Technische Universiteit Eindhoven University of Technology

Sieving for shortest lattice vectors using locality-sensitive hashing

Thijs Laarhoven

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

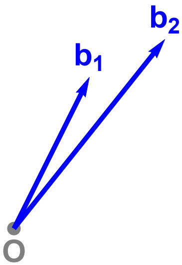
EPFL, Lausanne, Switzerland
(April 14, 2015)

Lattices

What is a lattice?

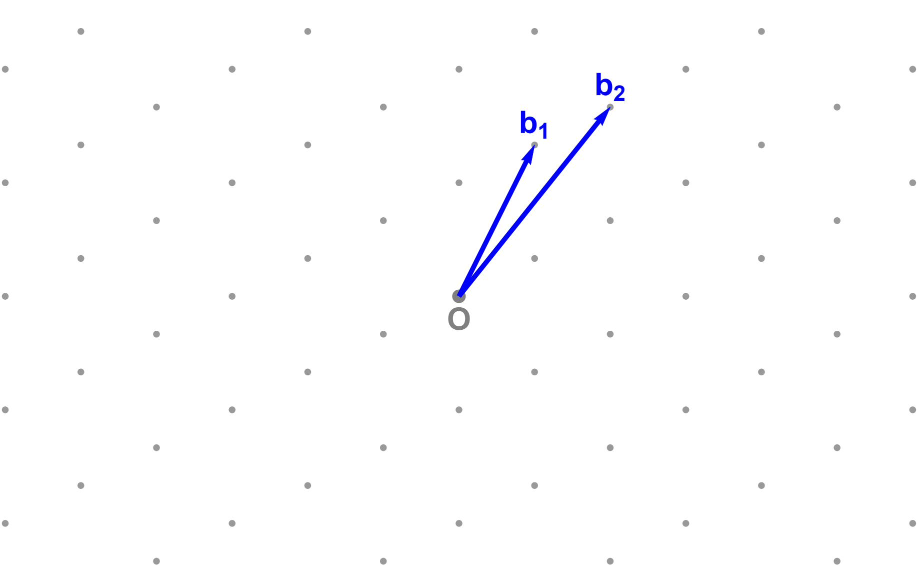
Lattices

What is a lattice?



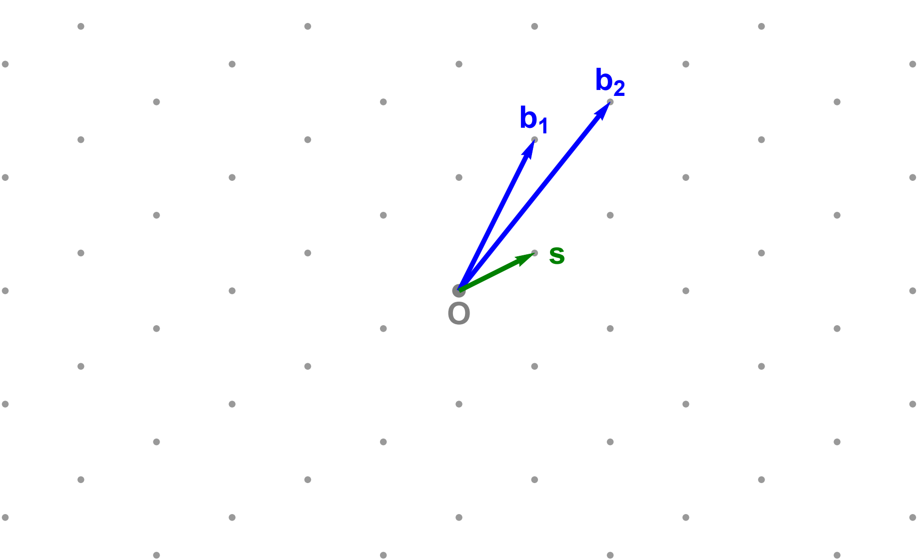
Lattices

What is a lattice?



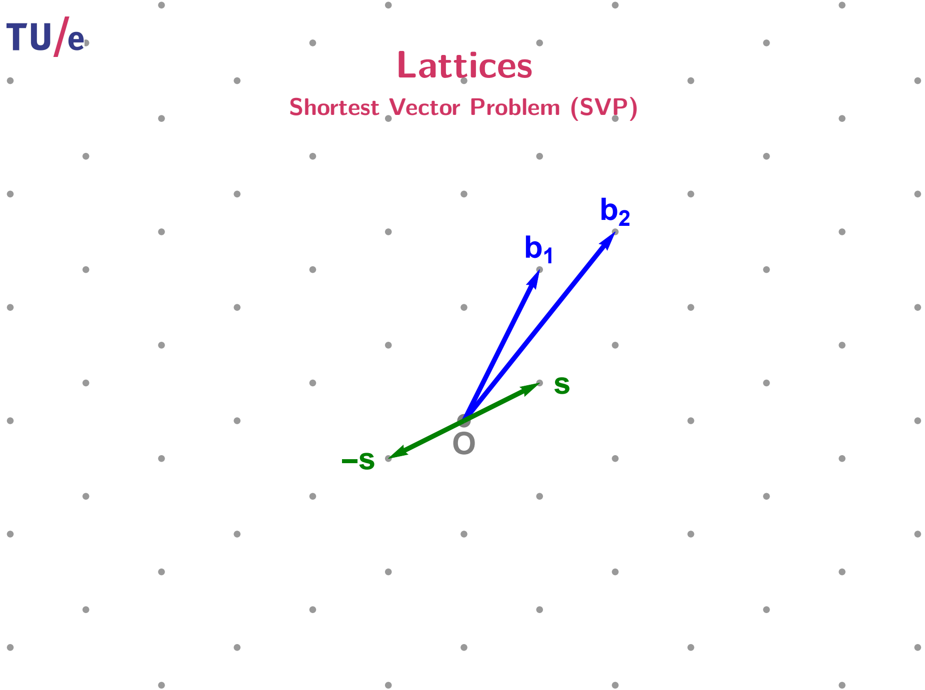
Lattices

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, ...]	$\Omega(n \log n)$	$O(\log n)$
	AKS-sieve [AKS01, NV08, ...]	$3.398n$	$1.985n$
	ListSieve [MV10]	$3.199n$	$1.327n$
	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]	$1.000n$	$0.500n$
Heuristic SVP	NV-sieve [NV08]	$0.415n$	$0.208n$
	GaussSieve [MV10]	$0.415n$?	$0.208n$
	2-level sieve [WLTB11]	$0.384n$	$0.256n$
	3-level sieve [ZPH14]	$0.378n$	$0.283n$
	Decomposition approach [BGJ14]	$0.378n$	$0.293n$
	HashSieve [Laa14, MLB15]	$0.337n$	$0.337n$
	SphereSieve [LdW15]	$0.298n$	$0.298n$

Nguyen-Vidick sieve

Nguyen-Vidick sieve

1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve

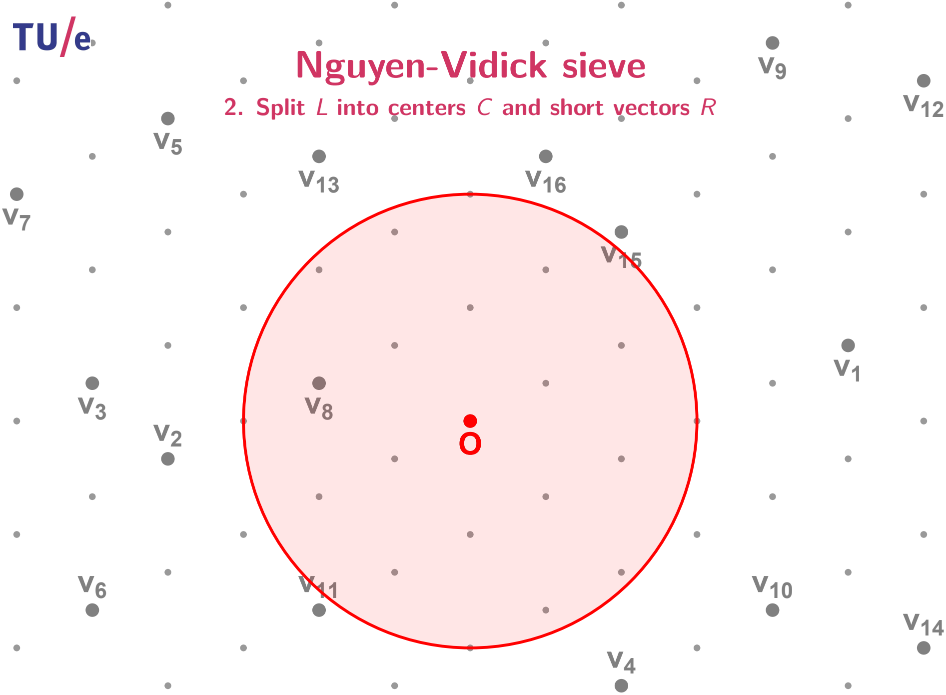
1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve

2. Split L into centers C and short vectors R

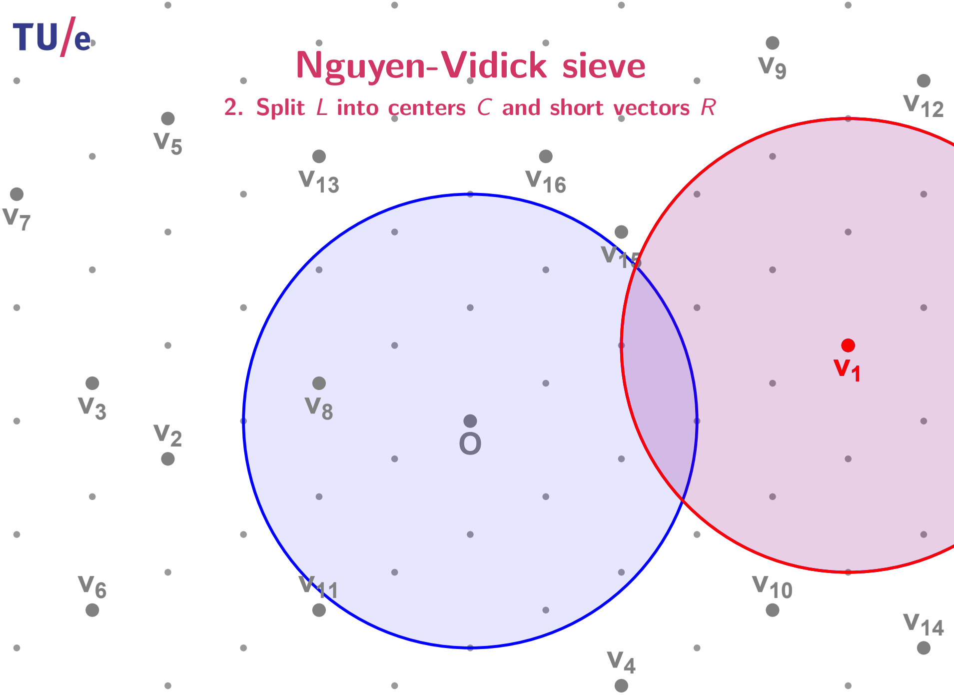
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



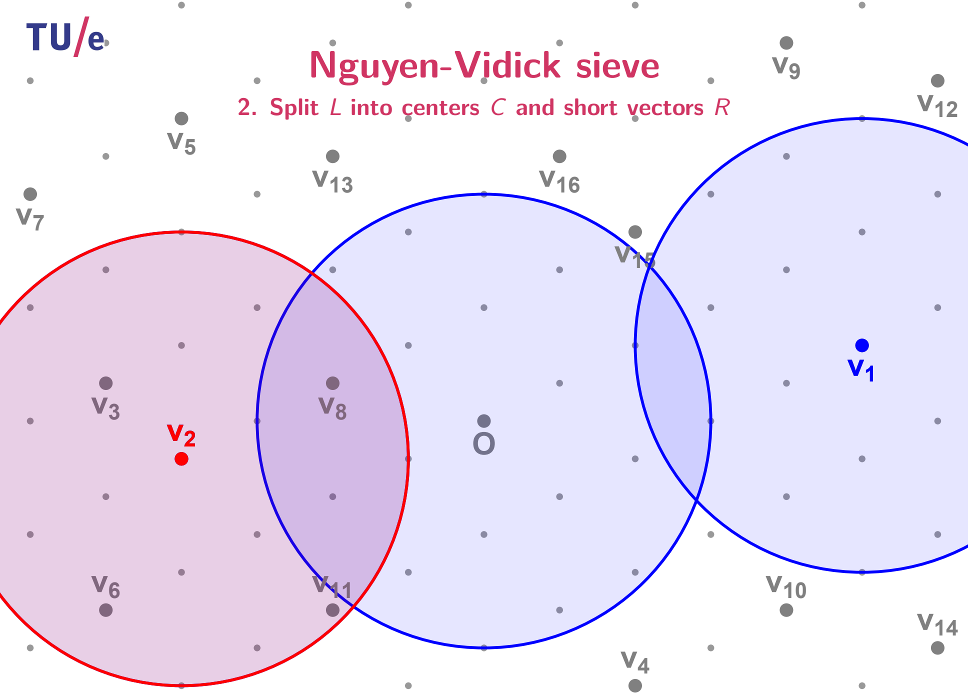
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



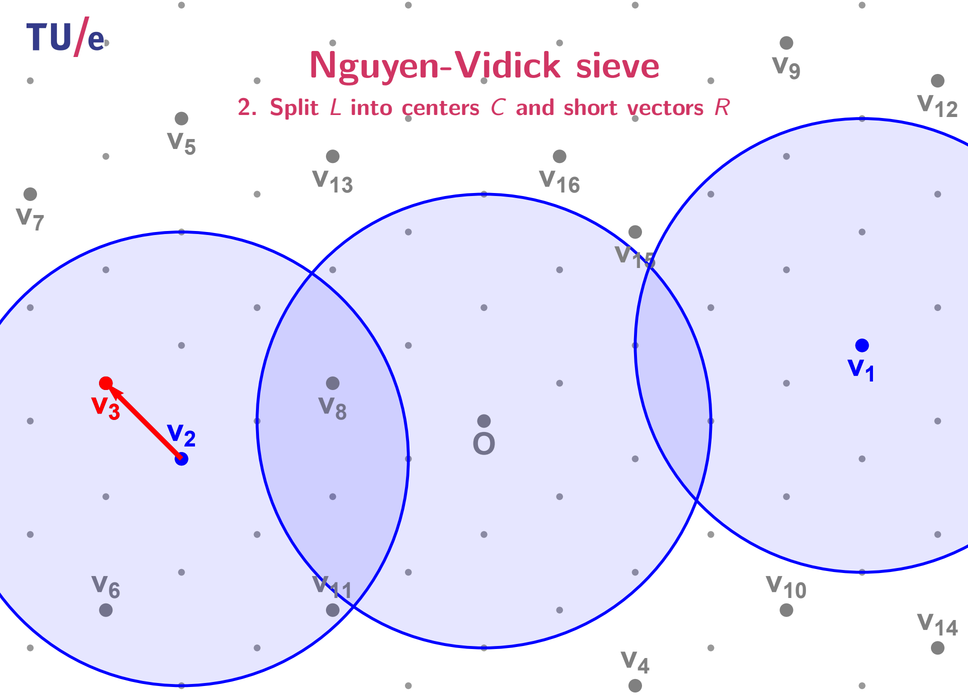
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



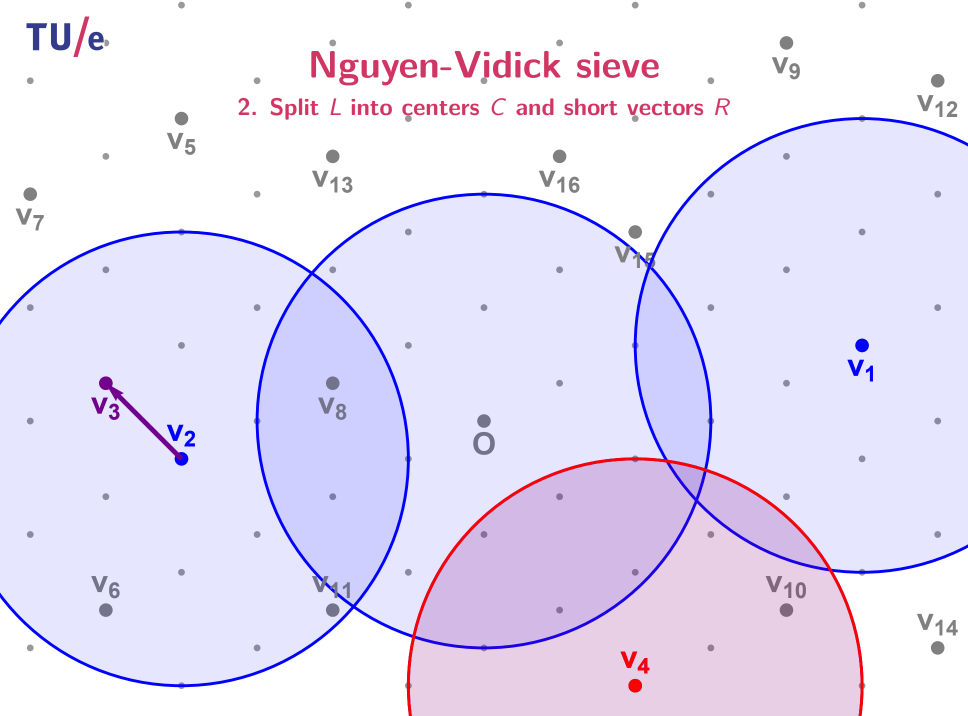
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



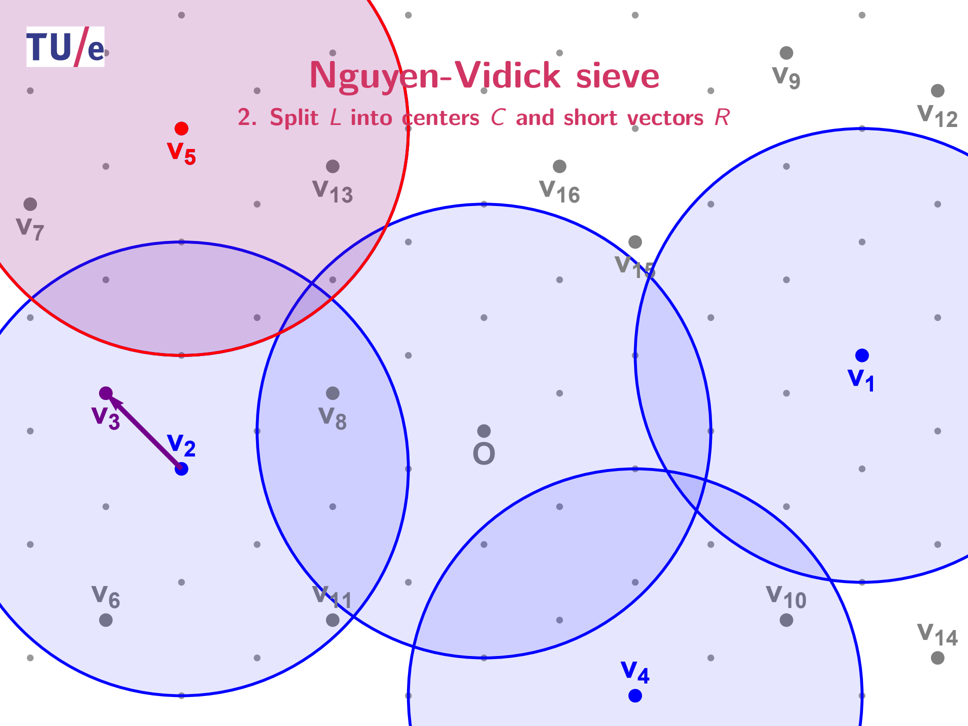
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



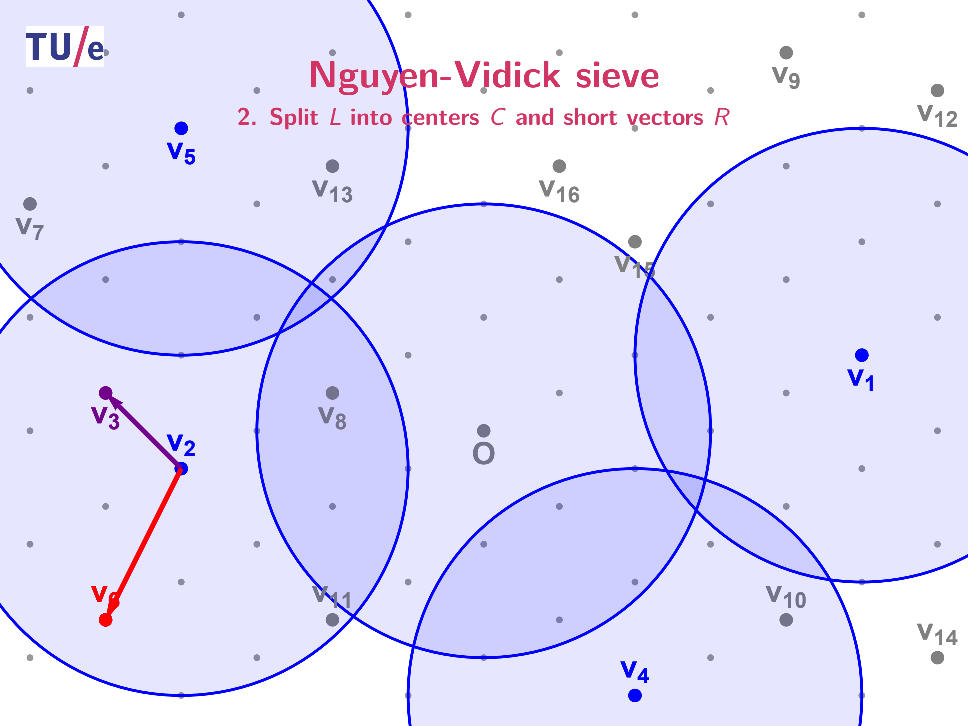
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



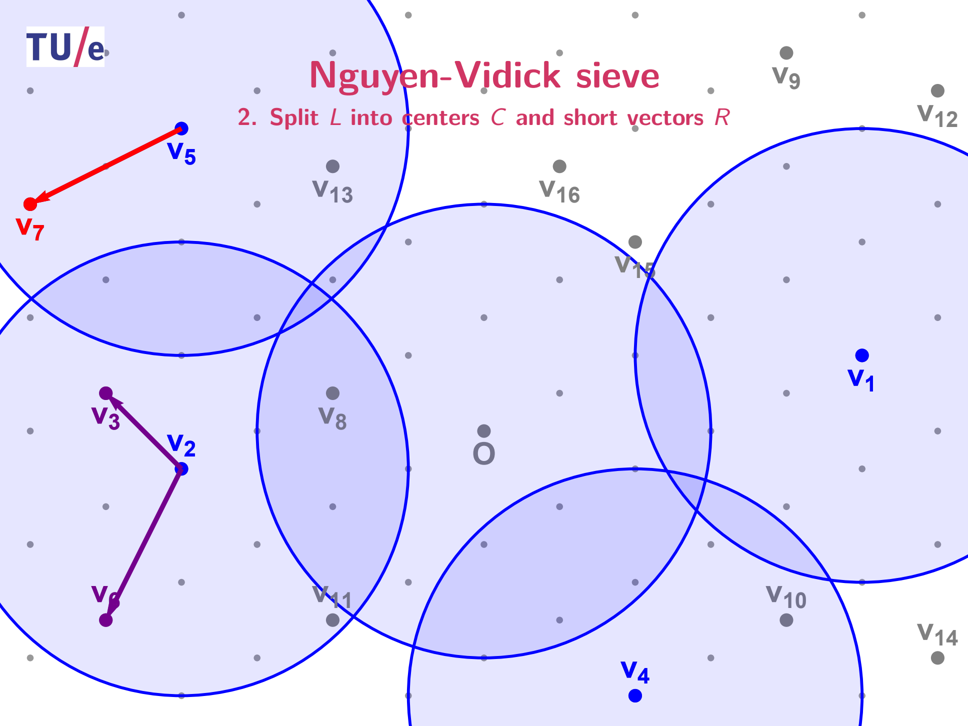
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



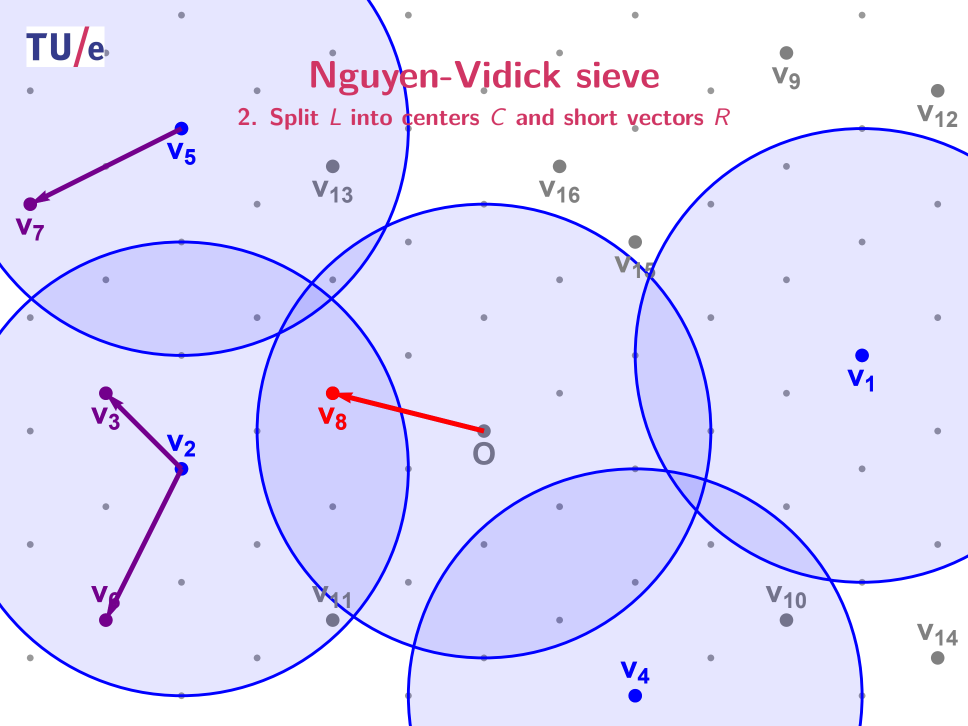
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



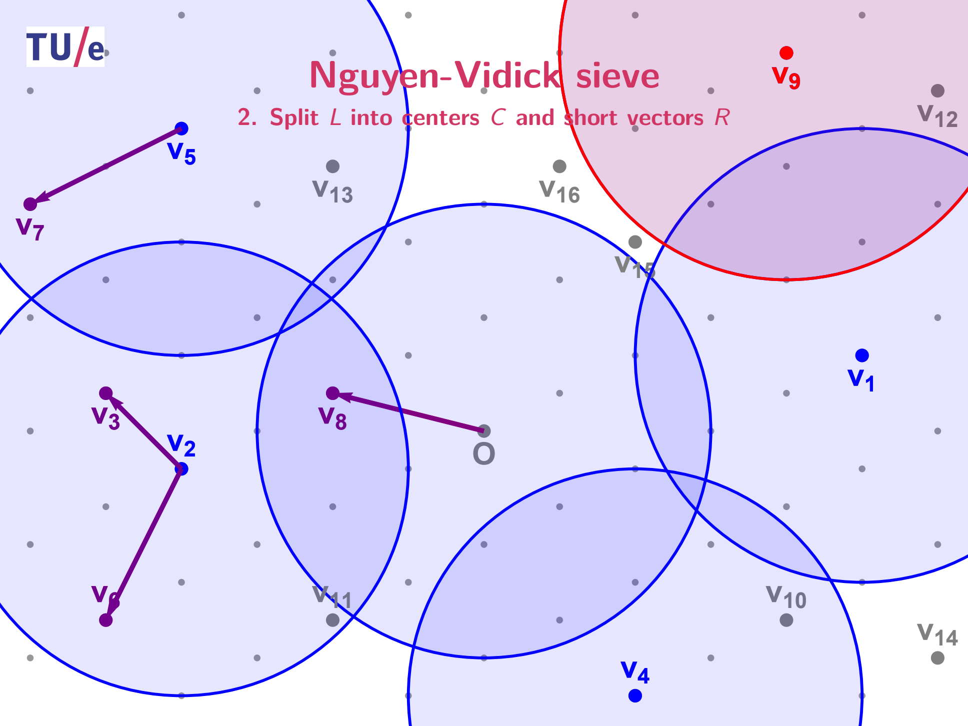
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



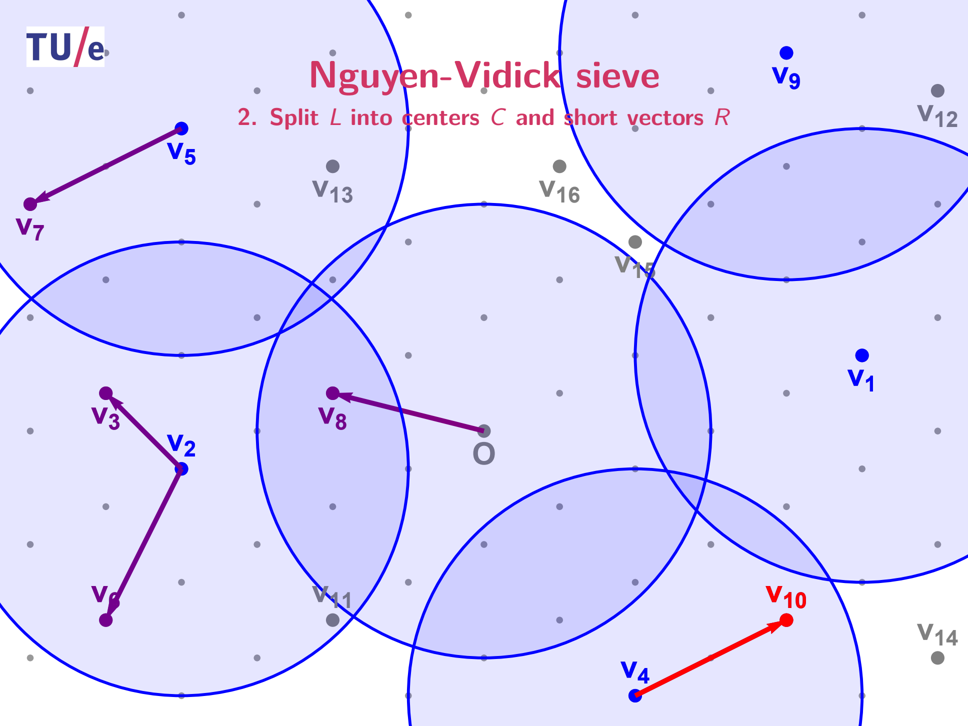
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



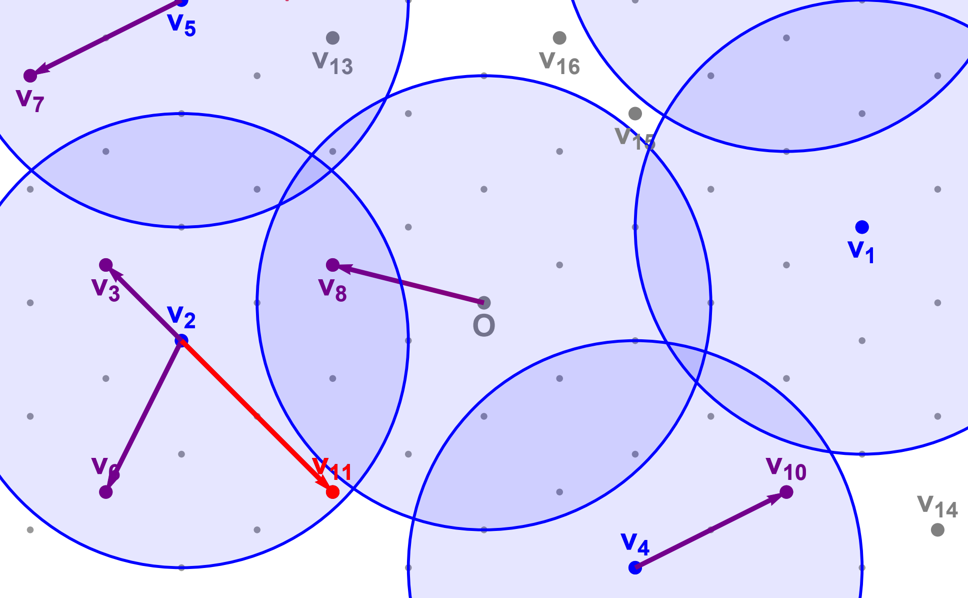
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



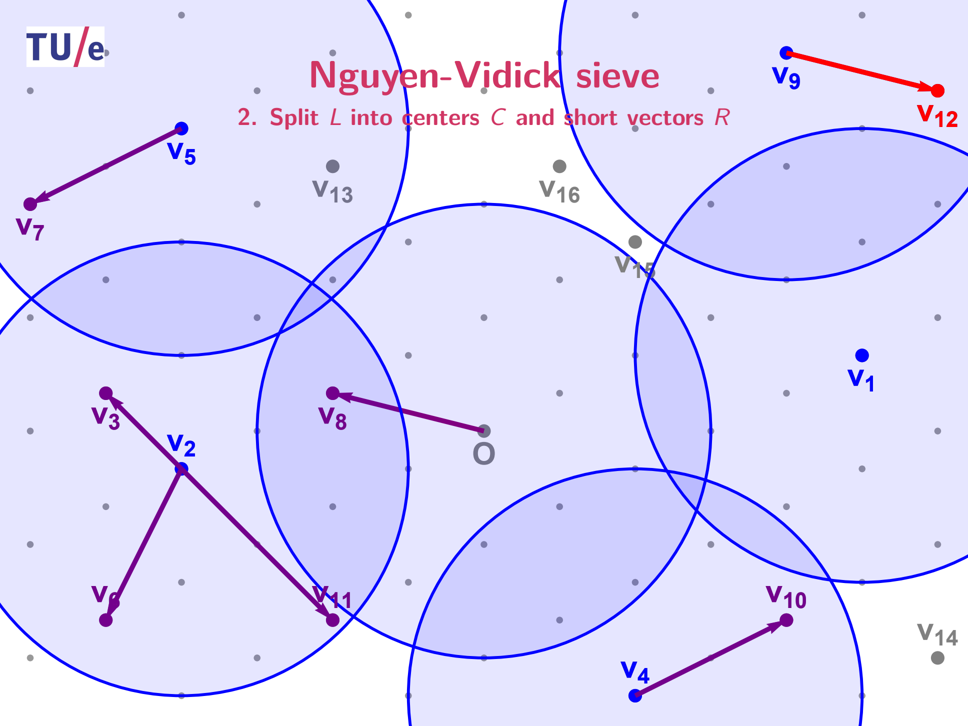
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



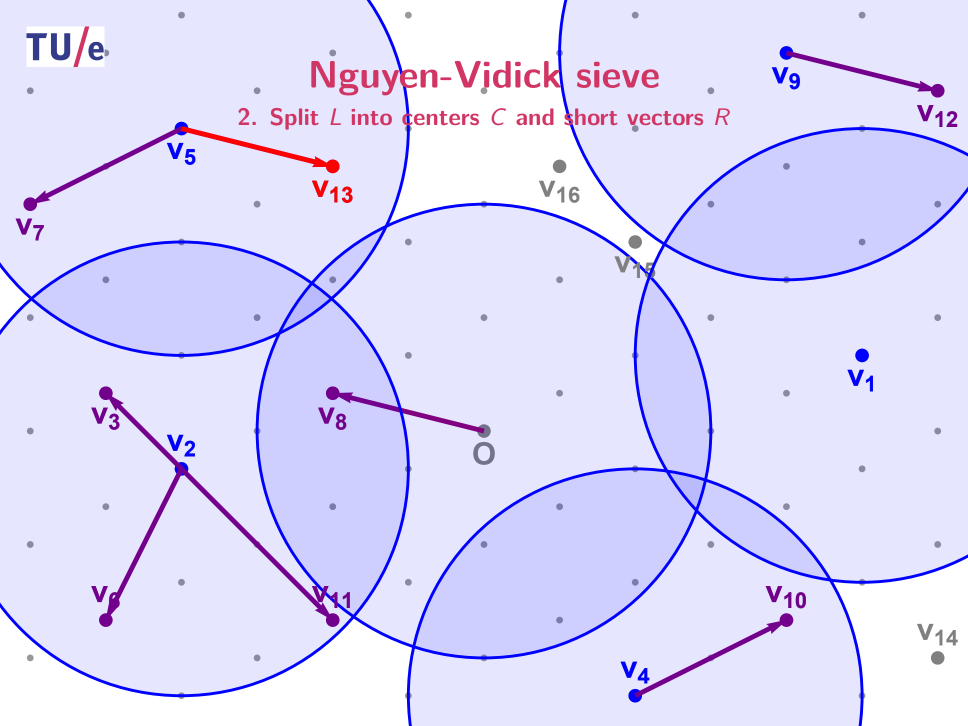
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



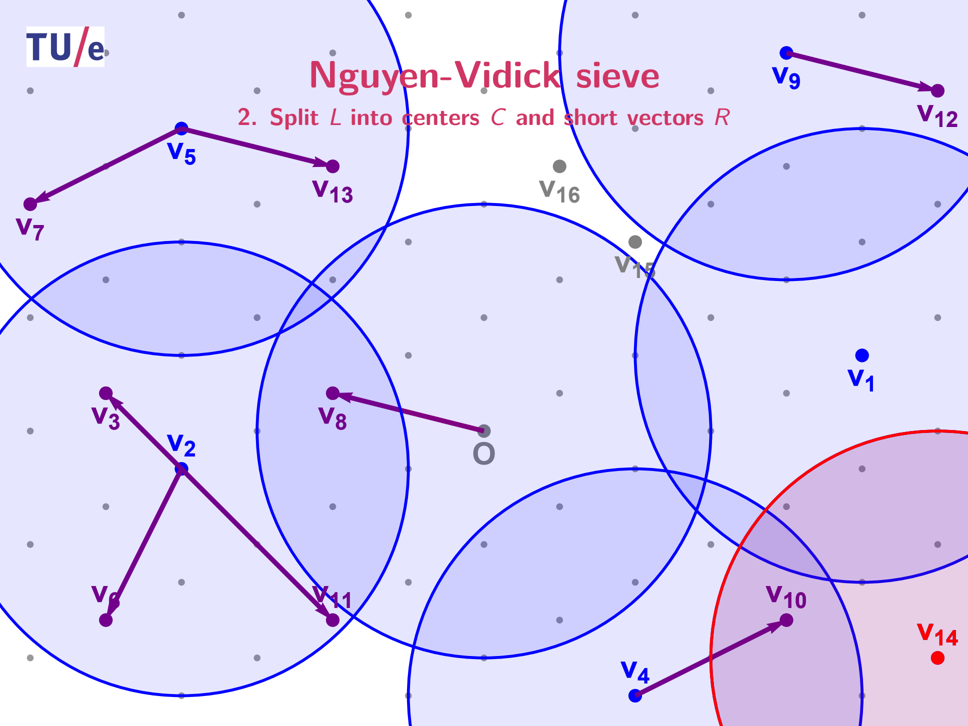
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



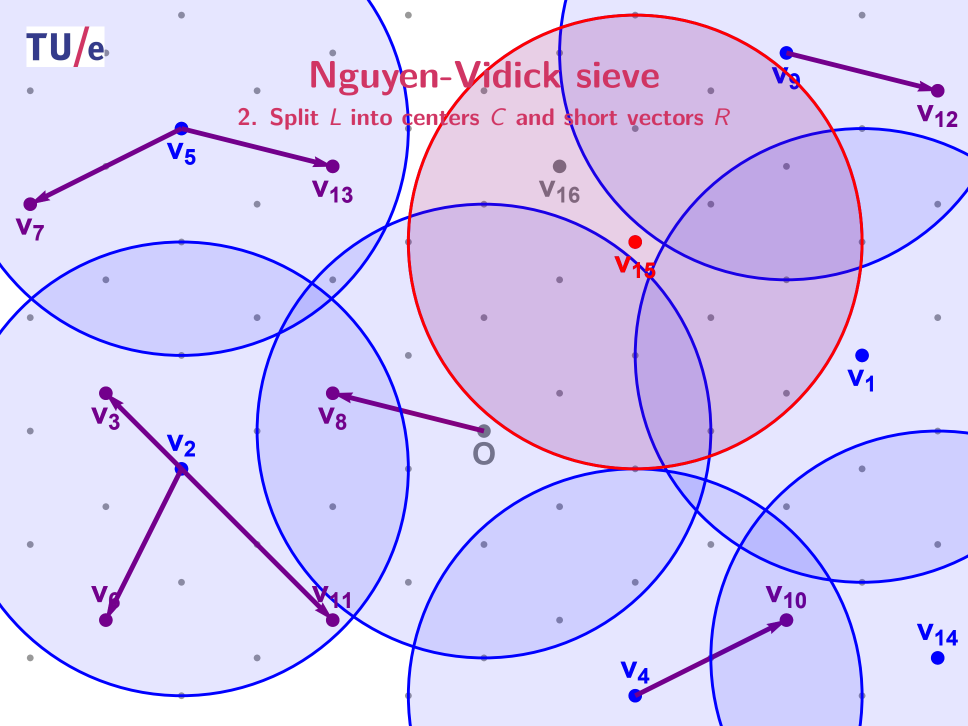
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



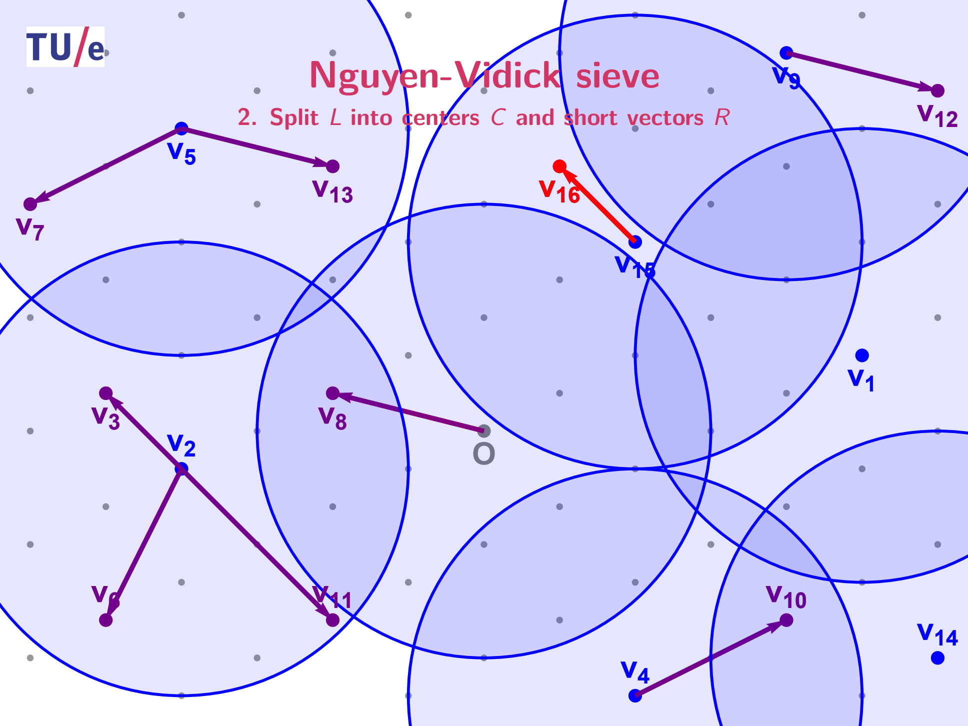
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



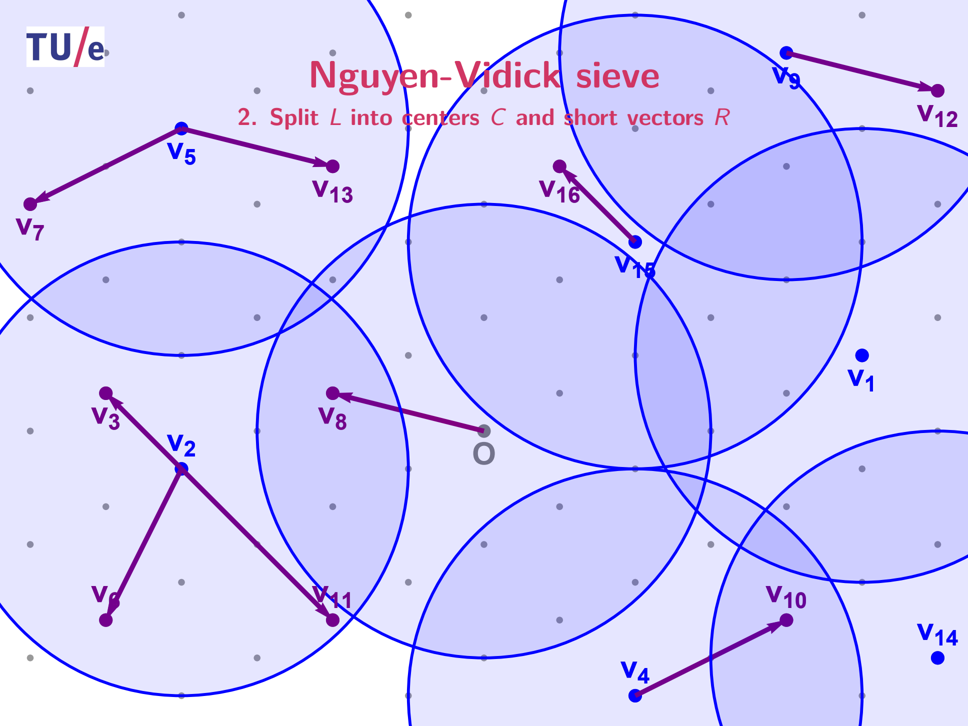
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



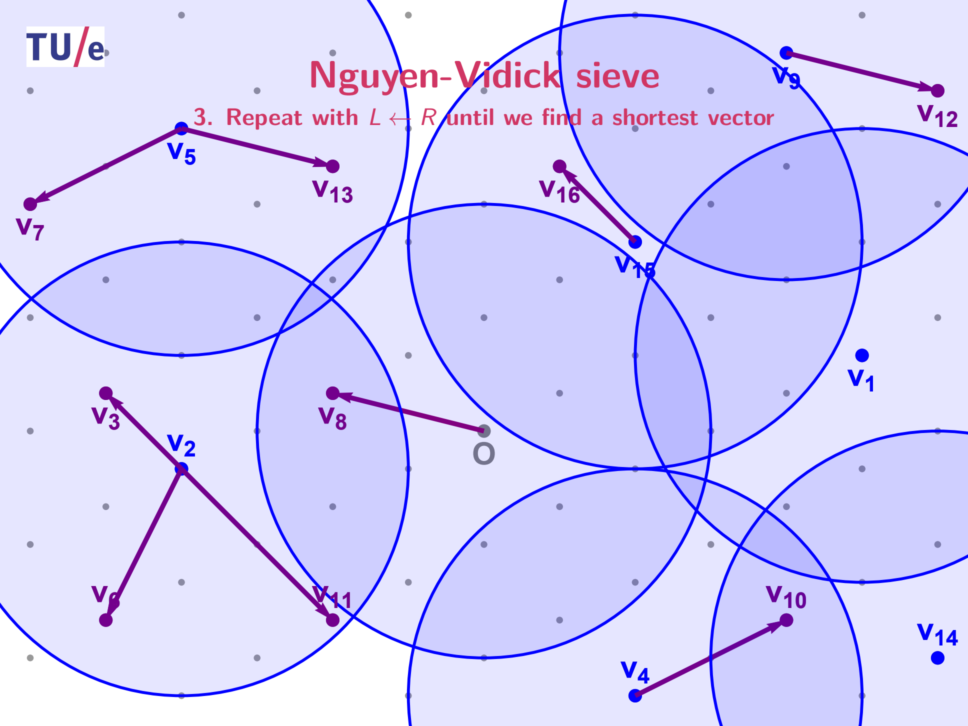
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



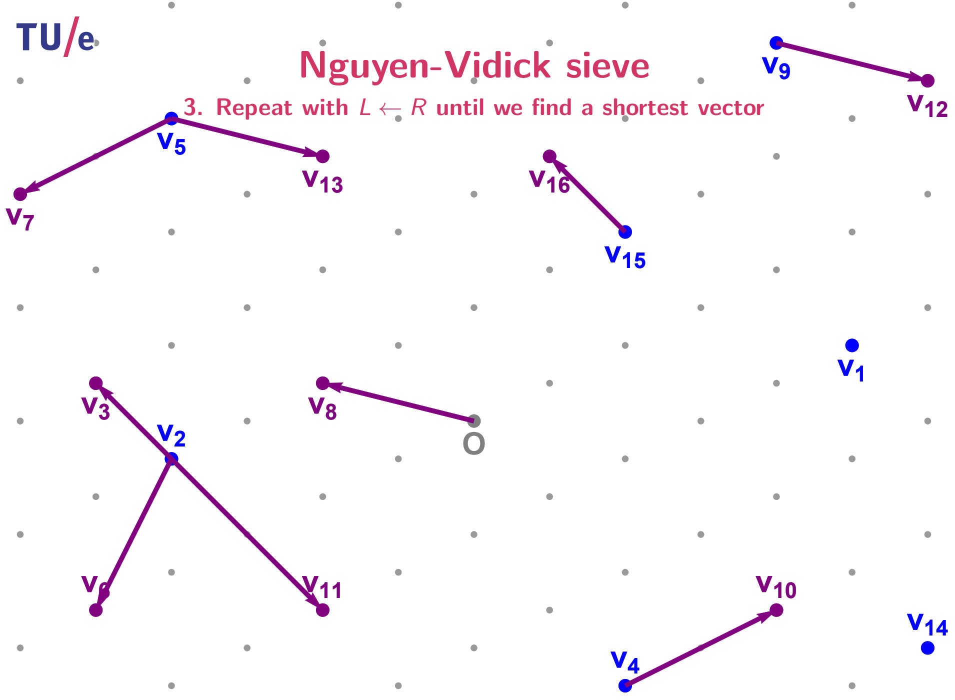
Nguyen-Vidick sieve

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



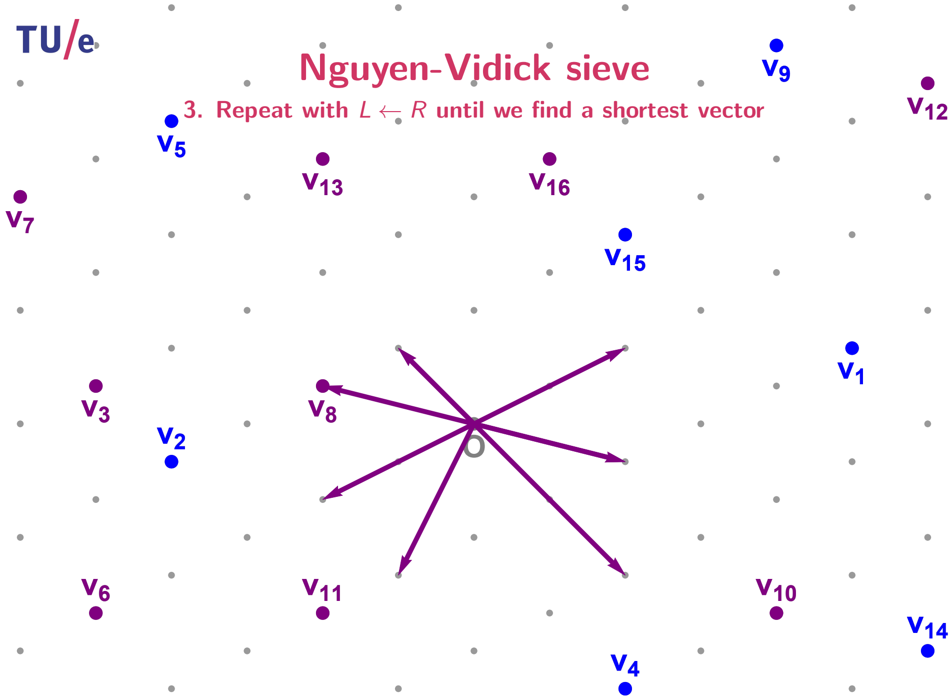
Nguyen-Vidick sieve

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



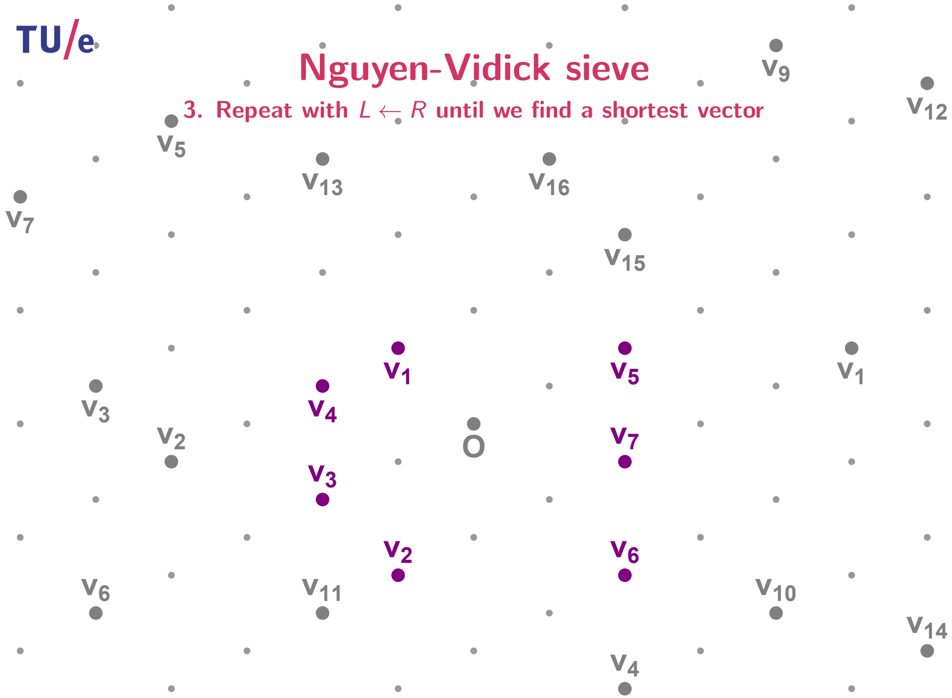
Nguyen-Vidick sieve

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
  - Each center covers $(\sin \frac{\pi}{3})^{-n} = \sqrt{3/4}^n$ of the space
  - Need $\sqrt{4/3}^n$ vectors to cover all directions

Nguyen-Vidick sieve

Overview

- Space complexity: $\sqrt{4/3}^n \approx 2^{0.21n+o(n)}$ vectors
  - Each center covers $(\sin \frac{\pi}{3})^{-n} = \sqrt{3/4}^n$ of the space
  - Need $\sqrt{4/3}^n$ vectors to cover all directions
- 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

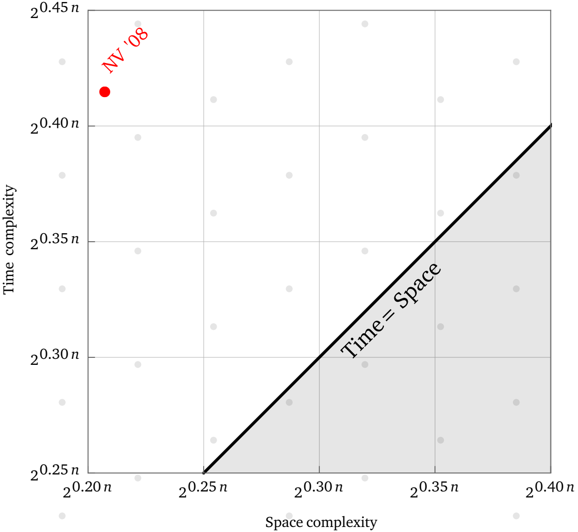
- Space complexity: $\sqrt{4/3}^n \approx 2^{0.21n+o(n)}$ vectors
  - Each center covers $(\sin \frac{\pi}{3})^{-n} = \sqrt{3/4}^n$ of the space
  - Need $\sqrt{4/3}^n$ vectors to cover all directions
- Time complexity: $(4/3)^n \approx 2^{0.42n+o(n)}$
  - Comparing a target vector to all centers: $2^{0.21n+o(n)}$
  - Repeating this for each list vector: $2^{0.21n+o(n)}$
  - Repeating the whole sieving procedure: $\mathrm{poly}(n)$

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

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

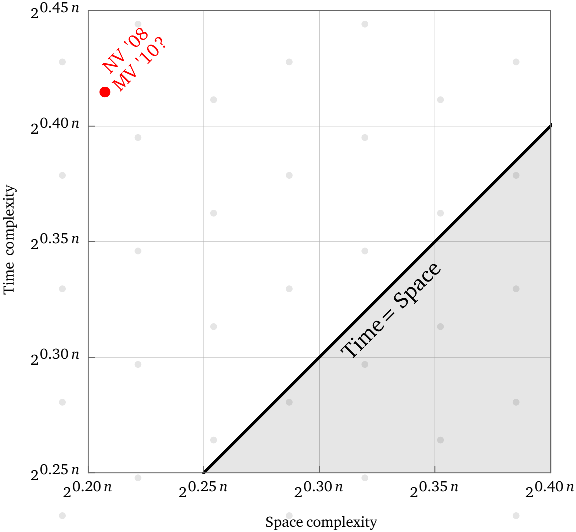
Nguyen-Vidick sieve

Space/time trade-off



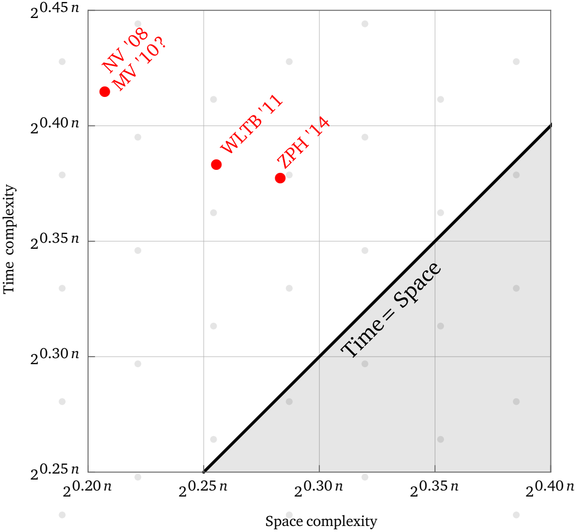
GaussSieve

Space/time trade-off



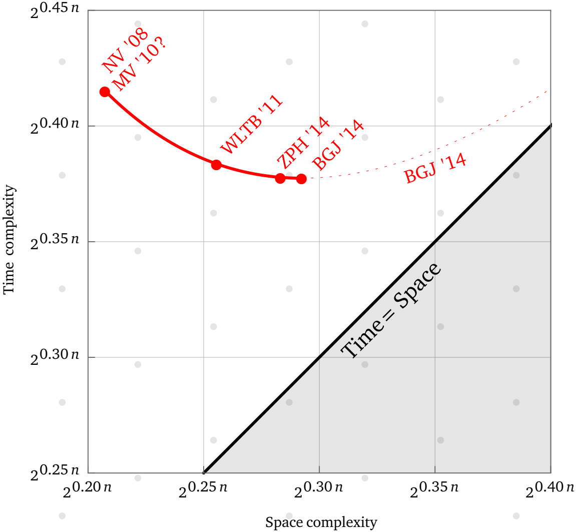
Leveled sieving

Space/time trade-off



Decomposition approach

Space/time trade-off



HashSieve

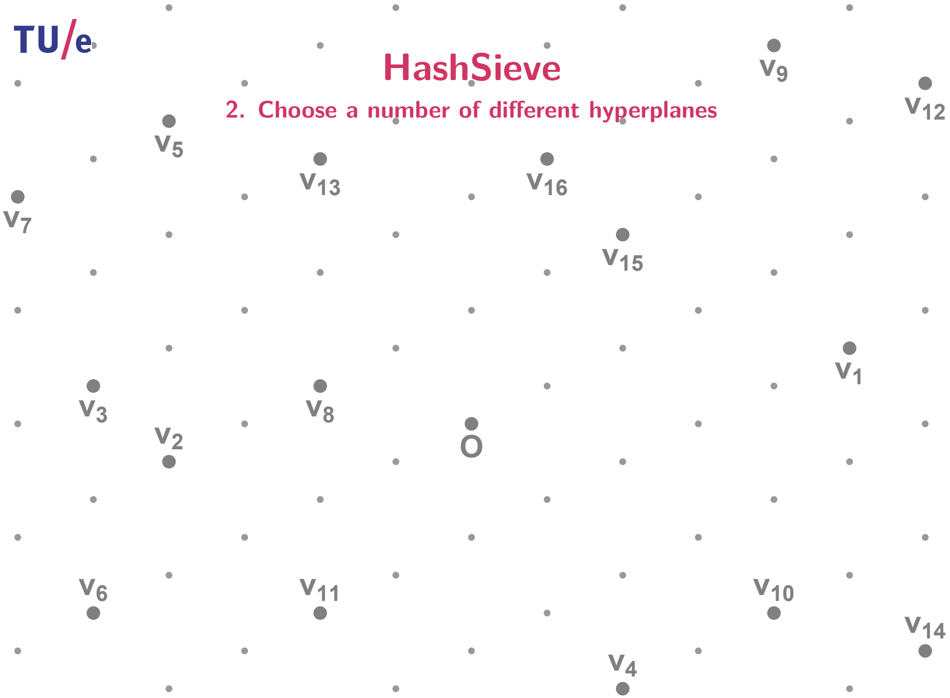
1. Sample a list L of random lattice vectors

HashSieve

1. Sample a list L of random lattice vectors

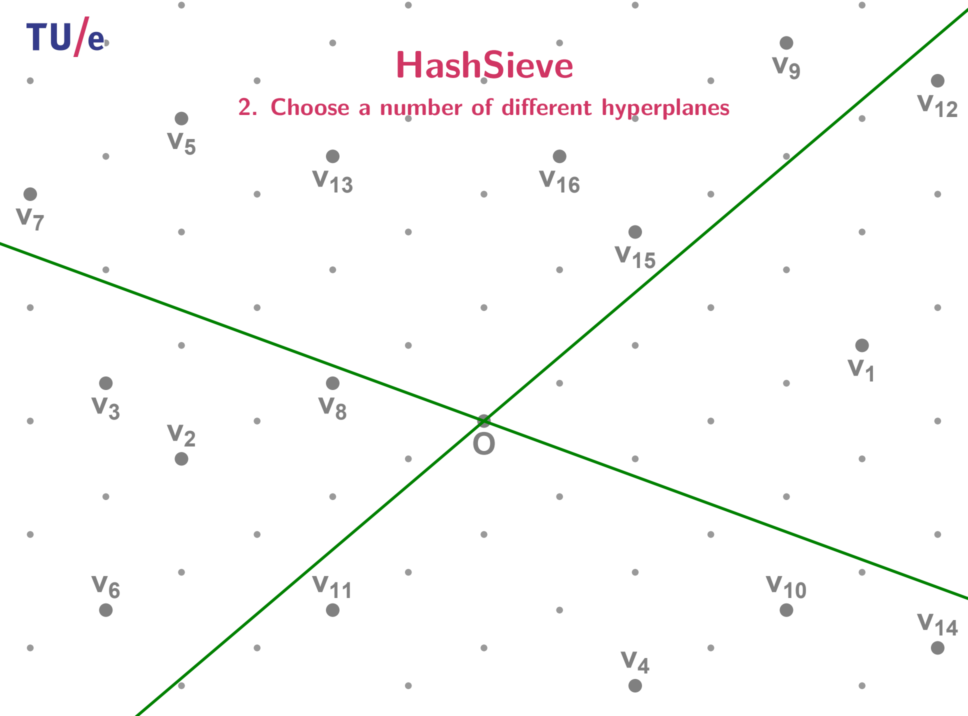
HashSieve

2. Choose a number of different hyperplanes



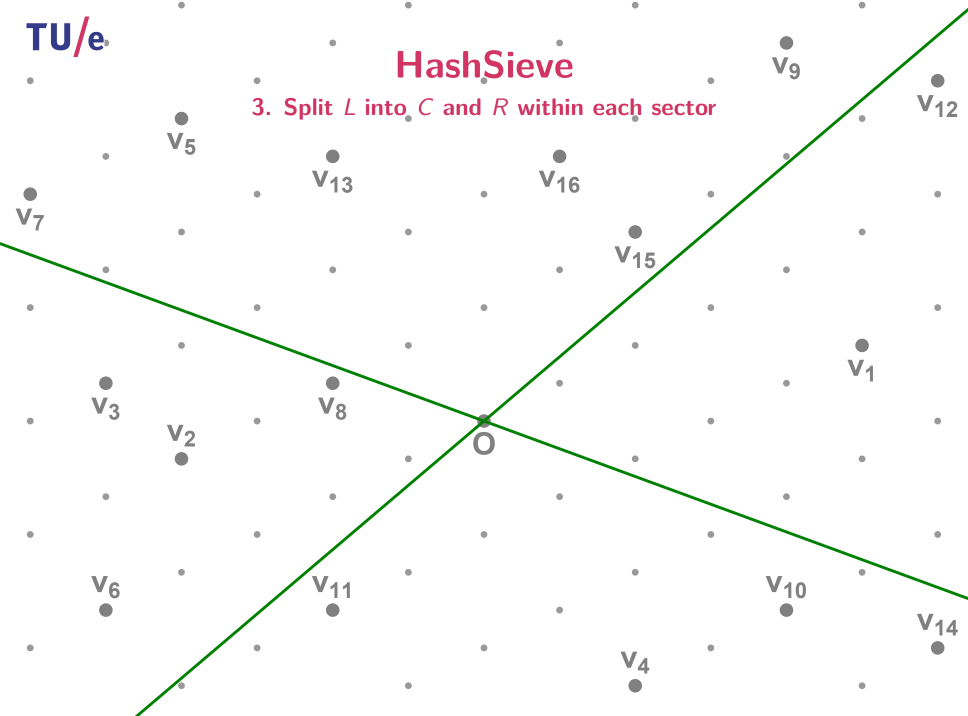
HashSieve

2. Choose a number of different hyperplanes



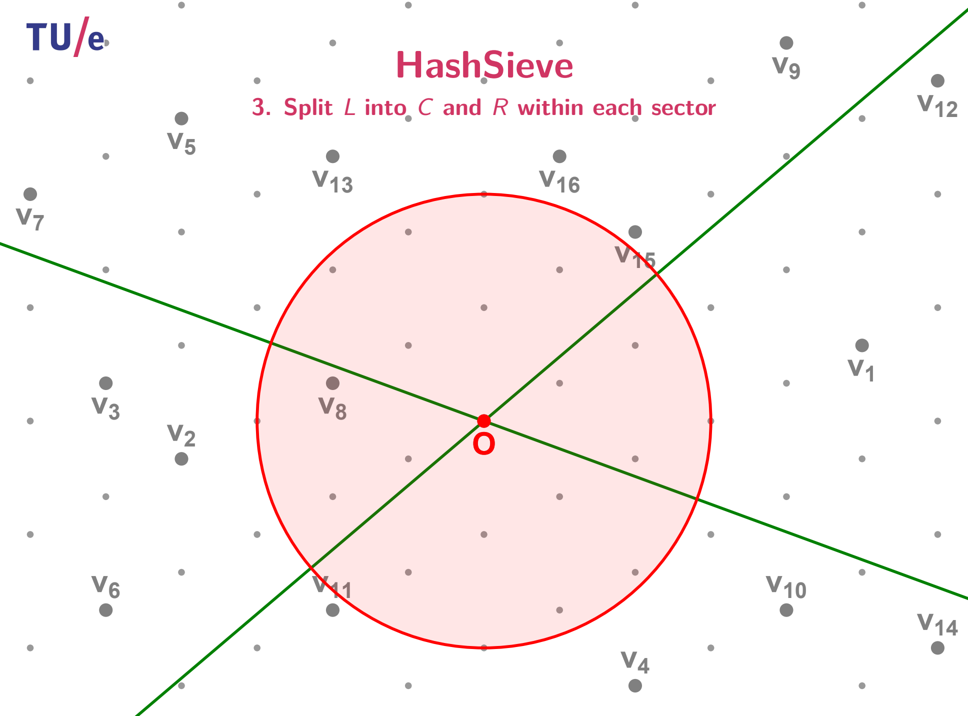
HashSieve

3. Split L into C and R within each sector



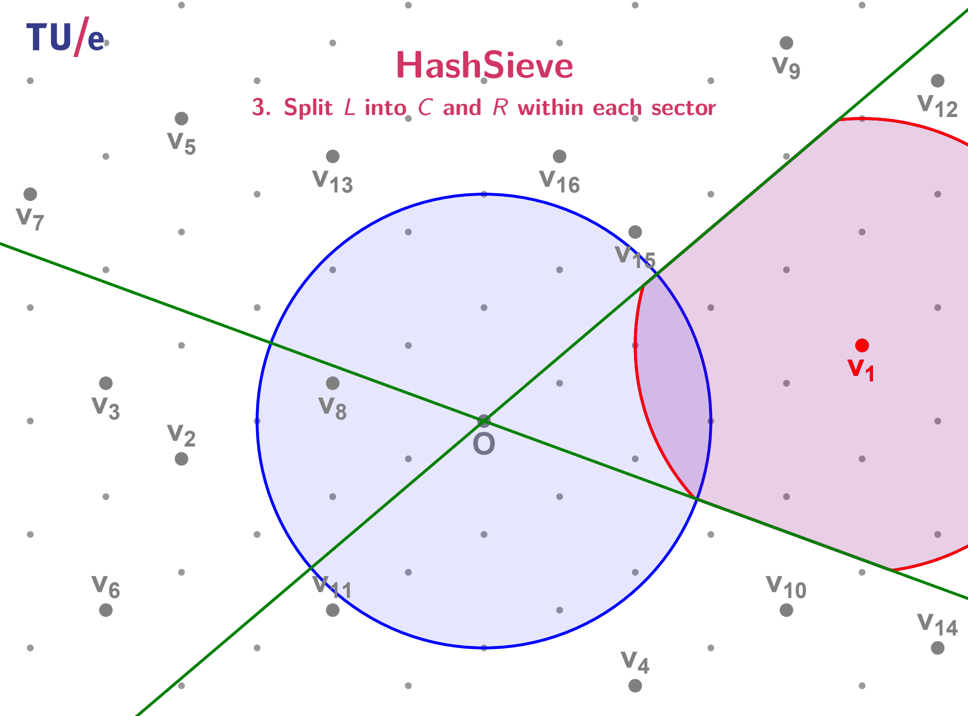
HashSieve

3. Split L into C and R within each sector



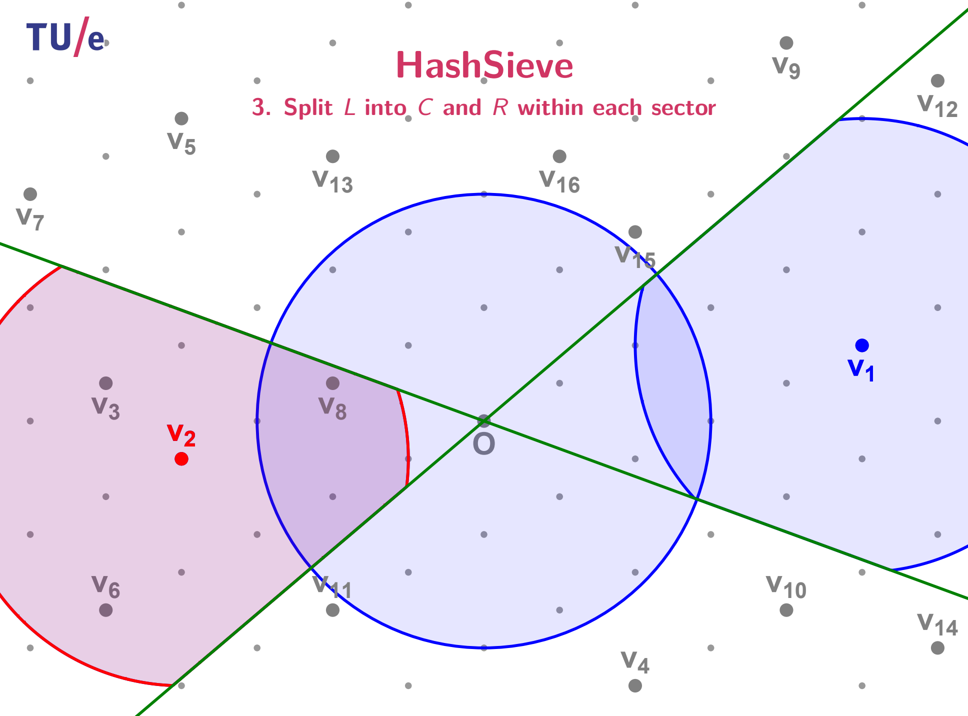
HashSieve

3. Split L into C and R within each sector



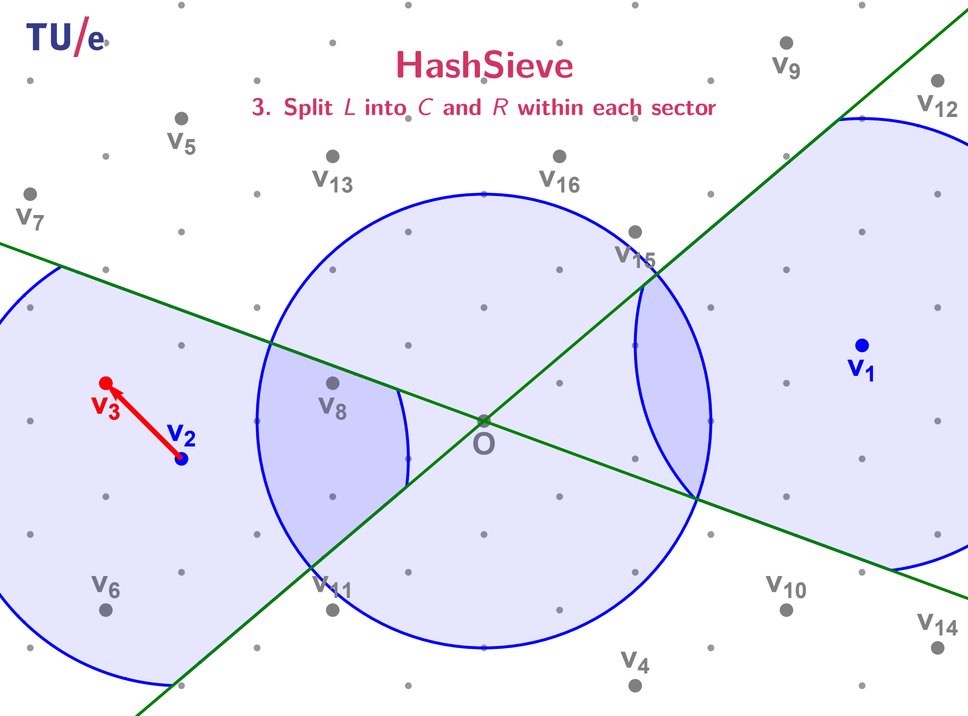
HashSieve

3. Split L into C and R within each sector



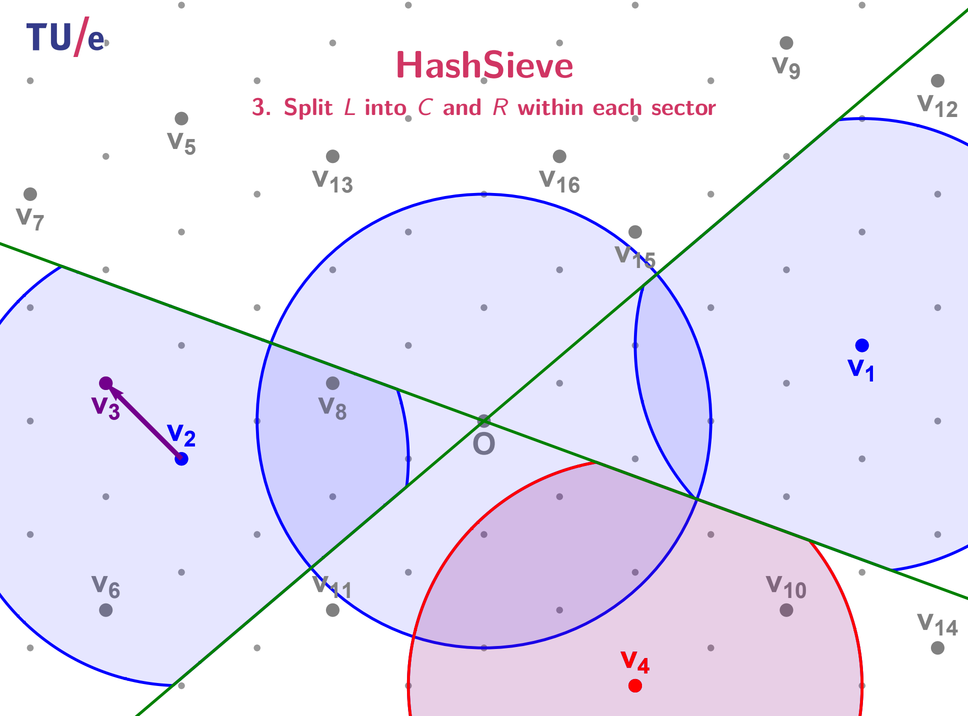
HashSieve

3. Split L into C and R within each sector



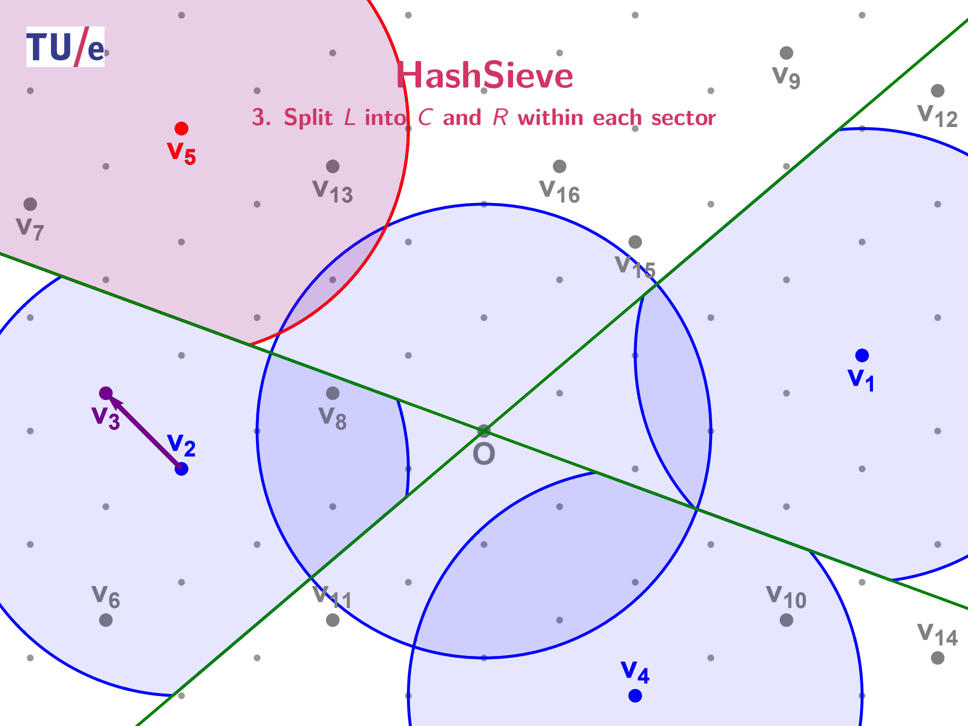
HashSieve

3. Split L into C and R within each sector



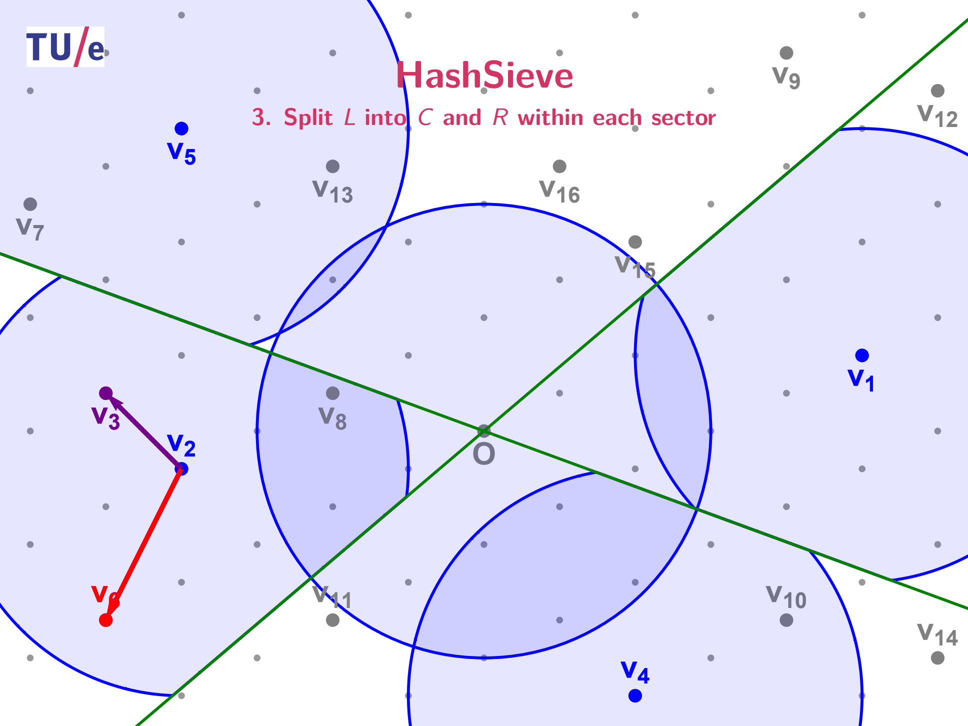
HashSieve

3. Split L into C and R within each sector



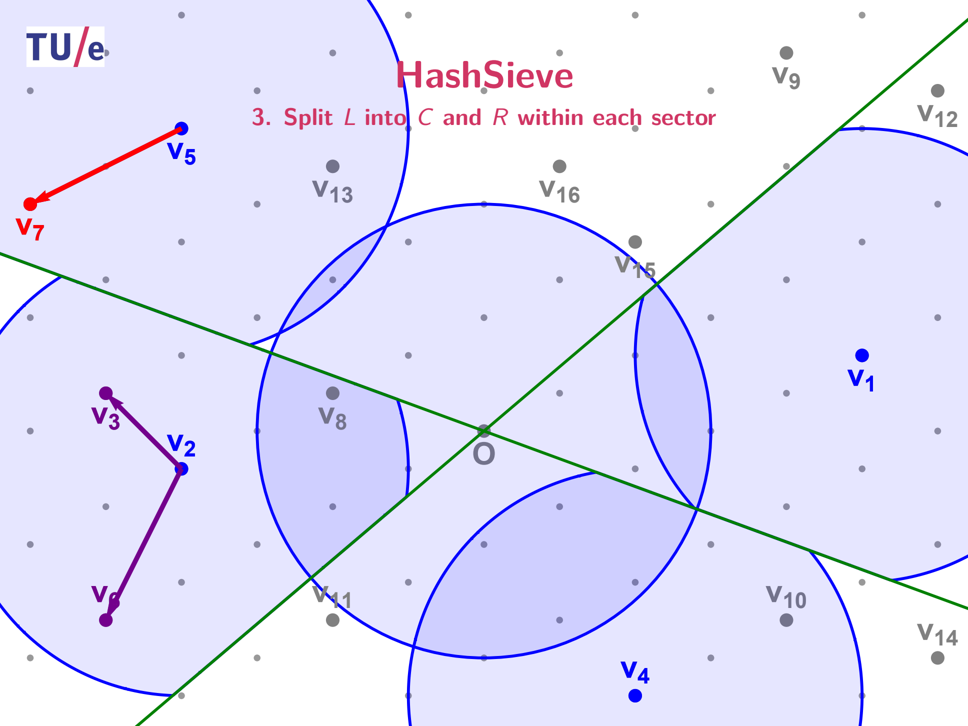
HashSieve

3. Split L into C and R within each sector



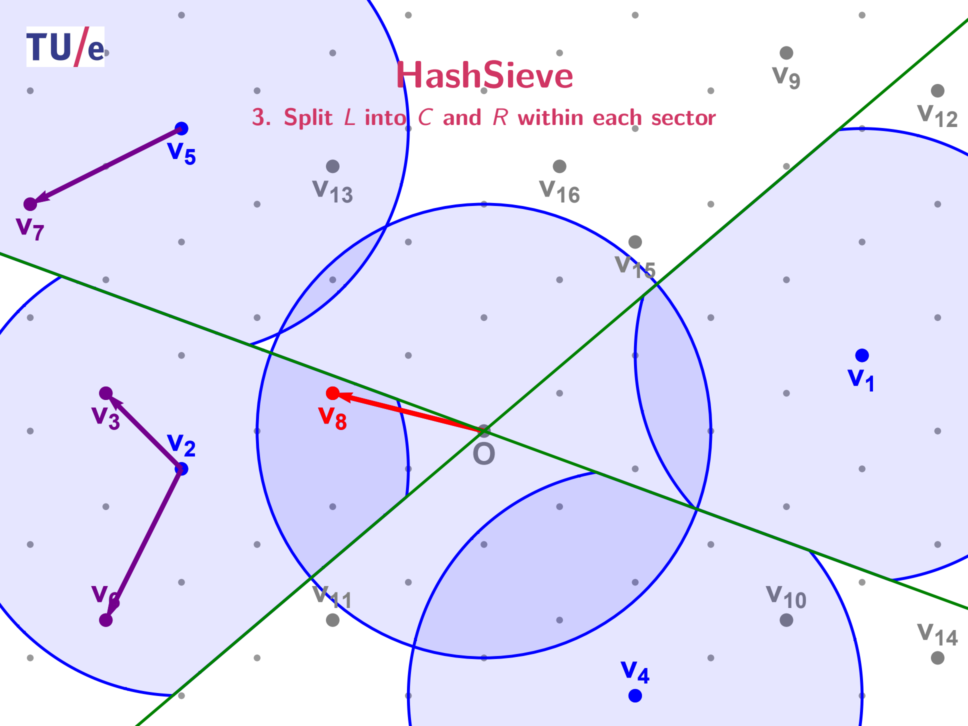
HashSieve

3. Split L into C and R within each sector



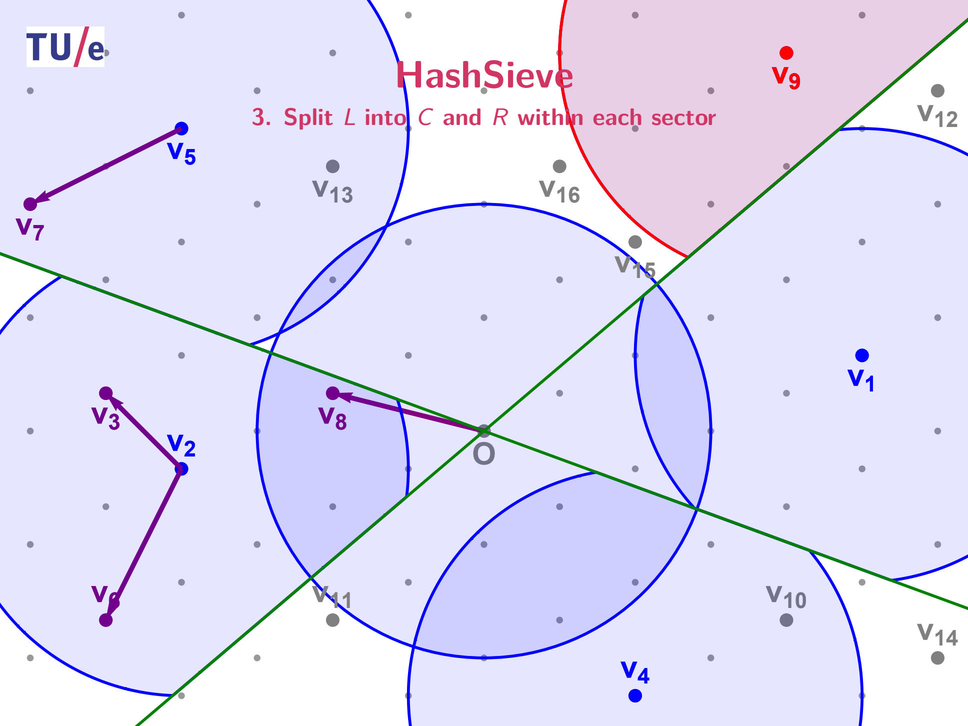
HashSieve

3. Split L into C and R within each sector



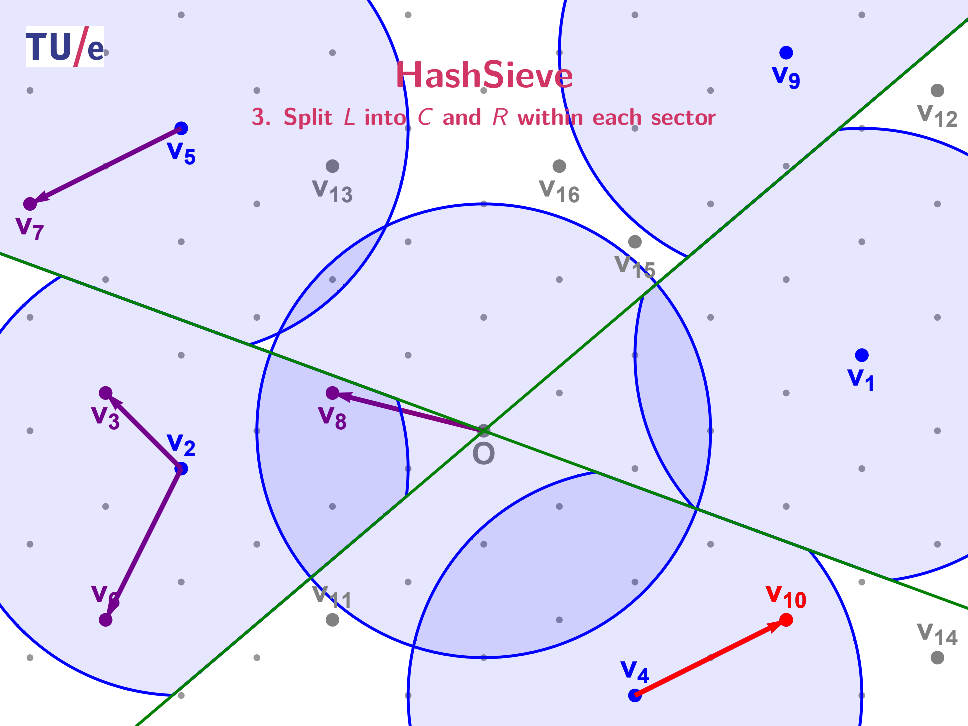
HashSieve

3. Split L into C and R within each sector



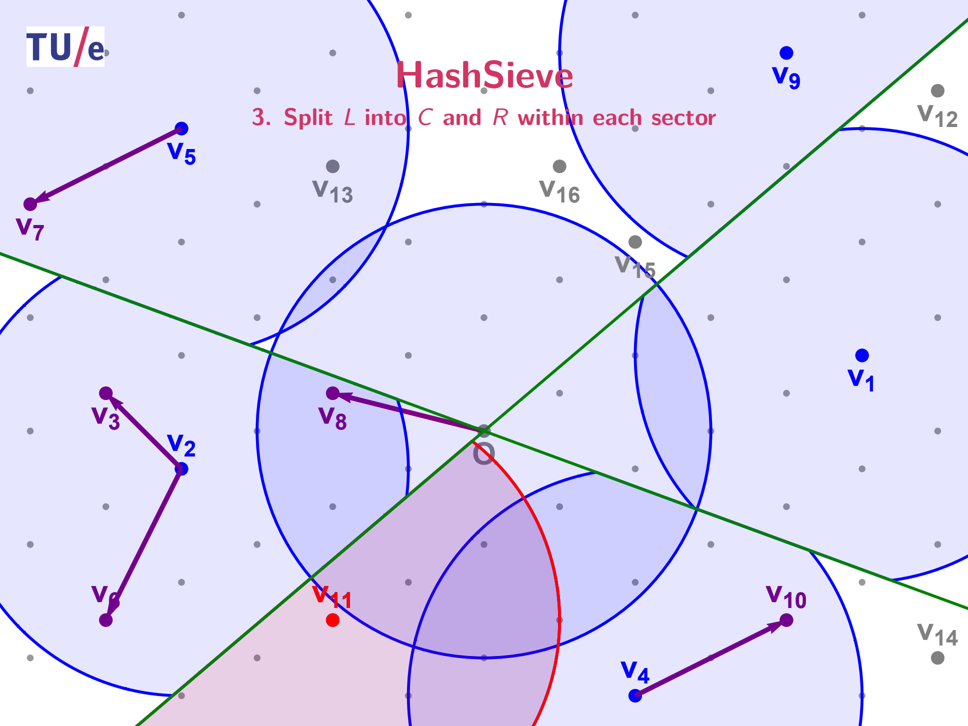
HashSieve

3. Split L into C and R within each sector



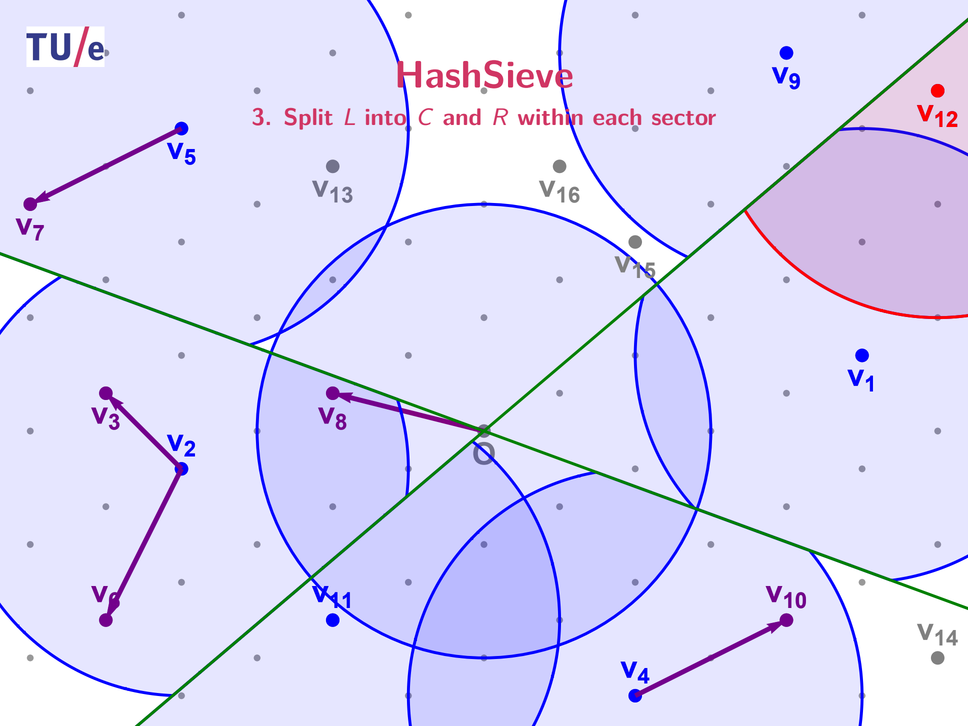
HashSieve

3. Split L into C and R within each sector



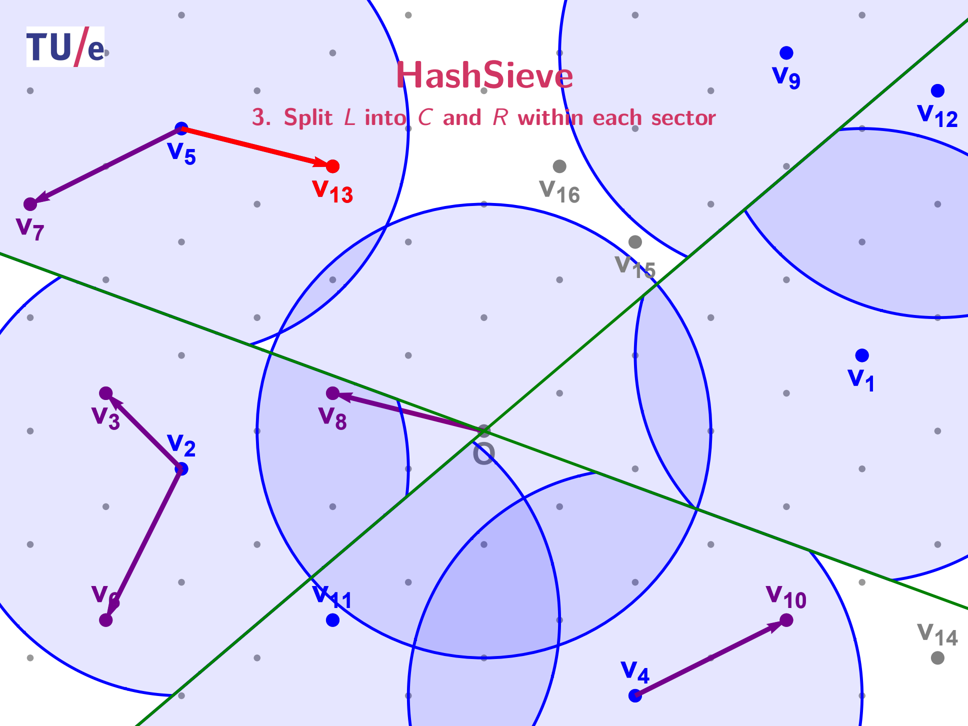
HashSieve

3. Split L into C and R within each sector



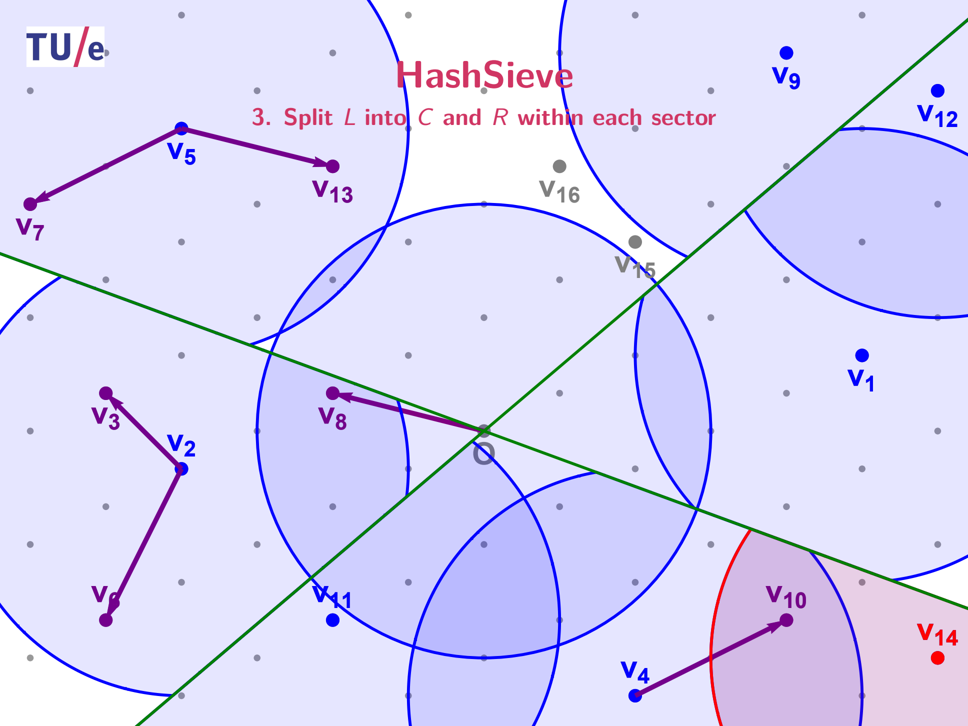
HashSieve

3. Split L into C and R within each sector



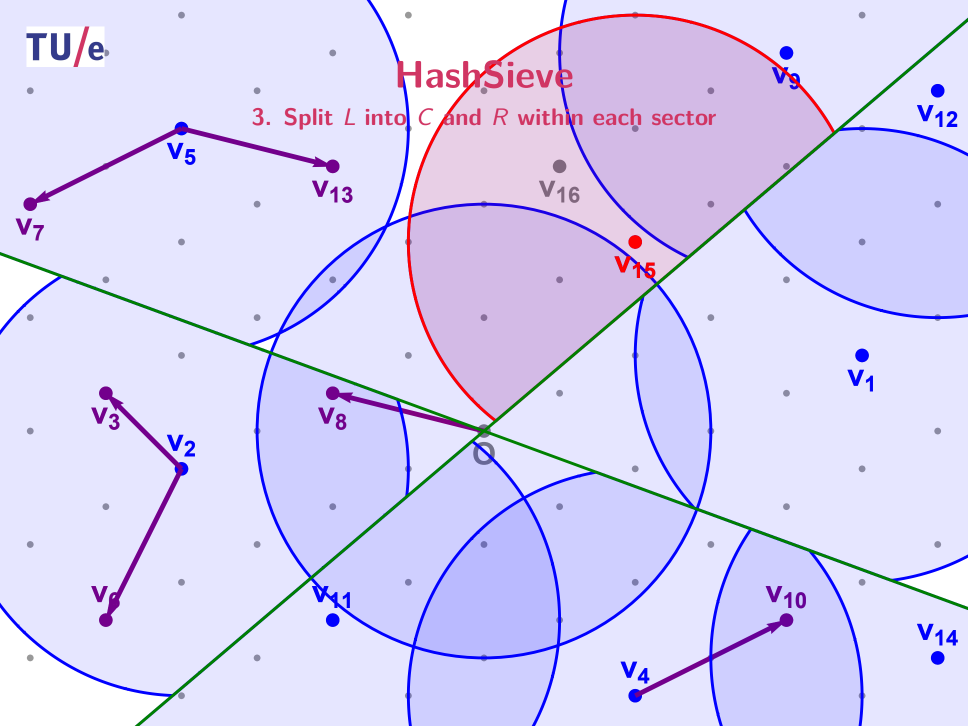
HashSieve

3. Split L into C and R within each sector



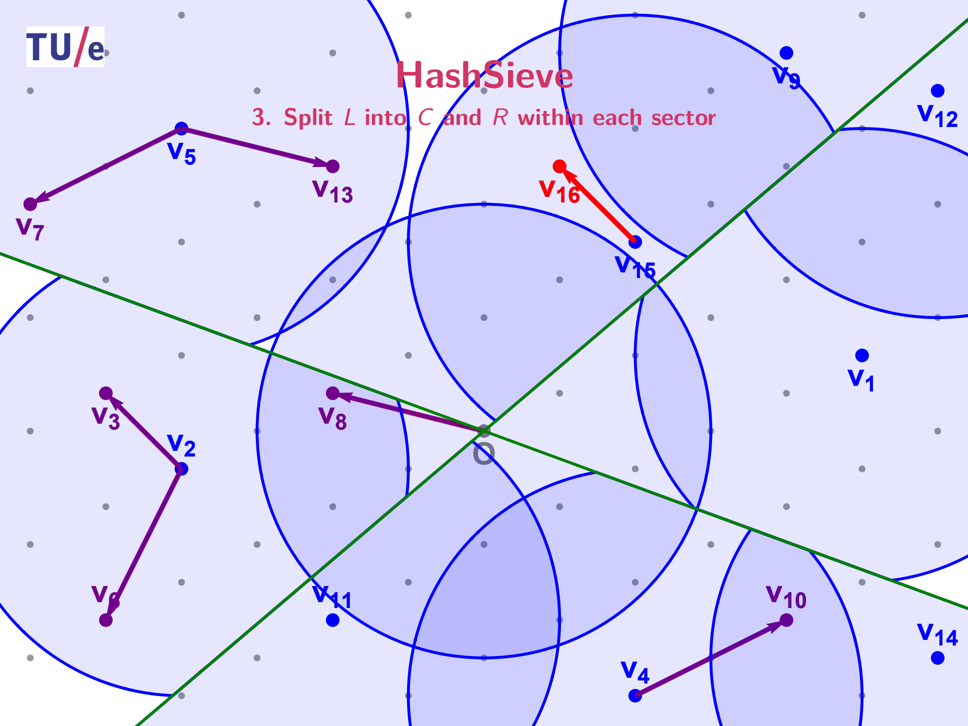
HashSieve

3. Split L into C and R within each sector



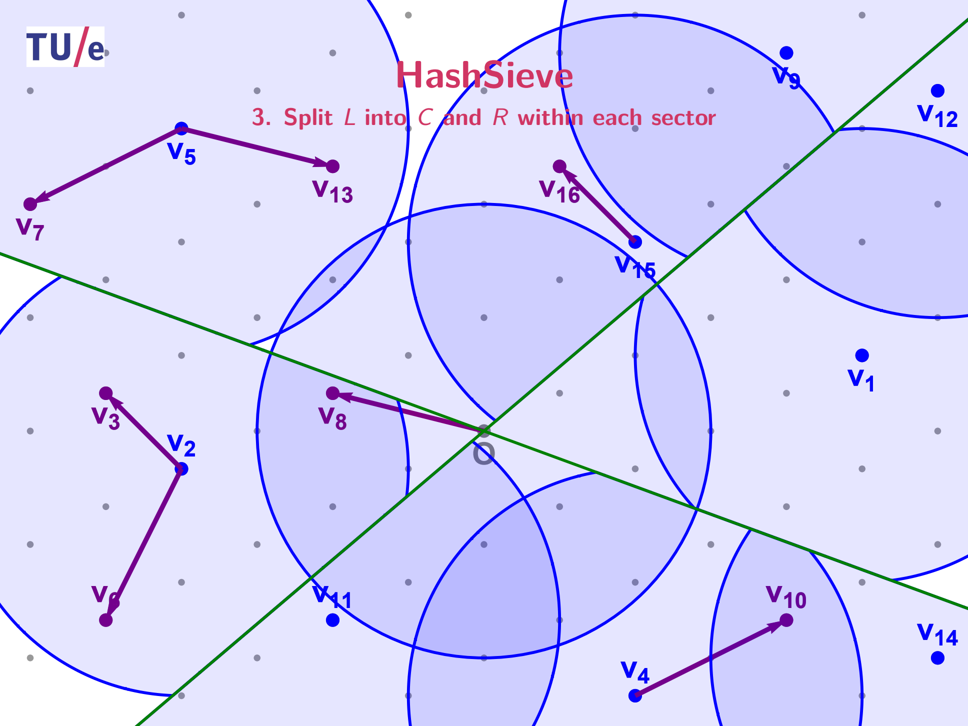
HashSieve

3. Split L into C and R within each sector



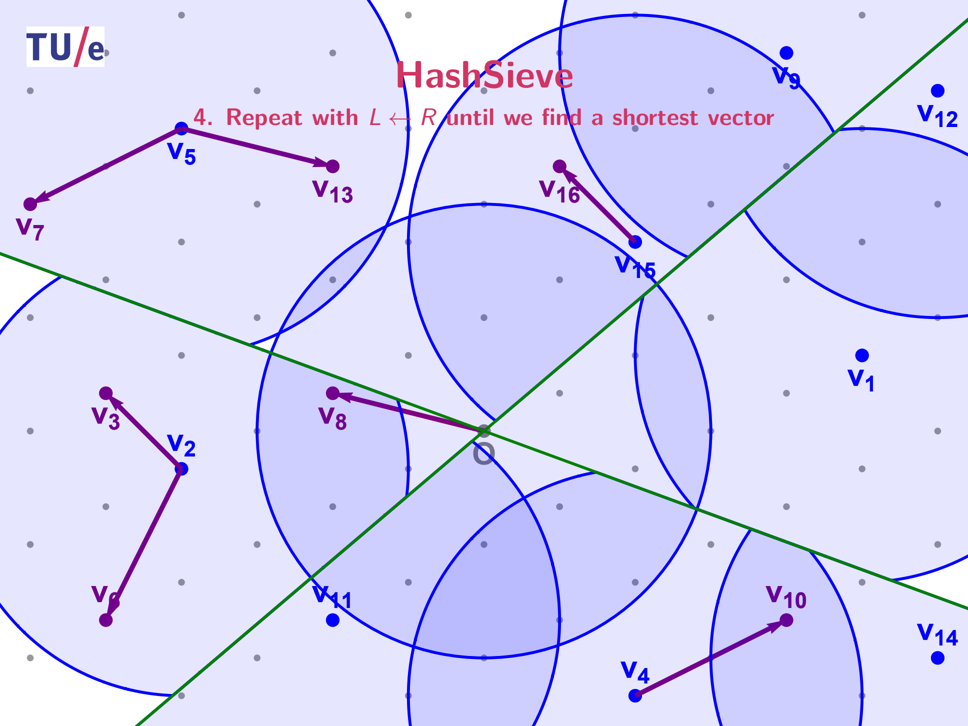
HashSieve

3. Split L into C and R within each sector



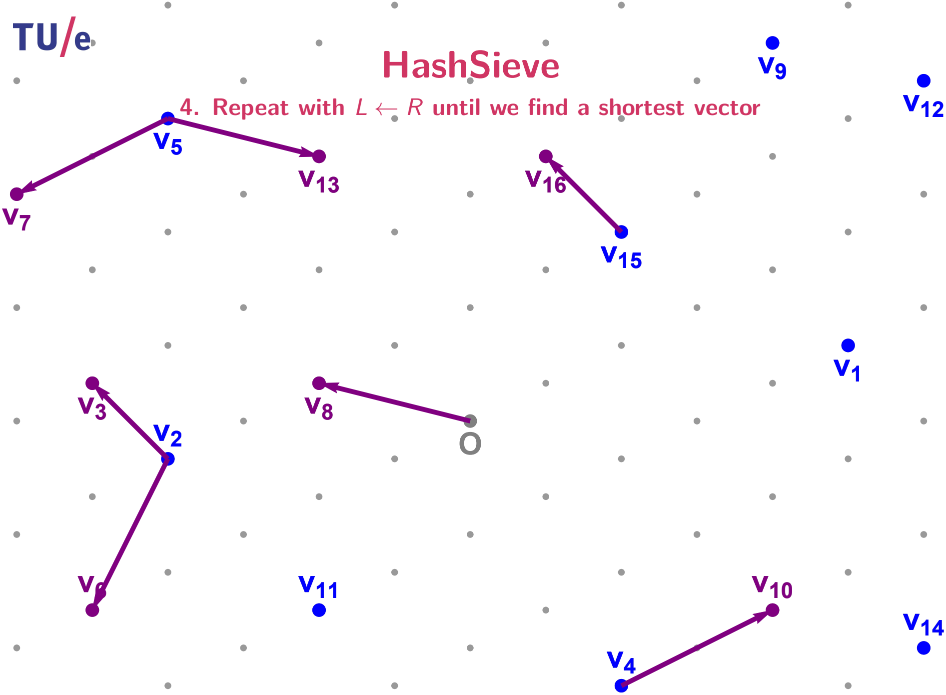
HashSieve

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



HashSieve

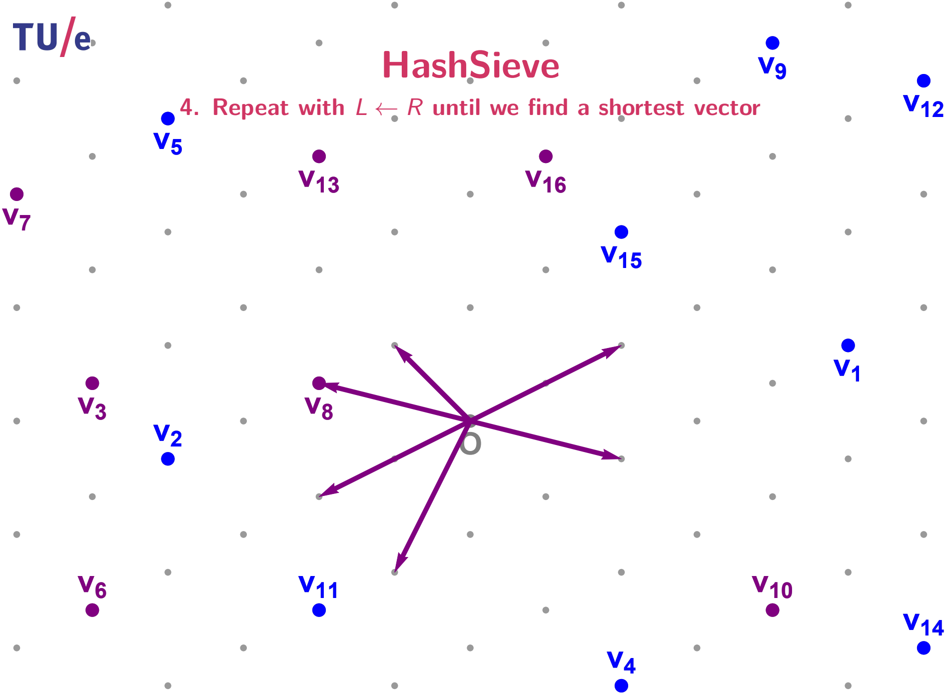
4. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

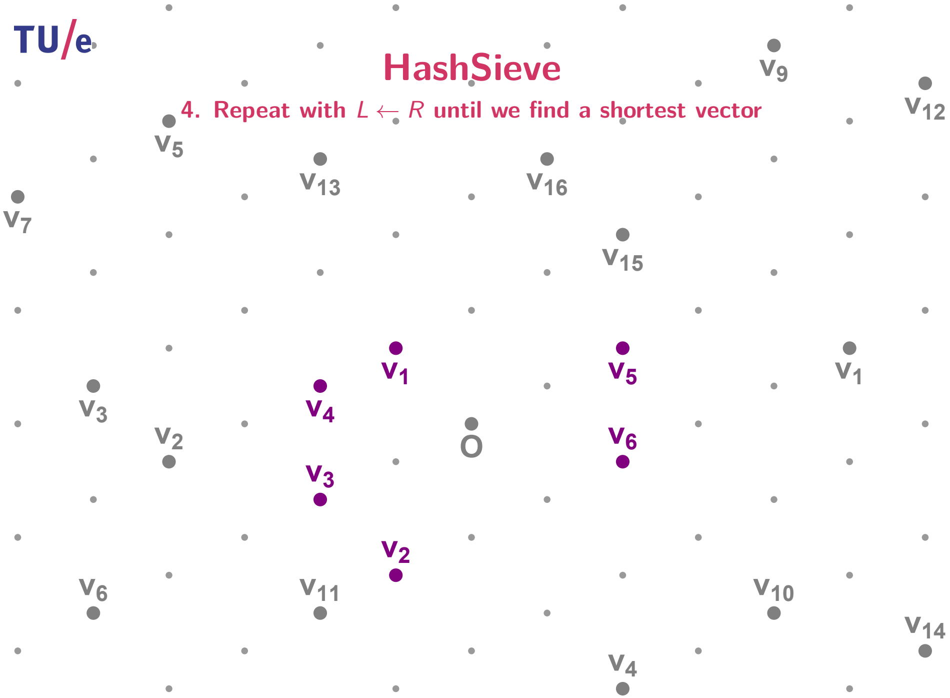
HashSieve

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



HashSieve

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



HashSieve

Overview

HashSieve

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

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

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

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

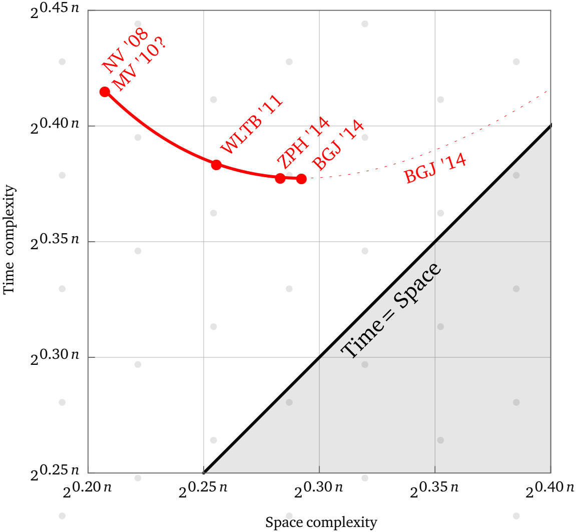
HashSieve

Experimental results

Dimension	90	92	94	96	98	100
Probing?	No			Yes (One level probing)		
BKZ-β	34				40	
Klein's d	70					
Sample Pool (k)	100	100	100	150	150	250
K	20	20	21	21	22	22
T	3126	3738	4470	465	533	637
Optimal K/T?	Yes			No		
Target norm?	No					
Used vectors (k)	≈2425	≈2998	≈4501	≈5565	≈7050	≈10054
Solution	2419	2440	2526	2522	2541	2571
Time (h)	0.86	1.72	3.74	6.52	10.03	18.19
Memory (GB)	≈310	≈380	≈872	≈95	≈113	≈256

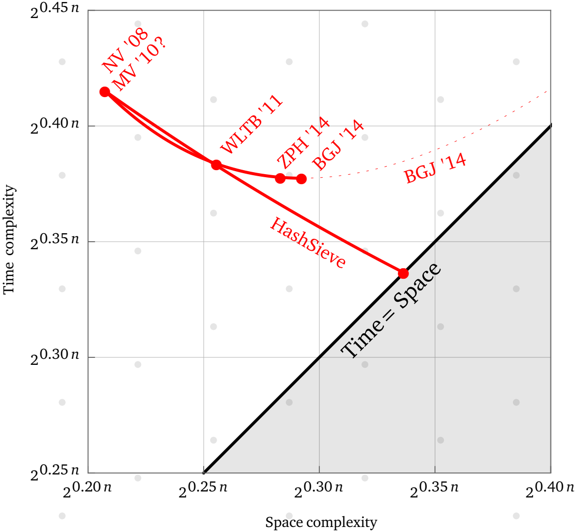
HashSieve

Space/time trade-off



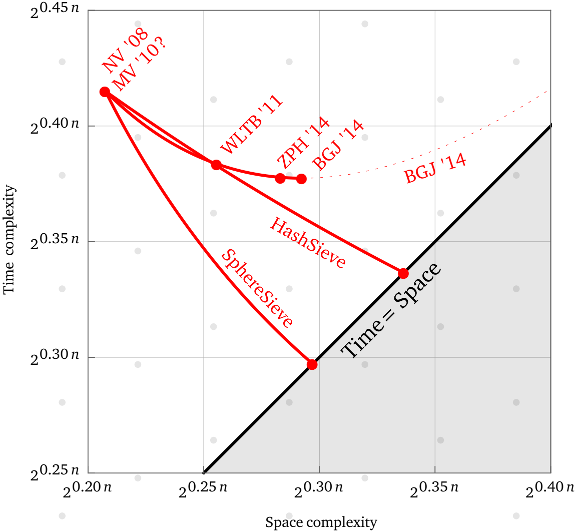
HashSieve

Space/time trade-off



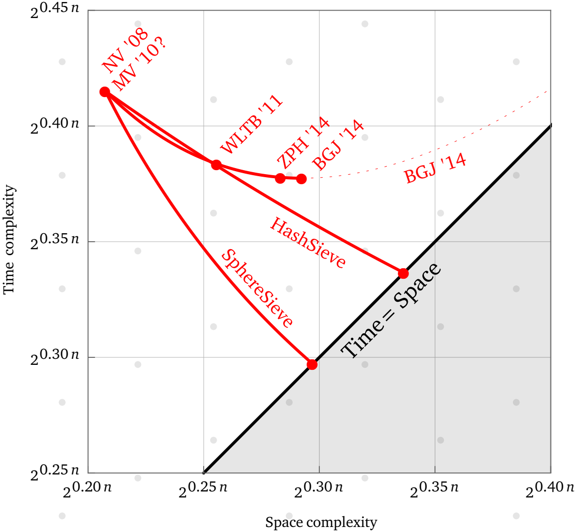
SphereSieve

Space/time trade-off



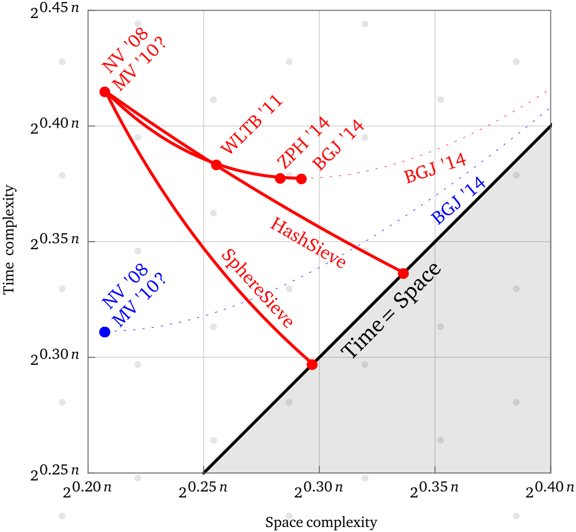
Sieving with quantum search

Space/time trade-off



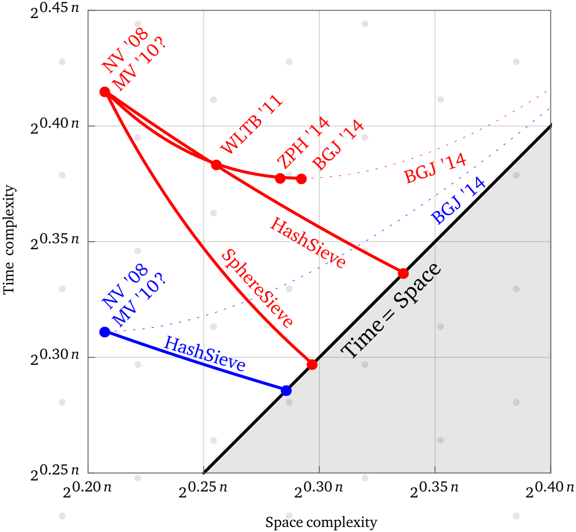
Sieving with quantum search

Space/time trade-off



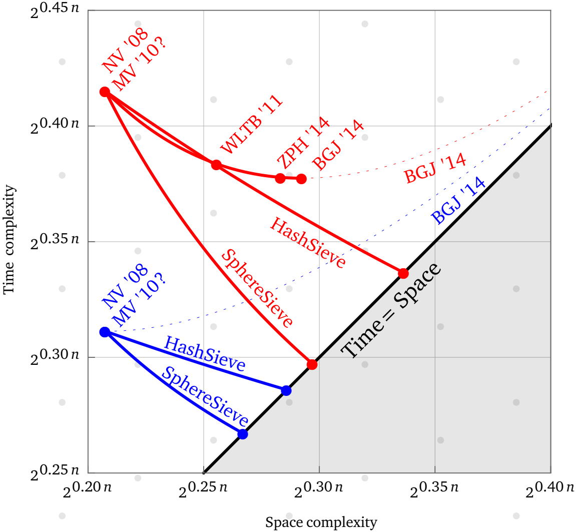
Sieving with quantum search

Space/time trade-off



Sieving with quantum search

Space/time trade-off



Questions