HashSieve: Theory and Practice

Part 1: Theory

Artur Mariano, Thijs Laarhoven, Christian Bischof

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

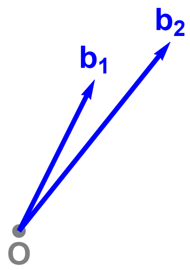
CROSSING seminar, Darmstadt, Germany
(October 15, 2015)

Lattices

What is a lattice?

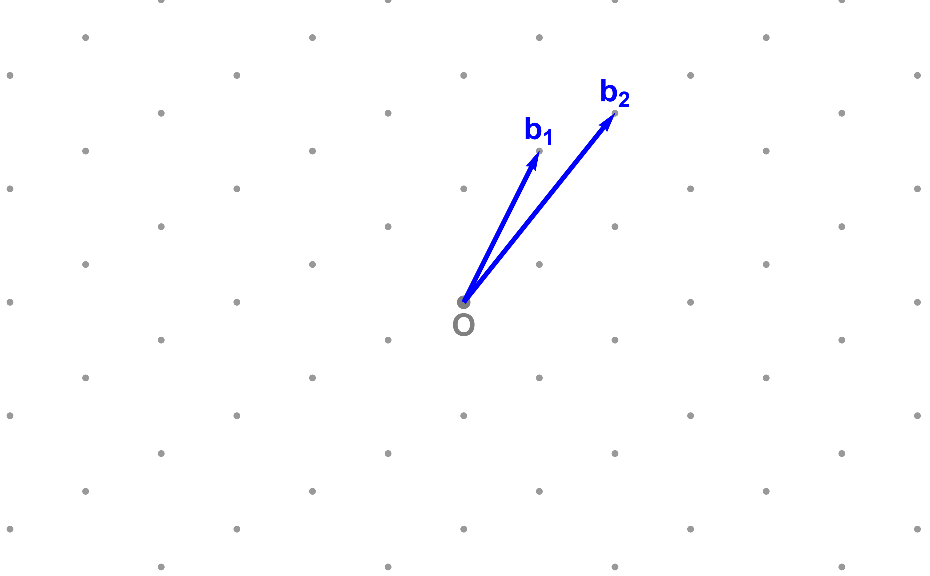
Lattices

What is a lattice?



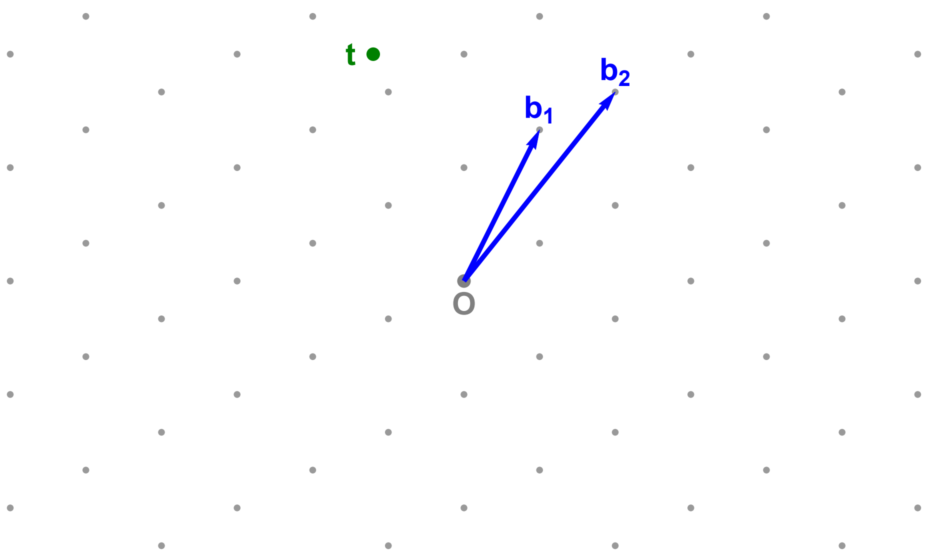
Lattices

What is a lattice?



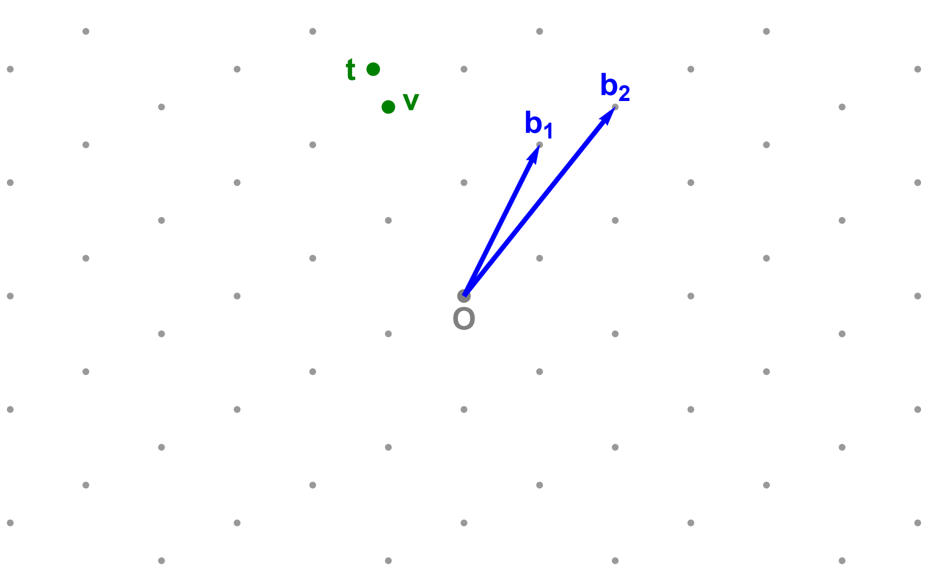
Lattices

Closest Vector Problem (CVP)



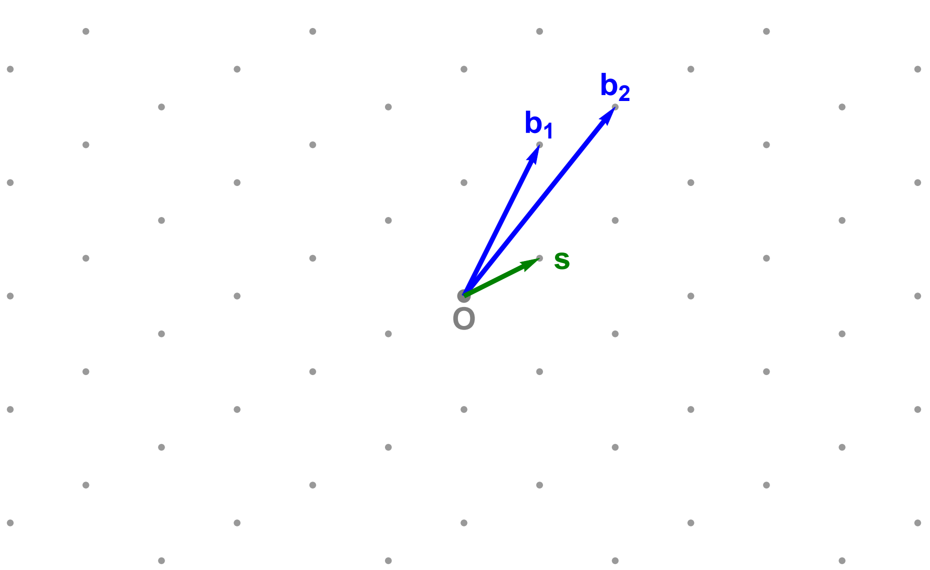
Lattices

Closest Vector Problem (CVP)



Lattices

Shortest Vector Problem (SVP)



Lattices

Applications

- "Constructive cryptography": Lattice-based cryptosystems
  - Based on hard lattice problems (SVP, CVP, LWE, SIS)
  - NTRU cryptosystem [HPS98, . . . , HPSSWZ15]
  - HIMMO key pre-distribution scheme [GMGPGRST14]
  - Fully Homomorphic Encryption [Gen09, . . . , CM15]
  - Candidate for "post-quantum cryptography"

Lattices

Applications

- "Constructive cryptography": Lattice-based cryptosystems
  - Based on hard lattice problems (SVP, CVP, LWE, SIS)
  - NTRU cryptosystem [HPS98, . . . , HPSSWZ15]
  - HIMMO key pre-distribution scheme [GMGPGRST14]
  - Fully Homomorphic Encryption [Gen09, . . . , CM15]
  - Candidate for "post-quantum cryptography"
- "Destructive cryptography": Lattice cryptanalysis
  - Attack knapsack-based cryptosystems [Sha82, LO85, . . .]
  - Attack RSA with Coppersmith's method [Cop97, . . .]
  - Attack lattice-based cryptosystems [Ngu99, JJ00, . . .]

Lattices

Applications

- "Constructive cryptography": Lattice-based cryptosystems
  - Based on hard lattice problems (SVP, CVP, LWE, SIS)
  - NTRU cryptosystem [HPS98, . . . , HPSSWZ15]
  - HIMMO key pre-distribution scheme [GMGPGRST14]
  - Fully Homomorphic Encryption [Gen09, . . . , CM15]
  - Candidate for "post-quantum cryptography"
- "Destructive cryptography": Lattice cryptanalysis
  - Attack knapsack-based cryptosystems [Sha82, LO85, . . .]
  - Attack RSA with Coppersmith's method [Cop97, . . .]
  - Attack lattice-based cryptosystems [Ngu99, JJ00, . . .]

How hard are hard lattice problems such as SVP?

Lattices

Exact SVP algorithms

	Algorithm	$\log_2(\text{Time})$	$\log_2(\text{Space})$
Provable SVP	Enumeration [Poh81, Kan83, ..., MW15]	$\Omega(n \log n)$	$O(\log n)$
	AKS-sieve [AKS01, NV08, MV10, HPS11]	$3.398n$	$1.985n$
	ListSieve [MV10, MDB14]	$3.199n$	$1.327n$
	AKS-sieve-birthday [PS09, HPS11]	$2.648n$	$1.324n$
	ListSieve-birthday [PS09]	$2.465n$	$1.233n$
	Voronoi cell algorithm [AEVZ02, MV10b]	$2.000n$	$1.000n$
	Discrete Gaussians [ADRS15, ADS15, Ste16]	$1.000n$	$1.000n$
Heuristic SVP	Nguyen-Vidick sieve [NV08]	$0.415n$	$0.208n$
	GaussSieve [MV10, ..., IKMT14, BNvdP14]	$0.415n?$	$0.208n$
	Two-level sieve [WLTB11]	$0.384n$	$0.256n$
	Three-level sieve [ZPH13]	$0.3778n$	$0.283n$
	Overlattice sieve [BGJ14]	$0.3774n$	$0.293n$
	Hyperplane LSH [Laa15, MLB15]	$0.337n$	$0.208n$

Nguyen-Vidick sieve

Nguyen-Vidick sieve

1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve

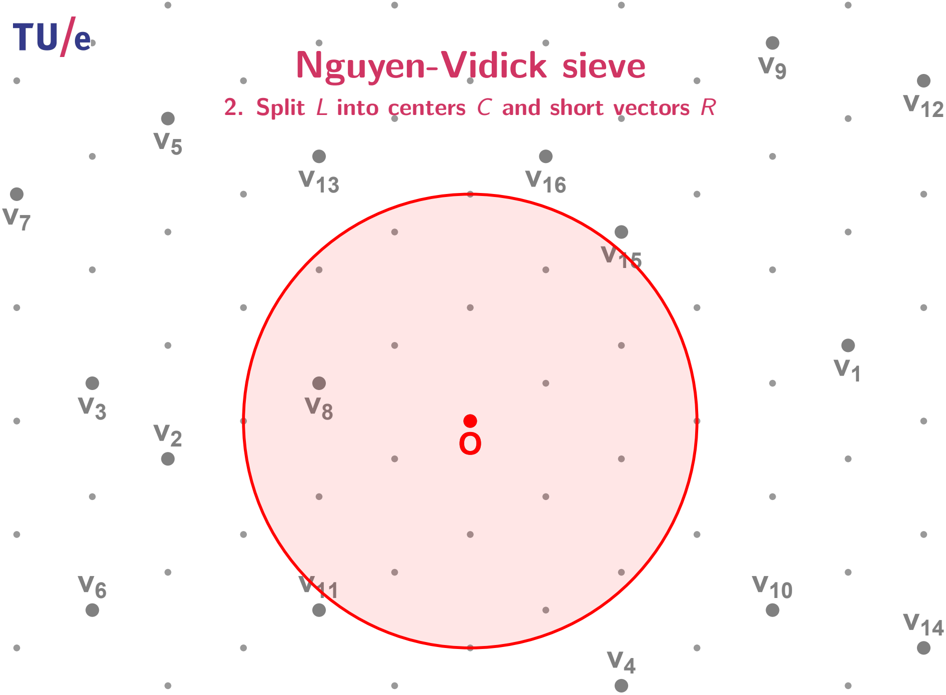
1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve

2. Split L into centers C and short vectors R

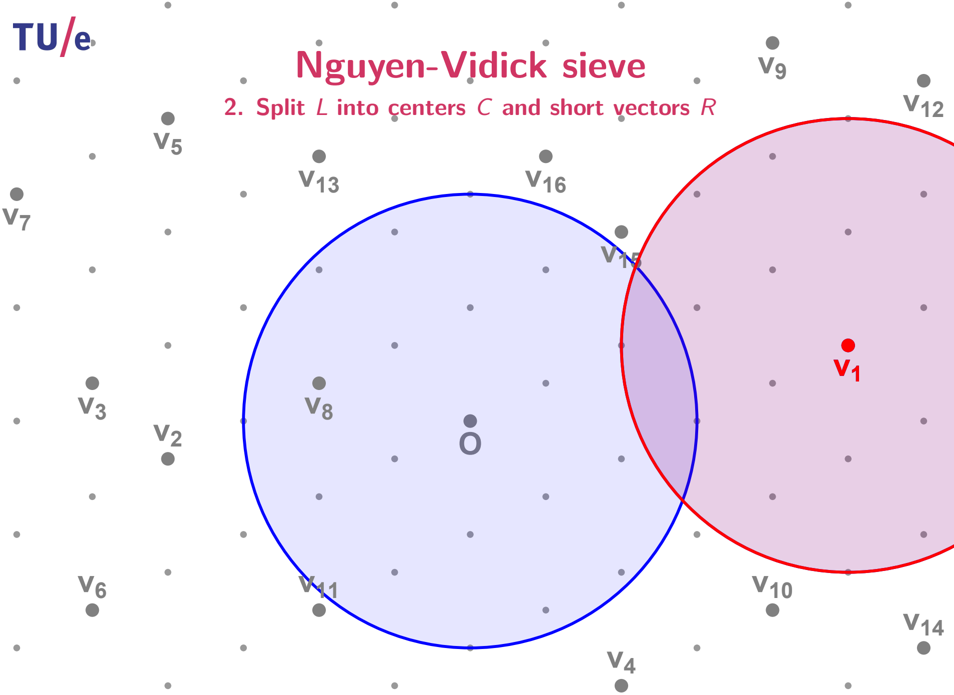
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



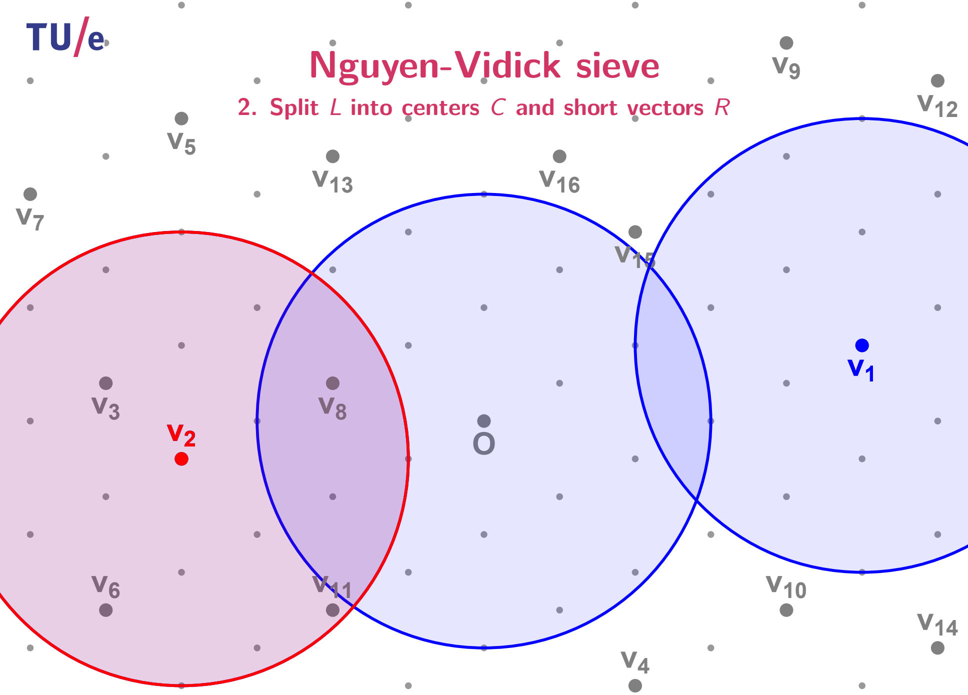
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



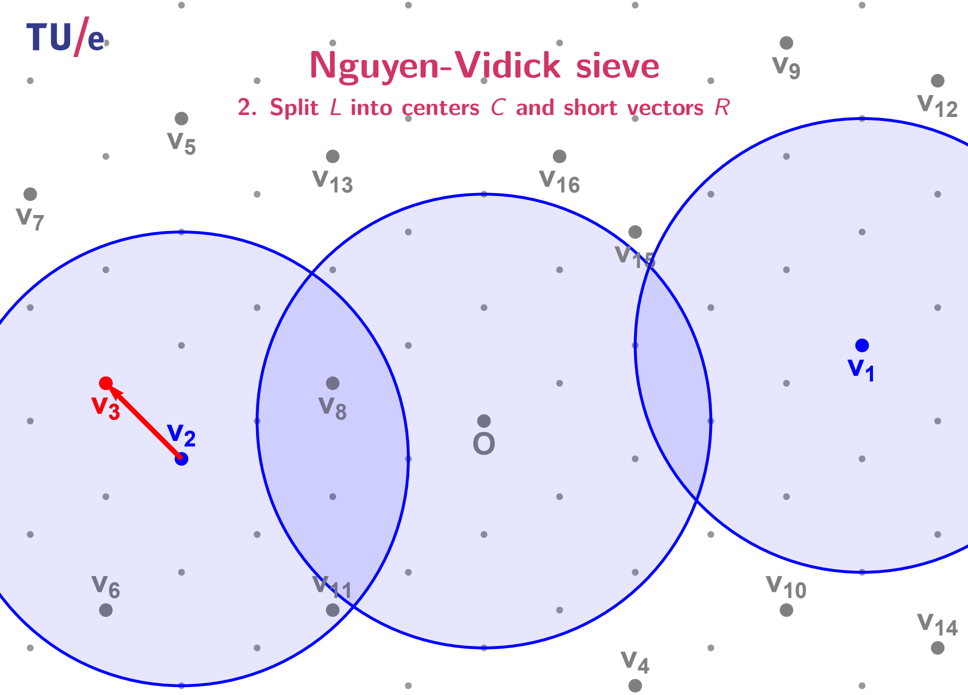
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



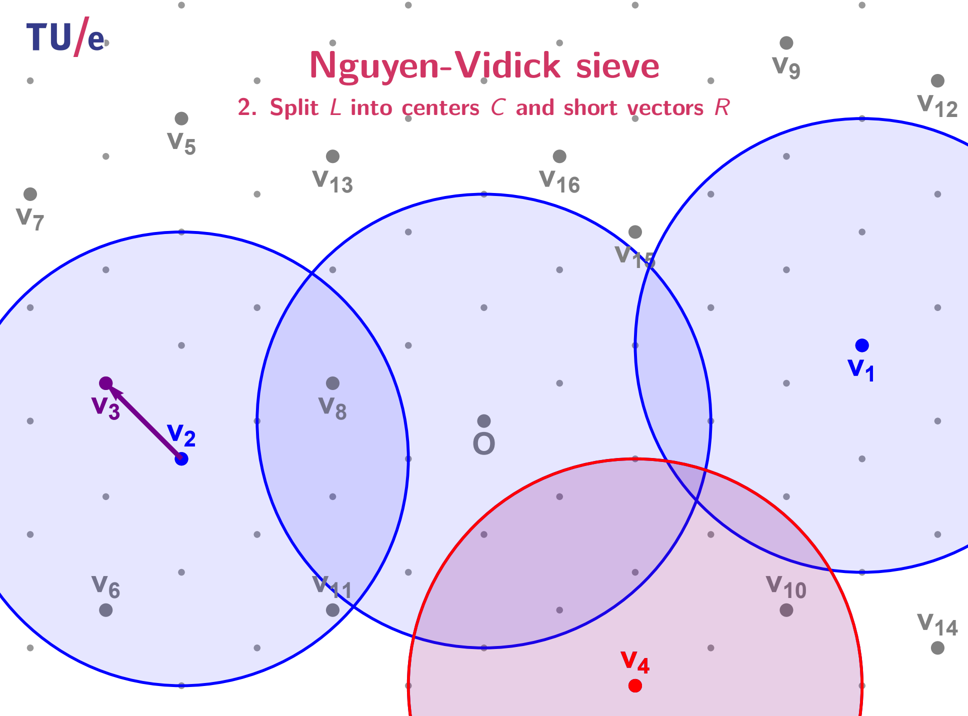
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



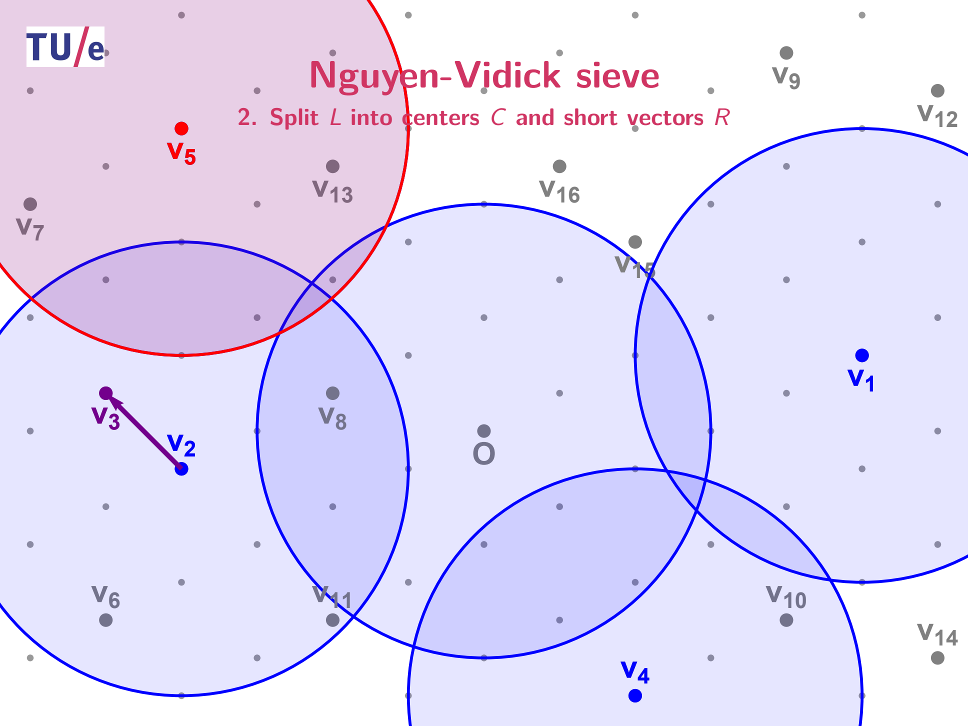
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



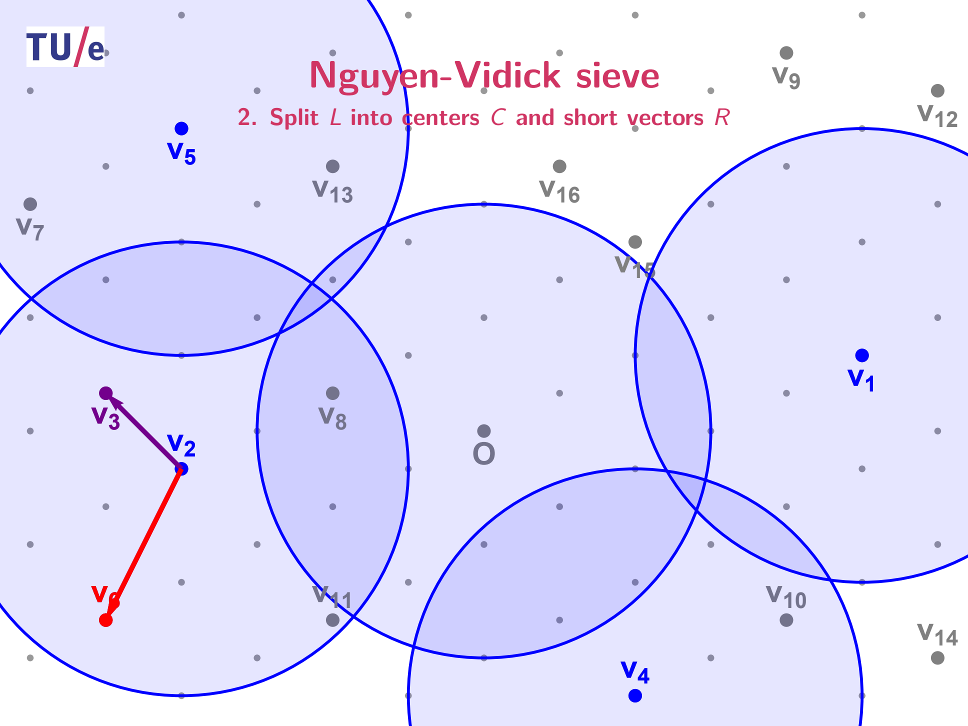
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



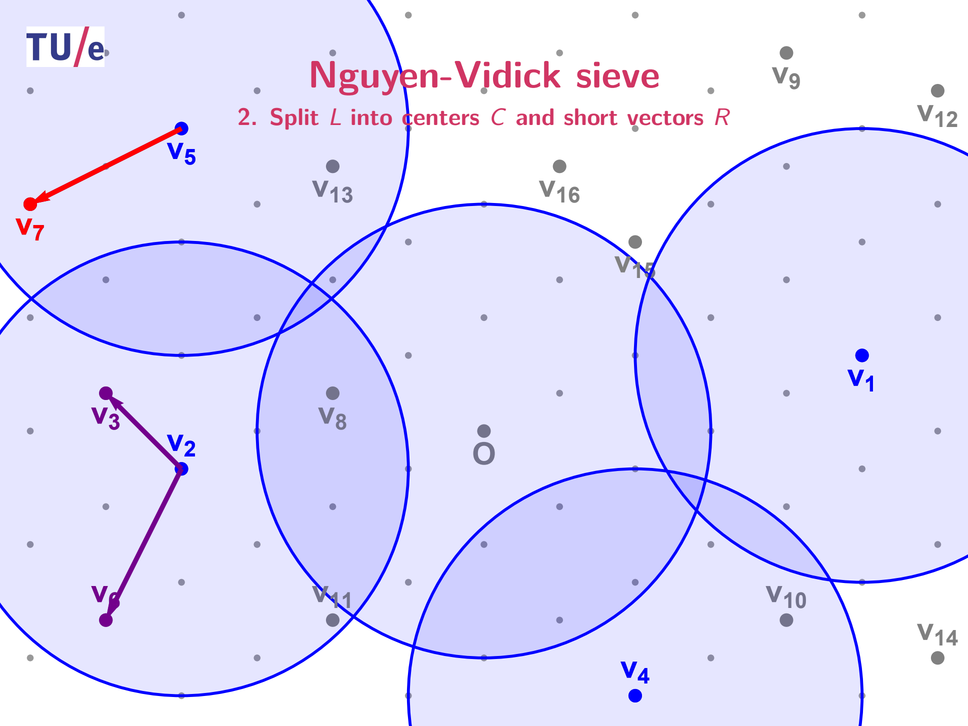
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



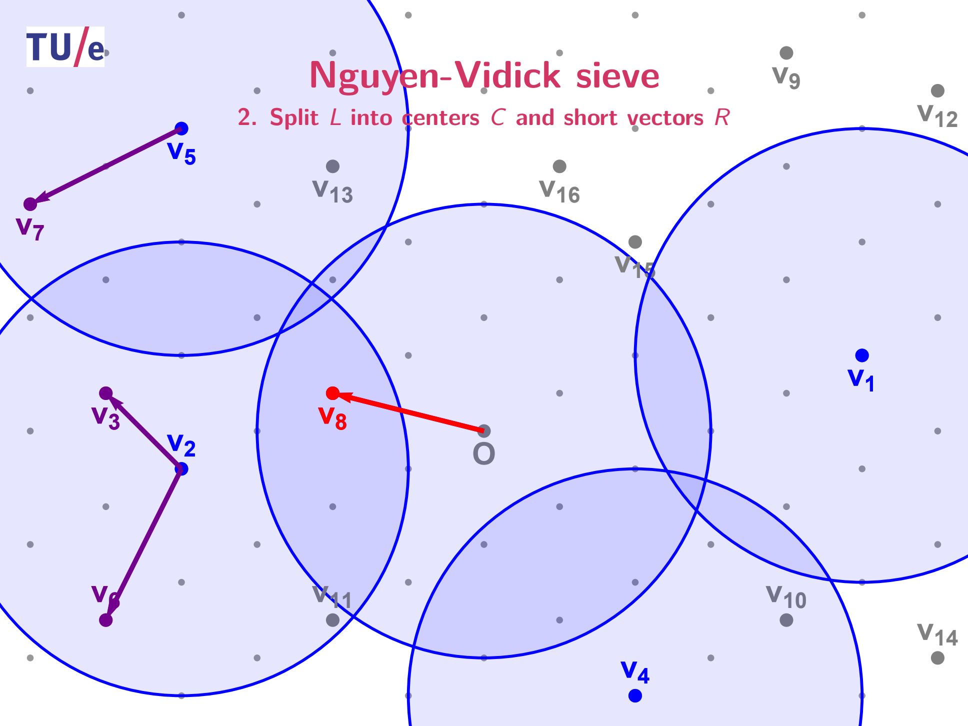
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



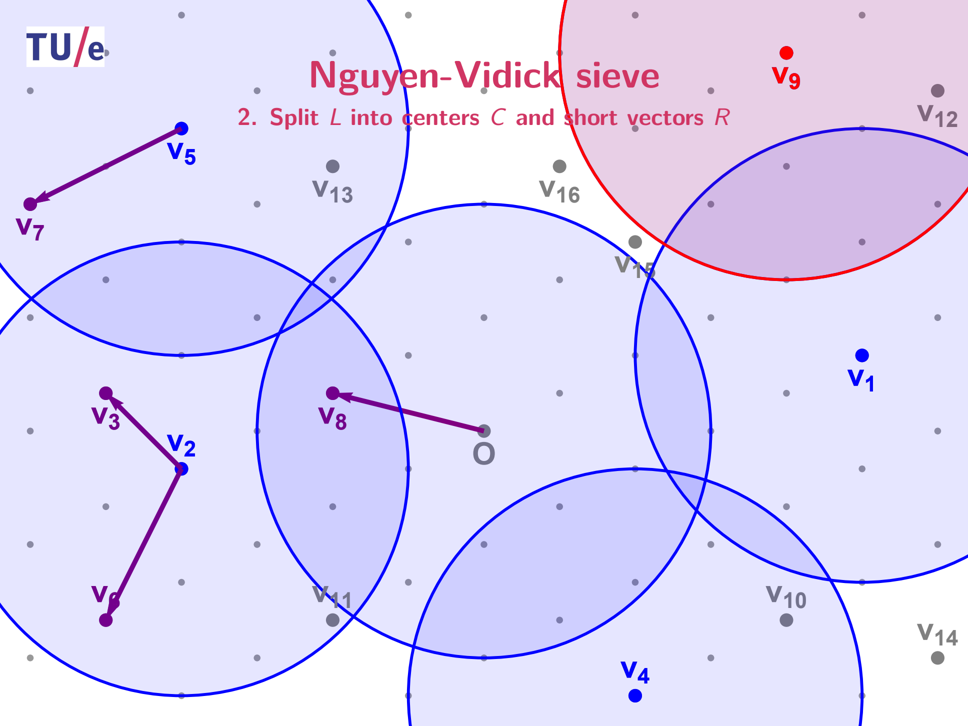
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



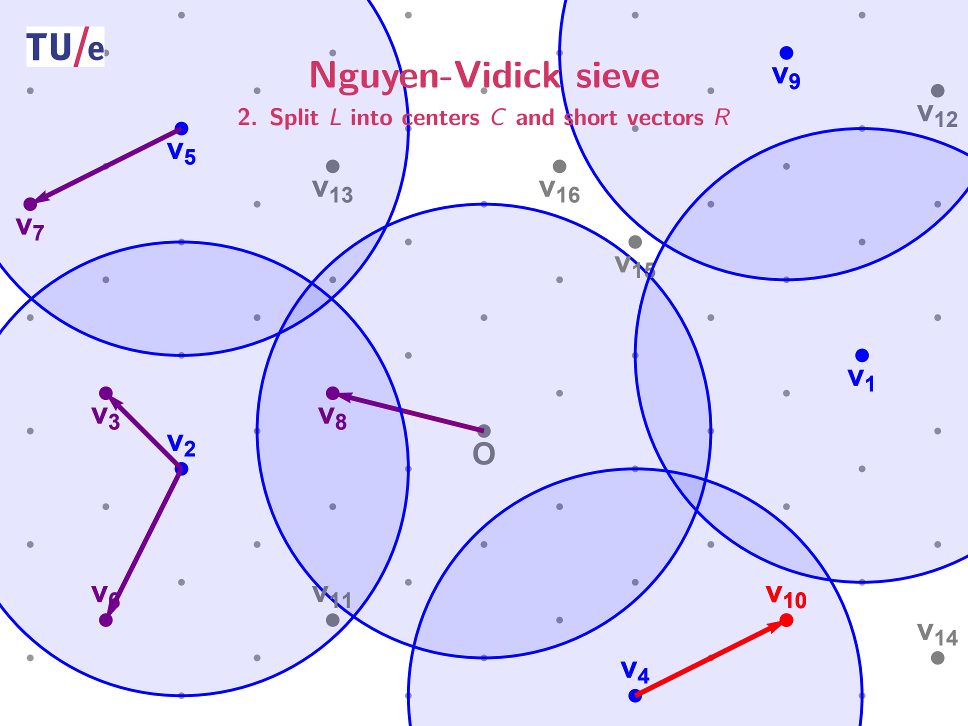
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



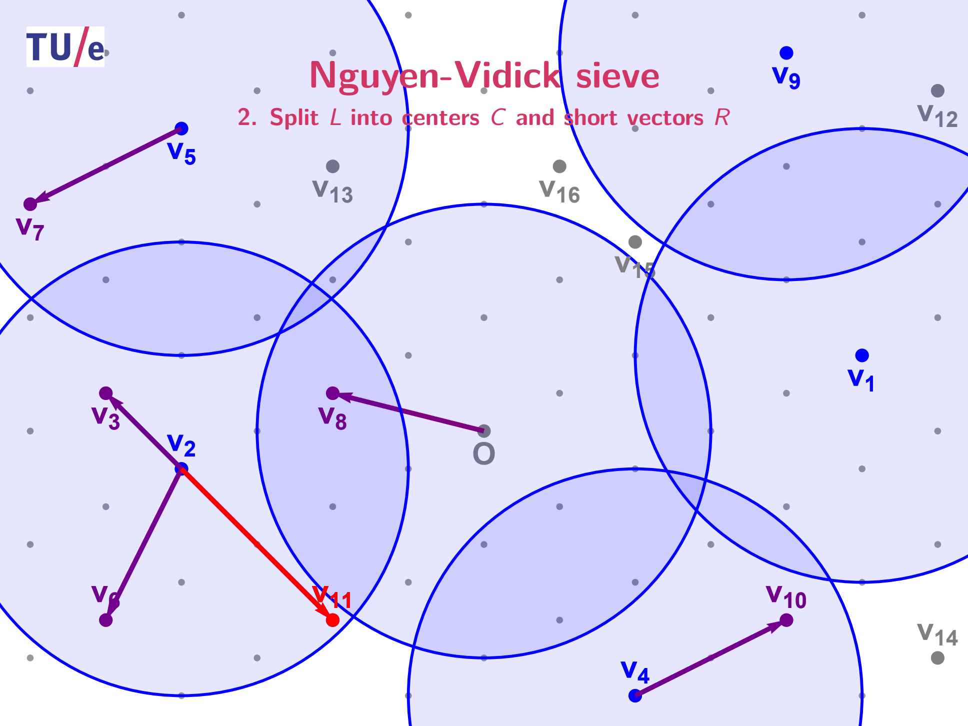
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



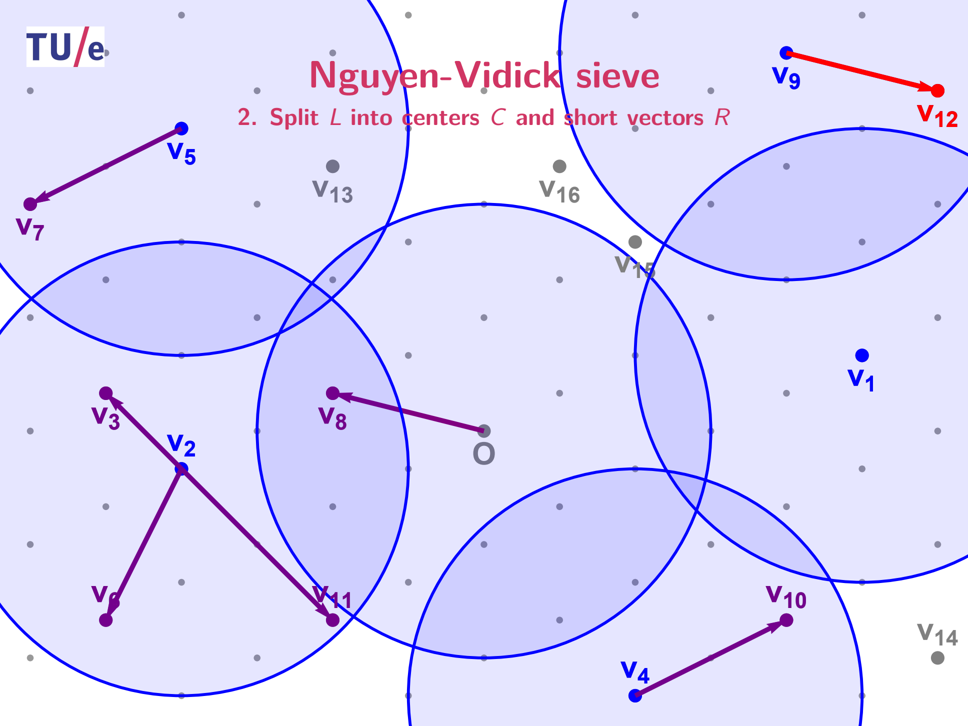
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



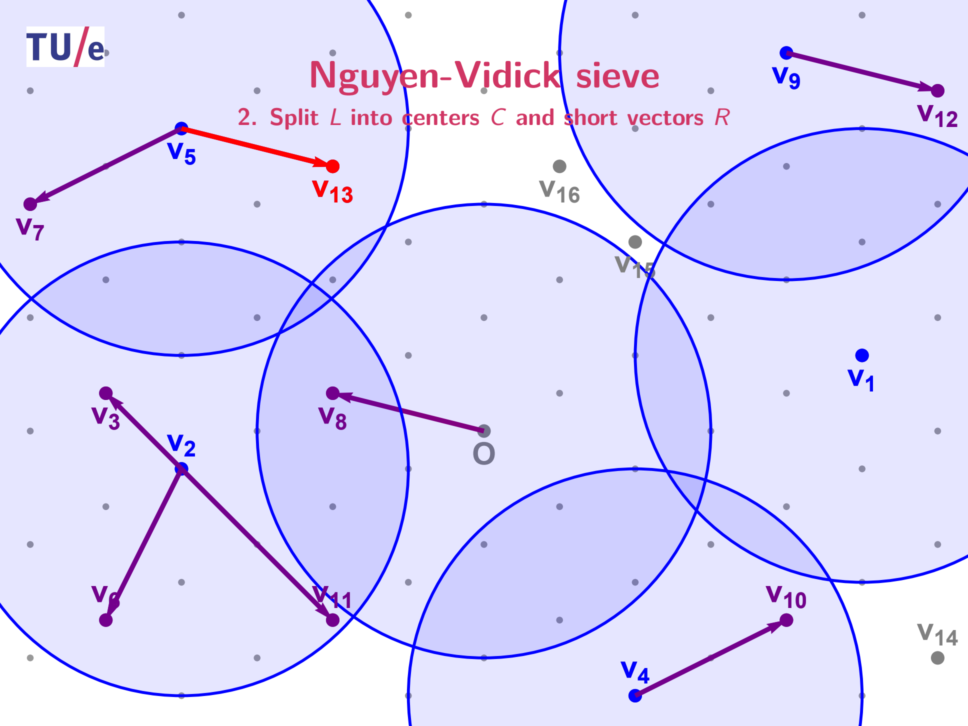
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



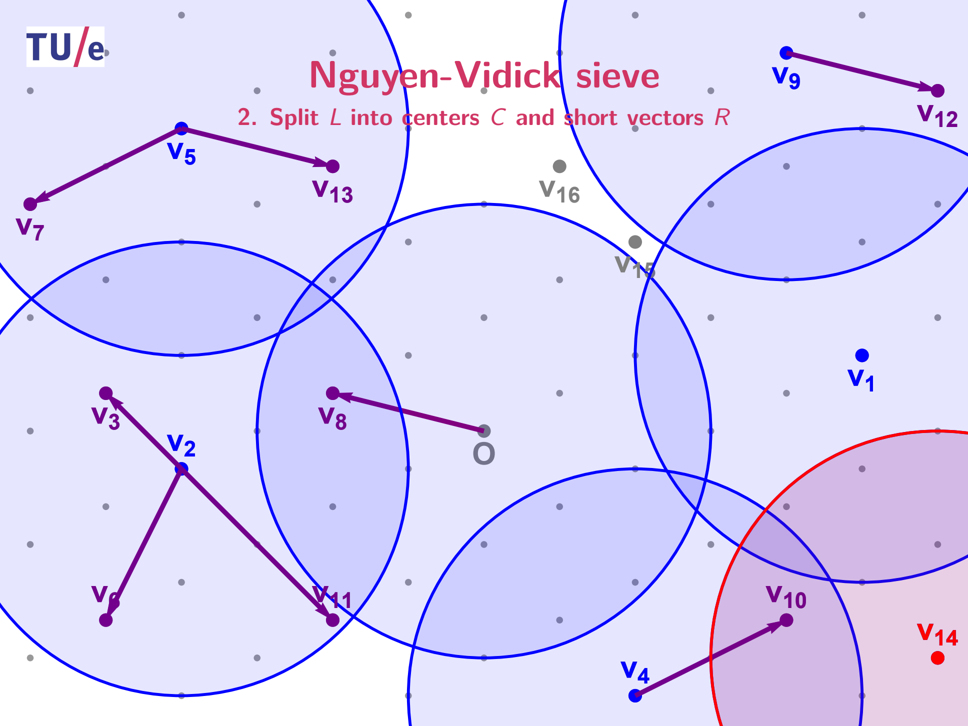
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



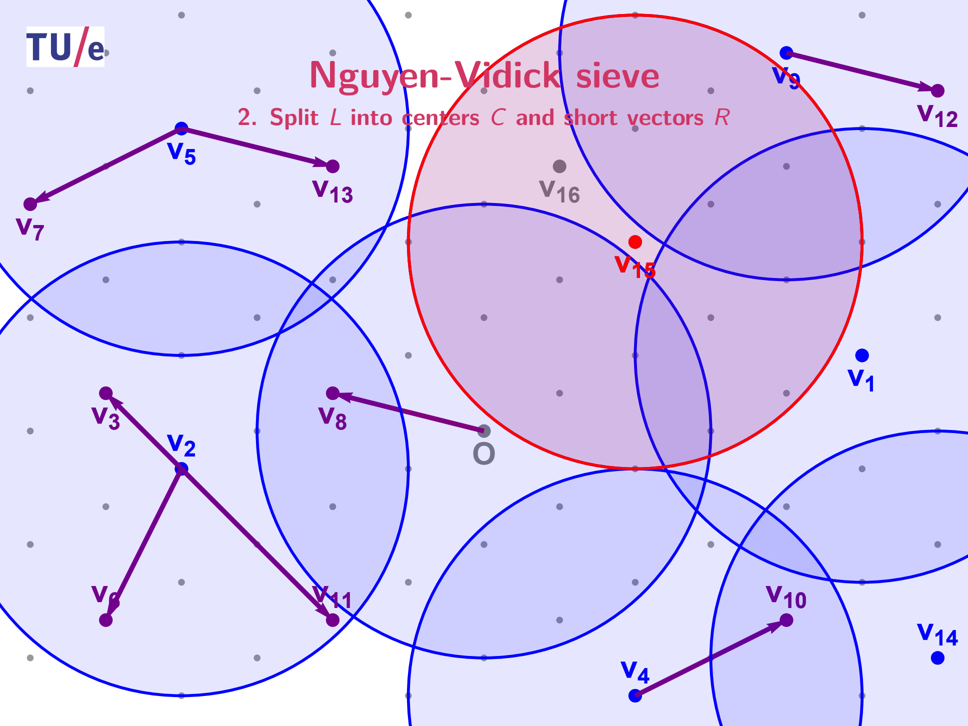
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



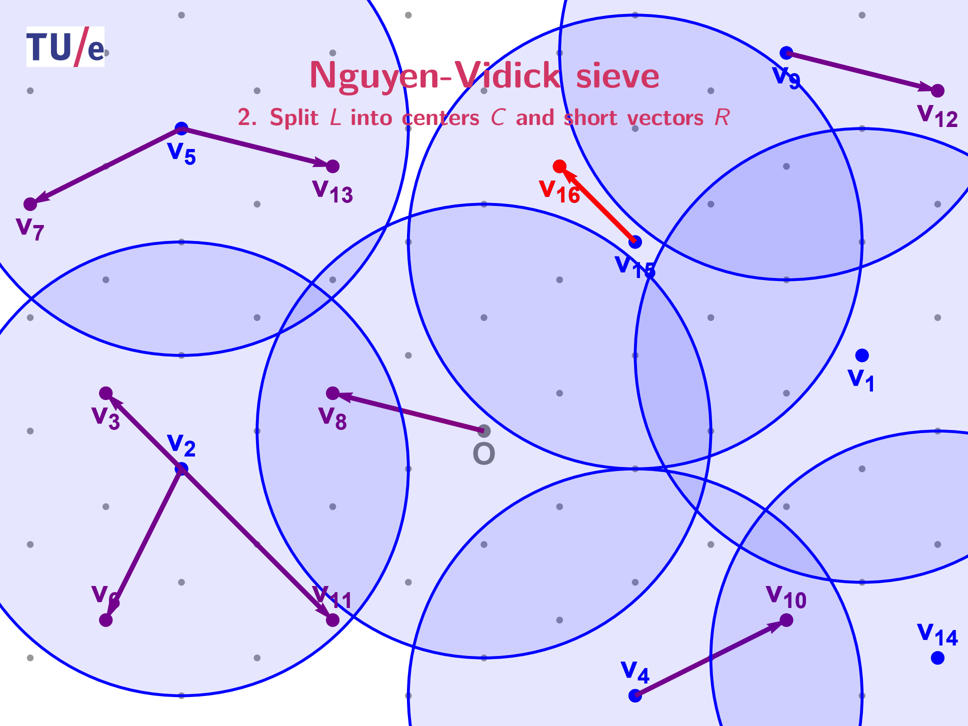
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



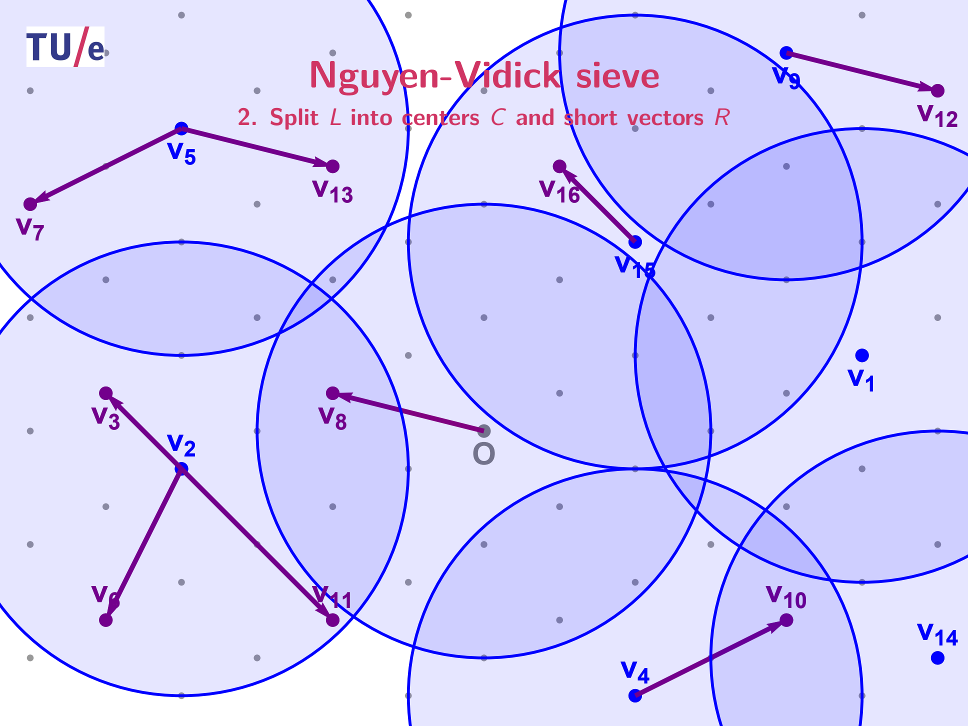
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



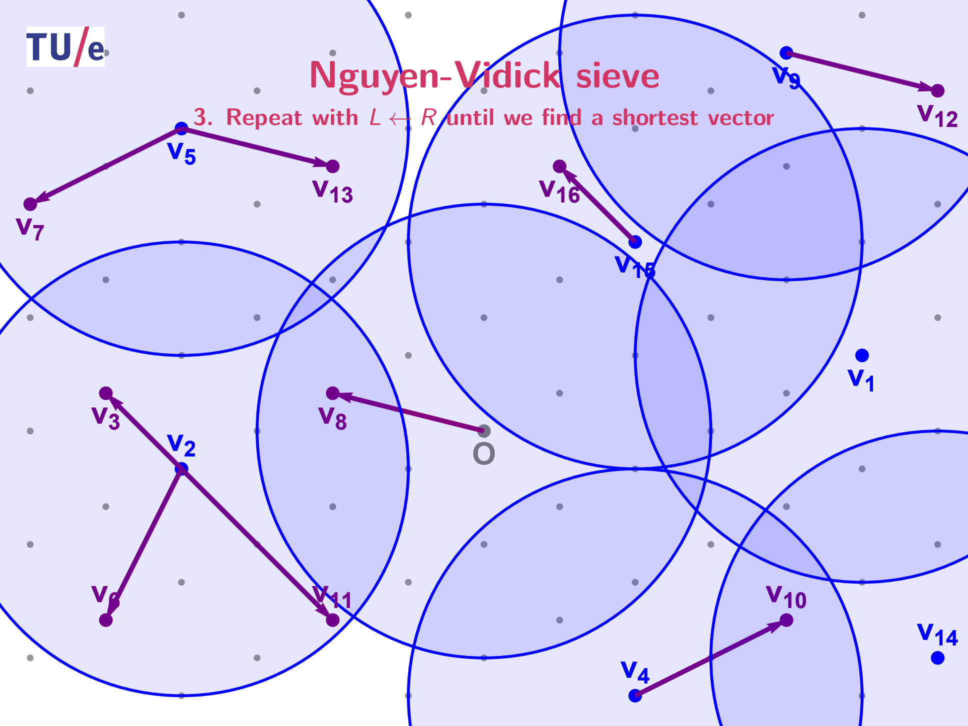
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



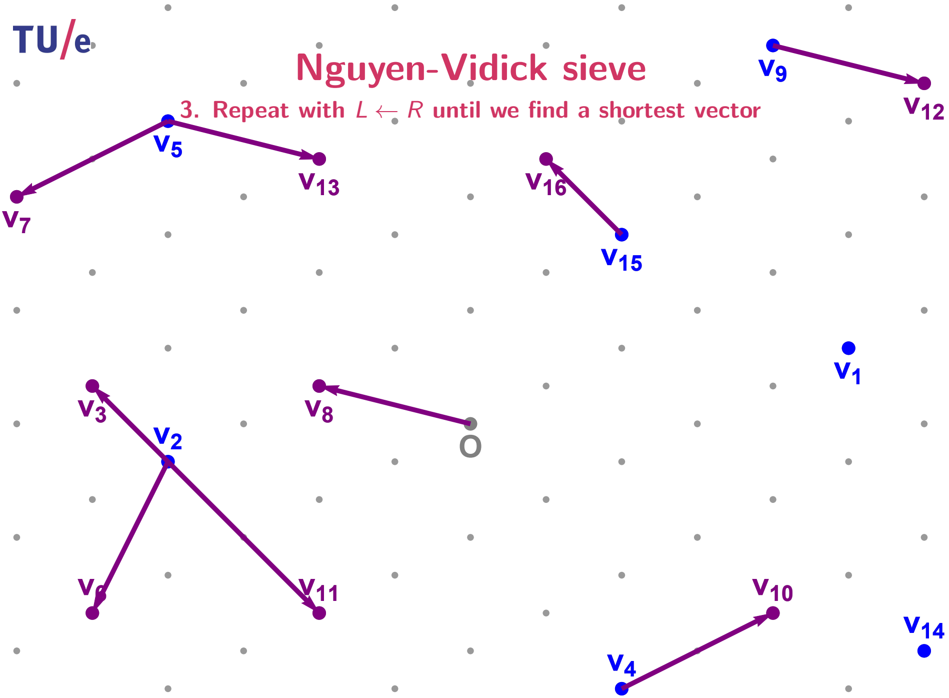
Nguyen-Vidick sieve

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



Nguyen-Vidick sieve

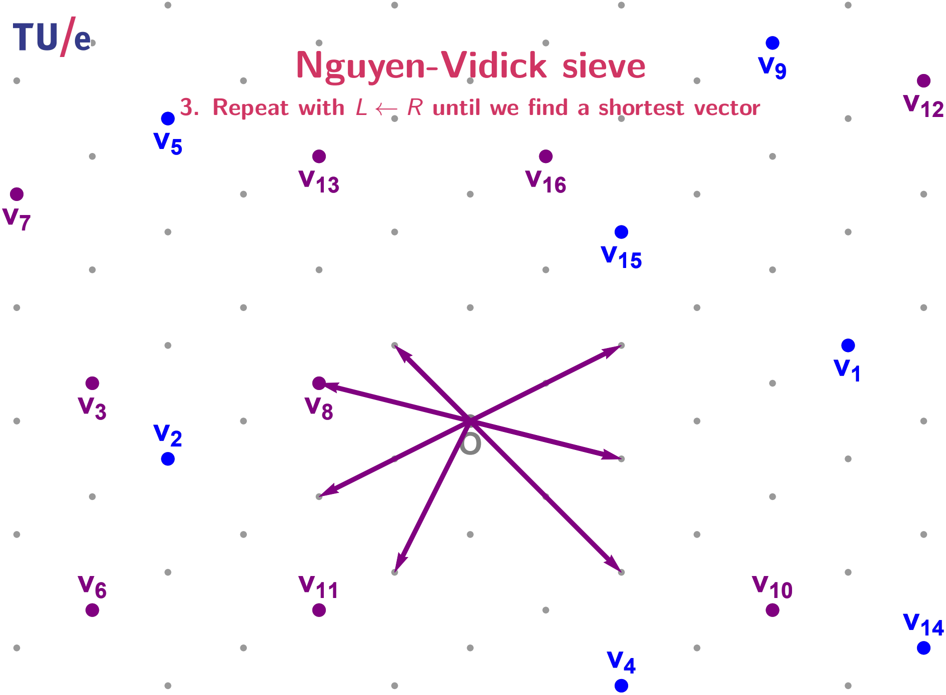
3. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

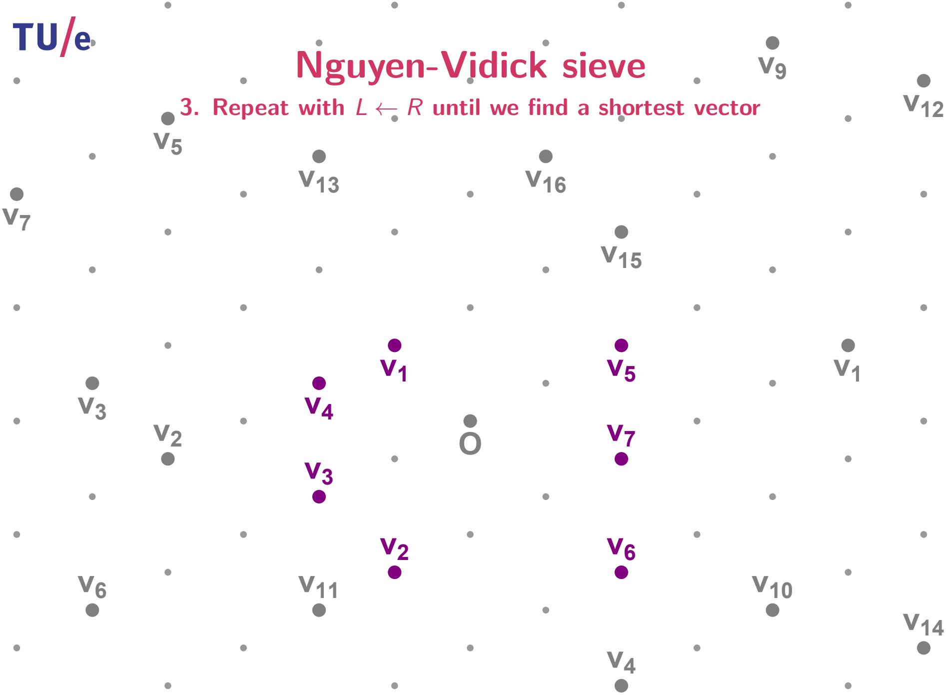
Nguyen-Vidick sieve

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

- Space complexity: $\sqrt{4/3}^n \approx 2^{0.21n+o(n)}$ vectors
  - Need $\sqrt{4/3}^n$ vectors to cover all corners of $\mathbb{R}^n$

Nguyen-Vidick sieve

Overview

- Space complexity: $\sqrt{4/3}^n \approx 2^{0.21n+o(n)}$ vectors
  - Need $\sqrt{4/3}^n$ vectors to cover all corners of $\mathbb{R}^n$
- Time complexity: $(4/3)^n \approx 2^{0.42n+o(n)}$
  - Comparing a target vector to all centers: $2^{0.21n+o(n)}$
  - Repeating this for each list vector: $2^{0.21n+o(n)}$
  - Repeating the whole sieving procedure: $\mathrm{poly}(n)$

Nguyen-Vidick sieve

Overview

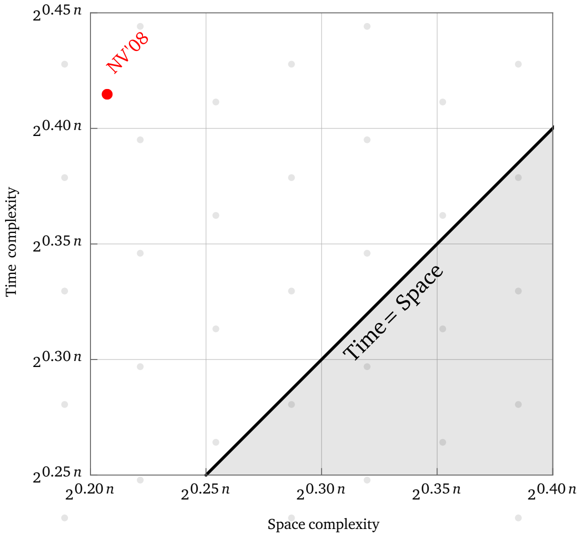
- Space complexity: $\sqrt{4/3}^n \approx 2^{0.21n+o(n)}$ vectors
  - Need $\sqrt{4/3}^n$ vectors to cover all corners of $\mathbb{R}^n$
- Time complexity: $(4/3)^n \approx 2^{0.42n+o(n)}$
  - Comparing a target vector to all centers: $2^{0.21n+o(n)}$
  - Repeating this for each list vector: $2^{0.21n+o(n)}$
  - Repeating the whole sieving procedure: $\text{poly}(n)$

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

The NV-sieve runs in time $2^{0.42n+o(n)}$ and space $2^{0.21n+o(n)}$.

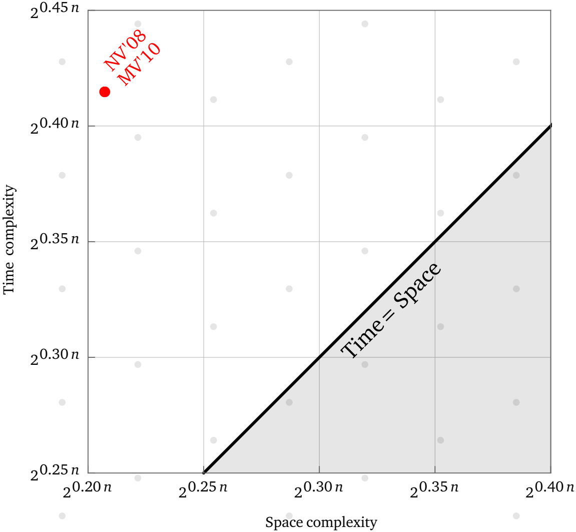
Nguyen-Vidick sieve

Space/time trade-off



GaussSieve

Space/time trade-off

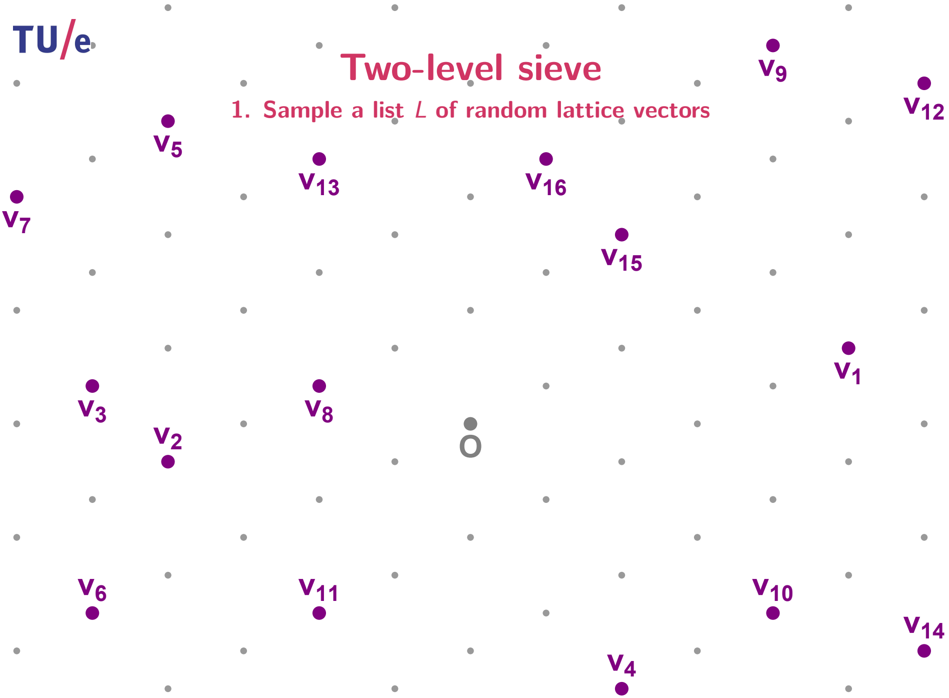


Two-level sieve

1. Sample a list L of random lattice vectors

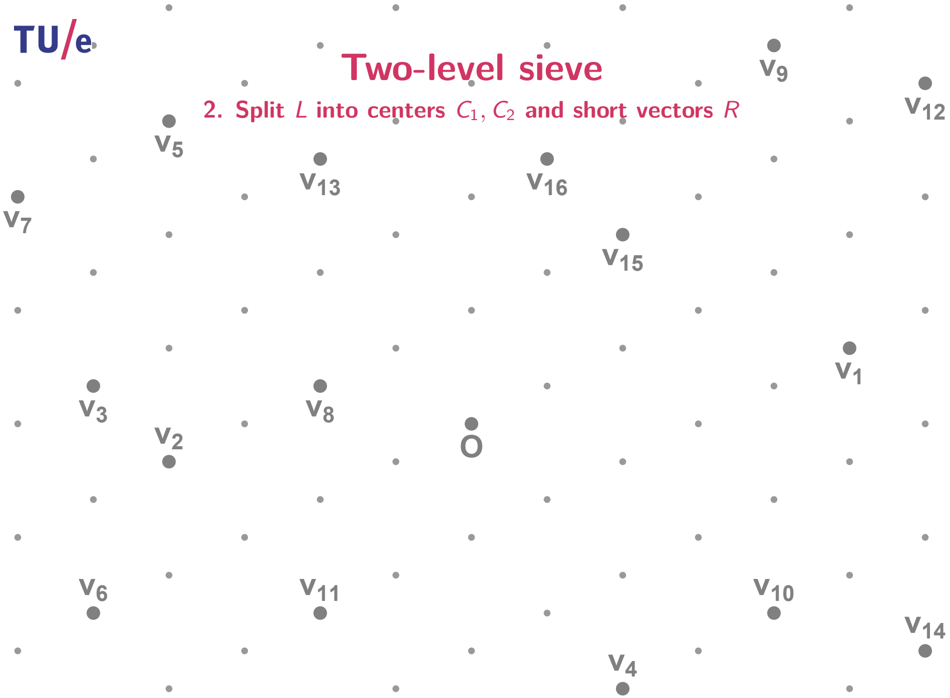
Two-level sieve

1. Sample a list L of random lattice vectors



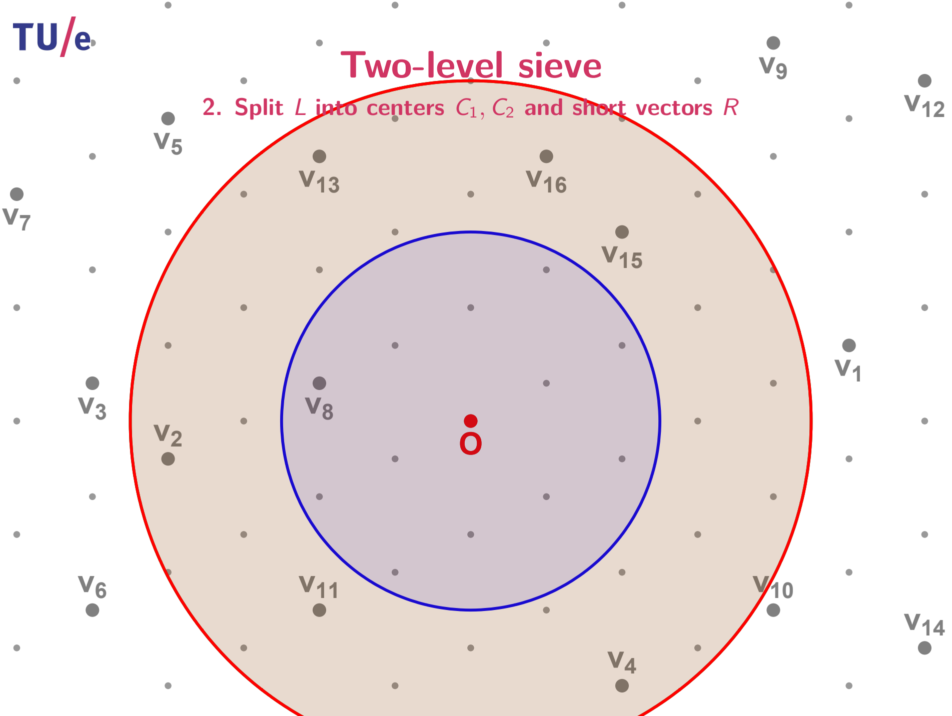
Two-level sieve

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



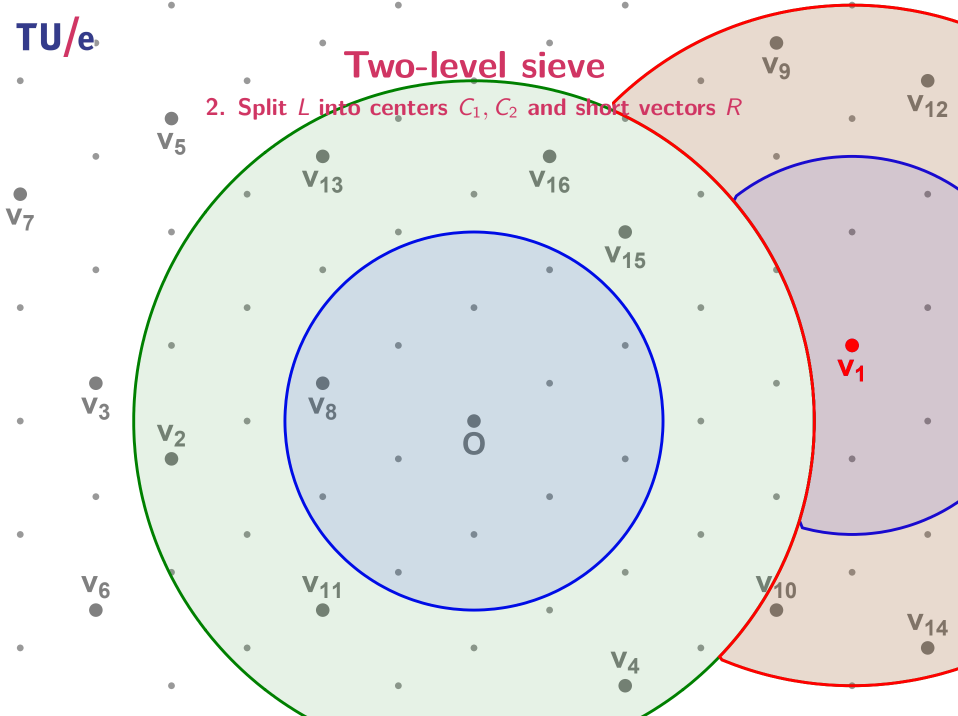
Two-level sieve

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



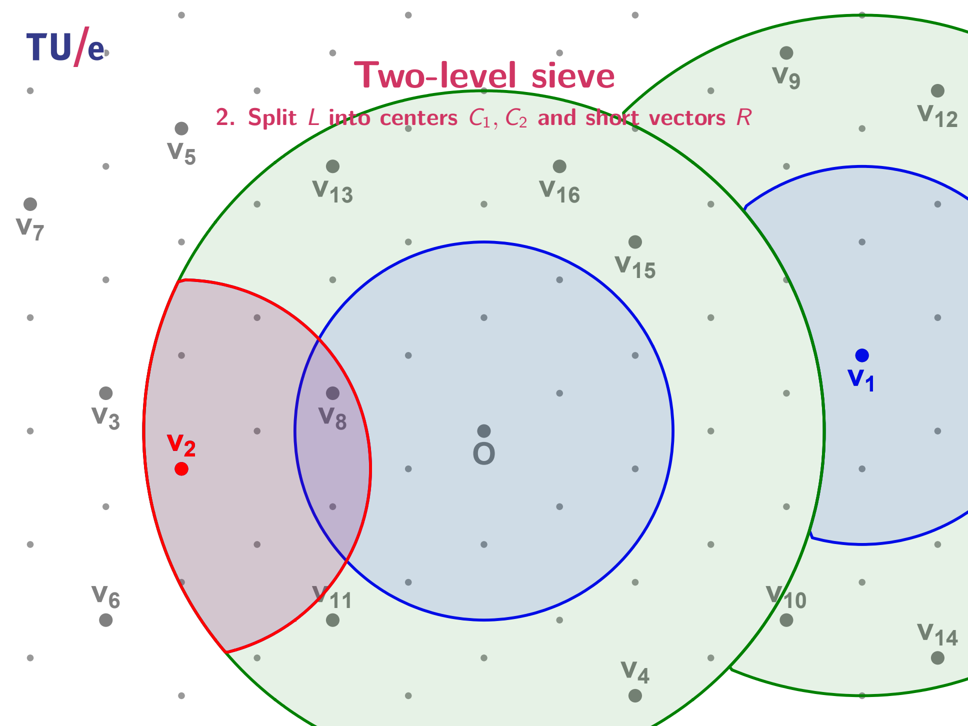
Two-level sieve

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



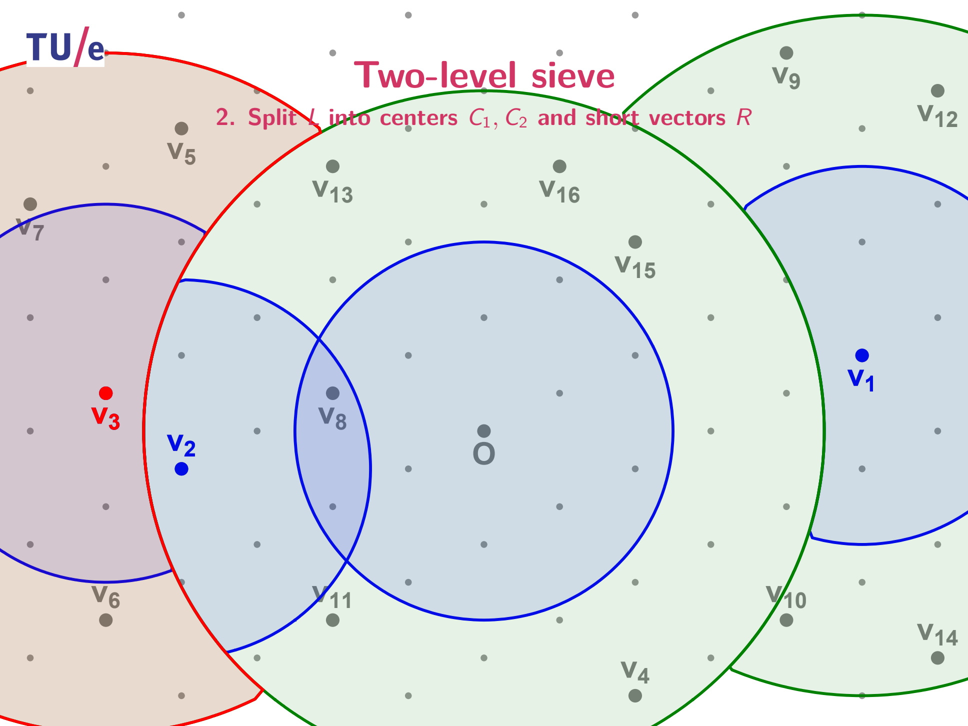
Two-level sieve

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



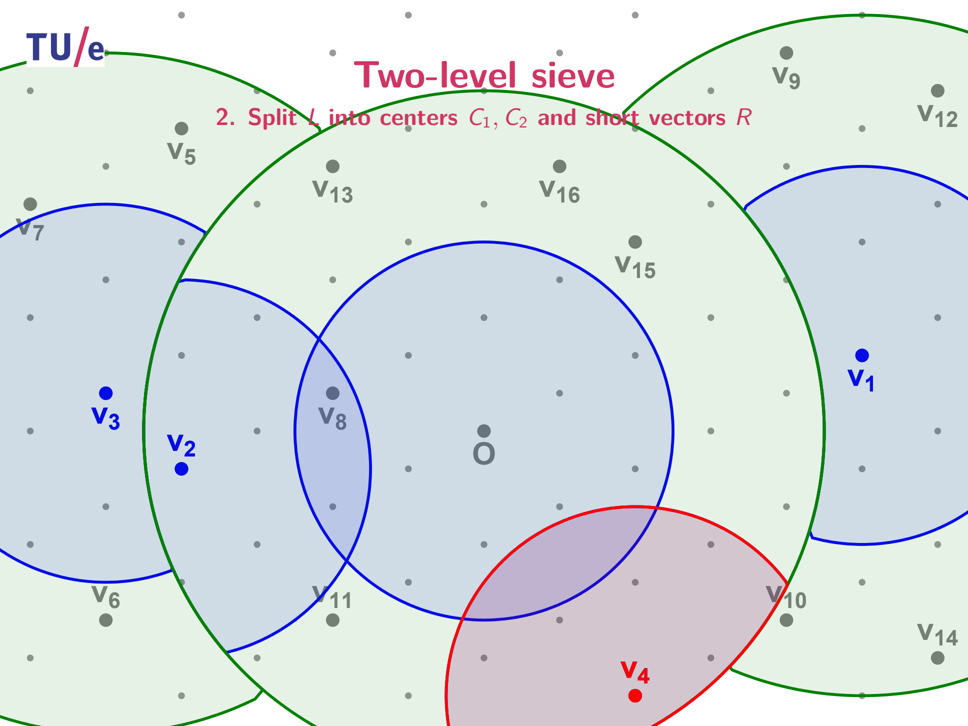
Two-level sieve

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



Two-level sieve

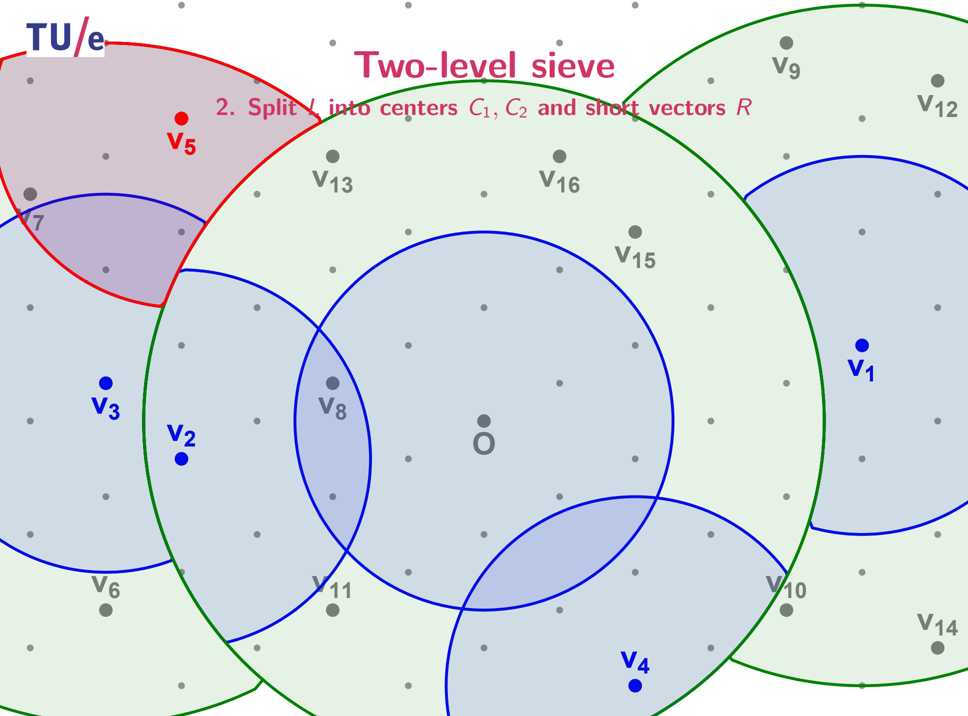
2. Split L into centers C_1, C_2 and short vectors R



TU/e

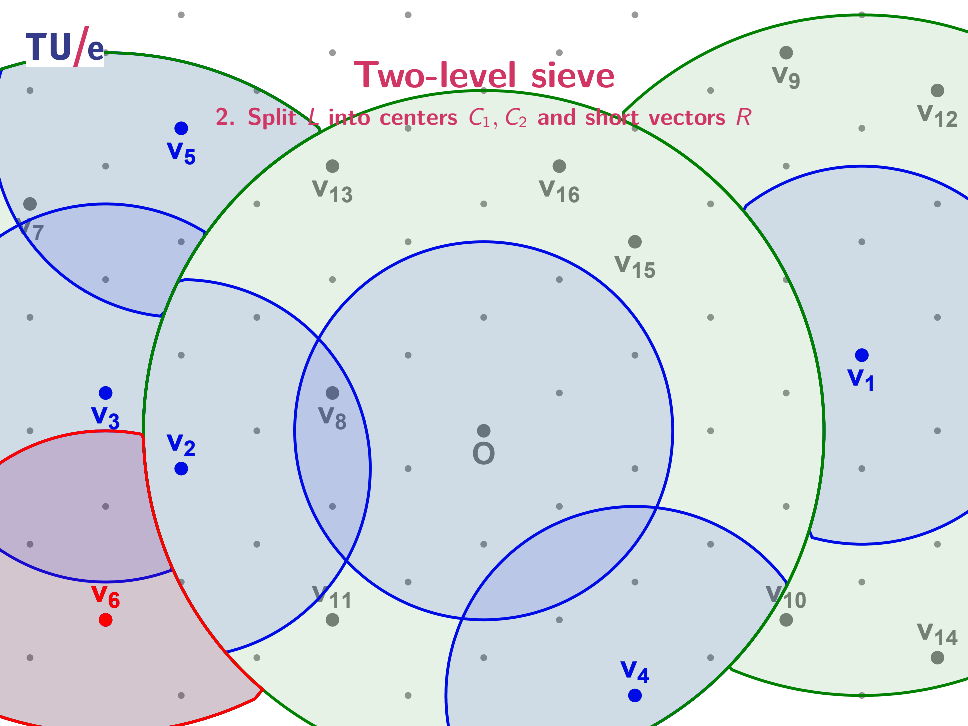
Two-level sieve

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



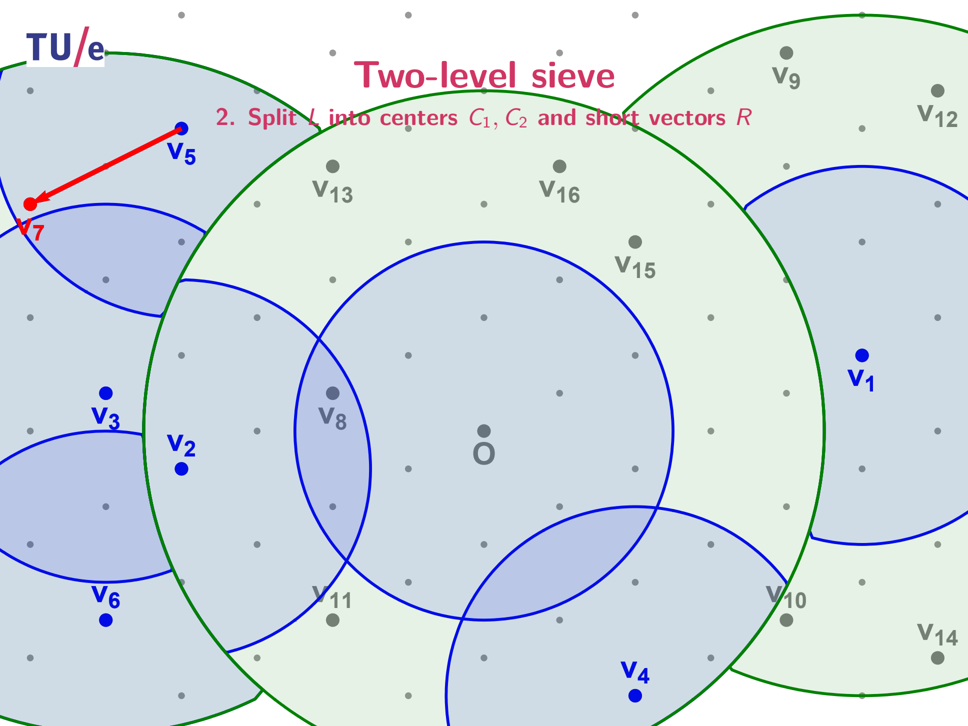
Two-level sieve

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



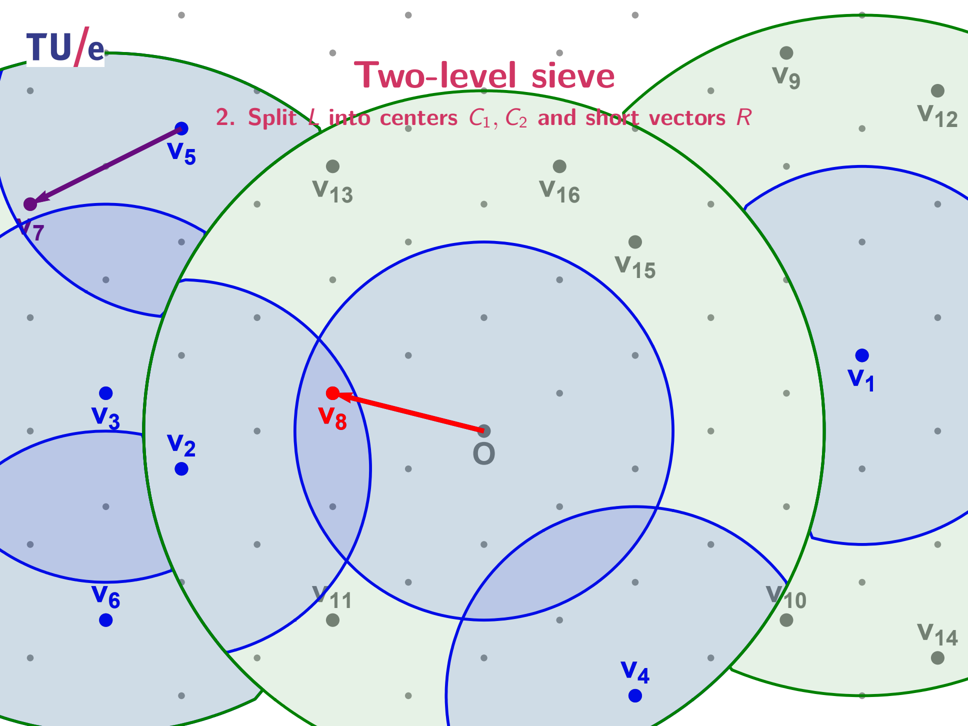
Two-level sieve

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



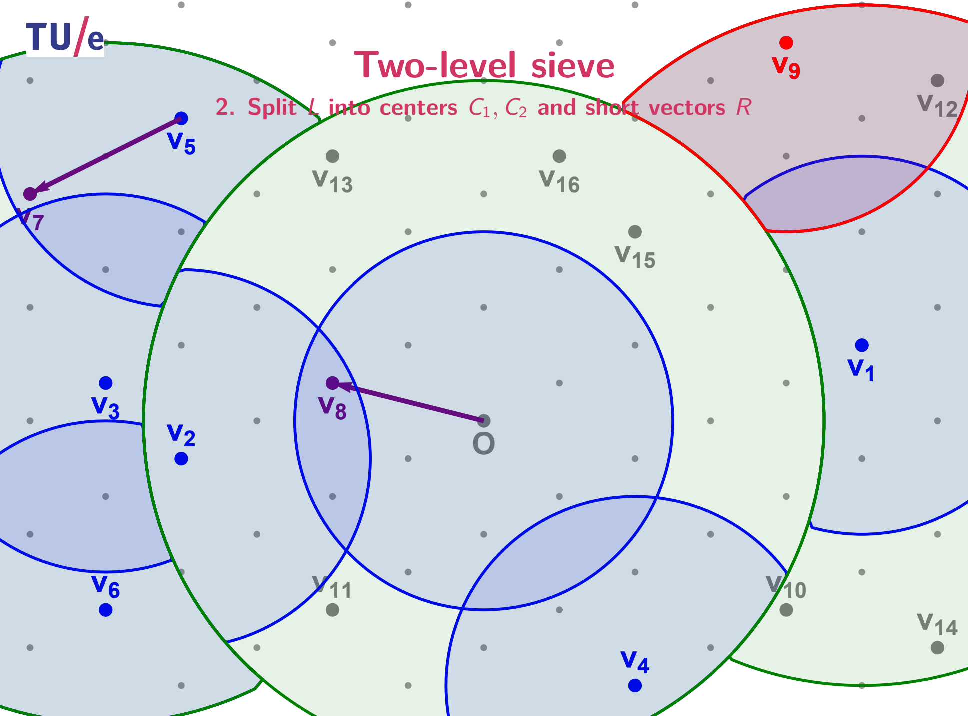
Two-level sieve

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



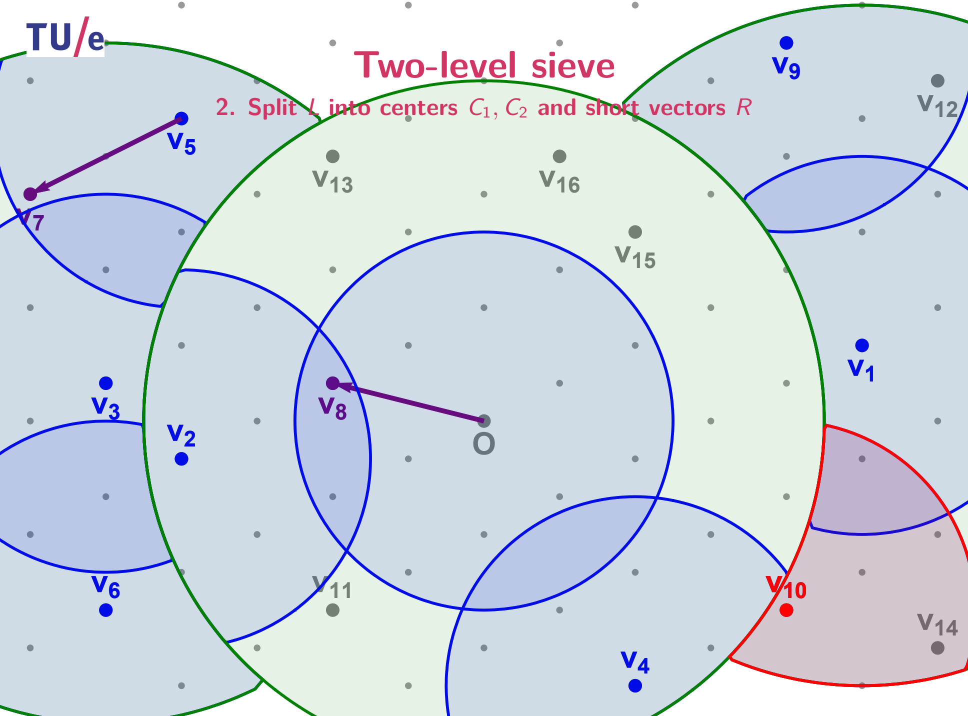
Two-level sieve

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



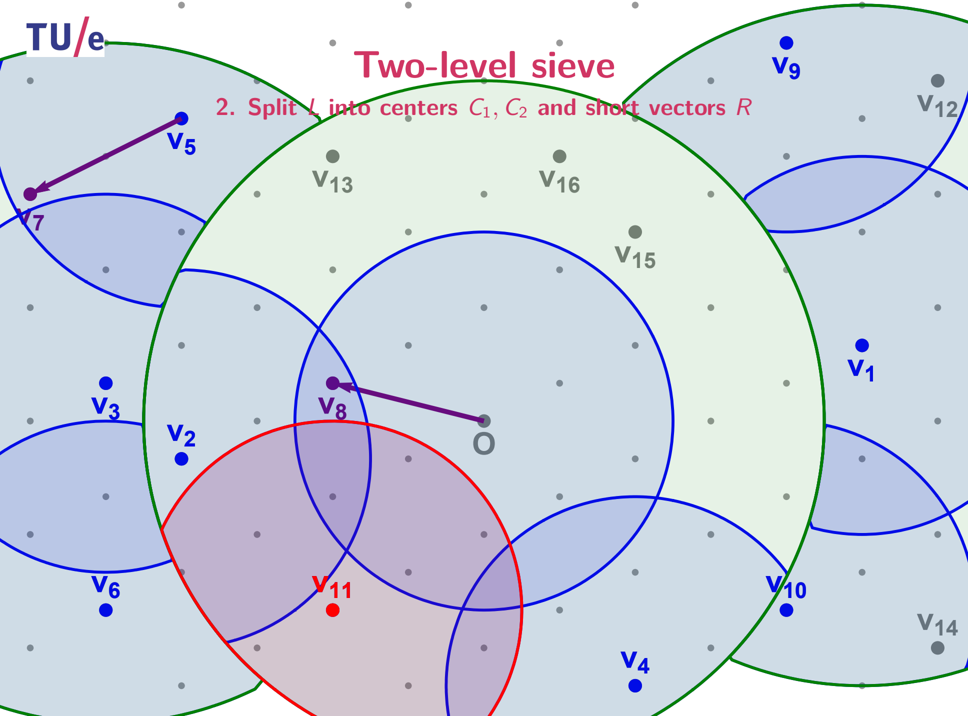
Two-level sieve

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



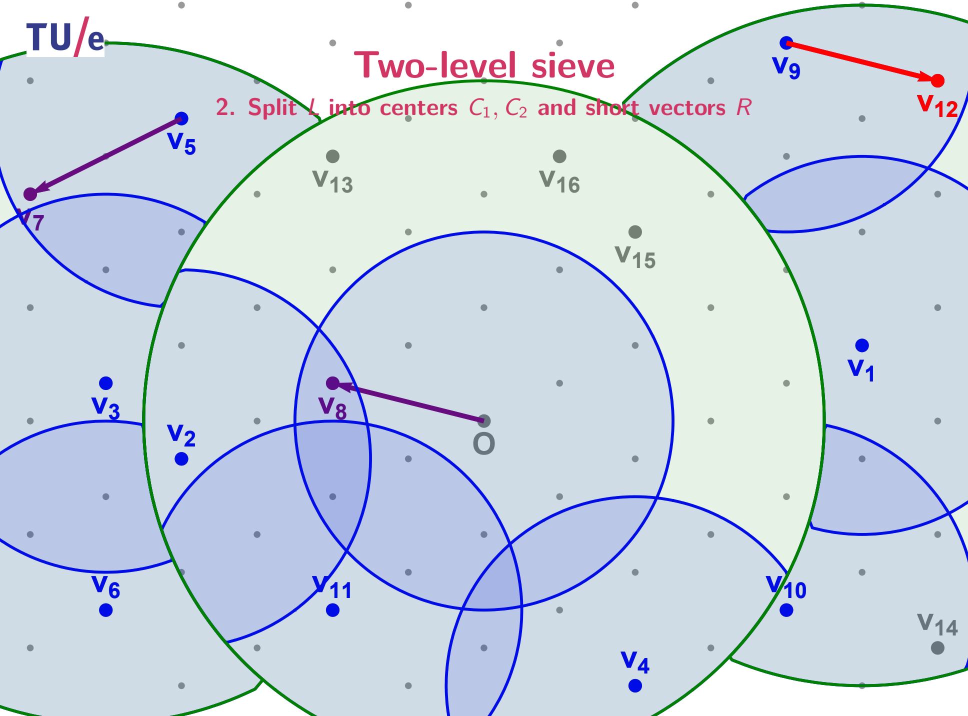
Two-level sieve

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



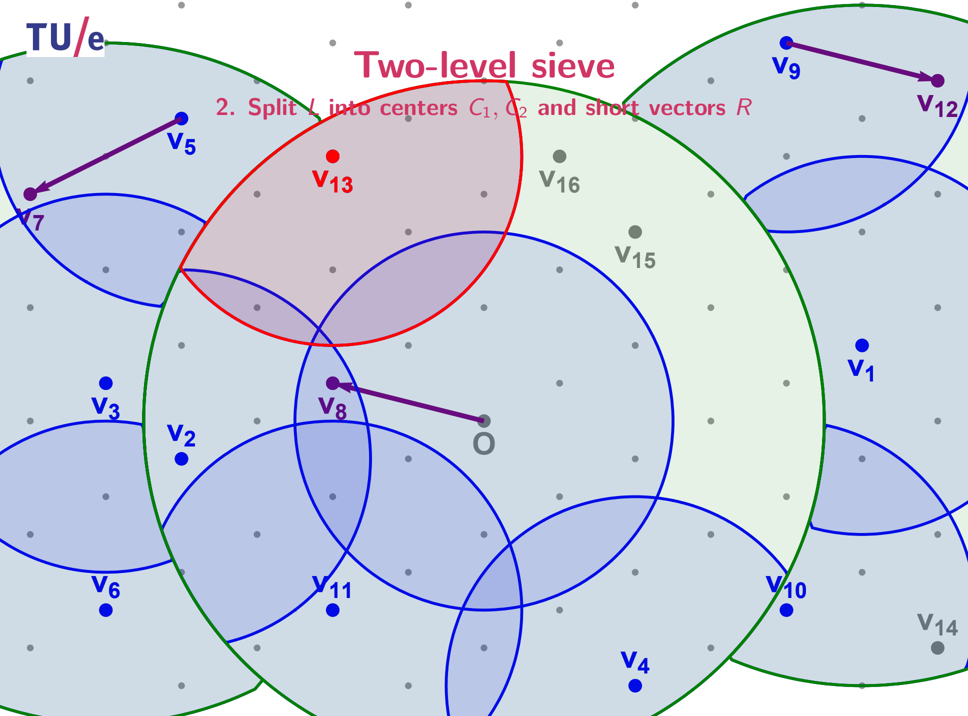
Two-level sieve

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



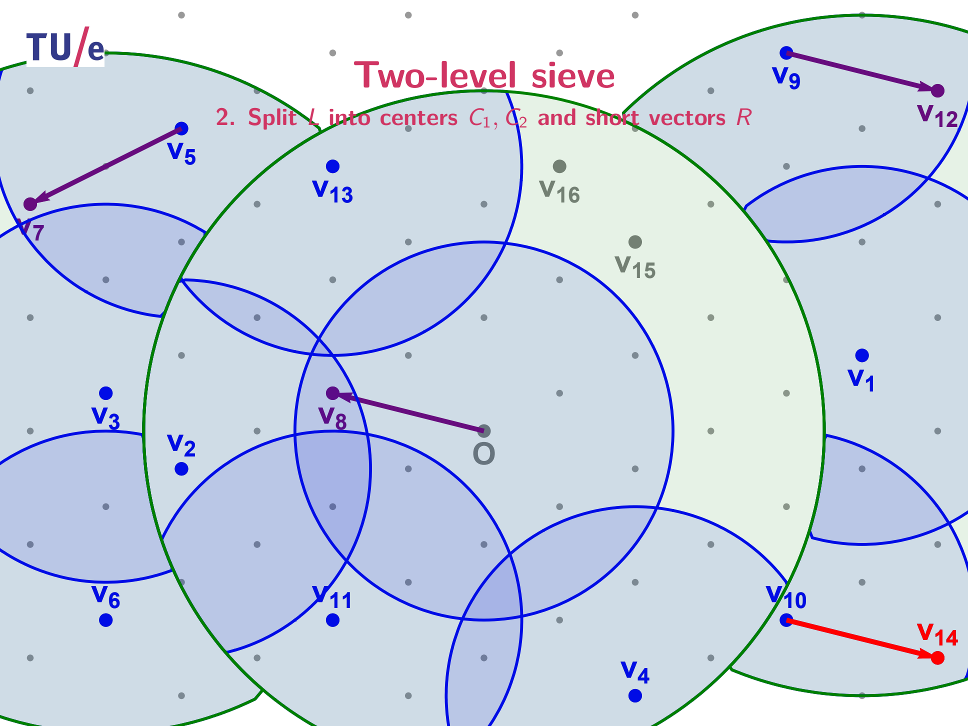
Two-level sieve

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



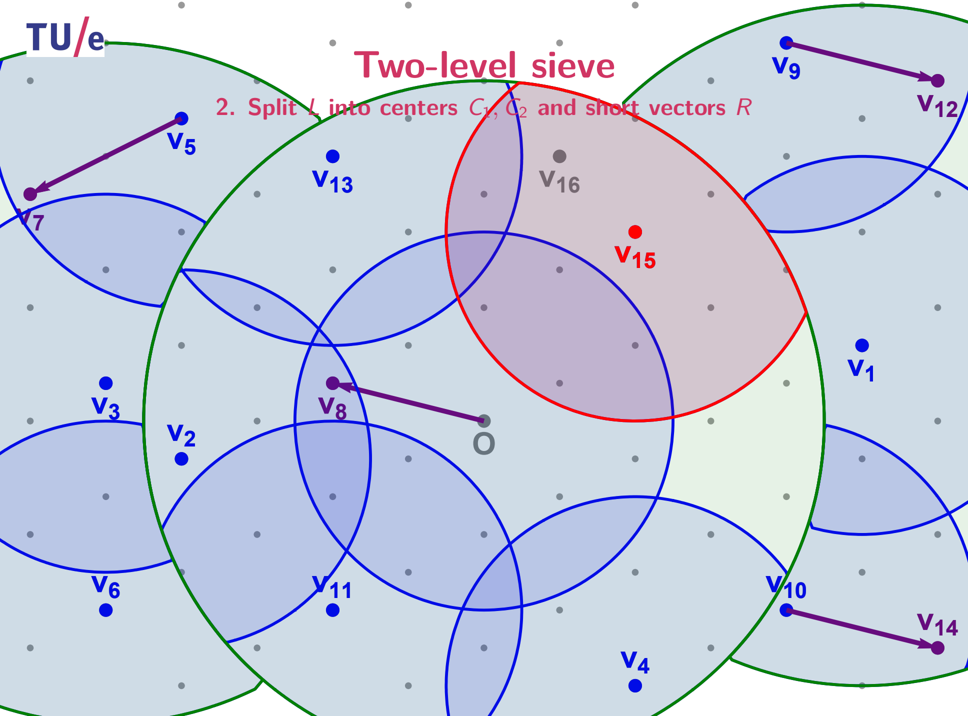
Two-level sieve

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



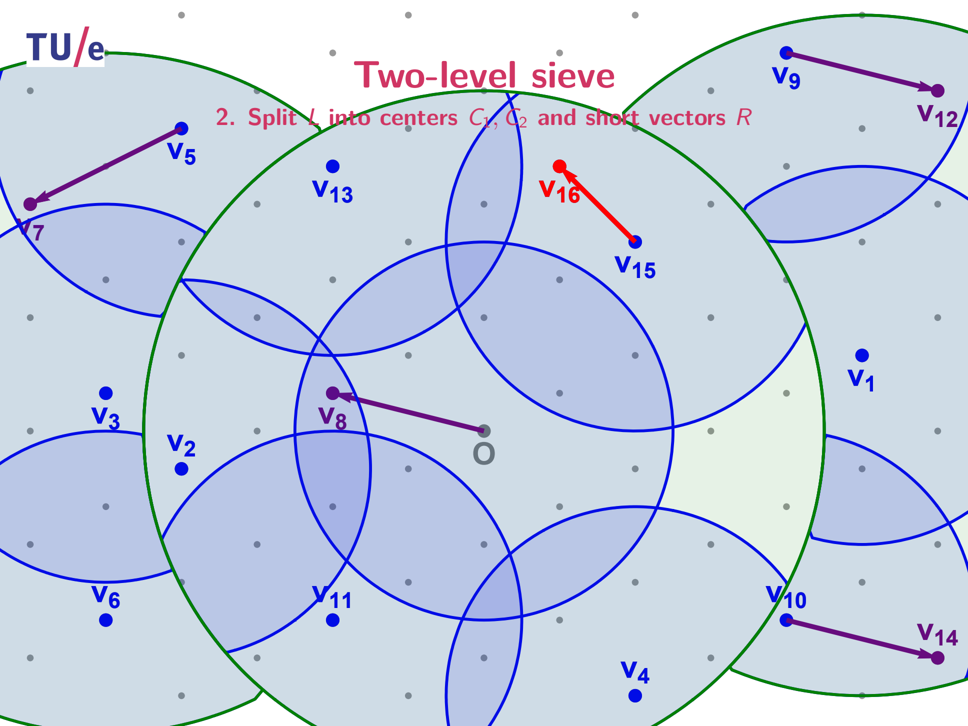
Two-level sieve

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



Two-level sieve

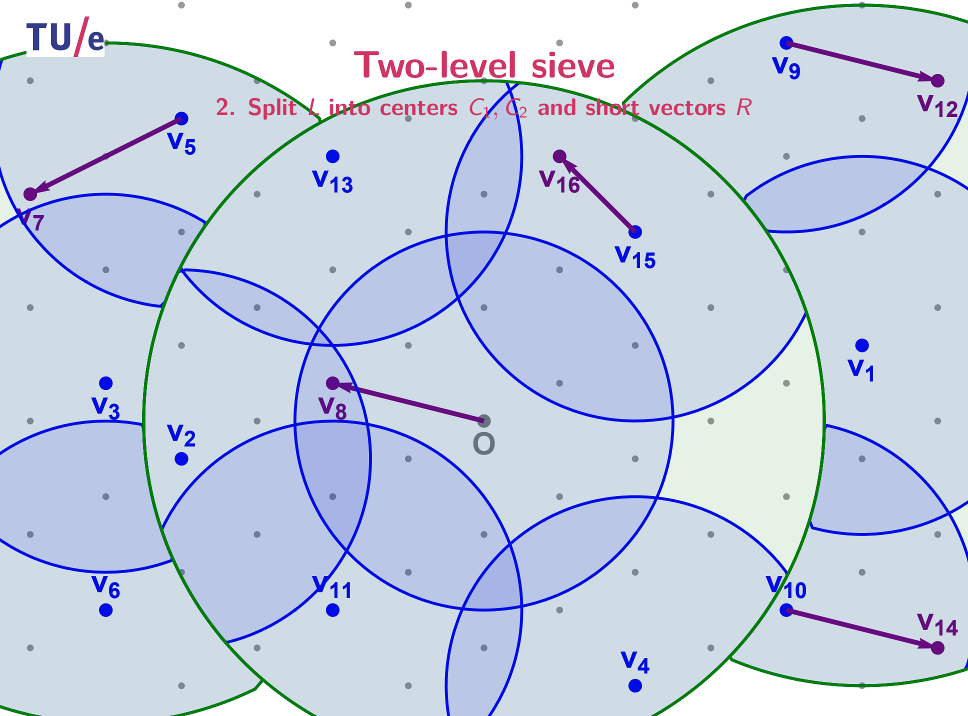
2. Split L into centers C_1, C_2 and short vectors R



TU/e

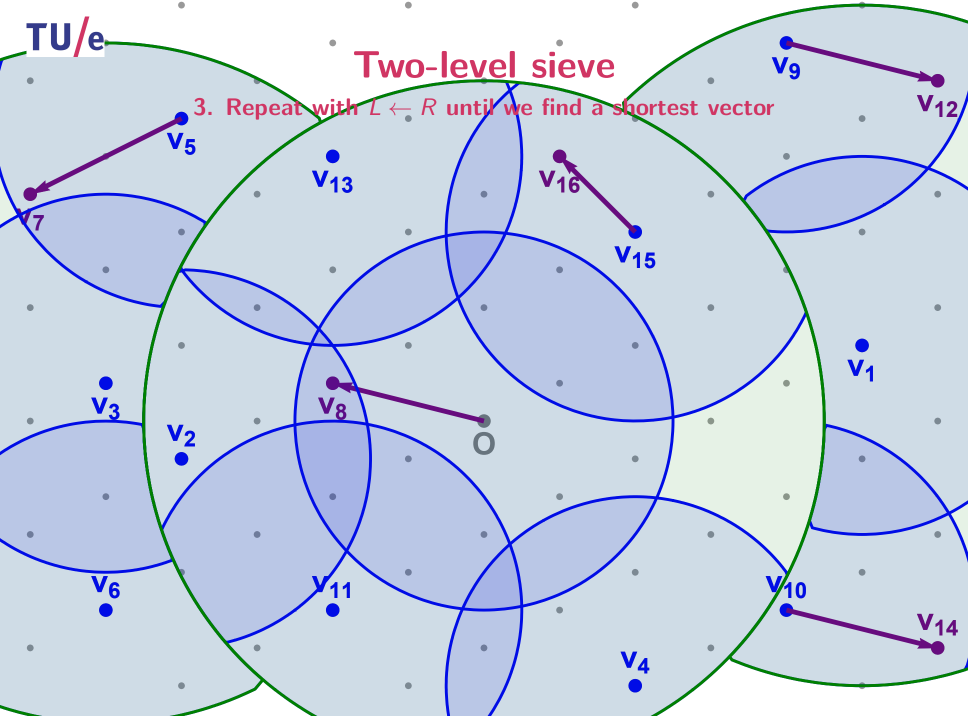
Two-level sieve

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



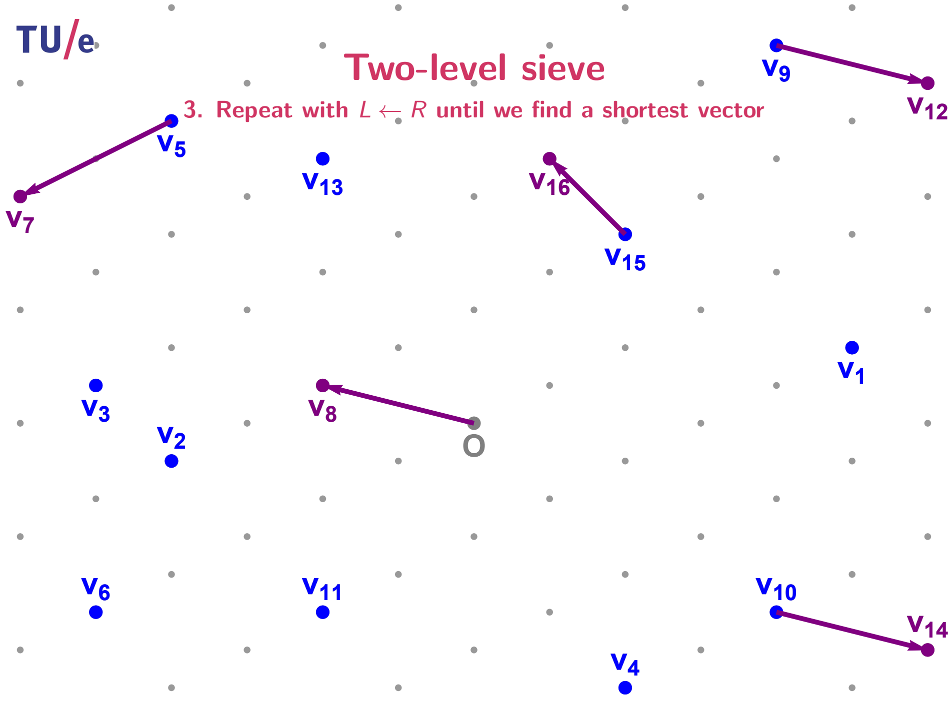
Two-level sieve

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



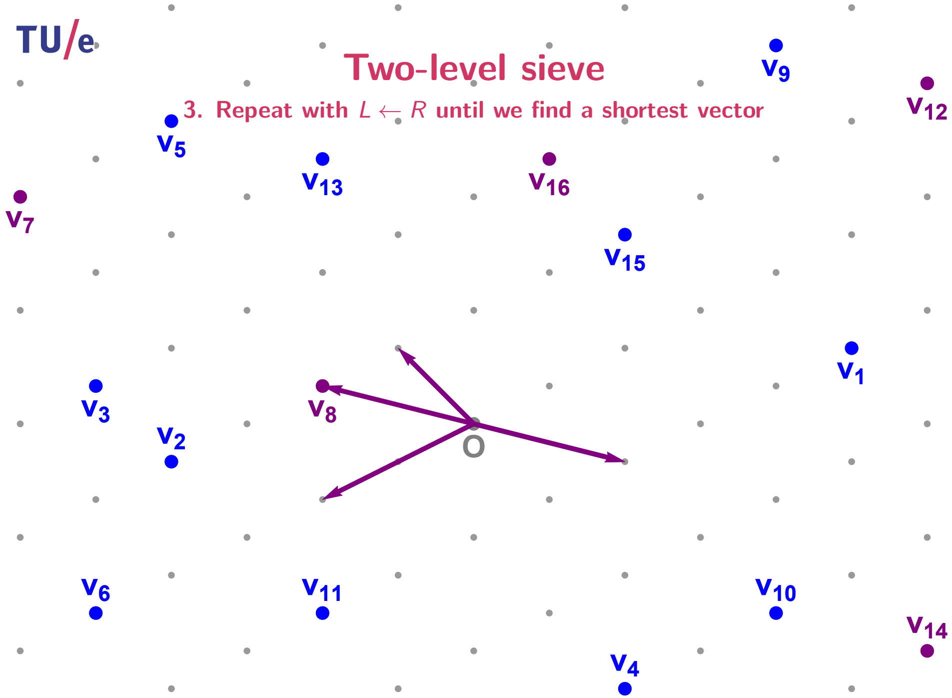
Two-level sieve

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



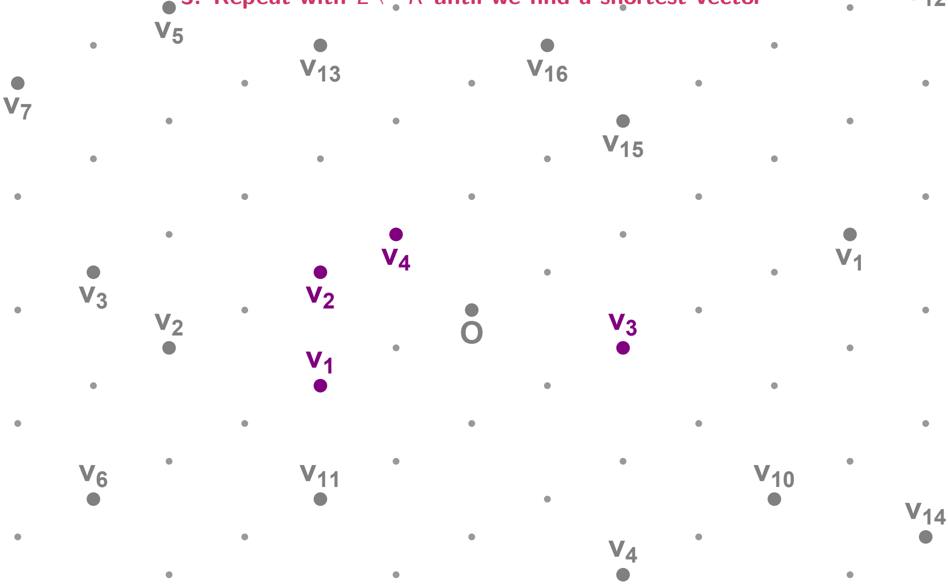
Two-level sieve

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



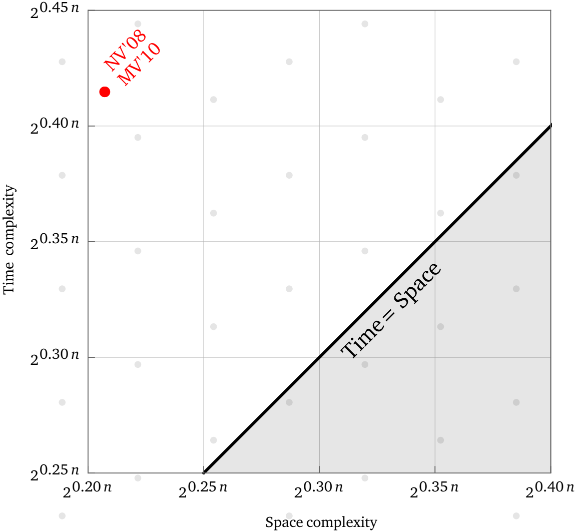
Two-level sieve

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



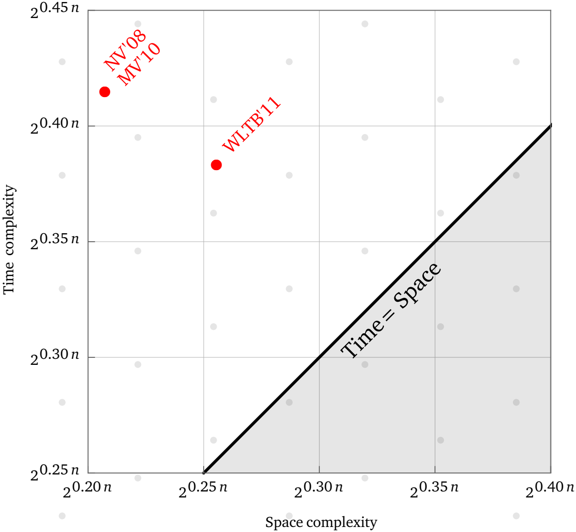
Two-level sieve

Space/time trade-off



Two-level sieve

Space/time trade-off



Three-level sieve

Overview

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

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

Three-level sieve

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 $2^{0.2557n}$.

Three-level sieve

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 $2^{0.2833n}$.

Three-level sieve

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)

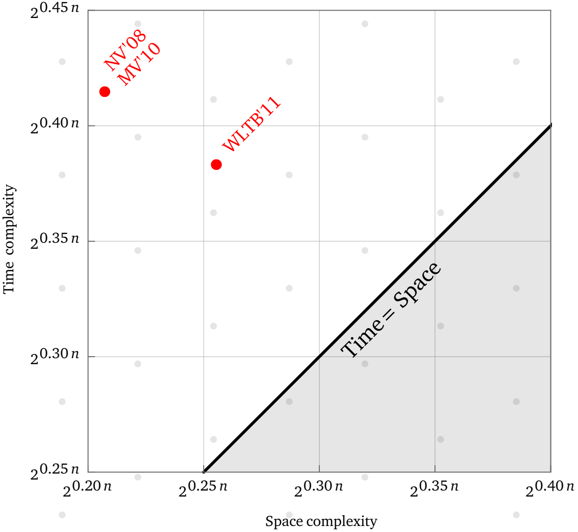
The three-level sieve runs in time $2^{0.3778n}$ and space $2^{0.2833n}$.

Conjecture

The four-level sieve runs in time $2^{0.3774n}$ and space $2^{0.2925n}$, and higher-level sieves are not faster than this.

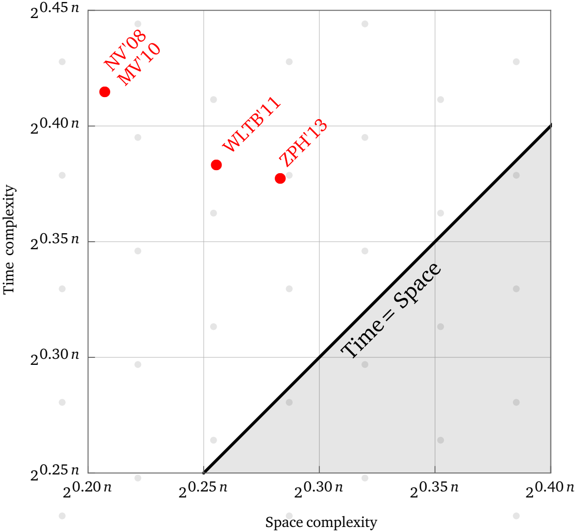
Three-level sieve

Space/time trade-off



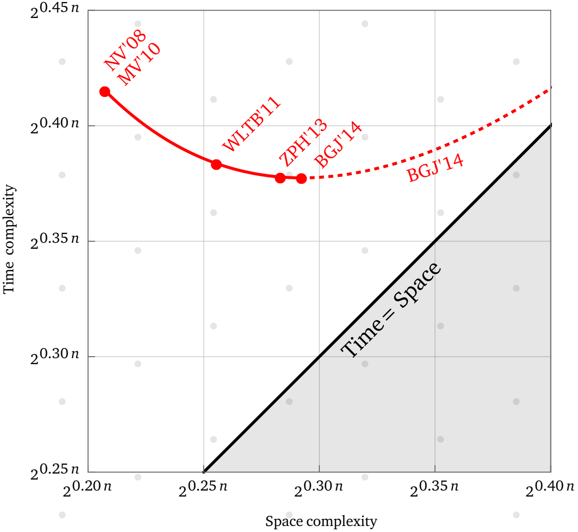
Three-level sieve

Space/time trade-off



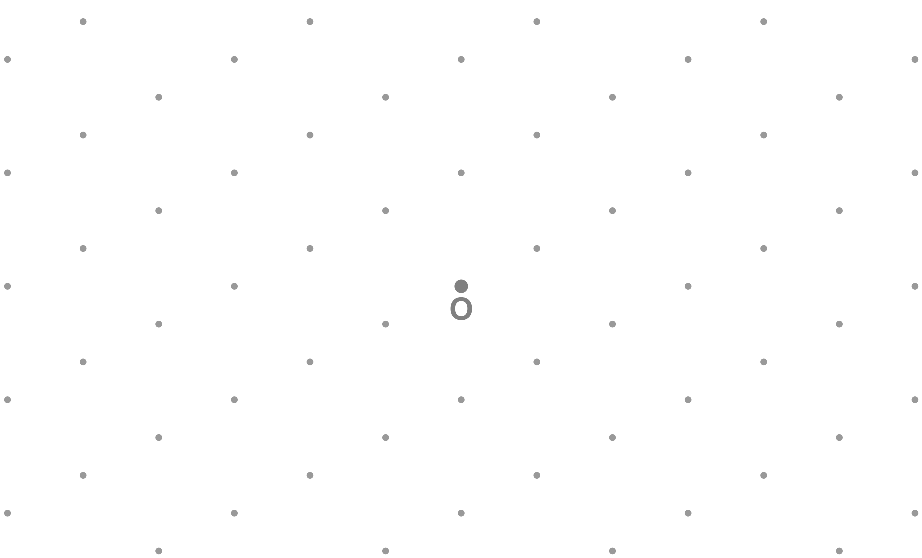
Decomposition approach

Space/time trade-off



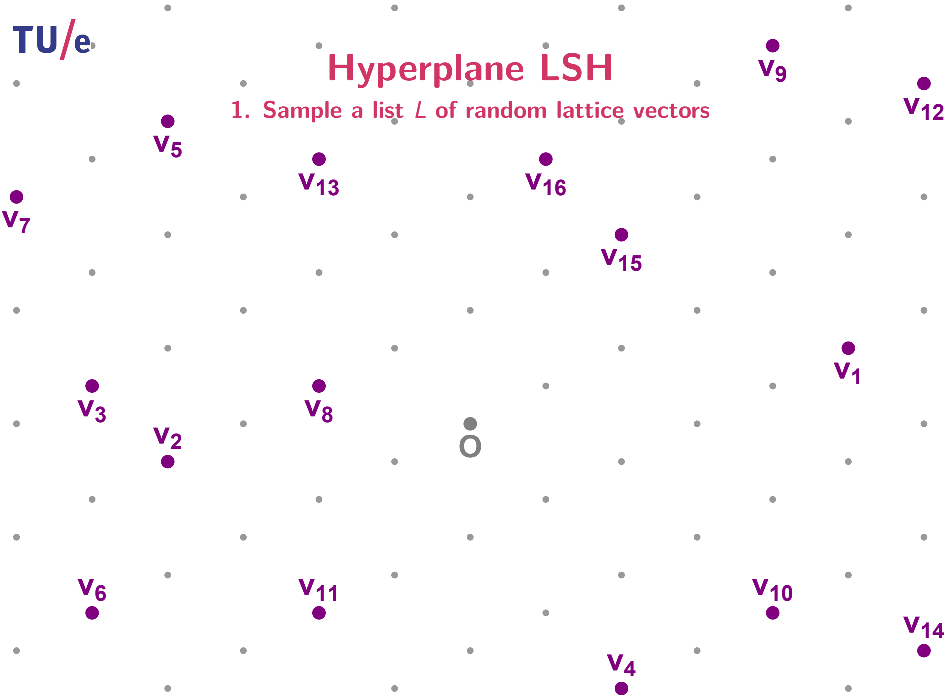
Hyperplane LSH

1. Sample a list L of random lattice vectors



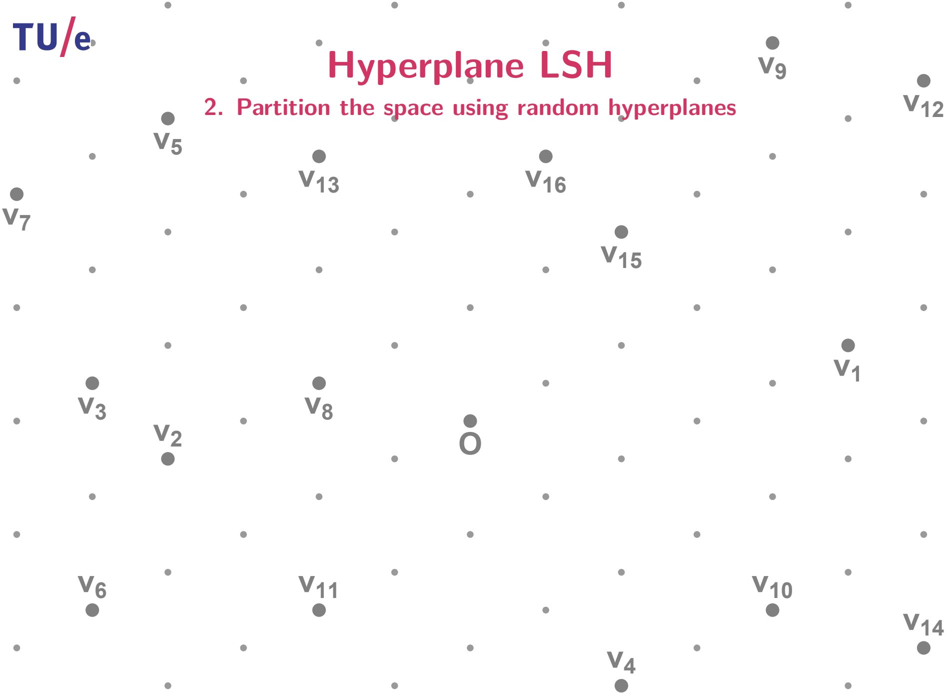
Hyperplane LSH

1. Sample a list L of random lattice vectors



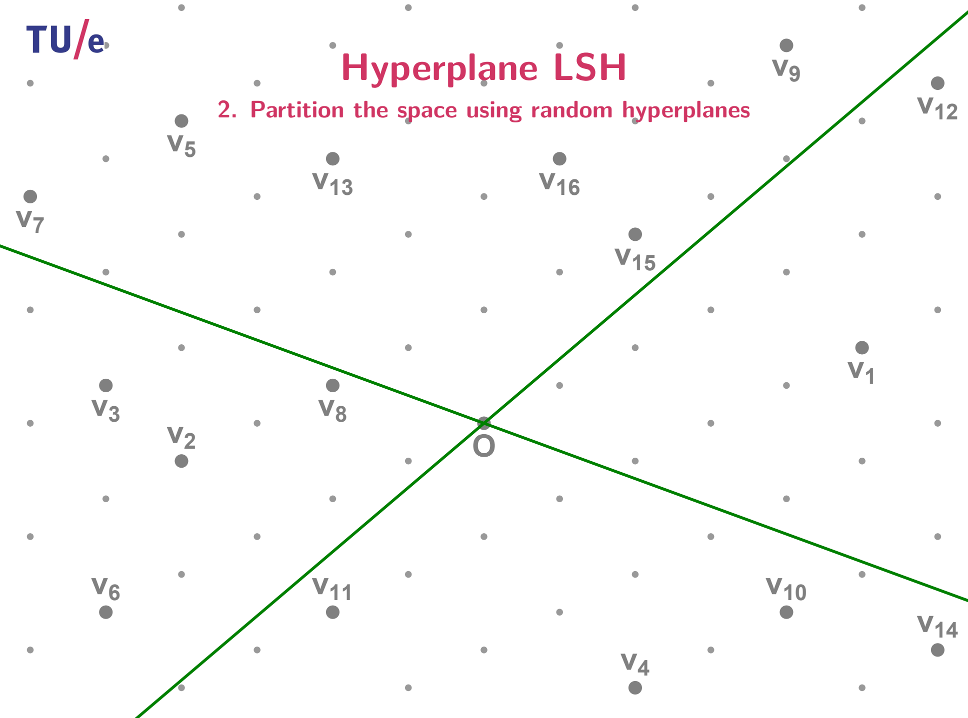
Hyperplane LSH

2. Partition the space using random hyperplanes



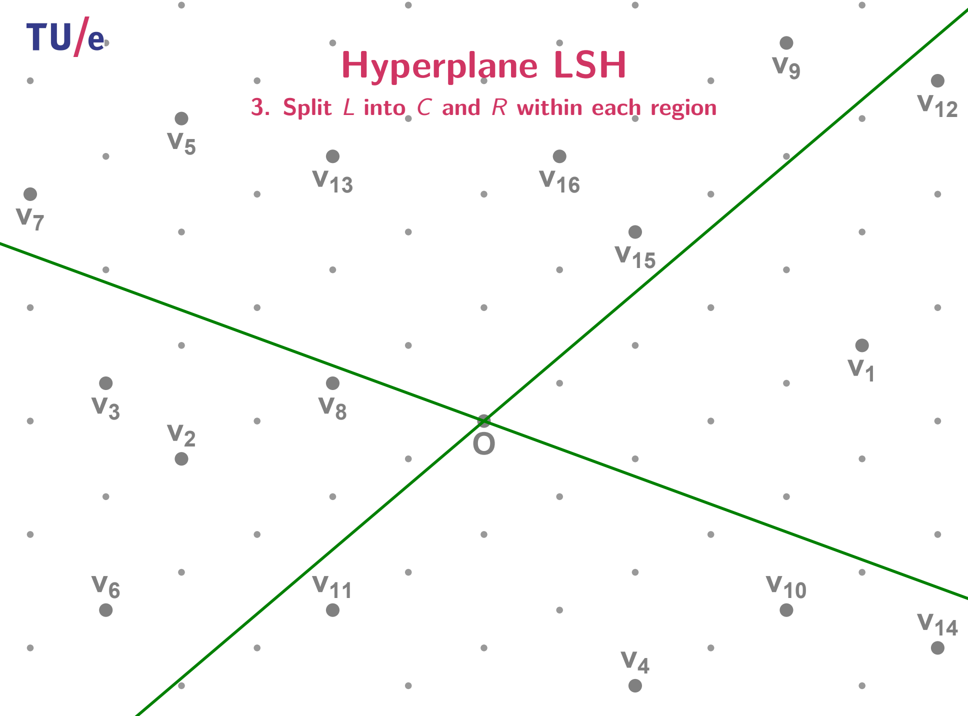
Hyperplane LSH

2. Partition the space using random hyperplanes



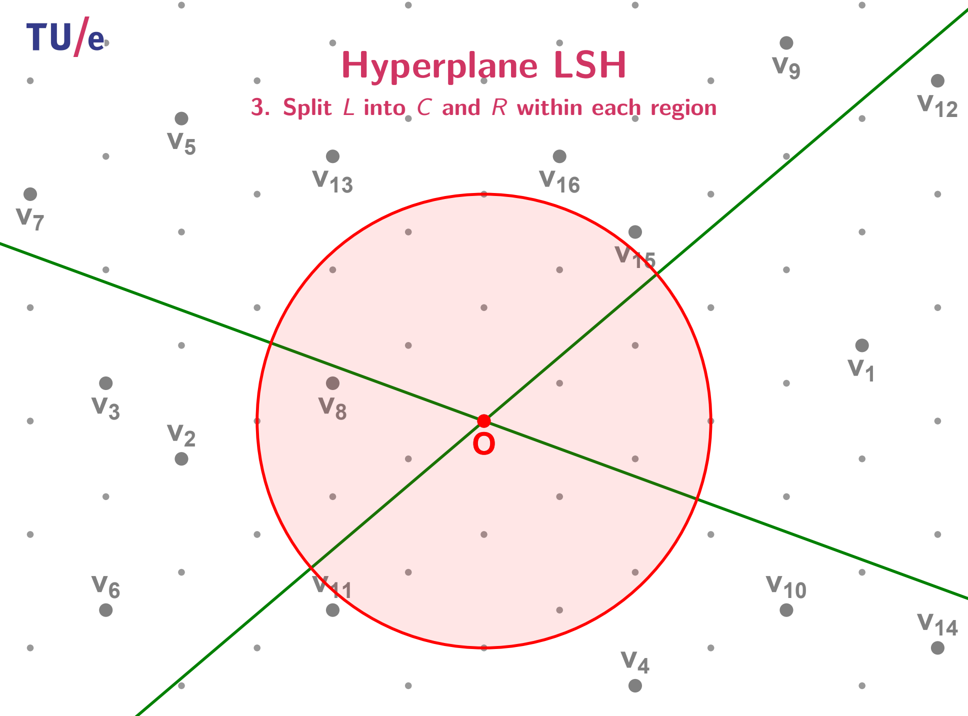
Hyperplane LSH

3. Split L into C and R within each region



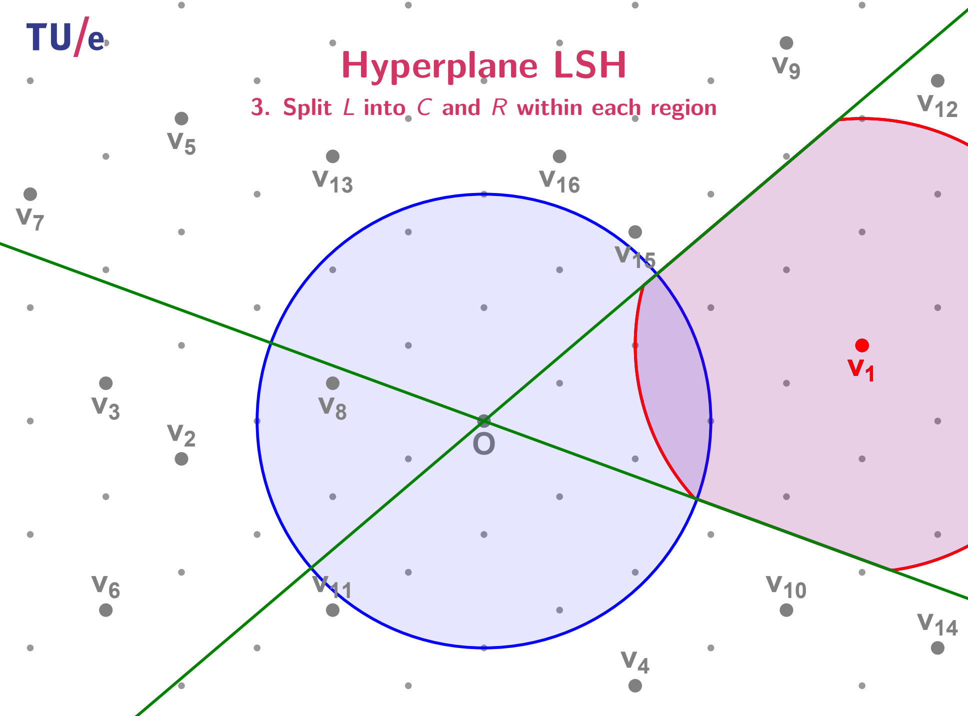
Hyperplane LSH

3. Split L into C and R within each region



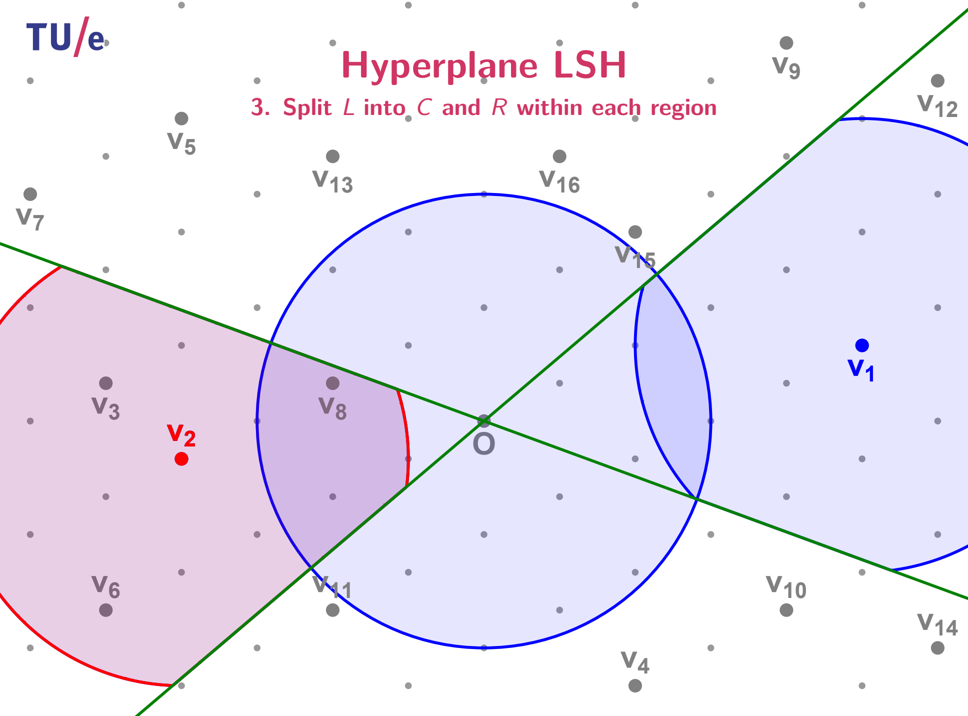
Hyperplane LSH

3. Split L into C and R within each region



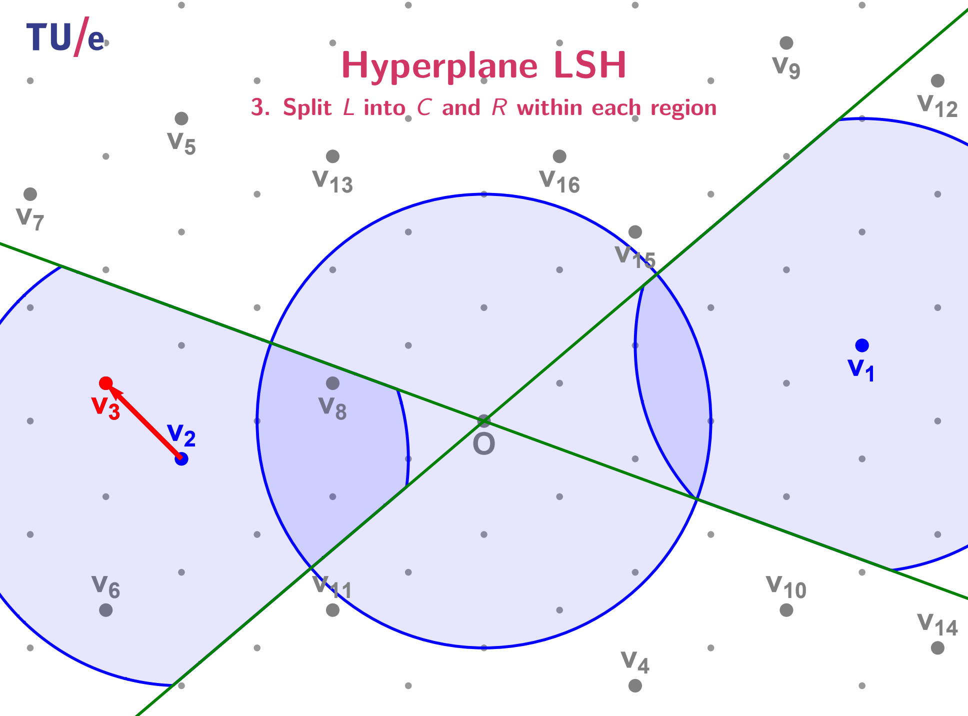
Hyperplane LSH

3. Split L into C and R within each region



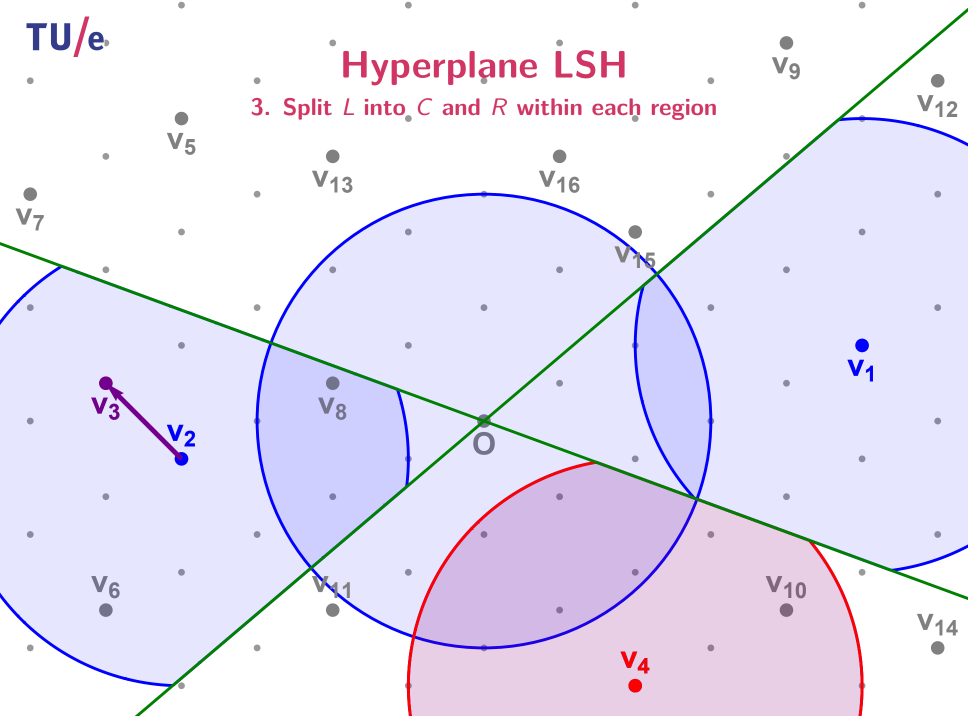
Hyperplane LSH

3. Split L into C and R within each region



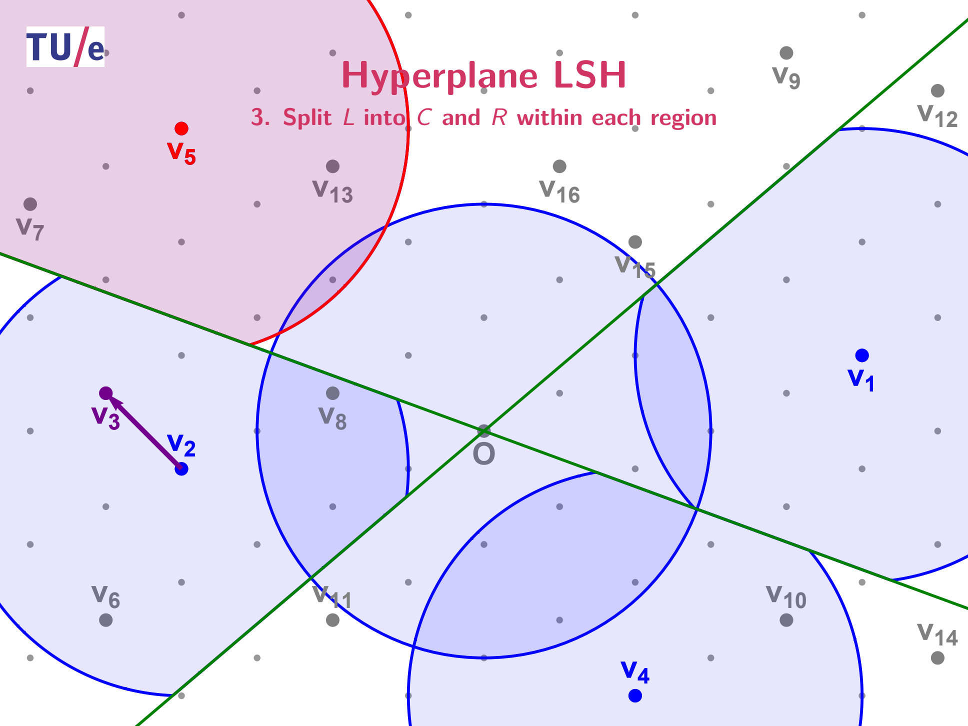
Hyperplane LSH

3. Split L into C and R within each region



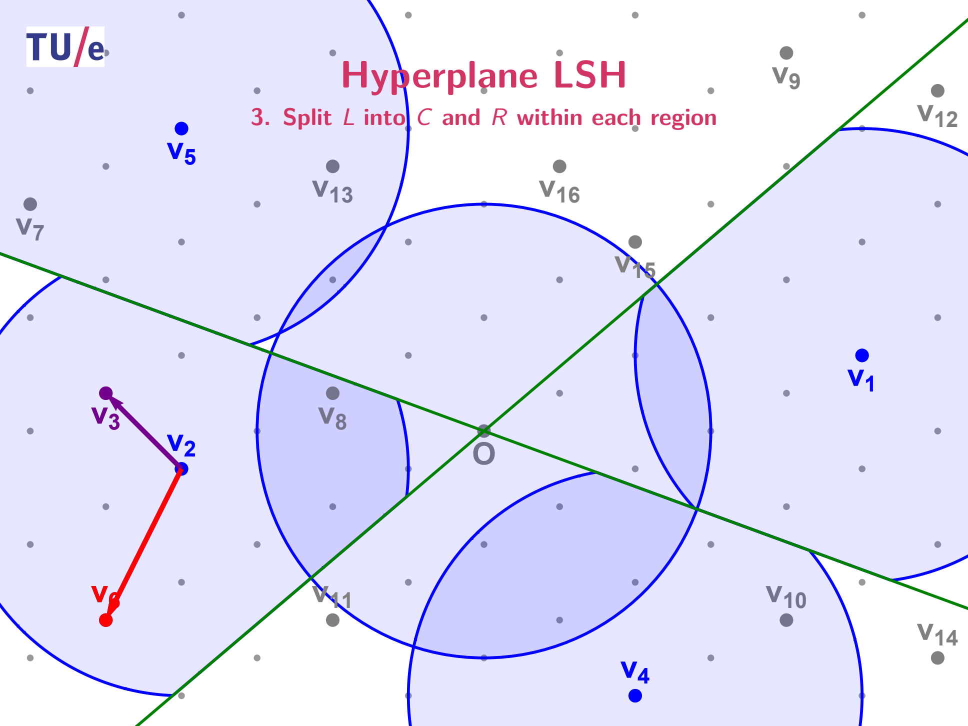
Hyperplane LSH

3. Split L into C and R within each region



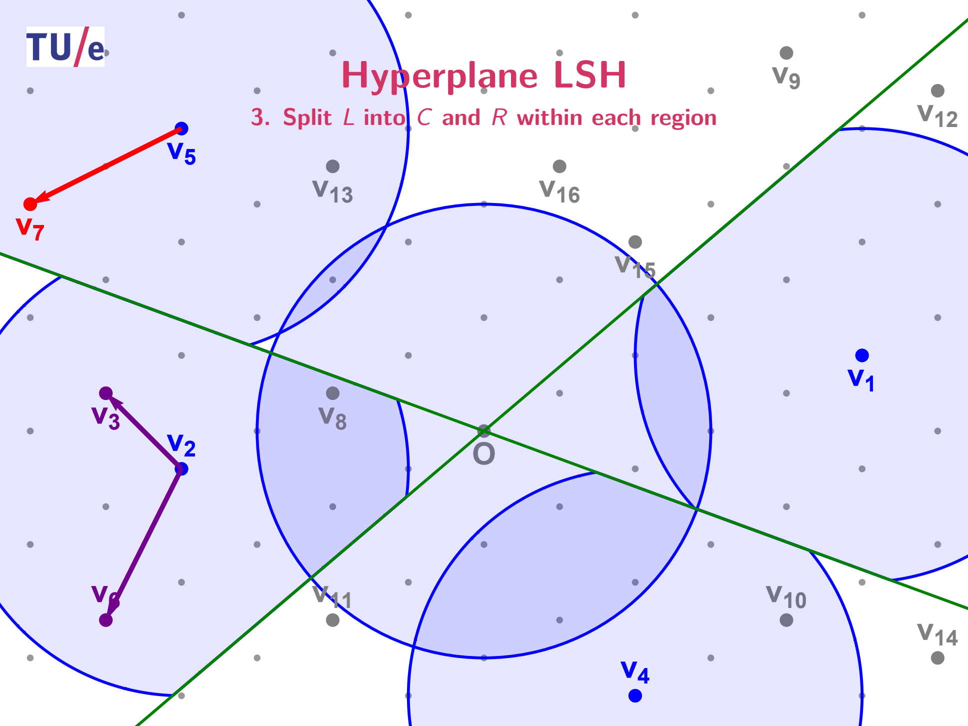
Hyperplane LSH

3. Split L into C and R within each region



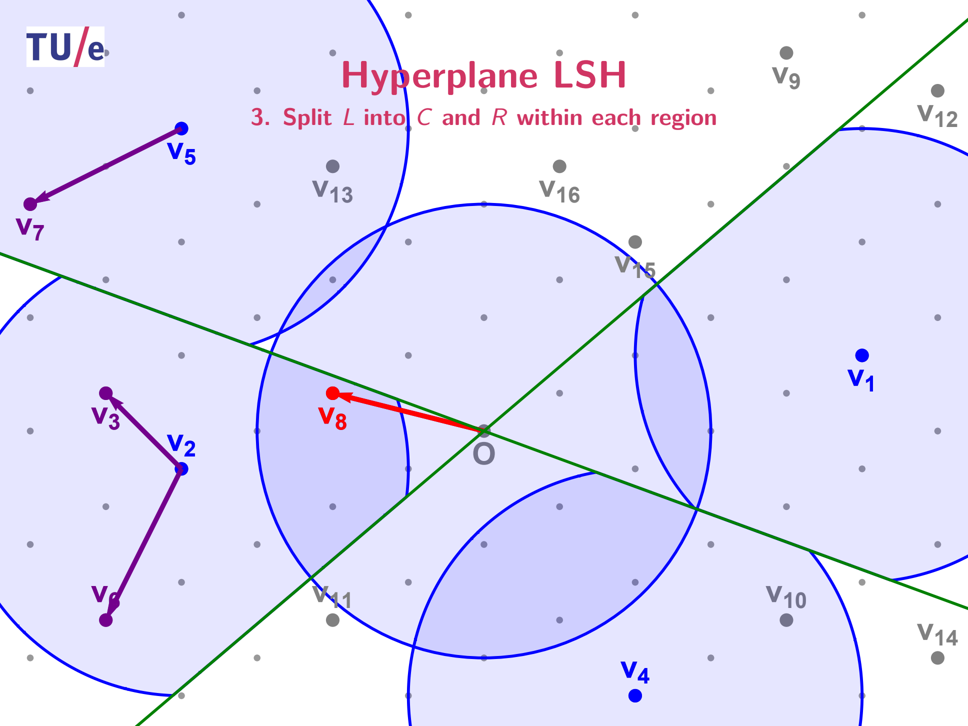
Hyperplane LSH

3. Split L into C and R within each region



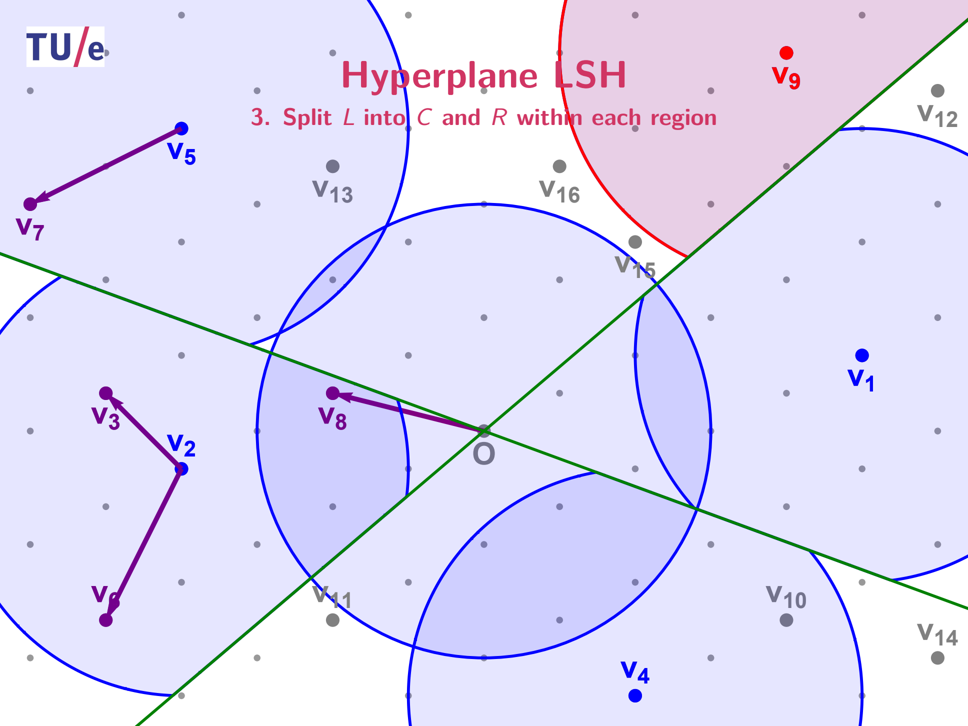
Hyperplane LSH

3. Split L into C and R within each region



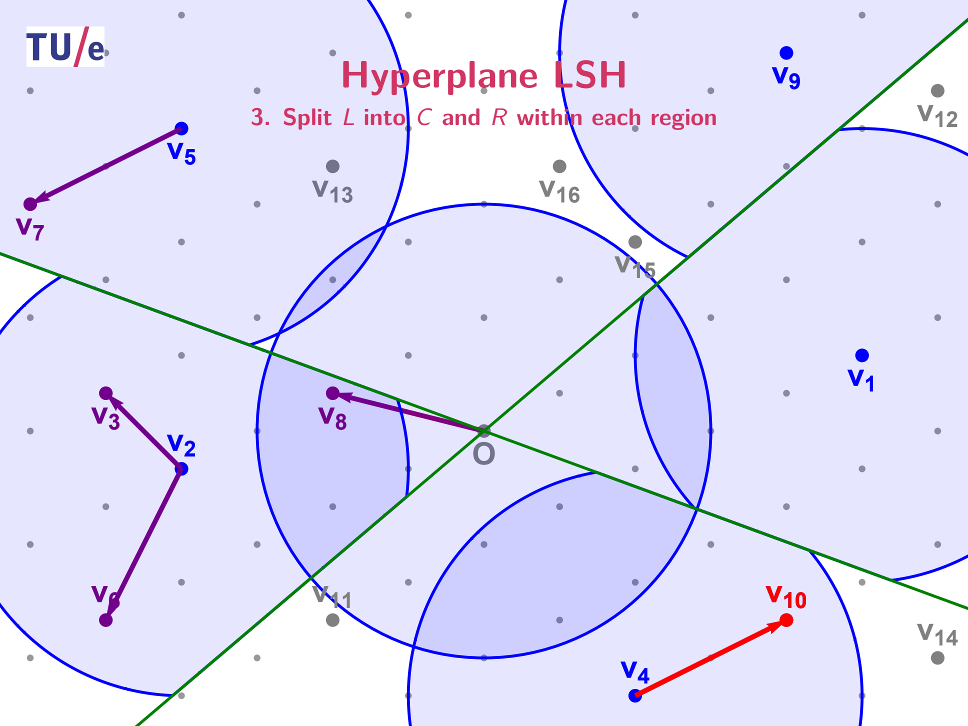
Hyperplane LSH

3. Split L into C and R within each region



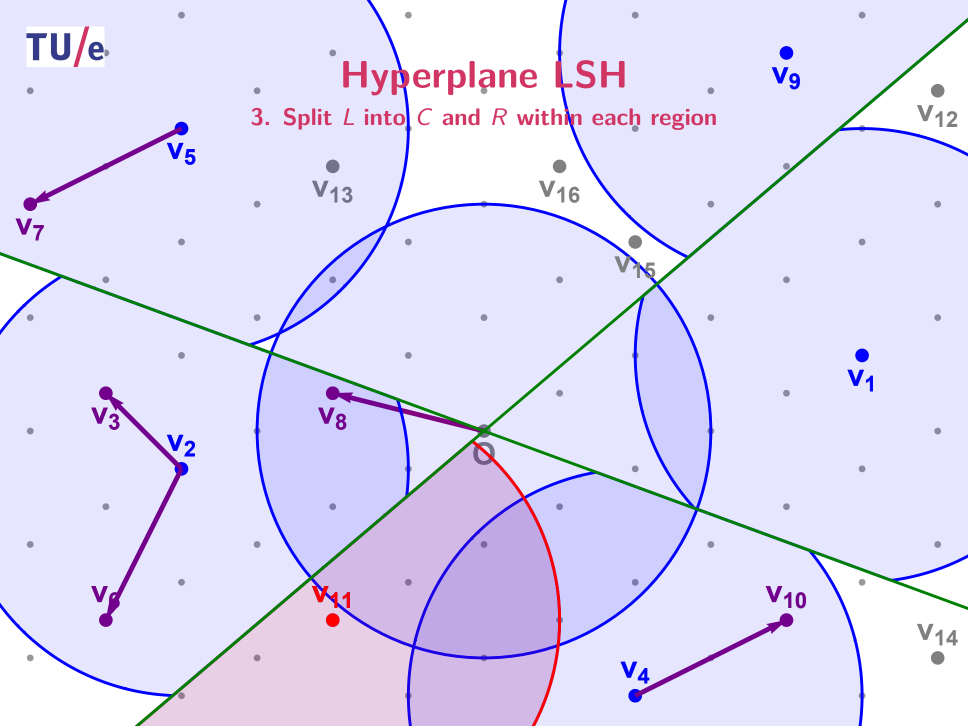
Hyperplane LSH

3. Split L into C and R within each region



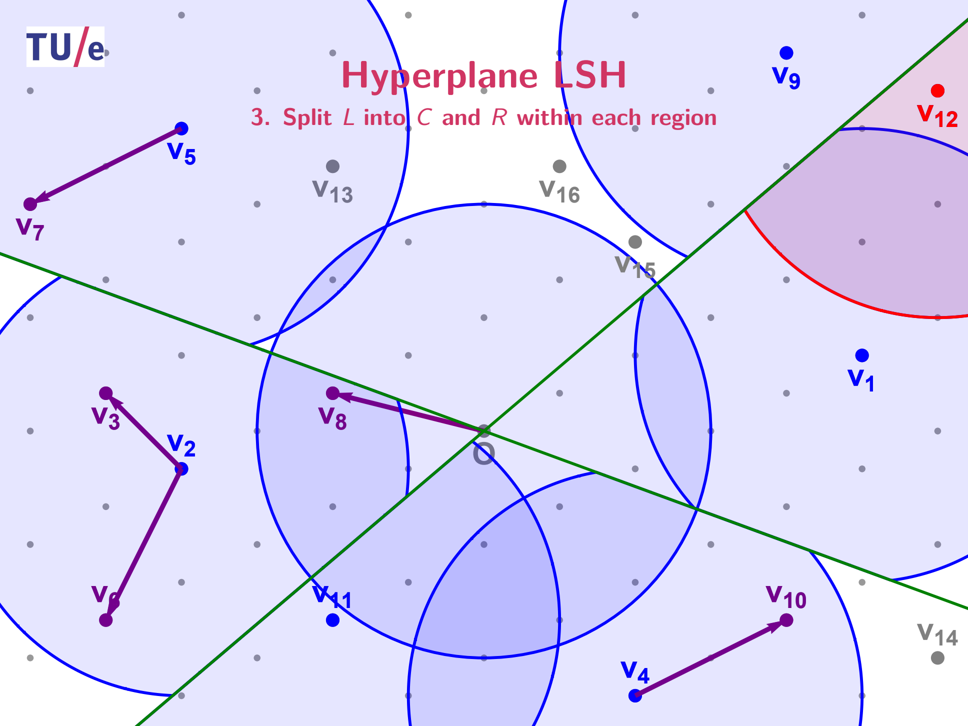
Hyperplane LSH

3. Split L into C and R within each region



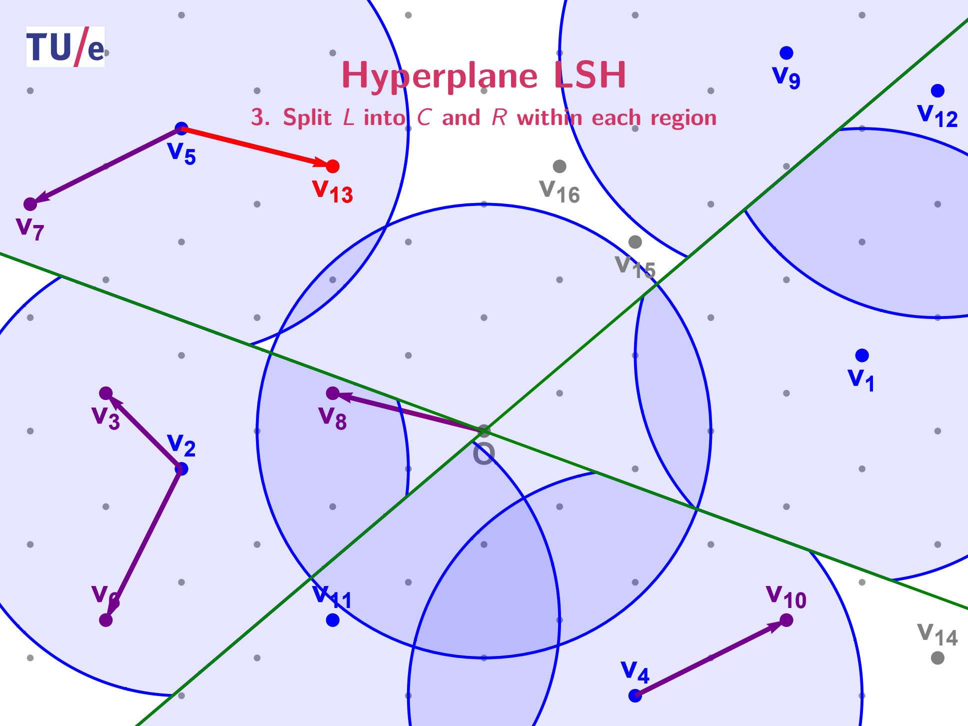
Hyperplane LSH

3. Split L into C and R within each region



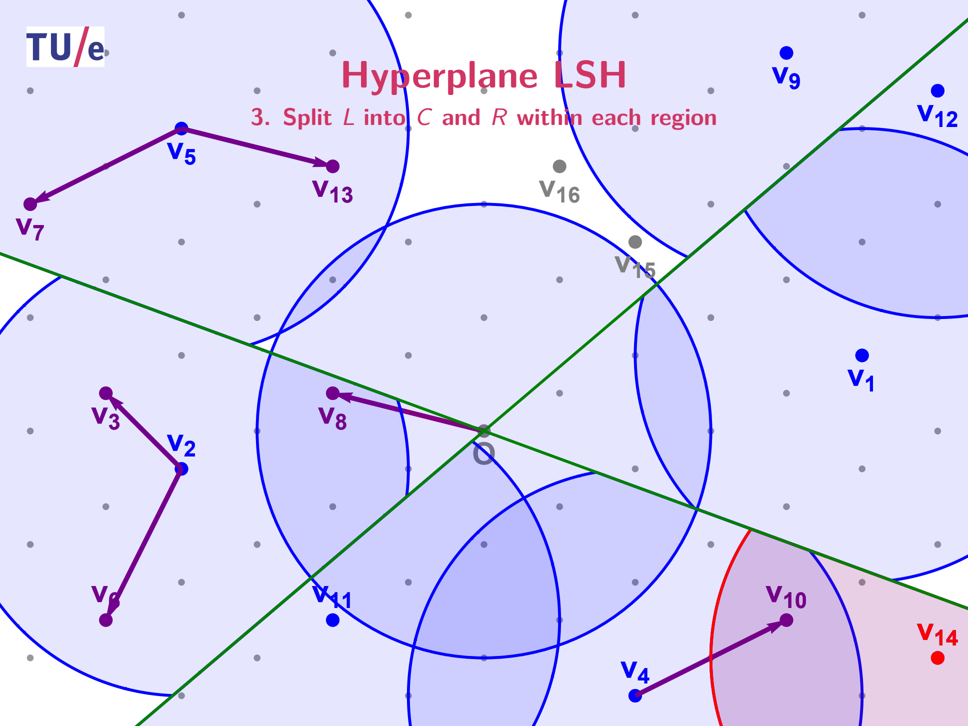
Hyperplane LSH

3. Split L into C and R within each region



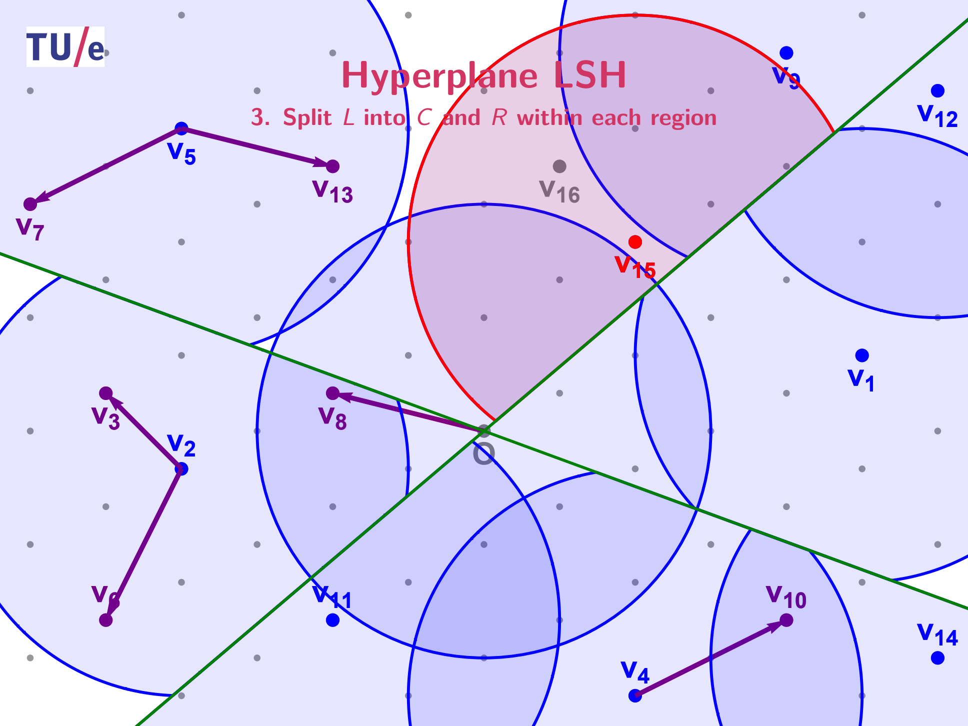
Hyperplane LSH

3. Split L into C and R within each region



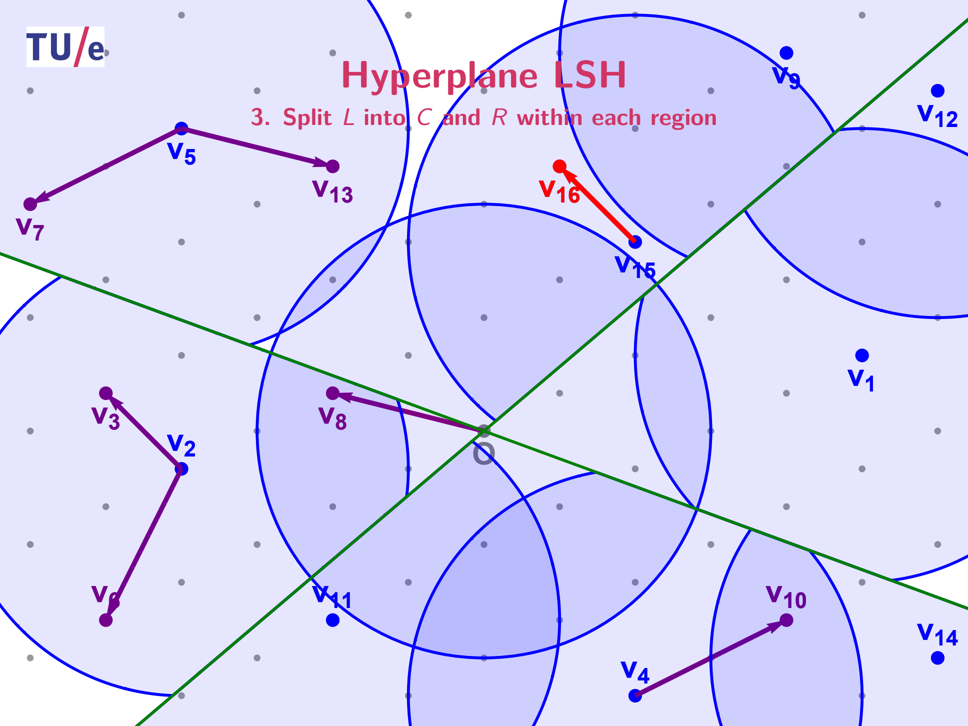
Hyperplane LSH

3. Split L into C and R within each region



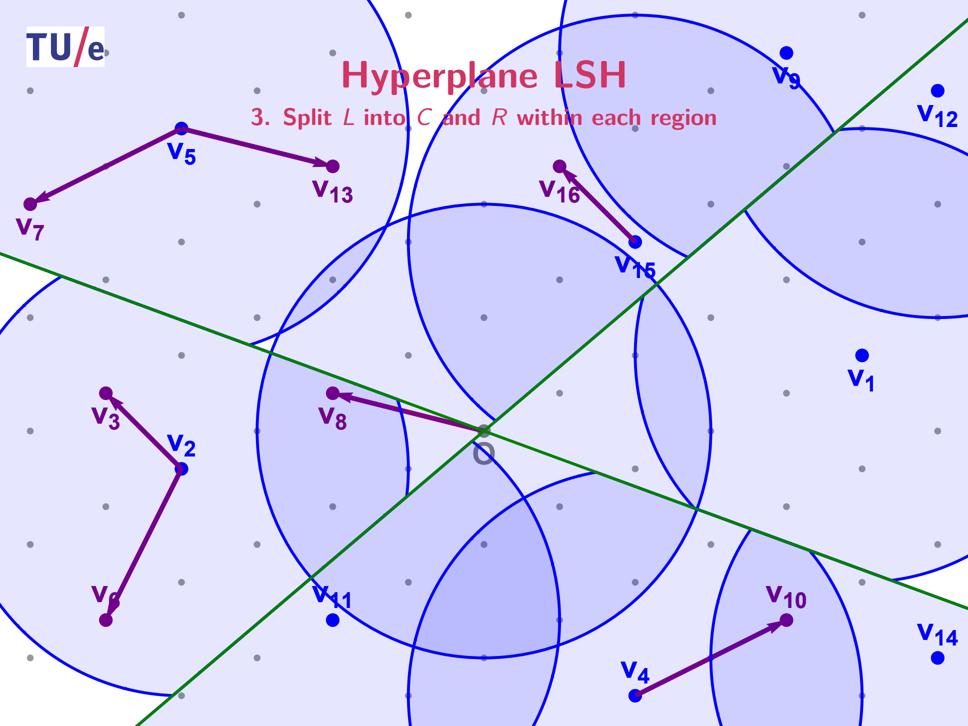
Hyperplane LSH

3. Split L into C and R within each region



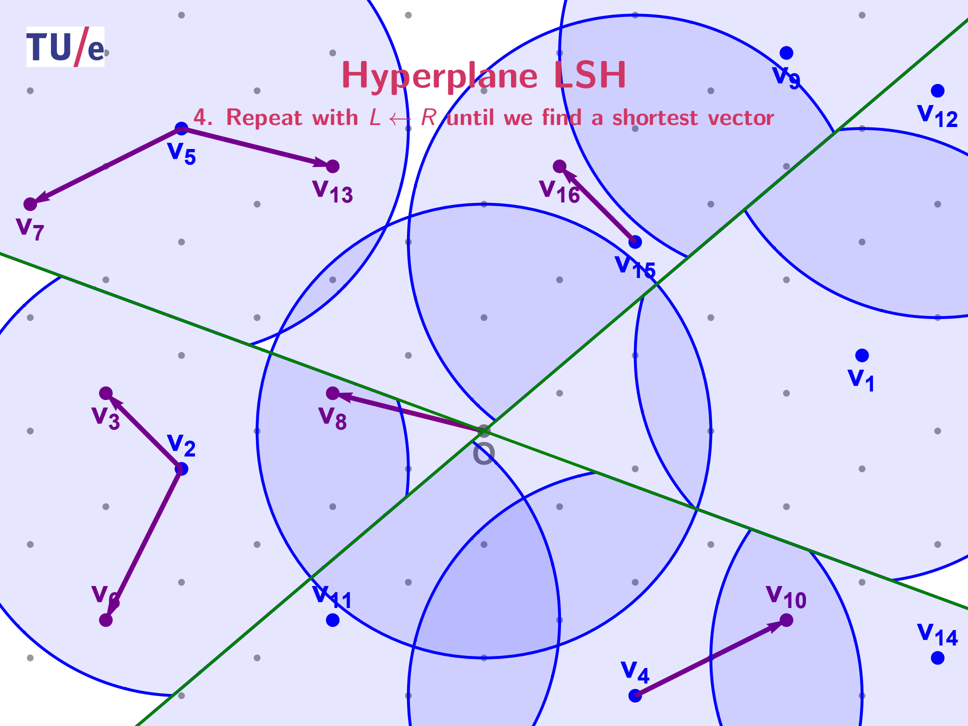
Hyperplane LSH

3. Split L into C and R within each region



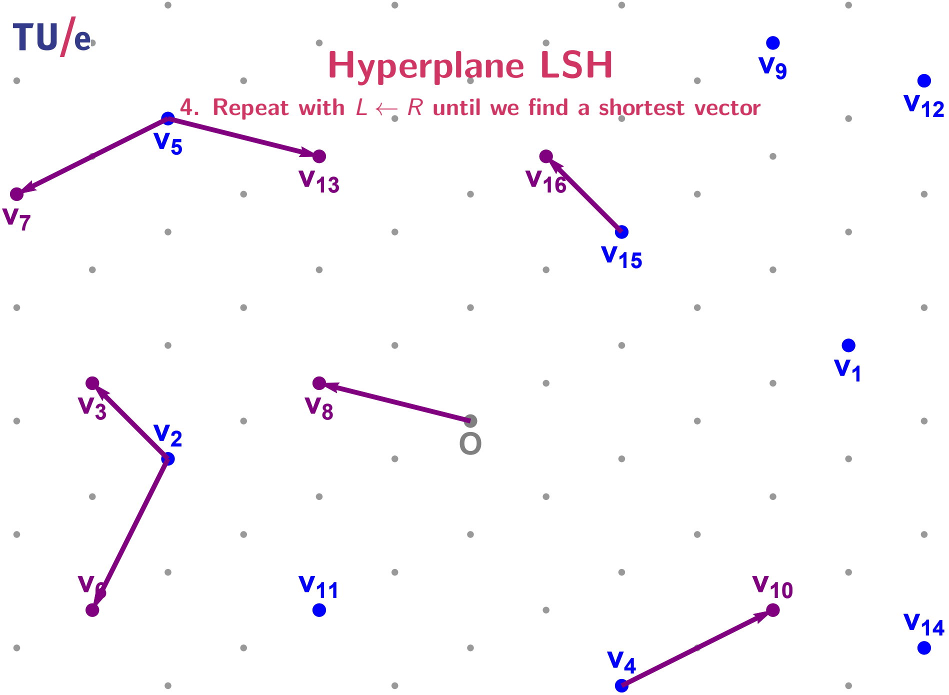
Hyperplane LSH

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



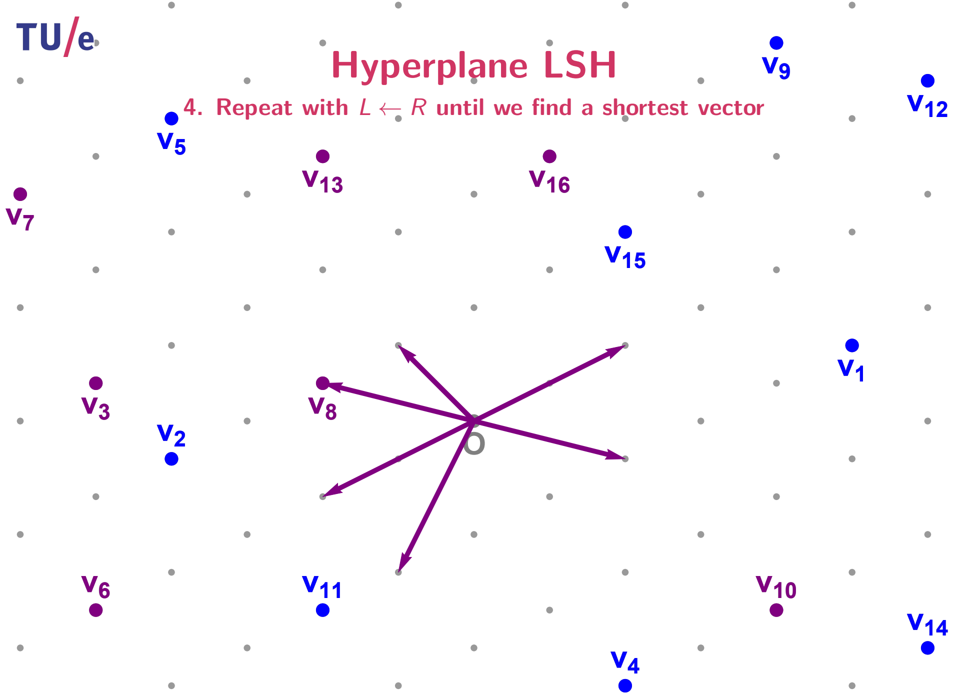
Hyperplane LSH

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



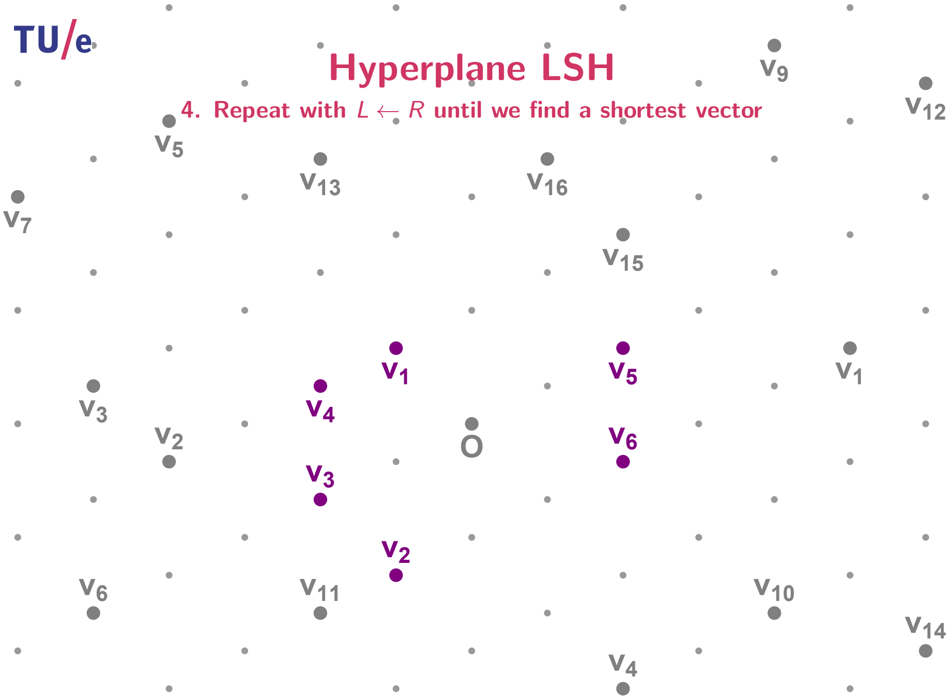
Hyperplane LSH

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



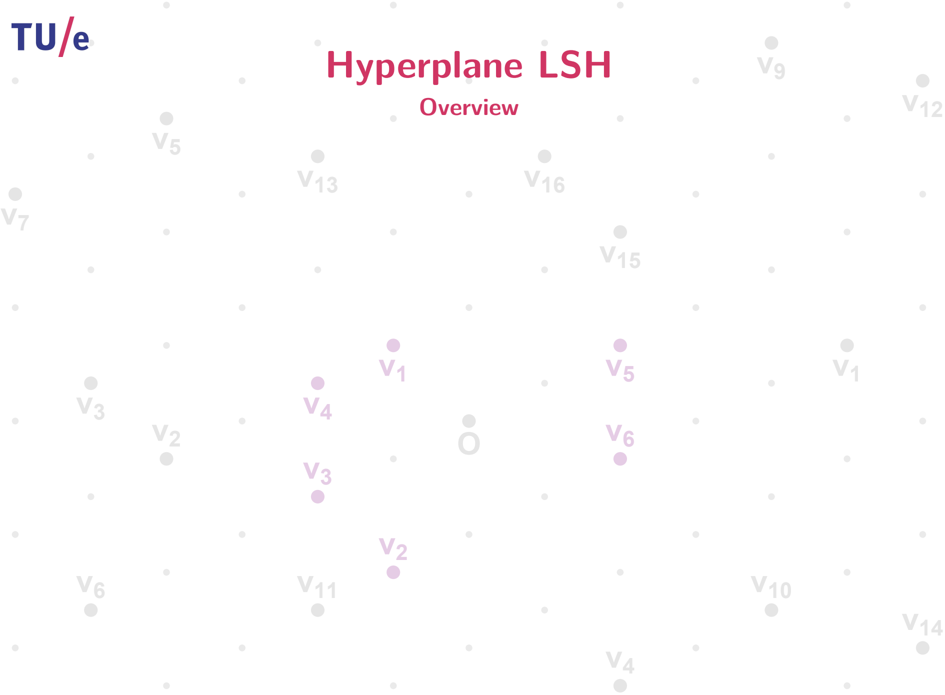
Hyperplane LSH

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



Hyperplane LSH

Overview



Hyperplane 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”

Hyperplane 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: $2^{0.337n+o(n)}$
  - Number of vectors: $2^{0.208n+o(n)}$
  - Number of hash tables: $2^{0.129n+o(n)}$
  - Each hash table contains all vectors

Hyperplane 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: $2^{0.337n+o(n)}$
  - Number of vectors: $2^{0.208n+o(n)}$
  - Number of hash tables: $2^{0.129n+o(n)}$
  - Each hash table contains all vectors
- Time complexity: $2^{0.337n+o(n)}$
  - Cost of computing hashes: $2^{0.129n+o(n)}$
  - Candidate nearest vectors: $2^{0.129n+o(n)}$
  - Repeat this for each list vector: $2^{0.208n+o(n)}$

Hyperplane LSH

Overview

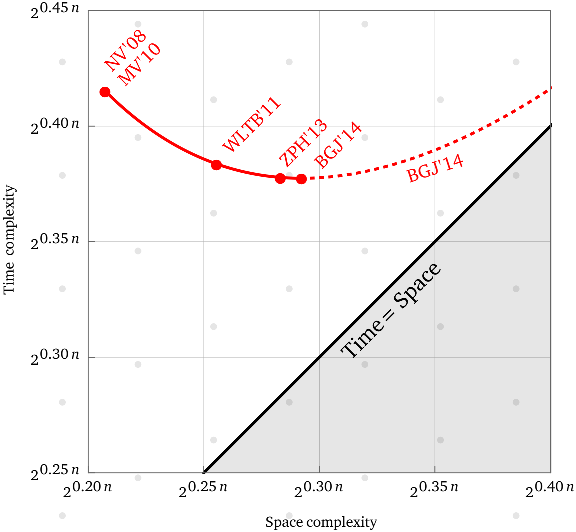
- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent "hash tables"
- Space complexity: $2^{0.337n+o(n)}$
  - Number of vectors: $2^{0.208n+o(n)}$
  - Number of hash tables: $2^{0.129n+o(n)}$
  - Each hash table contains all vectors
- Time complexity: $2^{0.337n+o(n)}$
  - Cost of computing hashes: $2^{0.129n+o(n)}$
  - Candidate nearest vectors: $2^{0.129n+o(n)}$
  - Repeat this for each list vector: $2^{0.208n+o(n)}$

Theorem

Sieving with hyperplane LSH heuristically solves SVP in time and space $2^{0.337n+o(n)}$.

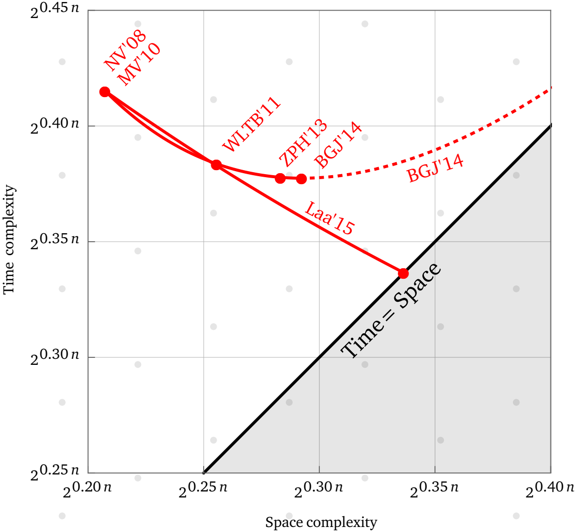
Hyperplane LSH

Space/time trade-off



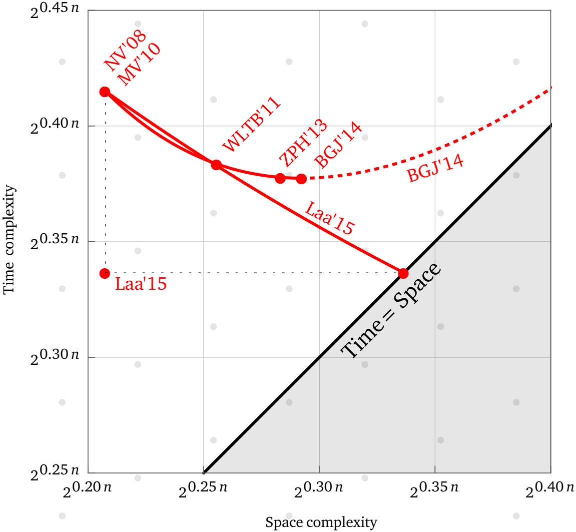
Hyperplane LSH

Space/time trade-off



Hyperplane LSH

Space/time trade-off



Questions

[vdP'12]