Sieving for shortest vectors in lattices using locality-sensitive hashing

Thijs Laarhoven

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

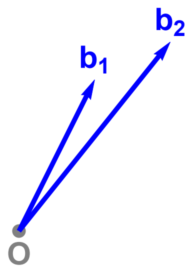
ENS Lyon, Lyon, France
(June 10, 2015)

Lattices

What is a lattice?

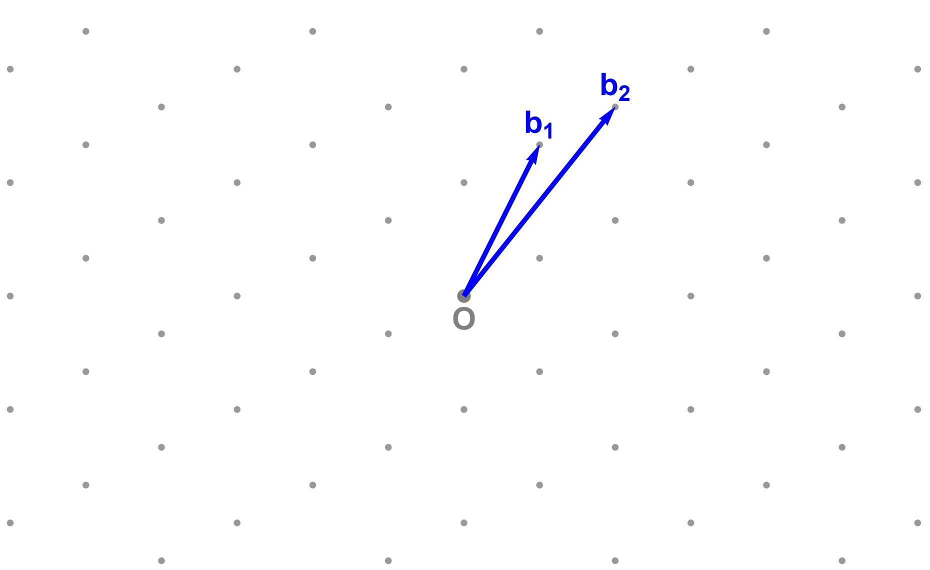
Lattices

What is a lattice?



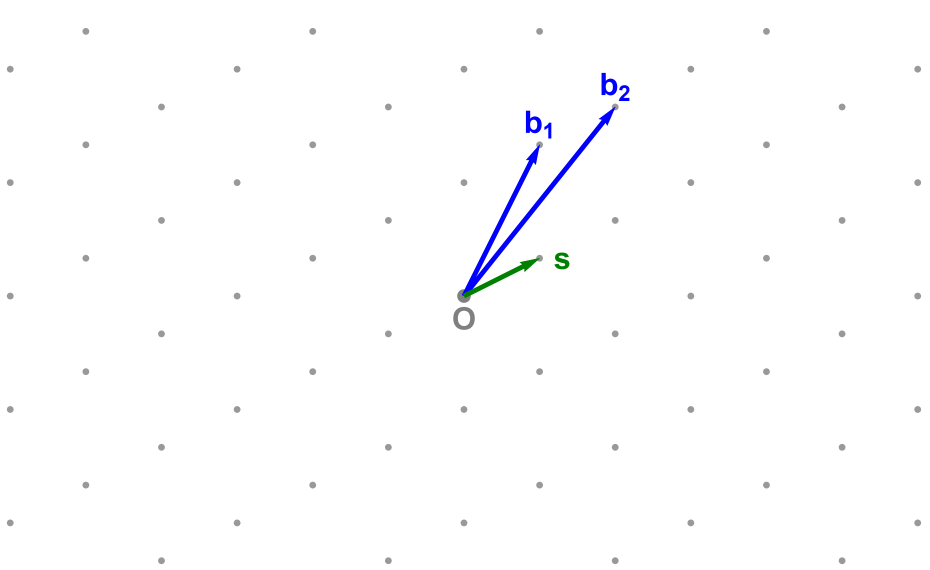
Lattices

What is a lattice?



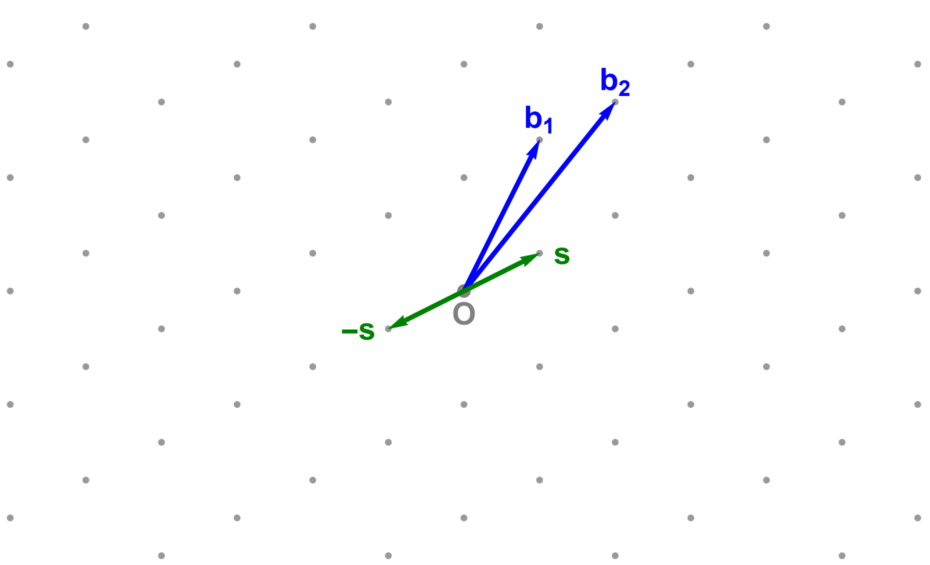
Lattices

Shortest Vector Problem (SVP)



Lattices

Shortest Vector Problem (SVP)



Lattices

SVP algorithms

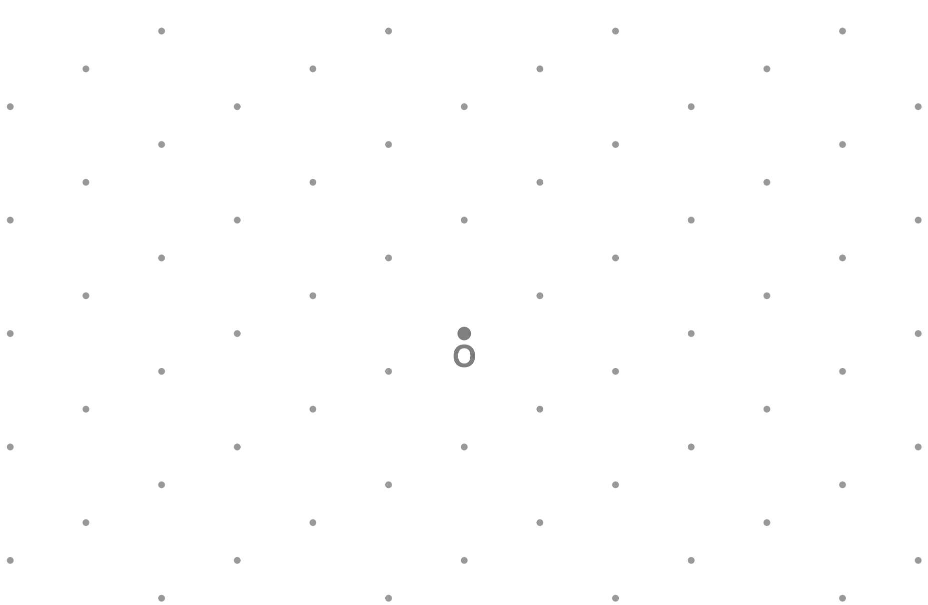
	Algorithm	$\log_2(\text{Time})$	$\log_2(\text{Space})$
Provable SVP	Enumeration [Poh81, Kan83, . . . , GNR10]	$\Omega(n \log n)$	$O(\log n)$
	AKS-sieve [AKS01, NV08, MV10, HPS11]	$3.398n$	$1.985n$
	ListSieve [MV10, MDB14]	$3.199n$	$1.327n$
	AKS-sieve-birthday [PS09, HPS11]	$2.648n$	$1.324n$
	ListSieve-birthday [PS09]	$2.465n$	$1.233n$
	Voronoi cell algorithm [MV10b]	$2.000n$	$1.000n$
	Discrete Gaussian sampling [ADRS15, ADS15]	$1.000n$	$1.000n$
Heuristic SVP	NV-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]	$0.3778n$	$0.283n$
	Decomposition approach [BGJ14]	$0.3774n$	$0.293n$
	HashSieve [Laa15, MLB15]	$0.337n$	$0.208n$*
	Another NNS sieve [BGJ15]	$0.311n$	$0.208n$
	SphereSieve [LdW15]	$0.298n$	$0.208n$
	CrossPolytopeSieve [BL15]	$0.298n$	$0.208n$*

Lattices

SVP algorithms

	Algorithm	$\log_2(\text{Time})$	$\log_2(\text{Space})$
Provable SVP	Enumeration [Poh81, Kan83, ..., GNR10]	$\Omega(n \log n)$	$O(\log n)$
	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 [MV10b]	$2.000n$	$1.000n$
	Discrete Gaussian sampling [ADRS15, ADS15]	$1.000n$	$1.000n$
Heuristic SVP	**NV-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]	$0.3778n$	$0.283n$
	Decomposition approach [BGJ14]	$0.3774n$	$0.293n$
	HashSieve [Laa15, MLB15]	$0.337n$	$0.208n$*
	Another NNS sieve [BGJ15]	$0.311n$	$0.208n$
	SphereSieve [LdW15]	$0.298n$	$0.208n$
	CrossPolytopeSieve [BL15]	$0.298n$	$0.208n$*

Nguyen-Vidick sieve



Nguyen-Vidick sieve

1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve

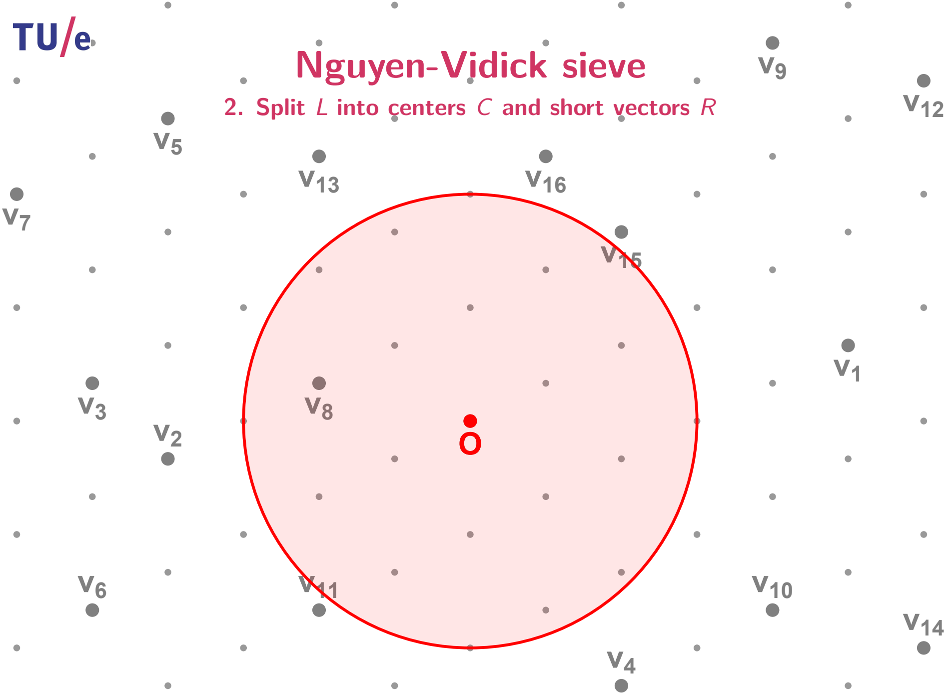
1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve

2. Split L into centers C and short vectors R

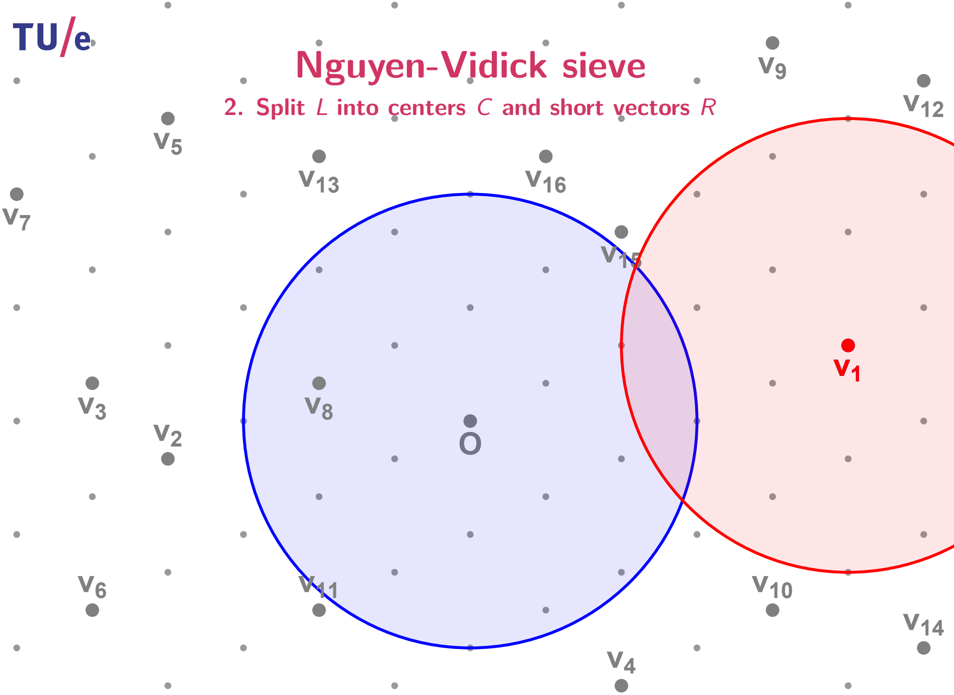
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



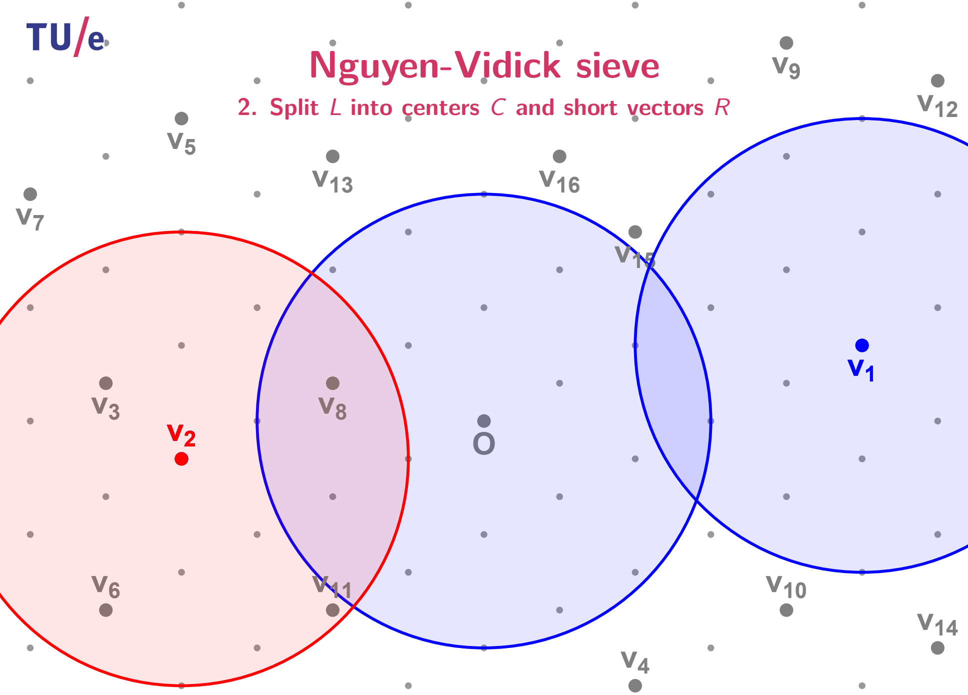
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



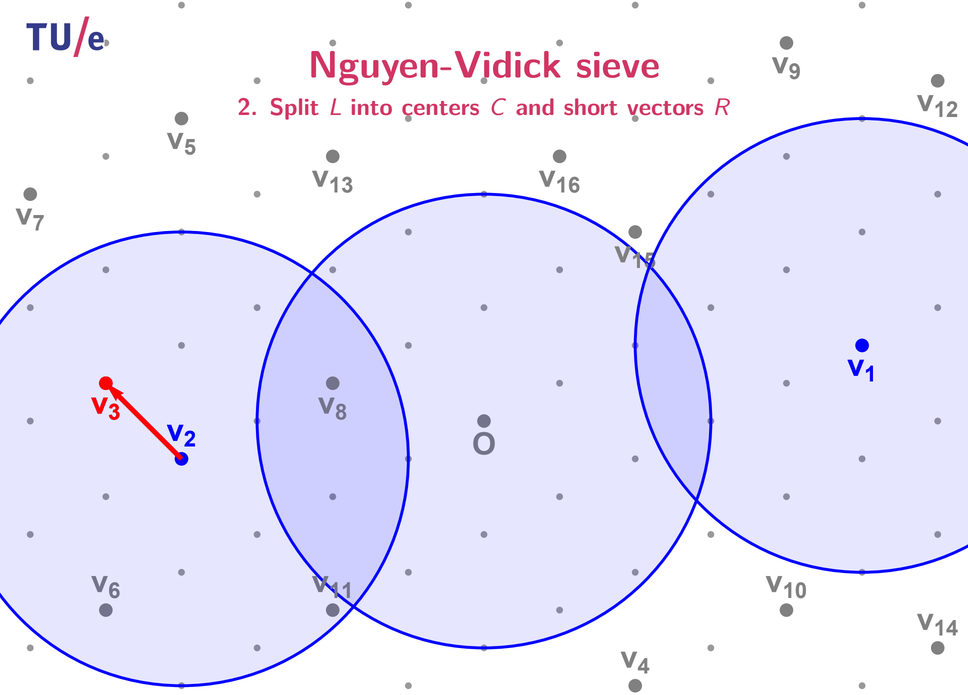
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



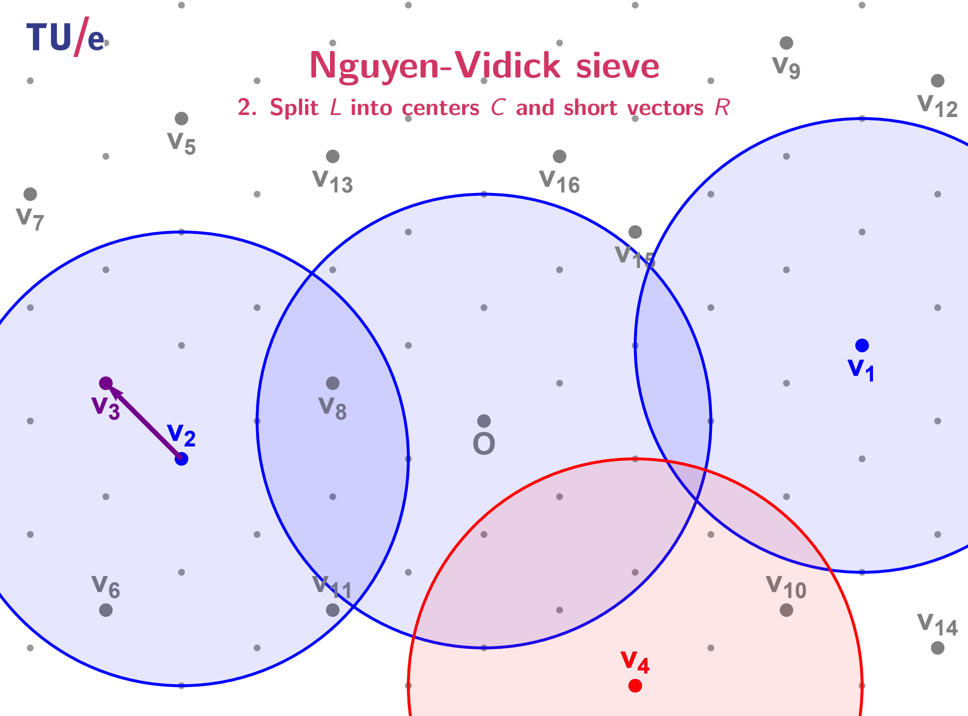
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



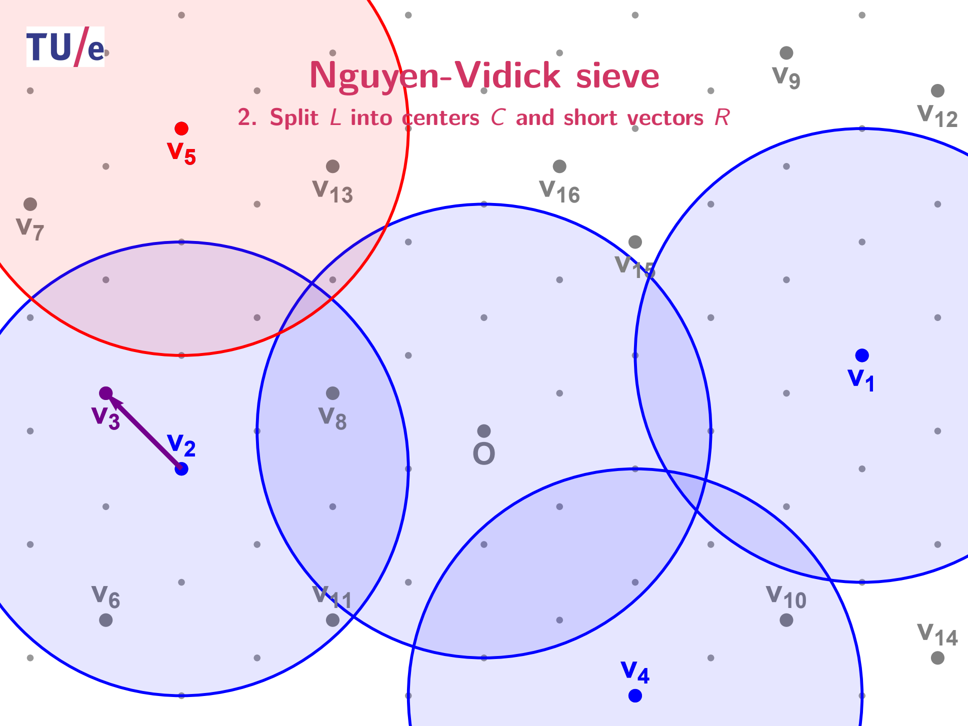
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



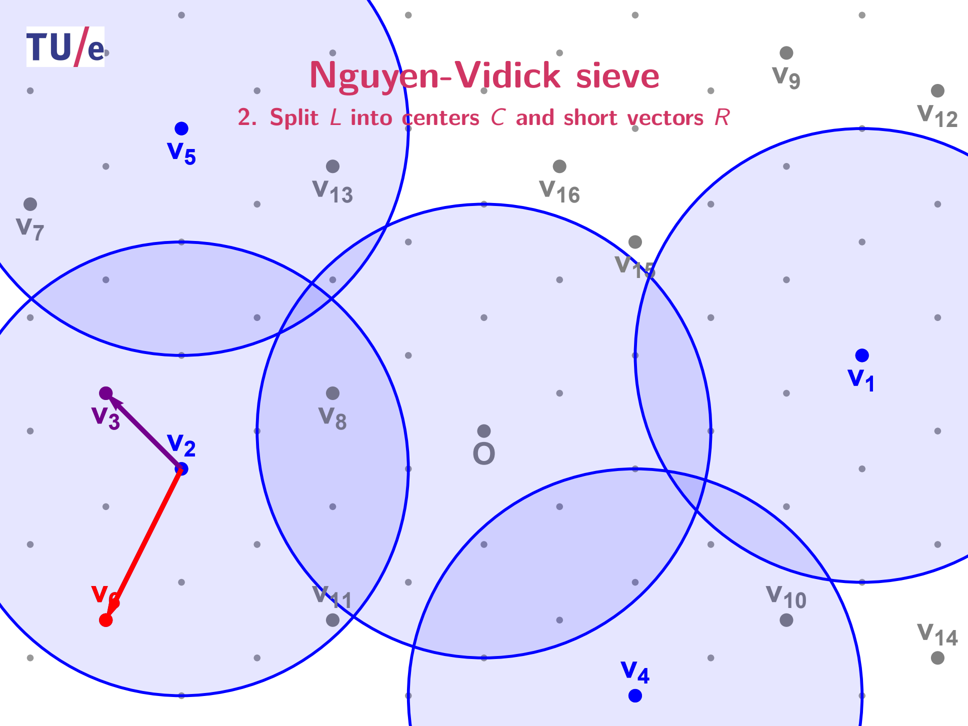
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



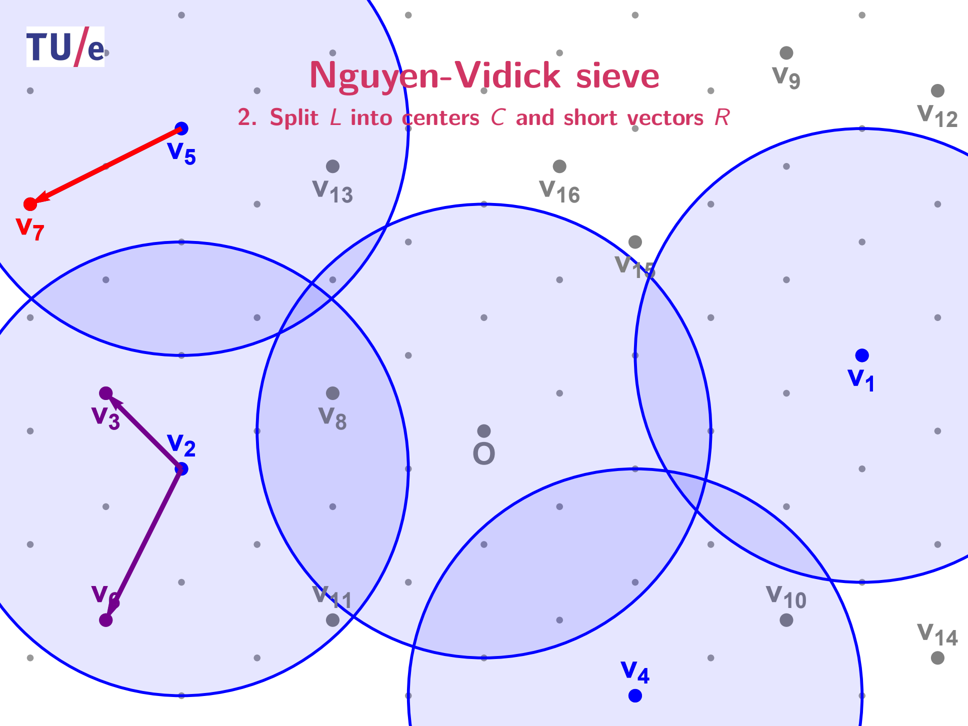
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



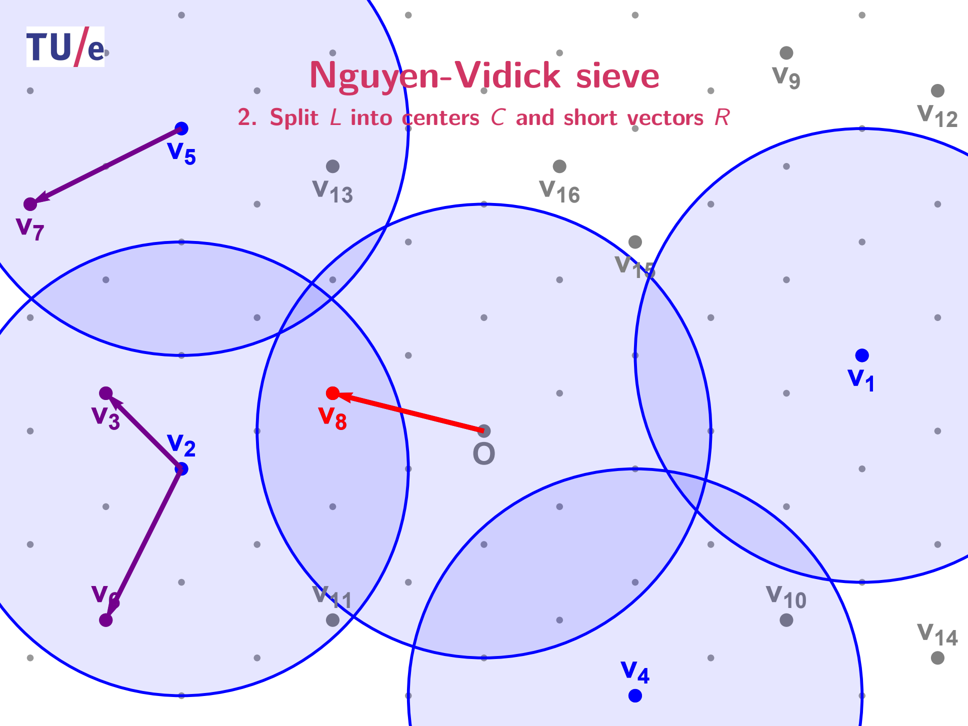
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



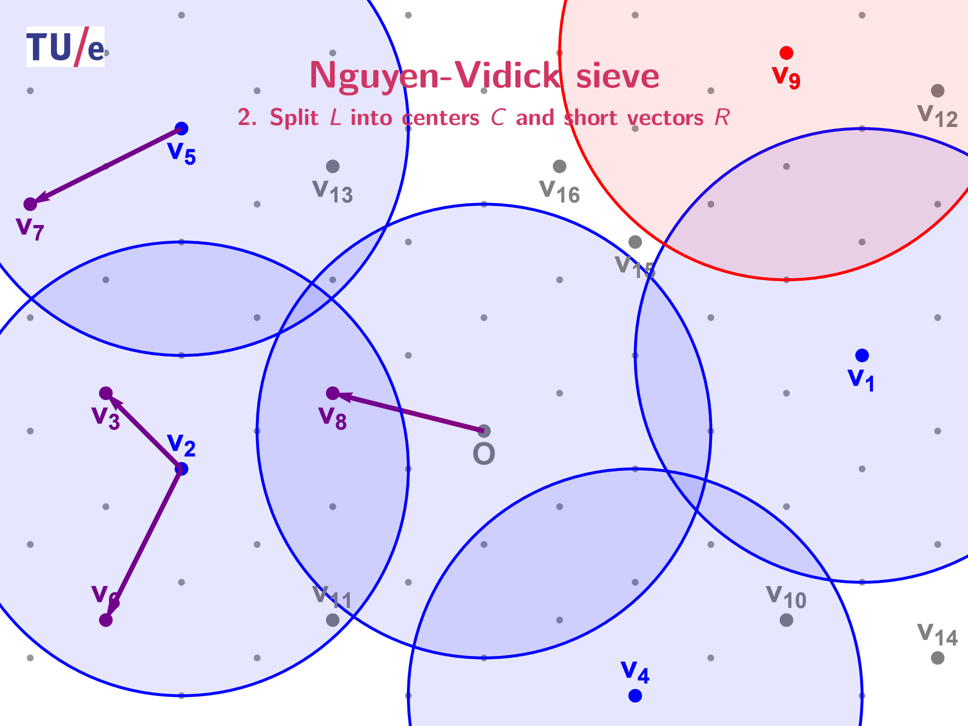
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



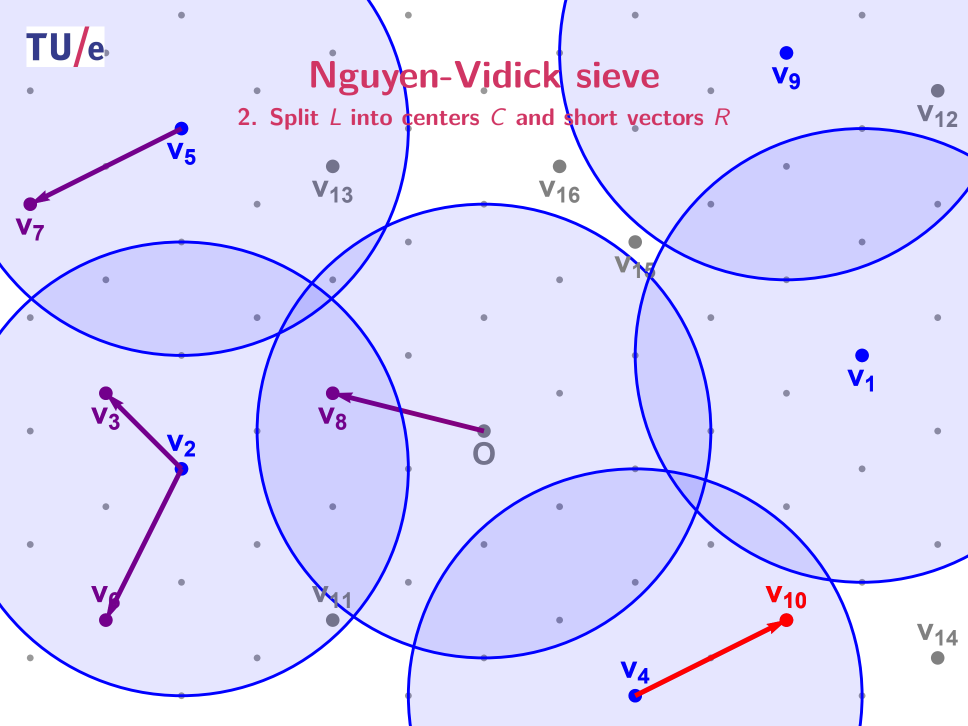
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



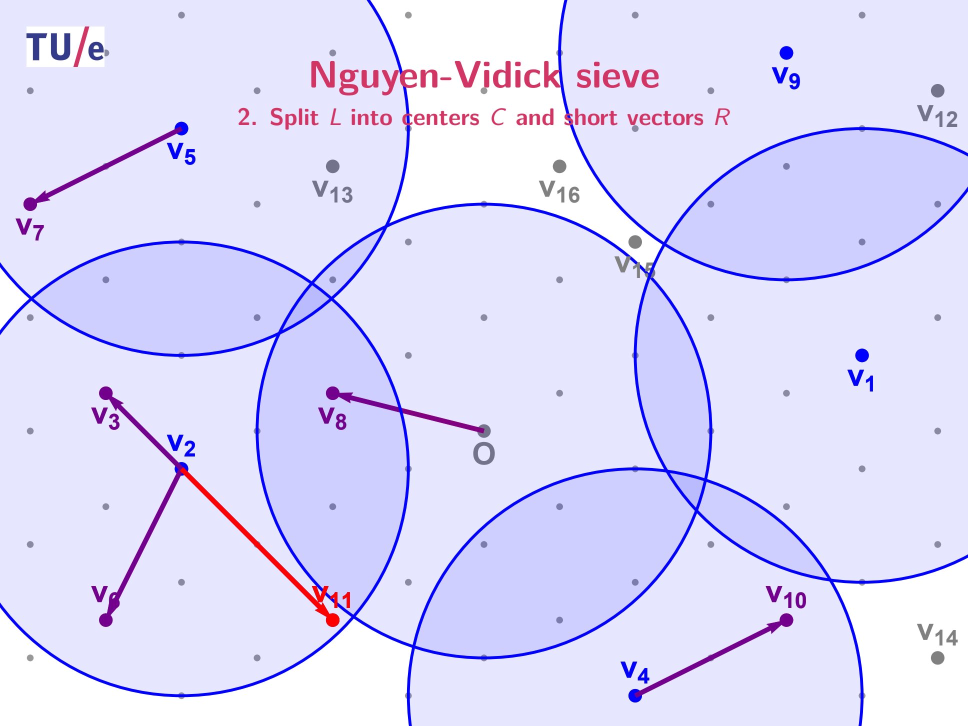
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



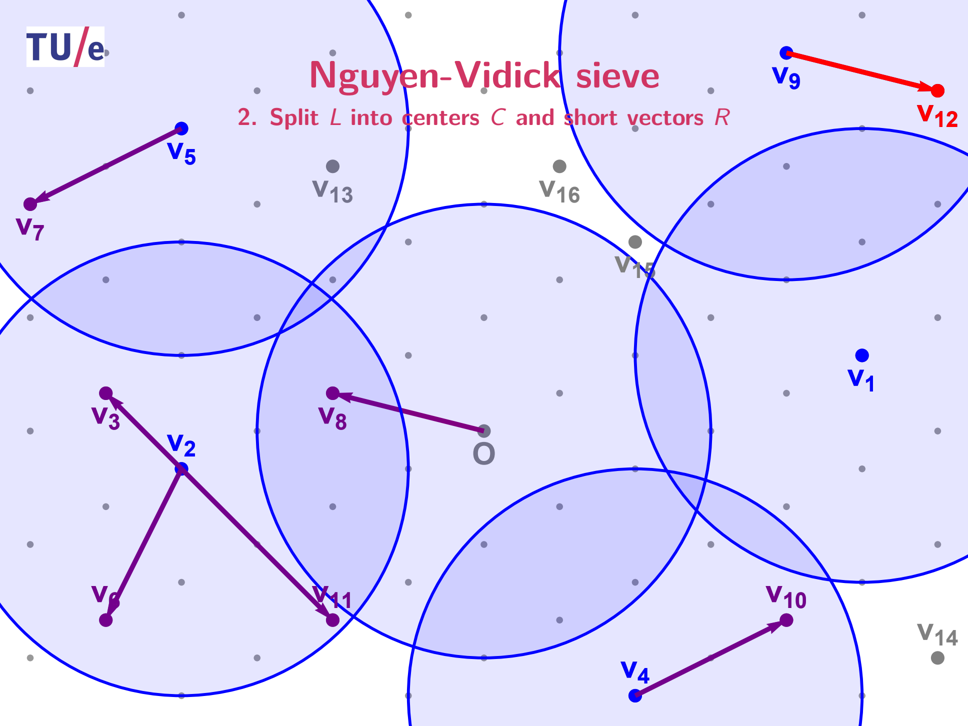
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



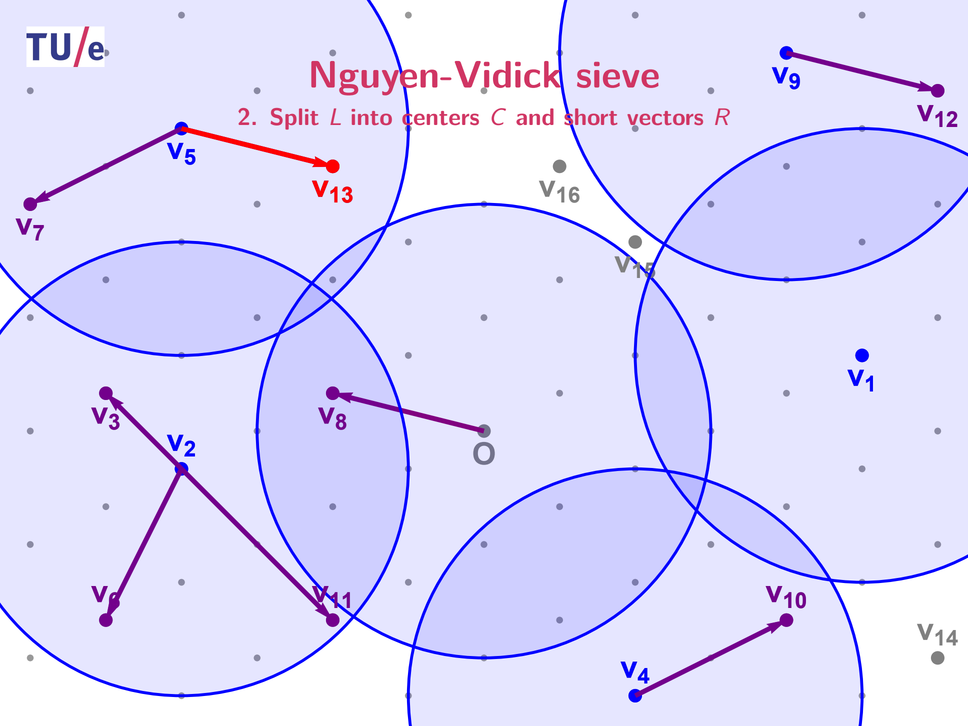
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



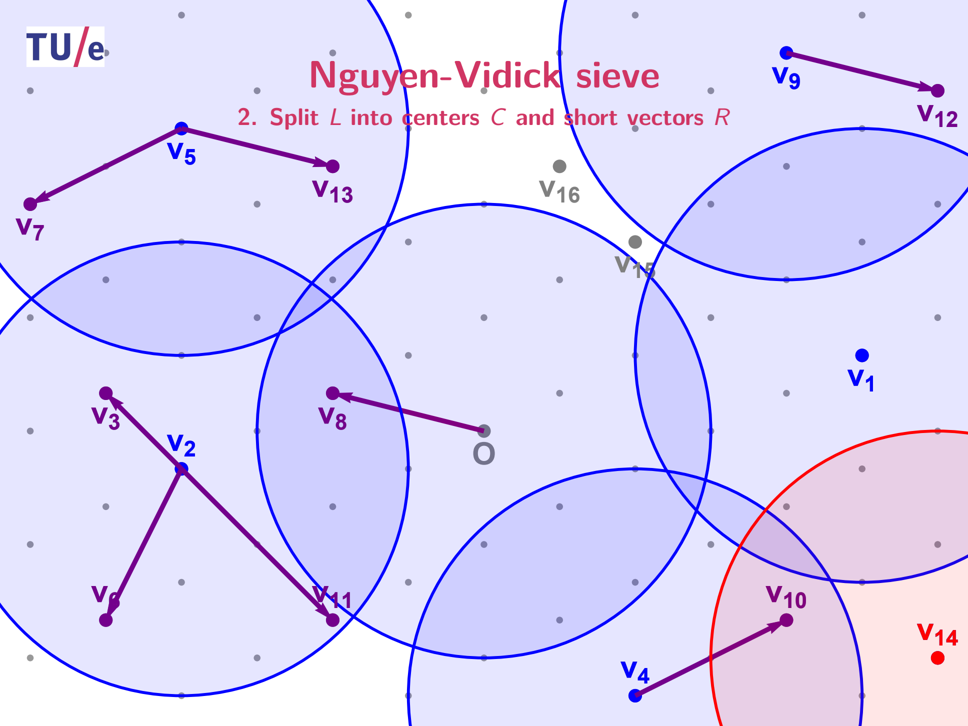
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



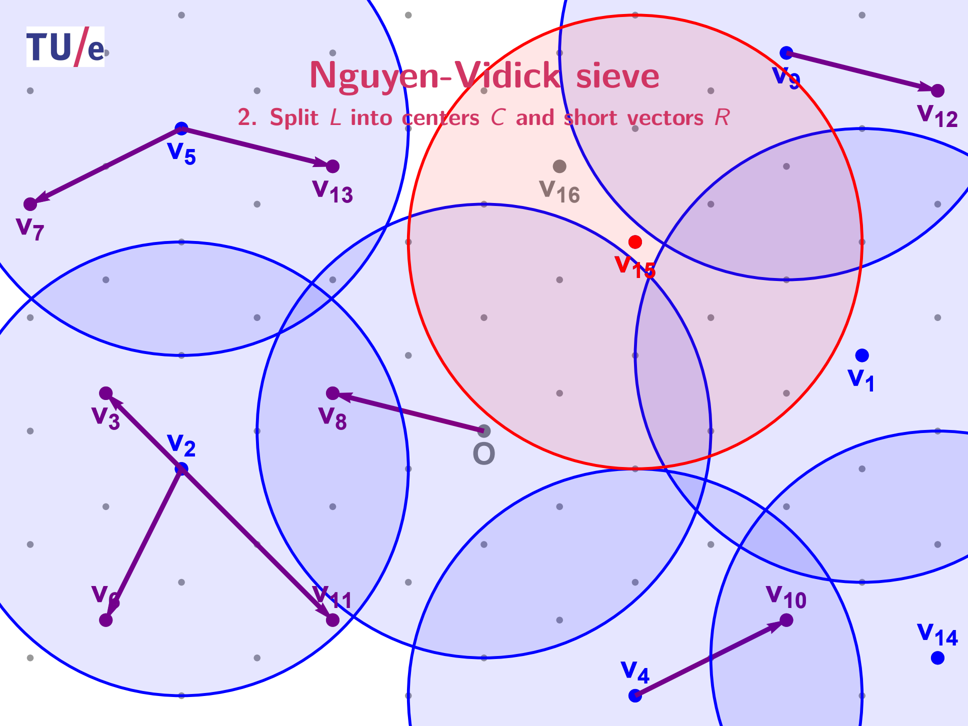
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



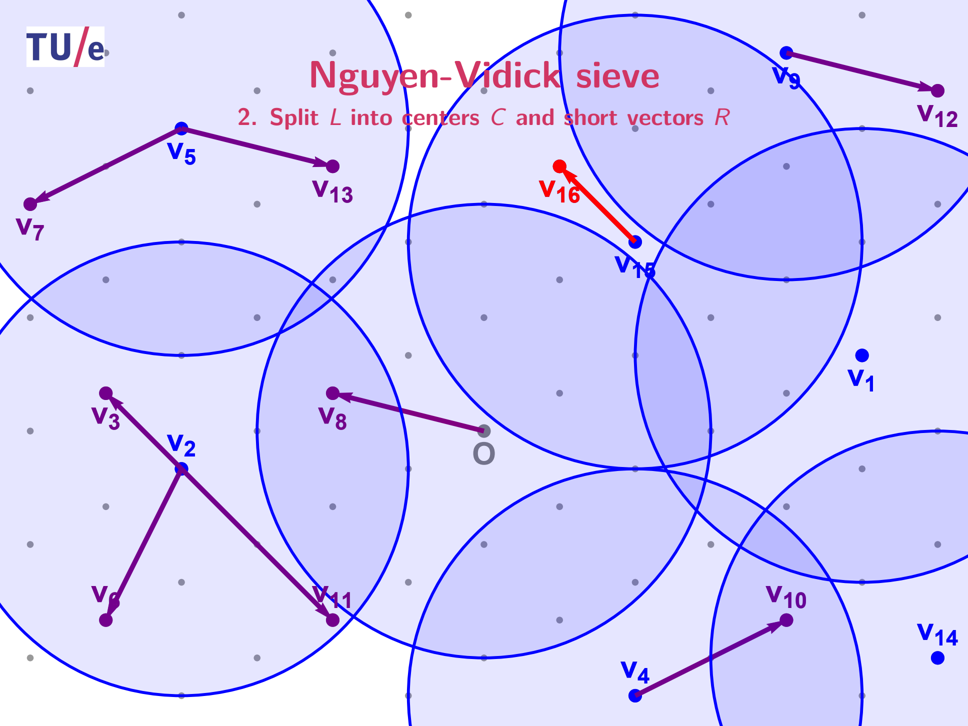
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



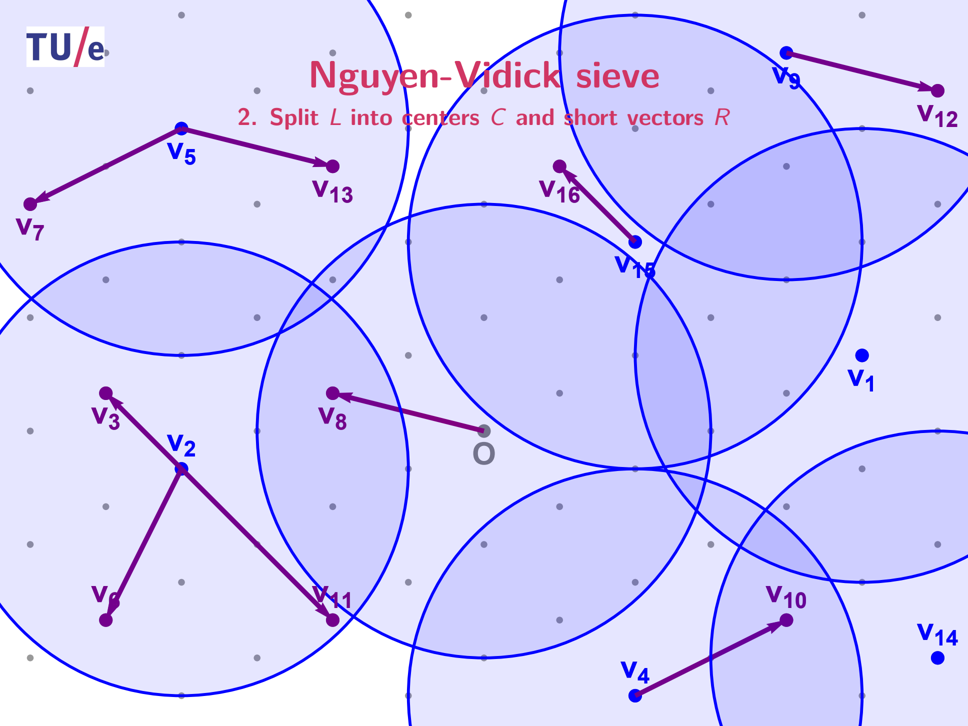
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



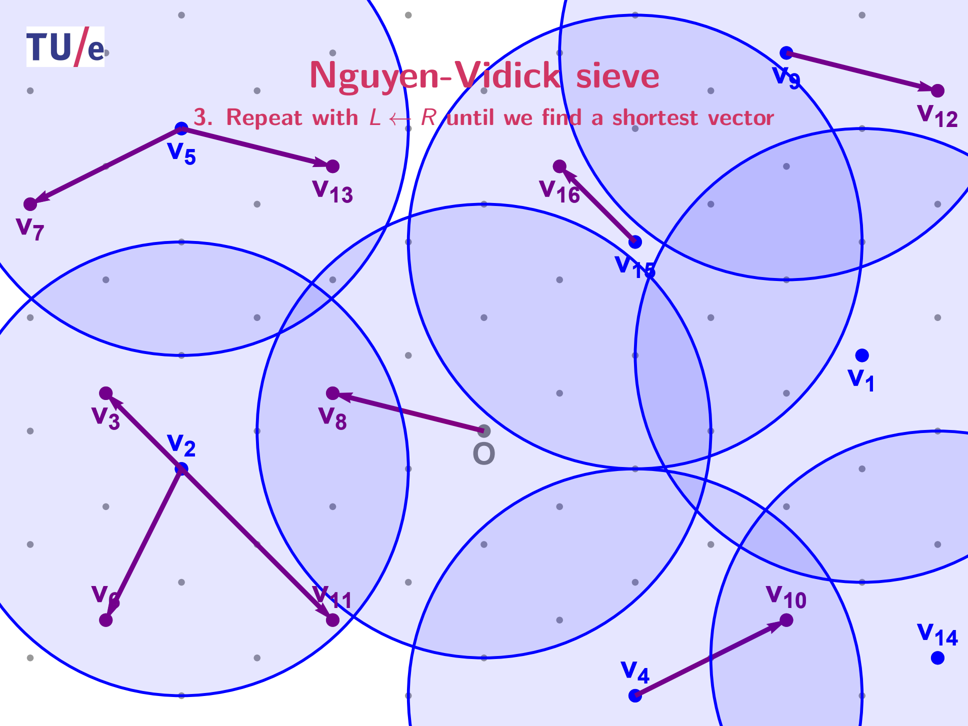
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



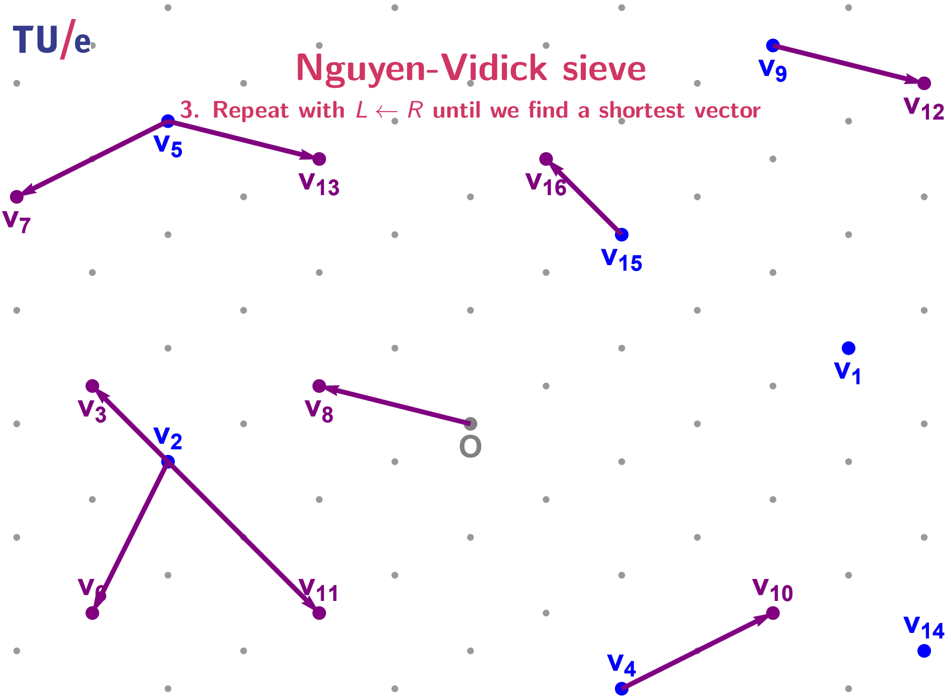
Nguyen-Vidick sieve

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



Nguyen-Vidick sieve

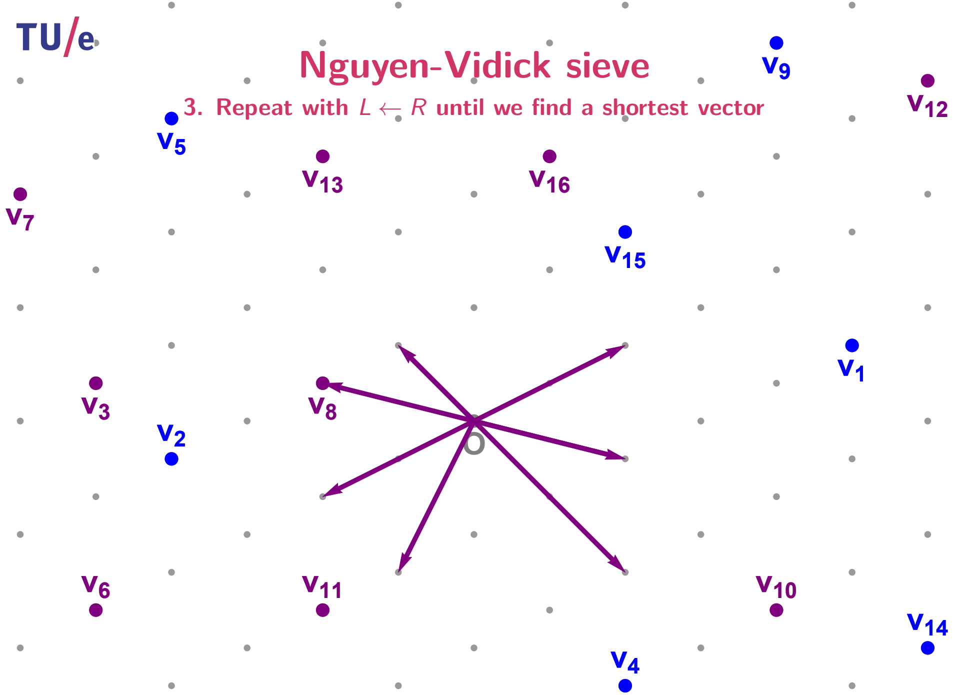
3. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

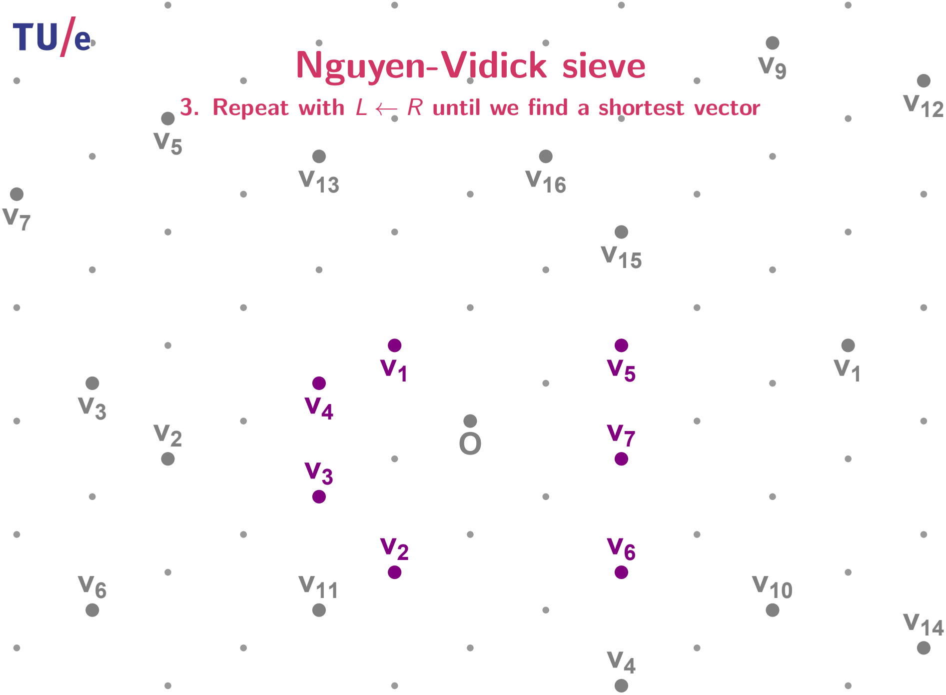
Nguyen-Vidick sieve

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

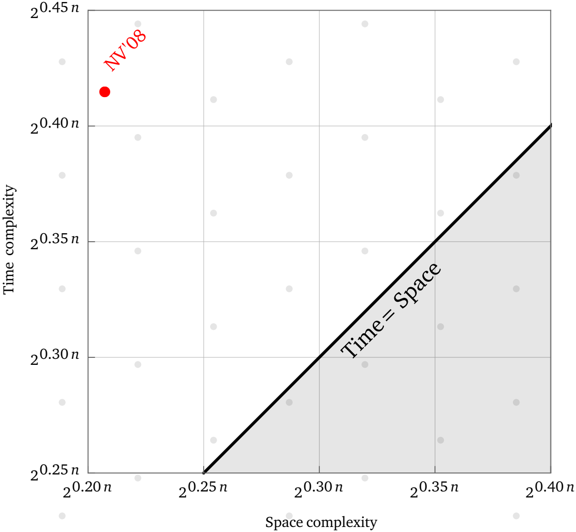
- Space complexity: $\sqrt{4/3}^n \approx 2^{0.21n+o(n)}$ vectors
  - Need $\sqrt{4/3}^n$ vectors to cover all corners of $\mathbb{R}^n$
- Time complexity: $(4/3)^n \approx 2^{0.42n+o(n)}$
  - Comparing a target vector to all centers: $2^{0.21n+o(n)}$
  - Repeating this for each list vector: $2^{0.21n+o(n)}$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

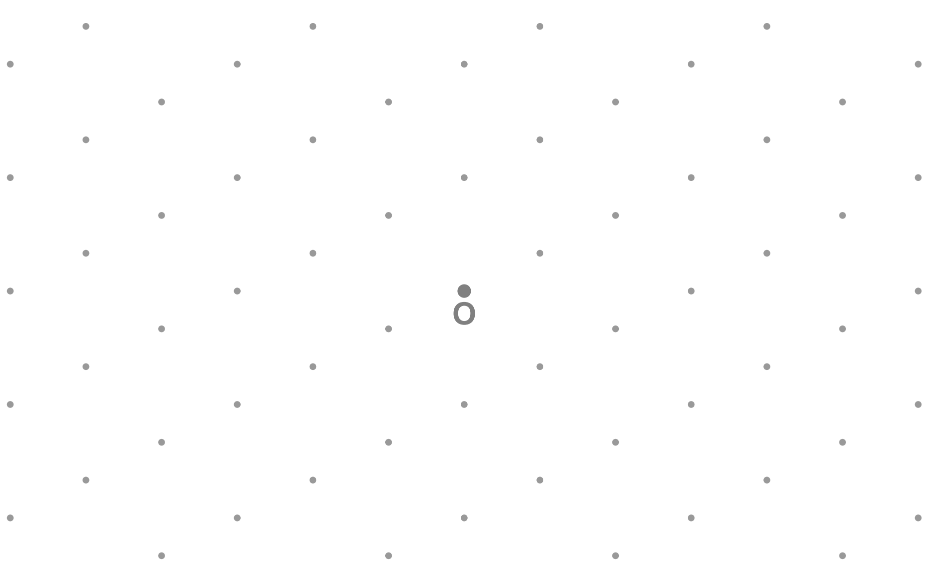
Nguyen-Vidick sieve

Space/time trade-off



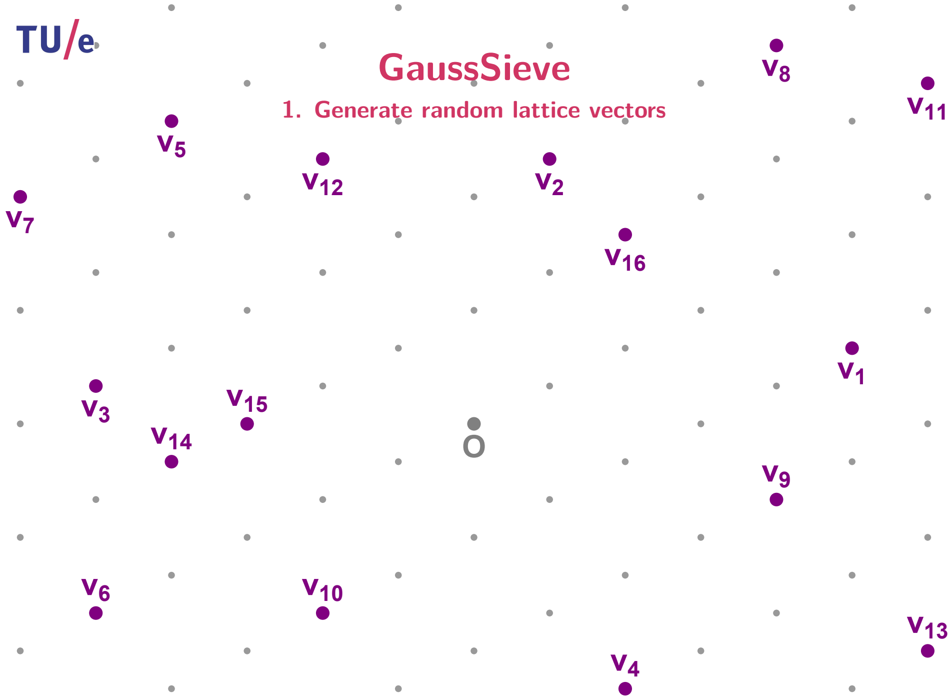
GaussSieve

1. Generate random lattice vectors



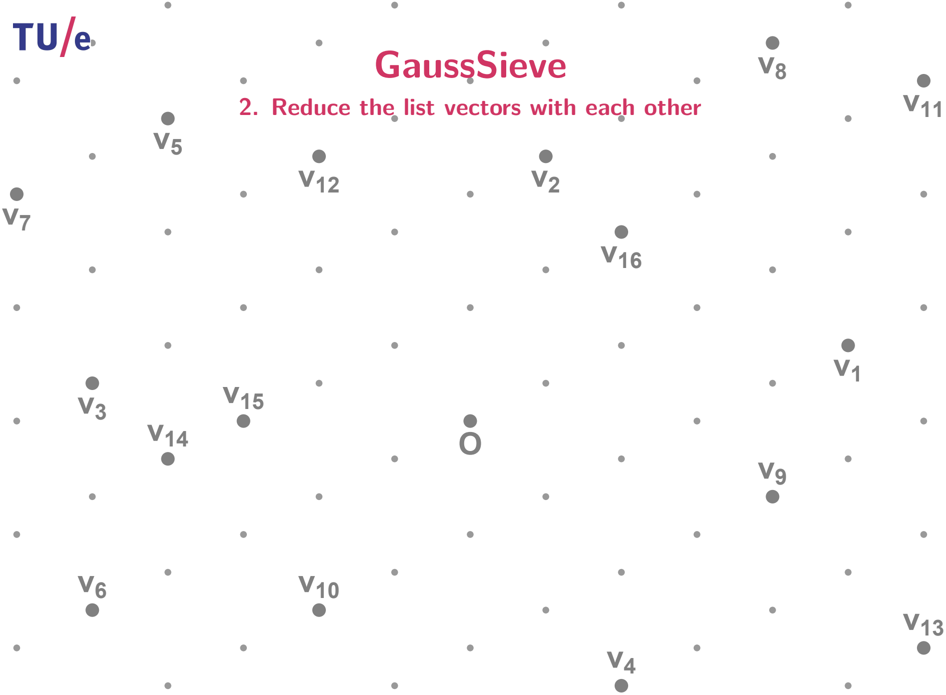
GaussSieve

1. Generate random lattice vectors



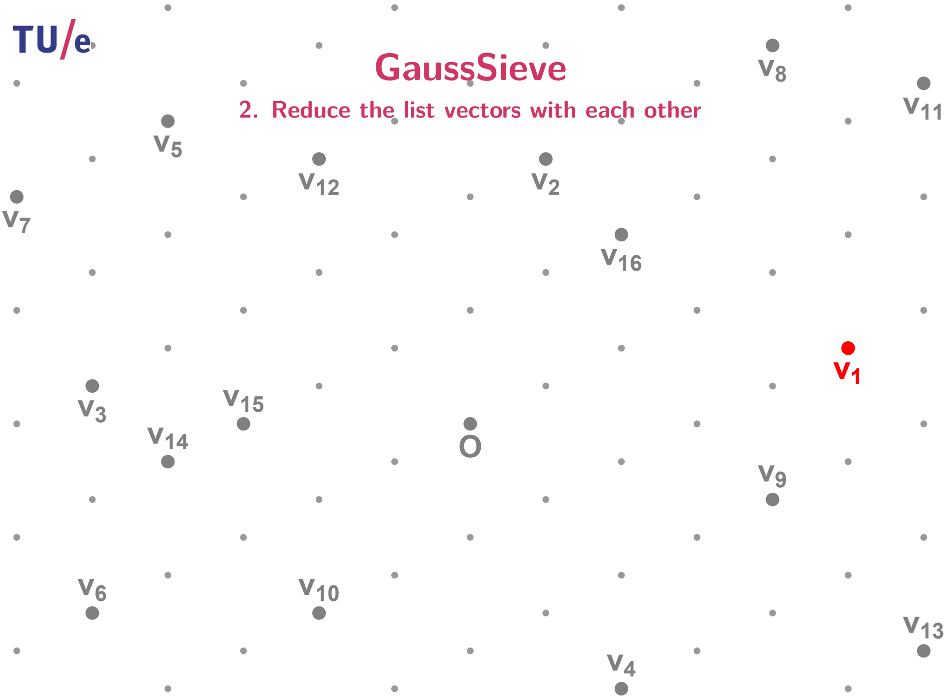
GaussSieve

2. Reduce the list vectors with each other



GaussSieve

2. Reduce the list vectors with each other

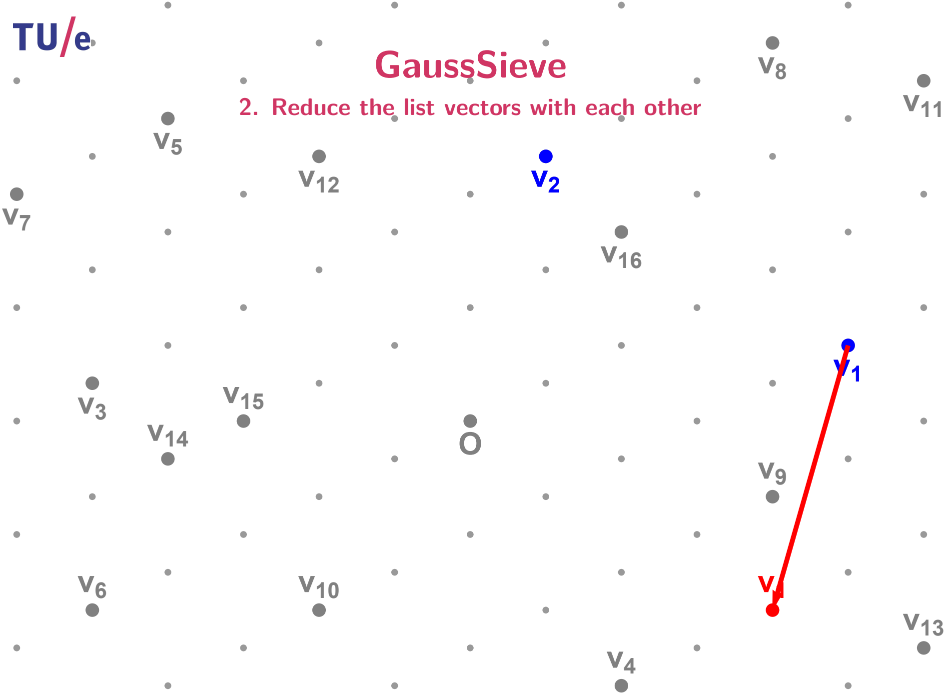


GaussSieve

2. Reduce the list vectors with each other

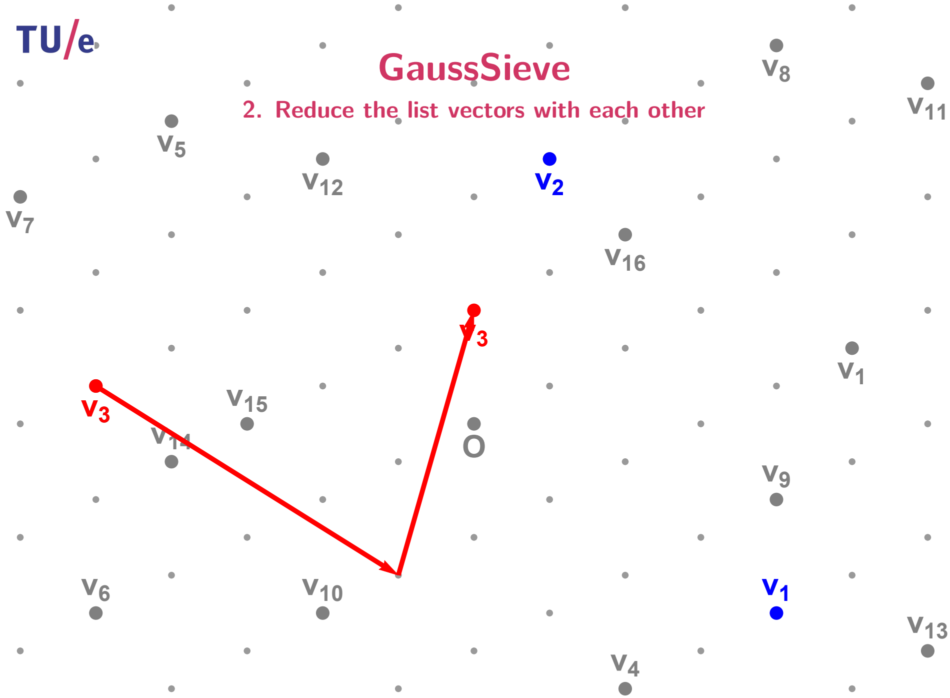
GaussSieve

2. Reduce the list vectors with each other



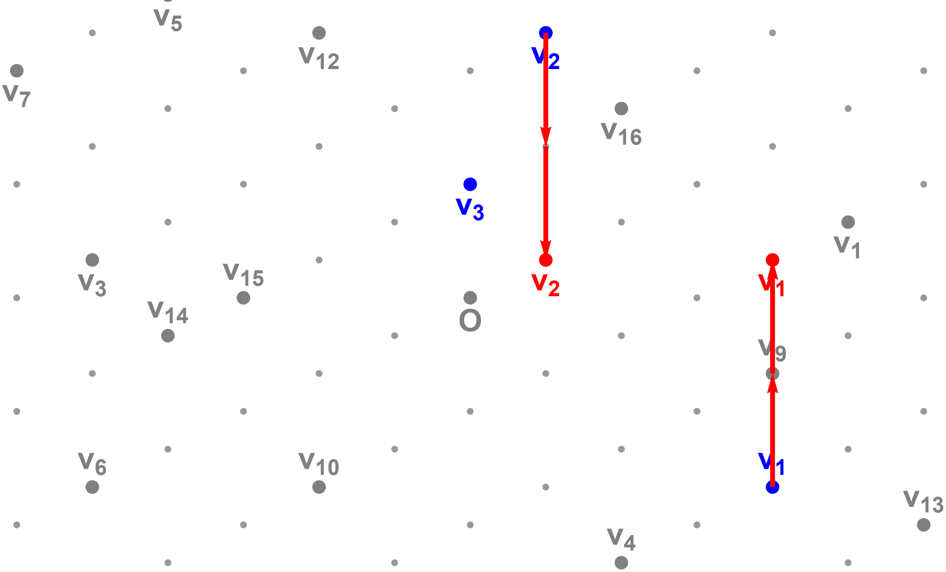
GaussSieve

2. Reduce the list vectors with each other



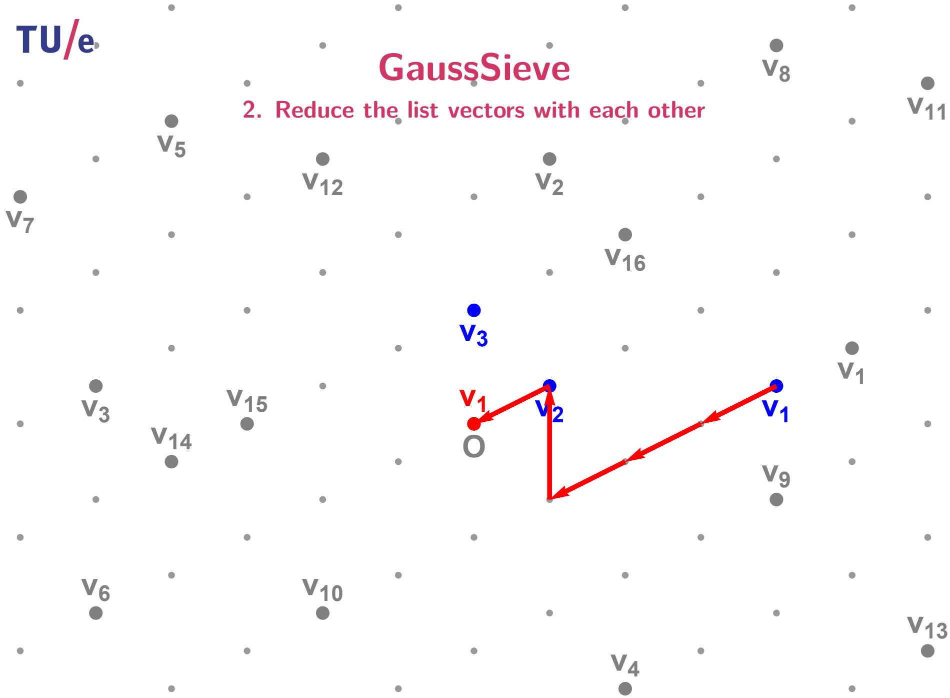
GaussSieve

2. Reduce the list vectors with each other



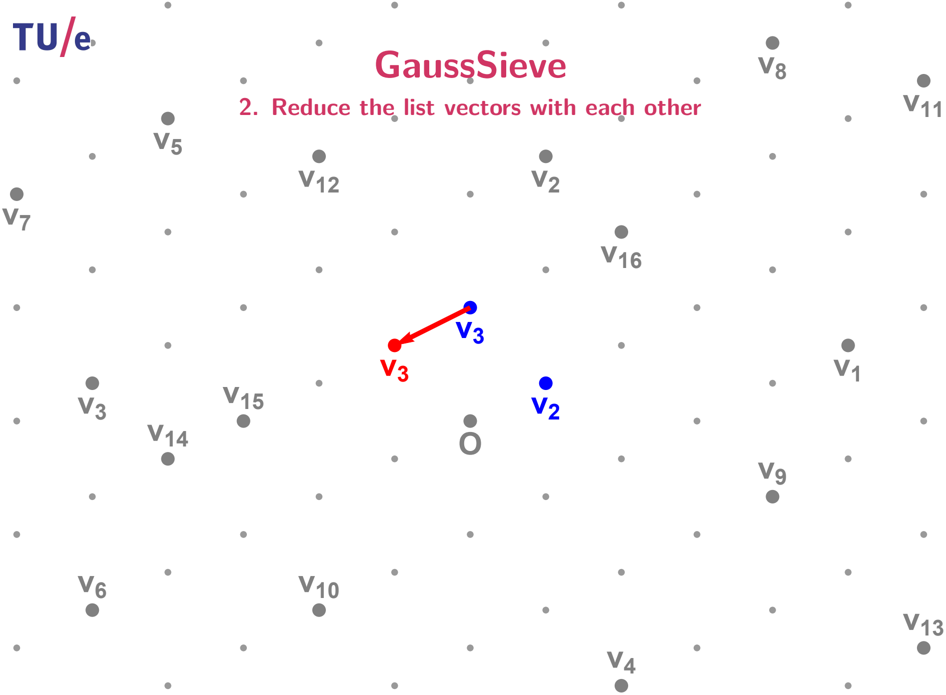
GaussSieve

2. Reduce the list vectors with each other



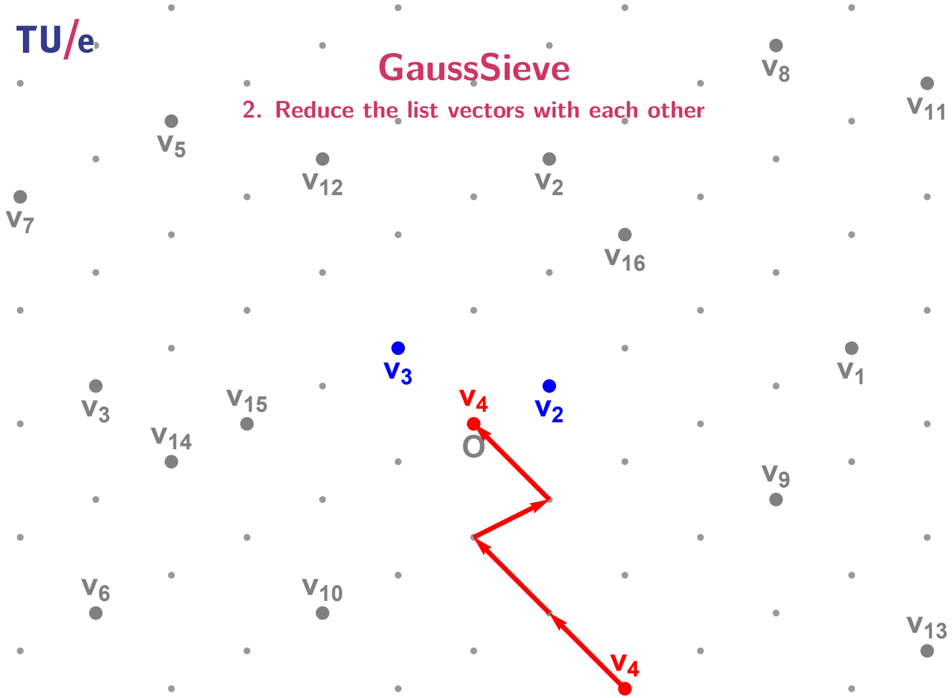
GaussSieve

2. Reduce the list vectors with each other



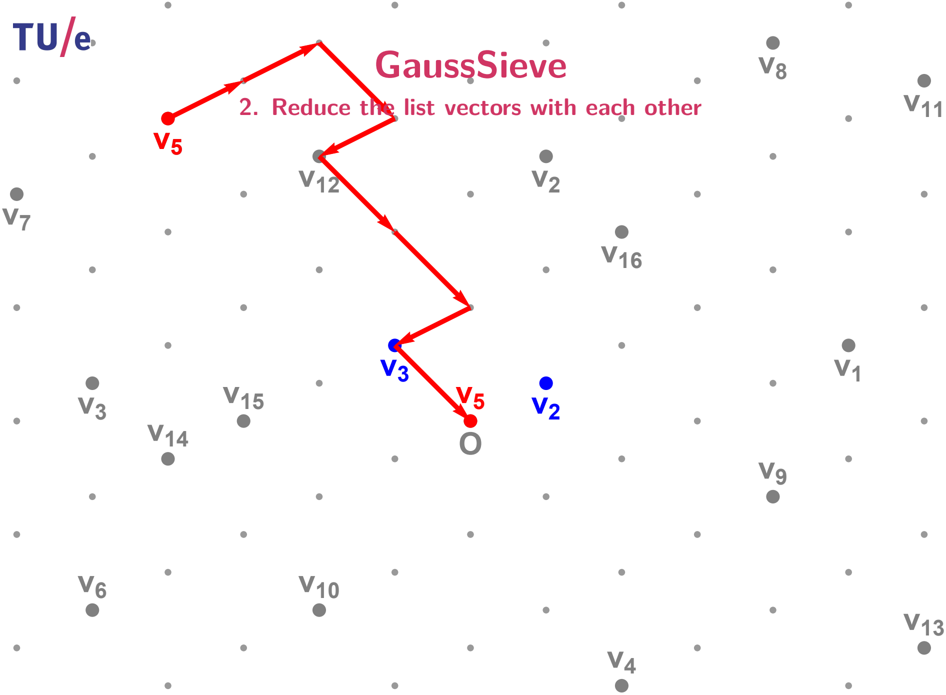
GaussSieve

2. Reduce the list vectors with each other



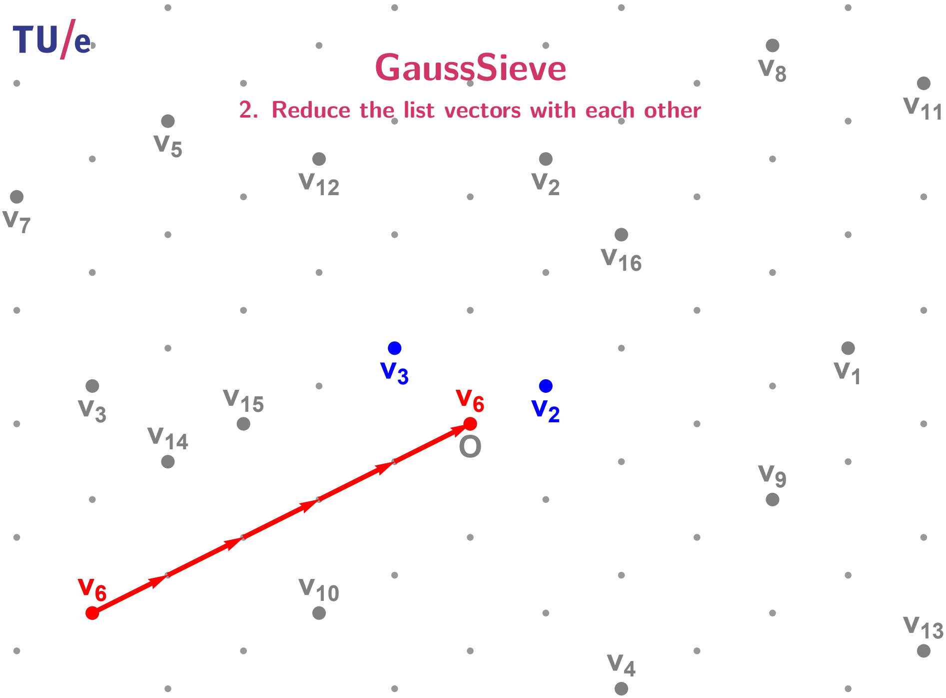
GaussSieve

2. Reduce the list vectors with each other



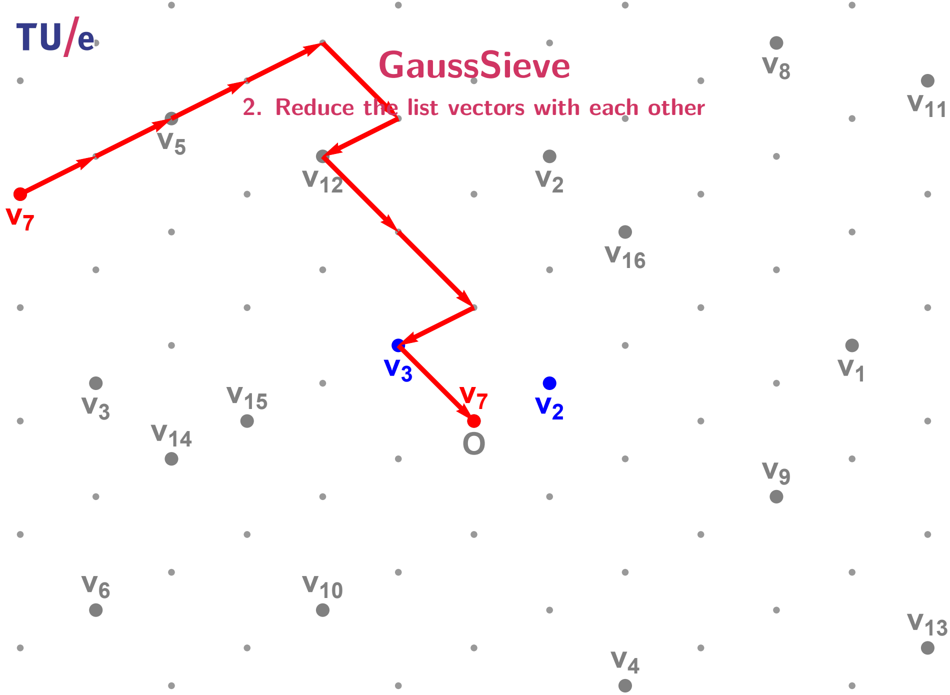
GaussSieve

2. Reduce the list vectors with each other



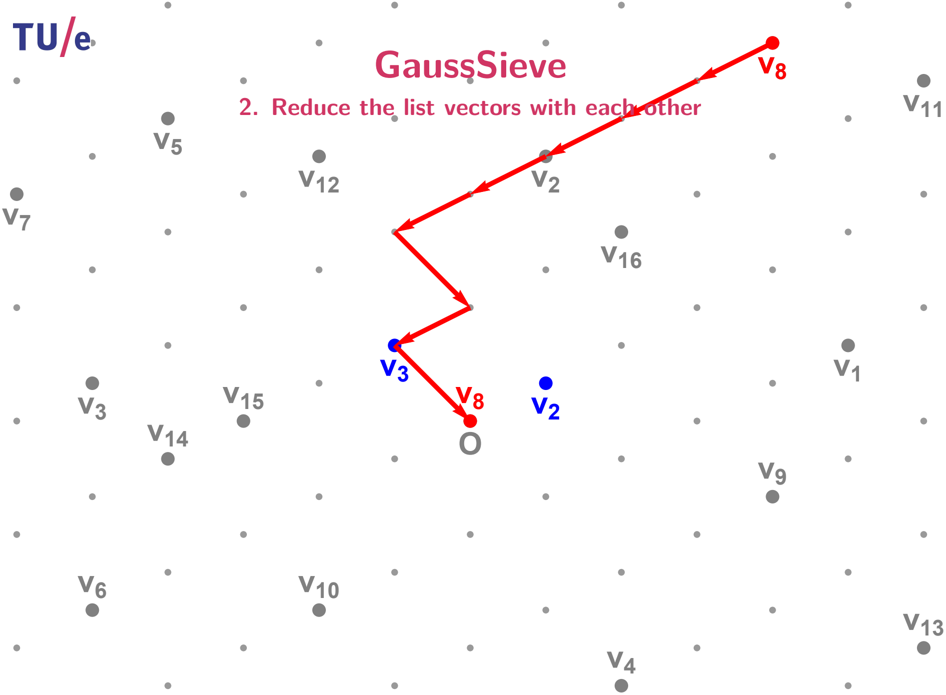
GaussSieve

2. Reduce the list vectors with each other



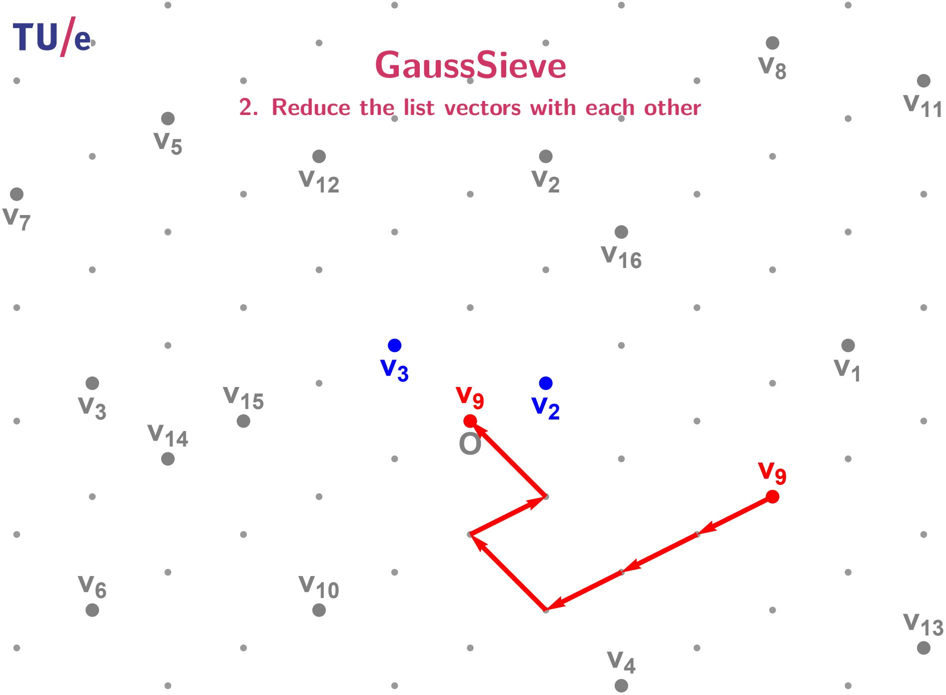
GaussSieve

2. Reduce the list vectors with each other



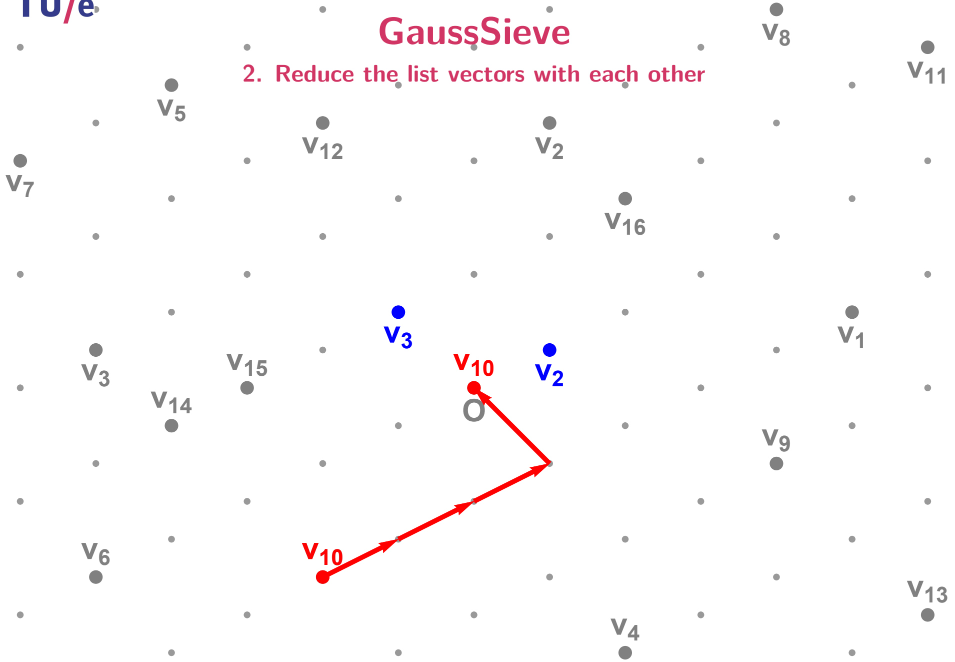
GaussSieve

2. Reduce the list vectors with each other



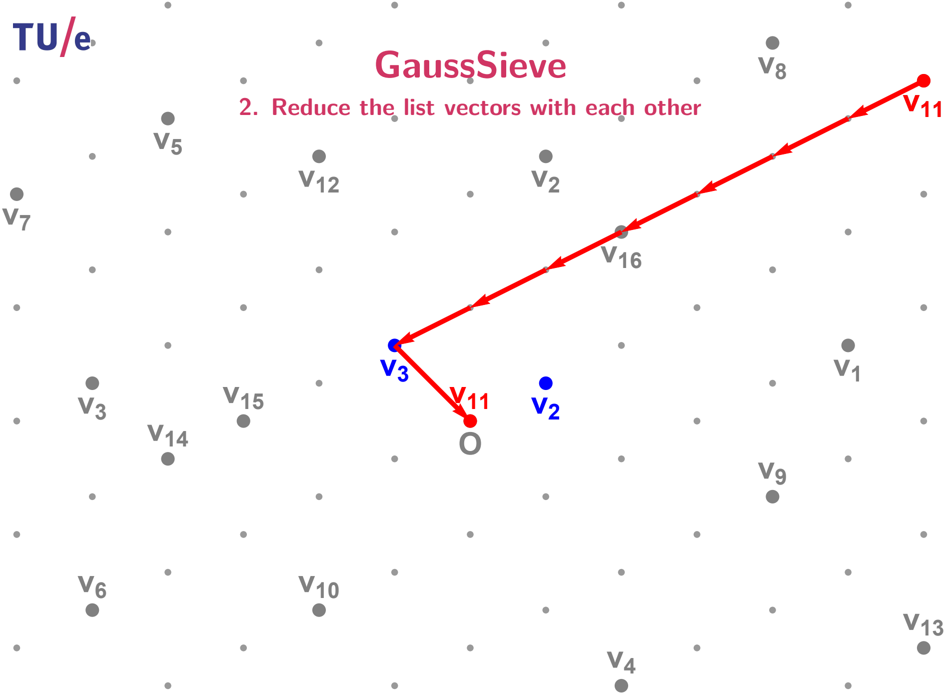
GaussSieve

2. Reduce the list vectors with each other



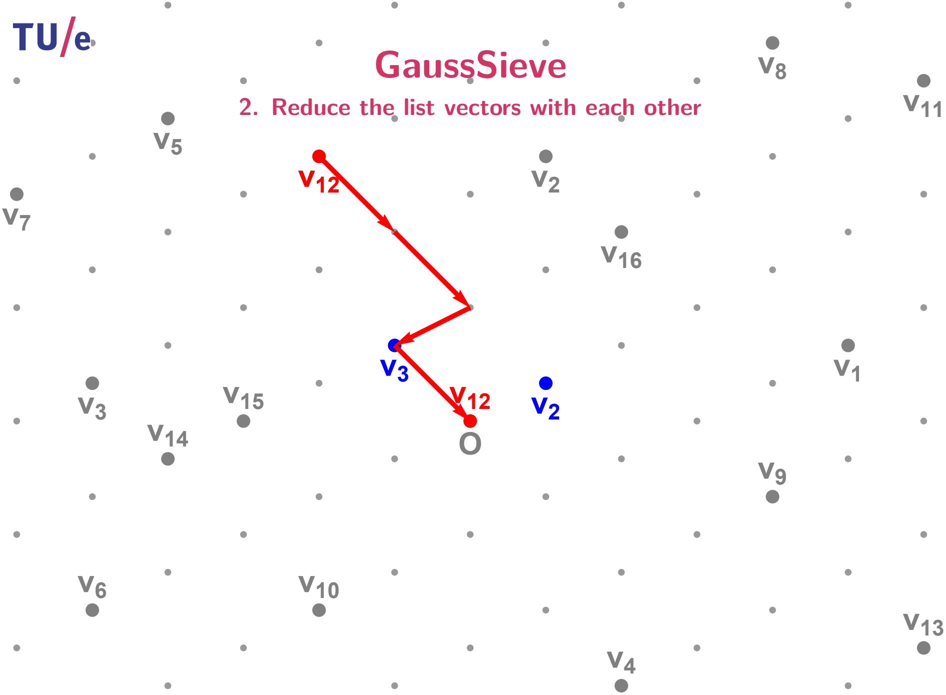
GaussSieve

2. Reduce the list vectors with each other



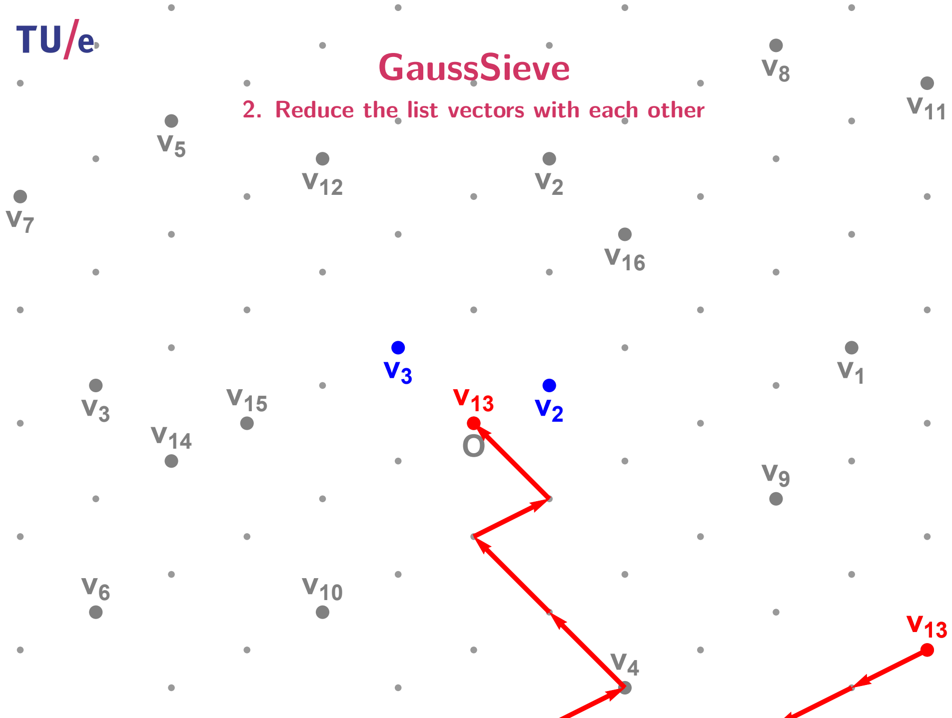
GaussSieve

2. Reduce the list vectors with each other



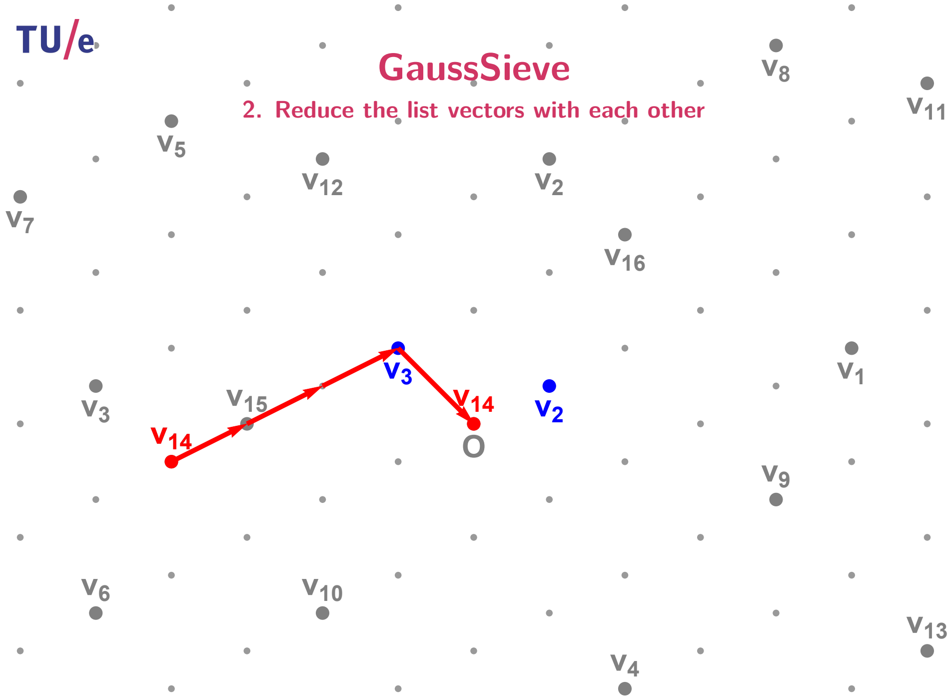
GaussSieve

2. Reduce the list vectors with each other



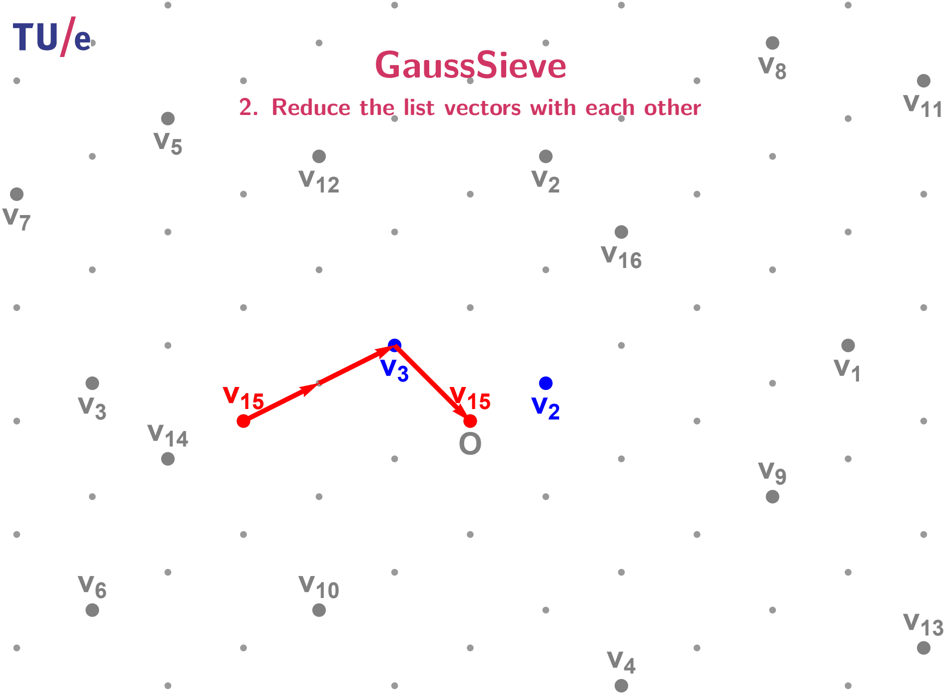
GaussSieve

2. Reduce the list vectors with each other



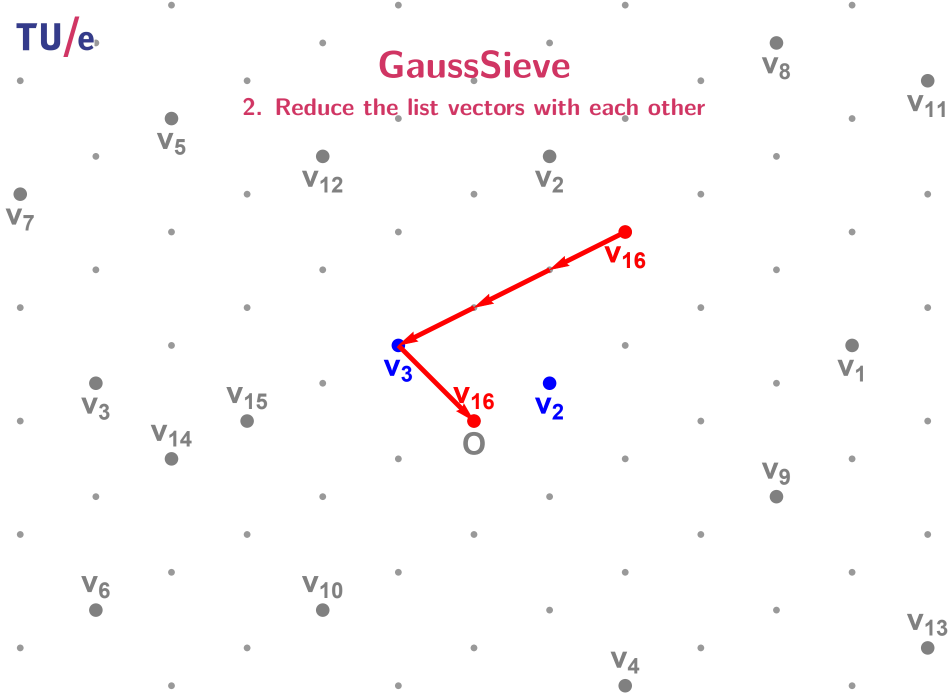
GaussSieve

2. Reduce the list vectors with each other



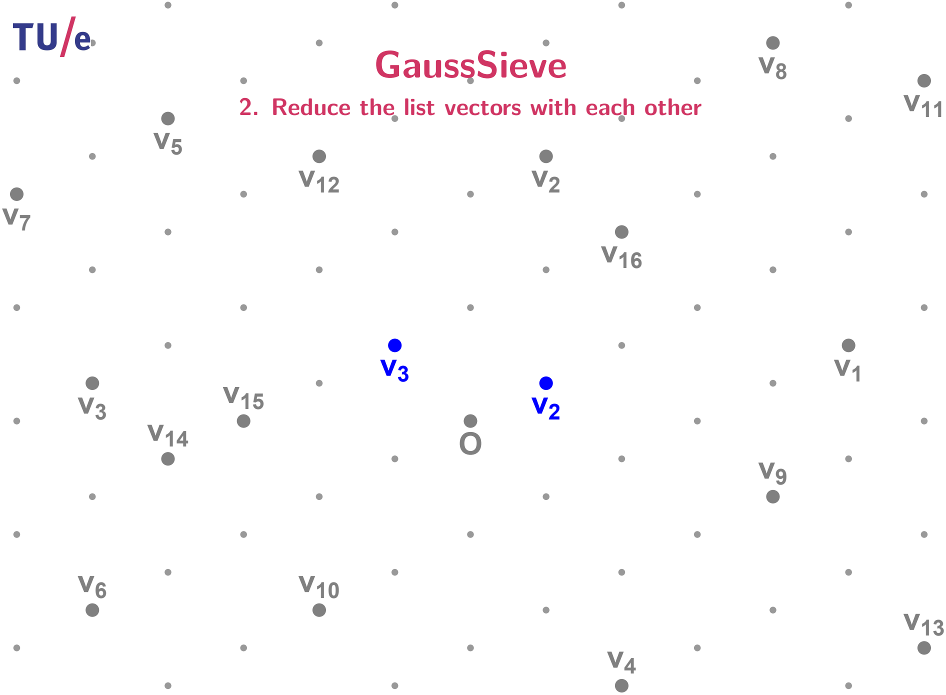
GaussSieve

2. Reduce the list vectors with each other



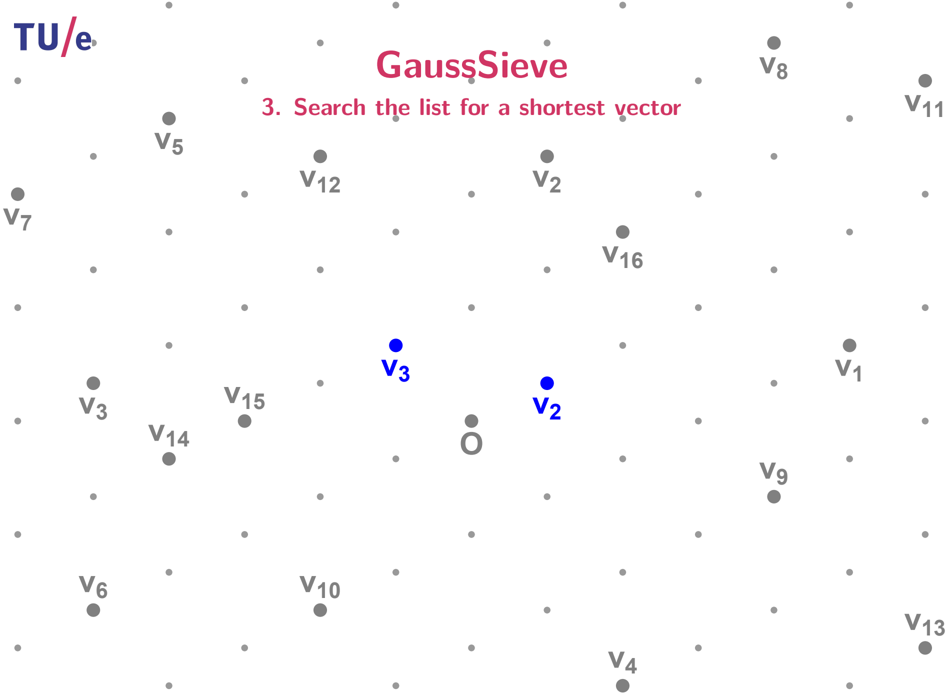
GaussSieve

2. Reduce the list vectors with each other



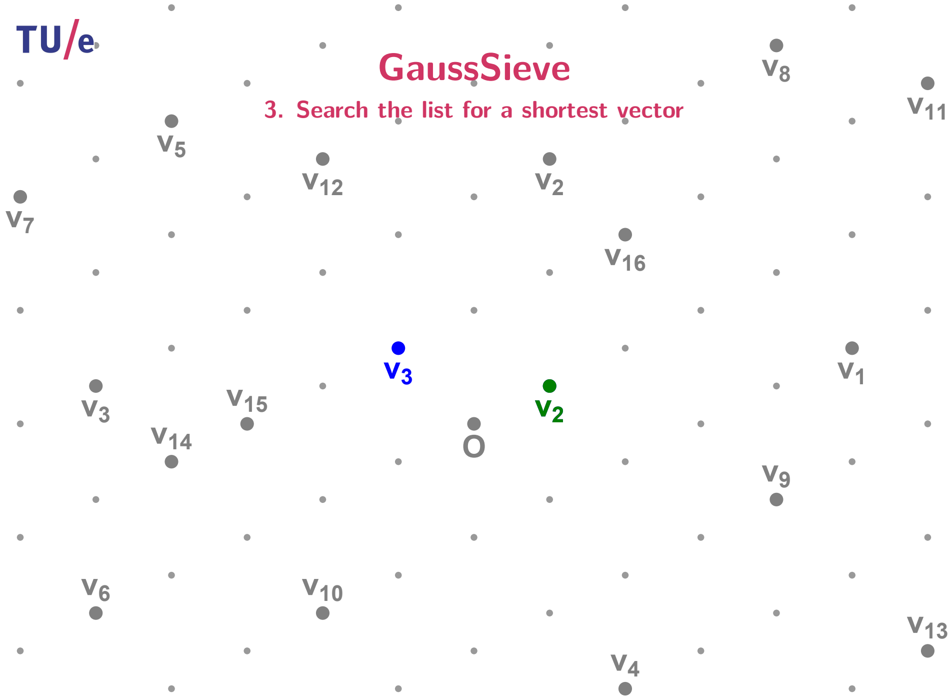
GaussSieve

3. Search the list for a shortest vector



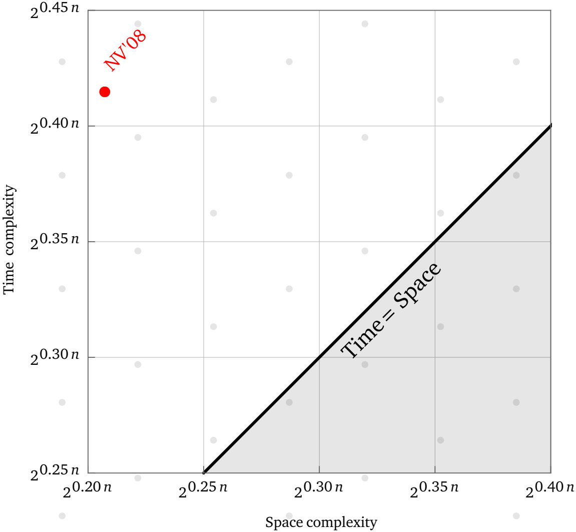
GaussSieve

3. Search the list for a shortest vector



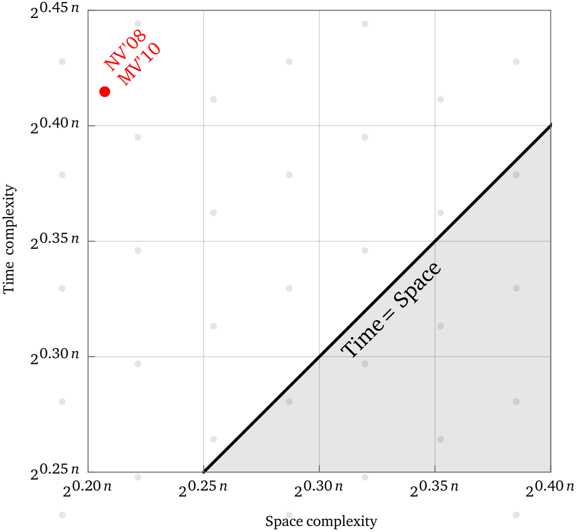
GaussSieve

Space/time trade-off



GaussSieve

Space/time trade-off

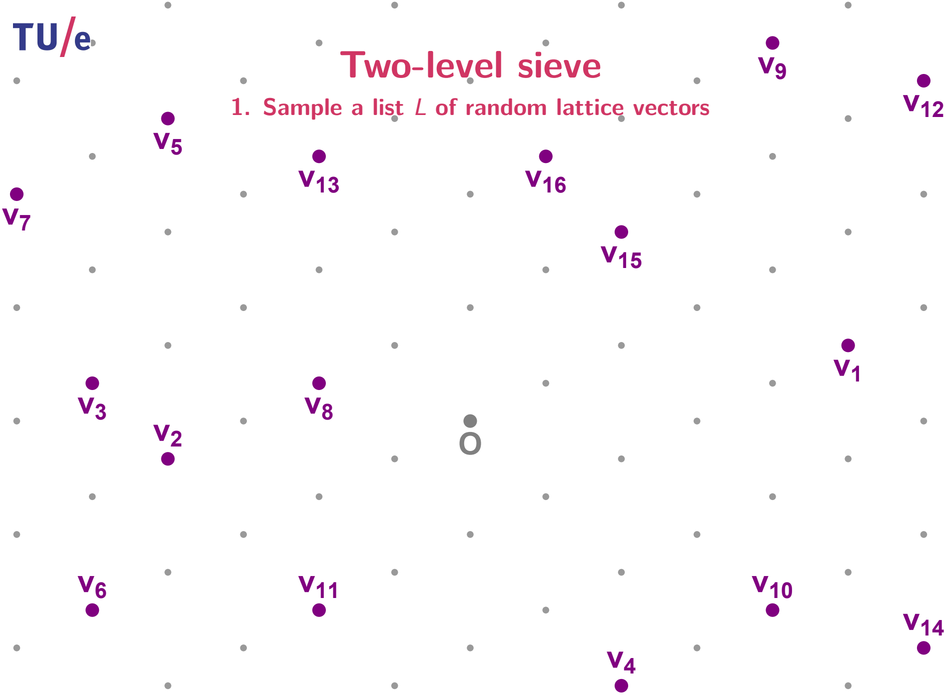


Two-level sieve

1. Sample a list L of random lattice vectors

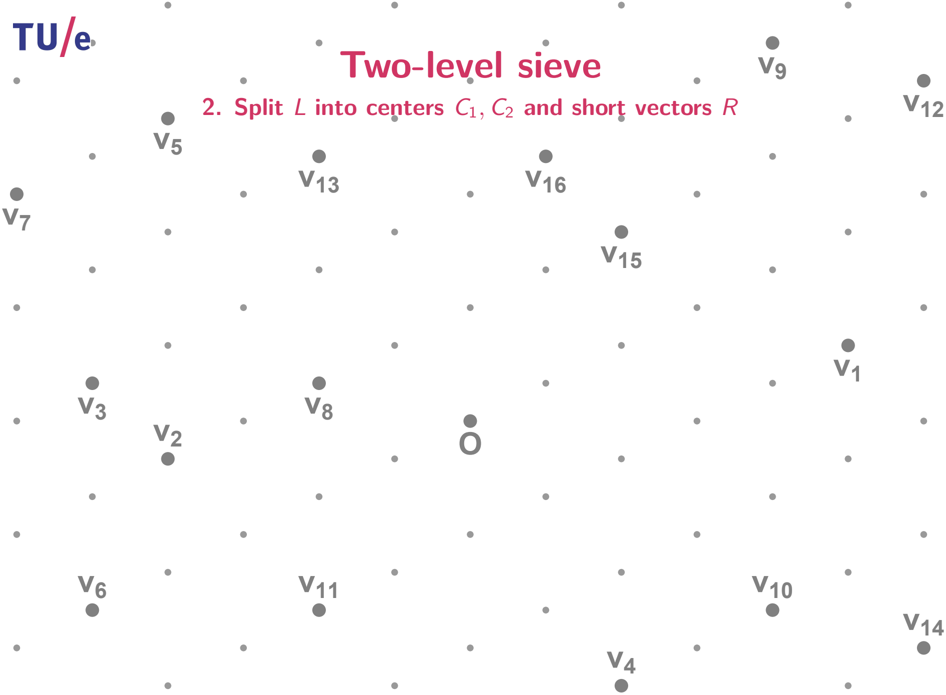
Two-level sieve

1. Sample a list L of random lattice vectors



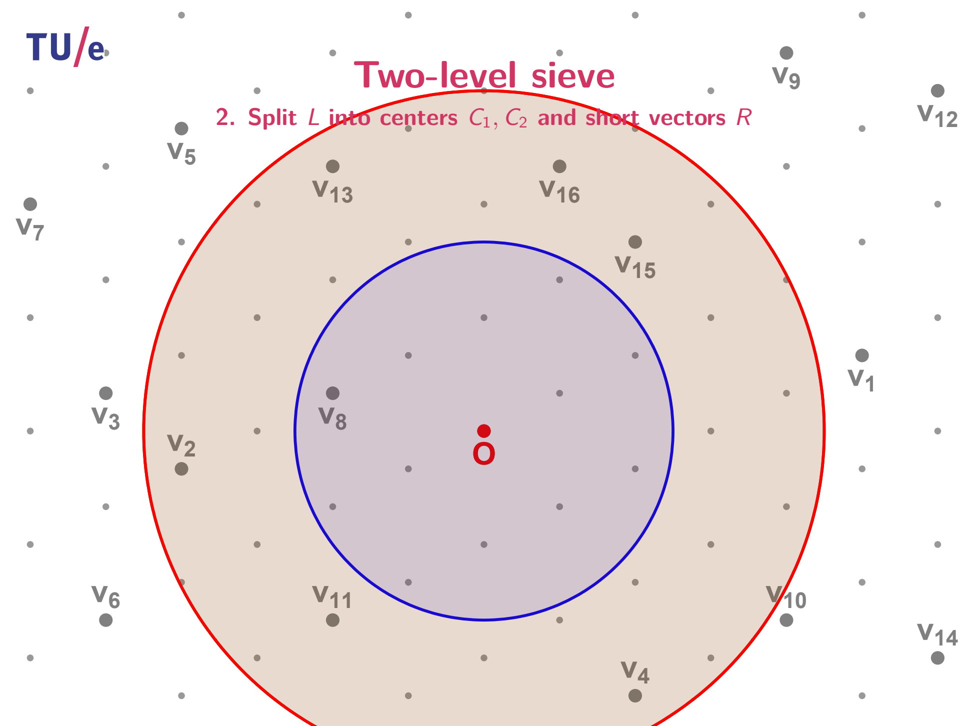
Two-level sieve

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



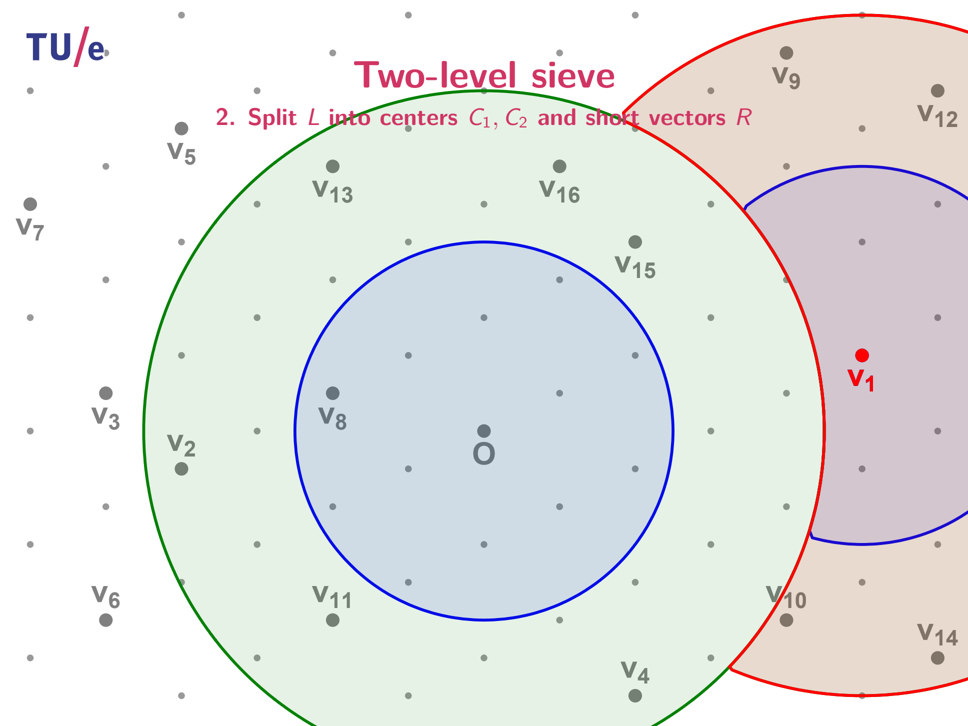
Two-level sieve

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



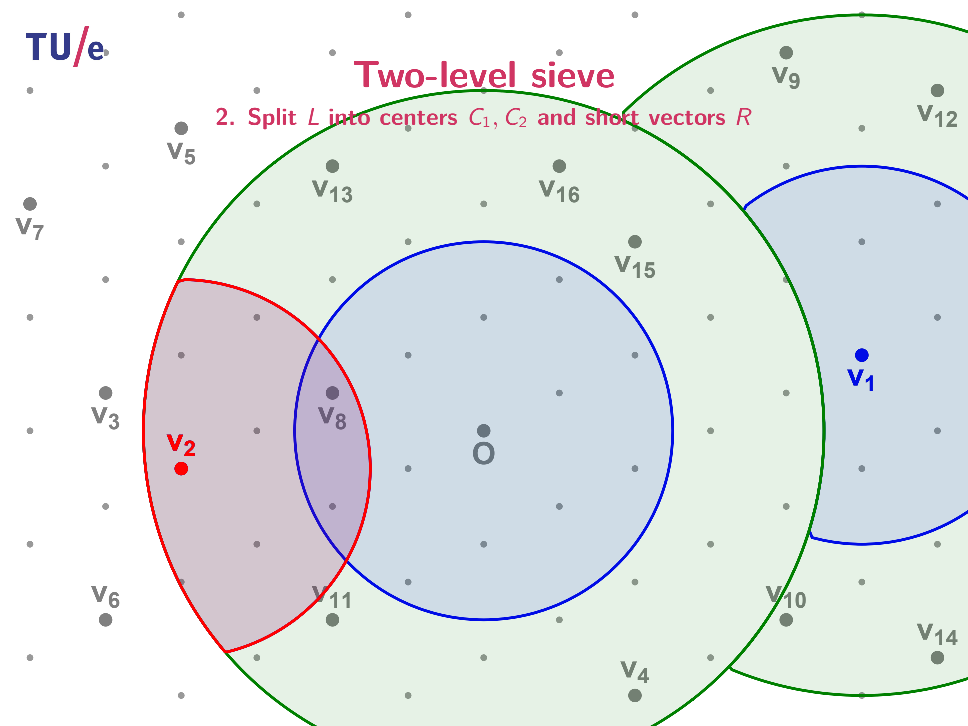
Two-level sieve

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



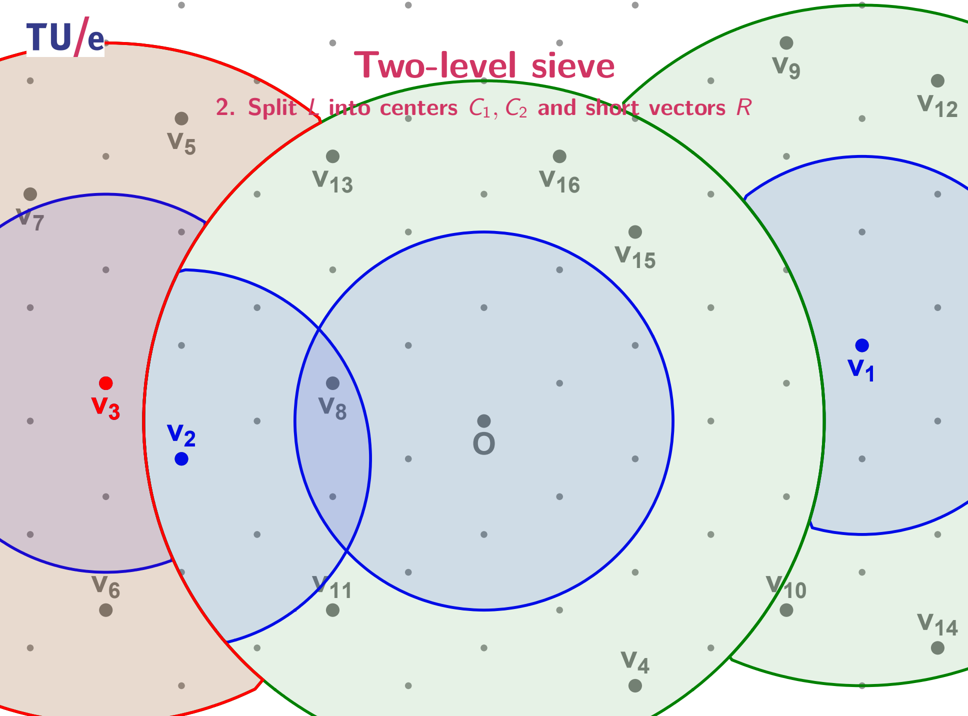
Two-level sieve

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



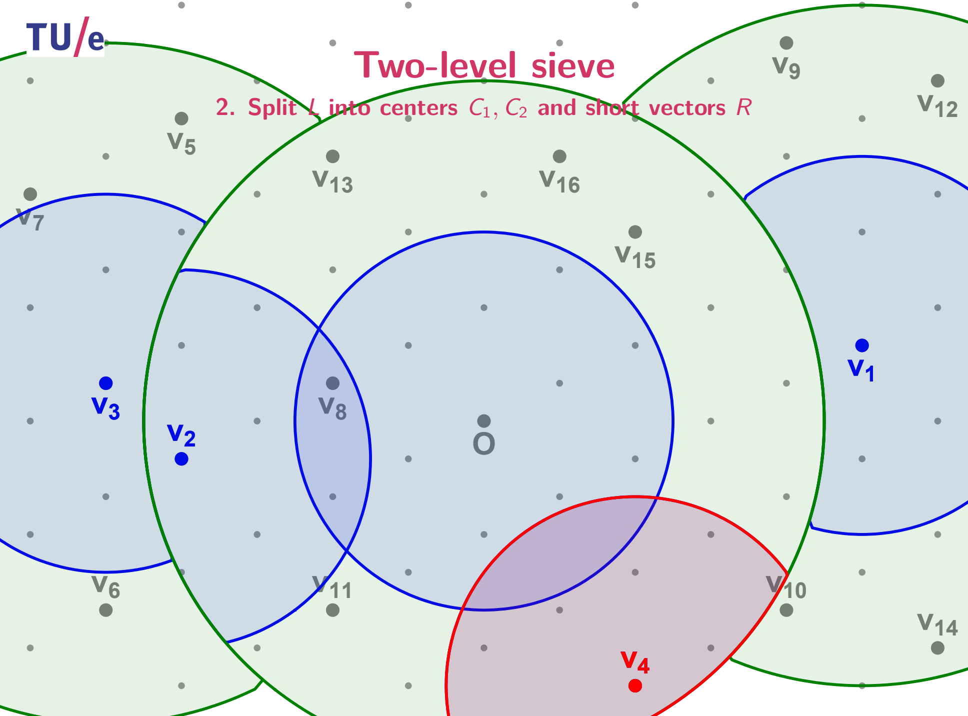
Two-level sieve

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



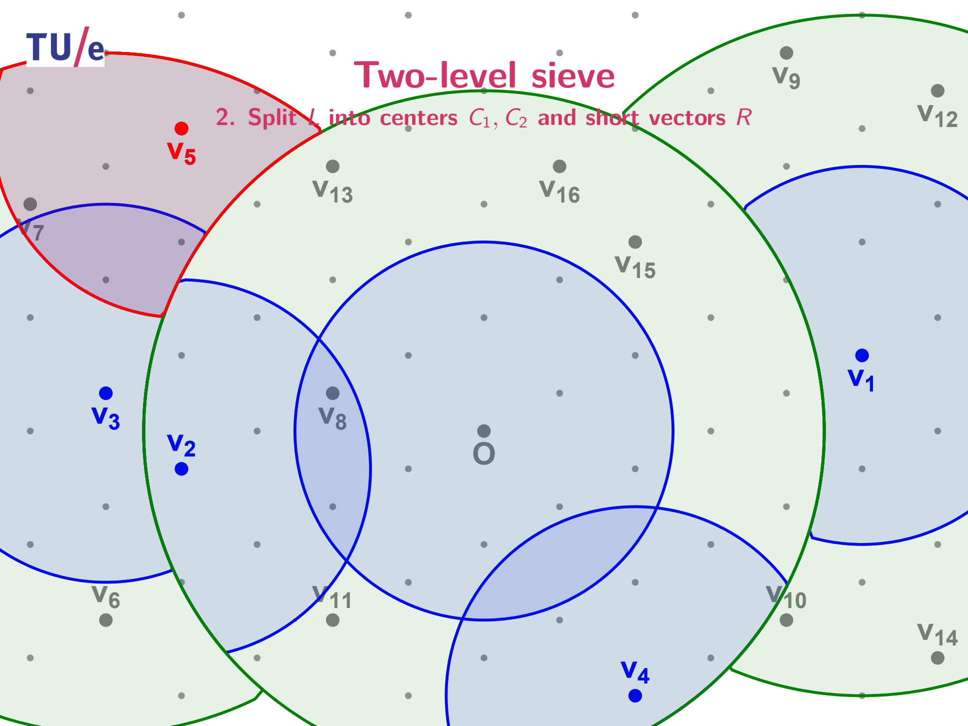
Two-level sieve

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



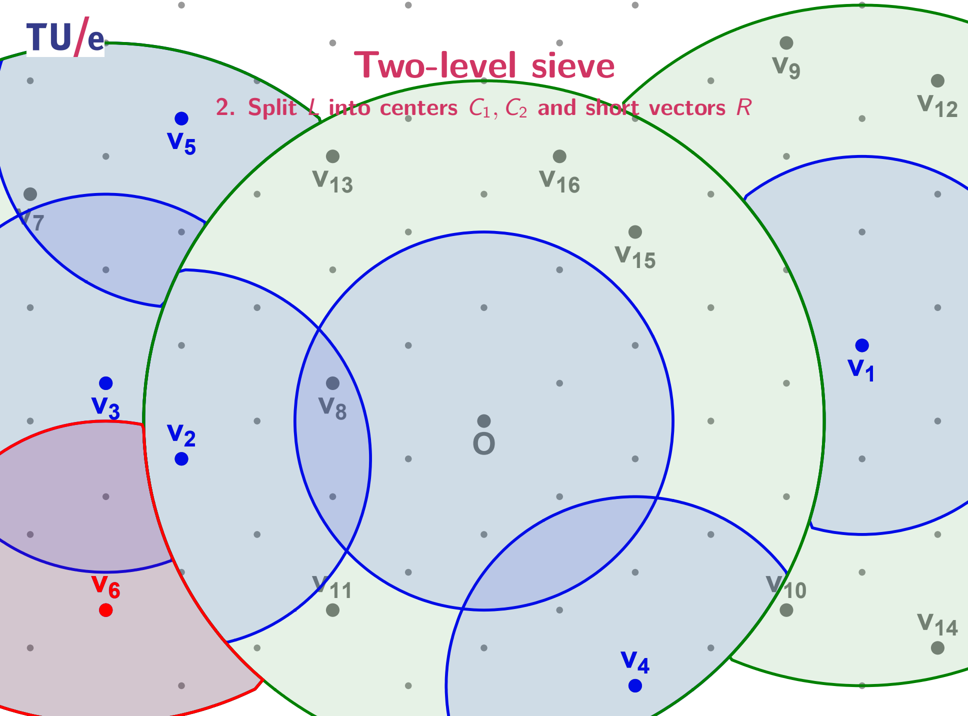
Two-level sieve

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



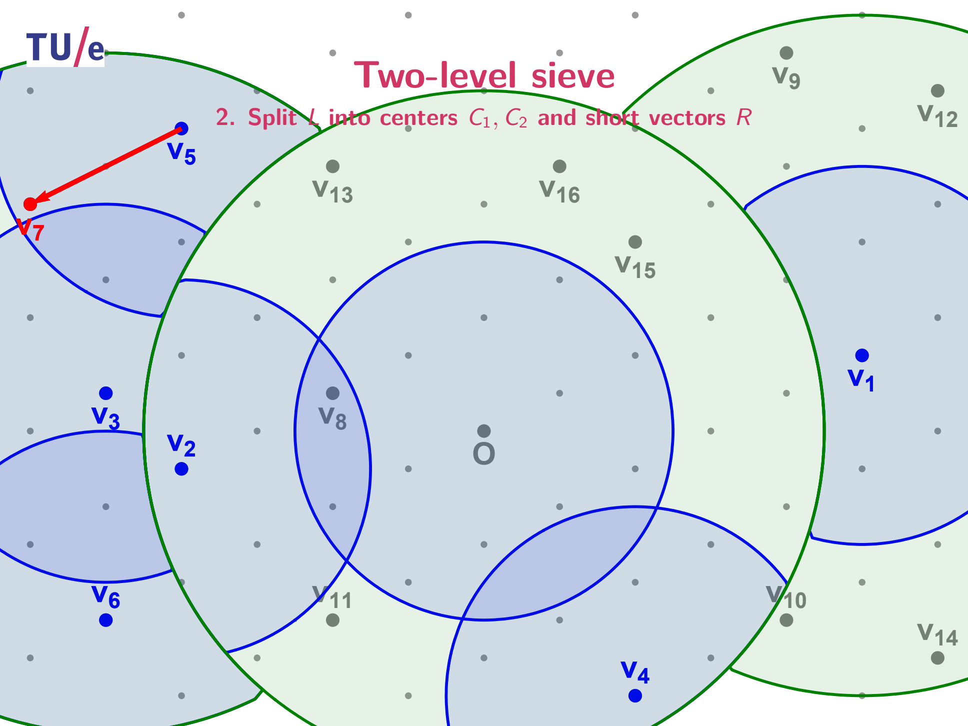
Two-level sieve

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



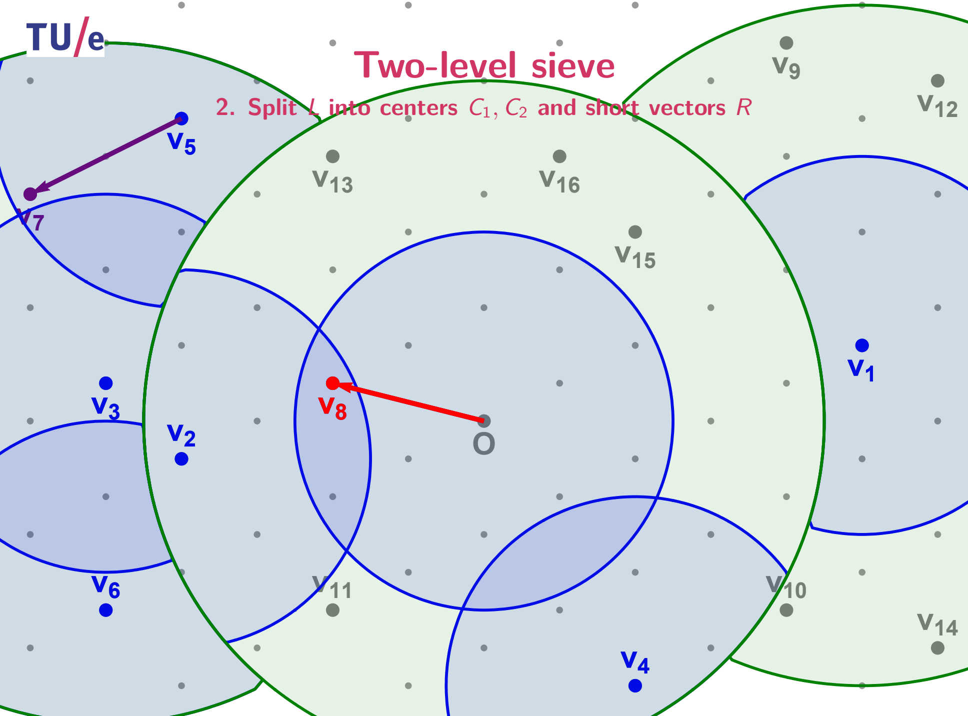
Two-level sieve

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



Two-level sieve

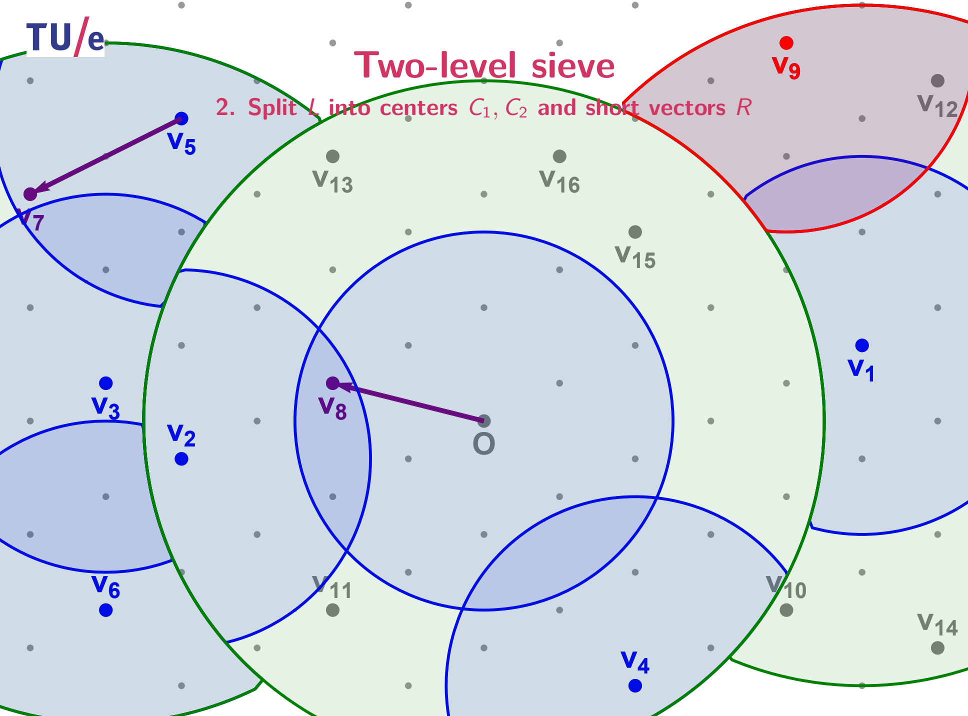
2. Split L into centers C_1, C_2 and short vectors R



TU/e

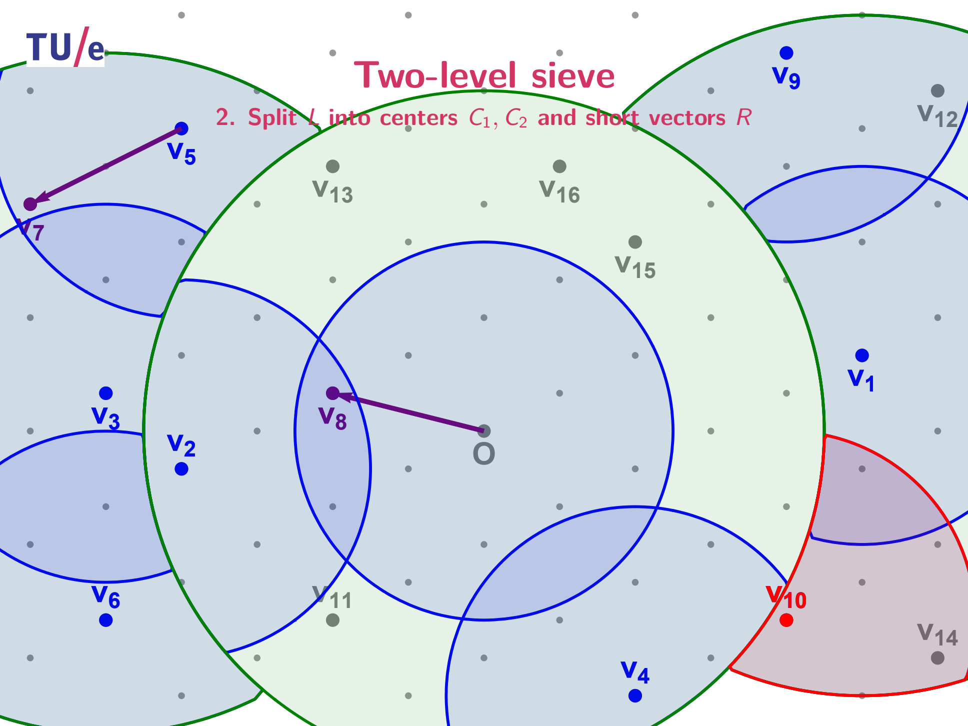
Two-level sieve

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



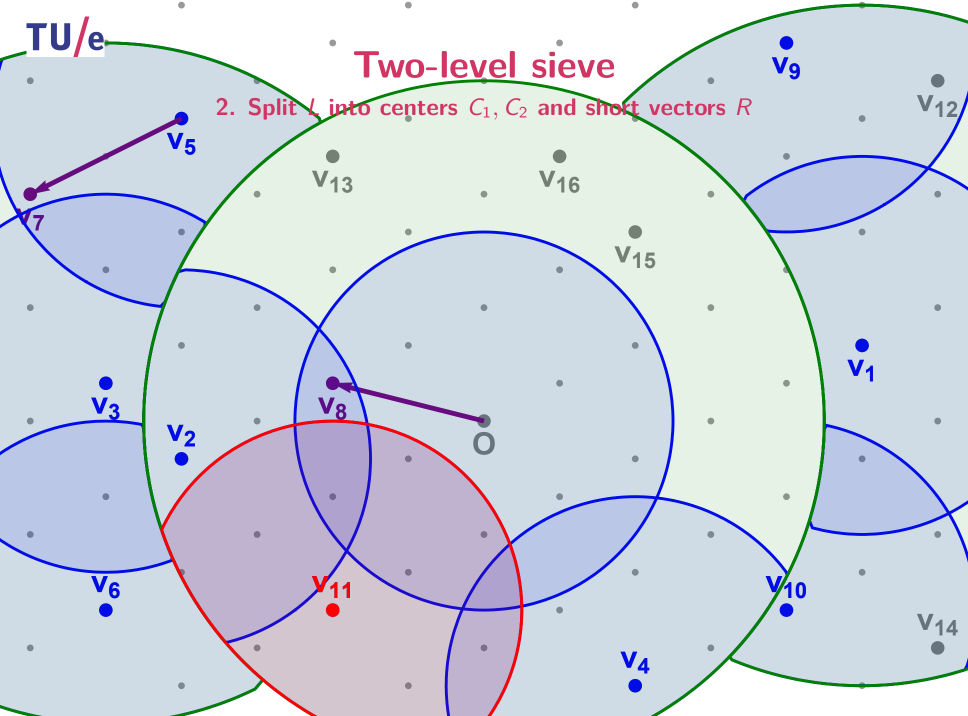
Two-level sieve

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



Two-level sieve

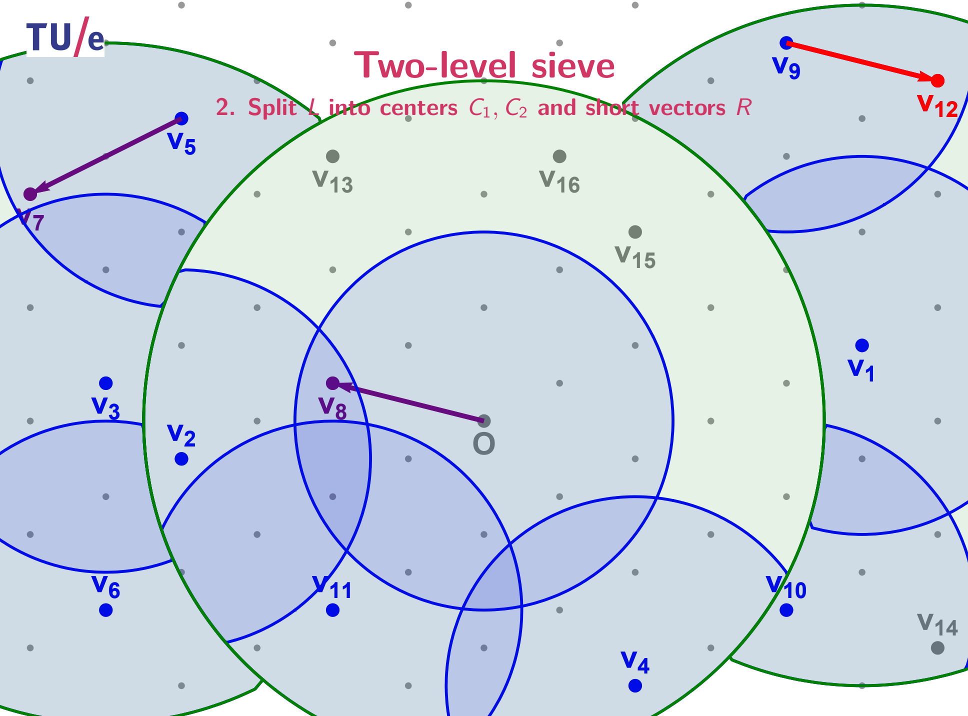
2. Split L into centers C_1, C_2 and short vectors R



TU/e

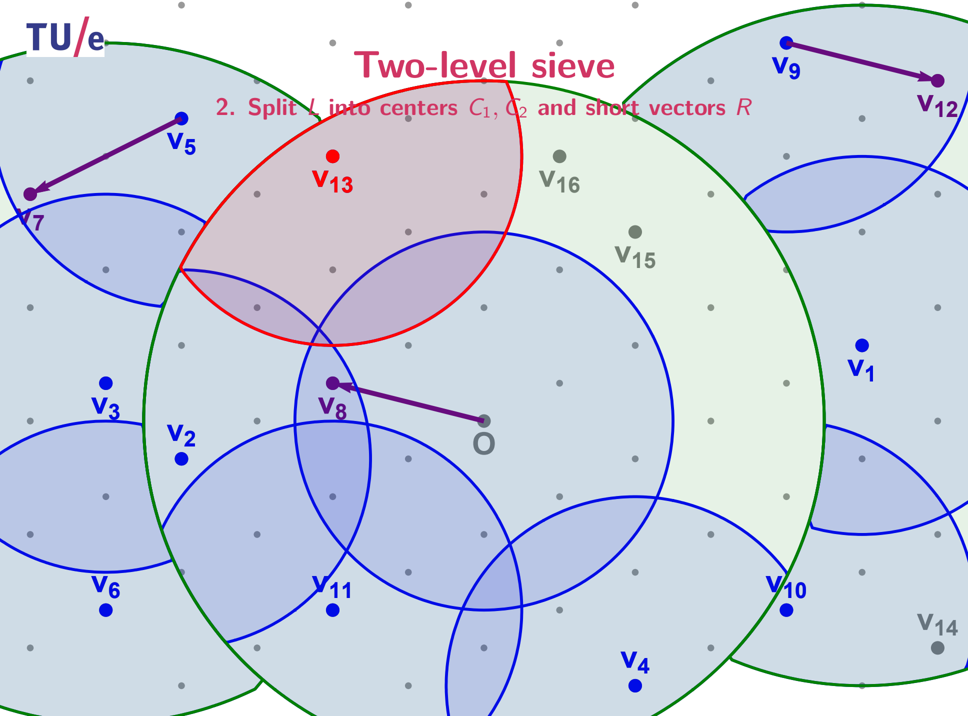
Two-level sieve

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



Two-level sieve

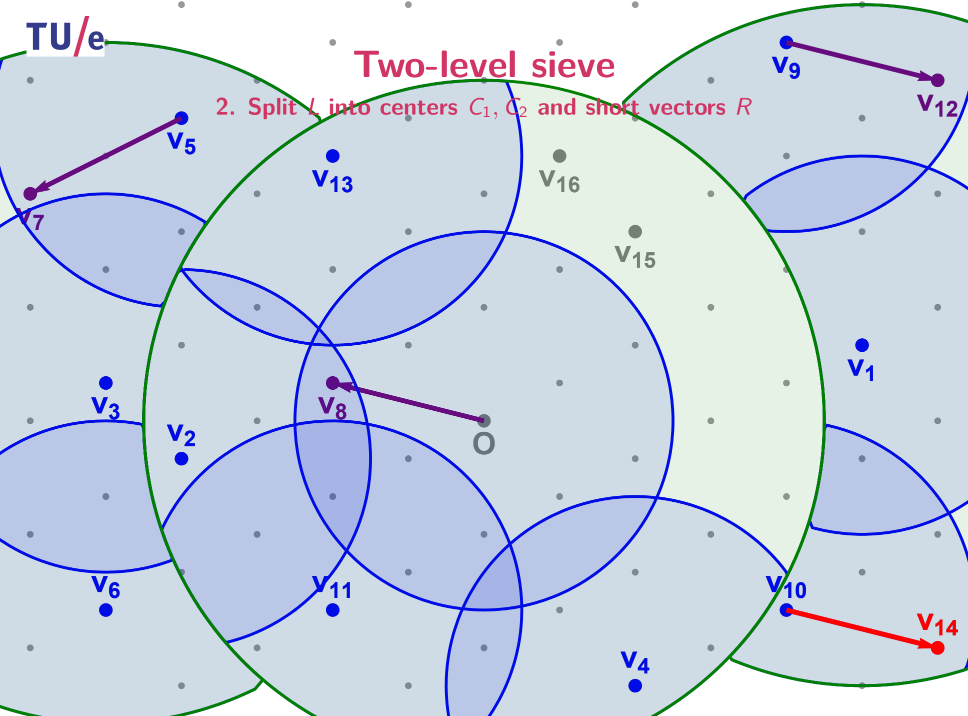
2. Split L into centers C_1, C_2 and short vectors R



TU/e

Two-level sieve

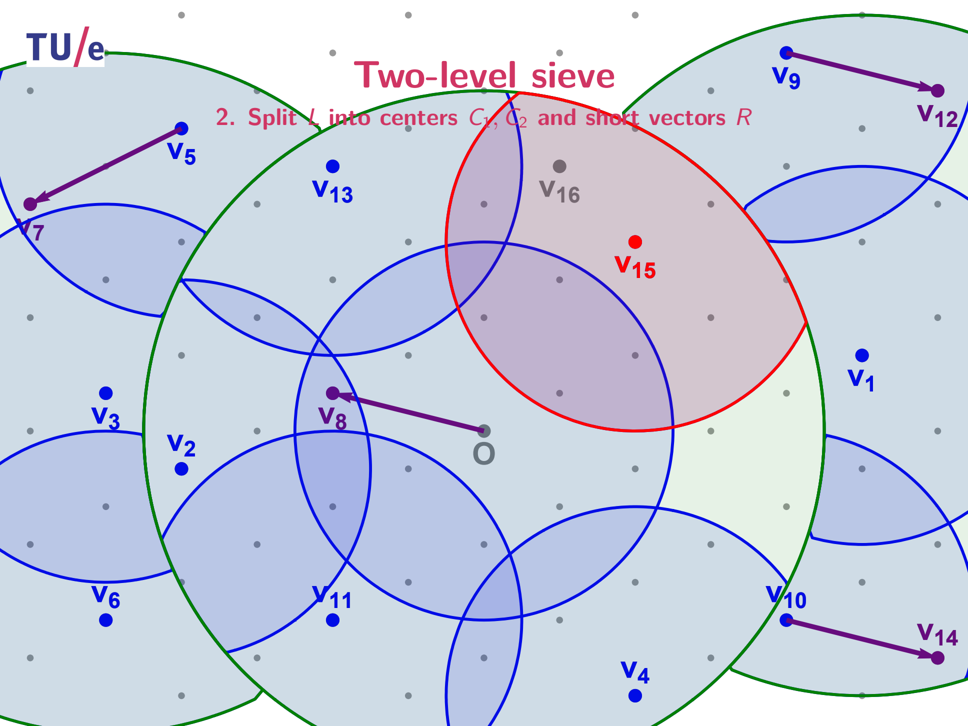
2. Split L into centers C_1, C_2 and short vectors R



TU/e

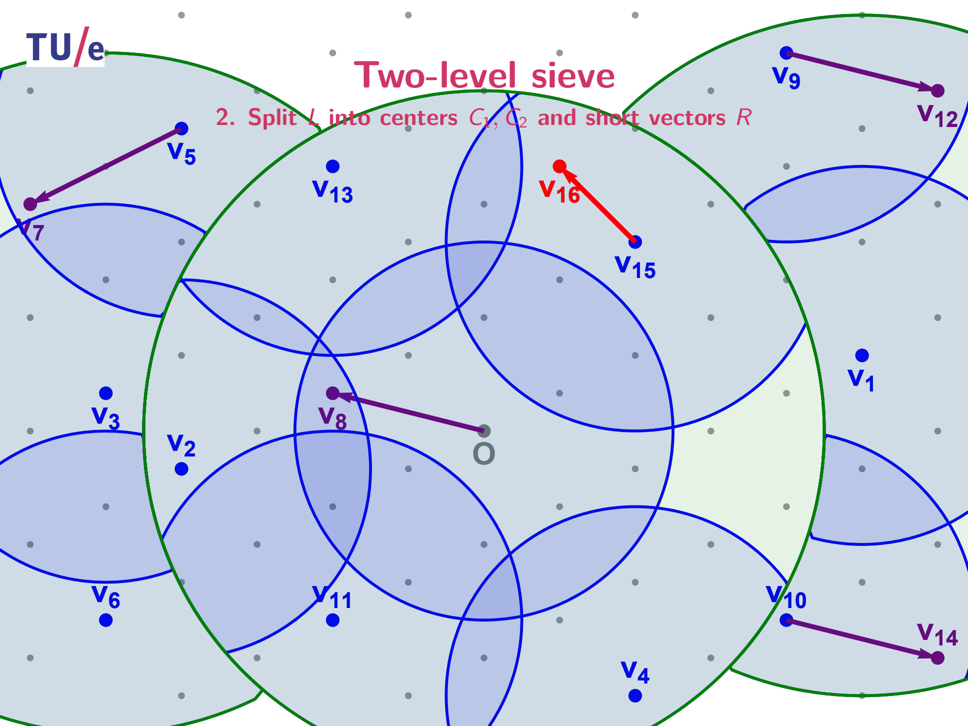
Two-level sieve

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



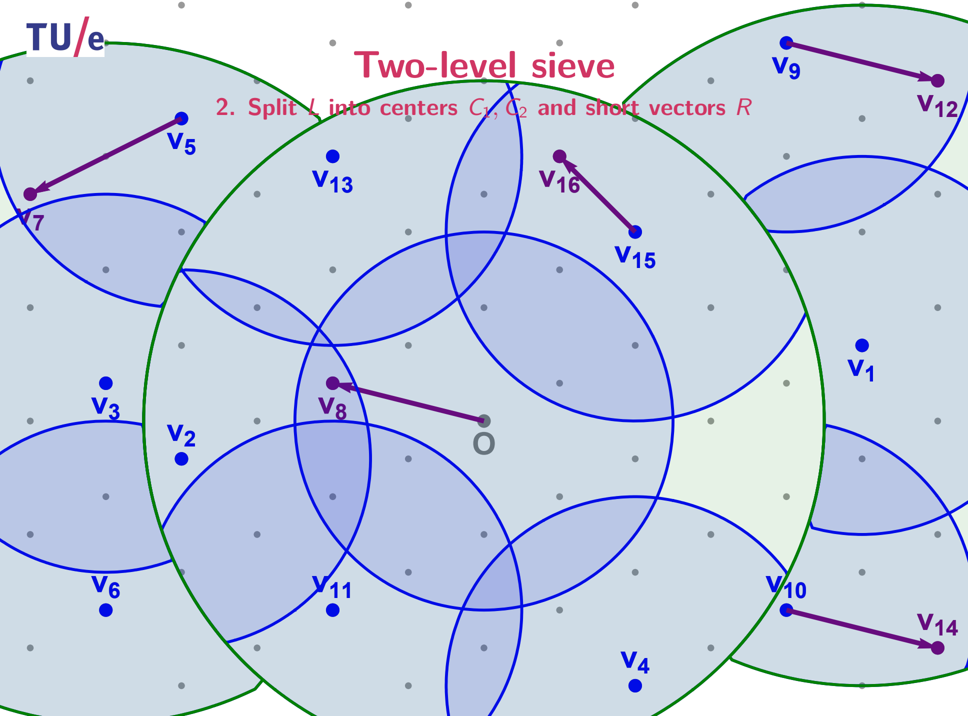
Two-level sieve

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



Two-level sieve

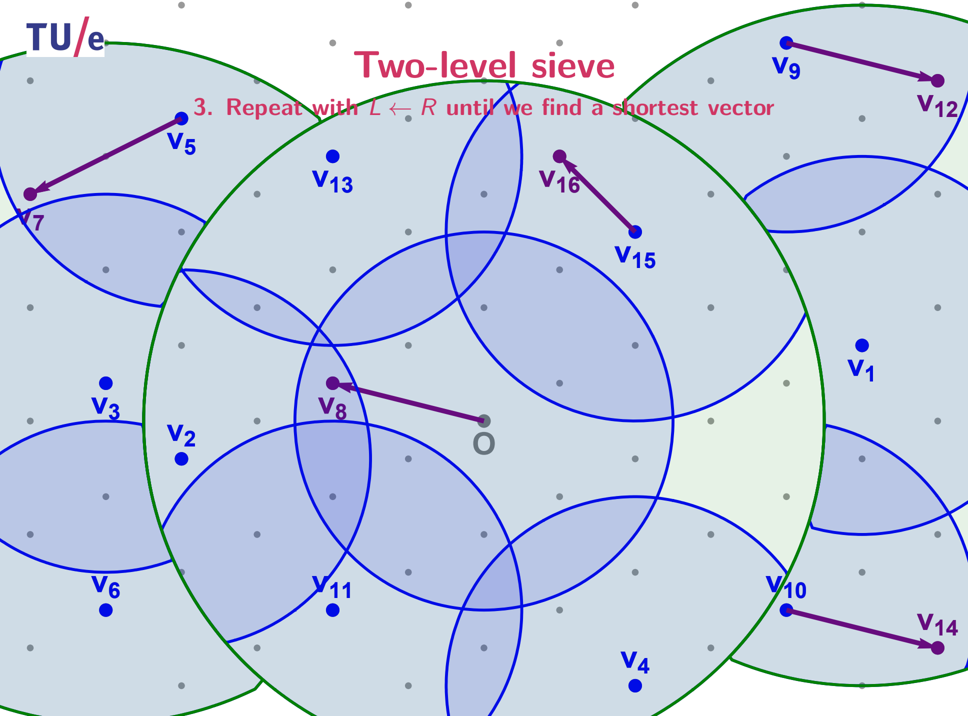
2. Split L into centers C_1, C_2 and short vectors R



TU/e

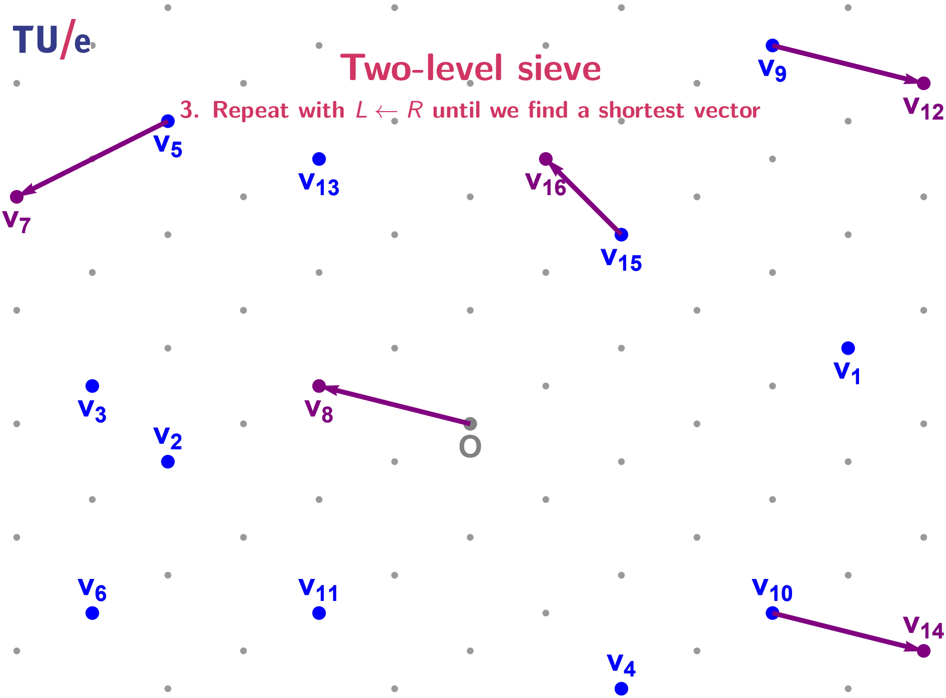
Two-level sieve

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



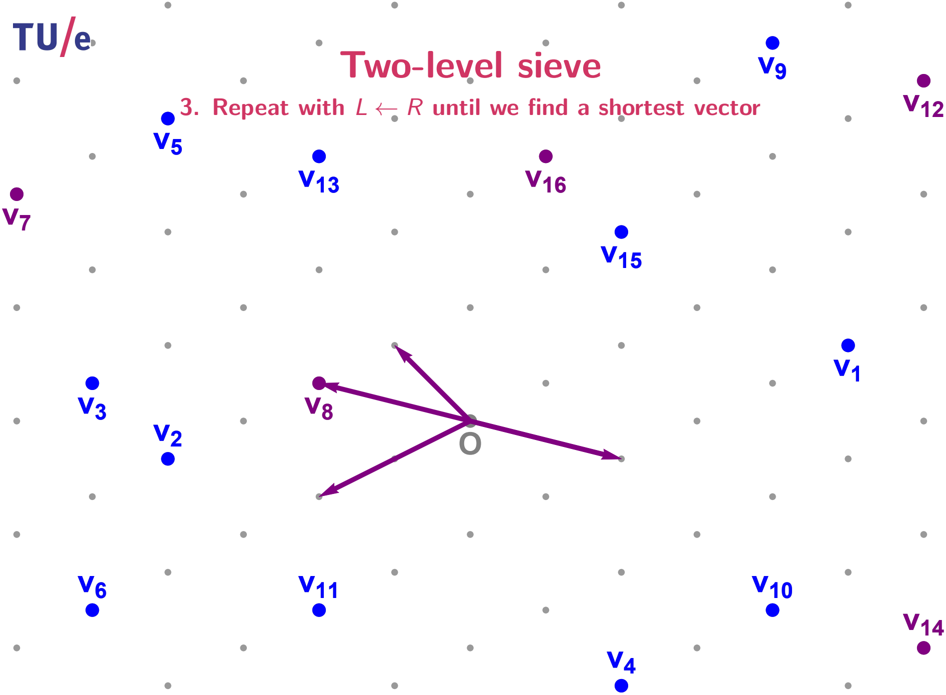
Two-level sieve

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



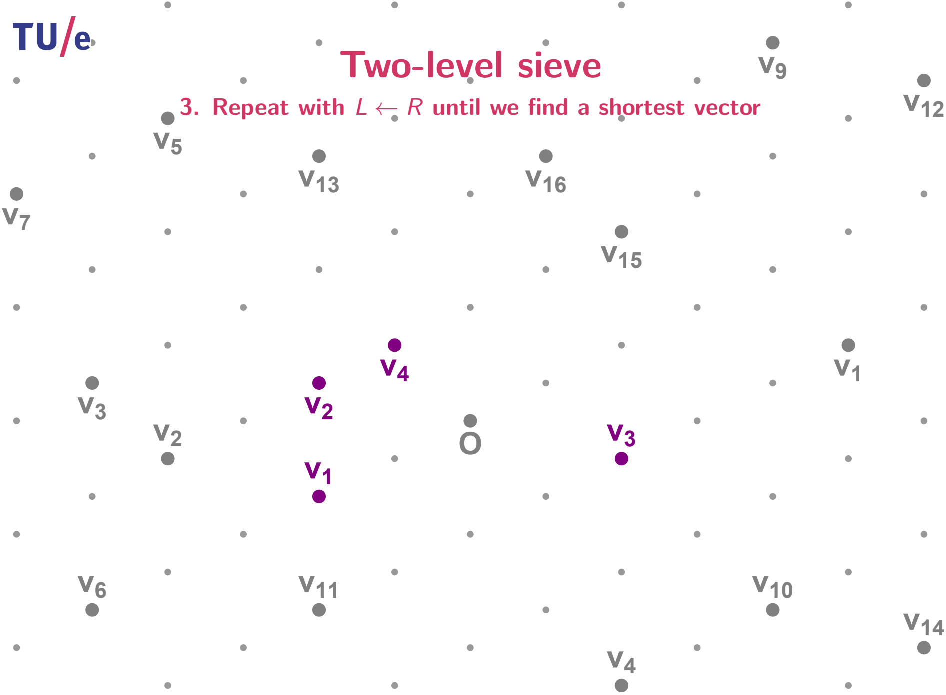
Two-level sieve

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



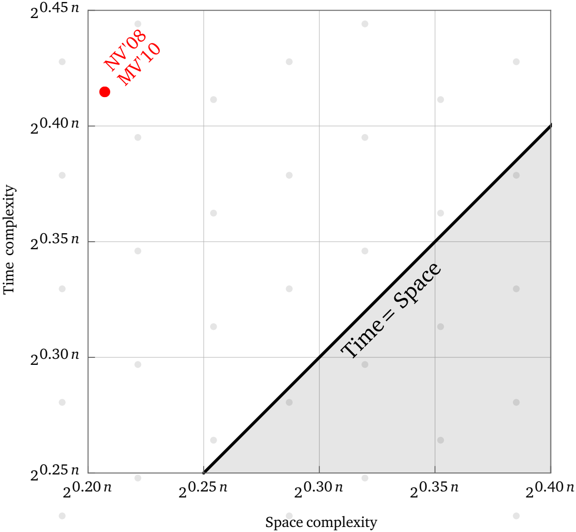
Two-level sieve

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



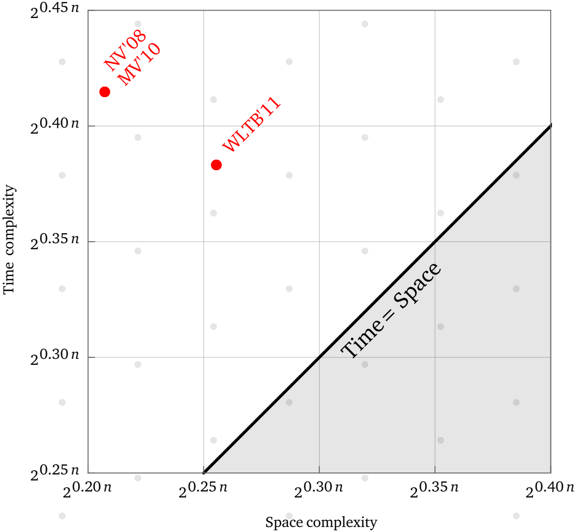
Two-level sieve

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 $2^{0.2557n}$.

TU/e

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)

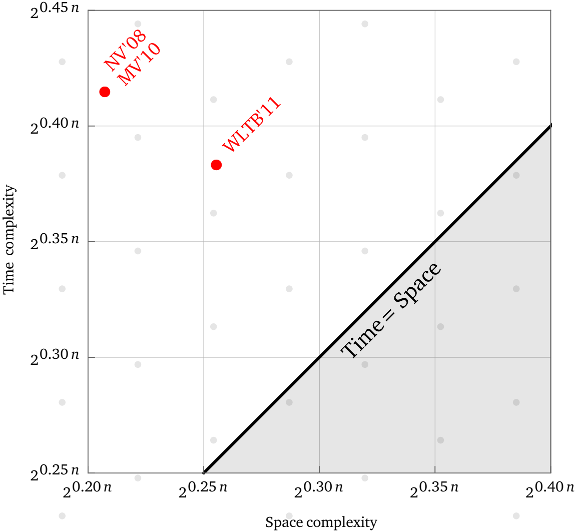
The three-level sieve runs in time $2^{0.3778n}$ and space $2^{0.2833n}$.

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.

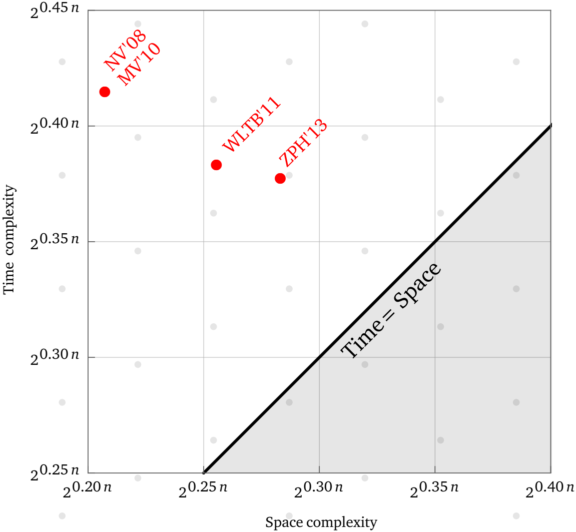
Three-level sieve

Space/time trade-off



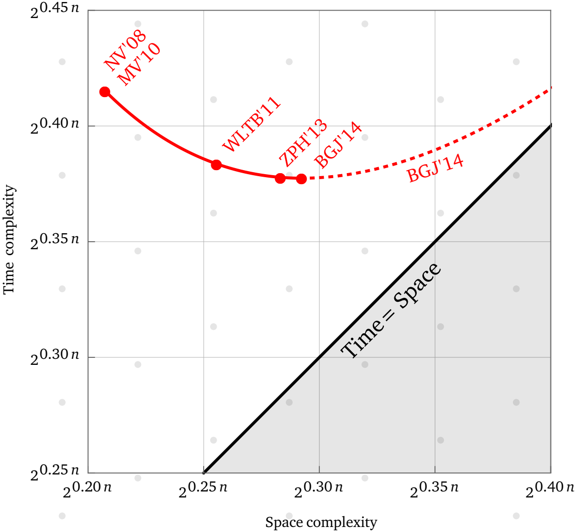
Three-level sieve

Space/time trade-off



Decomposition approach

Space/time trade-off



HashSieve (GS)

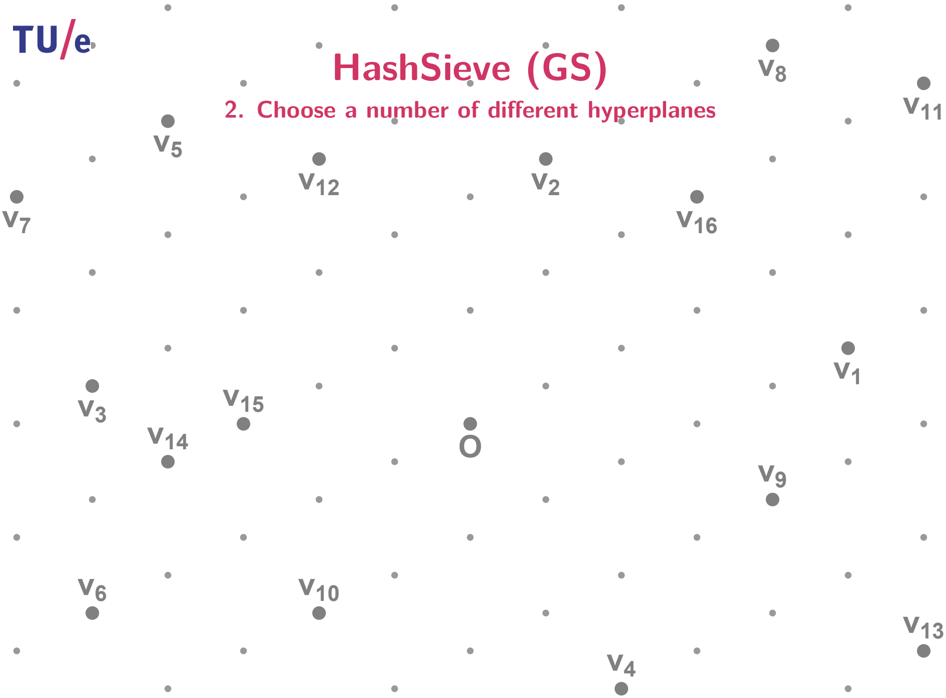
1. Generate random lattice vectors

HashSieve (GS)

1. Generate random lattice vectors

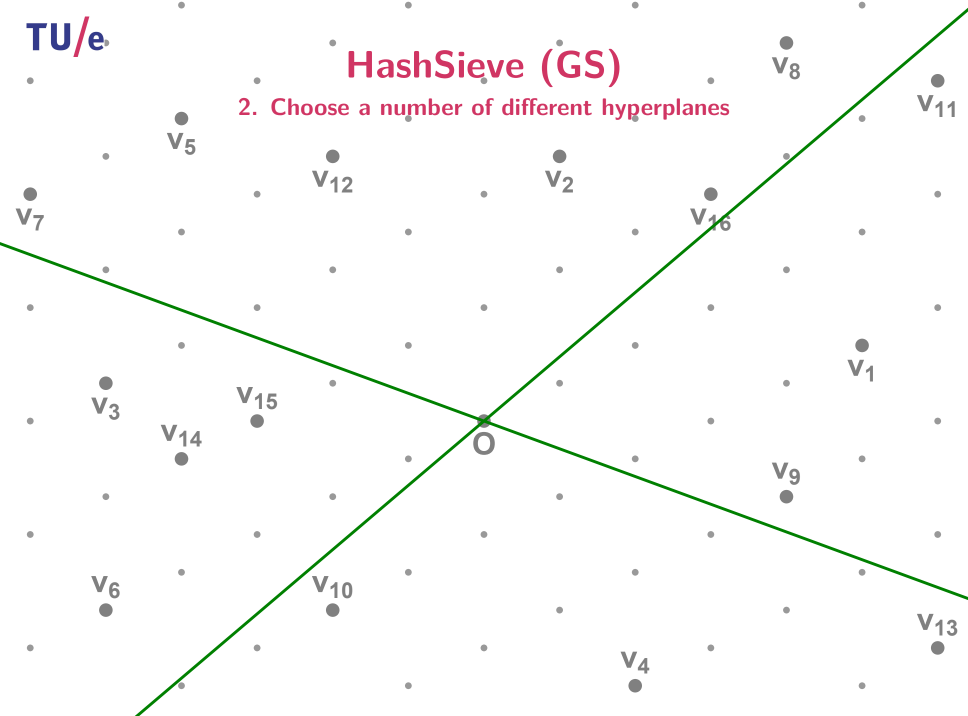
HashSieve (GS)

2. Choose a number of different hyperplanes



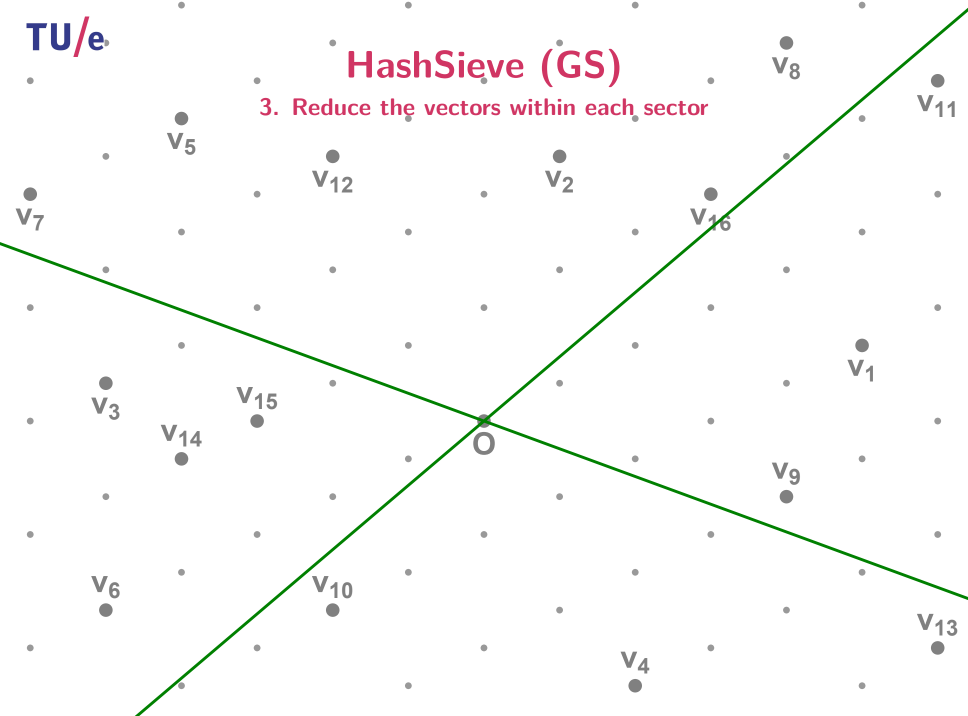
HashSieve (GS)

2. Choose a number of different hyperplanes



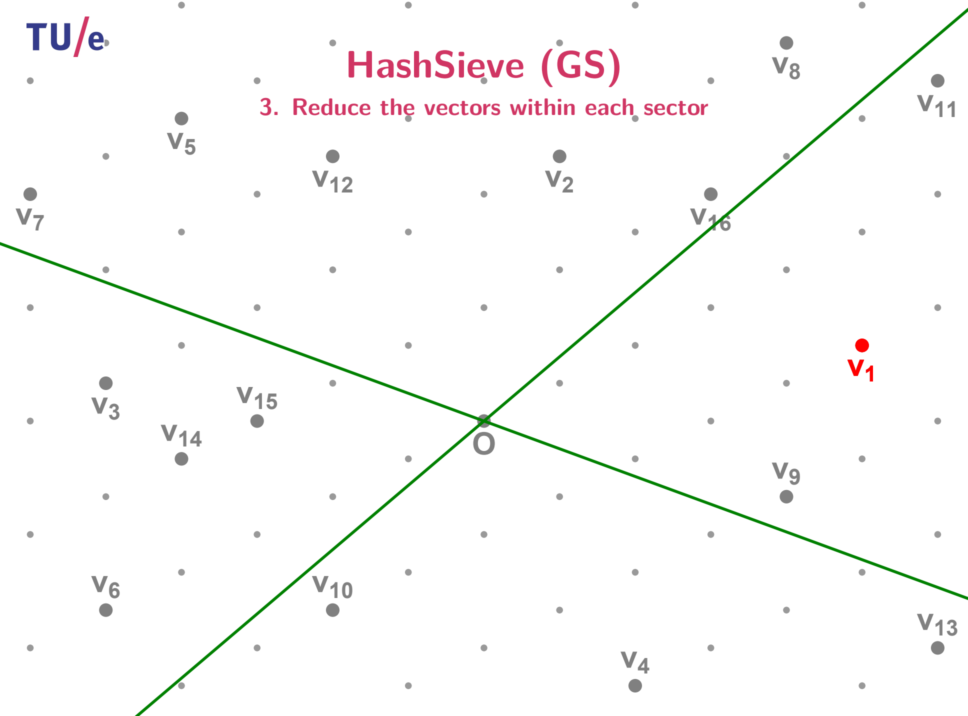
HashSieve (GS)

3. Reduce the vectors within each sector



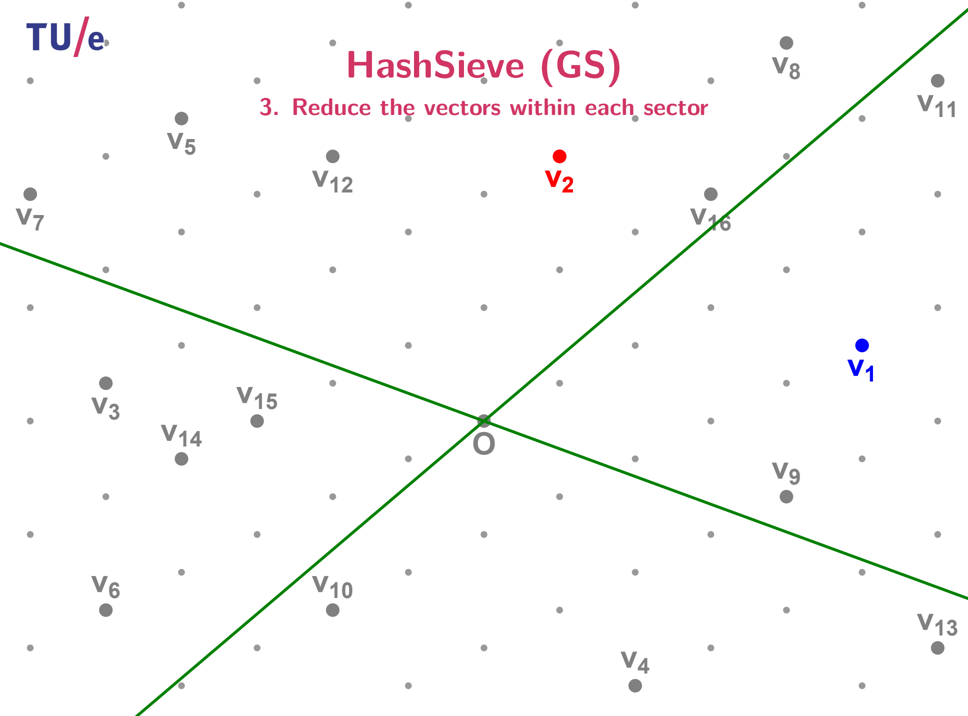
HashSieve (GS)

3. Reduce the vectors within each sector



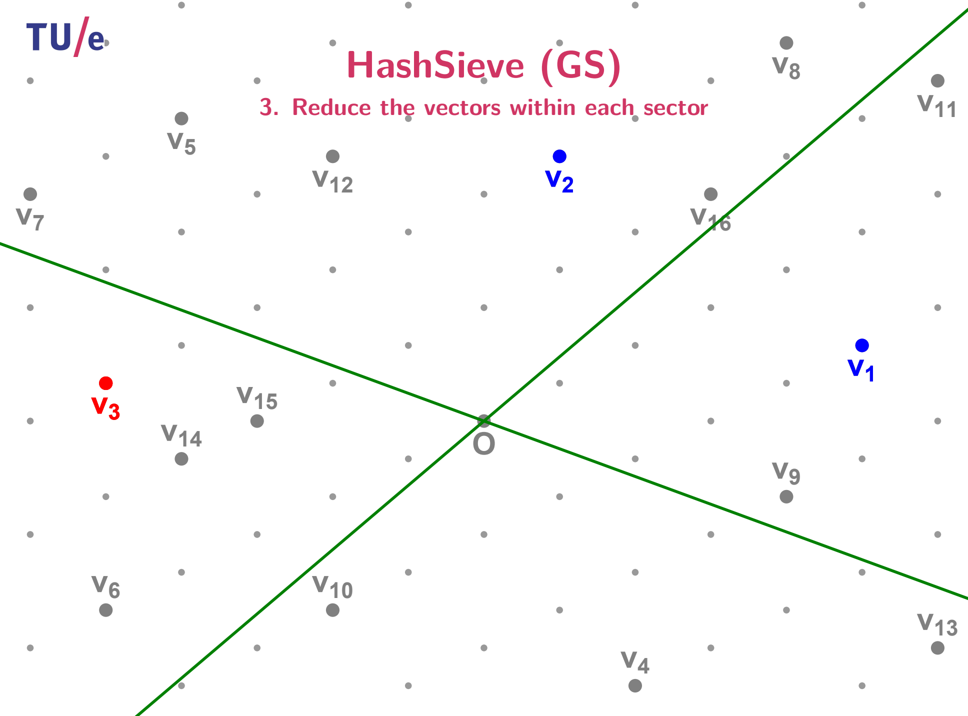
HashSieve (GS)

3. Reduce the vectors within each sector



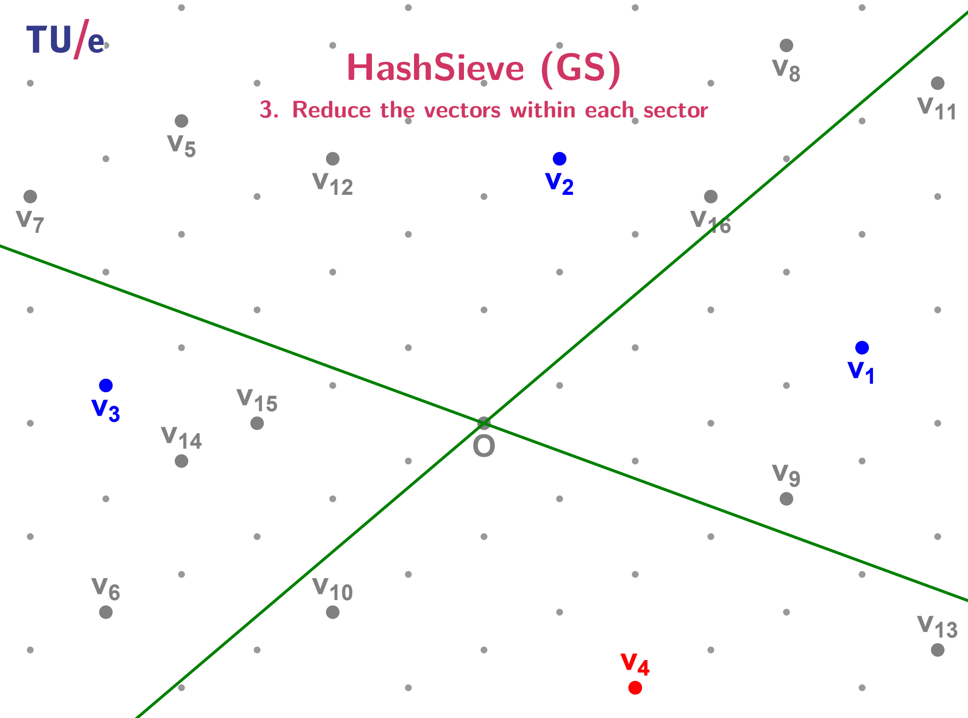
HashSieve (GS)

3. Reduce the vectors within each sector



HashSieve (GS)

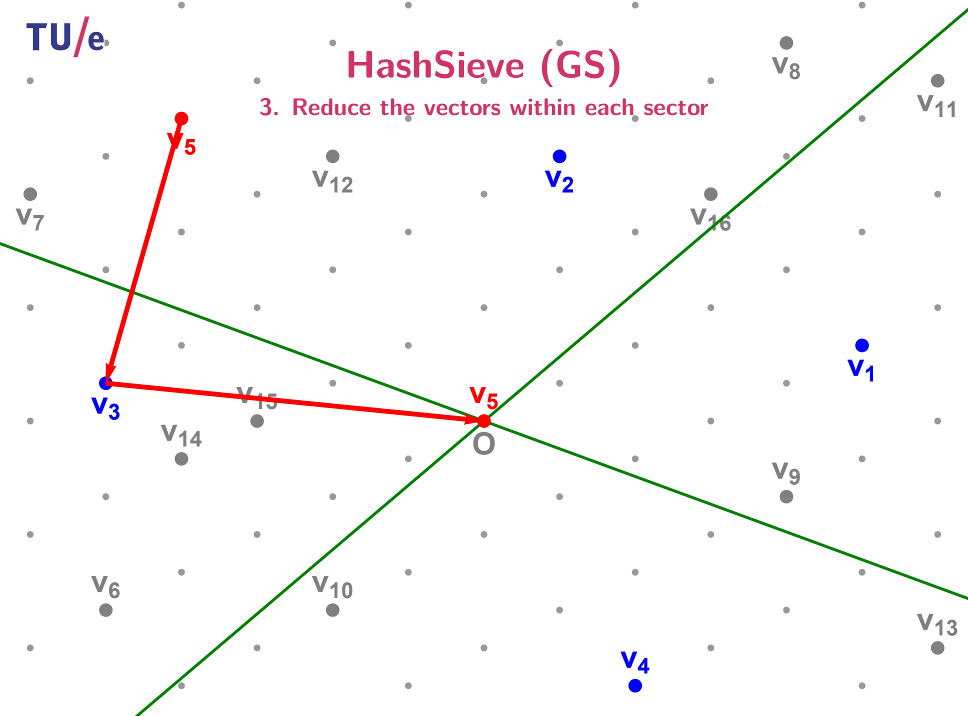
3. Reduce the vectors within each sector



TU/e

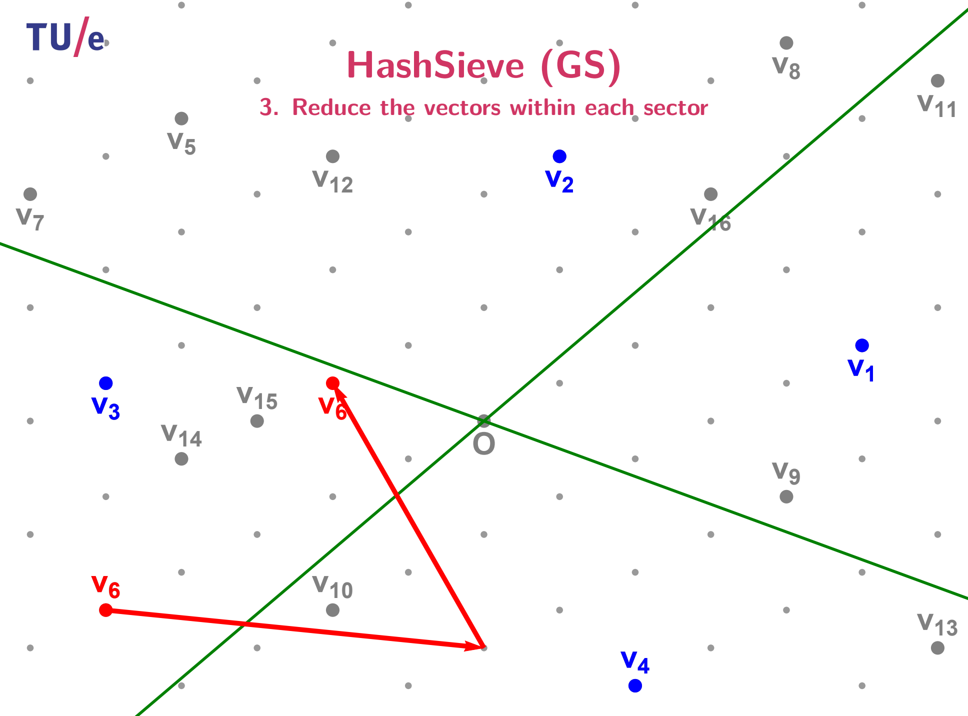
HashSieve (GS)

3. Reduce the vectors within each sector



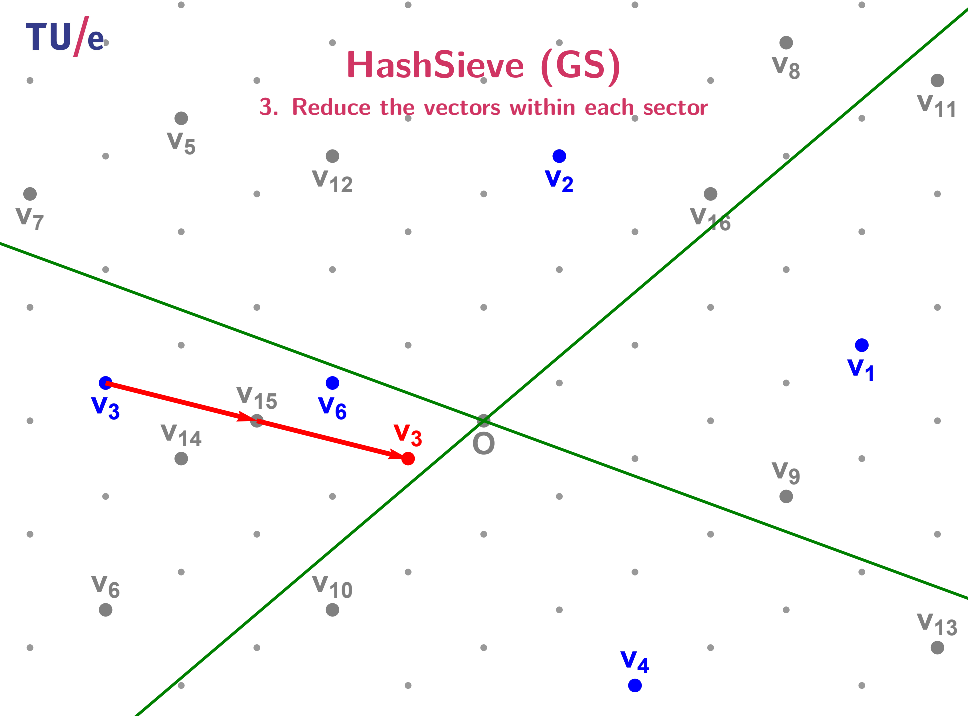
HashSieve (GS)

3. Reduce the vectors within each sector



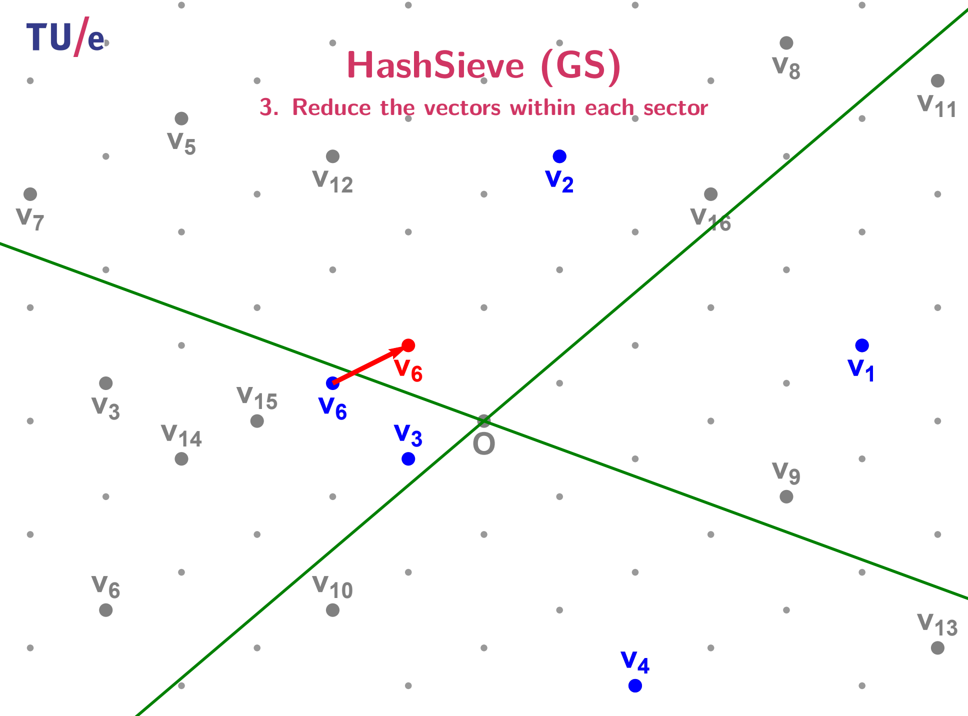
HashSieve (GS)

3. Reduce the vectors within each sector



HashSieve (GS)

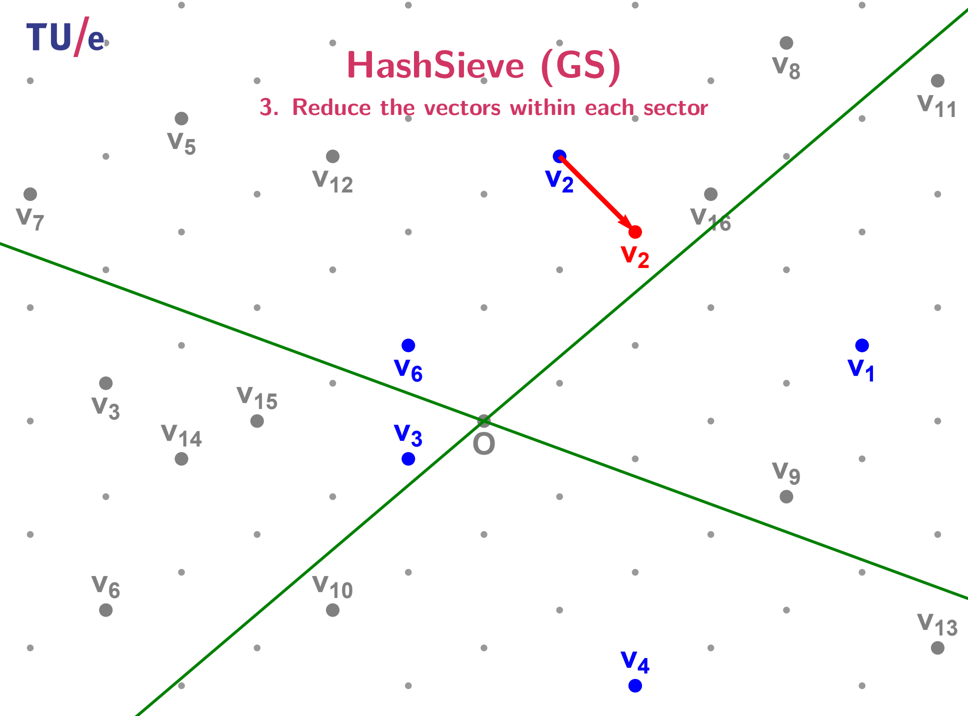
3. Reduce the vectors within each sector



TU/e

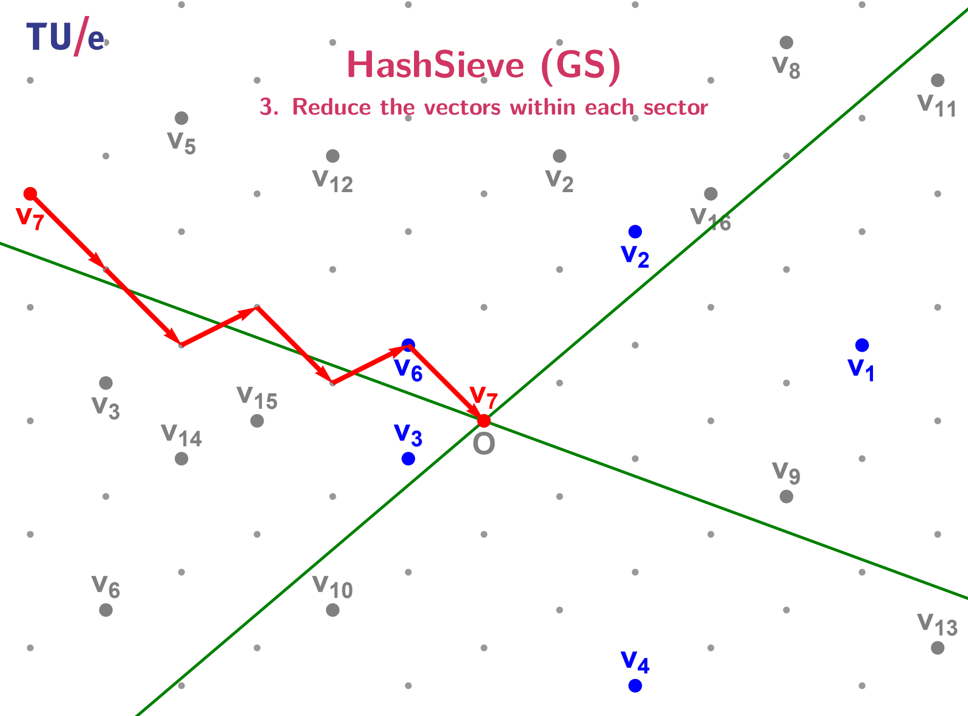
HashSieve (GS)

3. Reduce the vectors within each sector



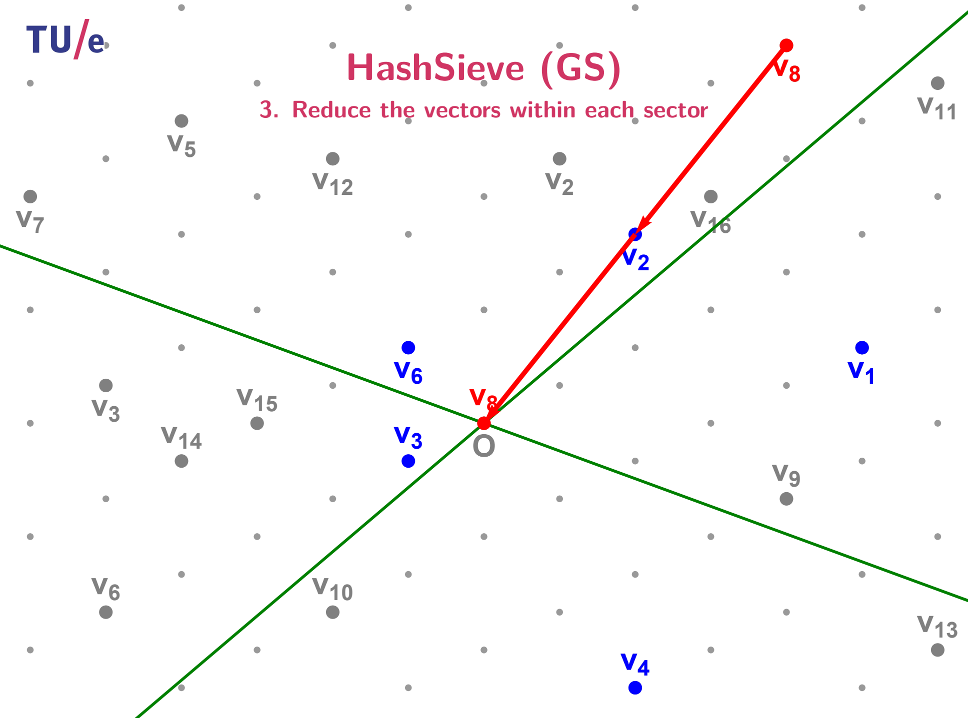
HashSieve (GS)

3. Reduce the vectors within each sector



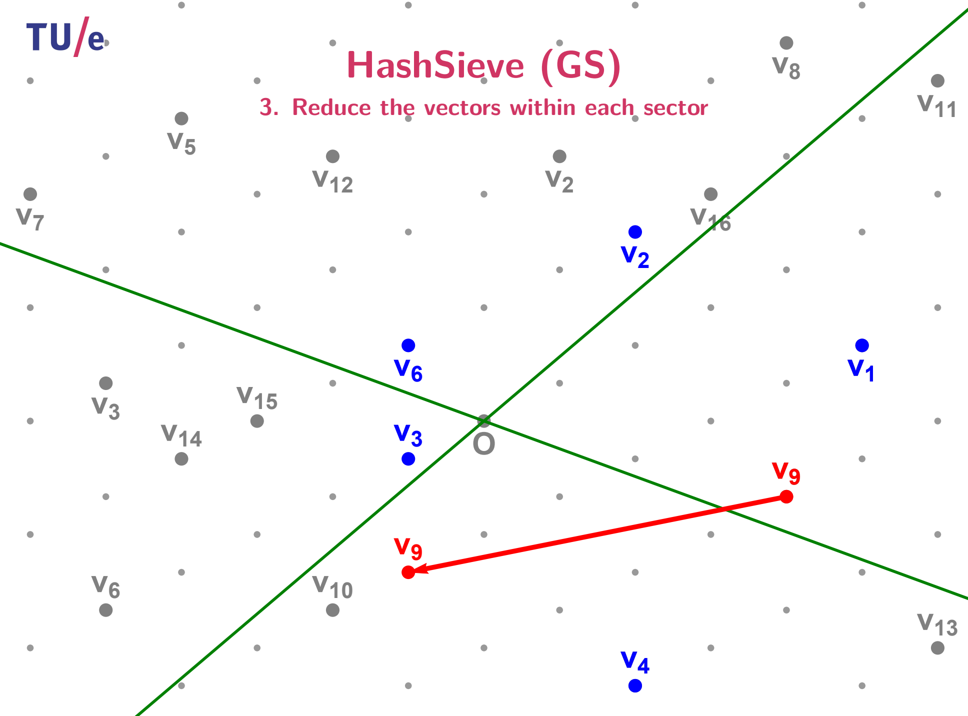
HashSieve (GS)

3. Reduce the vectors within each sector



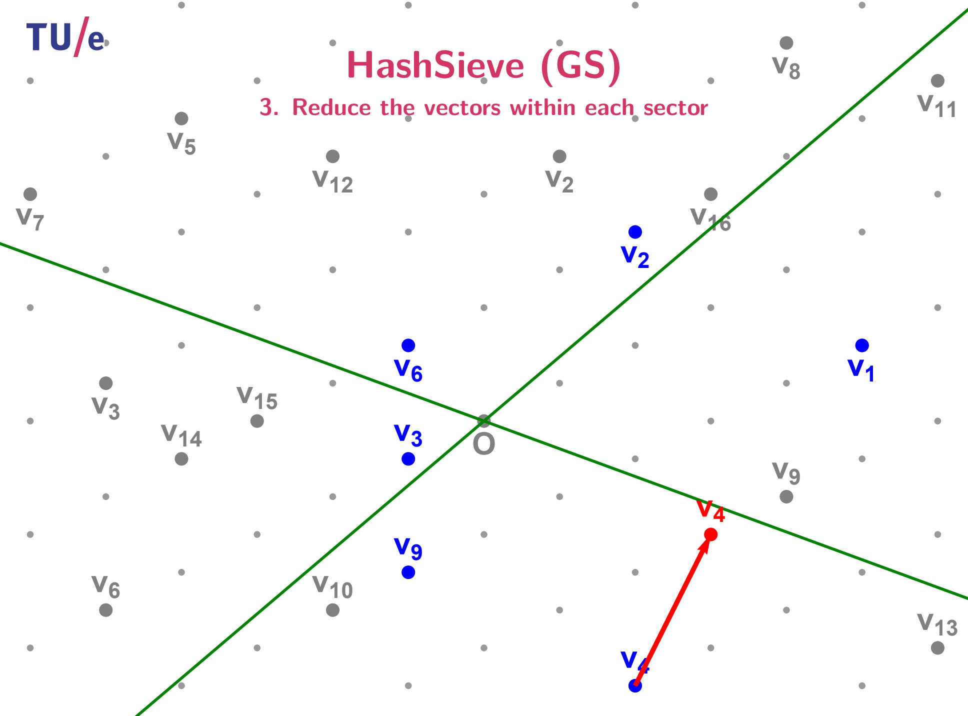
HashSieve (GS)

3. Reduce the vectors within each sector



HashSieve (GS)

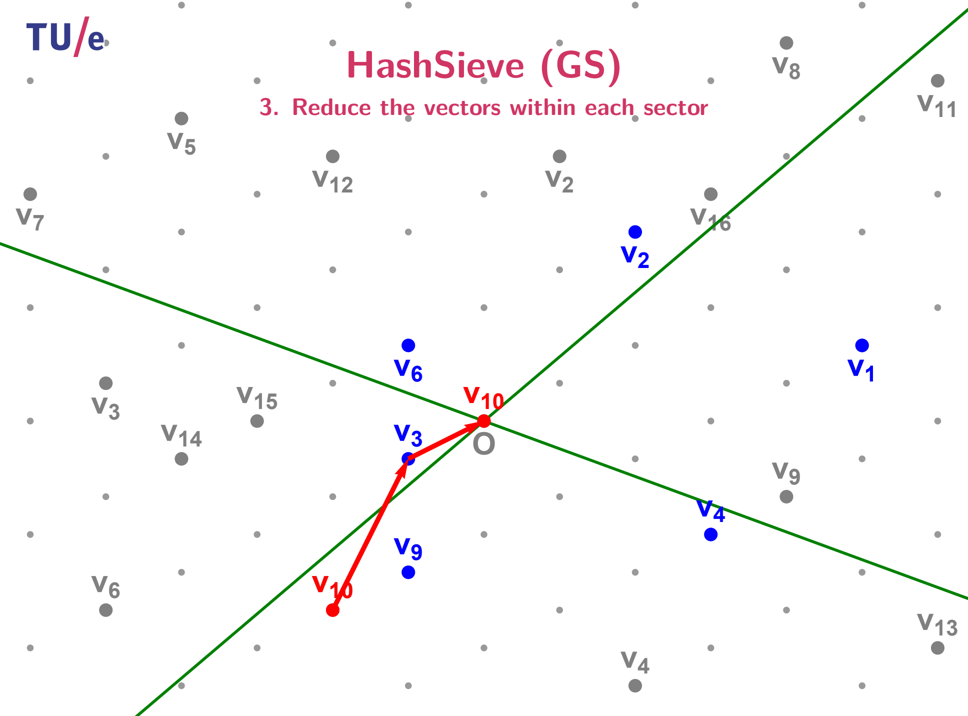
3. Reduce the vectors within each sector



TU/e

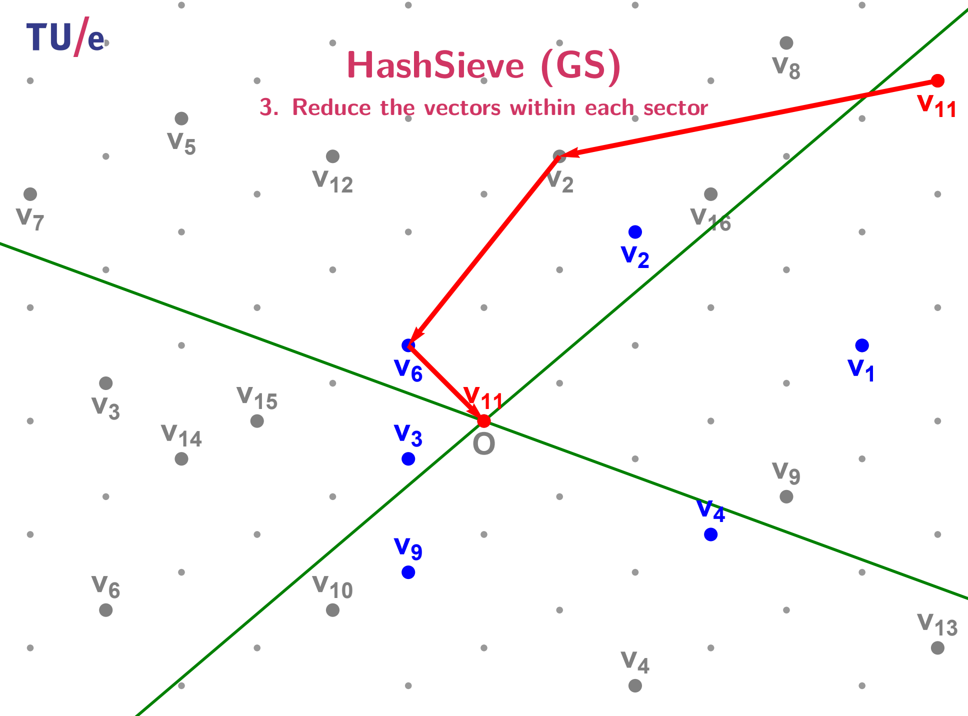
HashSieve (GS)

3. Reduce the vectors within each sector



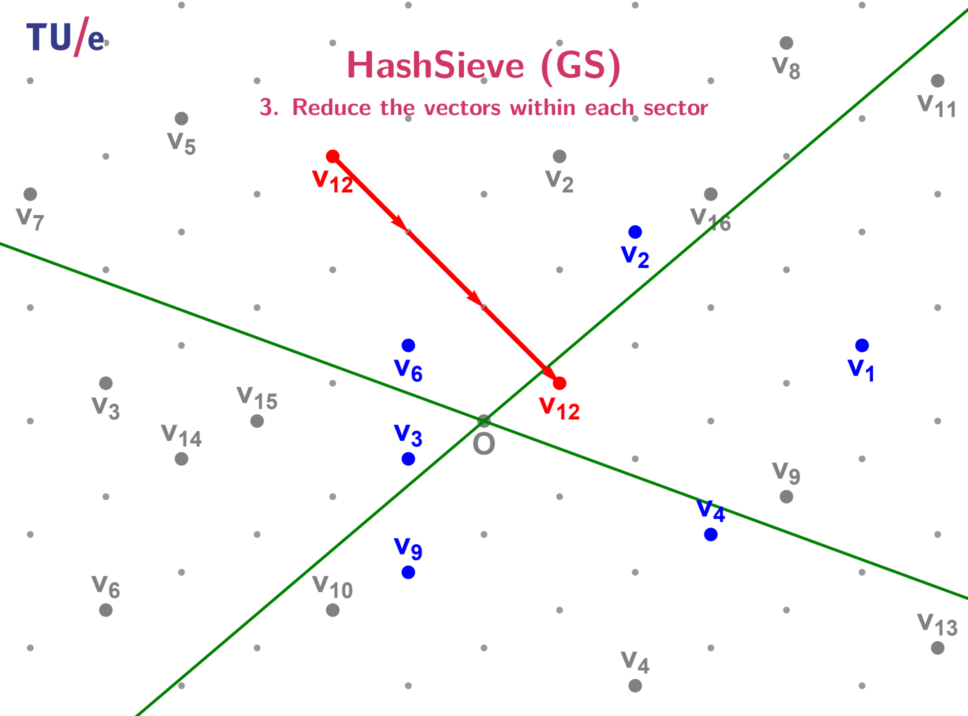
HashSieve (GS)

3. Reduce the vectors within each sector



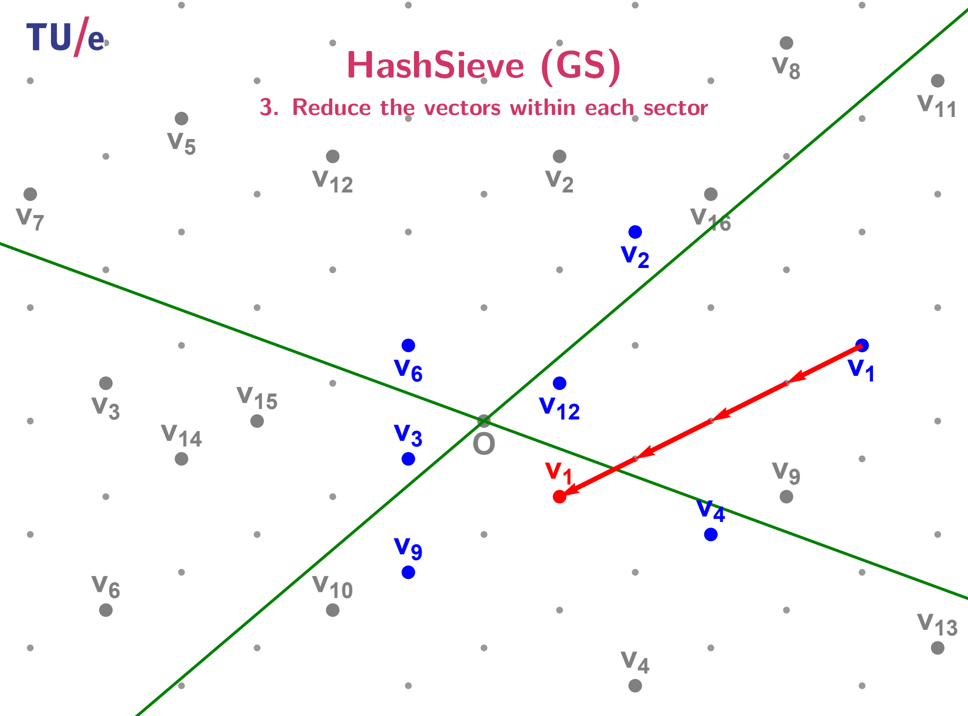
HashSieve (GS)

3. Reduce the vectors within each sector



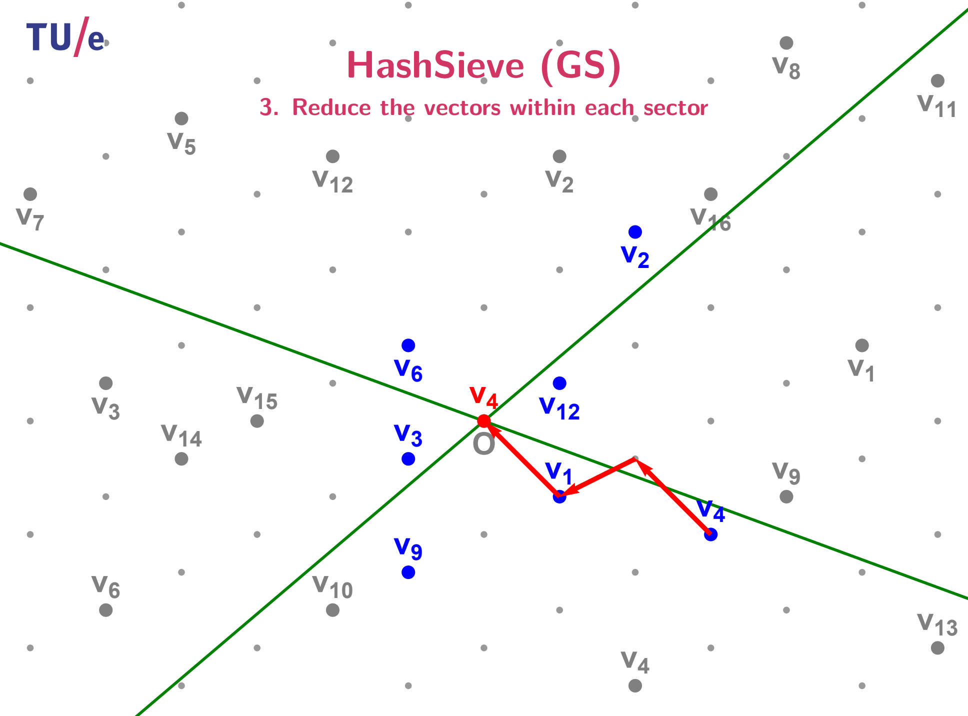
HashSieve (GS)

3. Reduce the vectors within each sector



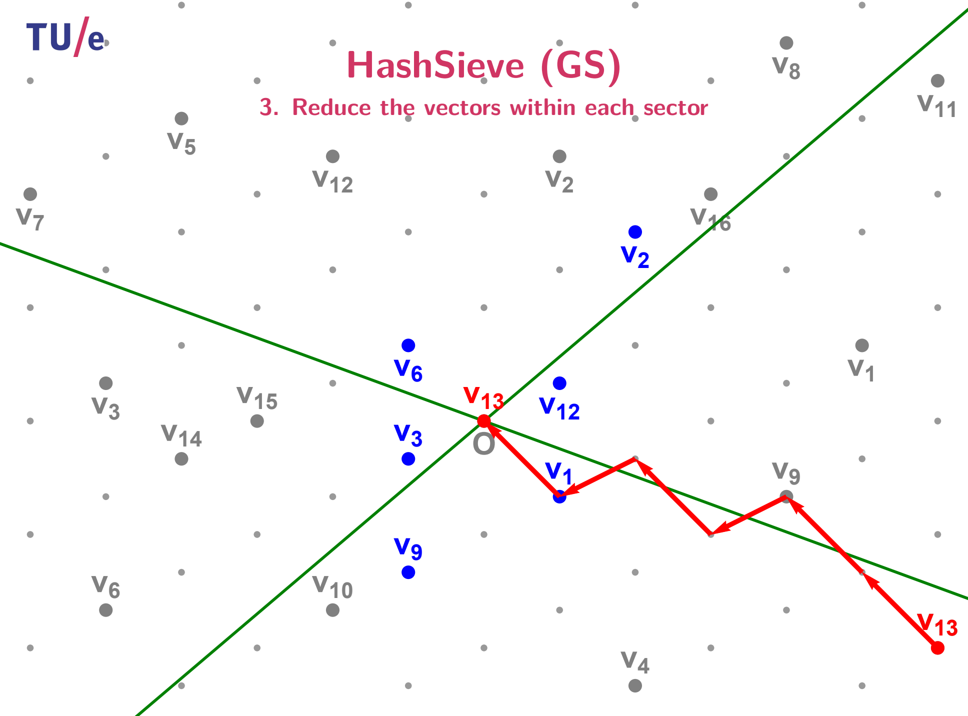
HashSieve (GS)

3. Reduce the vectors within each sector



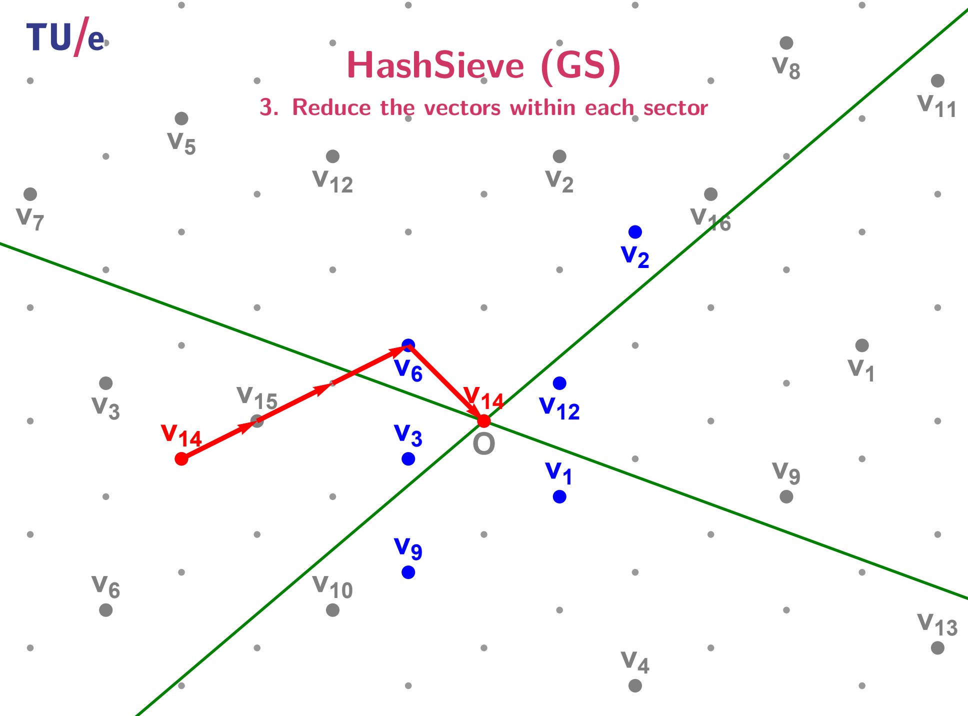
HashSieve (GS)

3. Reduce the vectors within each sector



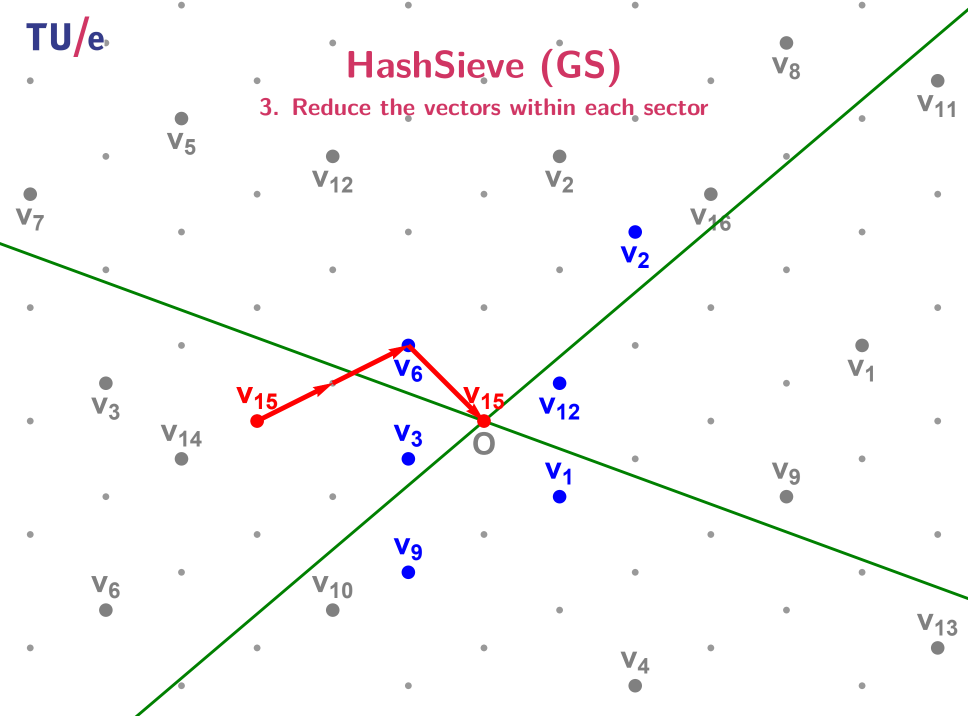
HashSieve (GS)

3. Reduce the vectors within each sector



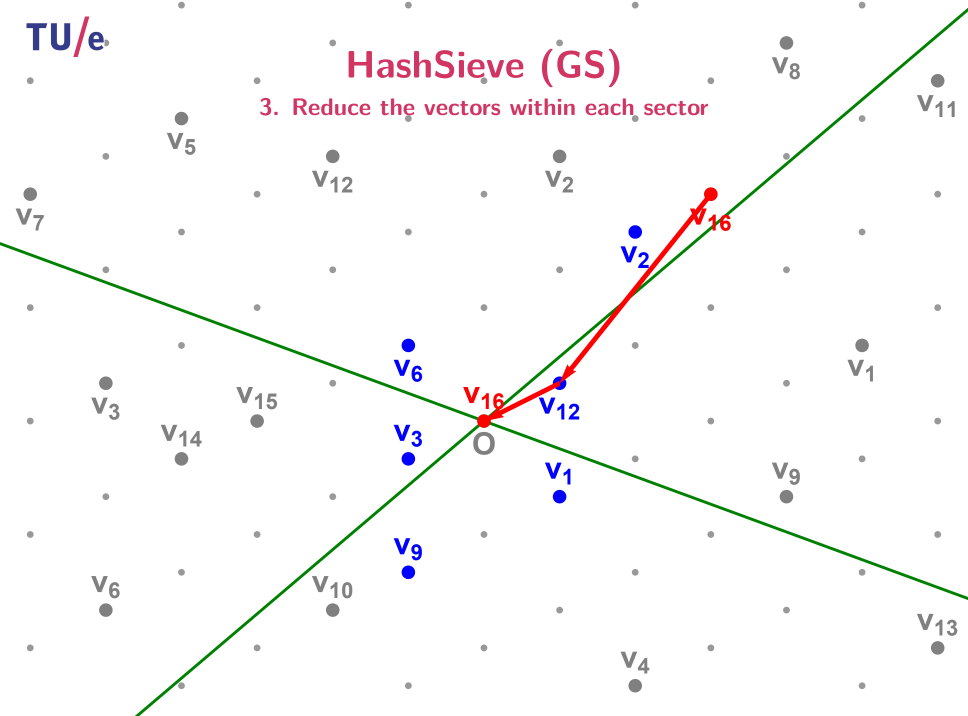
HashSieve (GS)

3. Reduce the vectors within each sector



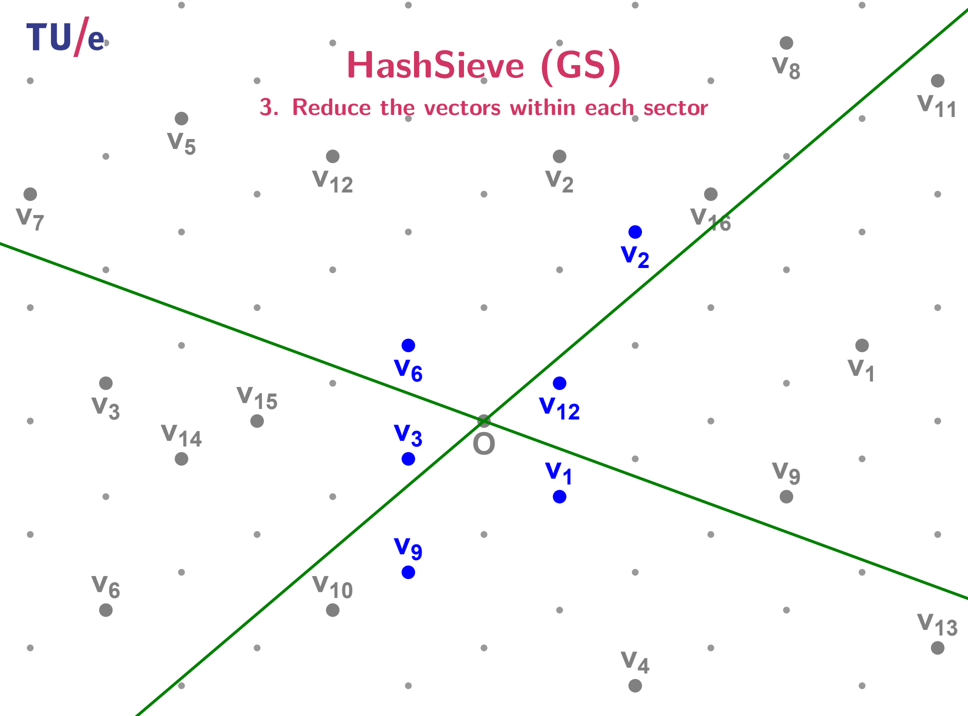
HashSieve (GS)

3. Reduce the vectors within each sector



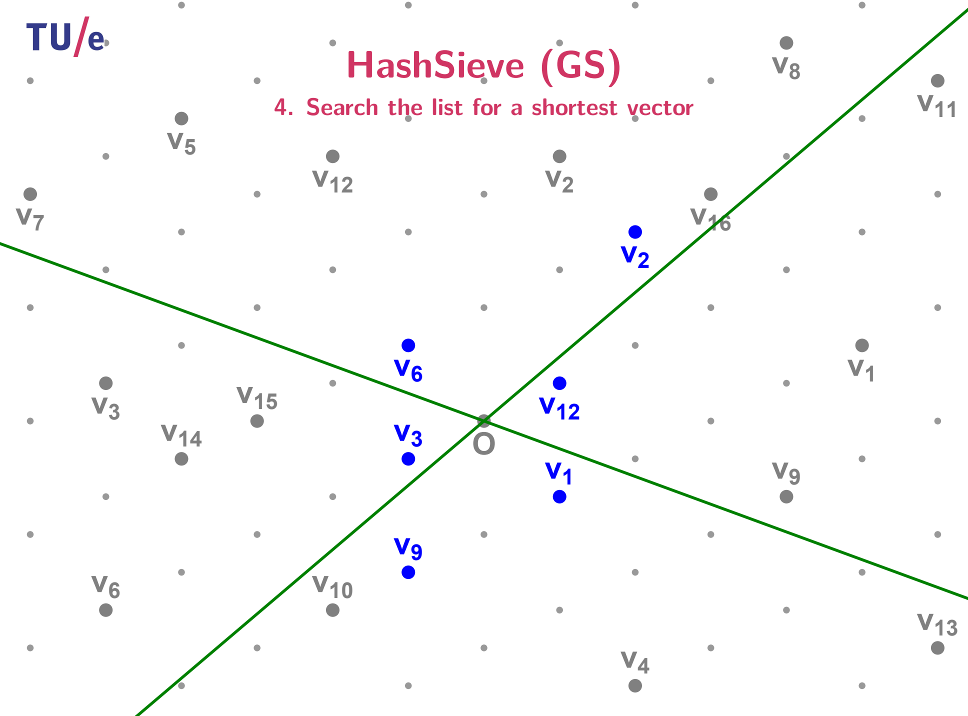
HashSieve (GS)

3. Reduce the vectors within each sector



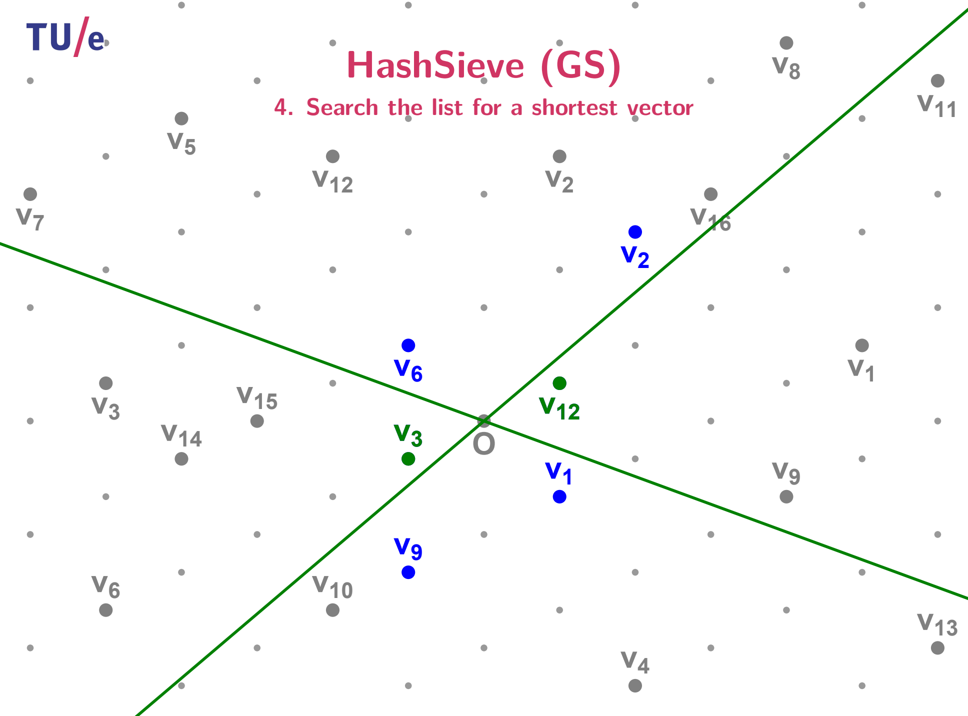
HashSieve (GS)

4. Search the list for a shortest vector



HashSieve (GS)

4. Search the list for a shortest vector



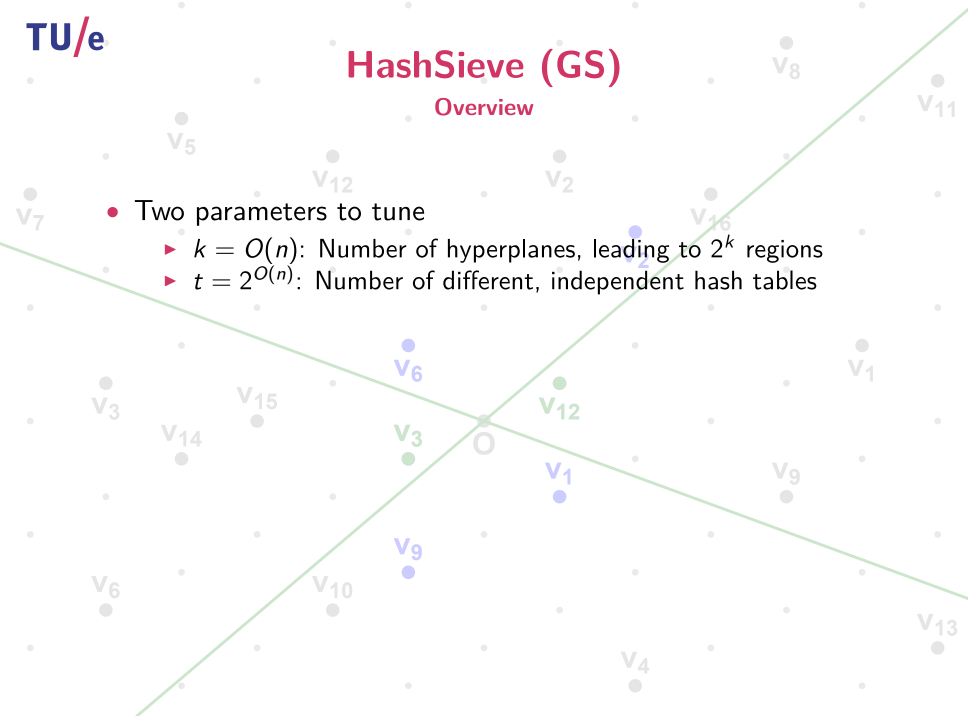
HashSieve (GS)

Overview

HashSieve (GS)

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



HashSieve (GS)

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

HashSieve (GS)

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

HashSieve (GS)

Overview

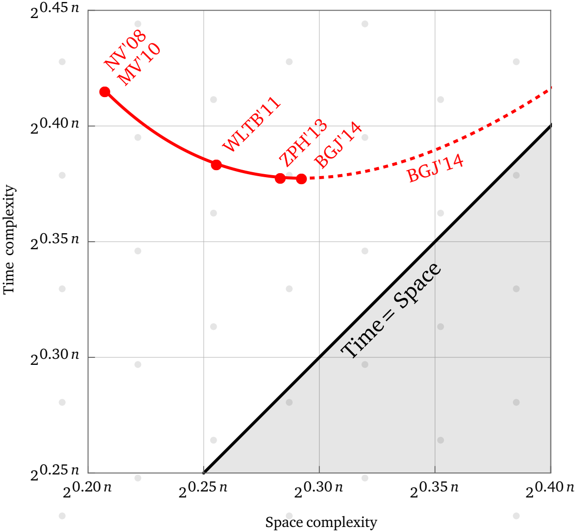
- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent hash tables
- Space complexity: $2^{0.34n+o(n)}$
  - Store $2^{0.13n}$ hash tables, each containing all $2^{0.21n}$ vectors
- Time complexity: $2^{0.34n+o(n)}$
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

Heuristic

The HashSieve (GS) runs in time and space $2^{0.34n+o(n)}$.

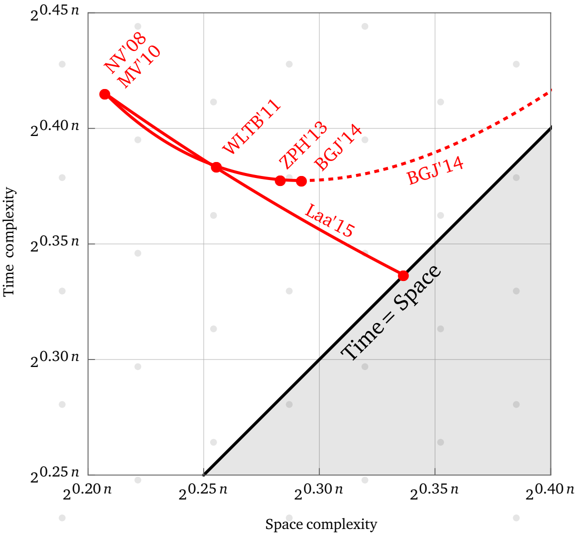
HashSieve (GS)

Space/time trade-off



HashSieve (GS)

Space/time trade-off



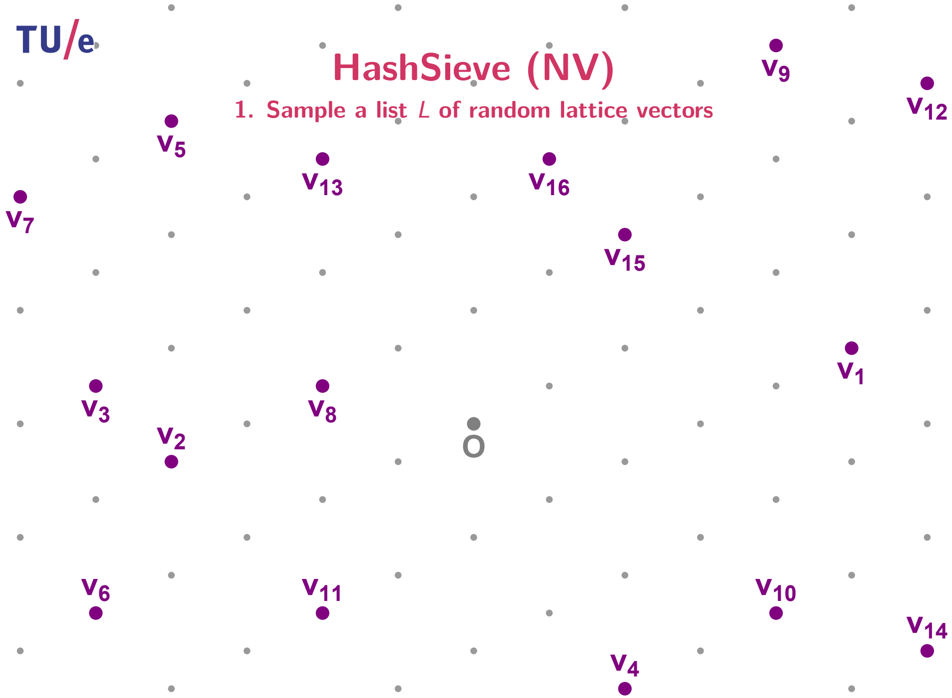
HashSieve (NV)

1. Sample a list L of random lattice vectors

TU/e

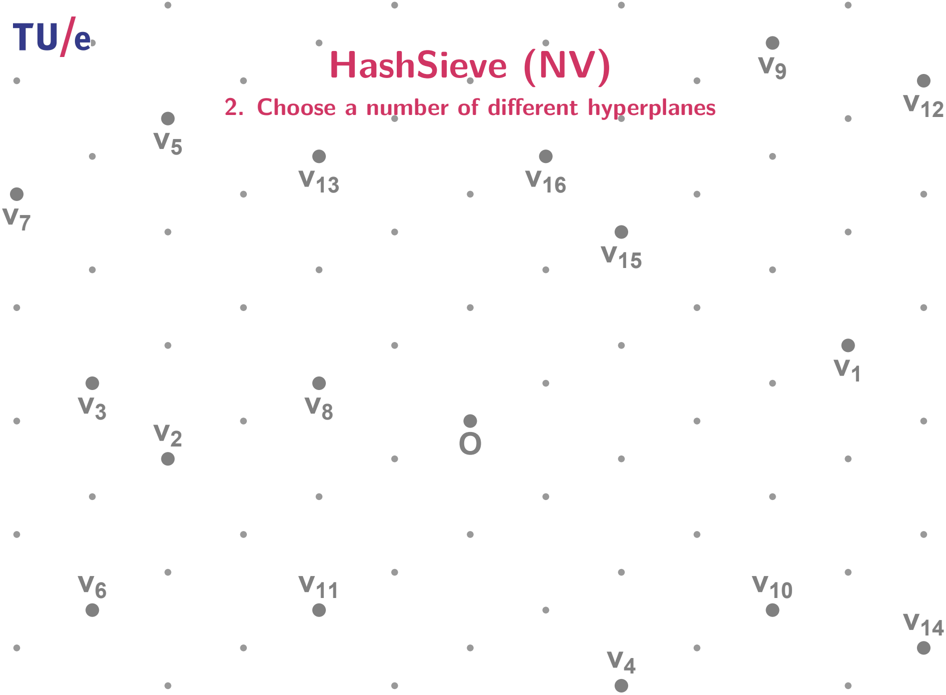
HashSieve (NV)

1. Sample a list L of random lattice vectors



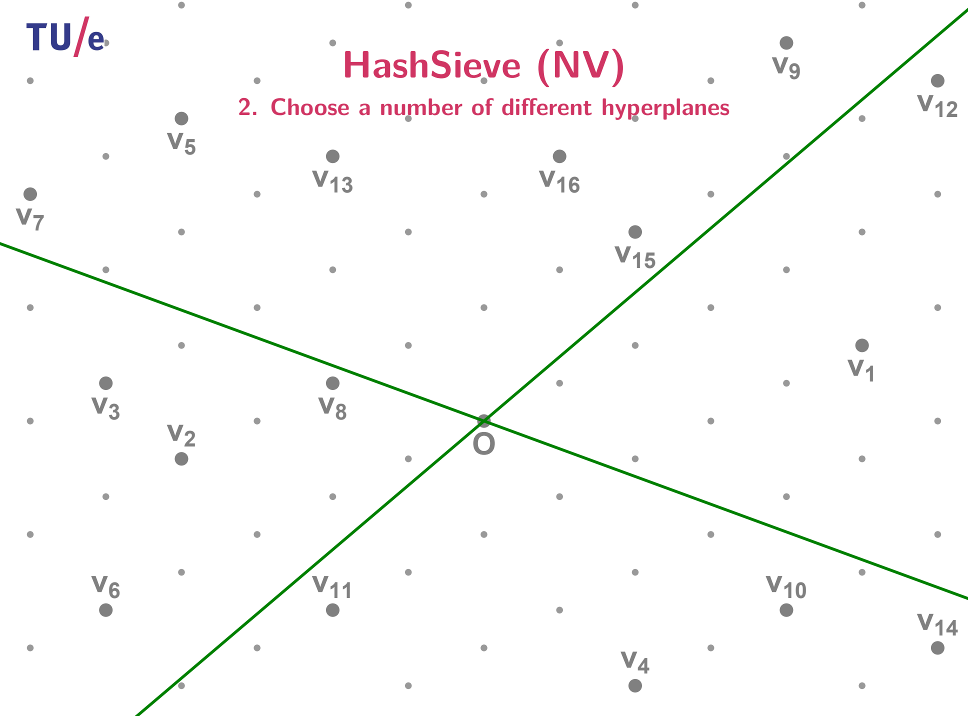
HashSieve (NV)

2. Choose a number of different hyperplanes



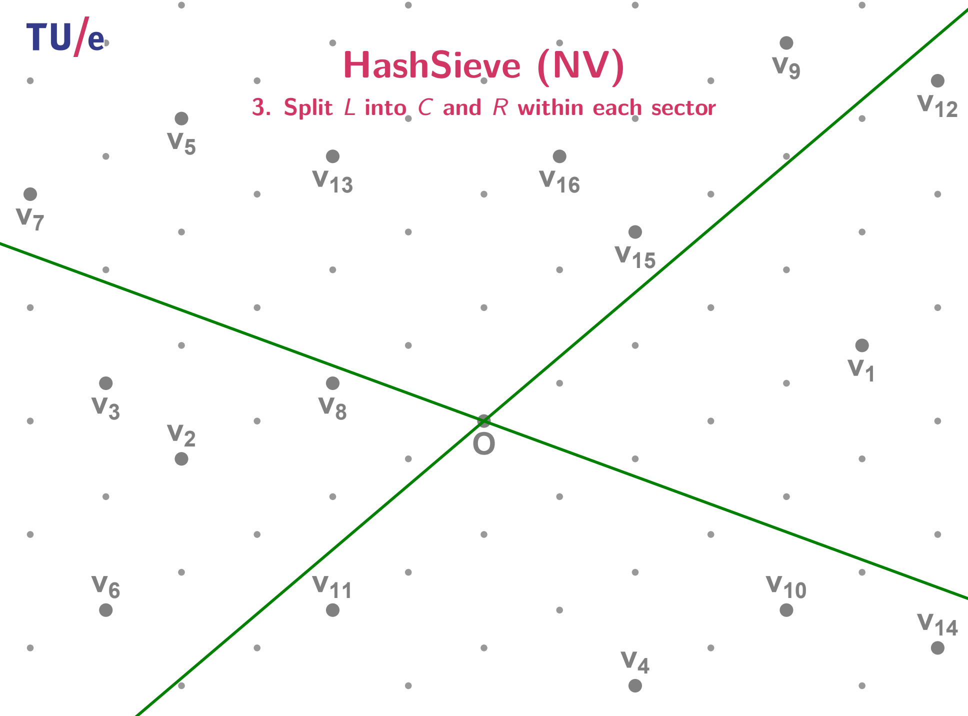
HashSieve (NV)

2. Choose a number of different hyperplanes



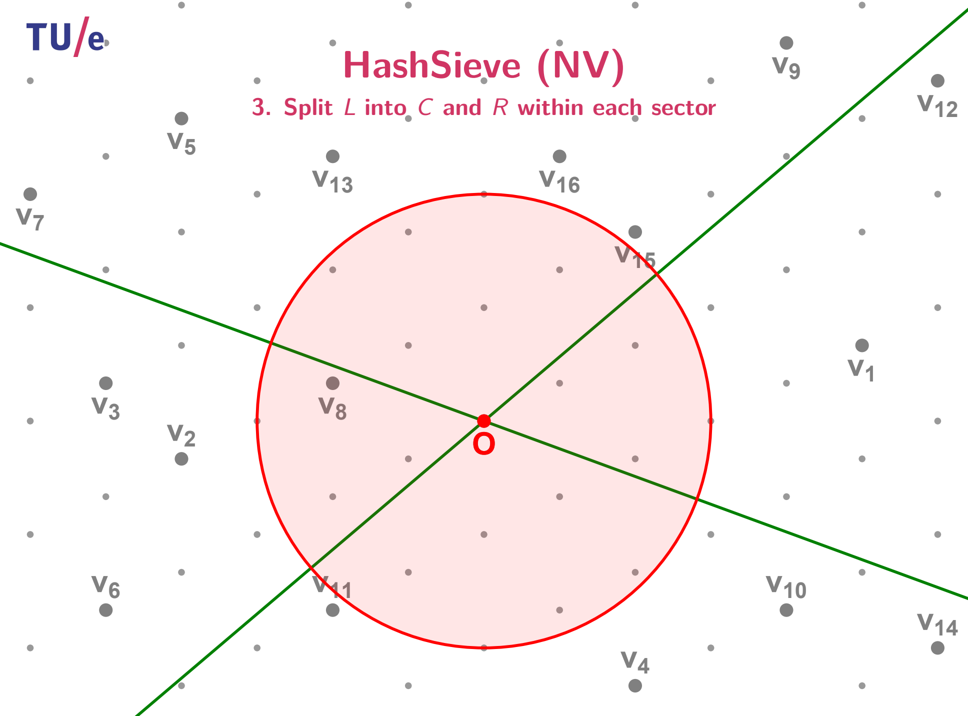
HashSieve (NV)

3. Split L into C and R within each sector



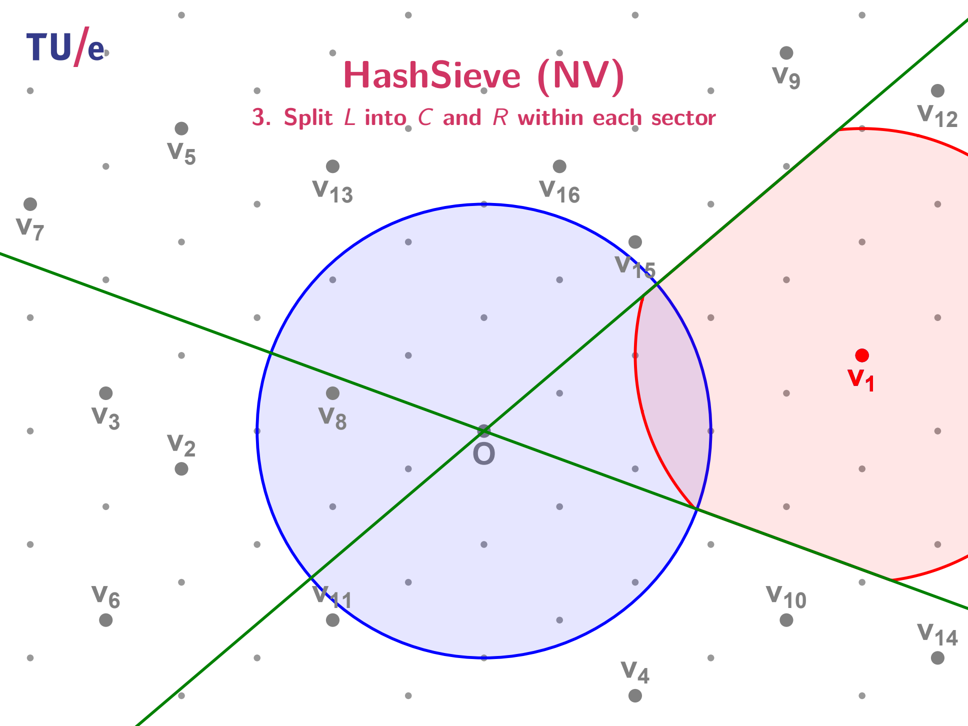
HashSieve (NV)

3. Split L into C and R within each sector



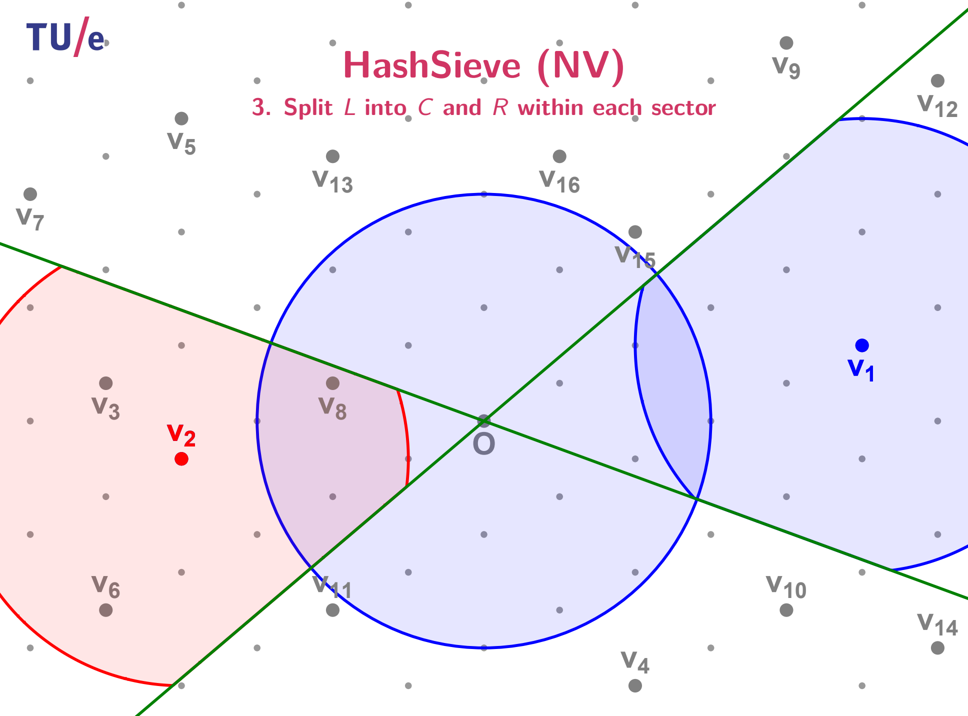
HashSieve (NV)

3. Split L into C and R within each sector



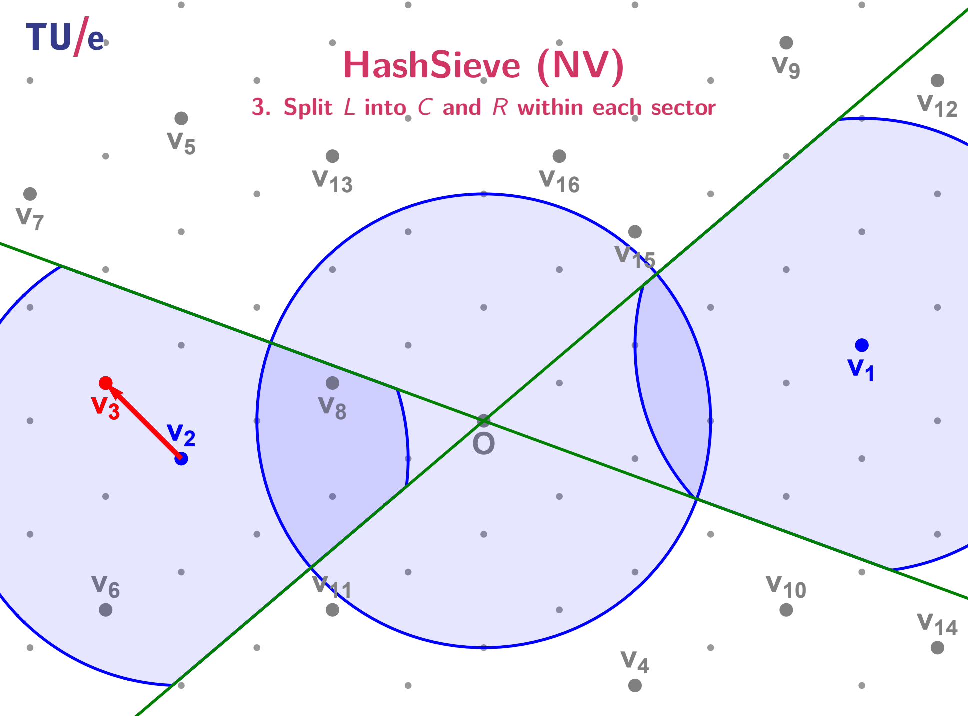
HashSieve (NV)

3. Split L into C and R within each sector



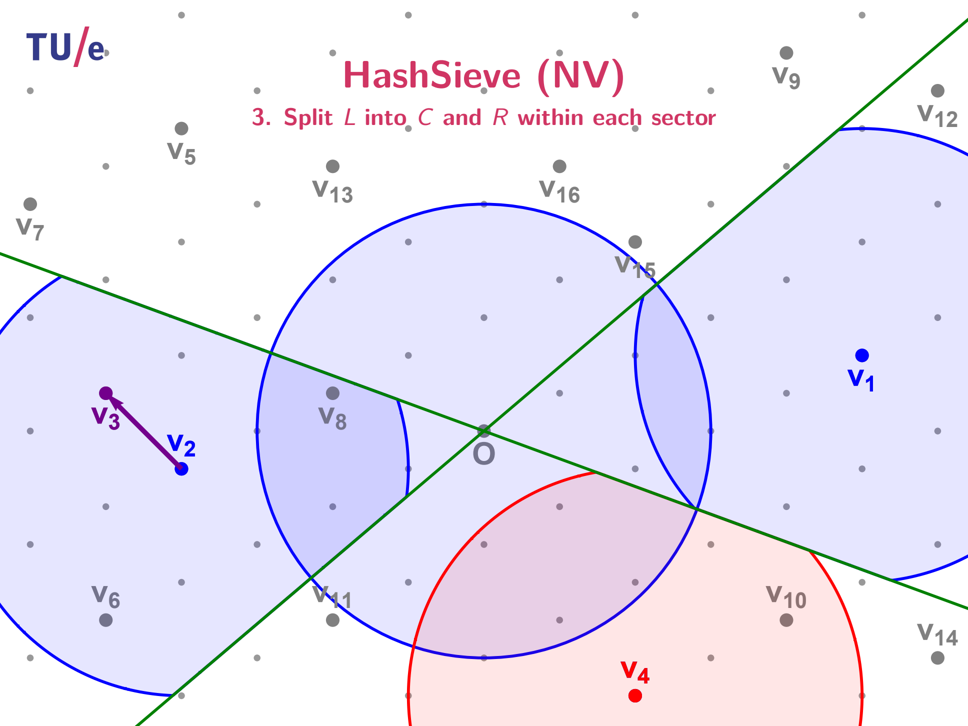
HashSieve (NV)

3. Split L into C and R within each sector



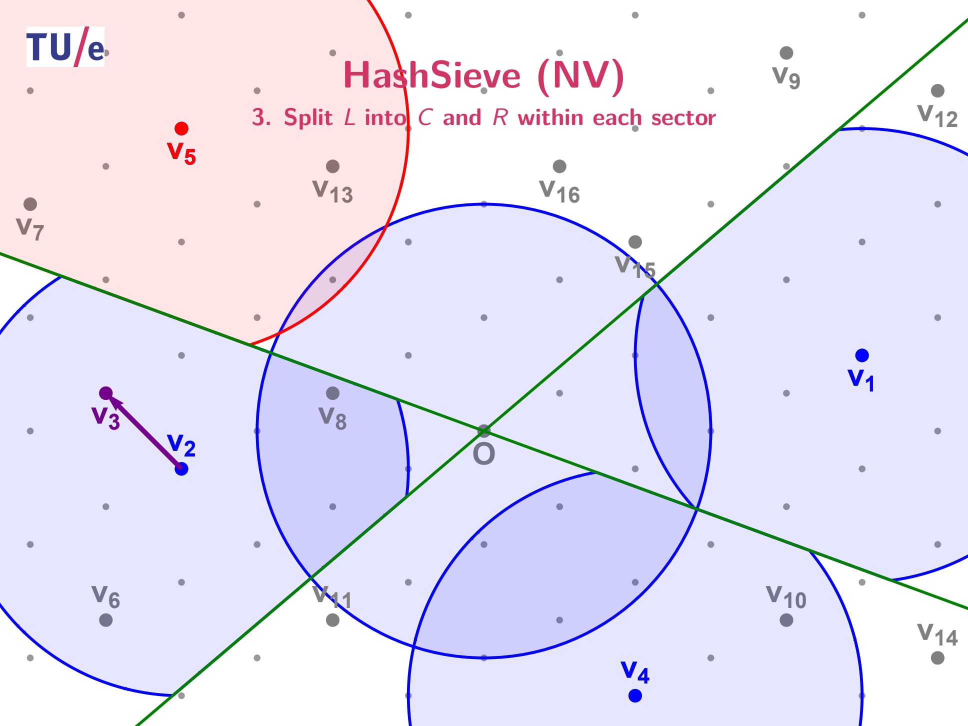
HashSieve (NV)

3. Split L into C and R within each sector



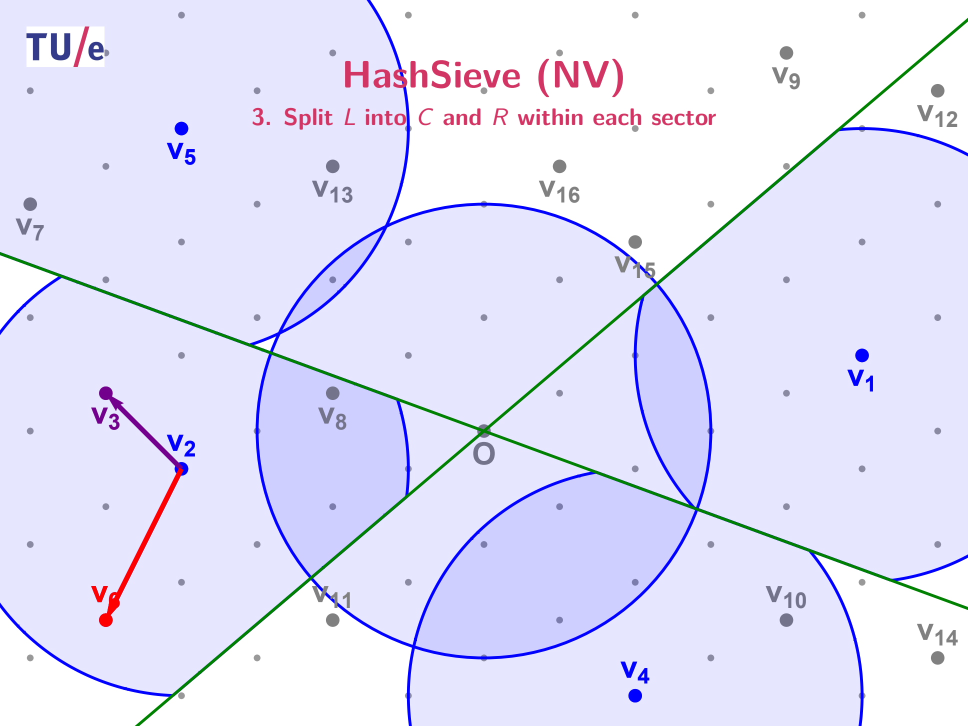
HashSieve (NV)

3. Split L into C and R within each sector



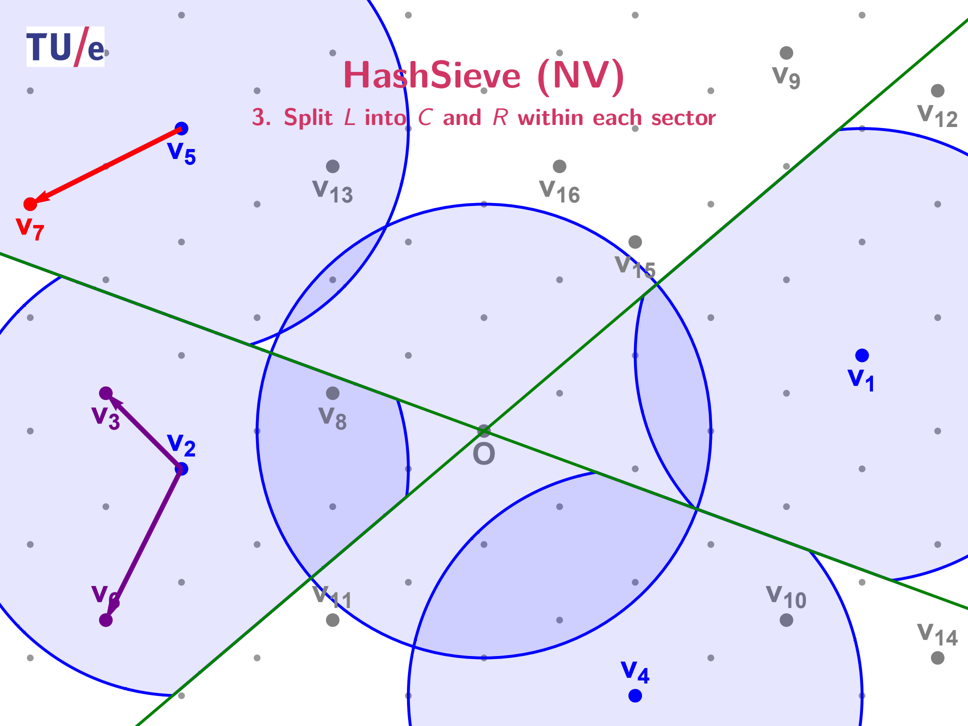
HashSieve (NV)

3. Split L into C and R within each sector



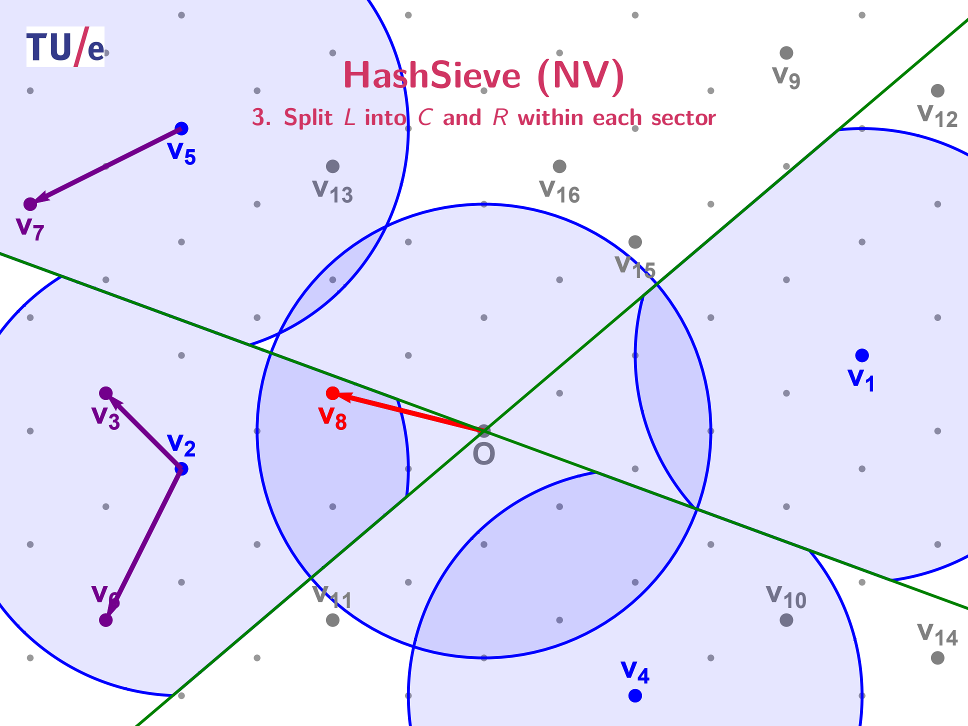
HashSieve (NV)

3. Split L into C and R within each sector



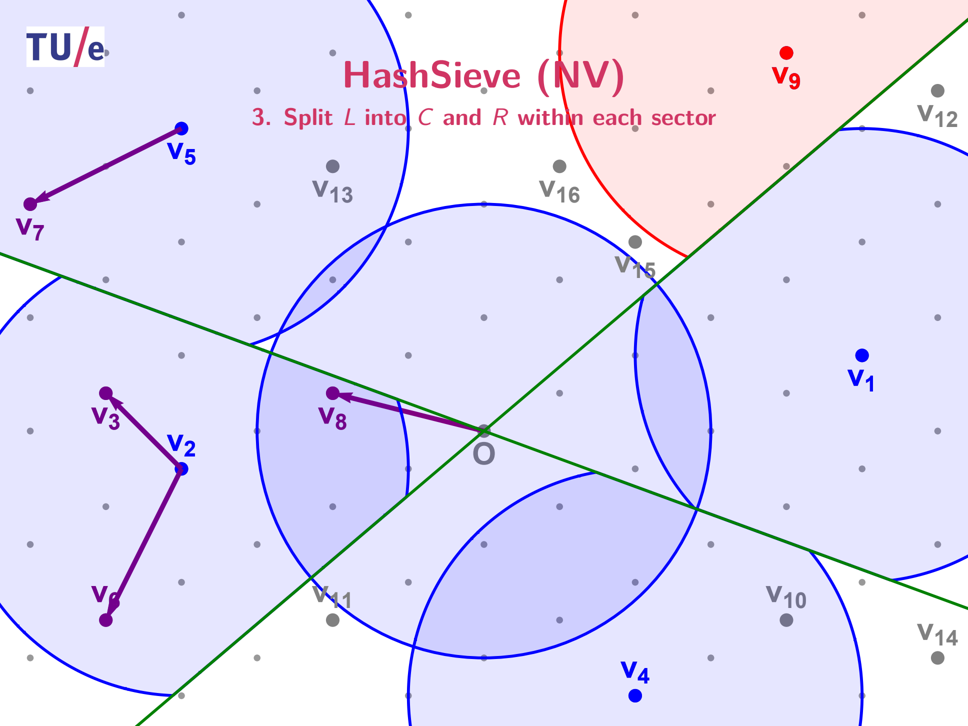
HashSieve (NV)

3. Split L into C and R within each sector



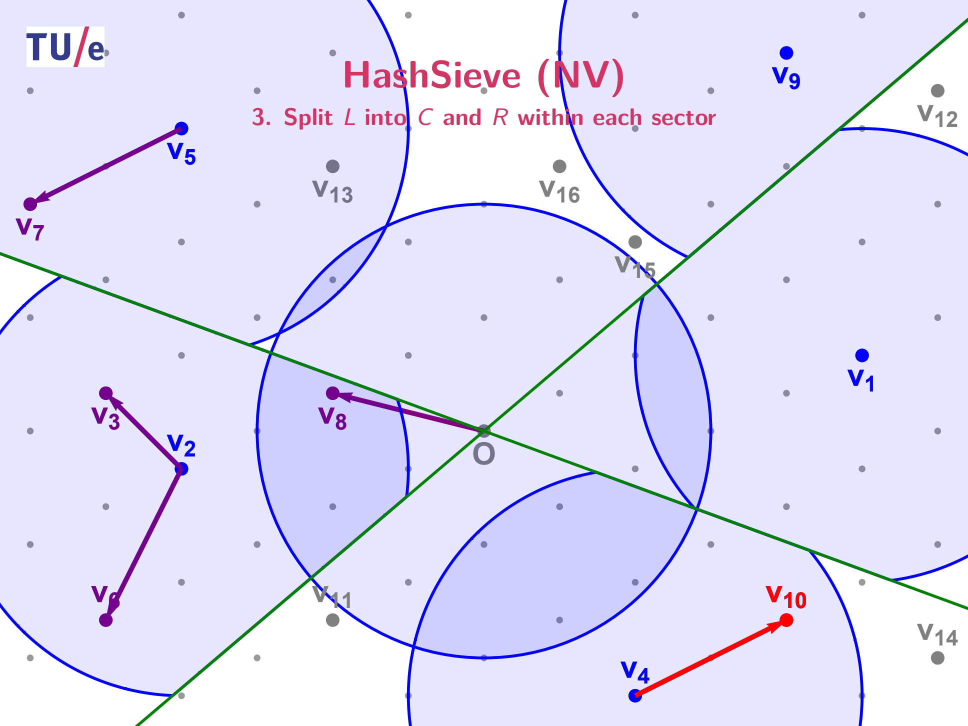
HashSieve (NV)

3. Split L into C and R within each sector



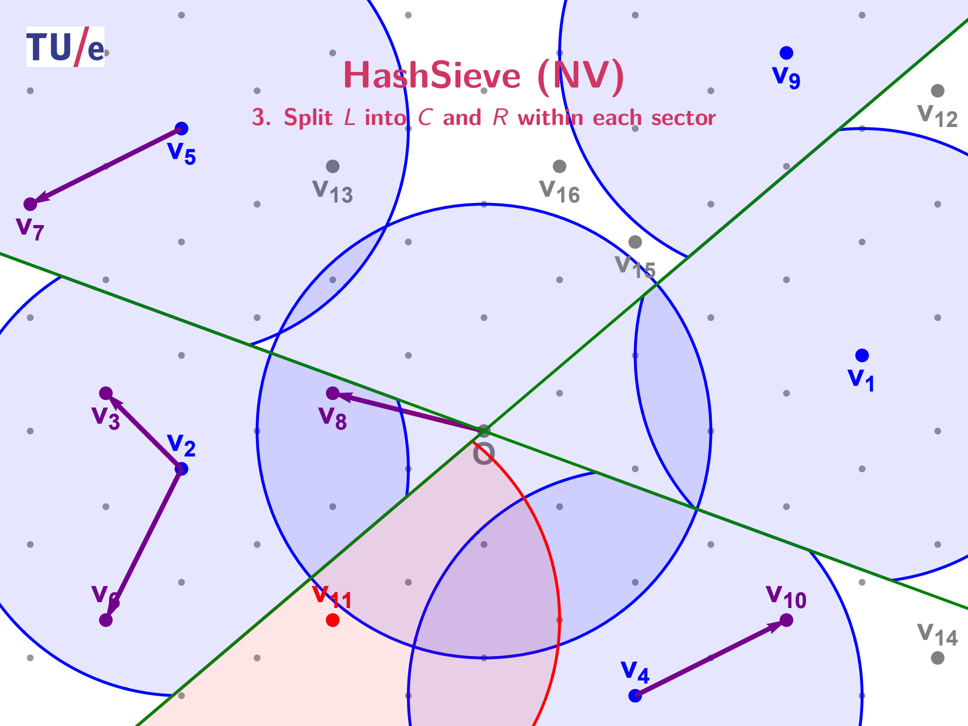
HashSieve (NV)

3. Split L into C and R within each sector



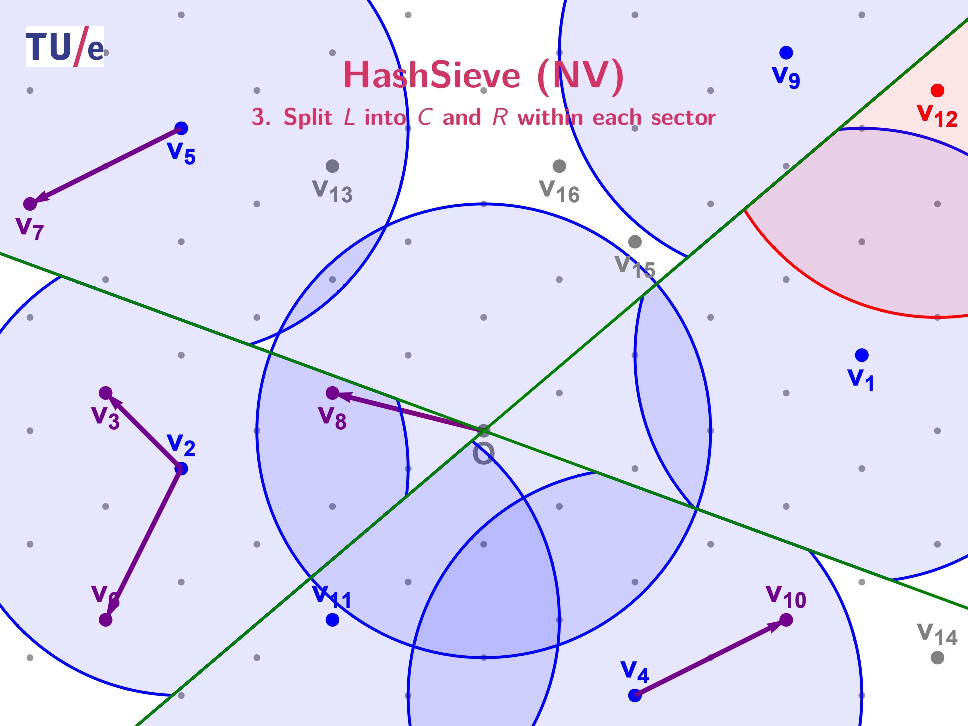
HashSieve (NV)

3. Split L into C and R within each sector



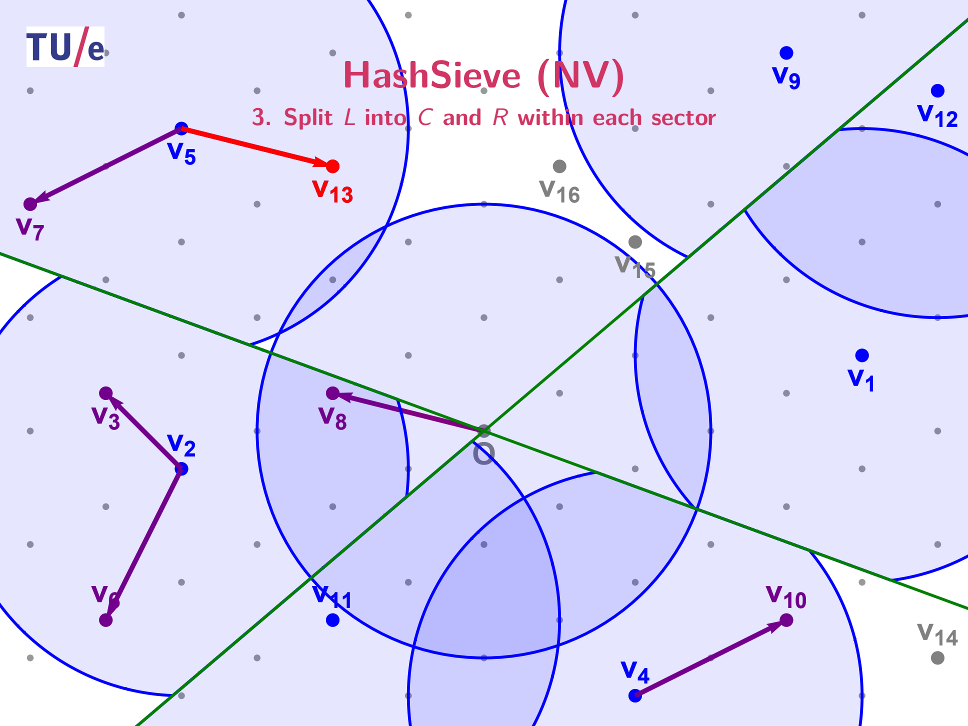
HashSieve (NV)

3. Split L into C and R within each sector



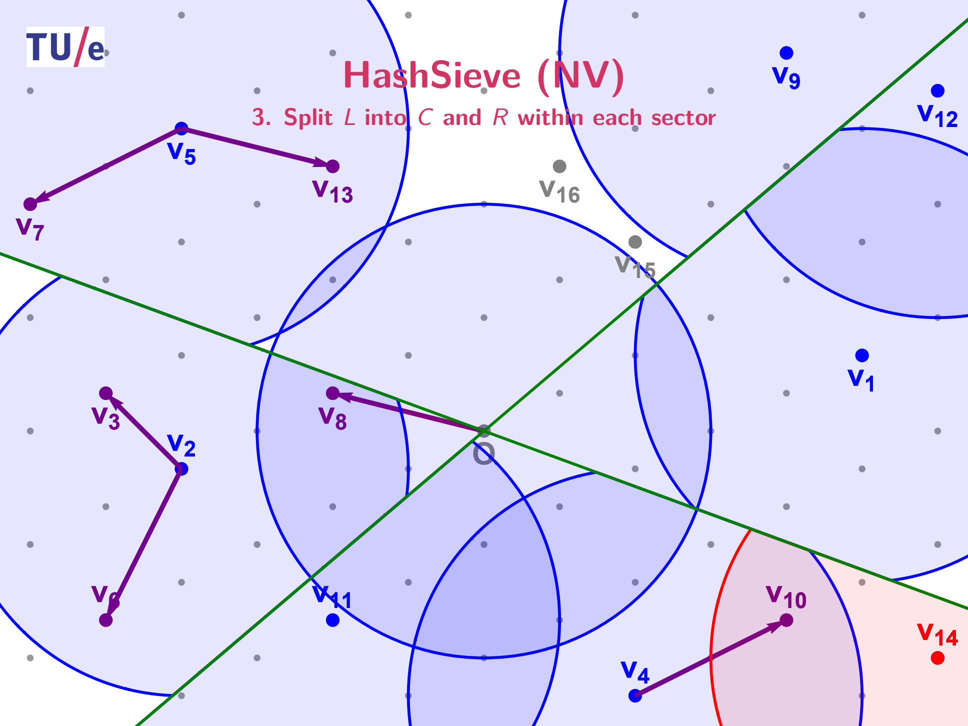
HashSieve (NV)

3. Split L into C and R within each sector



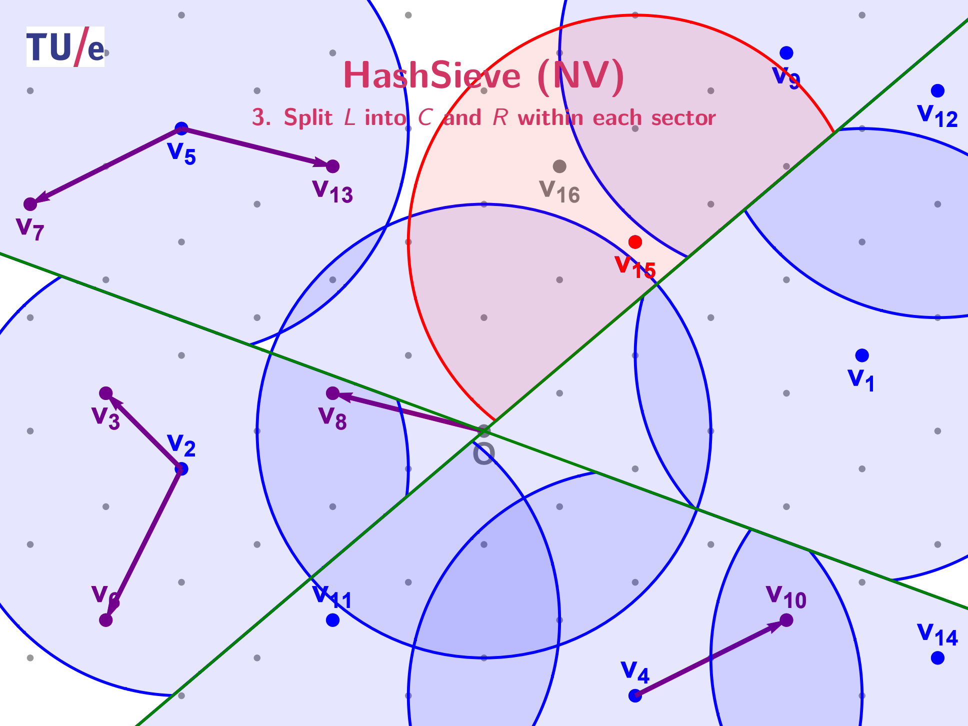
HashSieve (NV)

3. Split L into C and R within each sector



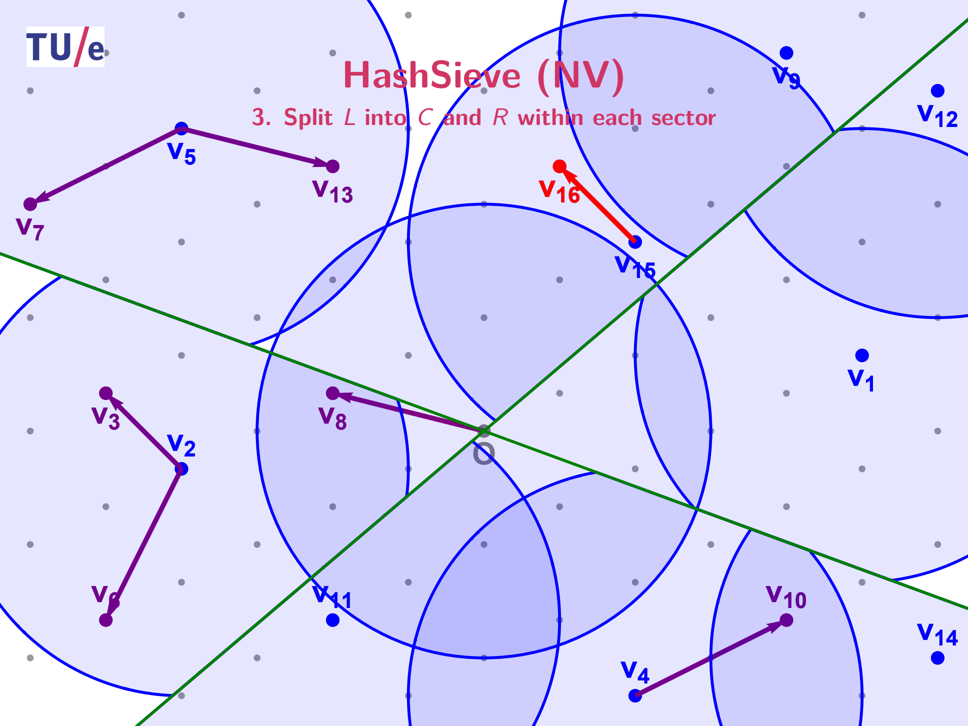
HashSieve (NV)

3. Split L into C and R within each sector



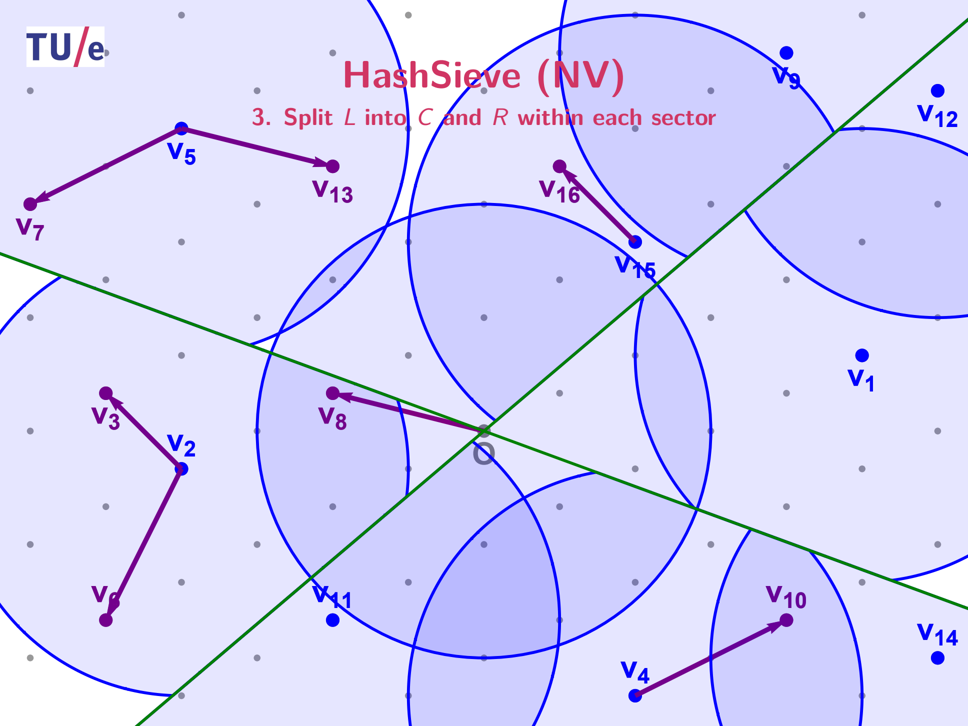
HashSieve (NV)

3. Split L into C and R within each sector



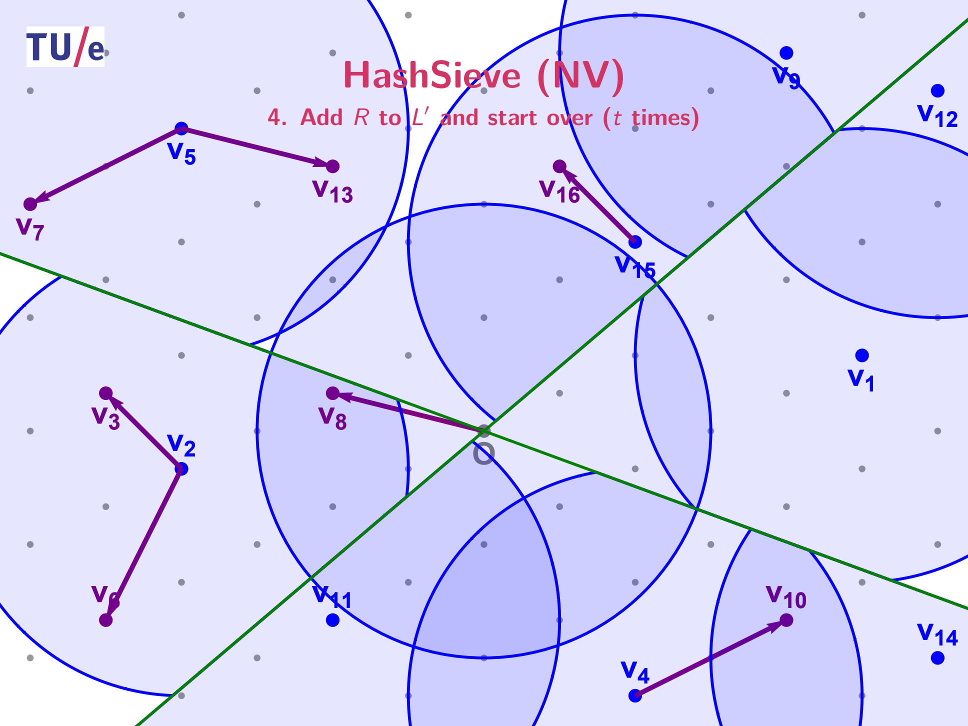
HashSieve (NV)

3. Split L into C and R within each sector



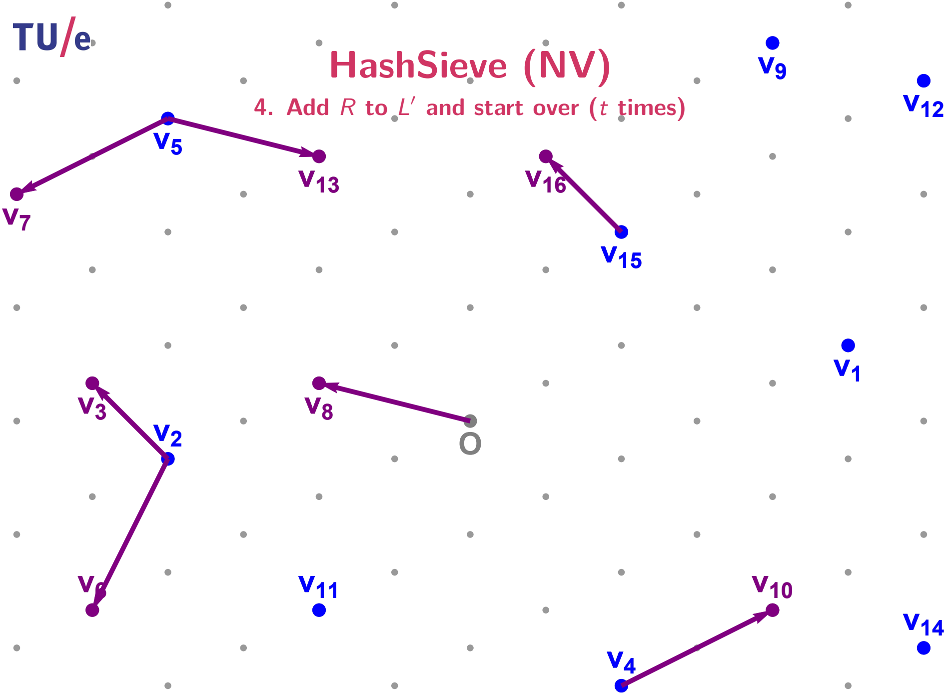
HashSieve (NV)

4. Add R to L' and start over (t times)



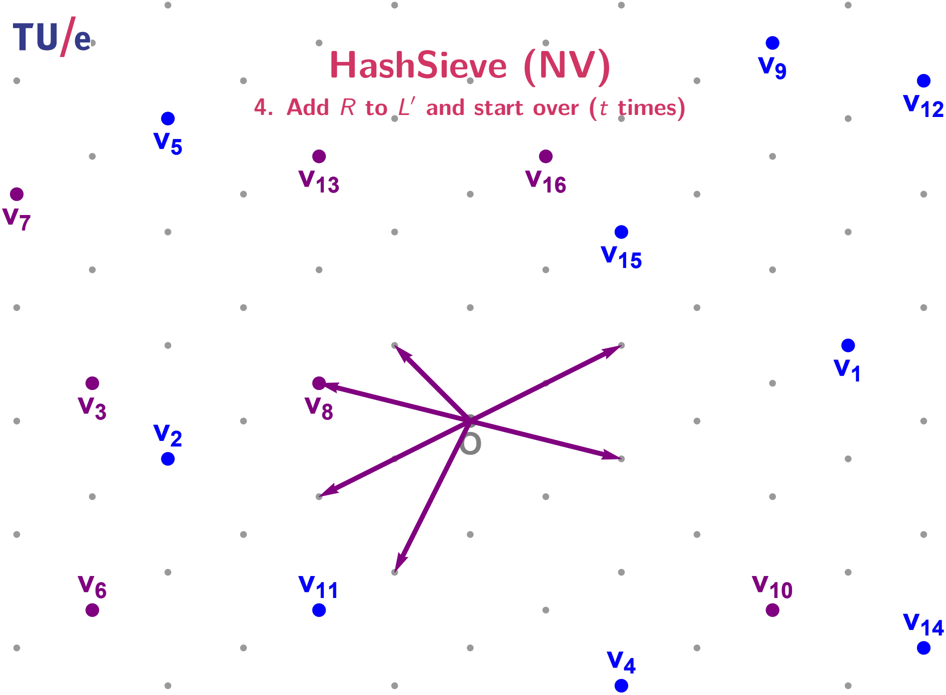
HashSieve (NV)

4. Add R to L' and start over (t times)



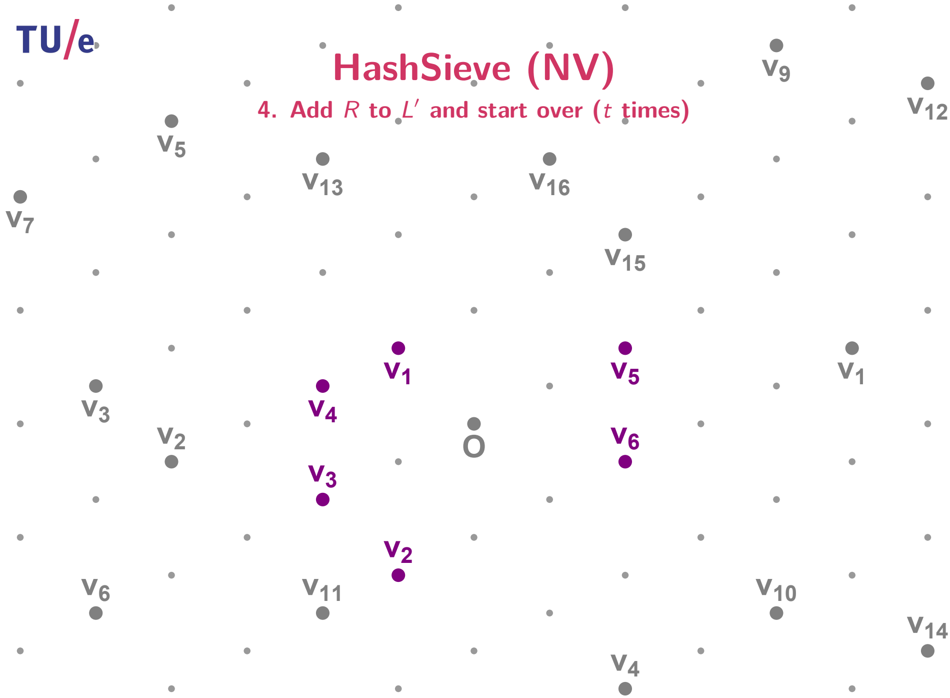
HashSieve (NV)

4. Add R to L' and start over (t times)



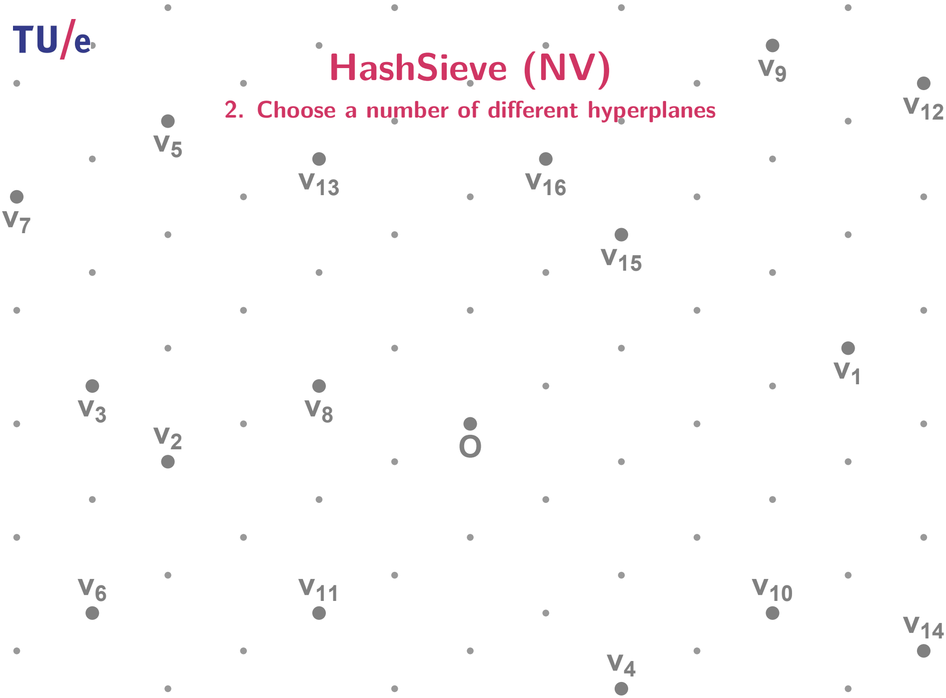
HashSieve (NV)

4. Add R to L' and start over (t times)



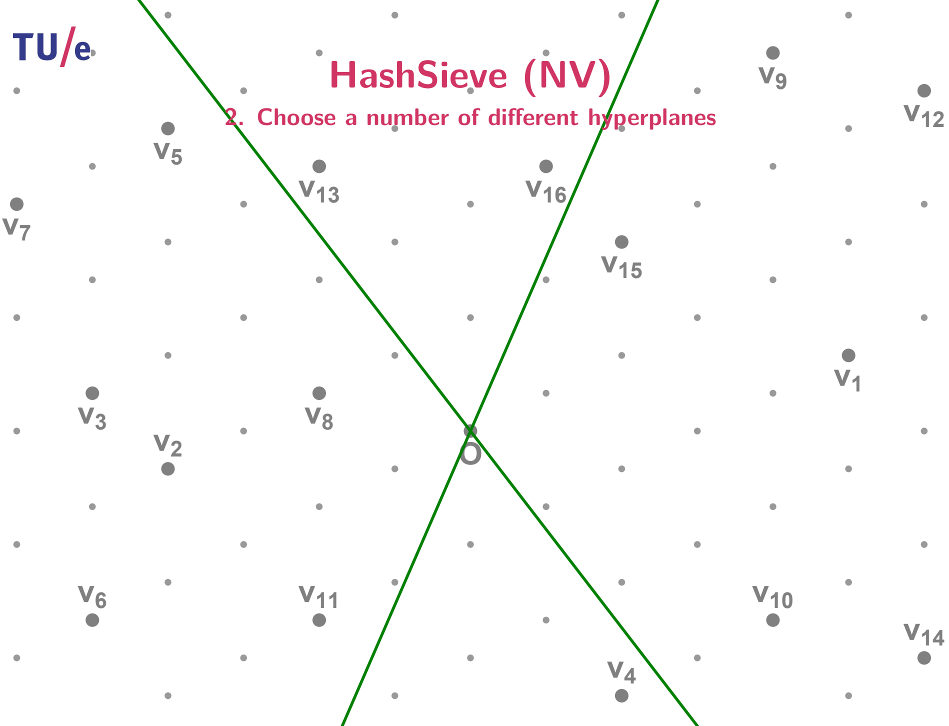
HashSieve (NV)

2. Choose a number of different hyperplanes



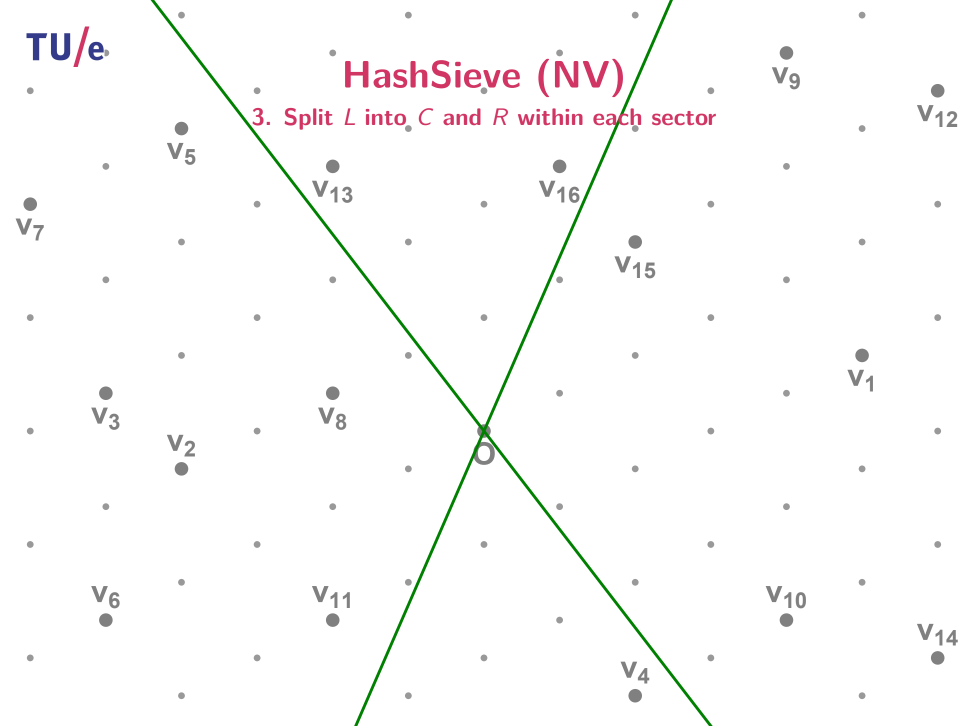
HashSieve (NV)

2. Choose a number of different hyperplanes



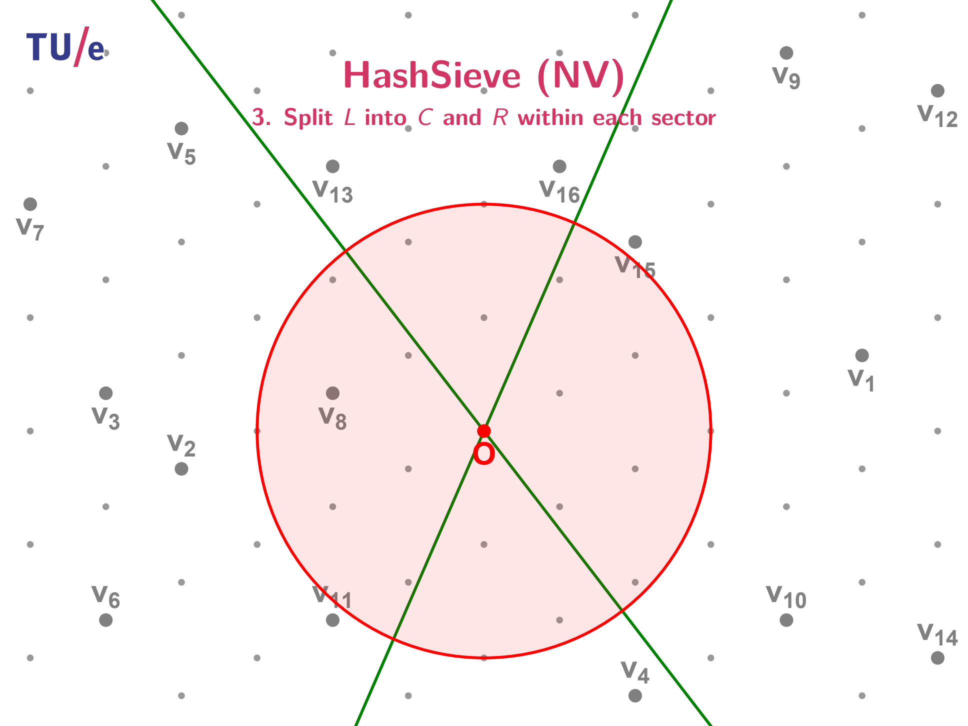
HashSieve (NV)

3. Split L into C and R within each sector



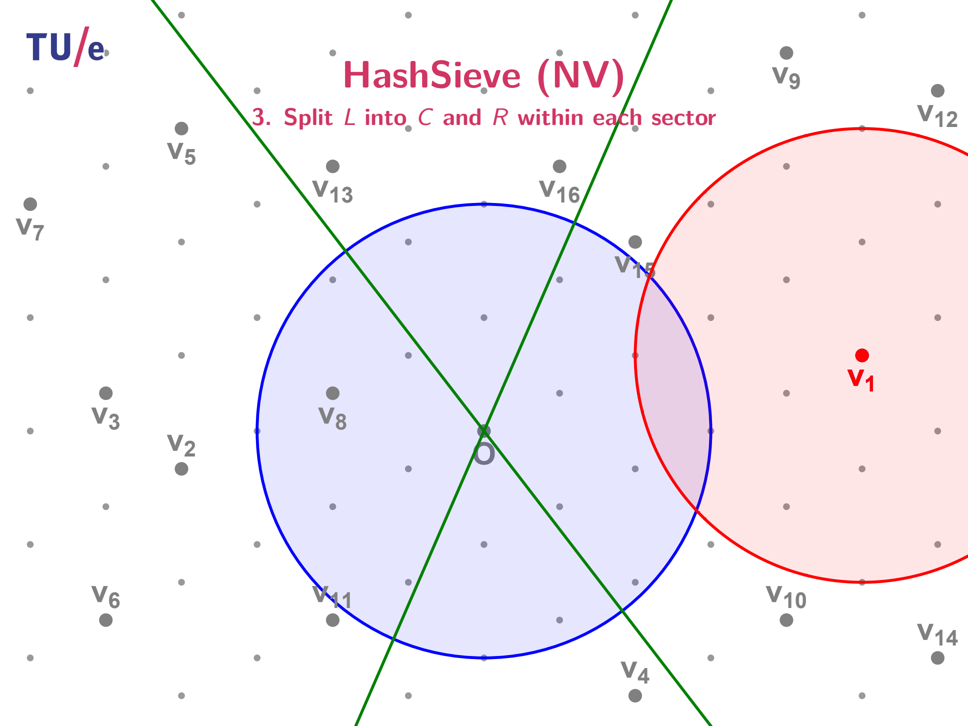
HashSieve (NV)

3. Split L into C and R within each sector



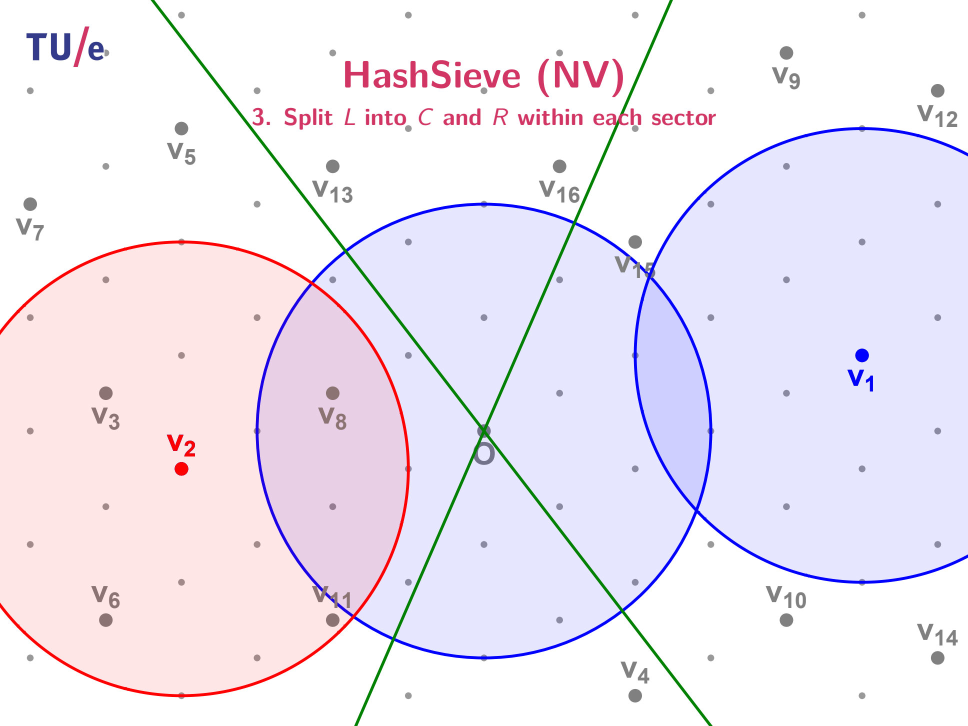
HashSieve (NV)

3. Split L into C and R within each sector



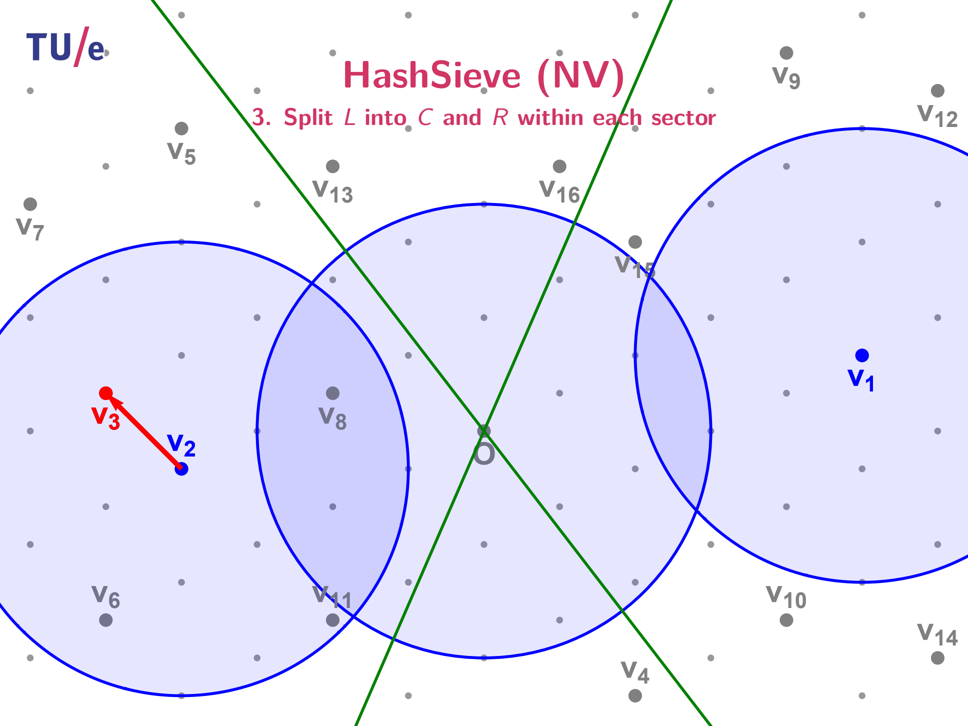
HashSieve (NV)

3. Split L into C and R within each sector



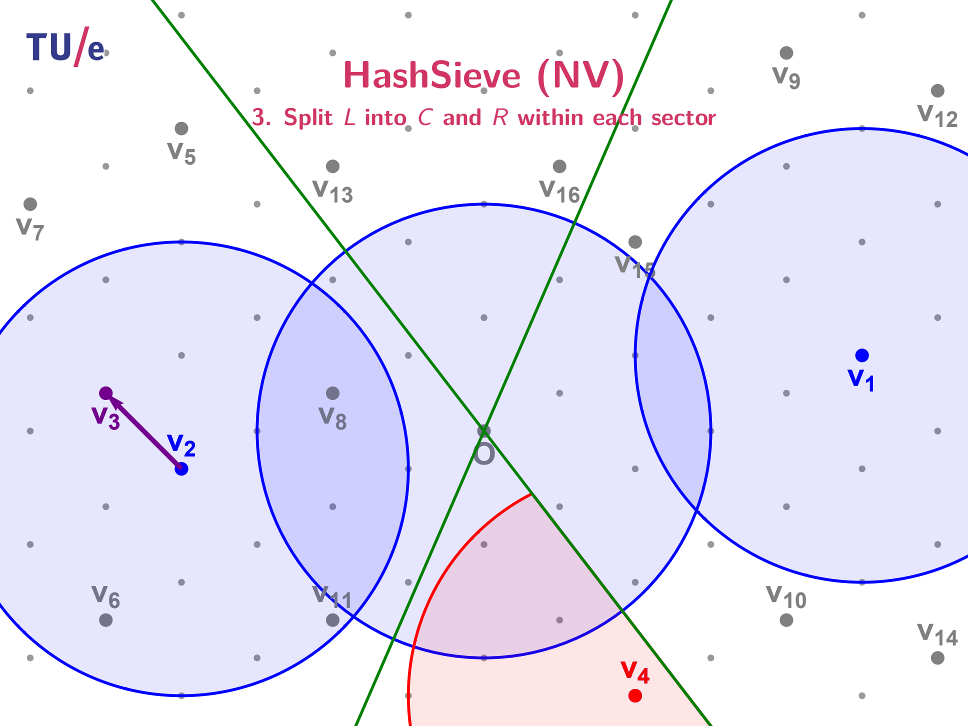
HashSieve (NV)

3. Split L into C and R within each sector



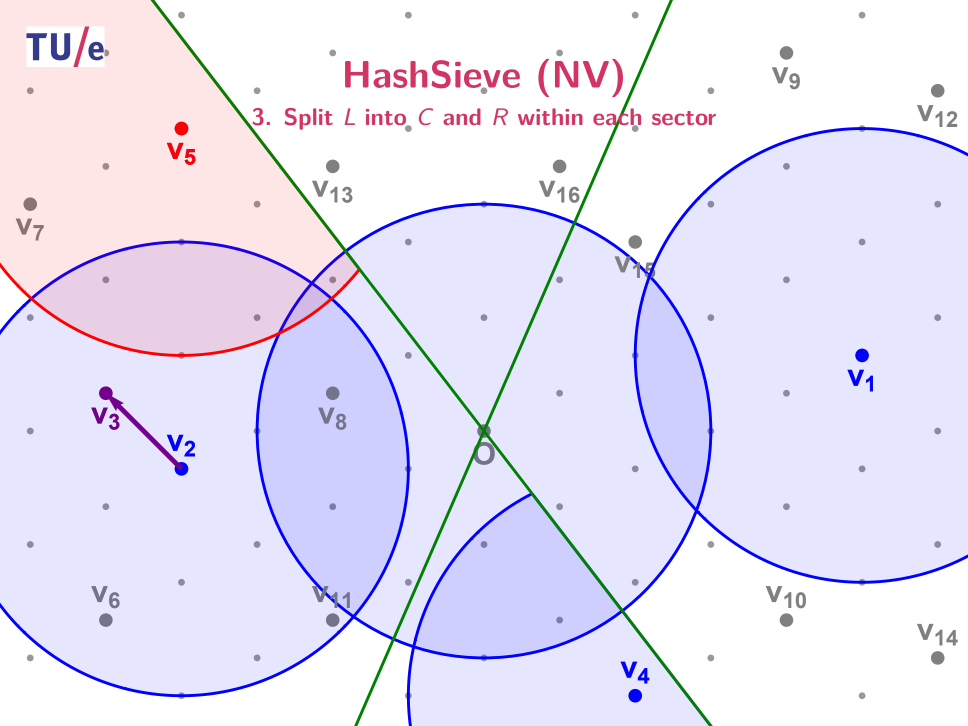
HashSieve (NV)

3. Split L into C and R within each sector



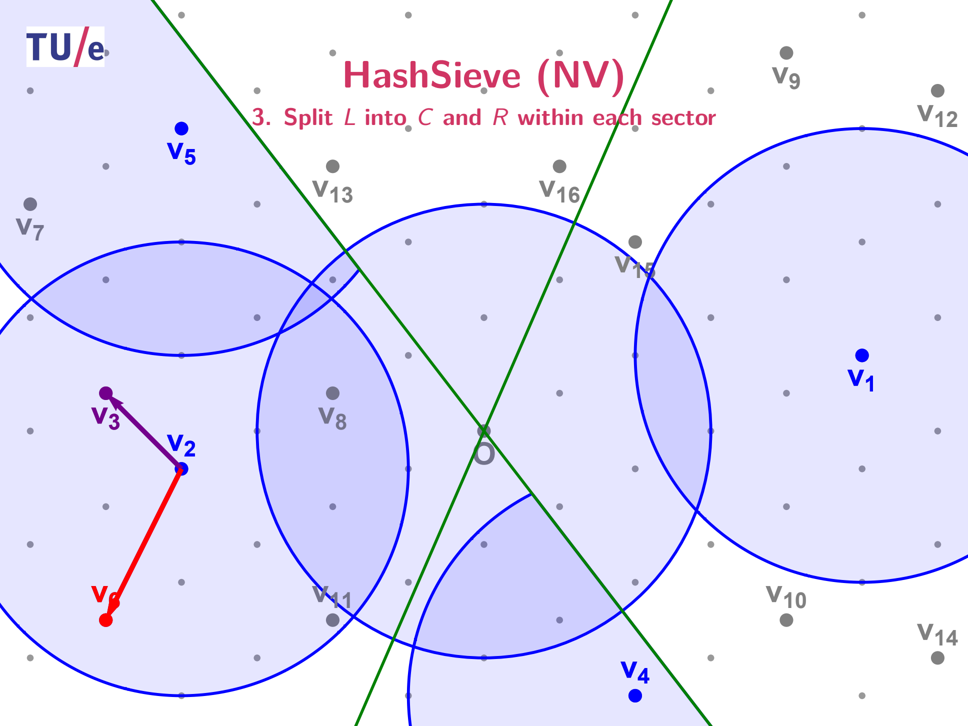
HashSieve (NV)

3. Split L into C and R within each sector



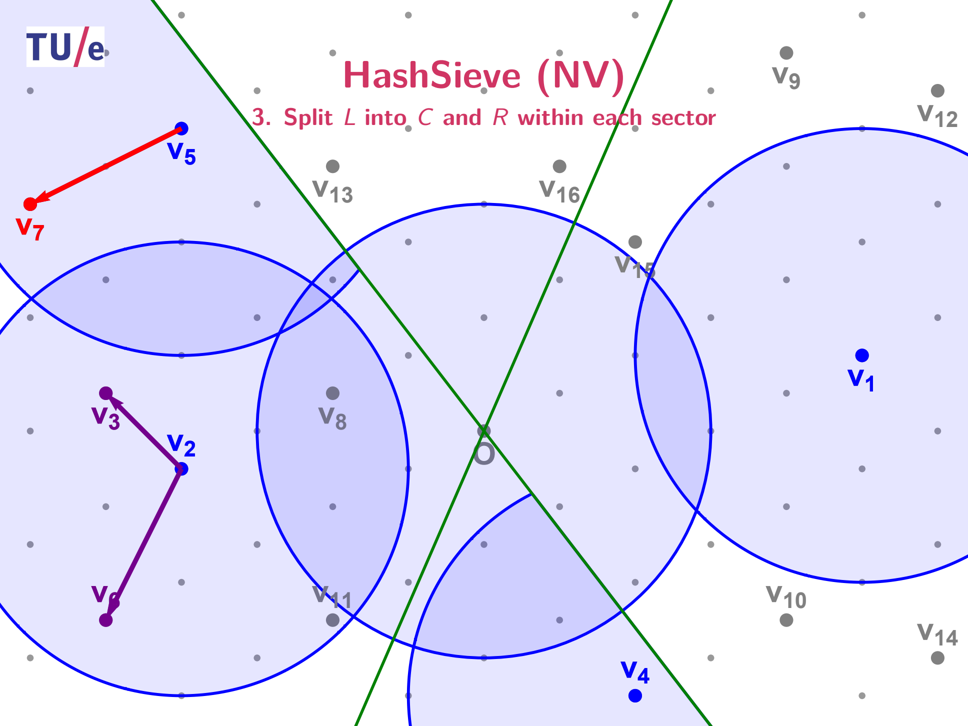
HashSieve (NV)

3. Split L into C and R within each sector



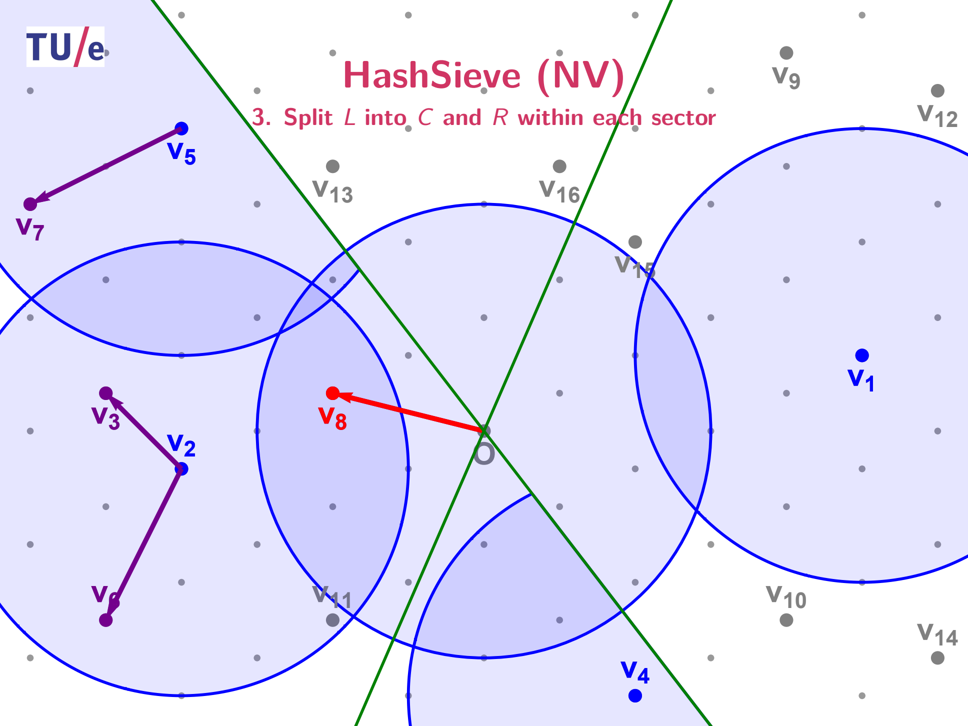
HashSieve (NV)

3. Split L into C and R within each sector



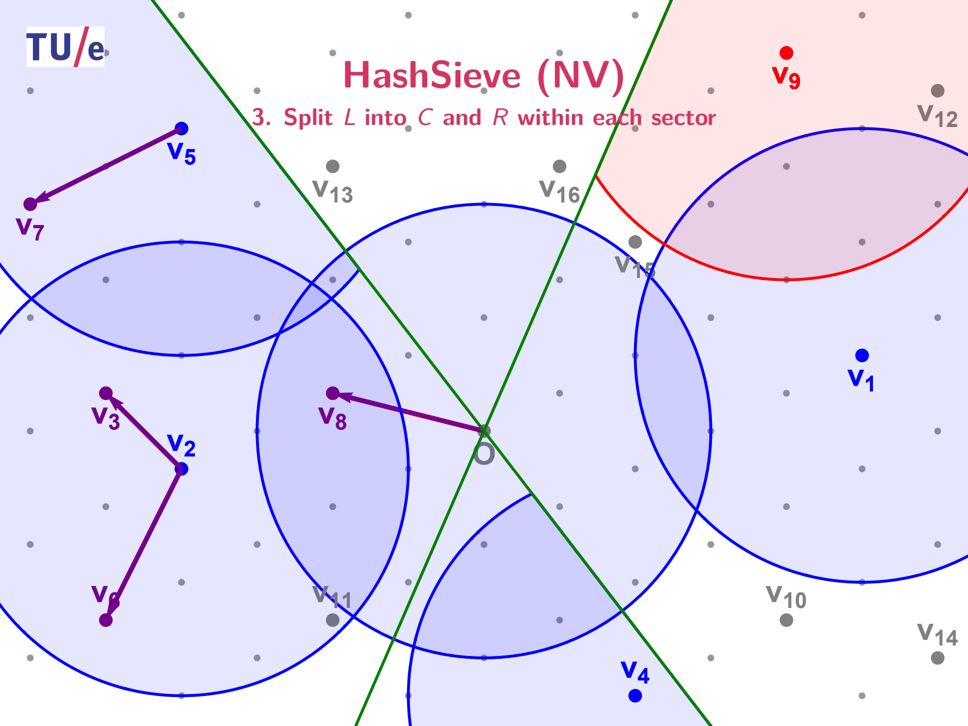
HashSieve (NV)

3. Split L into C and R within each sector



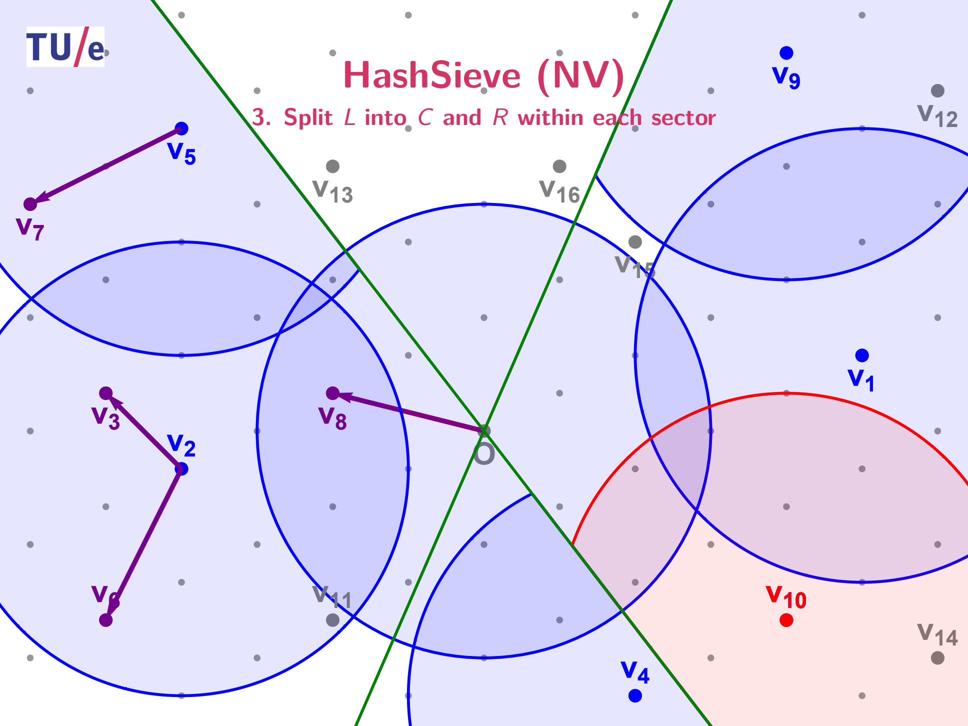
HashSieve (NV)

3. Split L into C and R within each sector



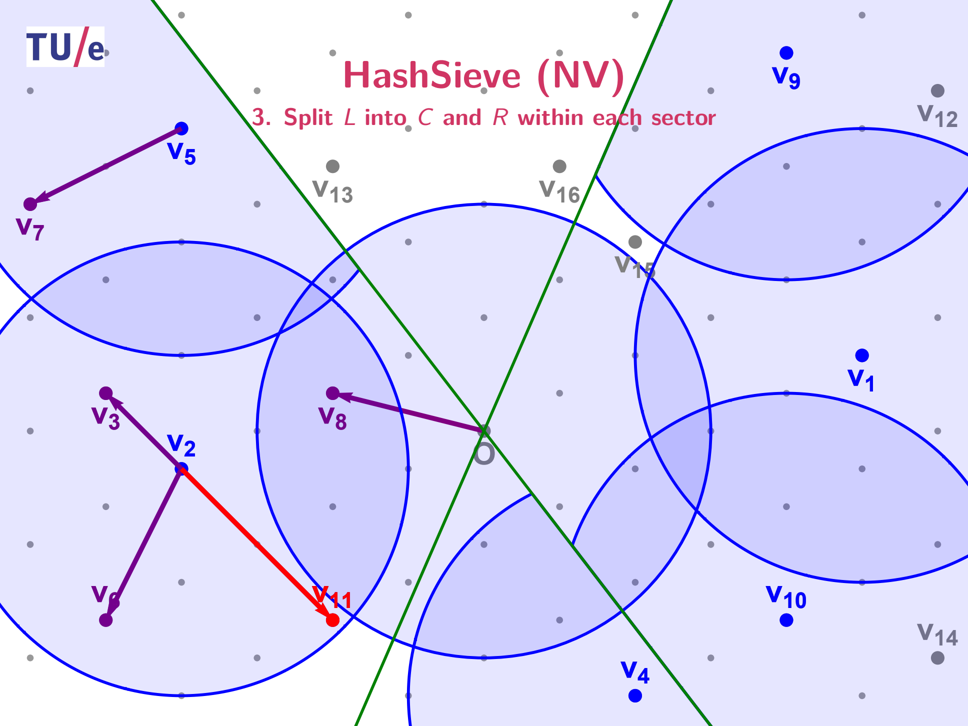
HashSieve (NV)

3. Split L into C and R within each sector



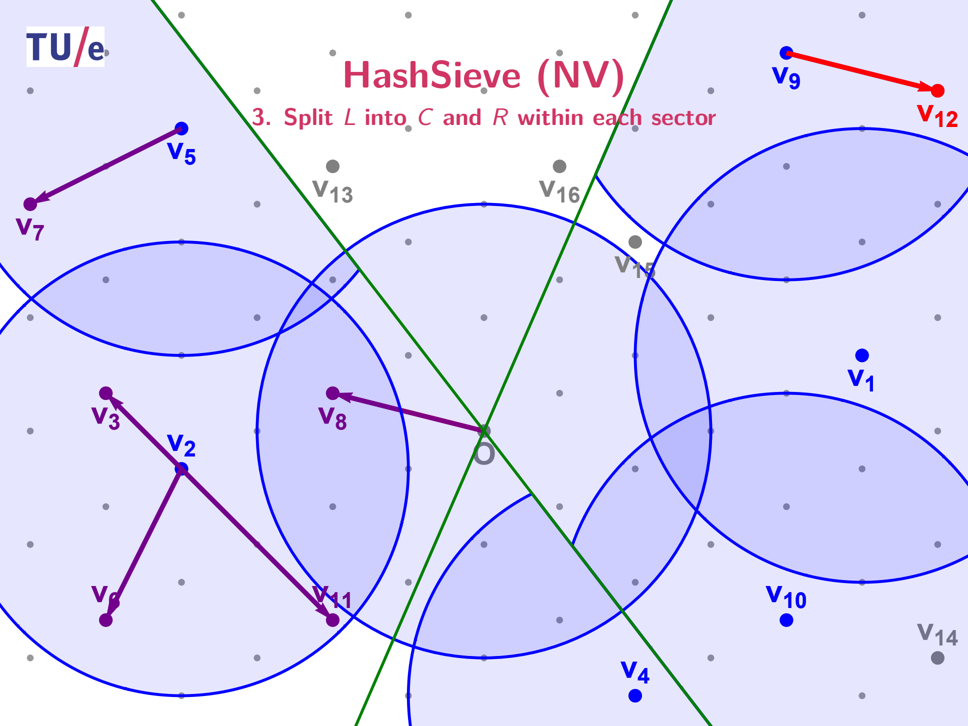
HashSieve (NV)

3. Split L into C and R within each sector



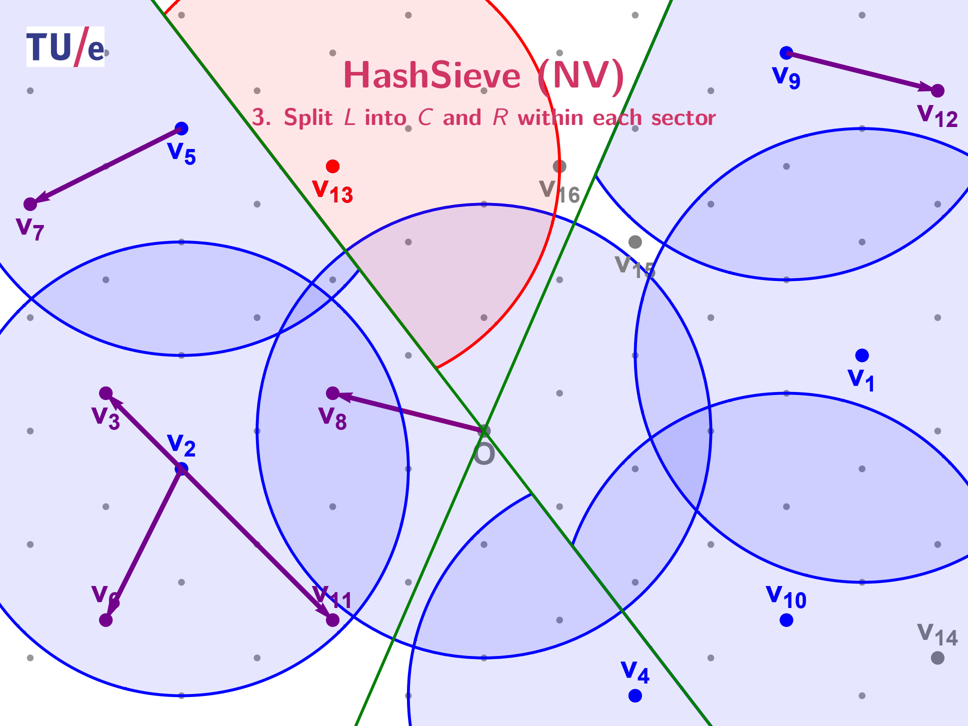
HashSieve (NV)

3. Split L into C and R within each sector



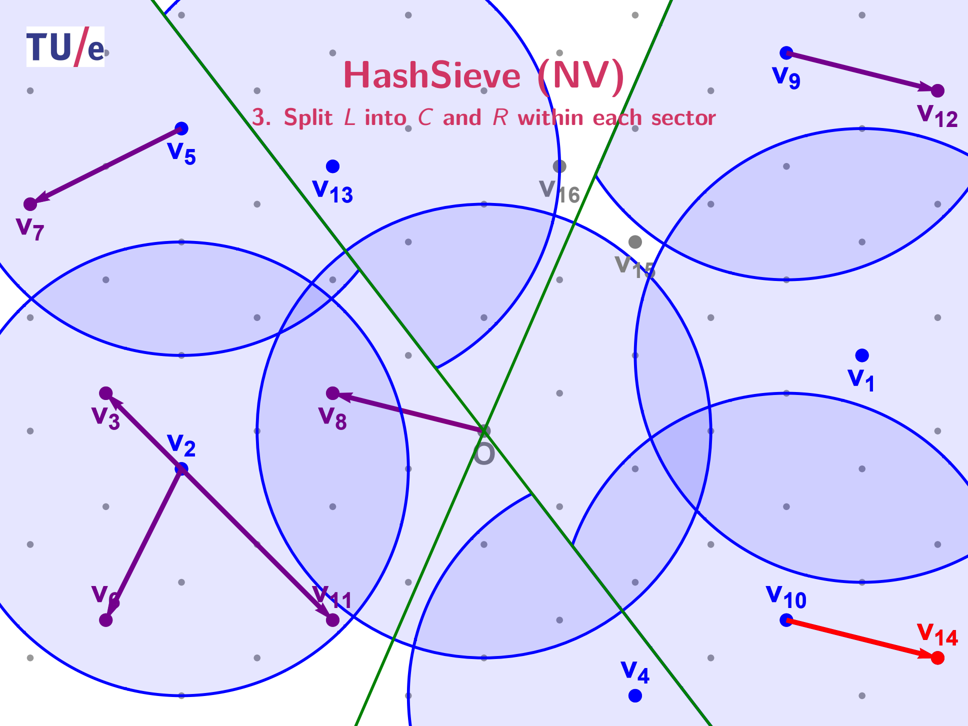
HashSieve (NV)

3. Split L into C and R within each sector



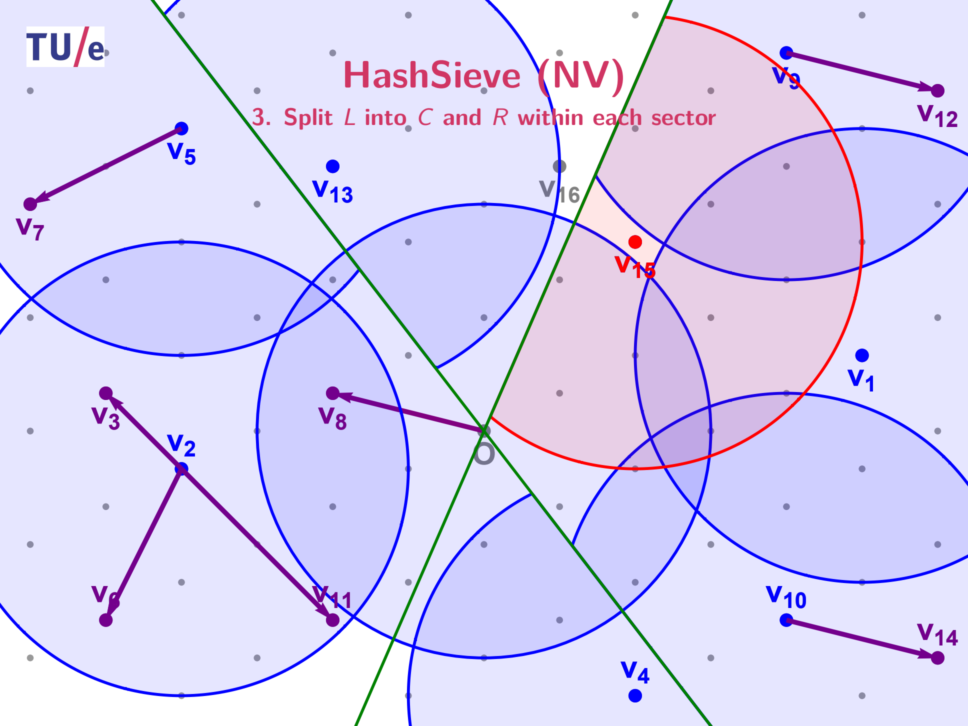
HashSieve (NV)

3. Split L into C and R within each sector



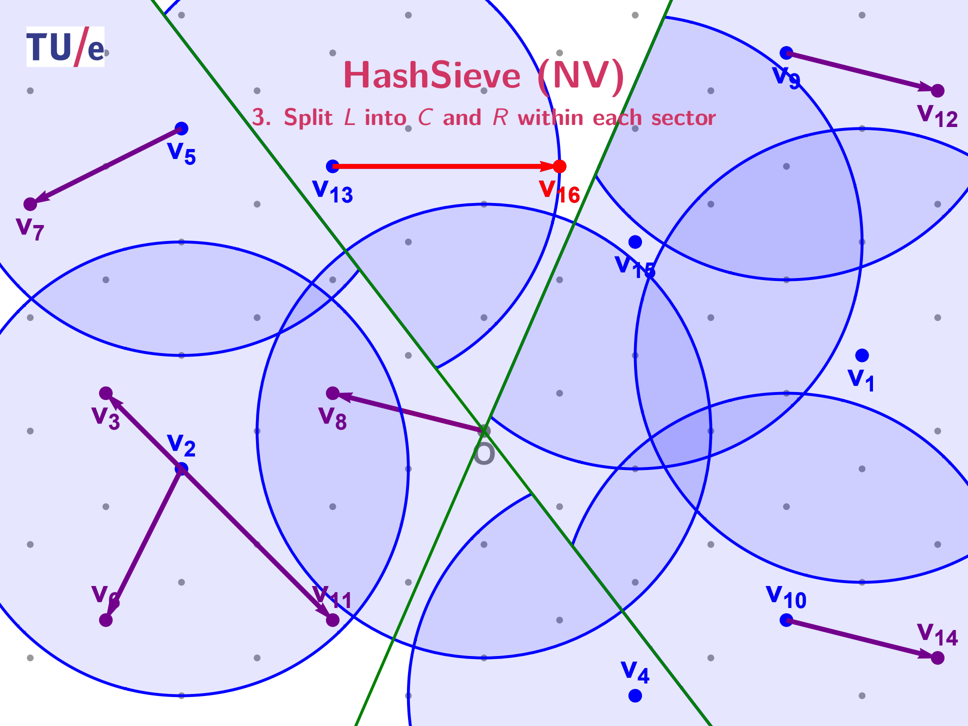
HashSieve (NV)

3. Split L into C and R within each sector



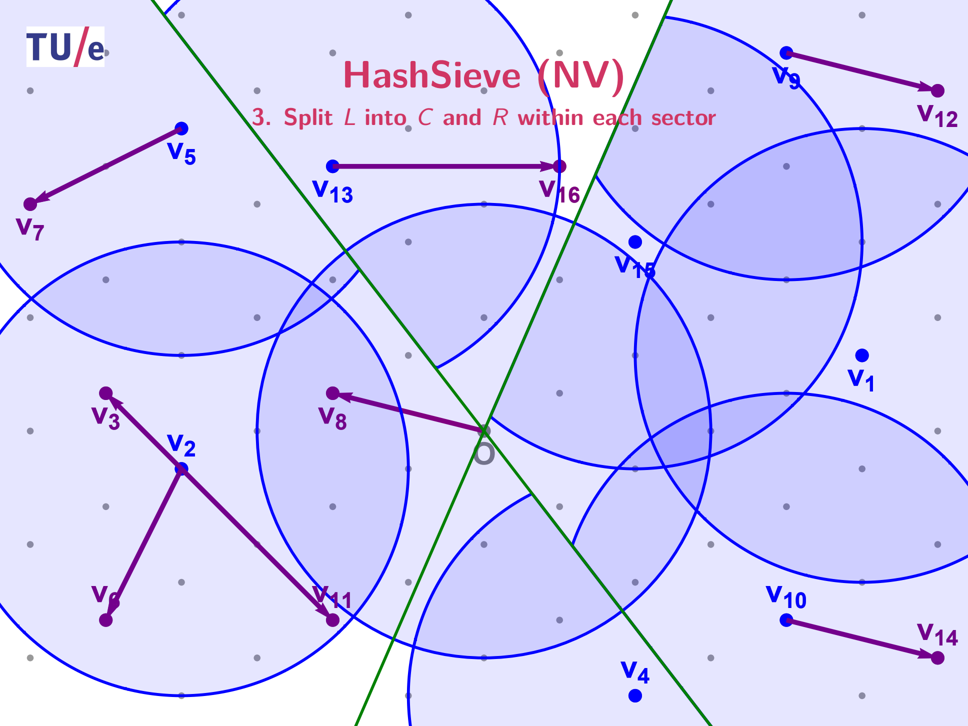
HashSieve (NV)

3. Split L into C and R within each sector



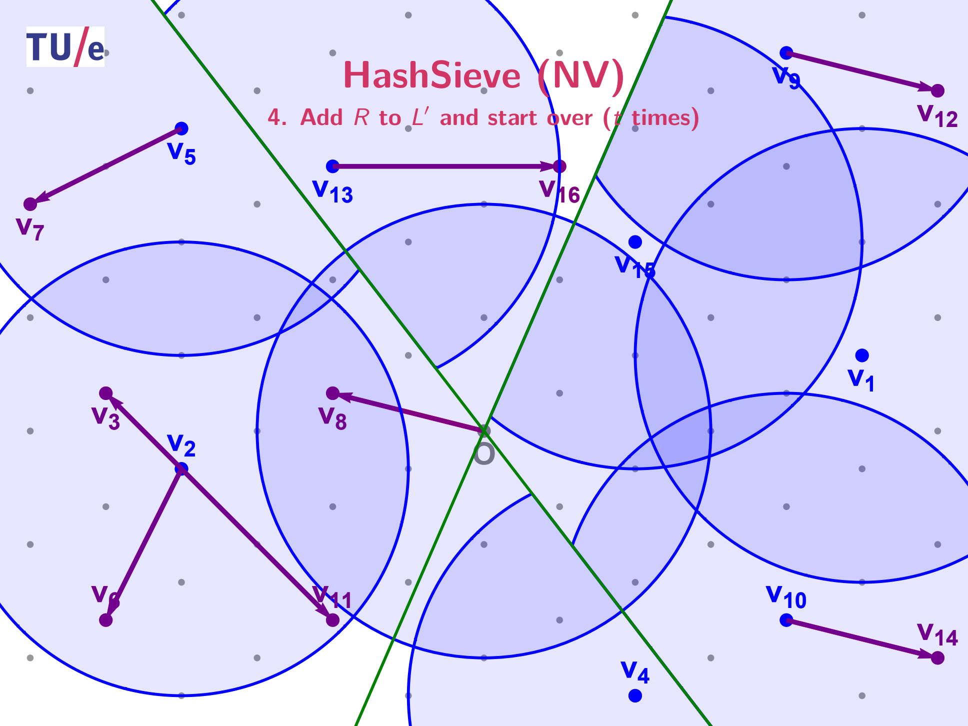
HashSieve (NV)

3. Split L into C and R within each sector



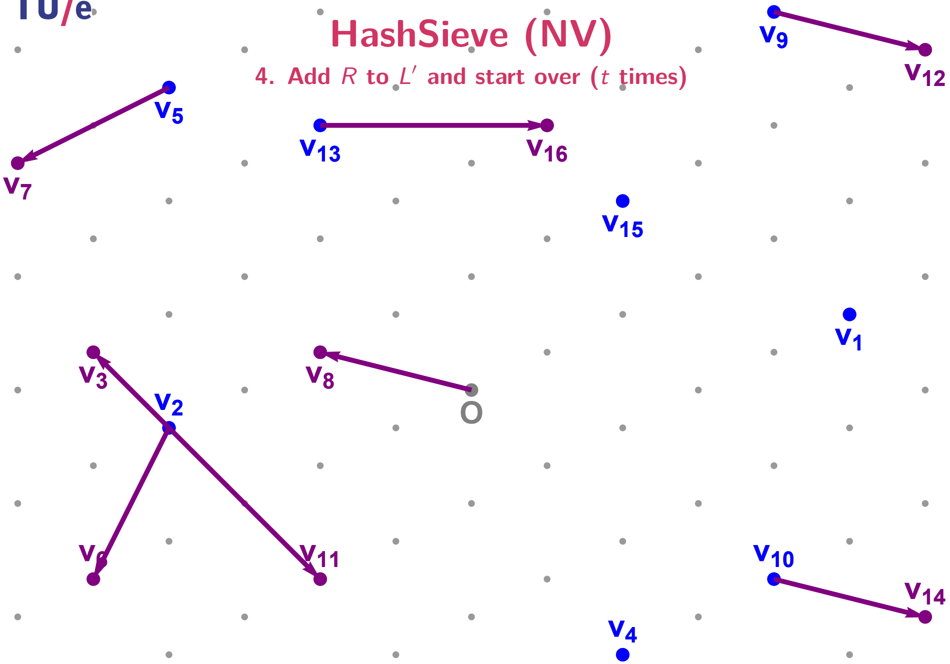
HashSieve (NV)

4. Add R to L' and start over (t times)



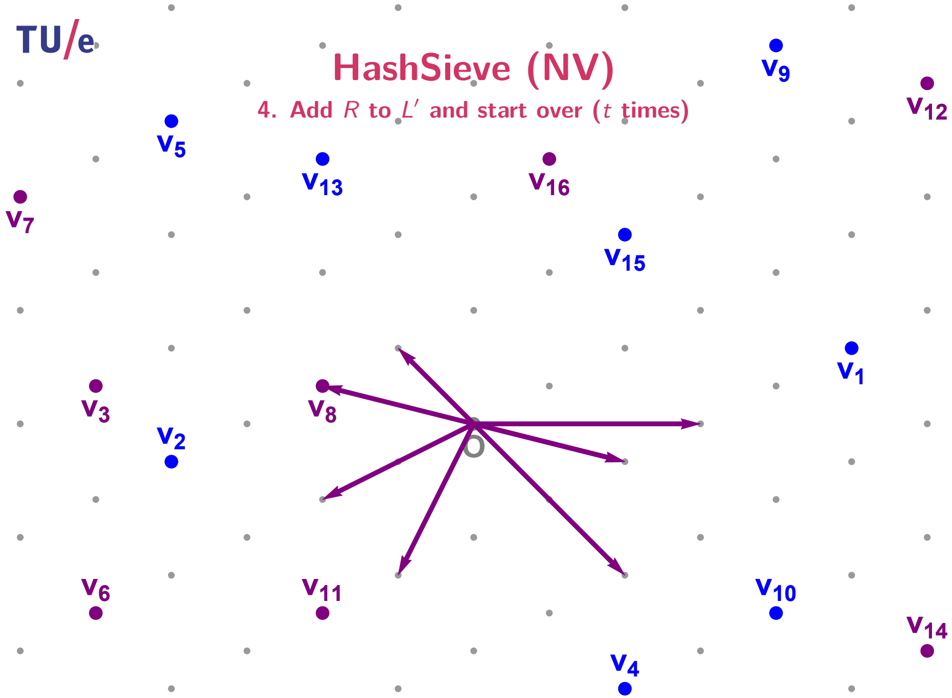
HashSieve (NV)

4. Add R to L' and start over (t times)



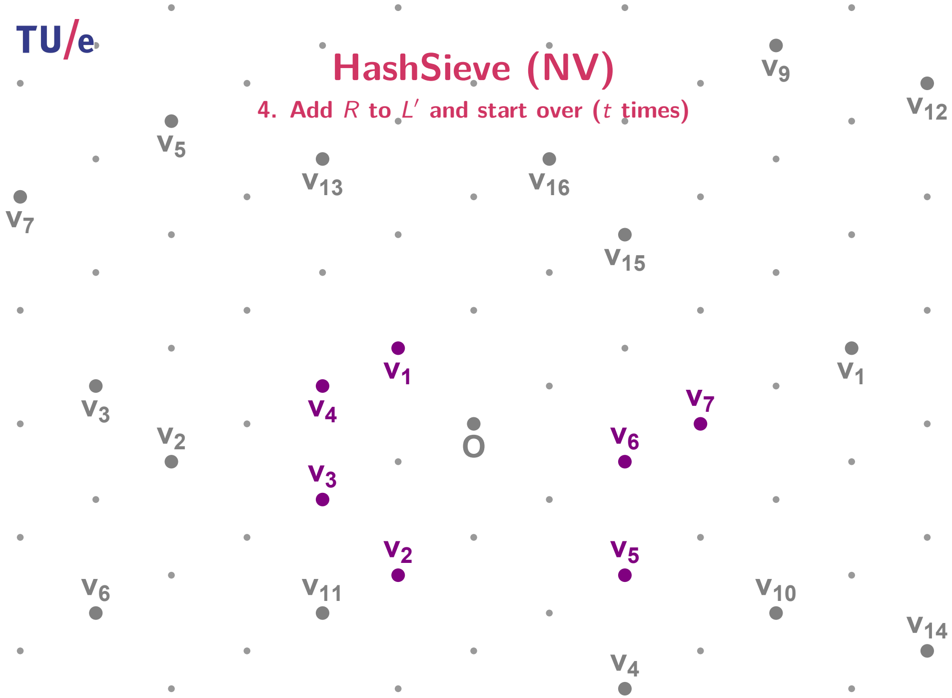
HashSieve (NV)

4. Add R to L' and start over (t times)



HashSieve (NV)

4. Add R to L' and start over (t times)



HashSieve (NV)

Overview

HashSieve (NV)

Overview

- Same k (hyperplanes) and t (hash tables) as before

HashSieve (NV)

Overview

- Same k (hyperplanes) and t (hash tables) as before
- Space complexity: $2^{0.21n+o(n)}$
  - Before: store $2^{0.13n}$ hash tables containing all $2^{0.21n}$ vectors
  - Now: process $2^{0.13n}$ hash tables one by one

HashSieve (NV)

Overview

- Same k (hyperplanes) and t (hash tables) as before
- Space complexity: $2^{0.21n+o(n)}$
  - Before: store $2^{0.13n}$ hash tables containing all $2^{0.21n}$ vectors
  - Now: process $2^{0.13n}$ hash tables one by one
- Time complexity: $2^{0.34n+o(n)}$
  - Compute one hash, and go through $2^{o(n)}$ vectors
  - Repeat this for each of $2^{0.21n}$ vectors
  - Repeat this for each of $2^{0.13n}$ hash tables

HashSieve (NV)

Overview

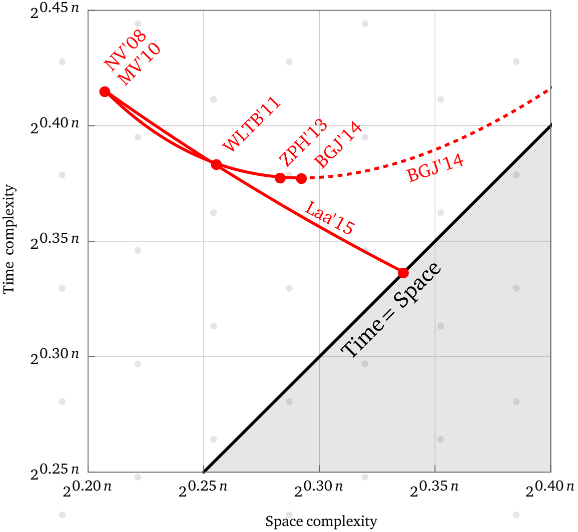
- Same k (hyperplanes) and t (hash tables) as before
- Space complexity: $2^{0.21n+o(n)}$
  - Before: store $2^{0.13n}$ hash tables containing all $2^{0.21n}$ vectors
  - Now: process $2^{0.13n}$ hash tables one by one
- Time complexity: $2^{0.34n+o(n)}$
  - Compute one hash, and go through $2^{o(n)}$ vectors
  - Repeat this for each of $2^{0.21n}$ vectors
  - Repeat this for each of $2^{0.13n}$ hash tables

Heuristic

The HashSieve (NV) runs in time $2^{0.34n+o(n)}$ and space $2^{0.21n+o(n)}$.

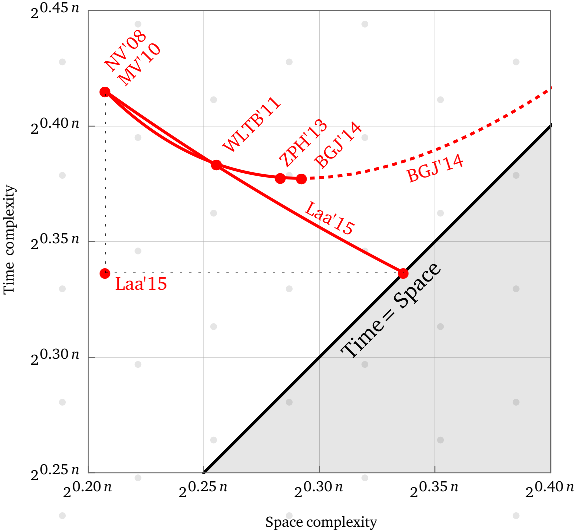
HashSieve (NV)

Space/time trade-off



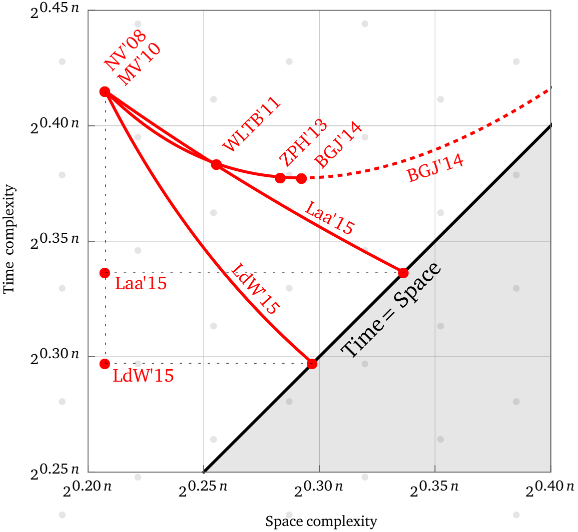
HashSieve (NV)

Space/time trade-off



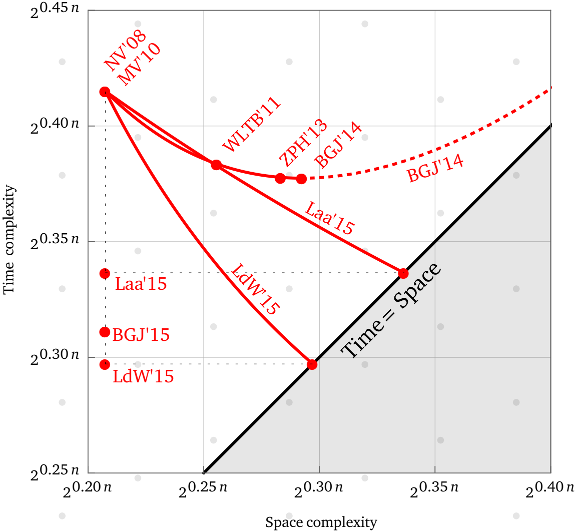
SphereSieve

Space/time trade-off



Another NNS sieve

Space/time trade-off



CrossPolytopeSieve

Main ideas

- HashSieve: divide space into "equal" parts with hyperplanes

CrossPolytopeSieve

Main ideas

- HashSieve: divide space into “equal” parts with hyperplanes
- Practice: orthogonal hyperplanes better than random ones
    - Orthogonal hyperplanes cause more “equal” divisions
    - Corresponds to “hypercube hashing”
    - High-level idea: more symmetric partitioning is better

CrossPolytopeSieve

Main ideas

- HashSieve: divide space into "equal" parts with hyperplanes
- Practice: orthogonal hyperplanes better than random ones
  - Orthogonal hyperplanes cause more "equal" divisions
  - Corresponds to "hypercube hashing"
  - High-level idea: more symmetric partitioning is better
- Observation: hypercube not "most symmetric" object
  - $k \approx n/5$, e.g. $k = 2$ hyperplanes in dimension $n = 10$
  - For $k = 2$ we have 4 regions and we use vertices of a square
  - Use more symmetric object: tetrahedron works better!

TU/e

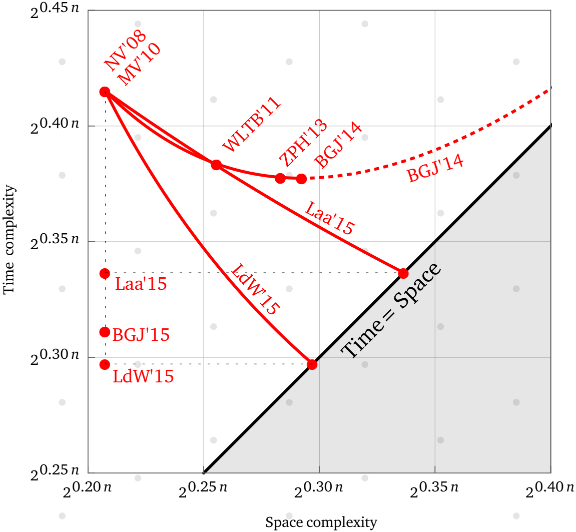
CrossPolytopeSieve

Main ideas

- HashSieve: divide space into "equal" parts with hyperplanes
- Practice: orthogonal hyperplanes better than random ones
  - Orthogonal hyperplanes cause more "equal" divisions
  - Corresponds to "hypercube hashing"
  - High-level idea: more symmetric partitioning is better
- Observation: hypercube not "most symmetric" object
  - $k \approx n/5$, e.g. $k = 2$ hyperplanes in dimension $n = 10$
  - For $k = 2$ we have 4 regions and we use vertices of a square
  - Use more symmetric object: tetrahedron works better!
- Idea: divide space into regions using regular polytopes
  - Hypercube $\implies$ HashSieve with orthogonal hyperplanes
  - Cross polytope $\implies$ CrossPolytopeSieve
  - Simplex $\implies$ similar to CrossPolytopeSieve

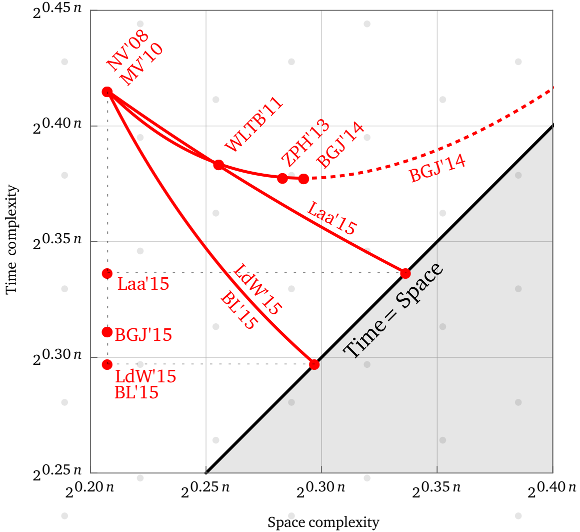
CrossPolytopeSieve

Space/time trade-off



CrossPolytopeSieve

Space/time trade-off



Open problems

- Heuristic sieving
  - Other LSH methods? (other symmetric objects?)
  - SVP challenge records?
  - Crossover point with enumeration?
  - Ideal lattice sieving speedups?
- Provable sieving
  - Better exponent with LSH?
  - Use idea of partitioning the space?
- Other SVP algorithms
  - Voronoi cell with LSH?
  - Solve CVPP faster with LSH?
  - Combine sieving with enumeration?

Questions