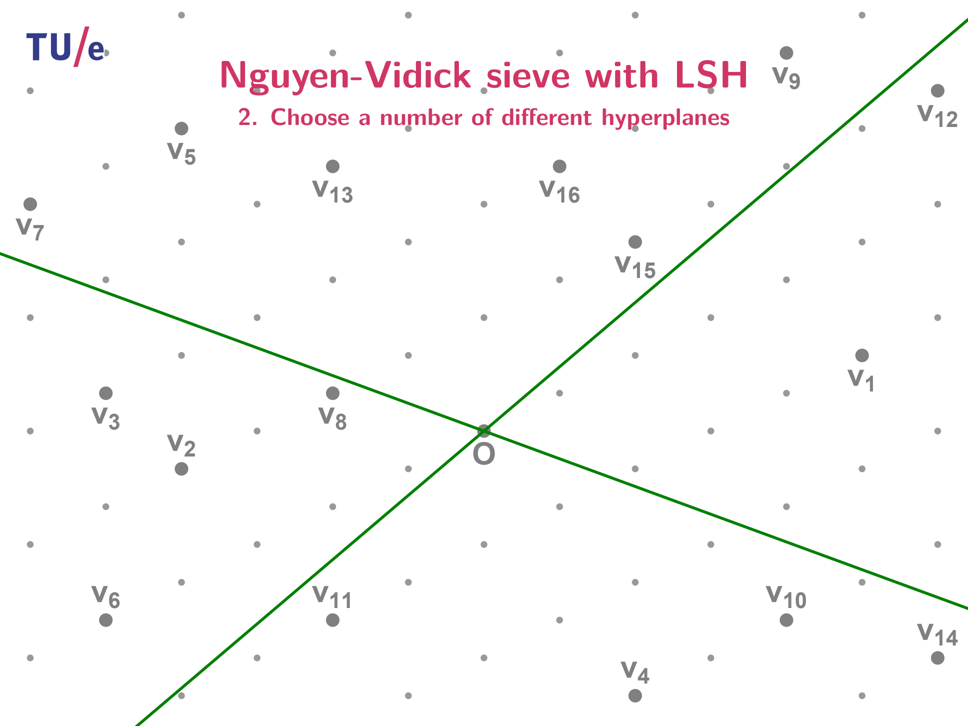
Nguyen-Vidick sieve with LSH

2. Choose a number of different hyperplanes



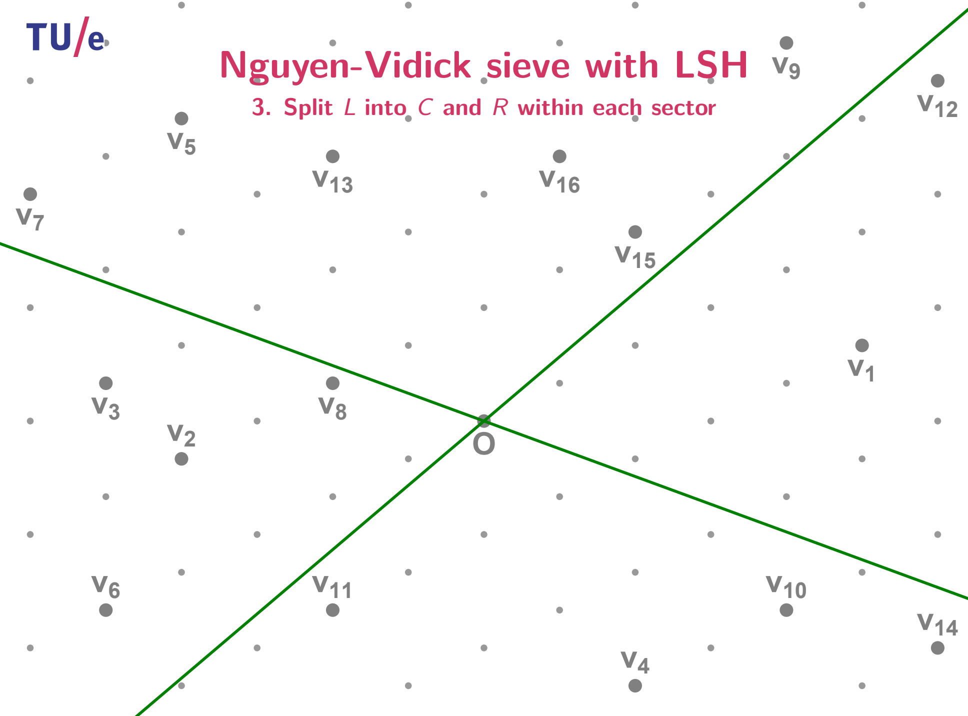
Nguyen-Vidick sieve with LSH

2. Choose a number of different hyperplanes



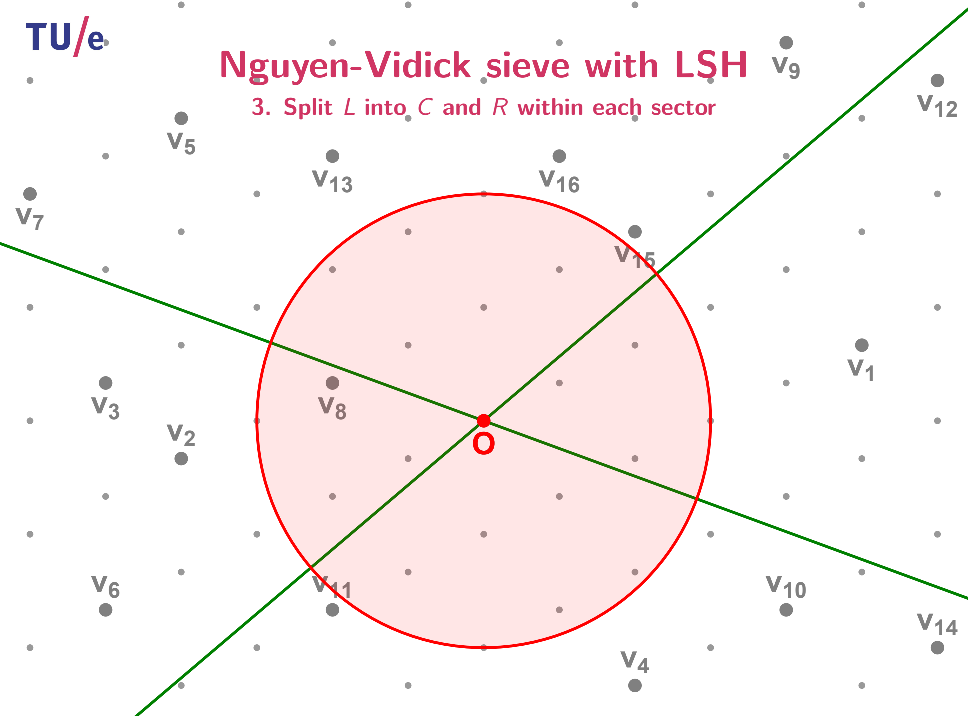
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



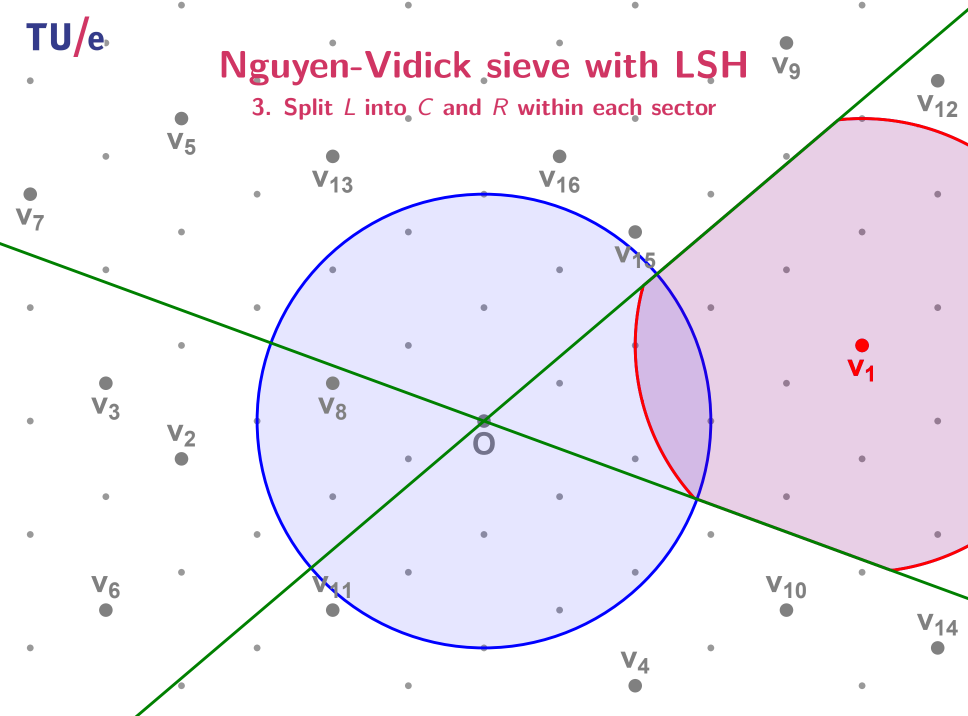
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



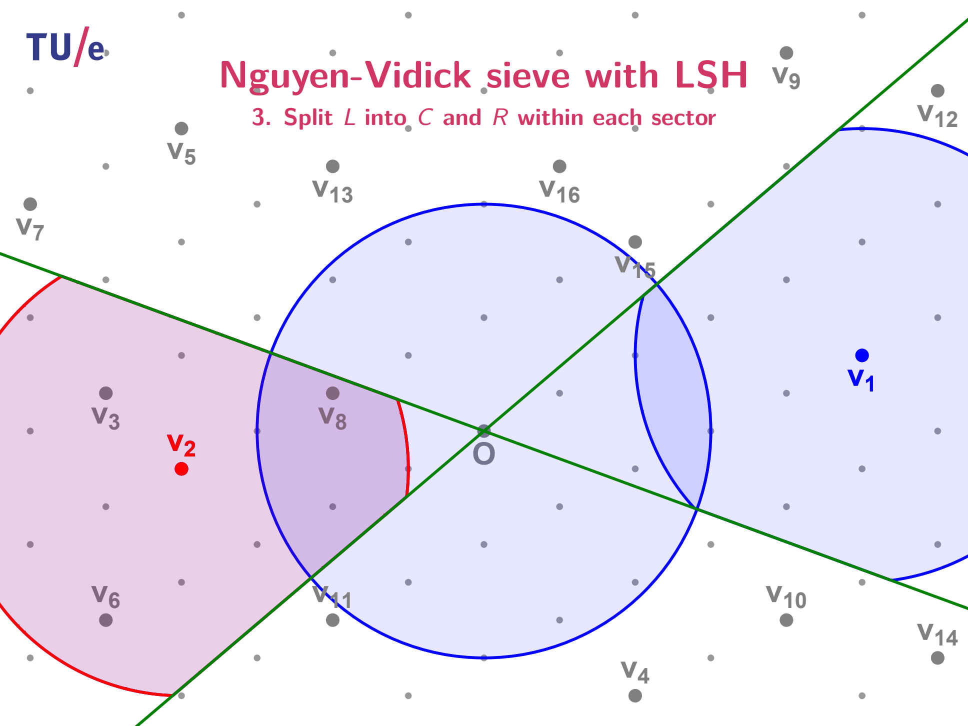
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



Nguyen-Vidick sieve with LSH

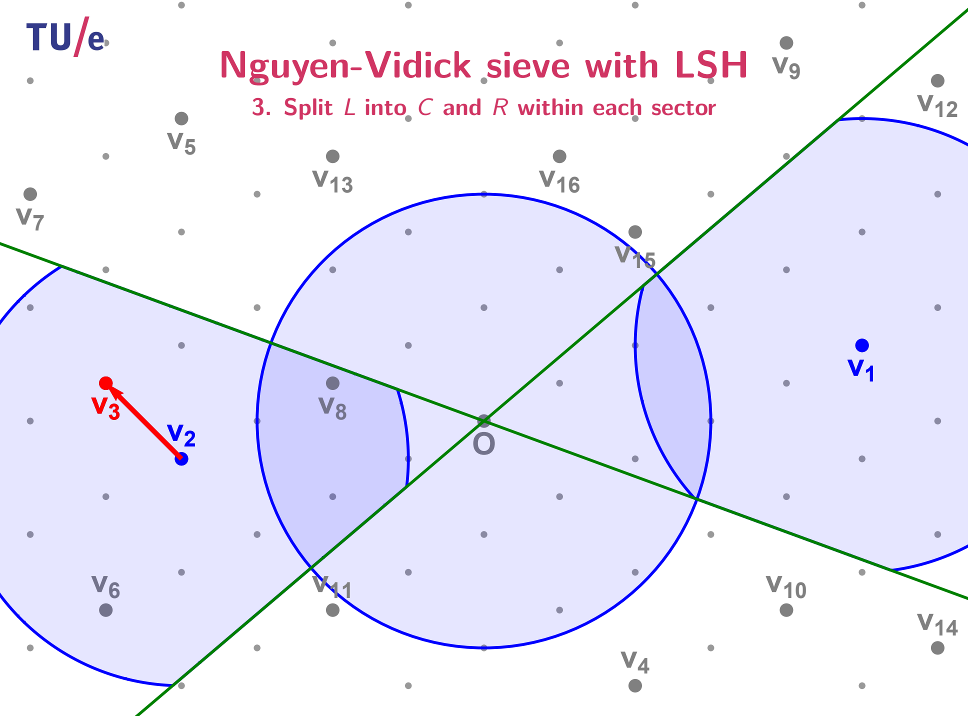
3. Split L into C and R within each sector



TU/e

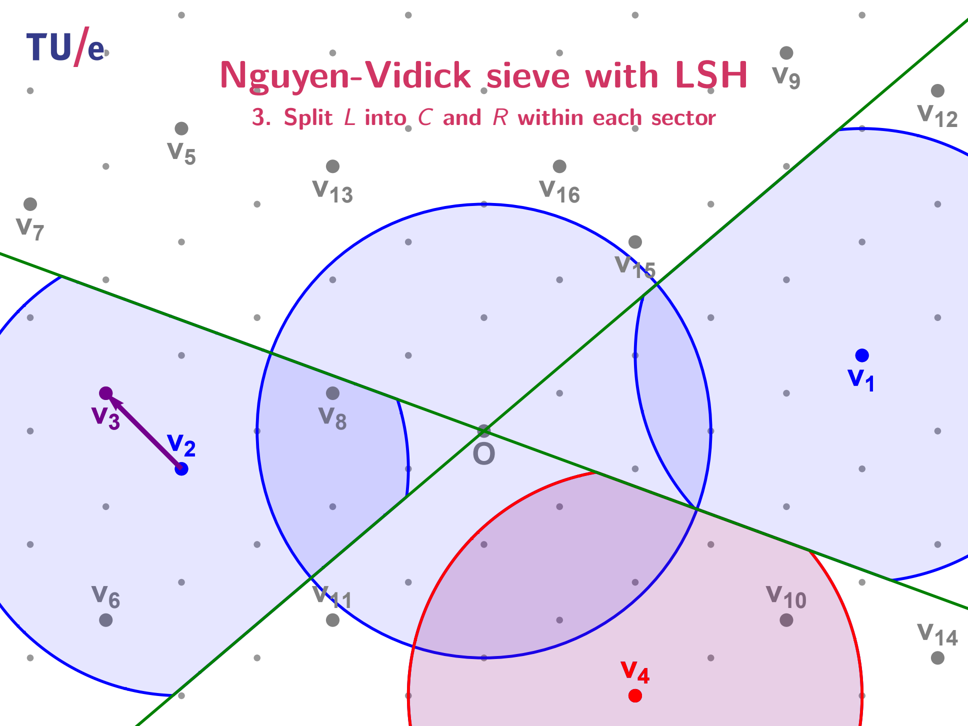
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



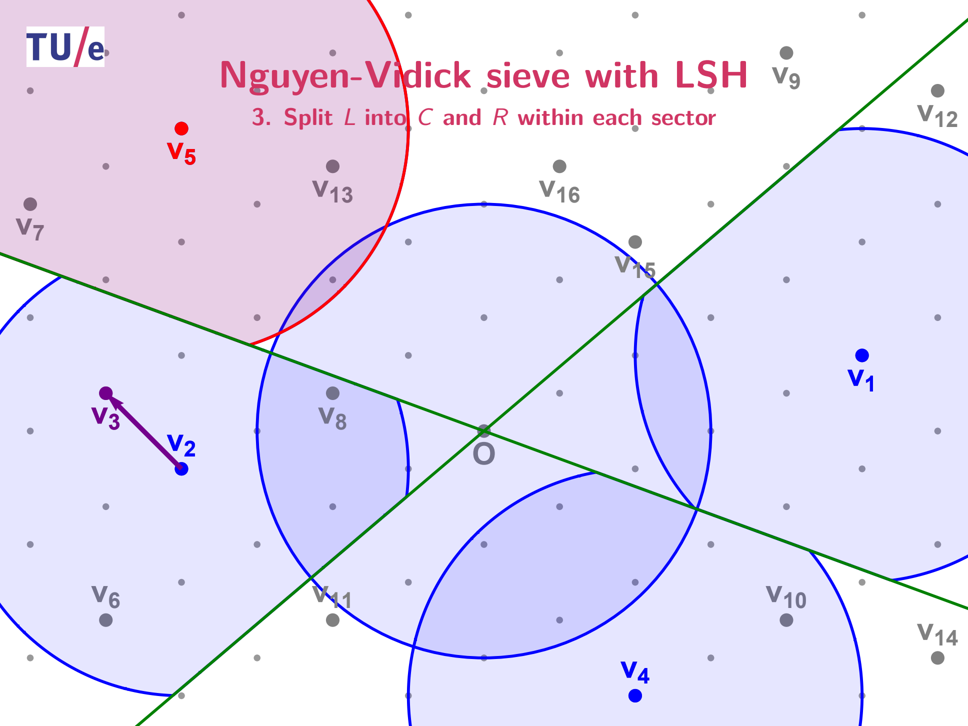
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



Nguyen-Vidick sieve with LSH

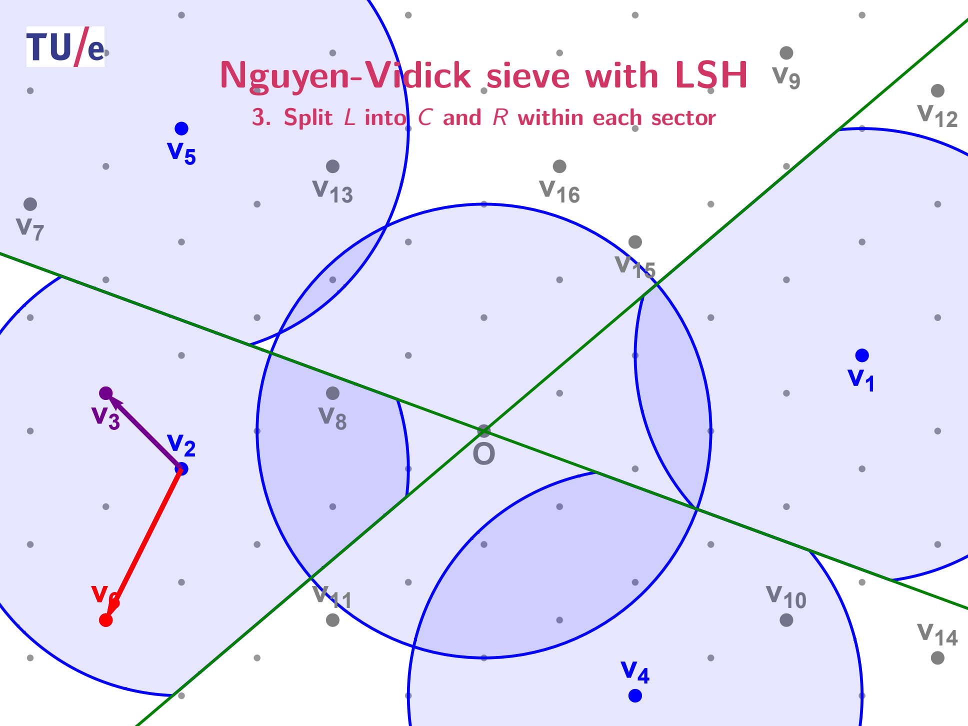
3. Split L into C and R within each sector



TU/e

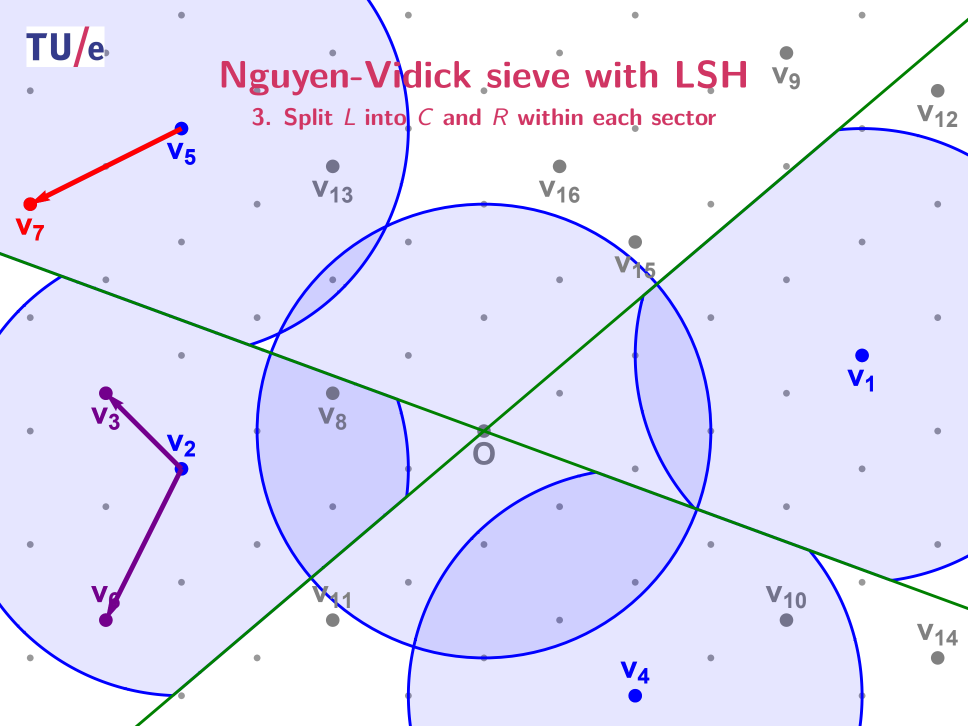
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



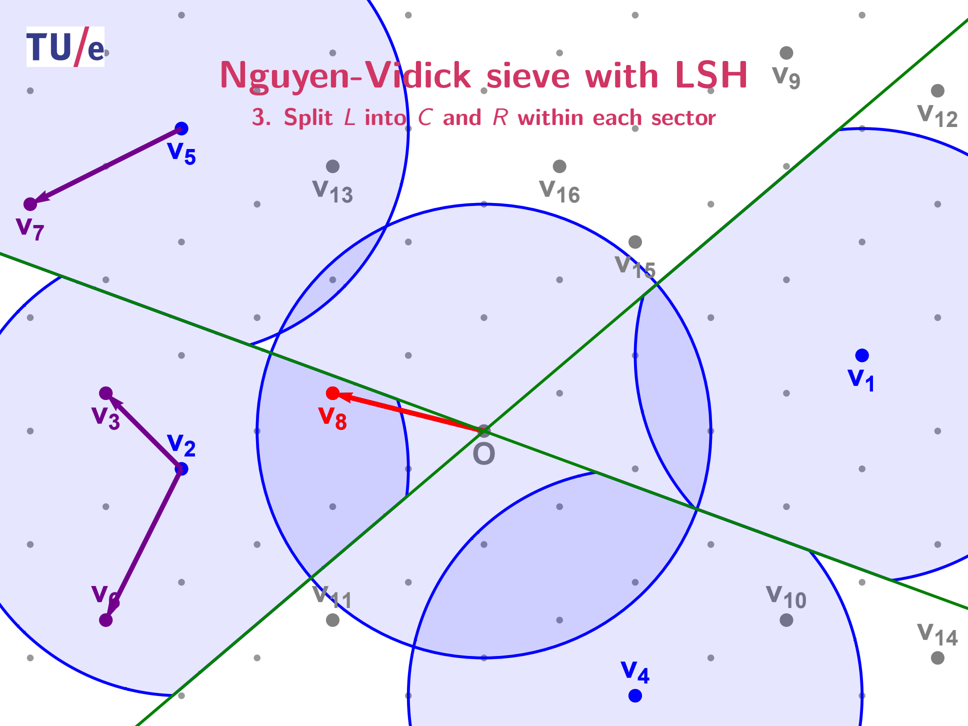
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



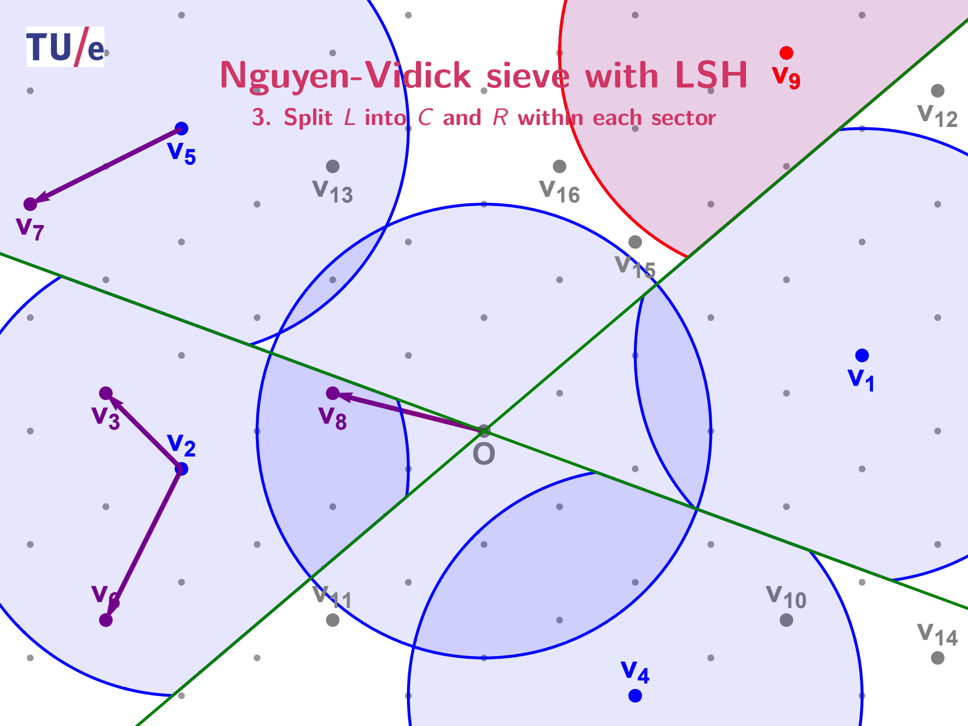
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



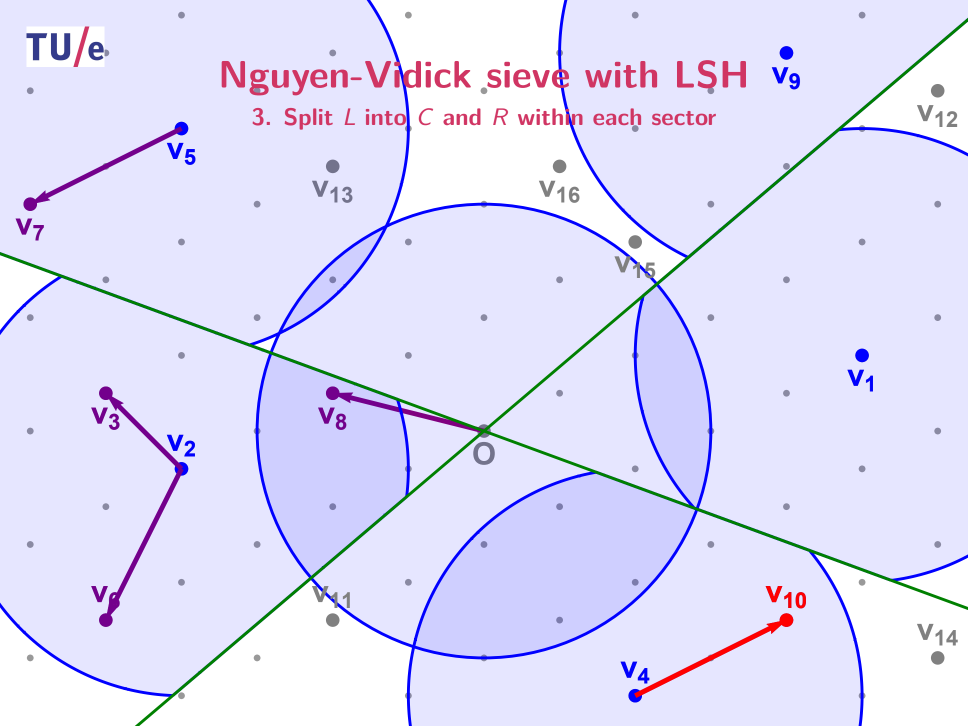
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



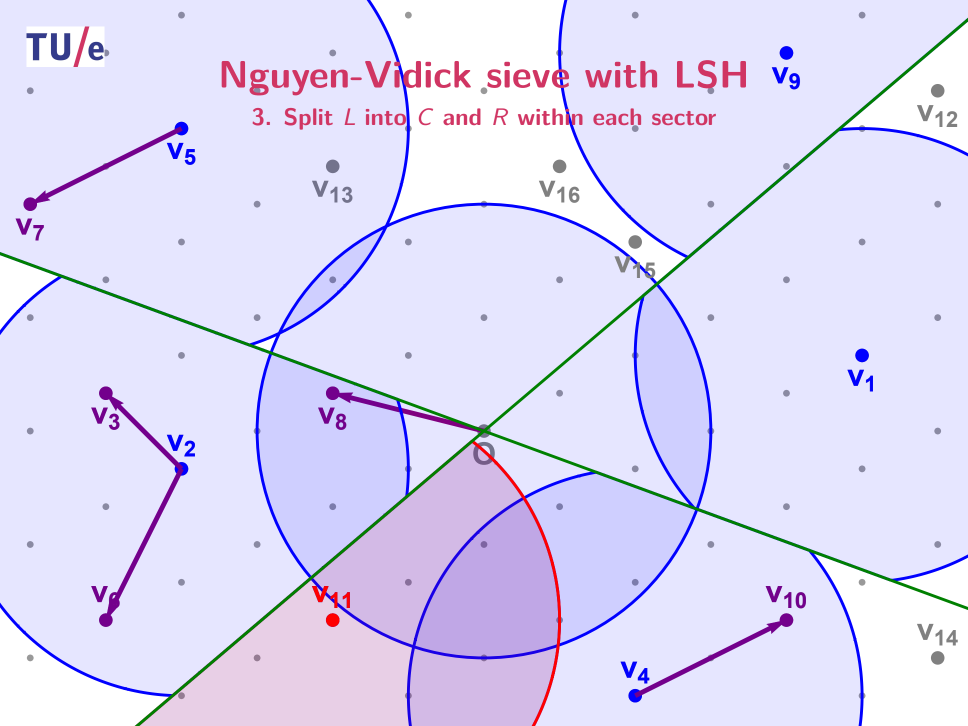
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



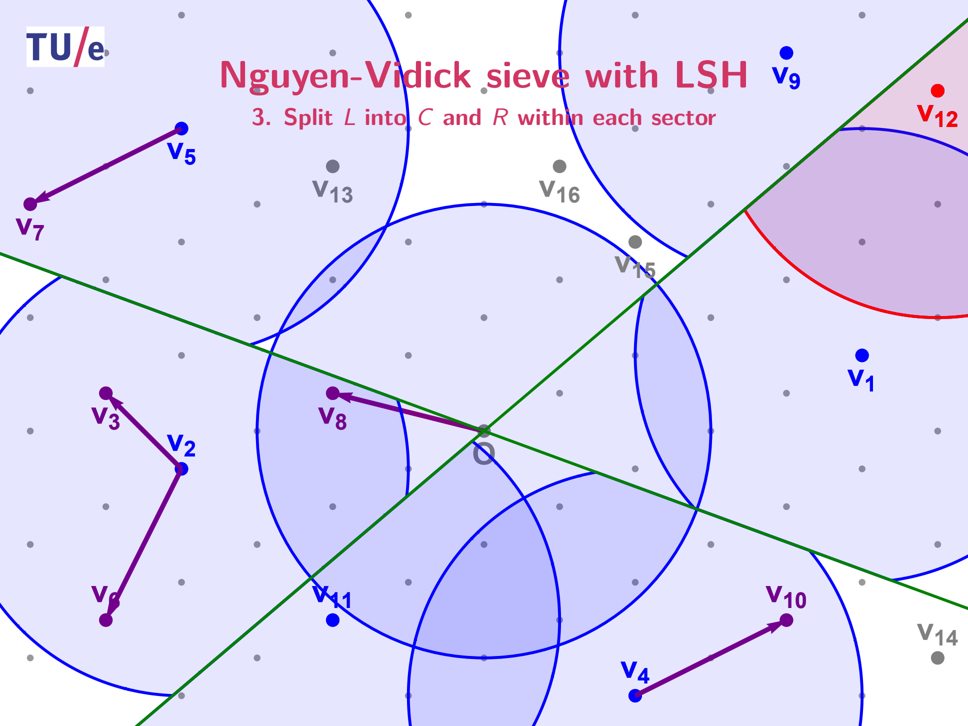
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



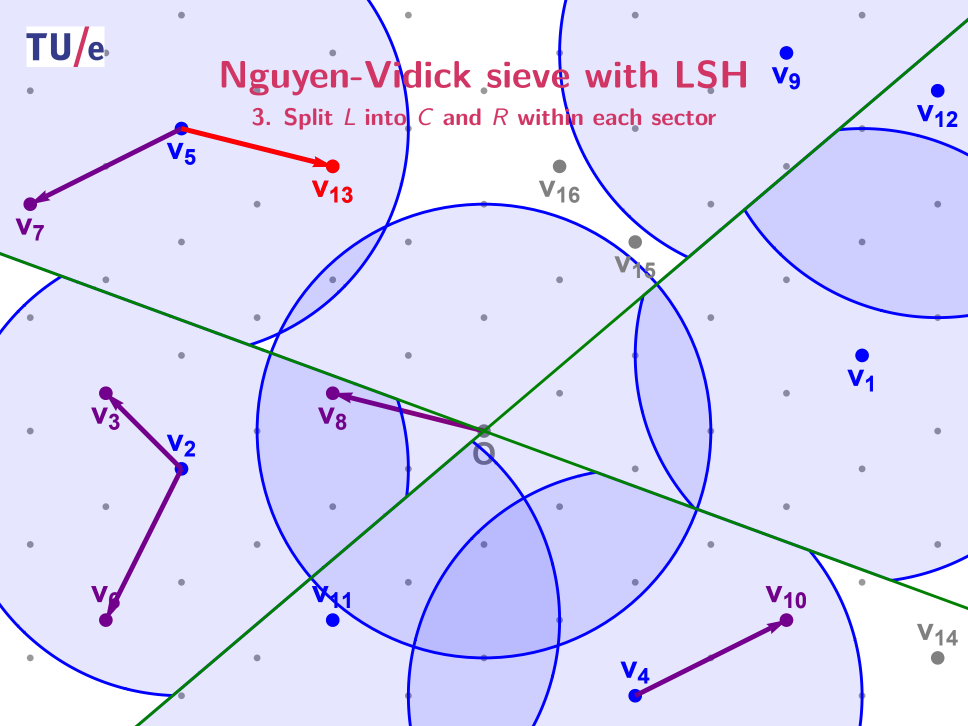
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



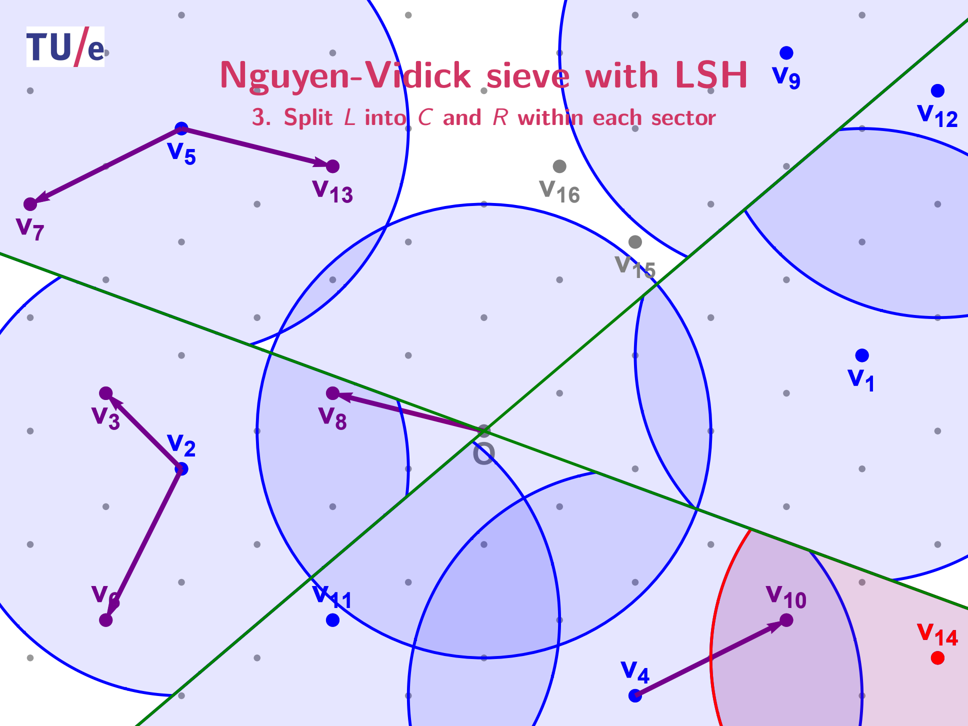
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



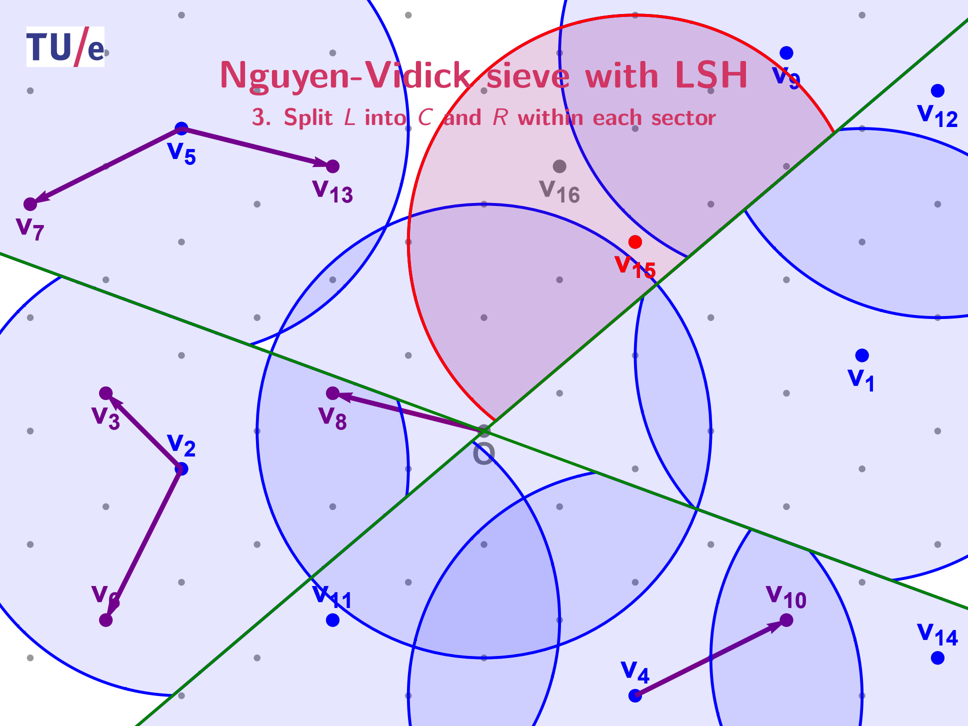
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



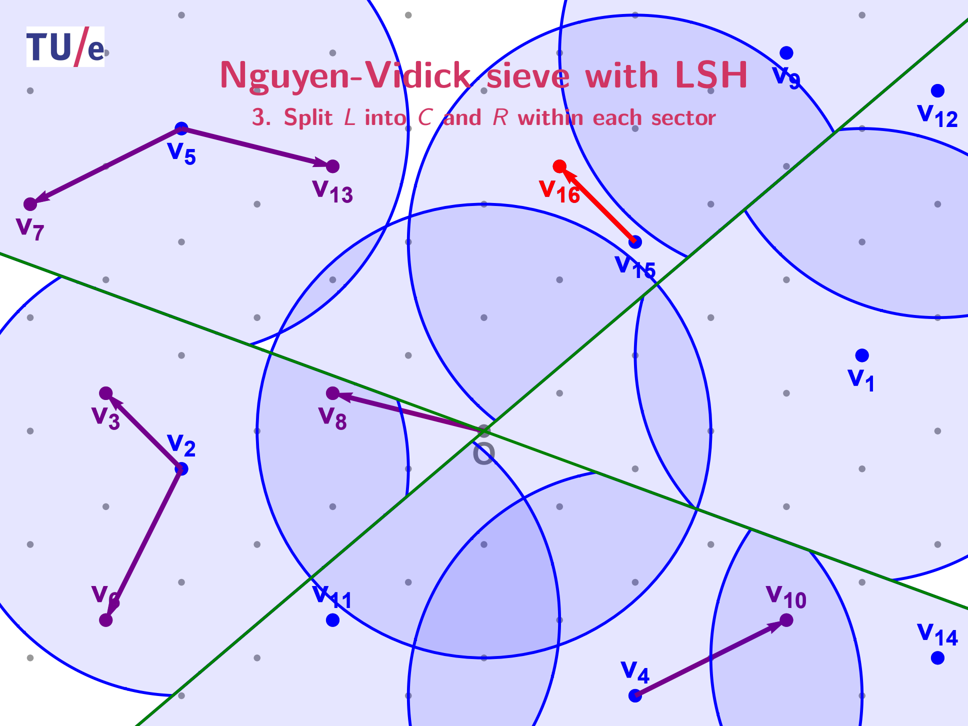
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



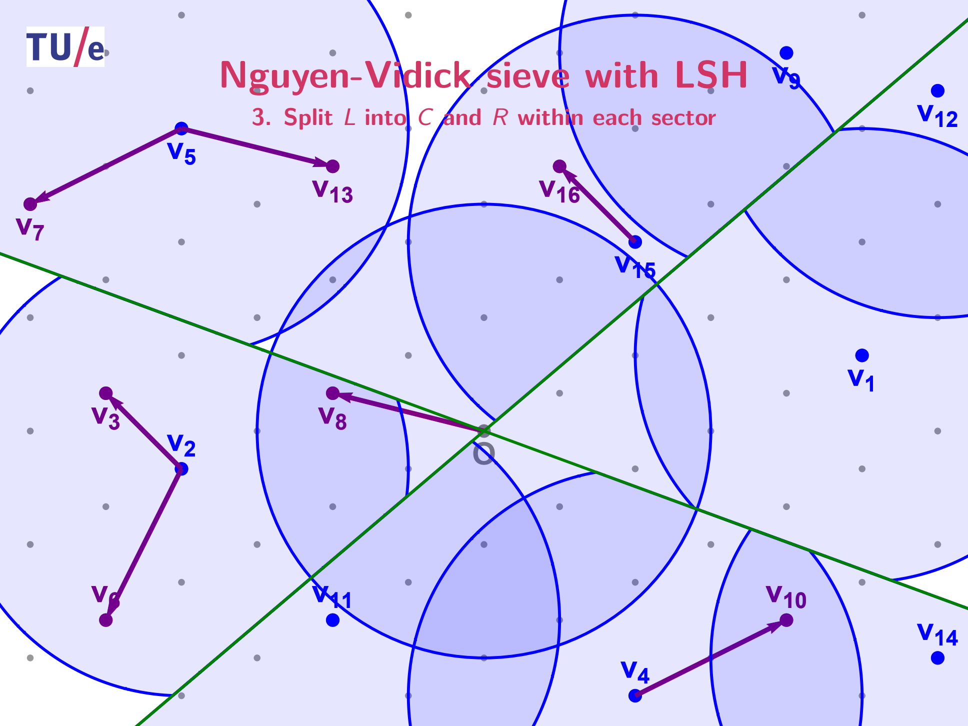
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



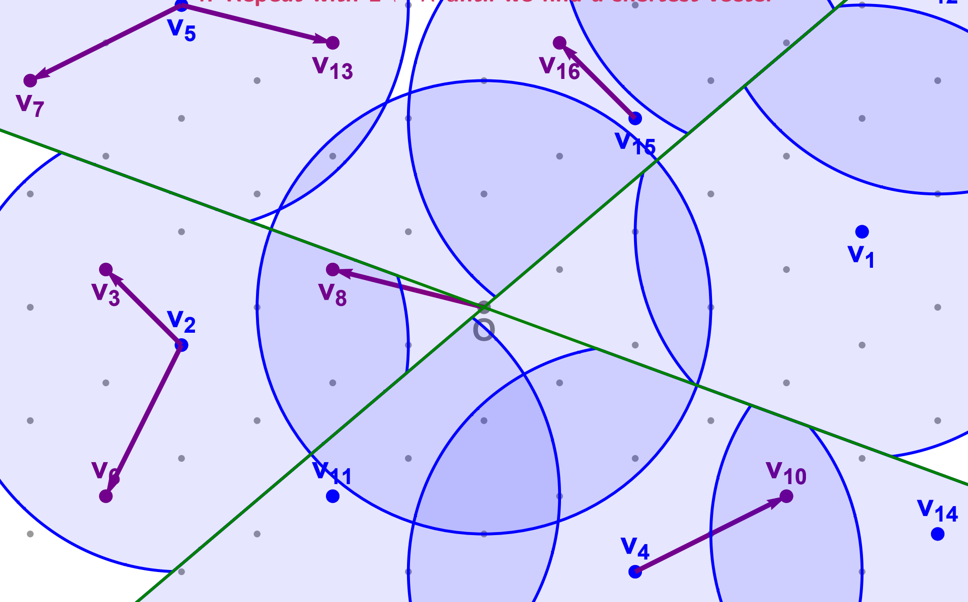
Nguyen-Vidick sieve with LSH

3. Split L into C and R within each sector



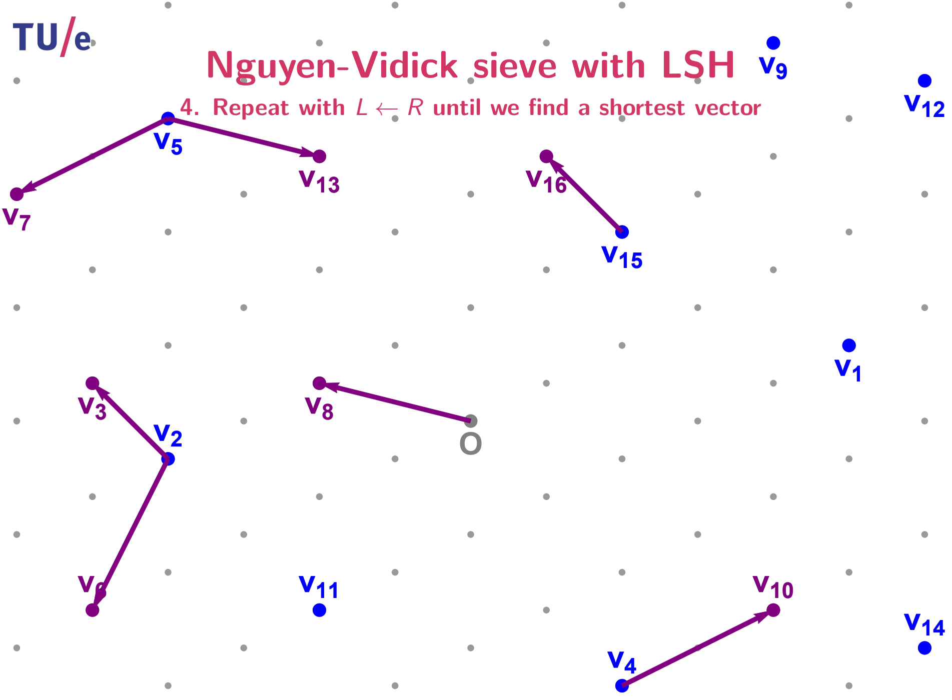
Nguyen-Vidick sieve with LSH

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



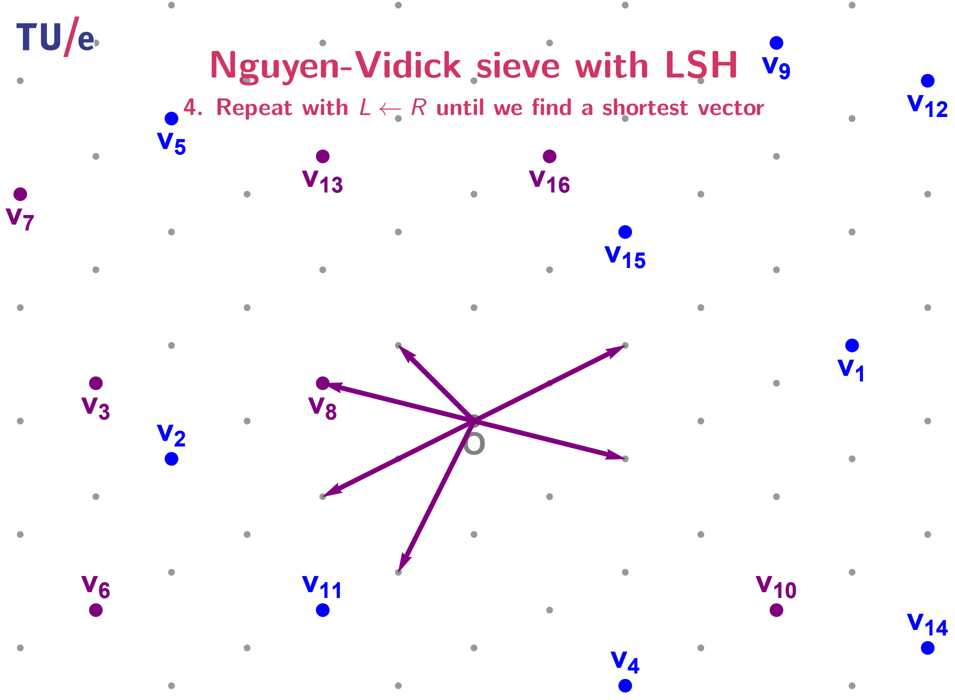
Nguyen-Vidick sieve with LSH

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



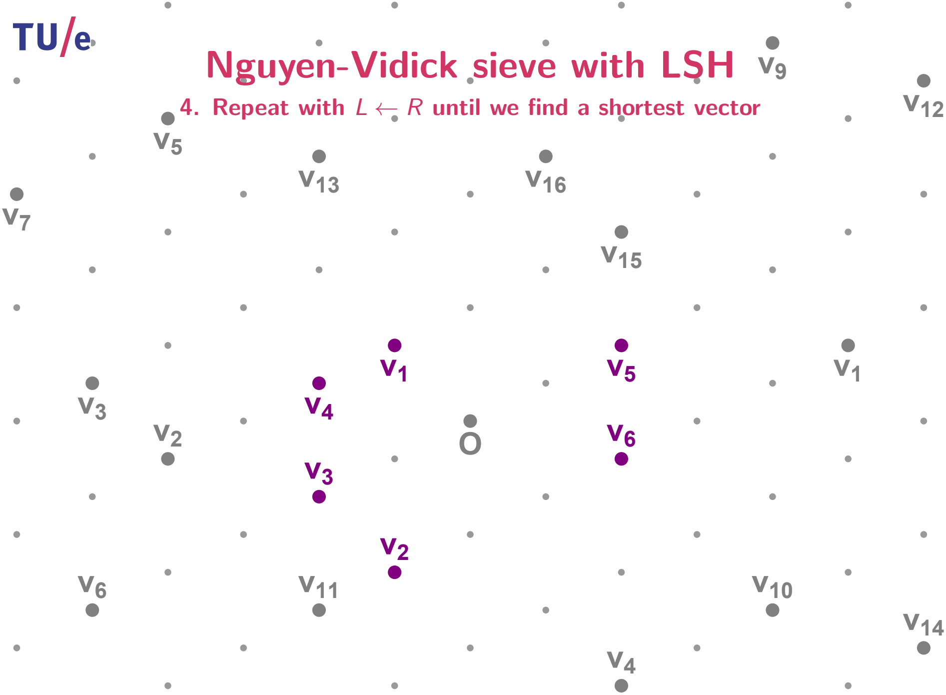
Nguyen-Vidick sieve with LSH

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



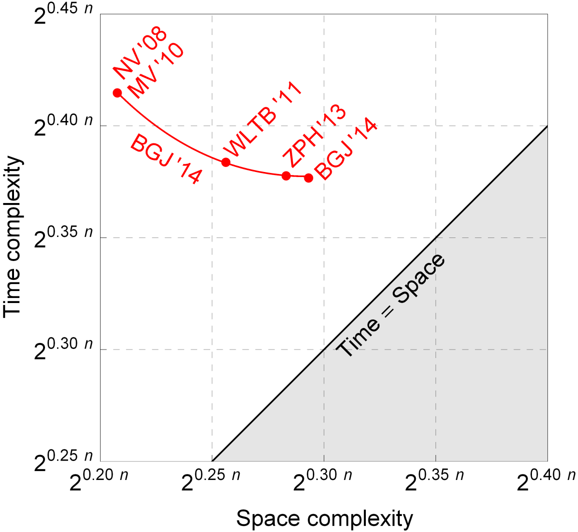
Nguyen-Vidick sieve with LSH

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



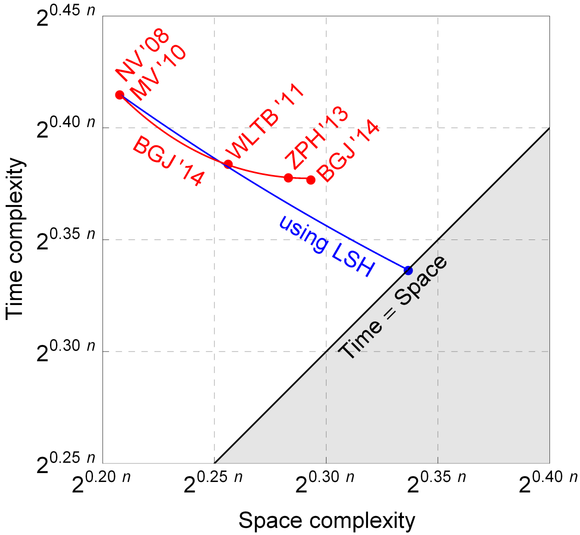
Sieving

Space/time trade-off



Sieving with LSH

Space/time trade-off



The End

Do not forget to register for the exam before **January 11, 2015**!