Sieving for shortest vectors in lattices using locality-sensitive hashing

Thijs Laarhoven

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

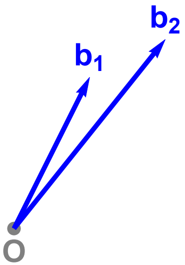
Microsoft Research, Redmond (WA), USA
(June 22, 2015)

Lattices

What is a lattice?

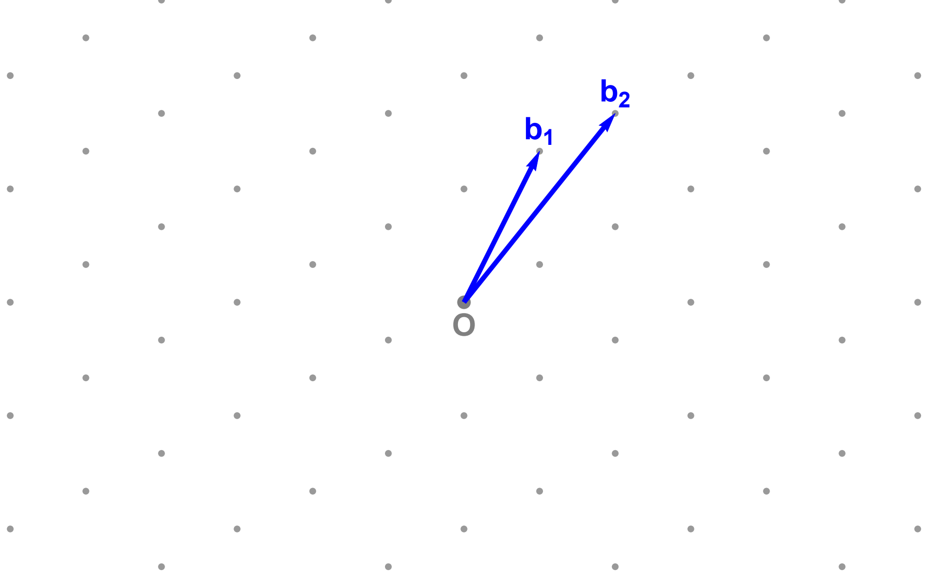
Lattices

What is a lattice?



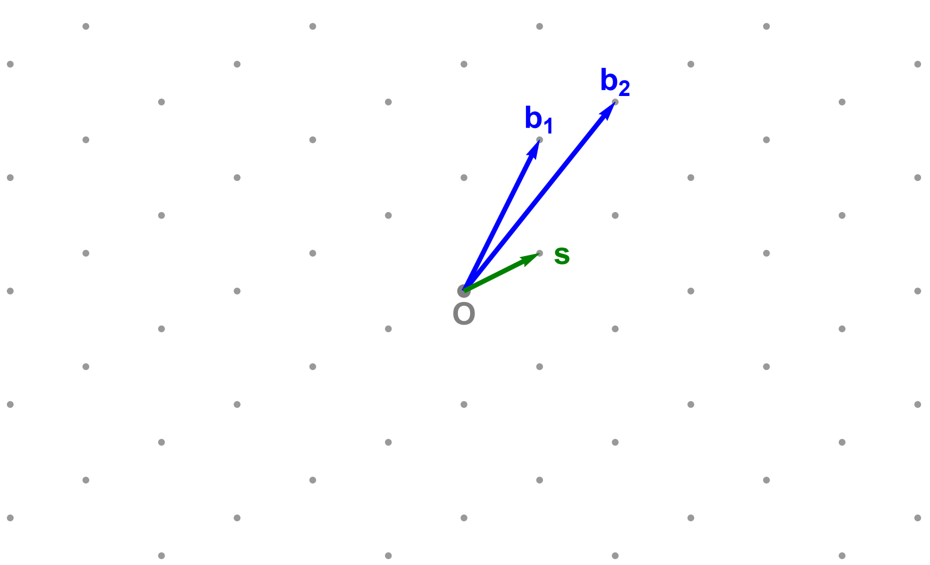
Lattices

What is a lattice?



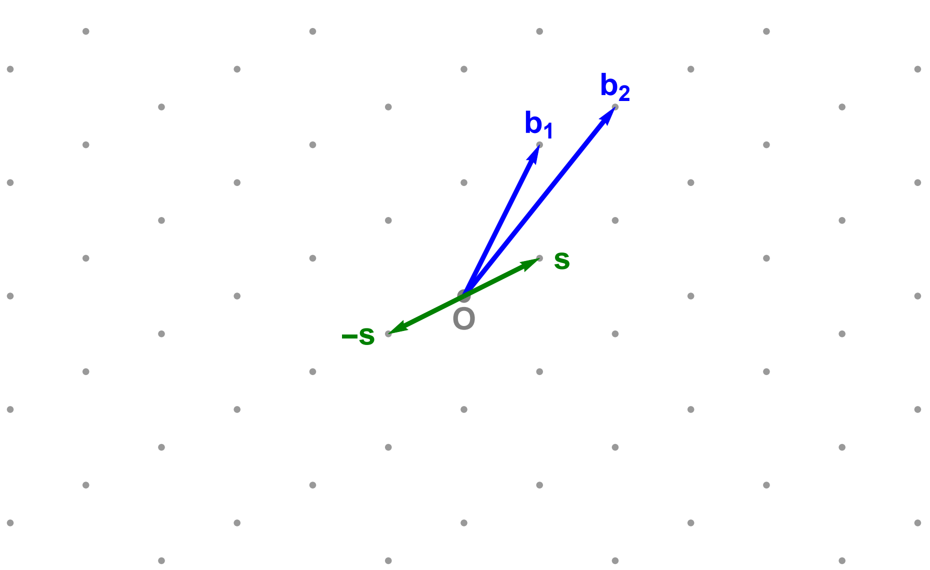
Lattices

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]	$0.298n$	$0.208n$*

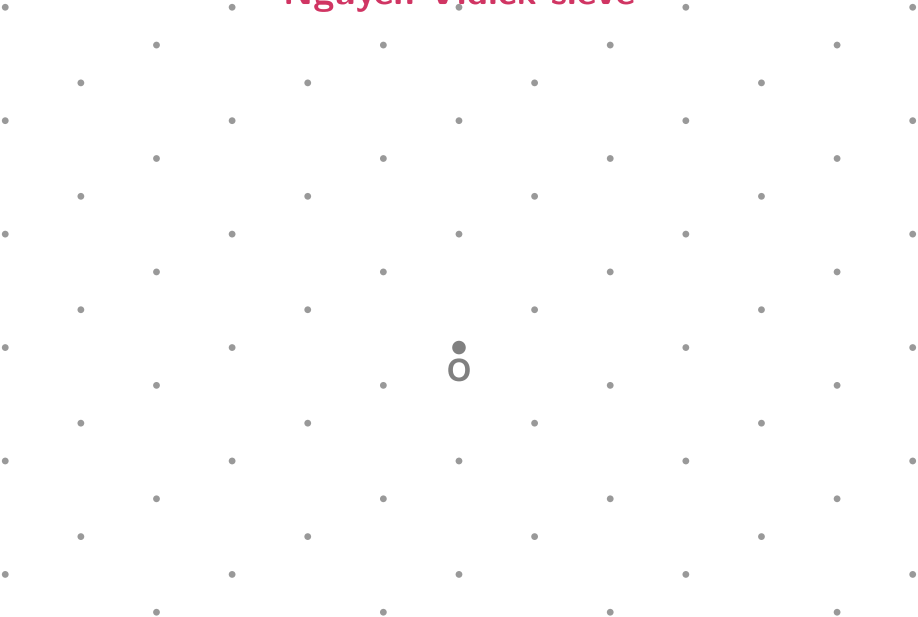
TU/e

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



TU/e

Nguyen-Vidick sieve

1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve

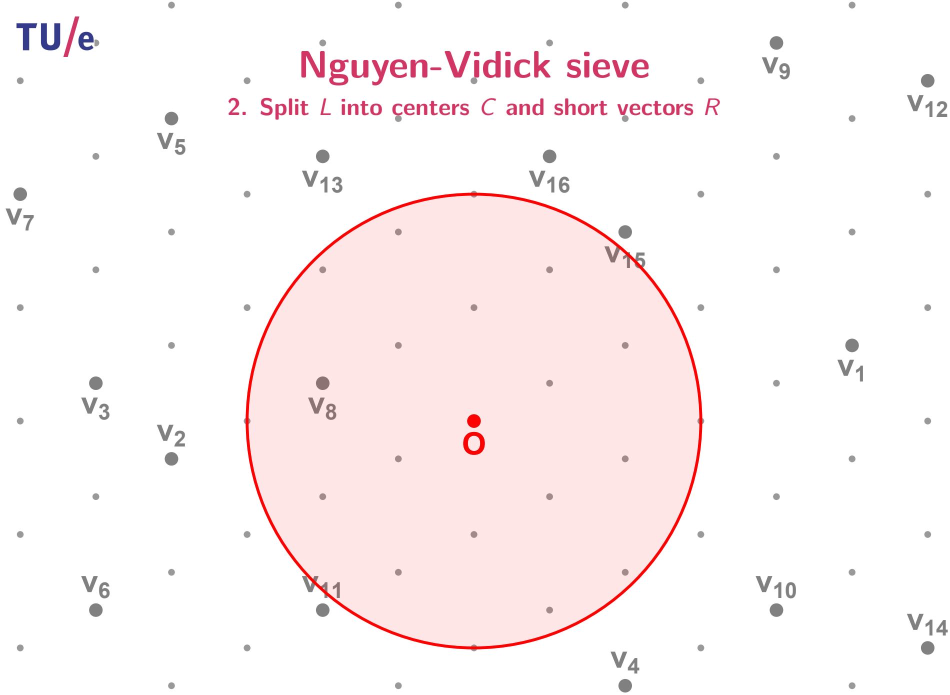
1. Sample a list L of random lattice vectors

Nguyen-Vidick sieve

2. Split L into centers C and short vectors R

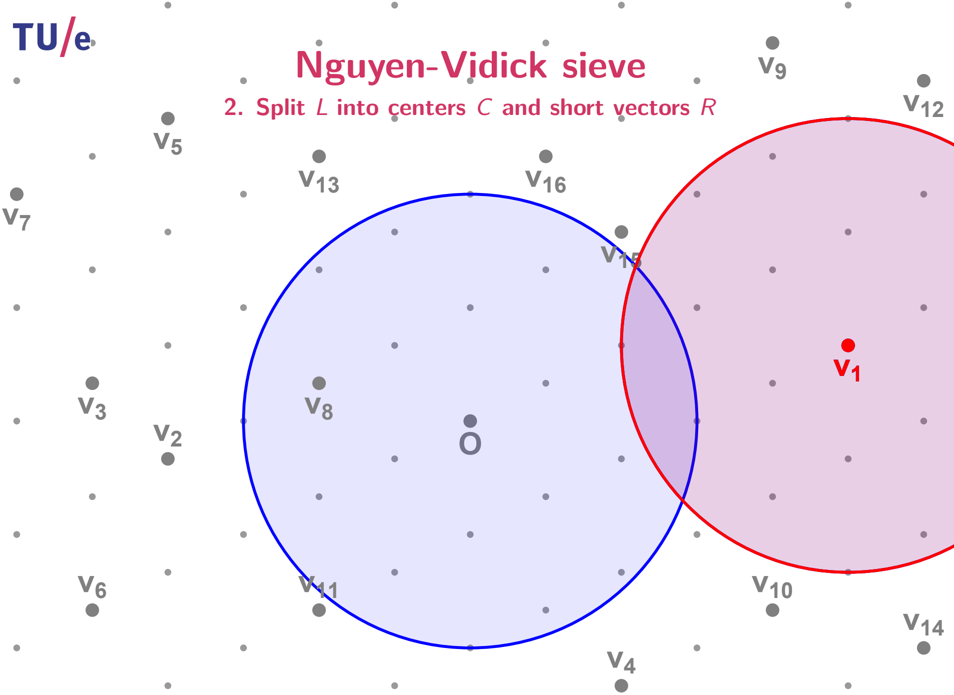
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



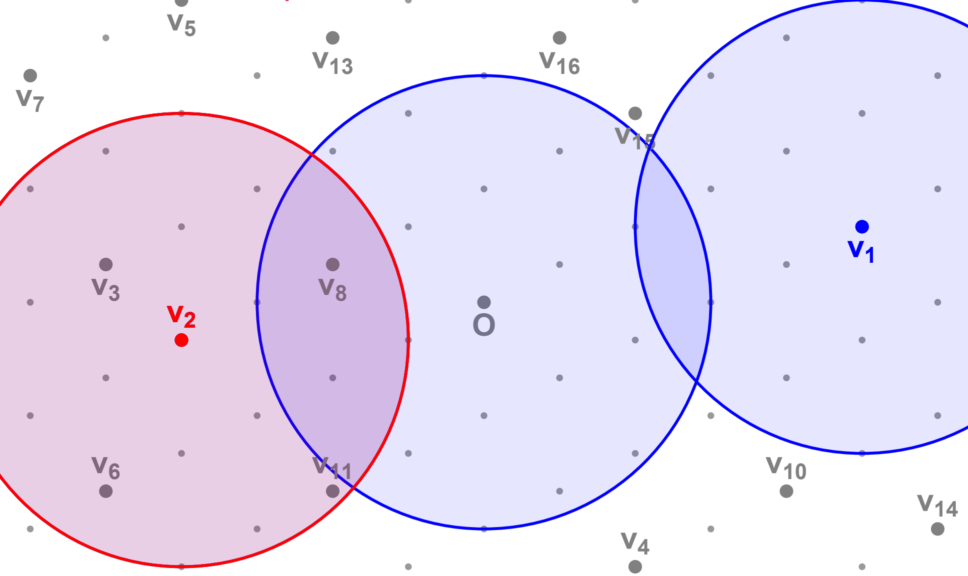
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



Nguyen-Vidick sieve

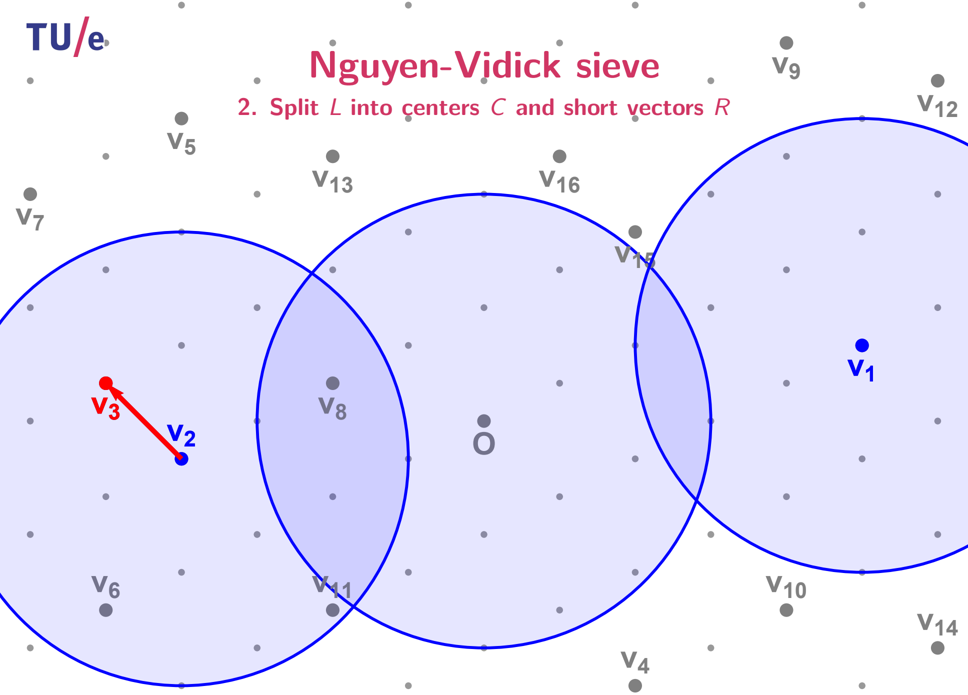
2. Split L into centers C and short vectors R



TU/e

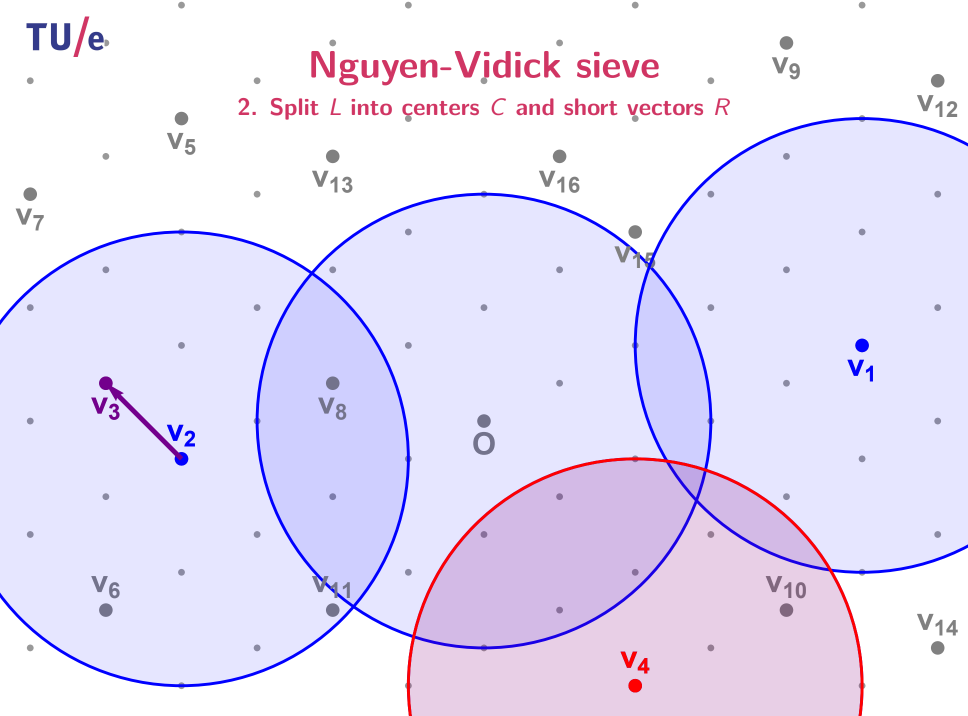
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



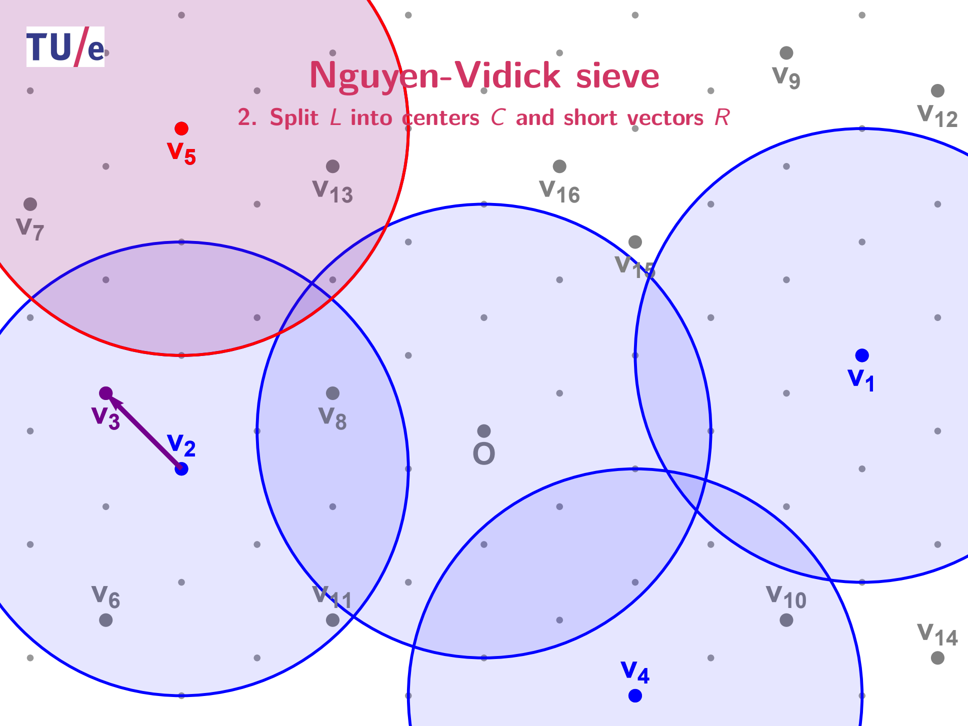
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



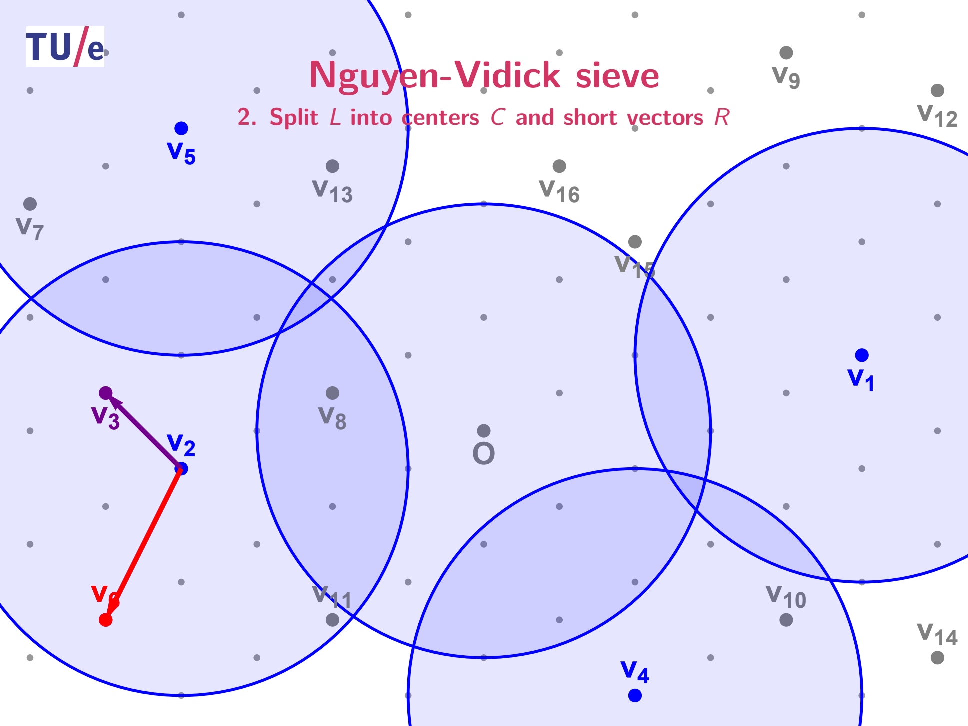
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



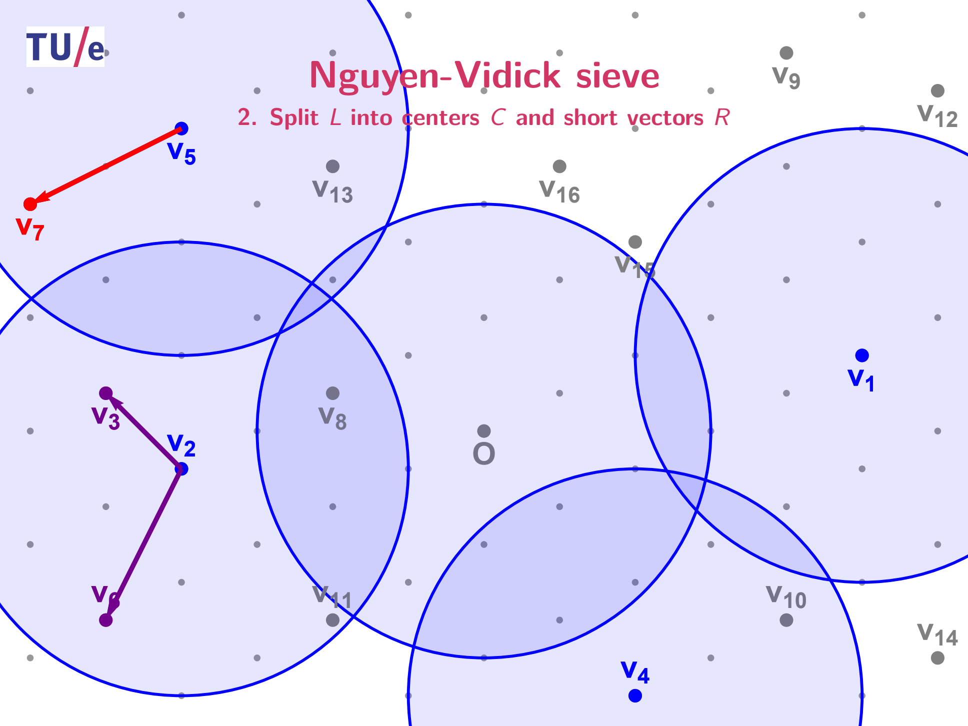
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



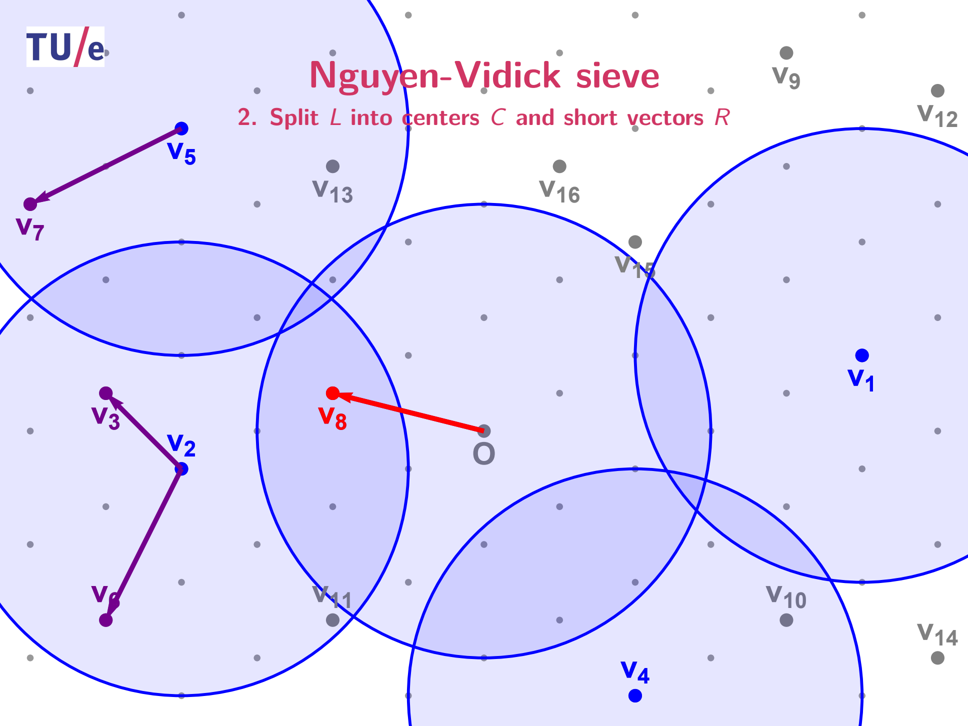
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



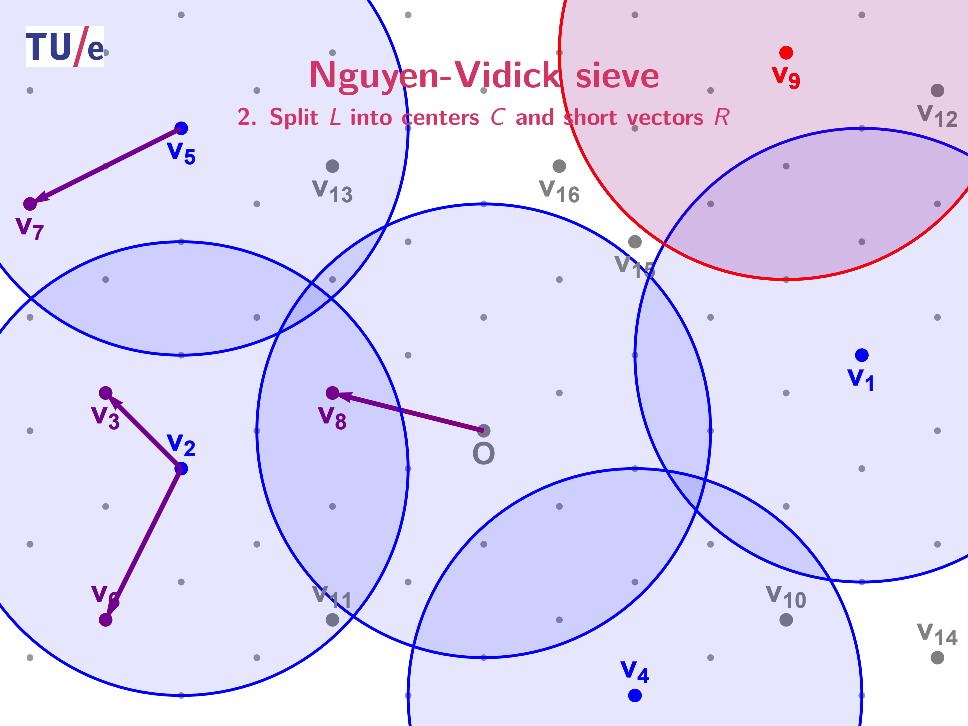
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



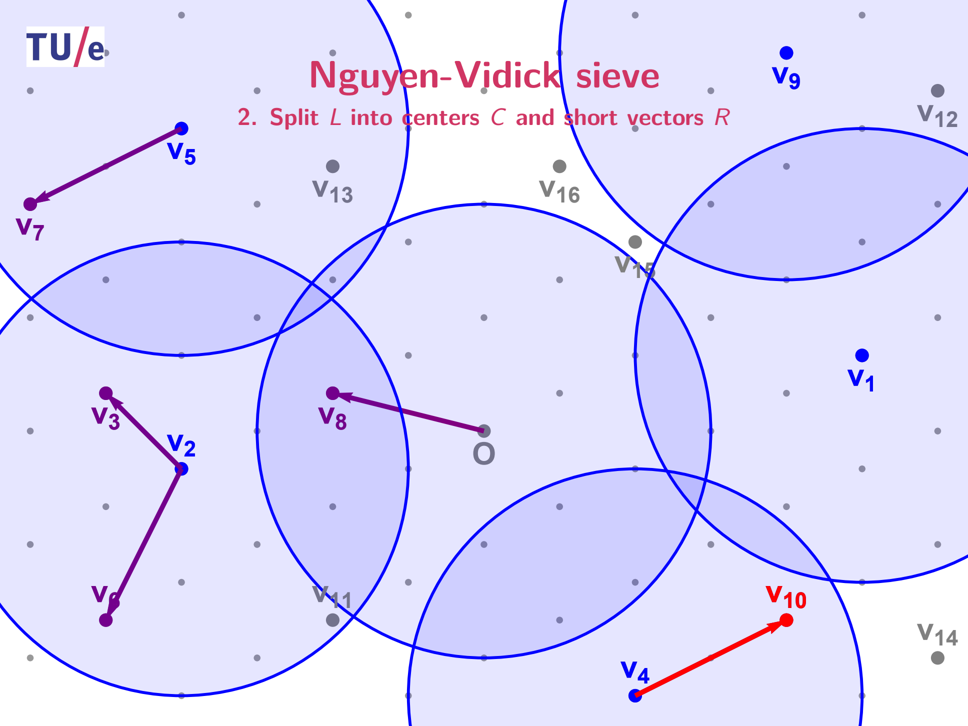
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



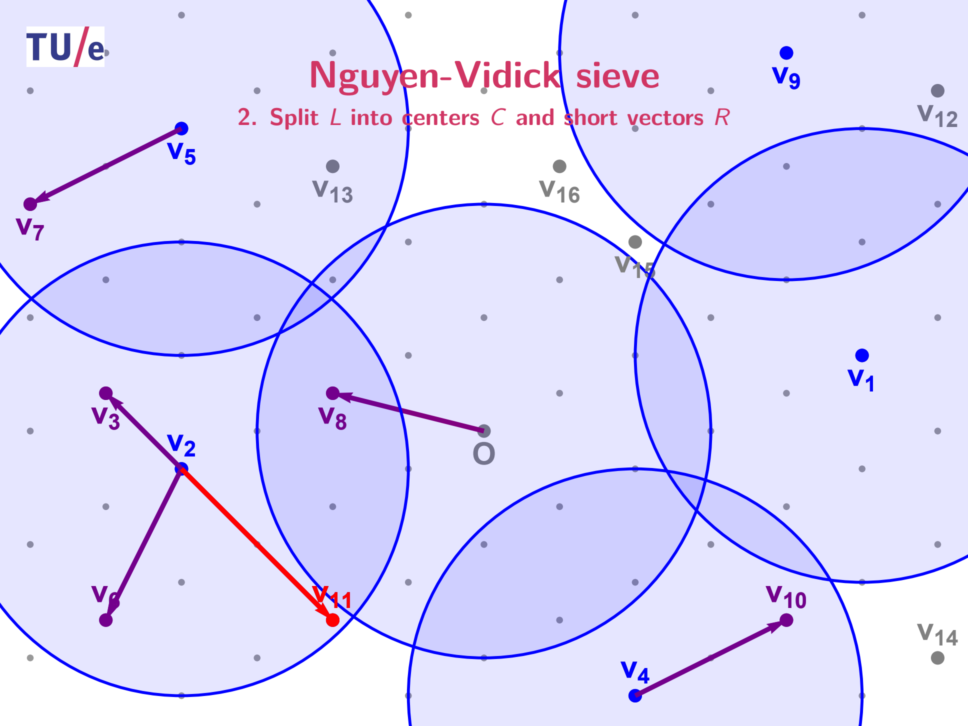
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



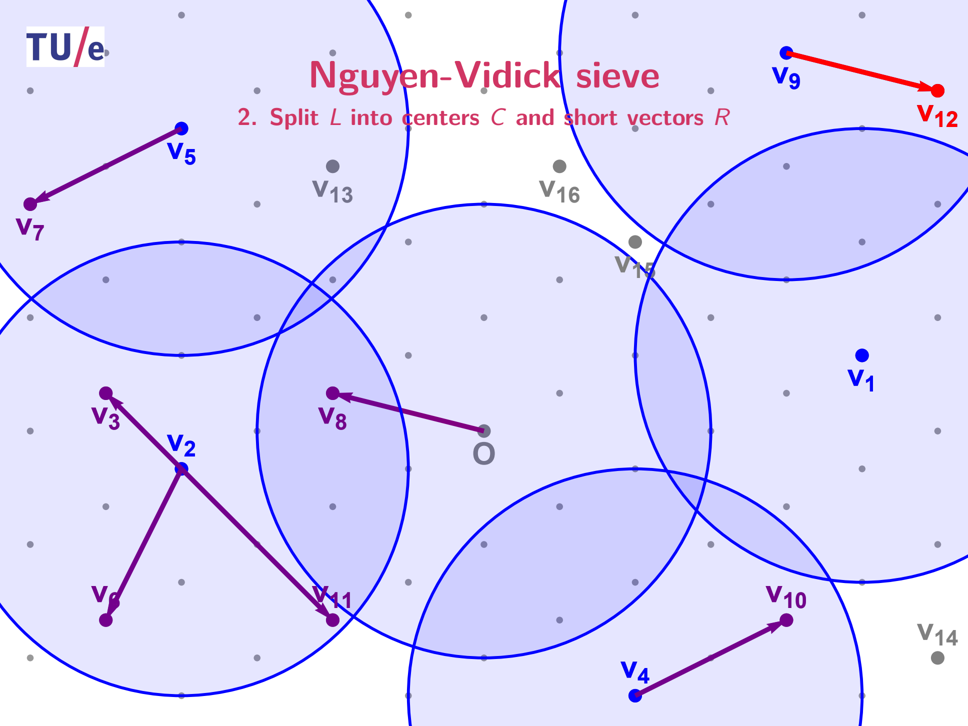
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



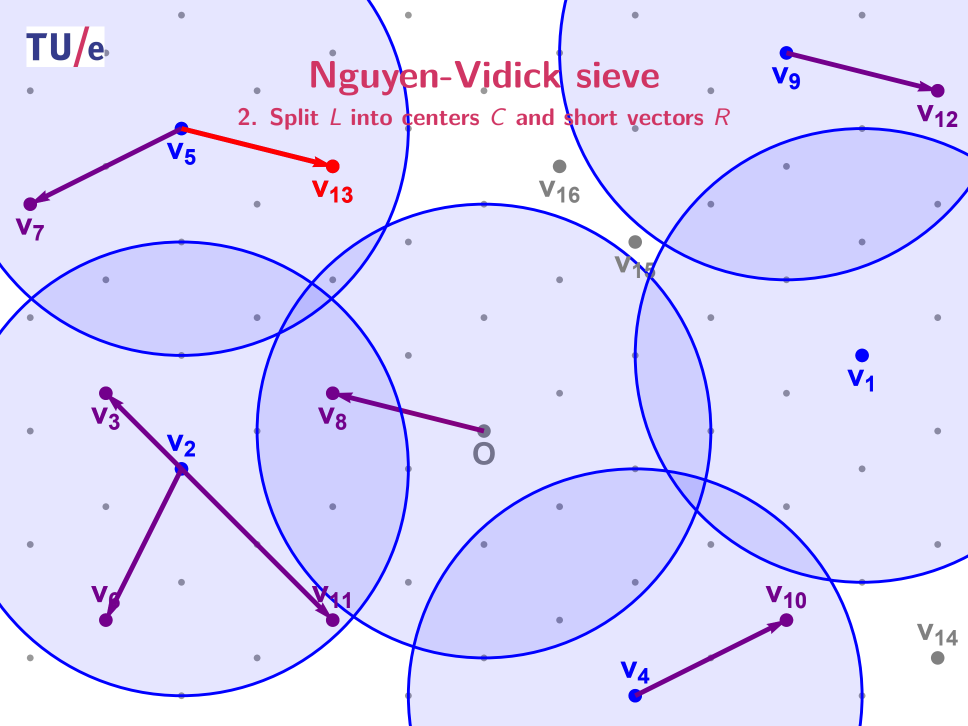
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



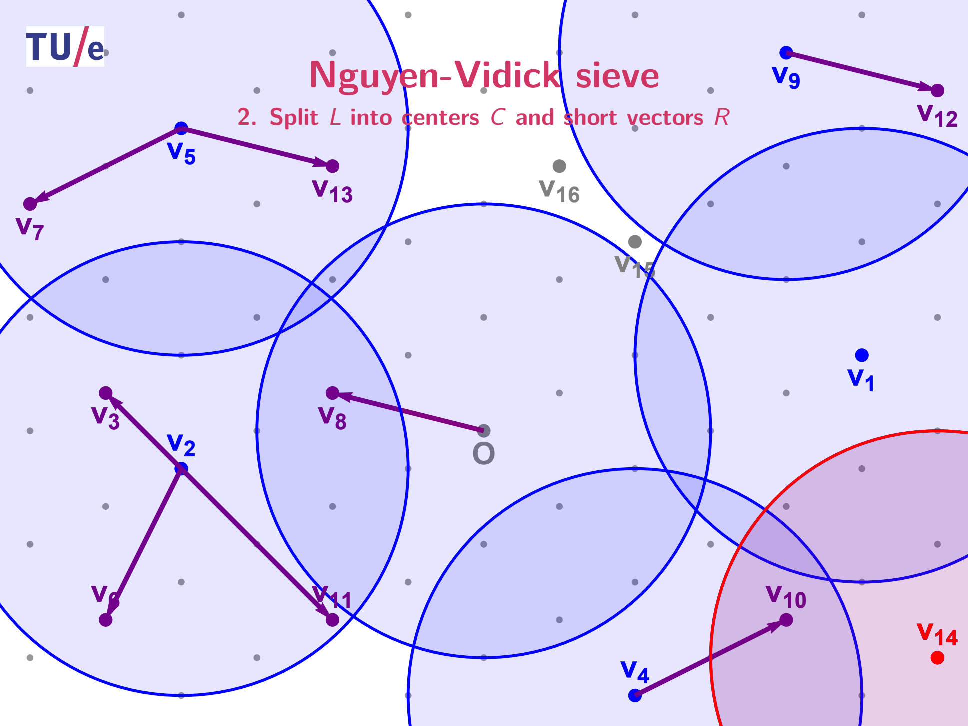
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



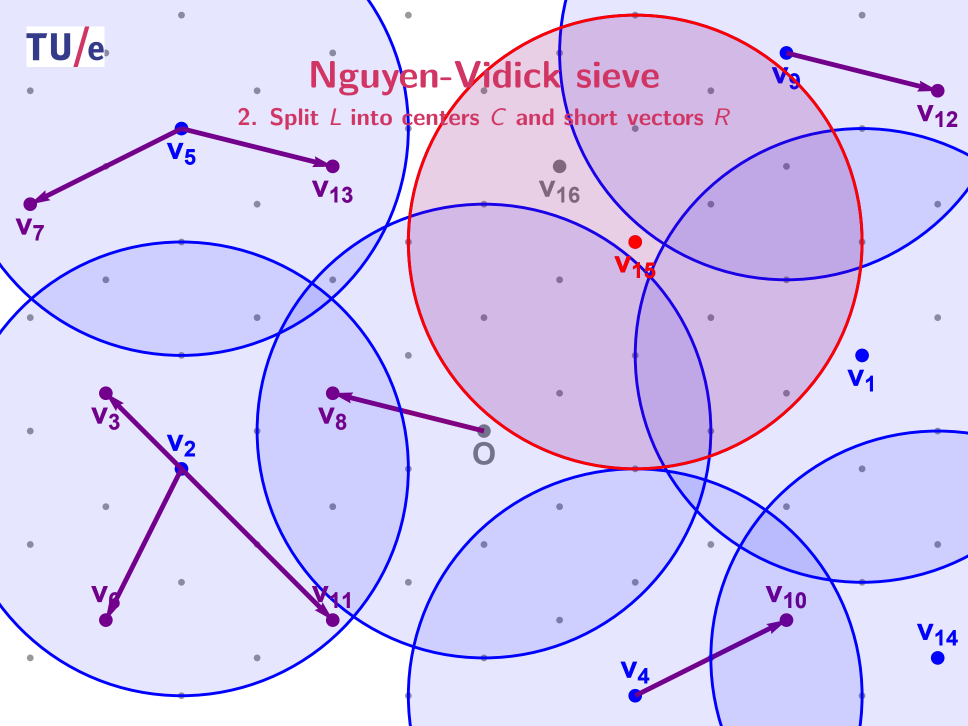
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



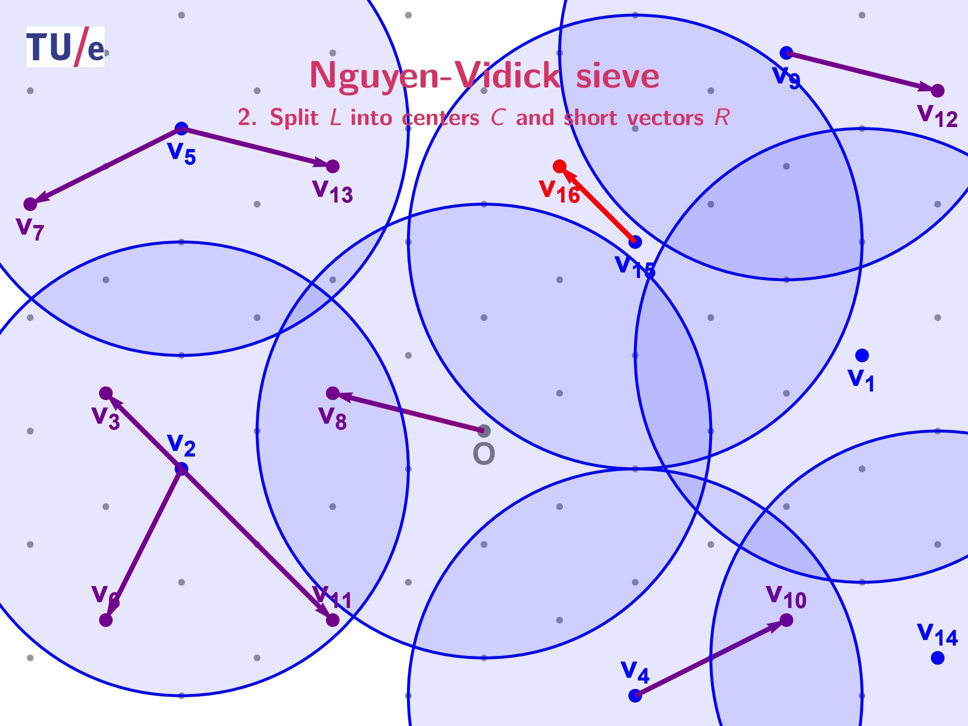
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



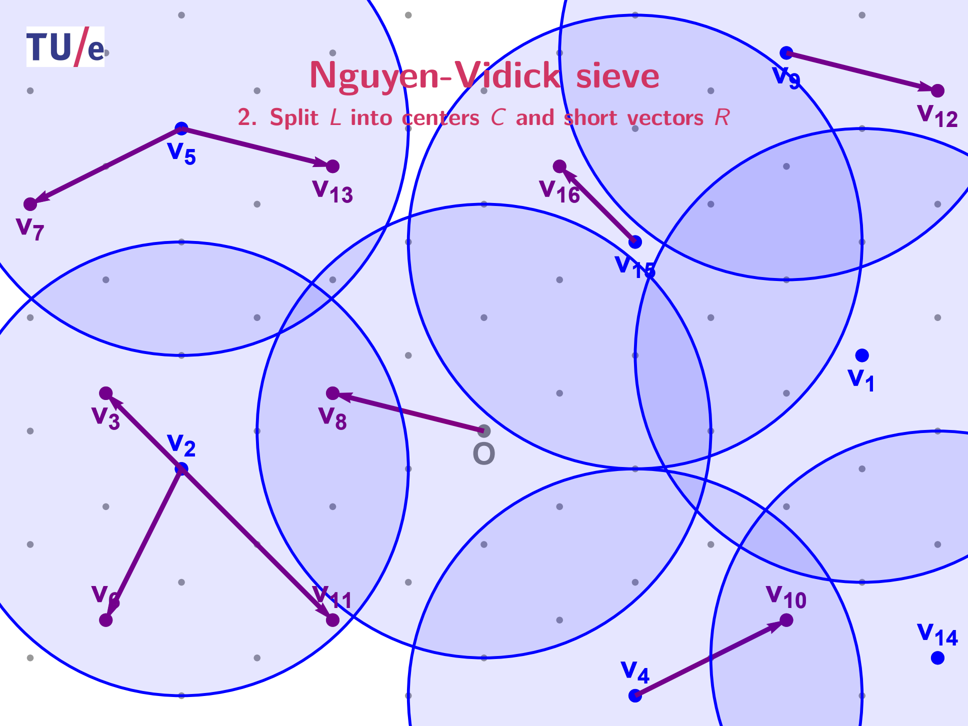
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



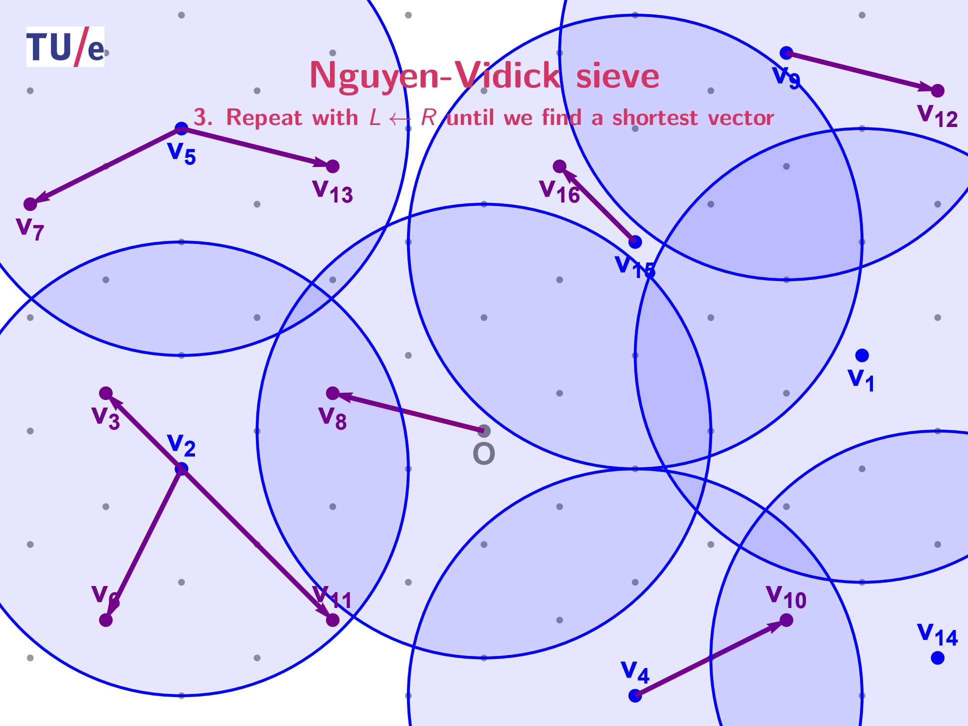
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



Nguyen-Vidick sieve

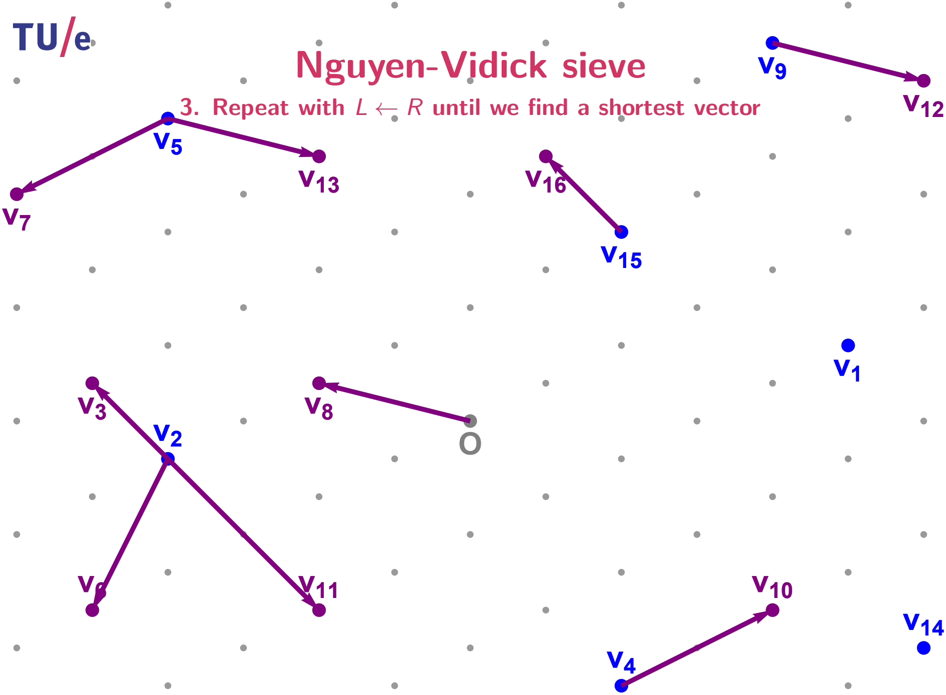
3. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

Nguyen-Vidick sieve

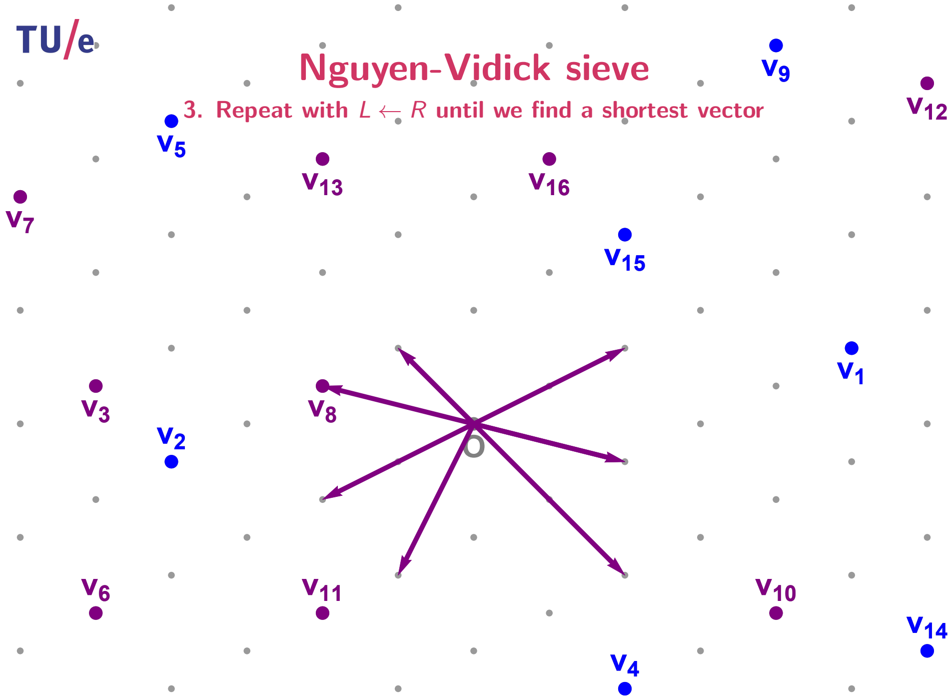
3. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

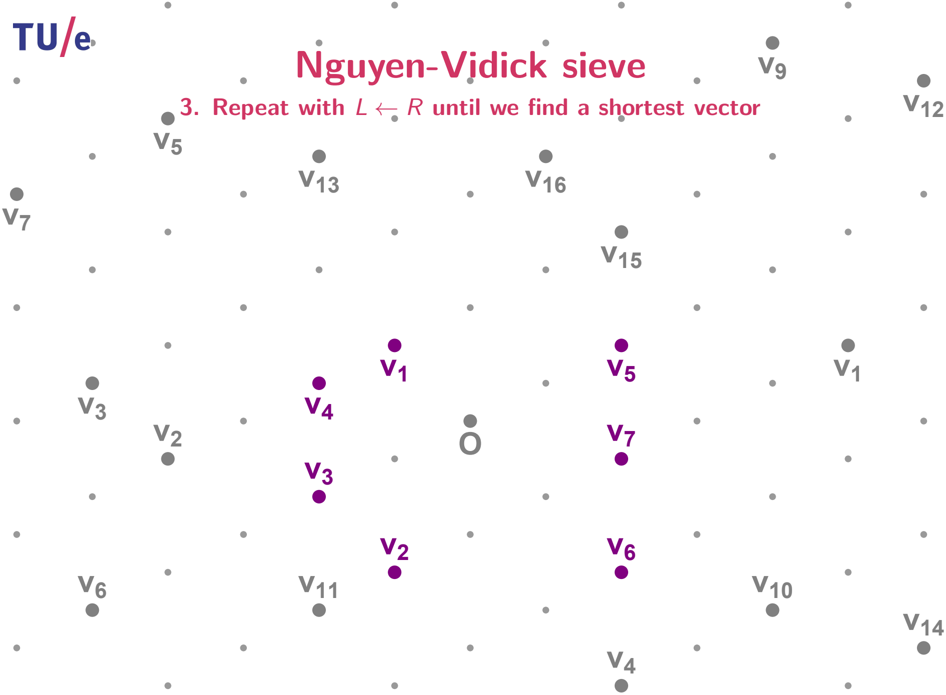
Nguyen-Vidick sieve

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

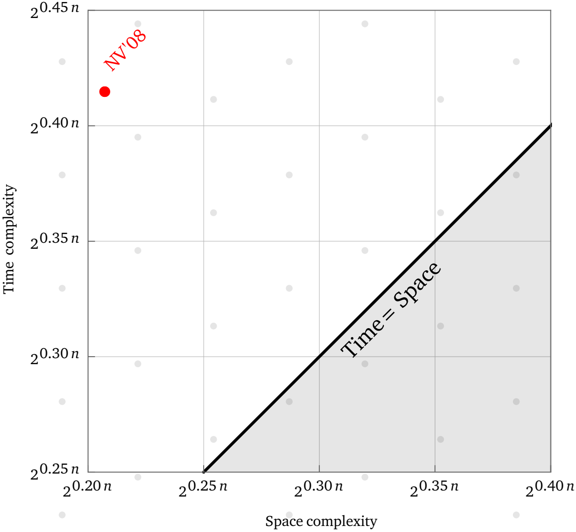
- Space complexity: $\sqrt{4/3}^n \approx 2^{0.21n+o(n)}$ vectors
  - Need $\sqrt{4/3}^n$ vectors to cover all corners of $\mathbb{R}^n$
- Time complexity: $(4/3)^n \approx 2^{0.42n+o(n)}$
  - Comparing a target vector to all centers: $2^{0.21n+o(n)}$
  - Repeating this for each list vector: $2^{0.21n+o(n)}$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

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

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

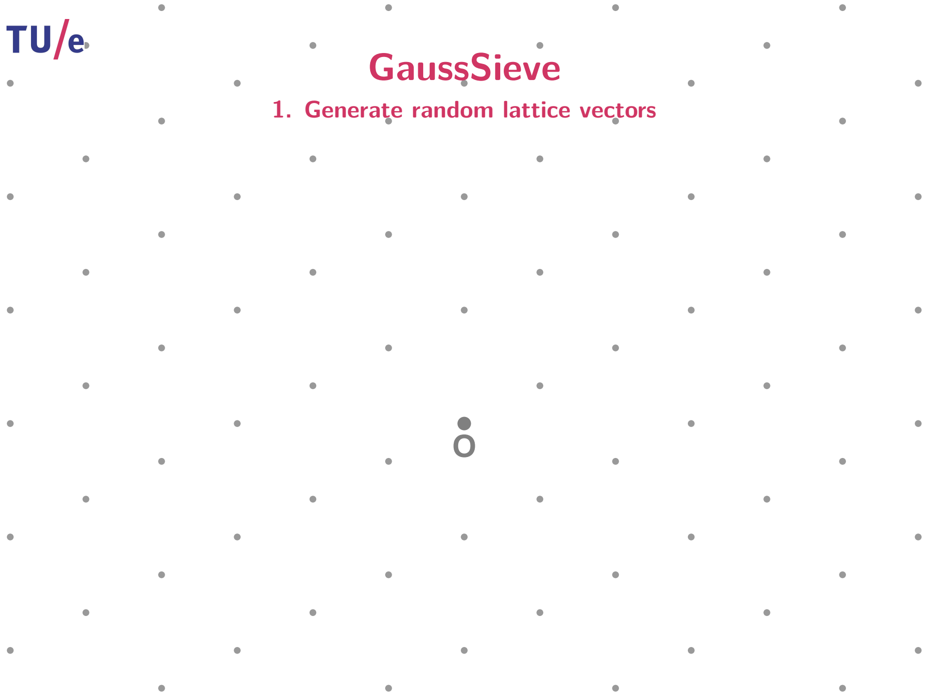
Nguyen-Vidick sieve

Space/time trade-off



GaussSieve

1. Generate random lattice vectors

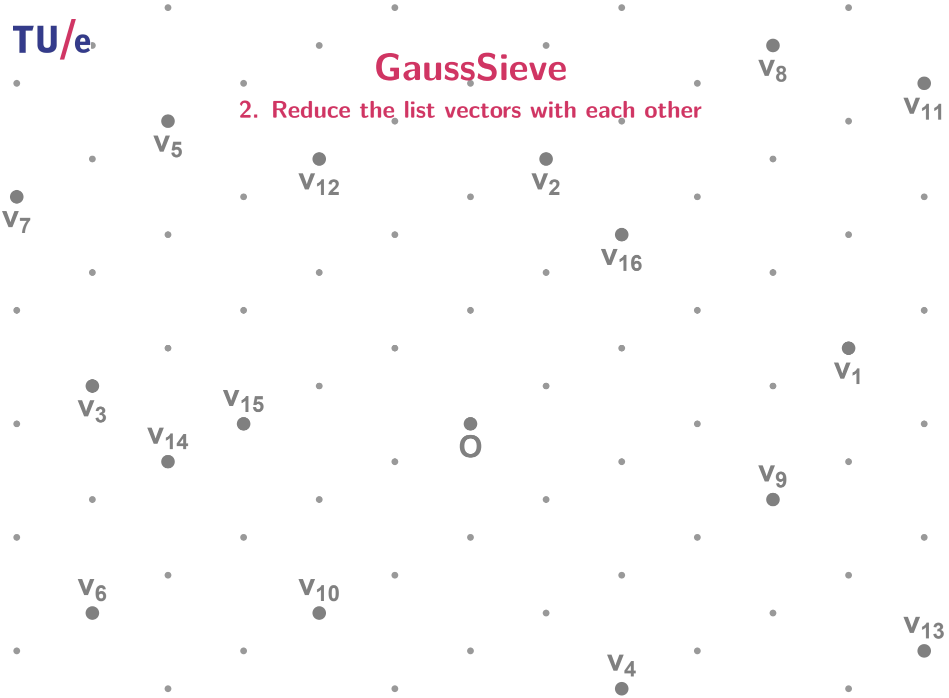


GaussSieve

1. Generate random lattice vectors

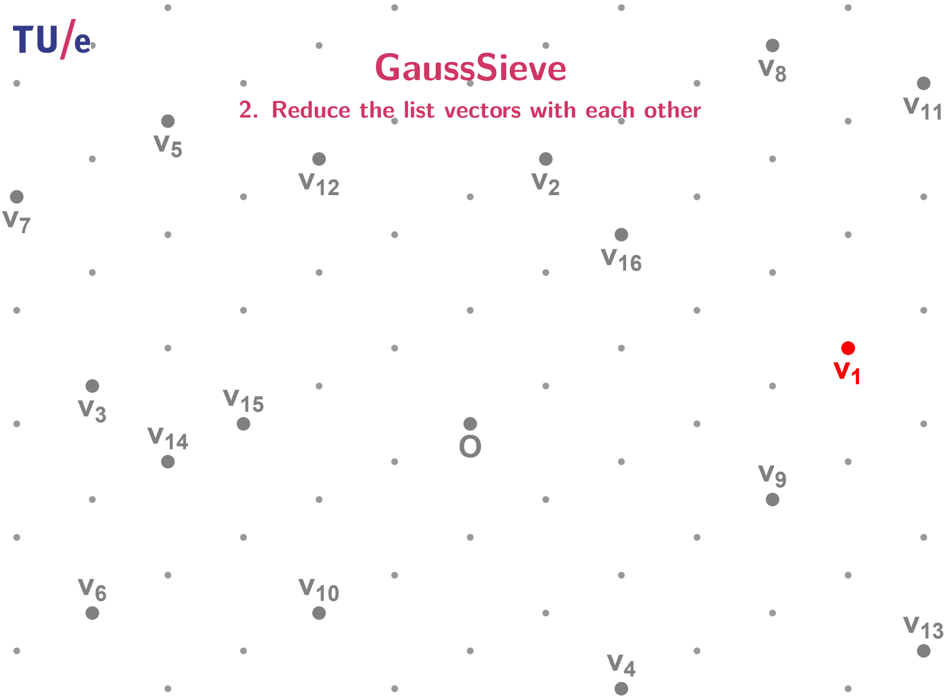
GaussSieve

2. Reduce the list vectors with each other



GaussSieve

2. Reduce the list vectors with each other

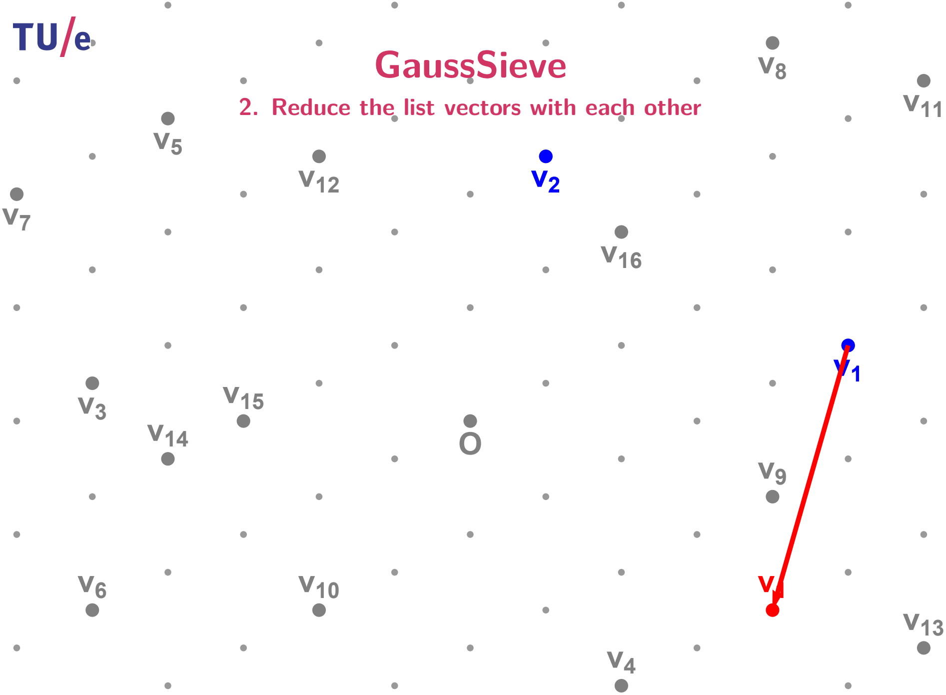


GaussSieve

2. Reduce the list vectors with each other

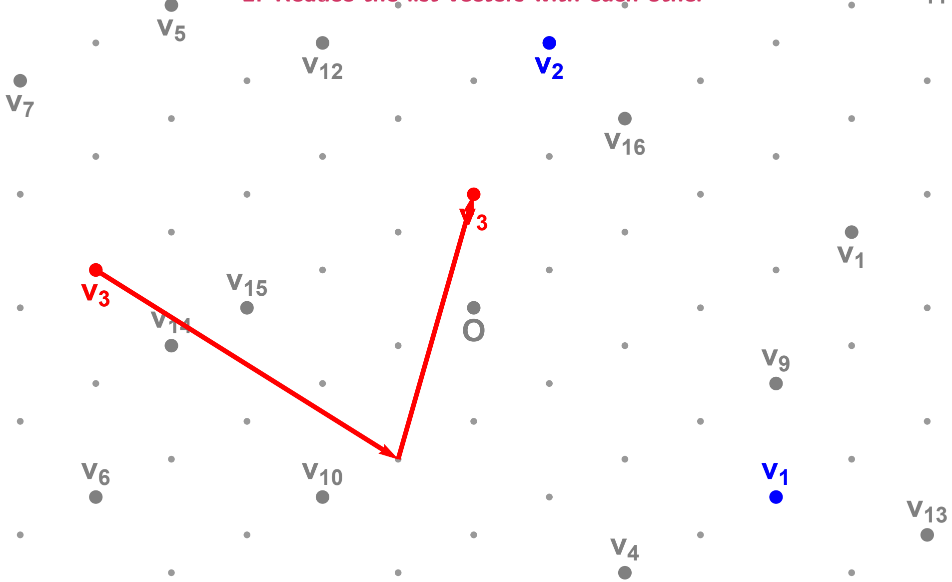
GaussSieve

2. Reduce the list vectors with each other



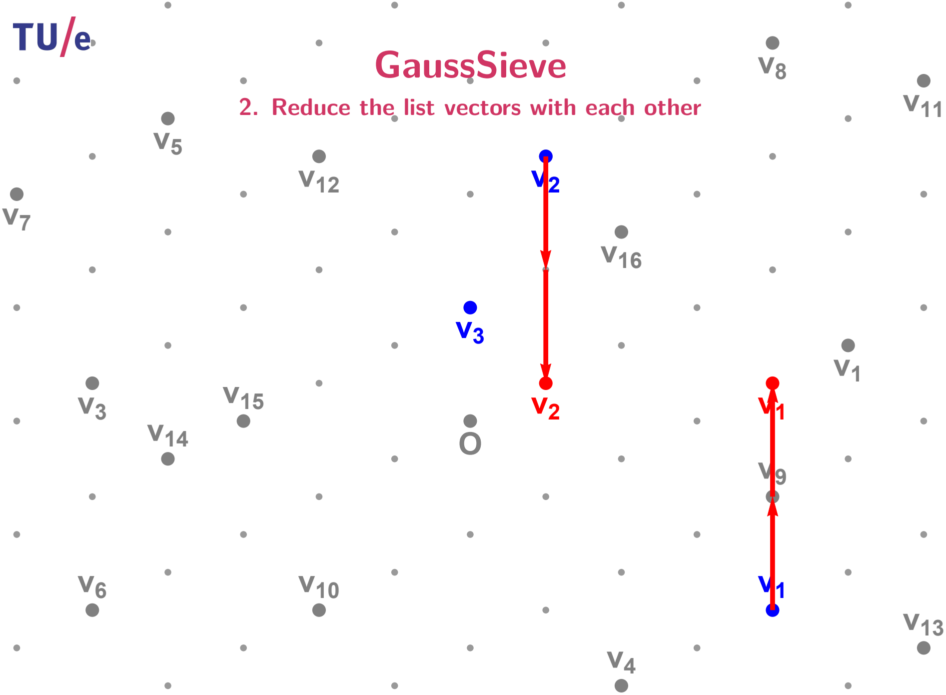
GaussSieve

2. Reduce the list vectors with each other



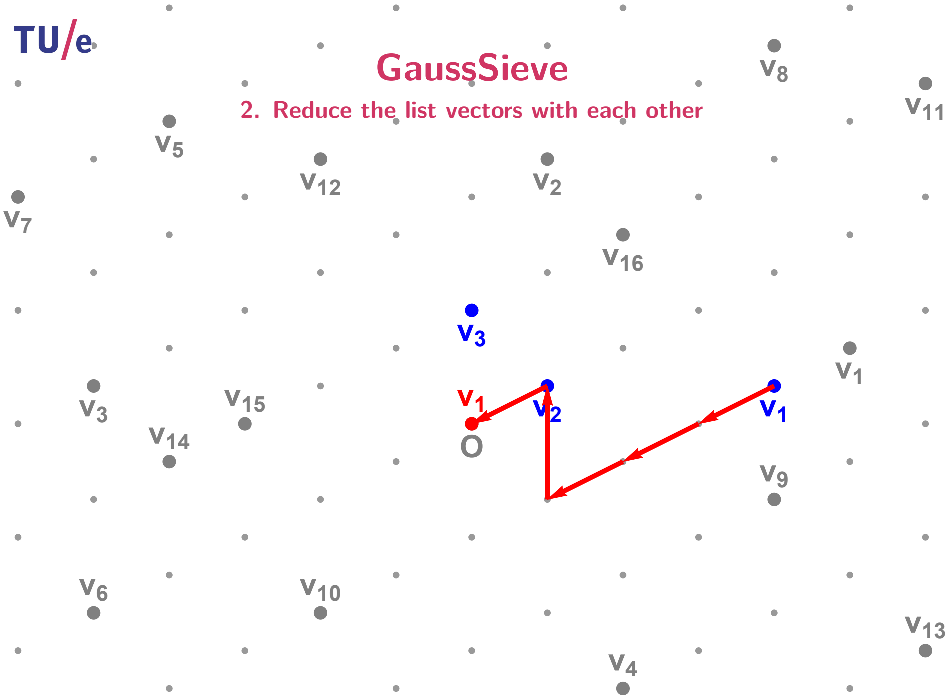
GaussSieve

2. Reduce the list vectors with each other



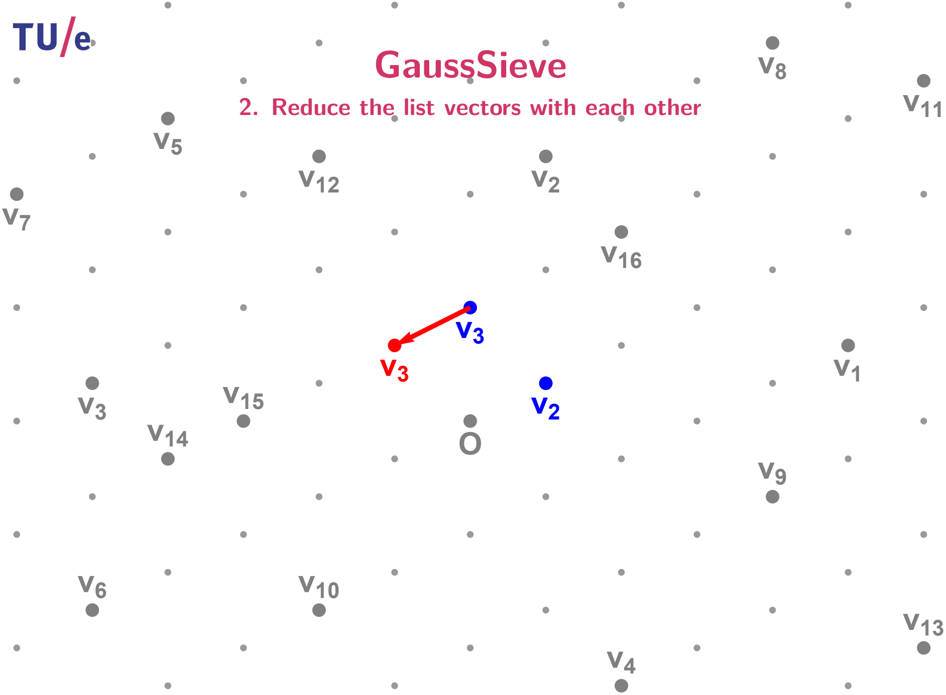
GaussSieve

2. Reduce the list vectors with each other



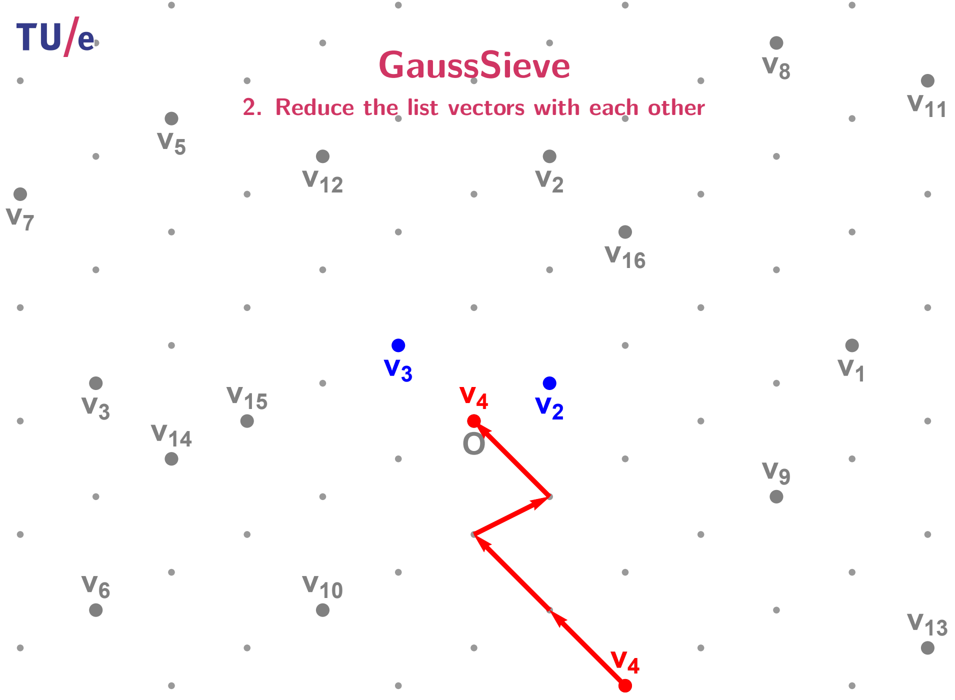
GaussSieve

2. Reduce the list vectors with each other



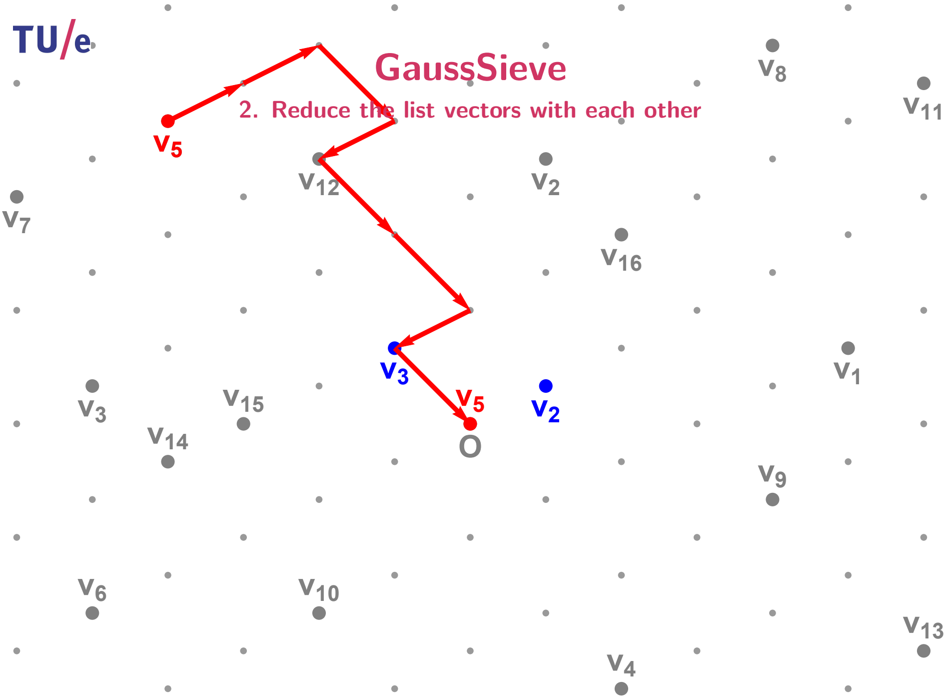
GaussSieve

2. Reduce the list vectors with each other



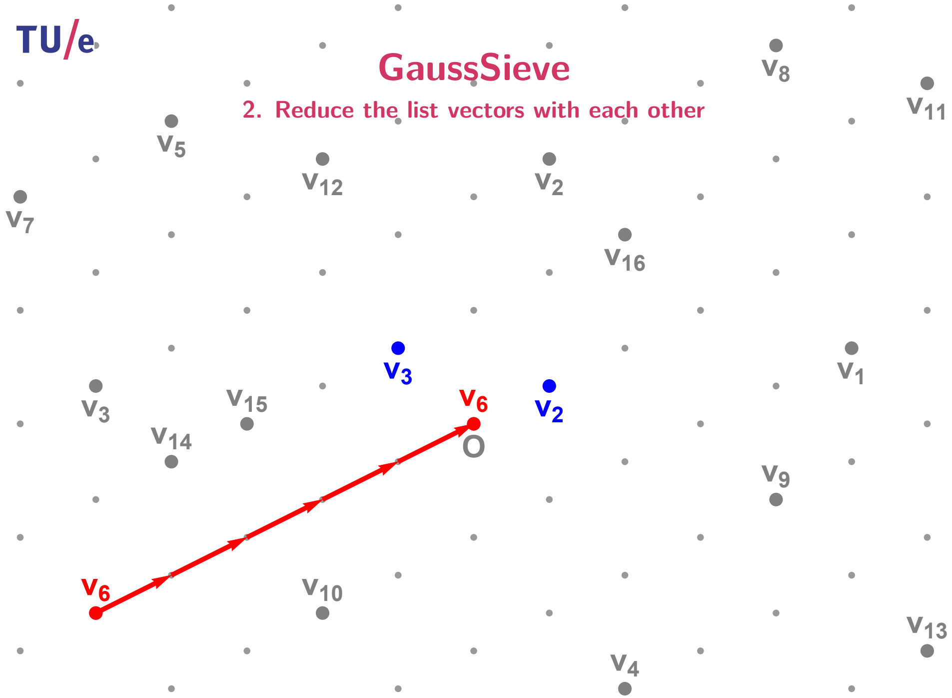
GaussSieve

2. Reduce the list vectors with each other



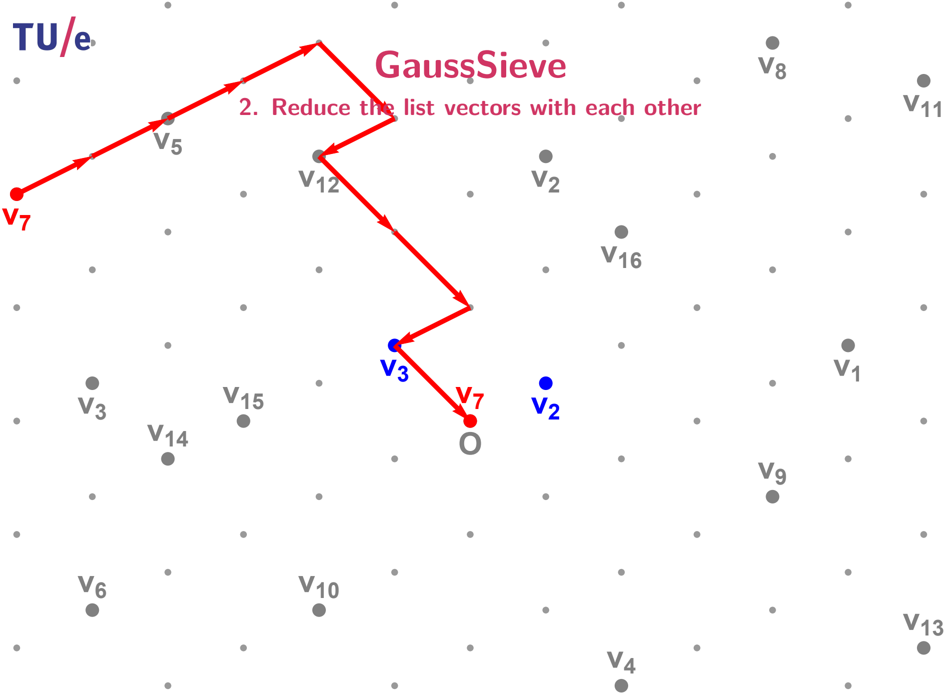
GaussSieve

2. Reduce the list vectors with each other



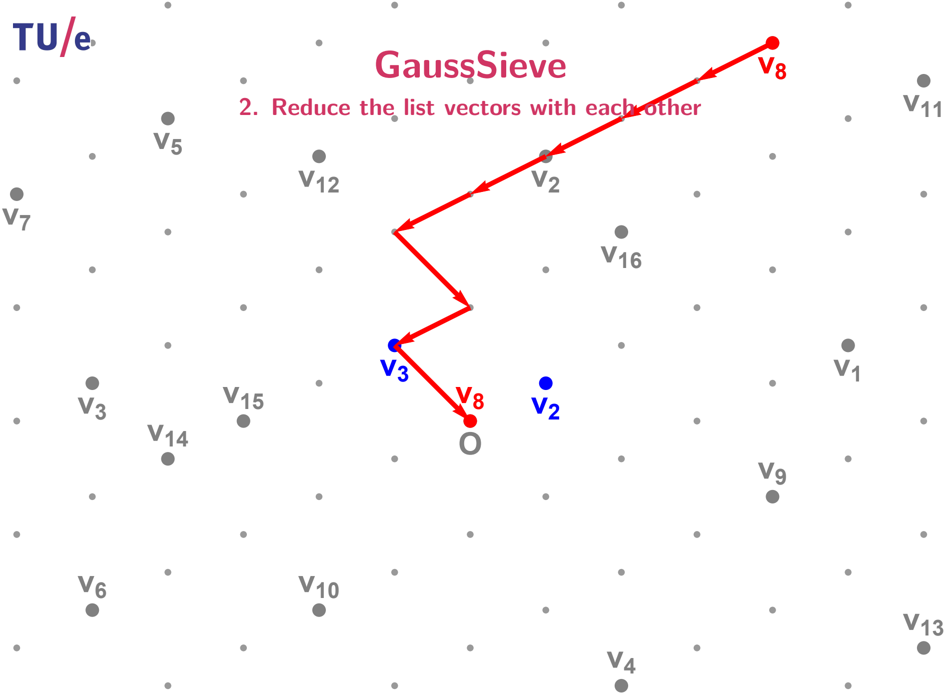
GaussSieve

2. Reduce the list vectors with each other



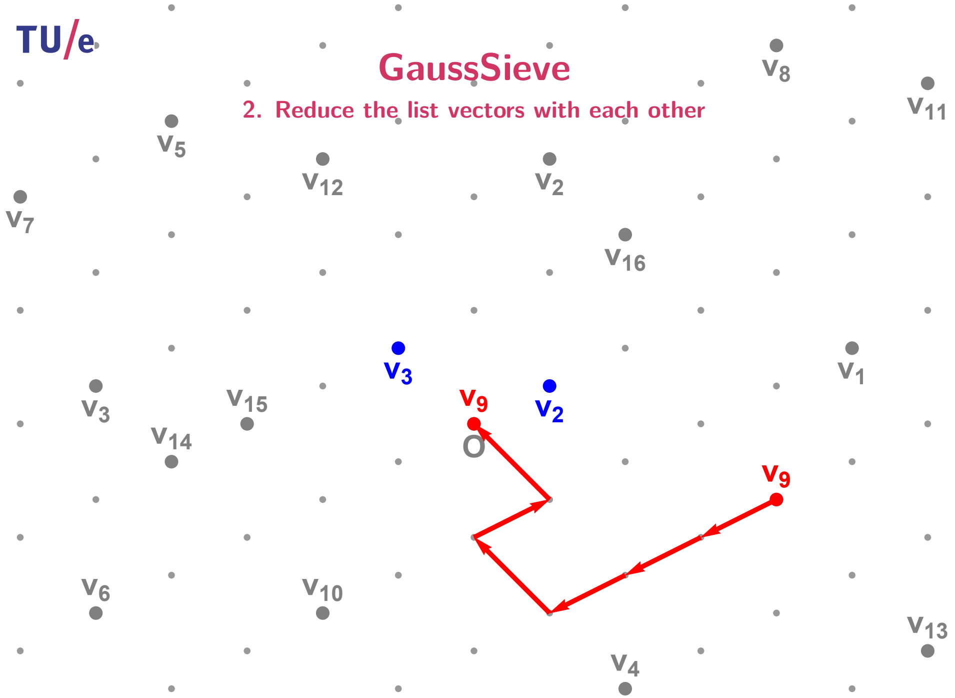
GaussSieve

2. Reduce the list vectors with each other



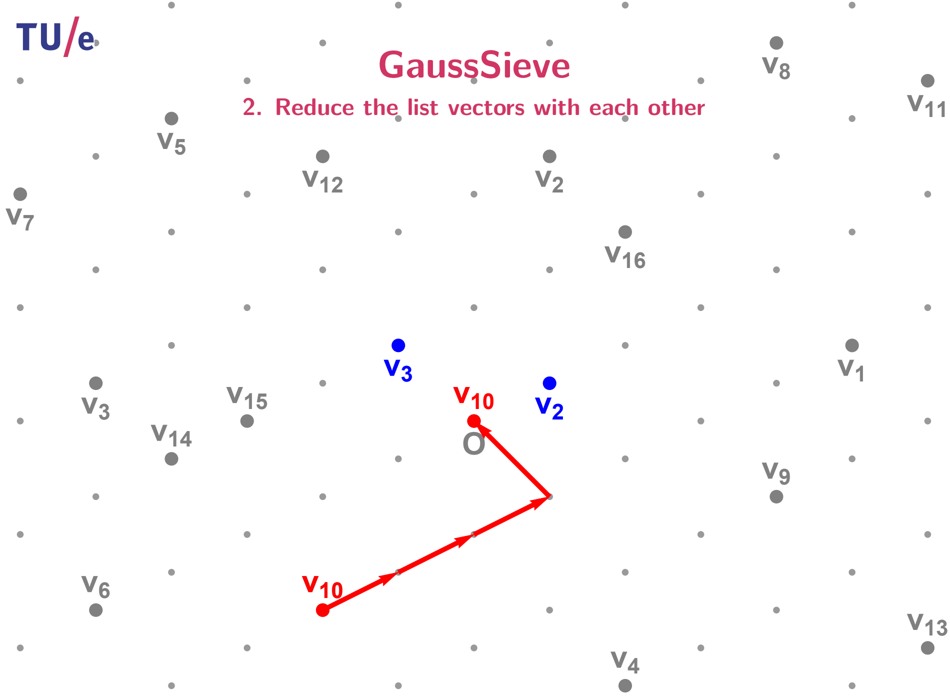
GaussSieve

2. Reduce the list vectors with each other



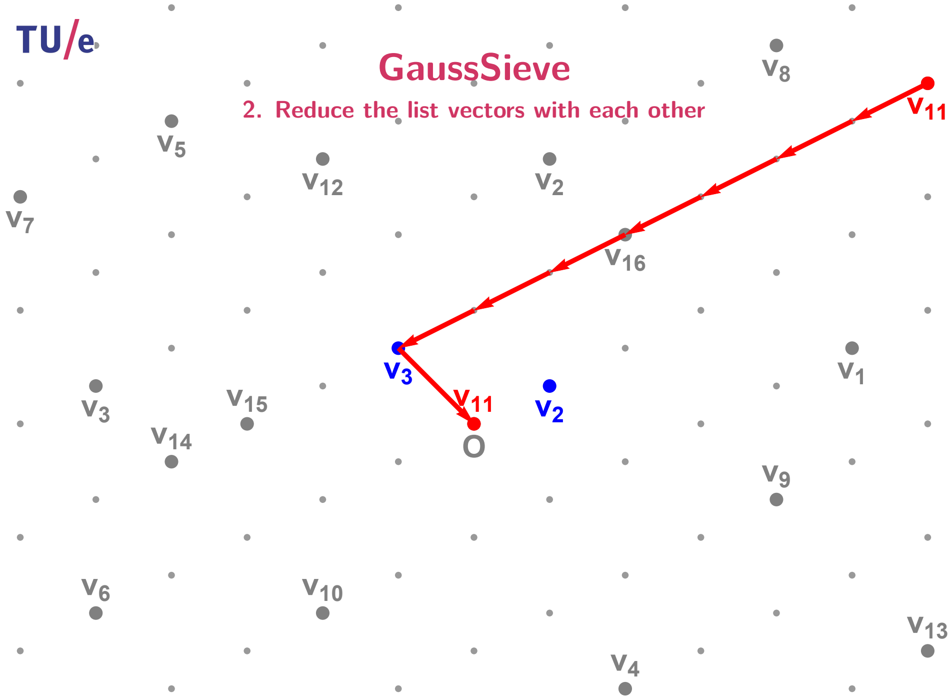
GaussSieve

2. Reduce the list vectors with each other



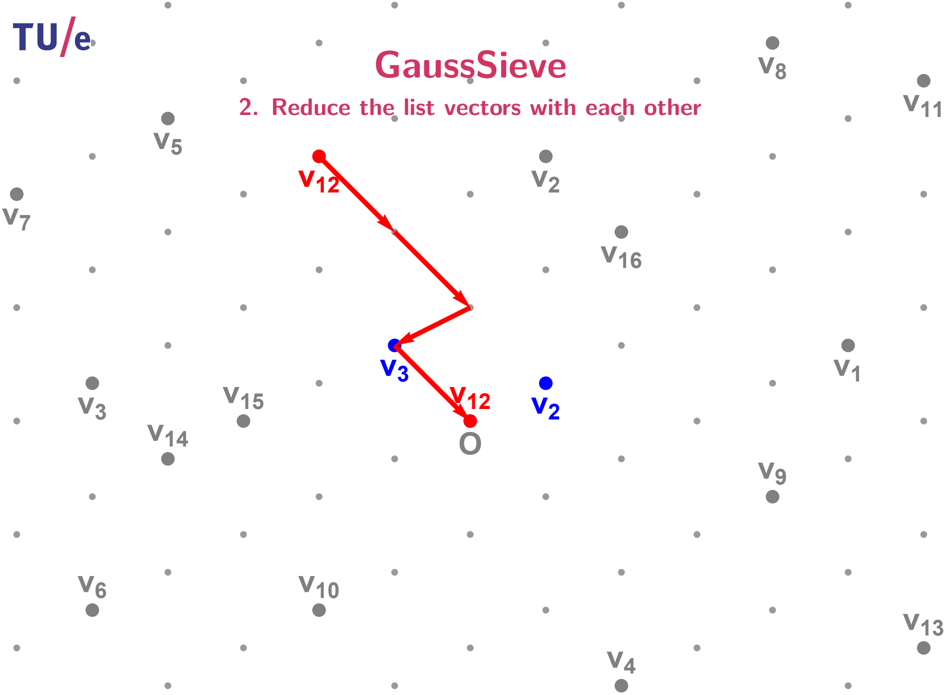
GaussSieve

2. Reduce the list vectors with each other



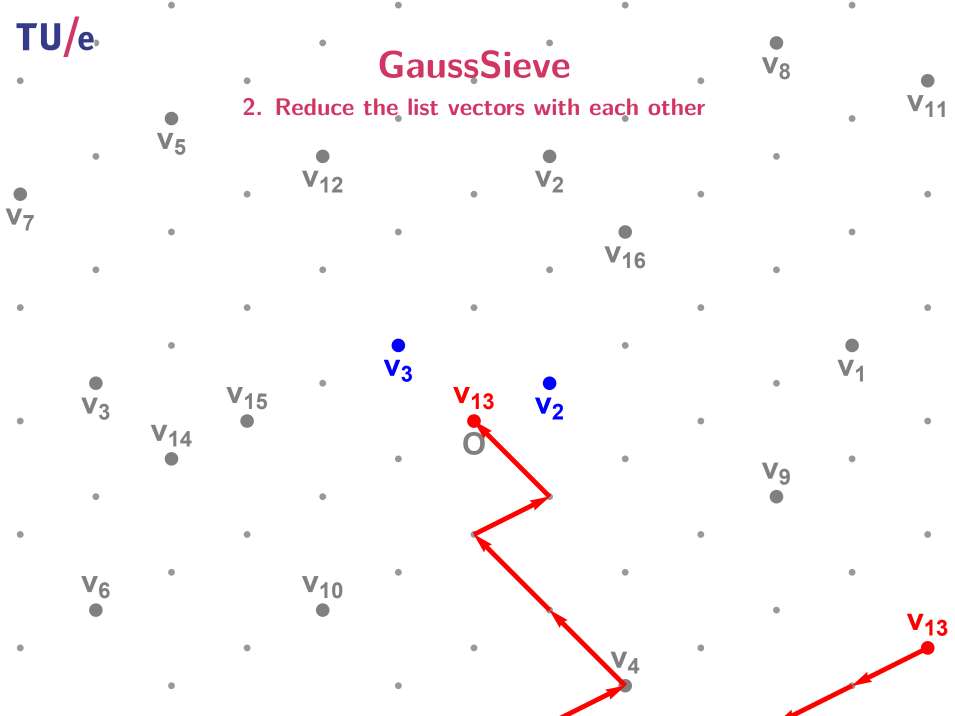
GaussSieve

2. Reduce the list vectors with each other



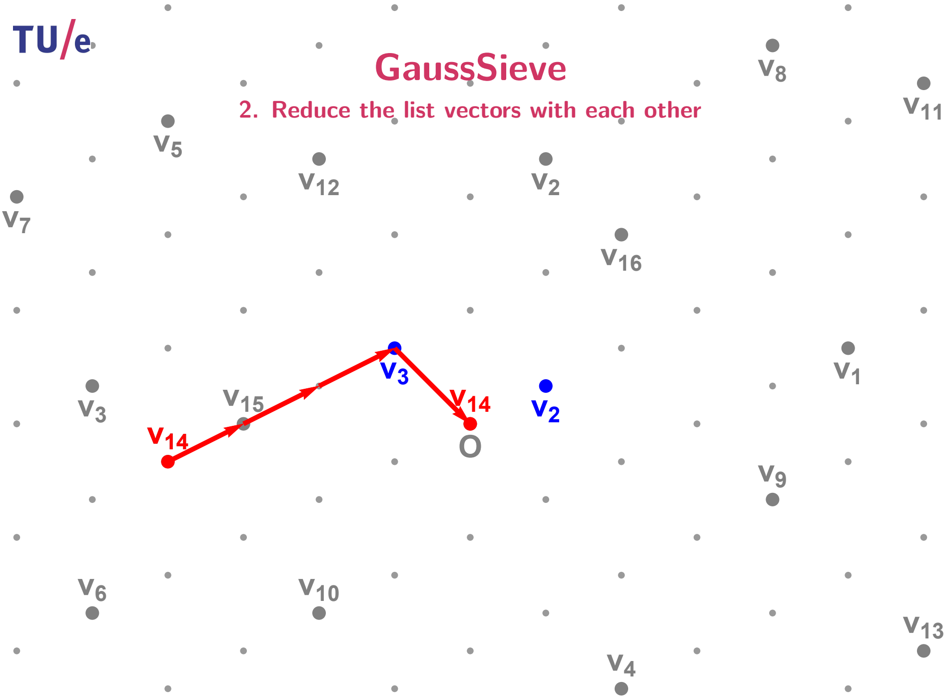
GaussSieve

2. Reduce the list vectors with each other



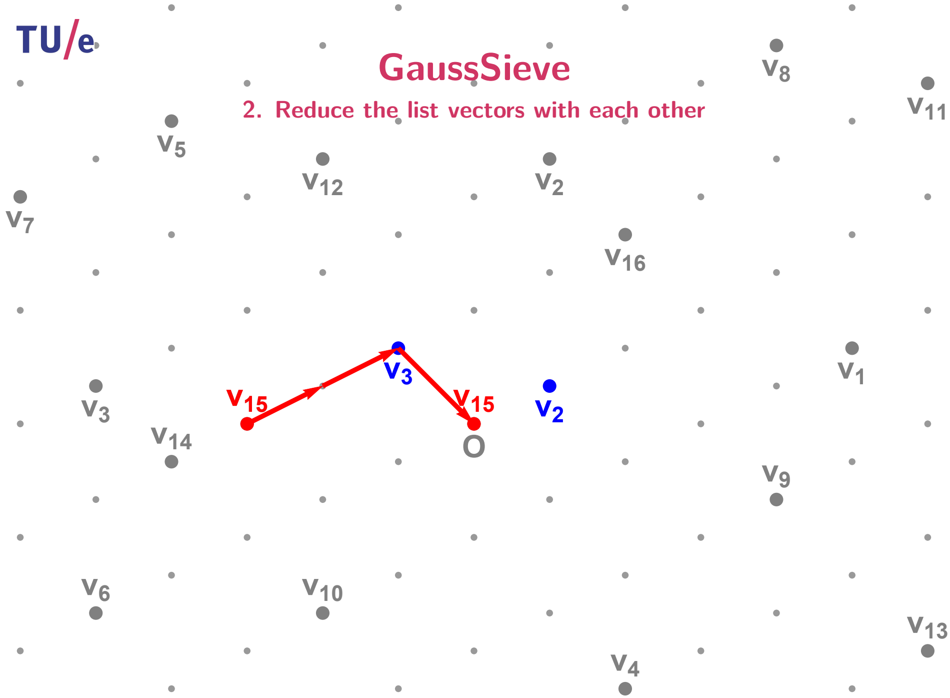
GaussSieve

2. Reduce the list vectors with each other



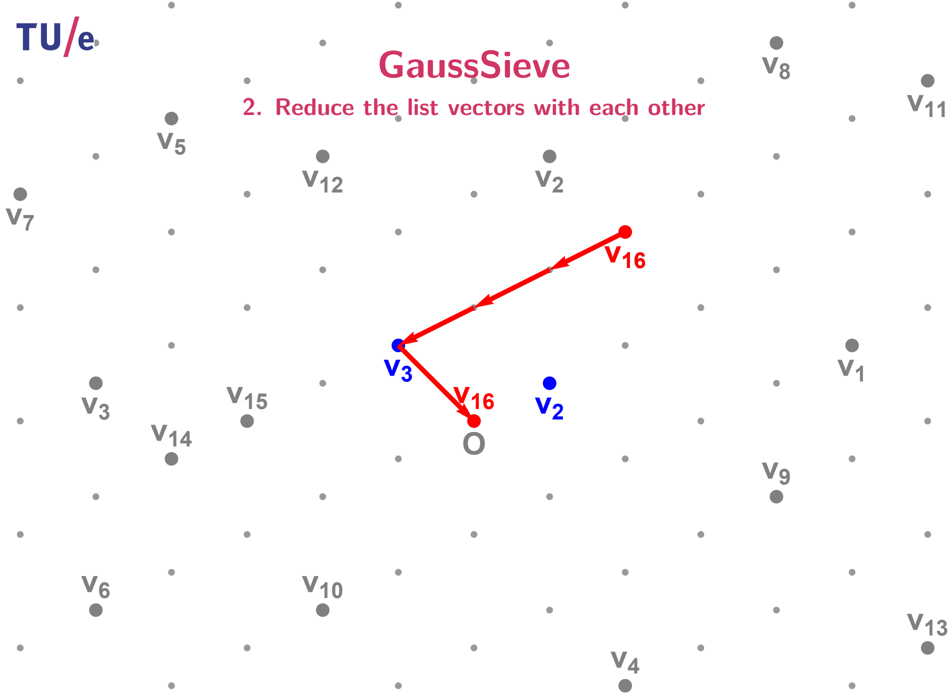
GaussSieve

2. Reduce the list vectors with each other



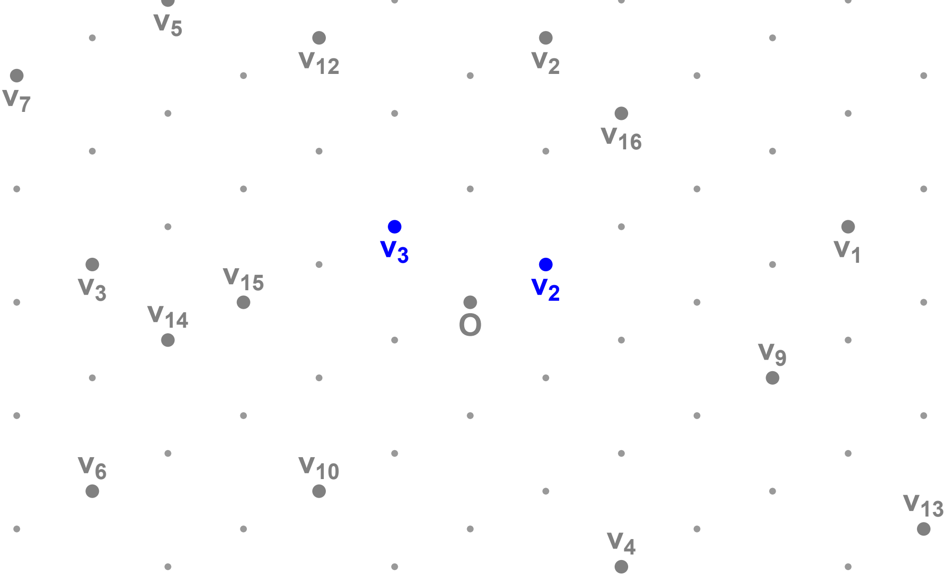
GaussSieve

2. Reduce the list vectors with each other



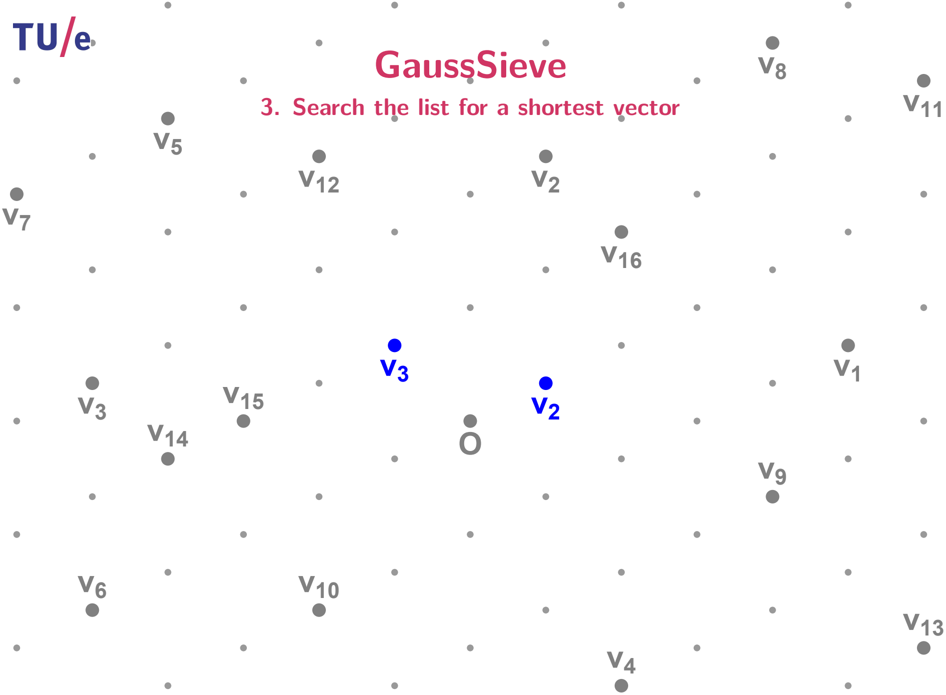
GaussSieve

2. Reduce the list vectors with each other



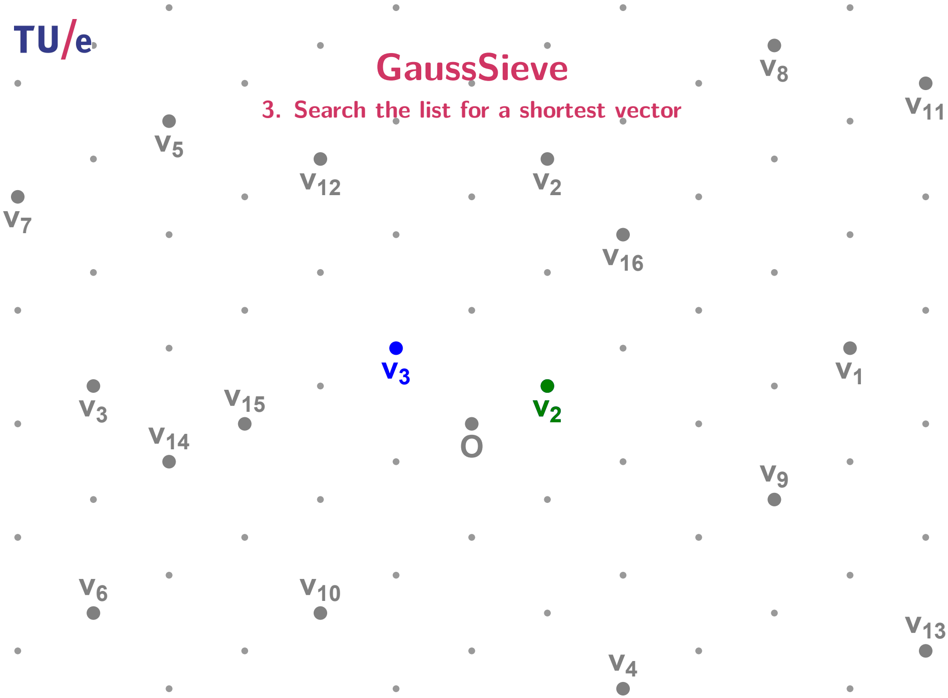
GaussSieve

3. Search the list for a shortest vector



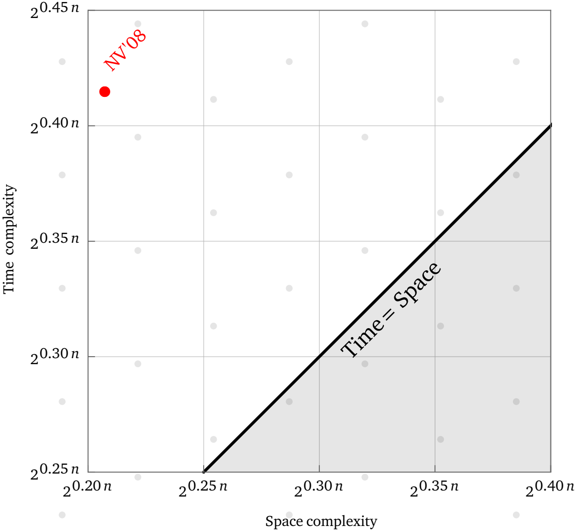
GaussSieve

3. Search the list for a shortest vector



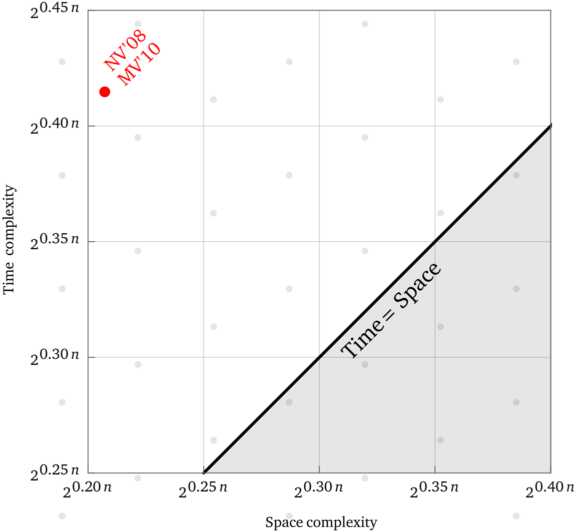
GaussSieve

Space/time trade-off



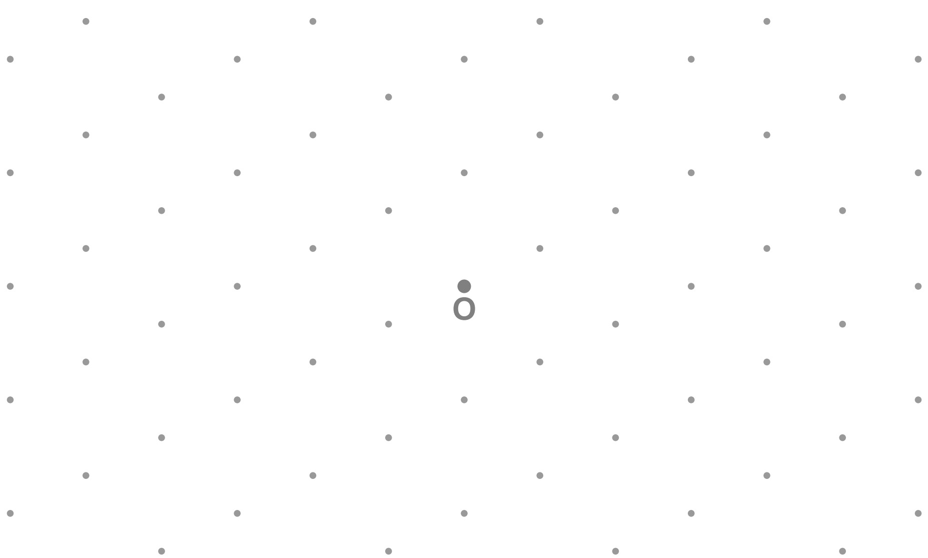
GaussSieve

Space/time trade-off



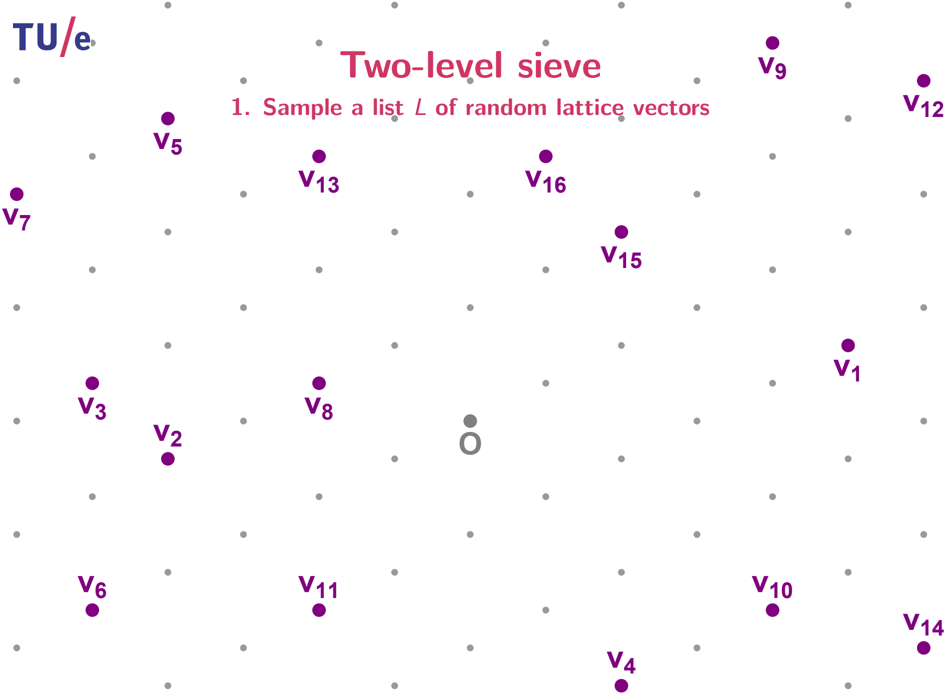
Two-level sieve

1. Sample a list L of random lattice vectors



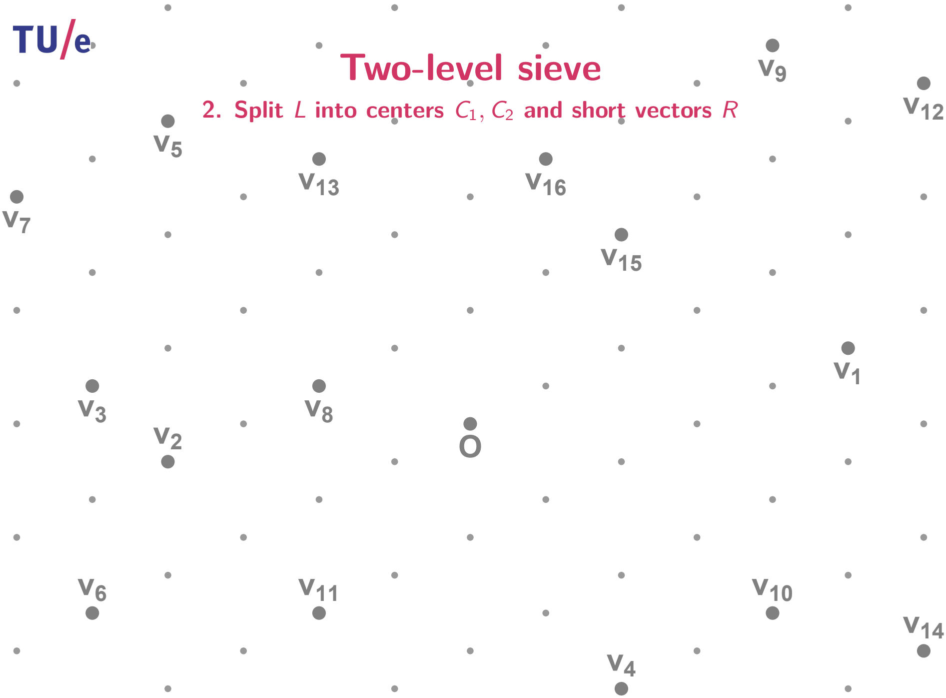
Two-level sieve

1. Sample a list L of random lattice vectors



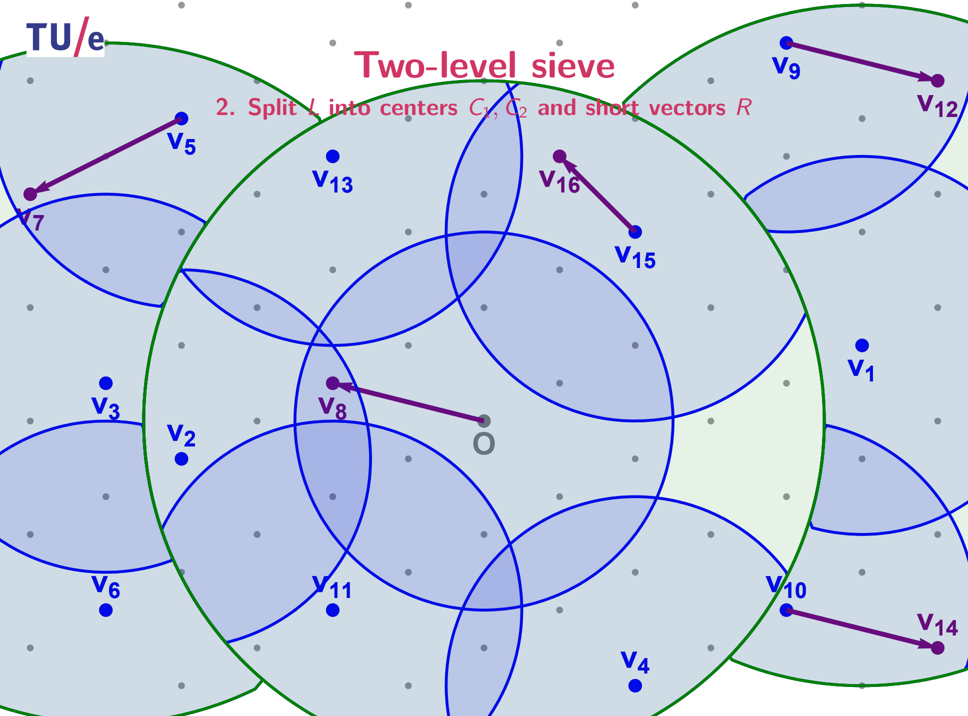
Two-level sieve

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



Two-level sieve

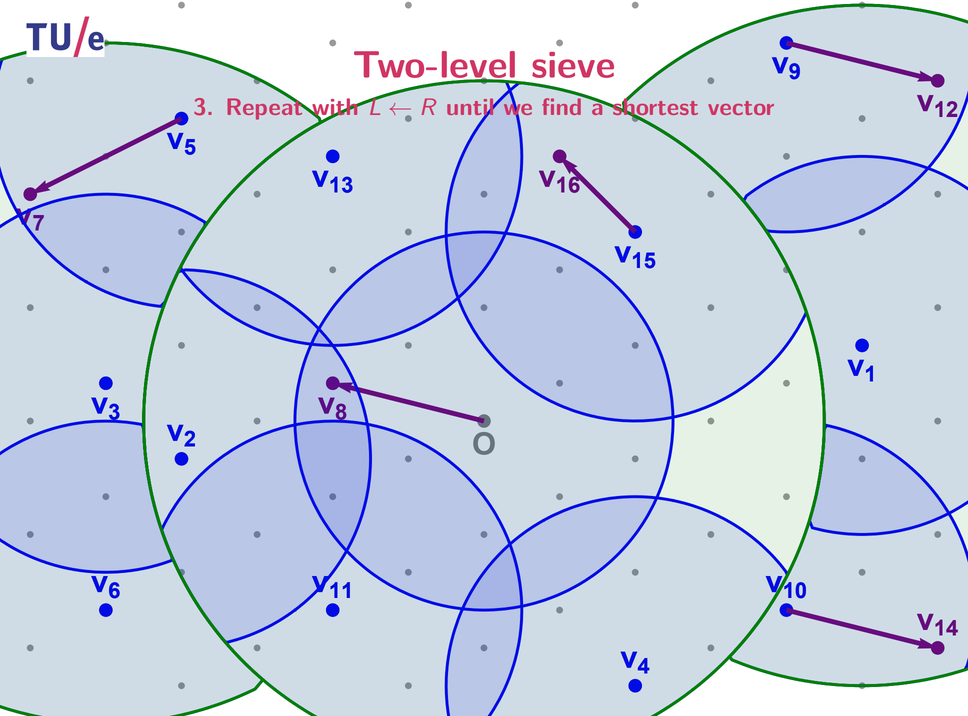
2. Split L into centers C_1, C_2 and short vectors R



TU/e

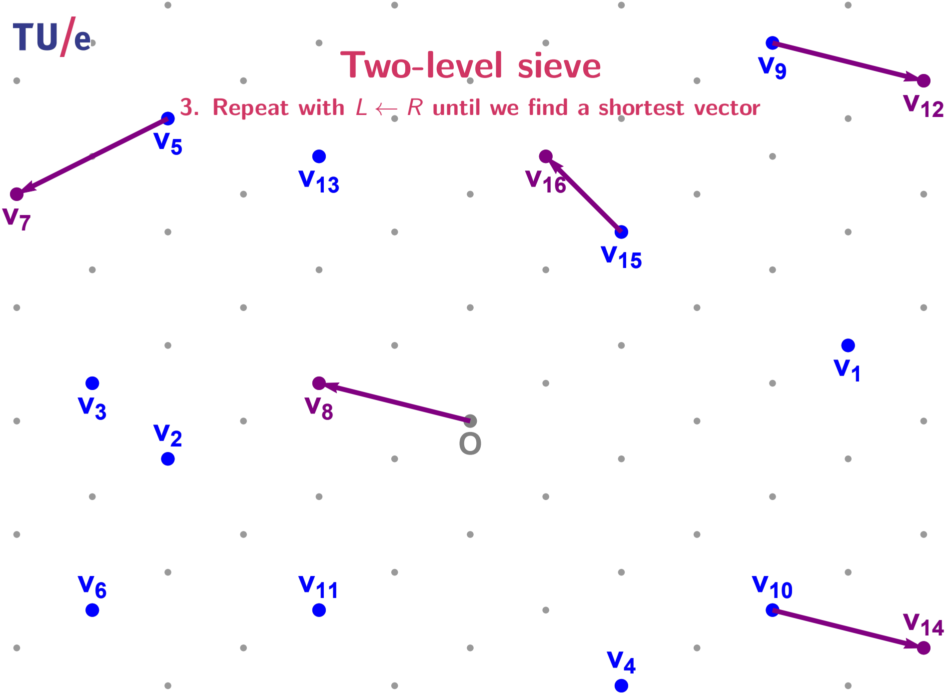
Two-level sieve

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



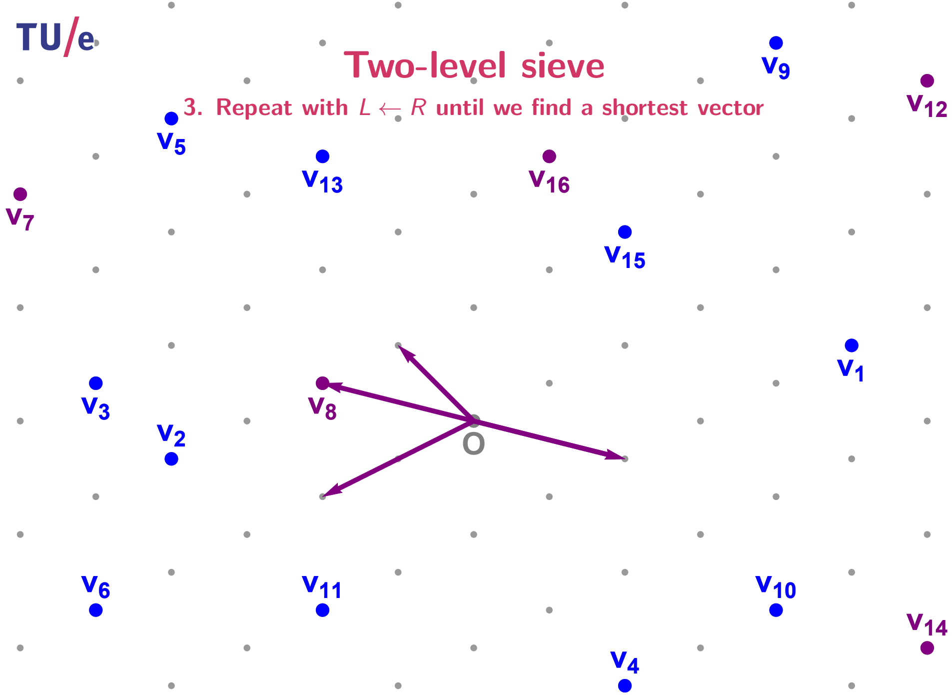
Two-level sieve

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



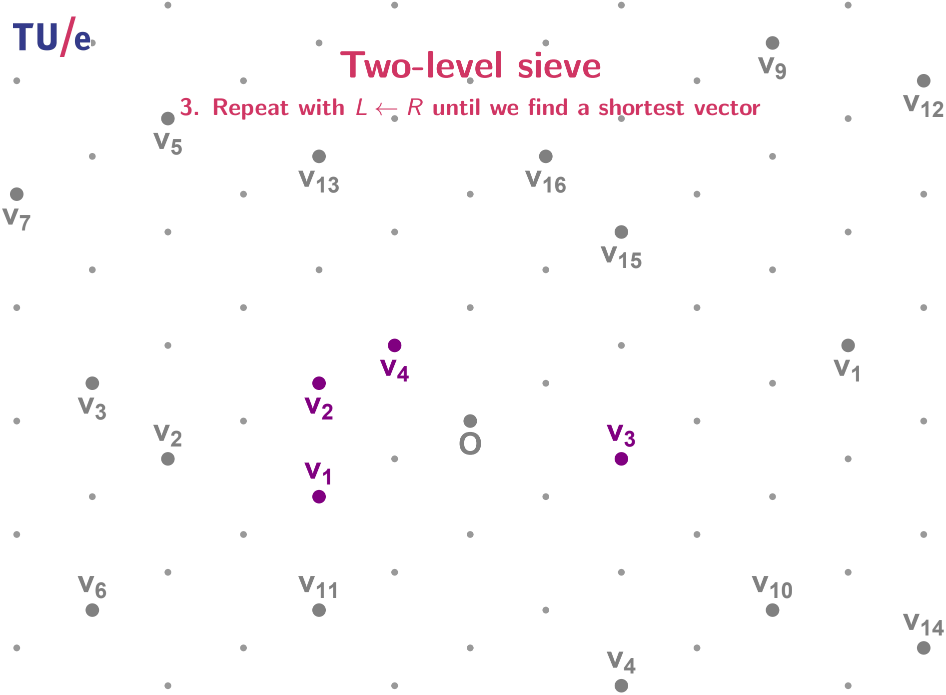
Two-level sieve

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



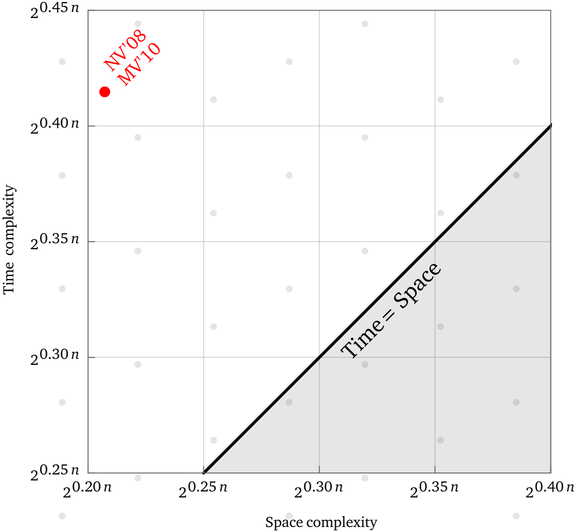
Two-level sieve

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



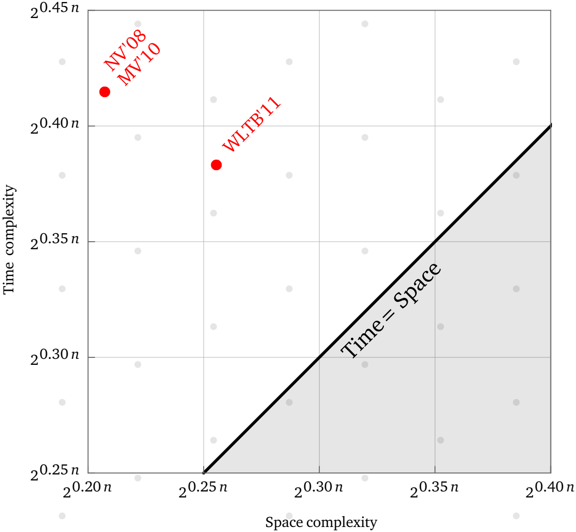
Two-level sieve

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 $2^{0.2557n}$.

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)

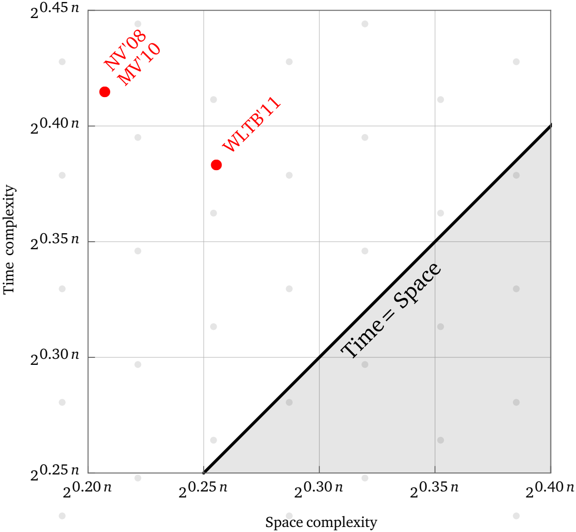
The three-level sieve runs in time $2^{0.3778n}$ and space $2^{0.2833n}$.

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.

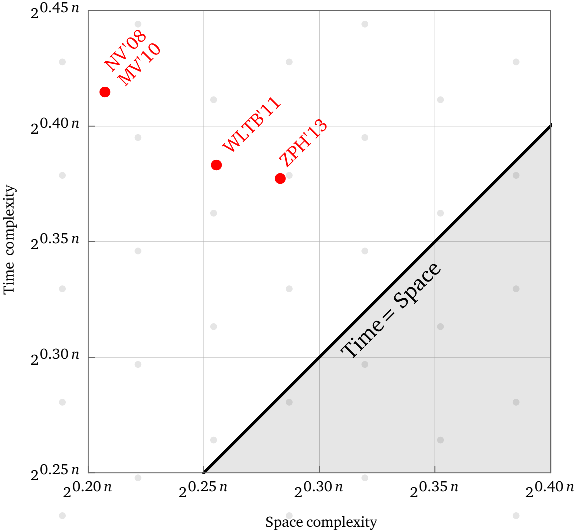
Three-level sieve

Space/time trade-off



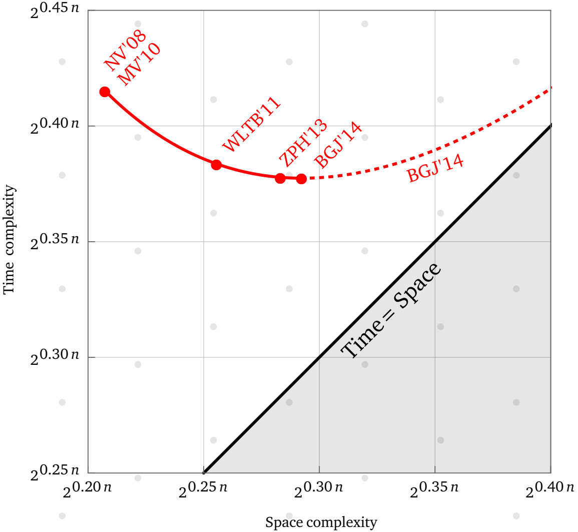
Three-level sieve

Space/time trade-off



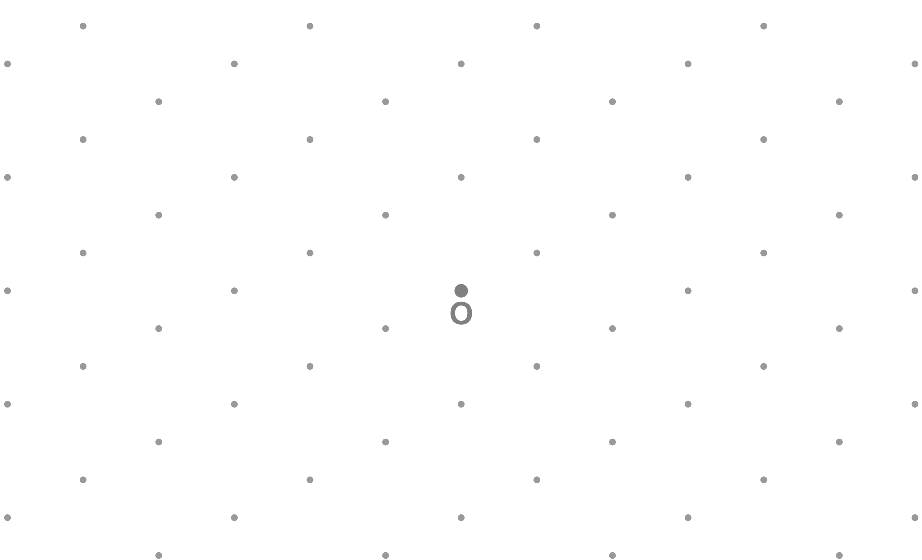
Decomposition approach

Space/time trade-off



HashSieve (GS)

1. Generate random lattice vectors



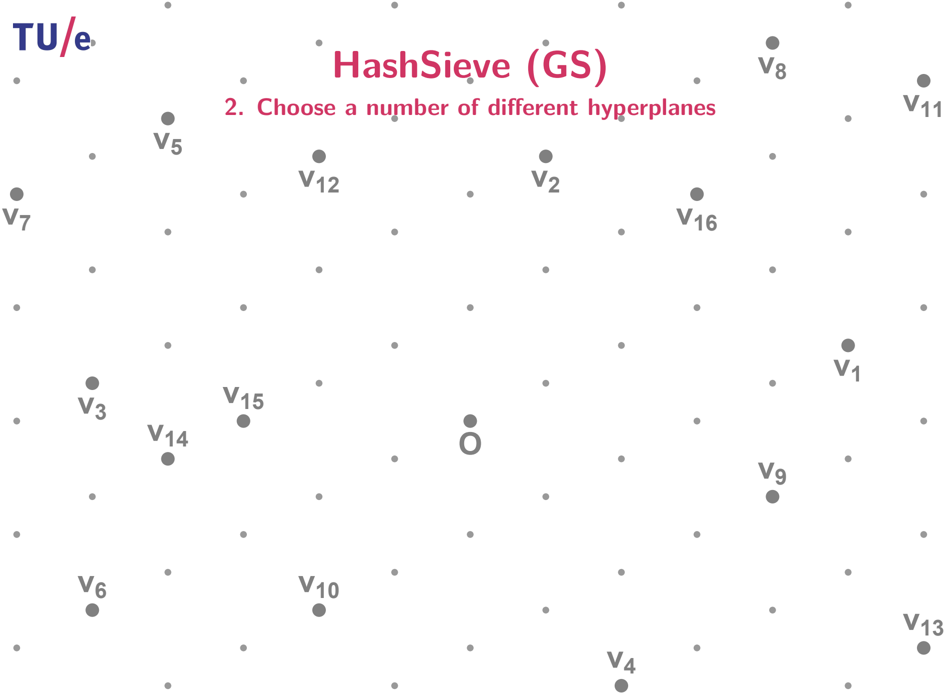
TU/e

HashSieve (GS)

1. Generate random lattice vectors

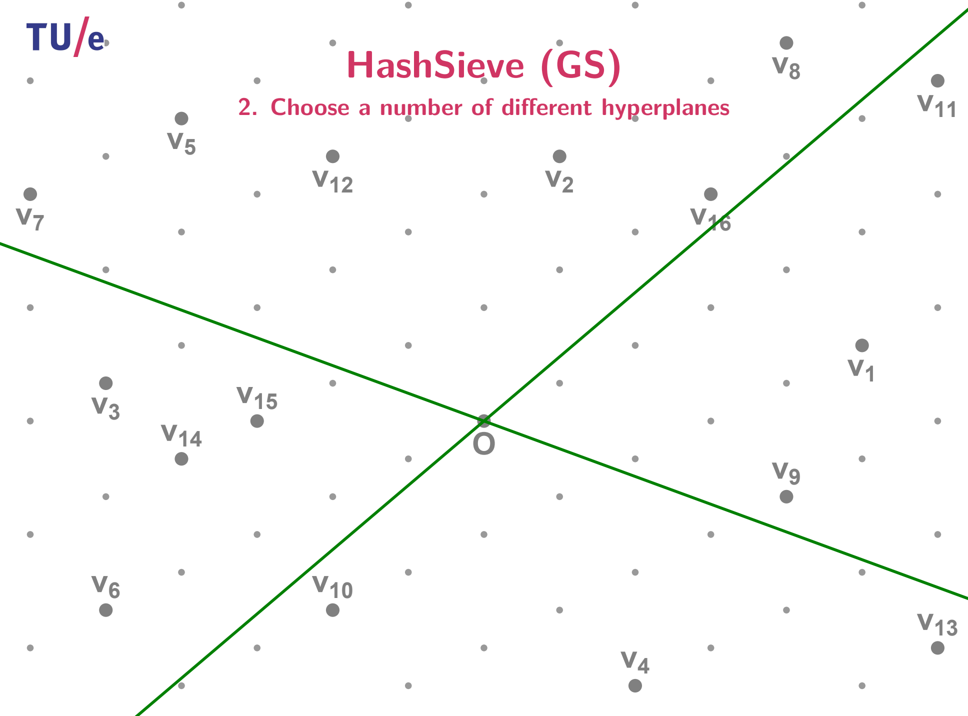
HashSieve (GS)

2. Choose a number of different hyperplanes



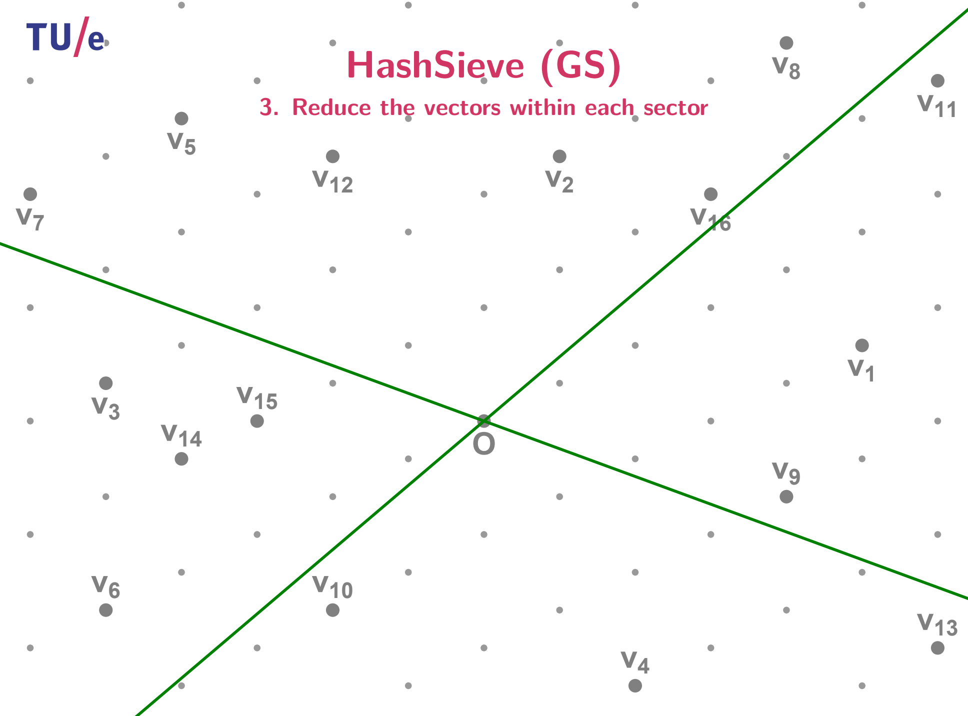
HashSieve (GS)

2. Choose a number of different hyperplanes



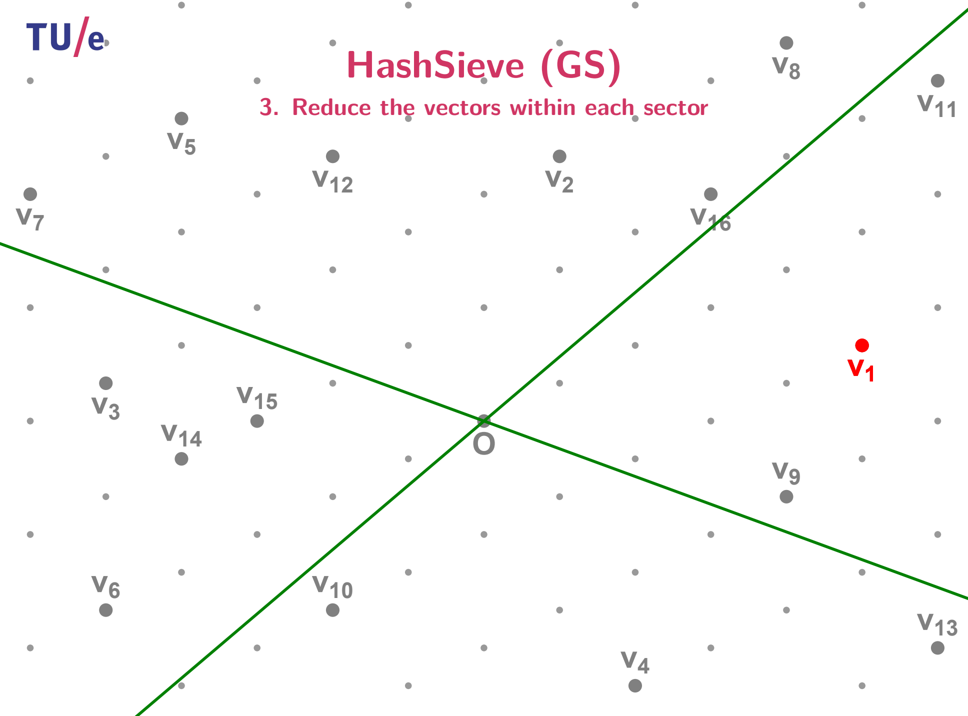
HashSieve (GS)

3. Reduce the vectors within each sector



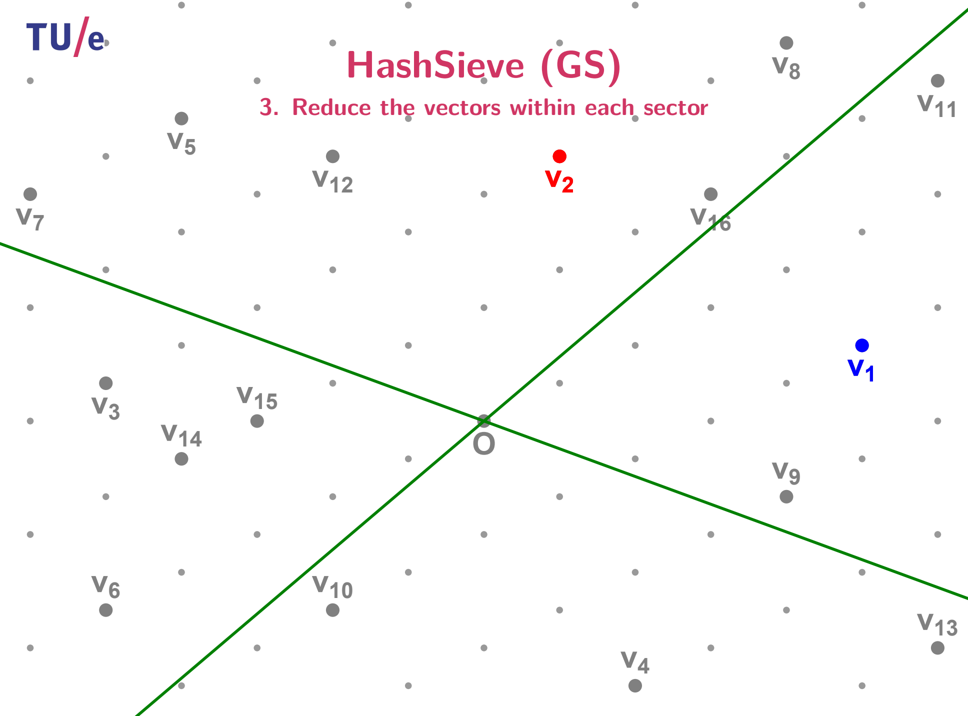
HashSieve (GS)

3. Reduce the vectors within each sector



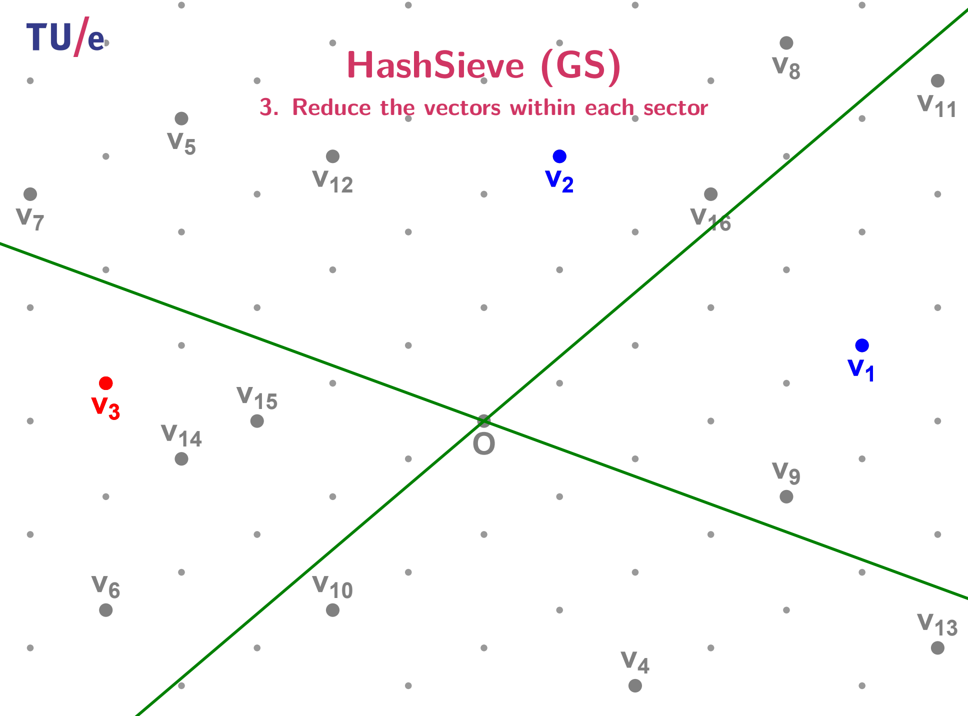
HashSieve (GS)

3. Reduce the vectors within each sector



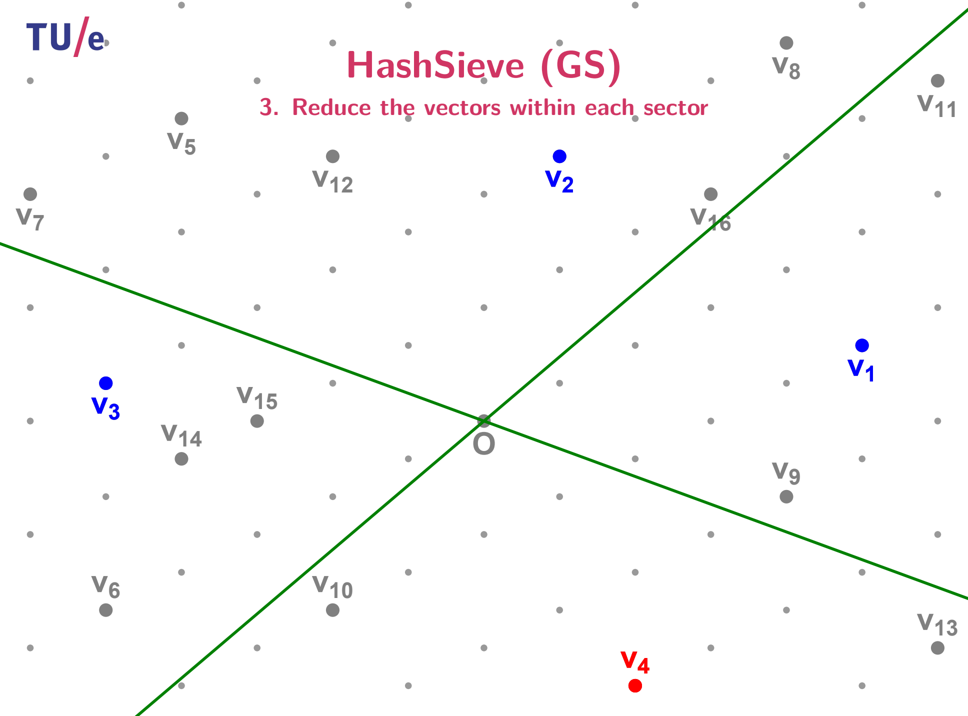
HashSieve (GS)

3. Reduce the vectors within each sector



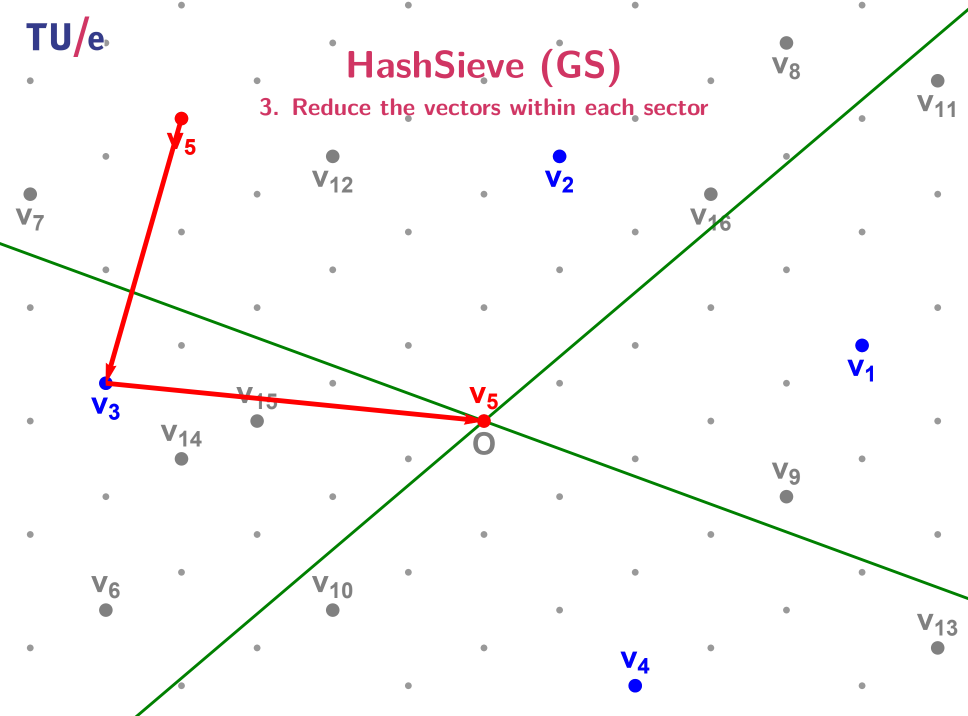
HashSieve (GS)

3. Reduce the vectors within each sector



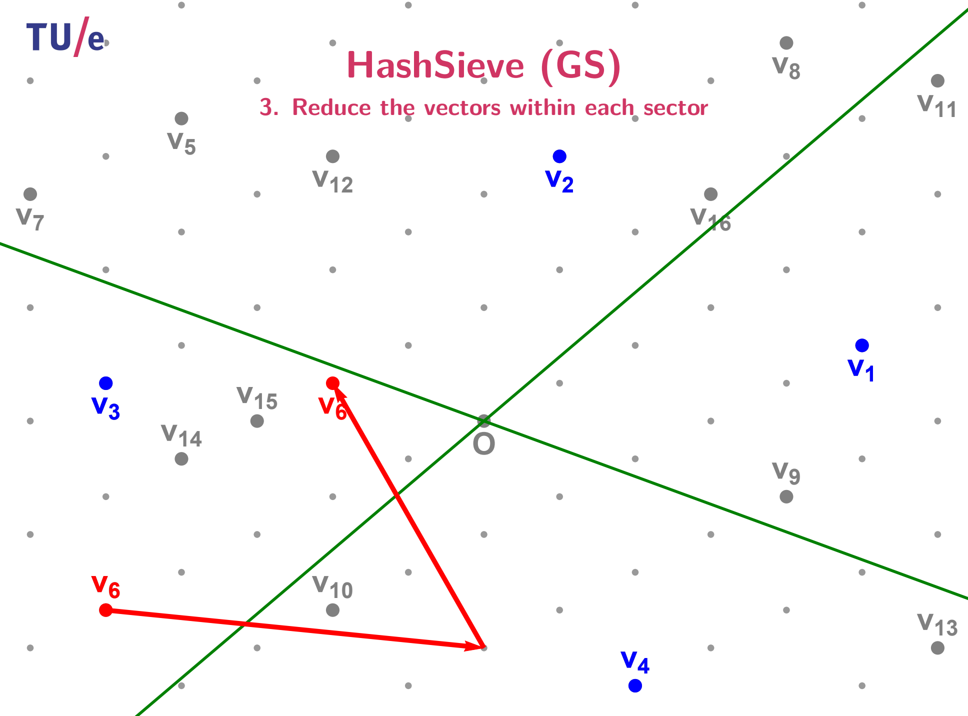
HashSieve (GS)

3. Reduce the vectors within each sector



HashSieve (GS)

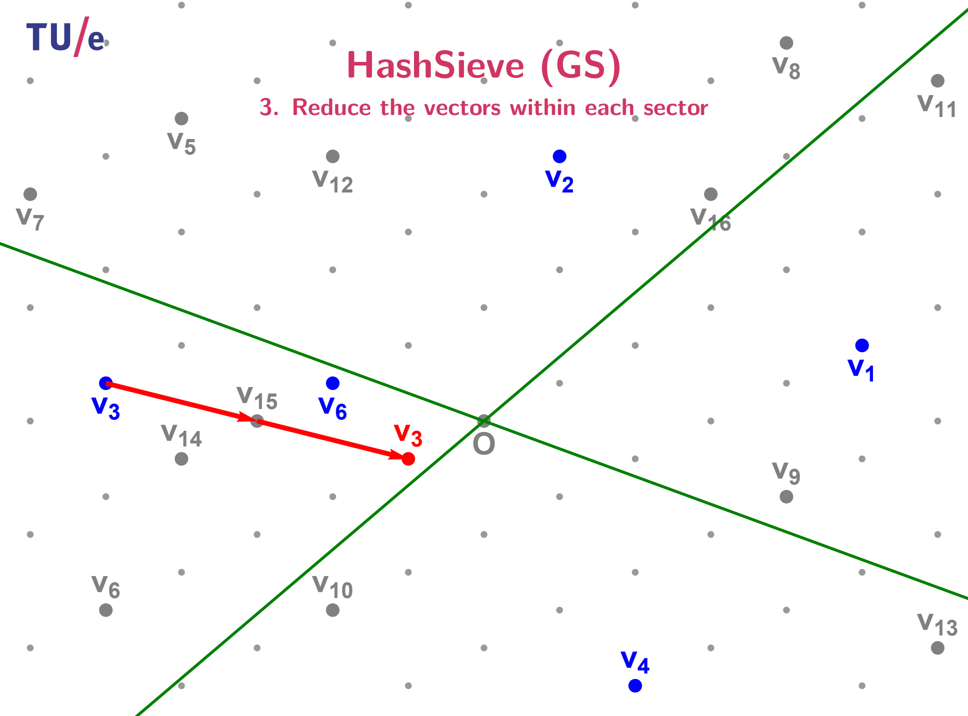
3. Reduce the vectors within each sector



TU/e

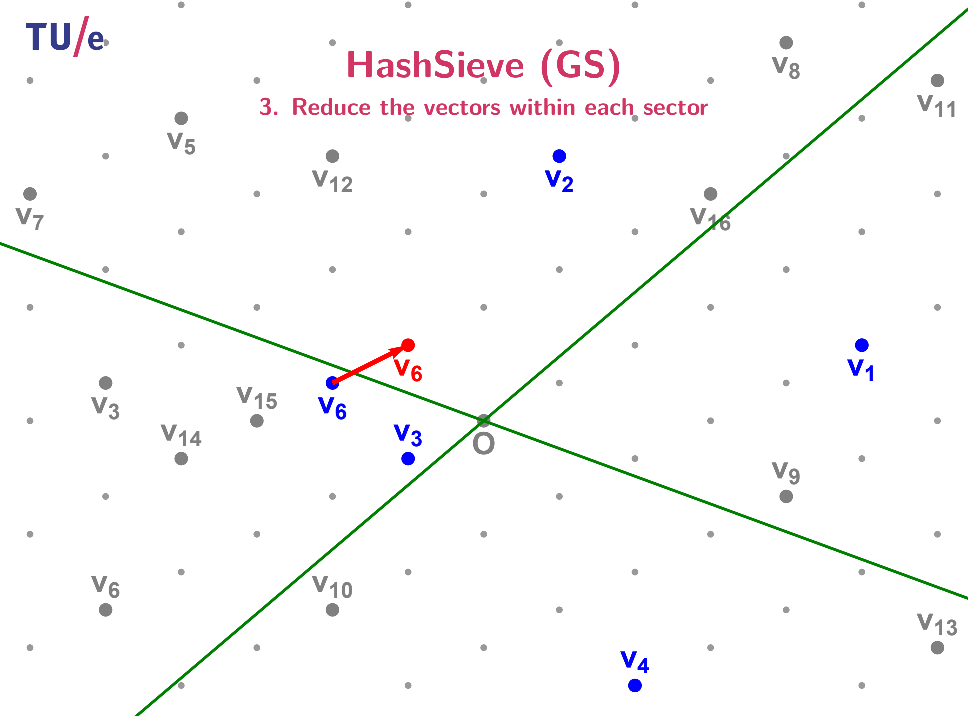
HashSieve (GS)

3. Reduce the vectors within each sector



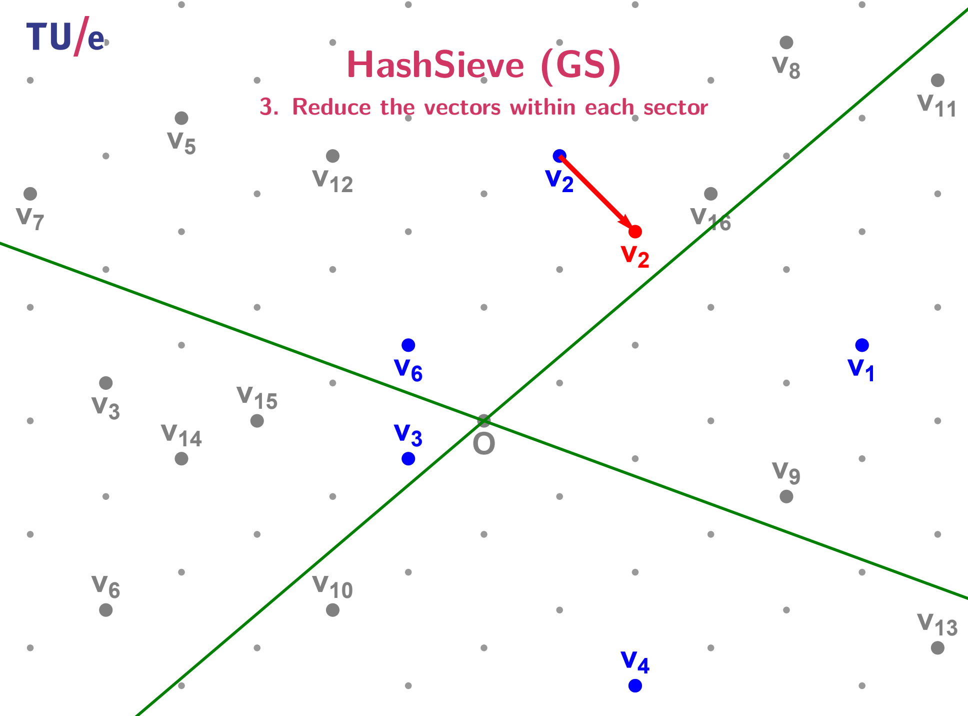
HashSieve (GS)

3. Reduce the vectors within each sector



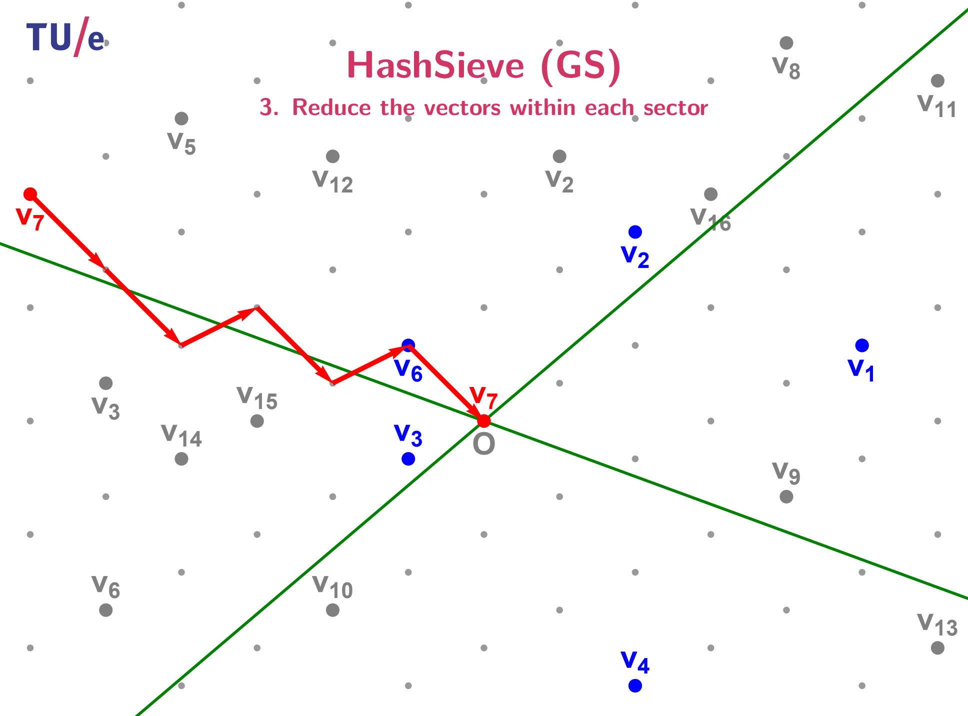
HashSieve (GS)

3. Reduce the vectors within each sector



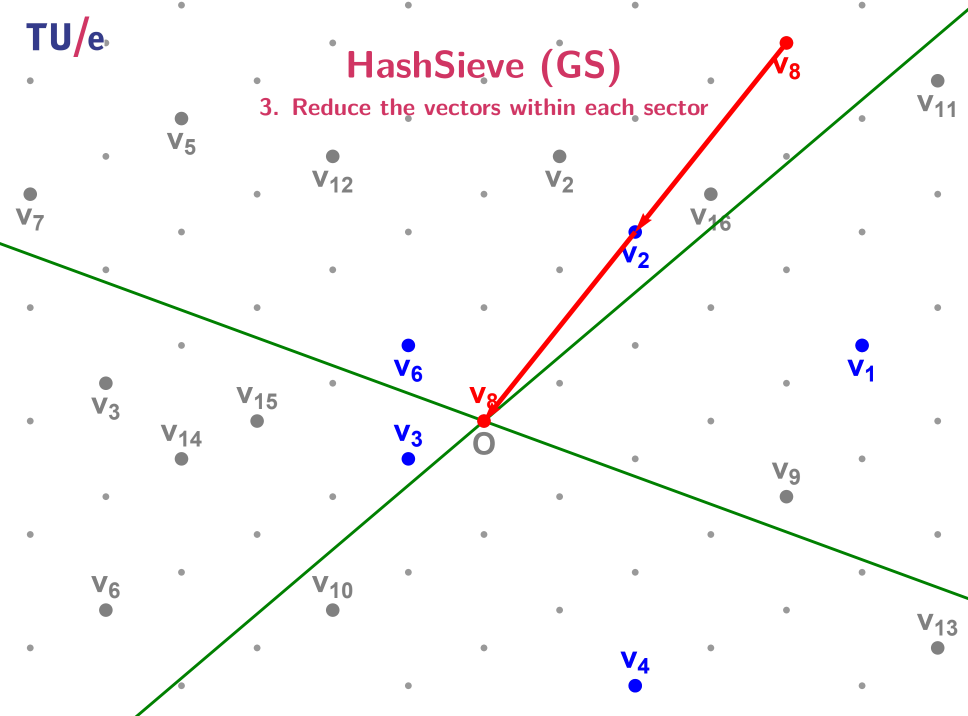
HashSieve (GS)

3. Reduce the vectors within each sector



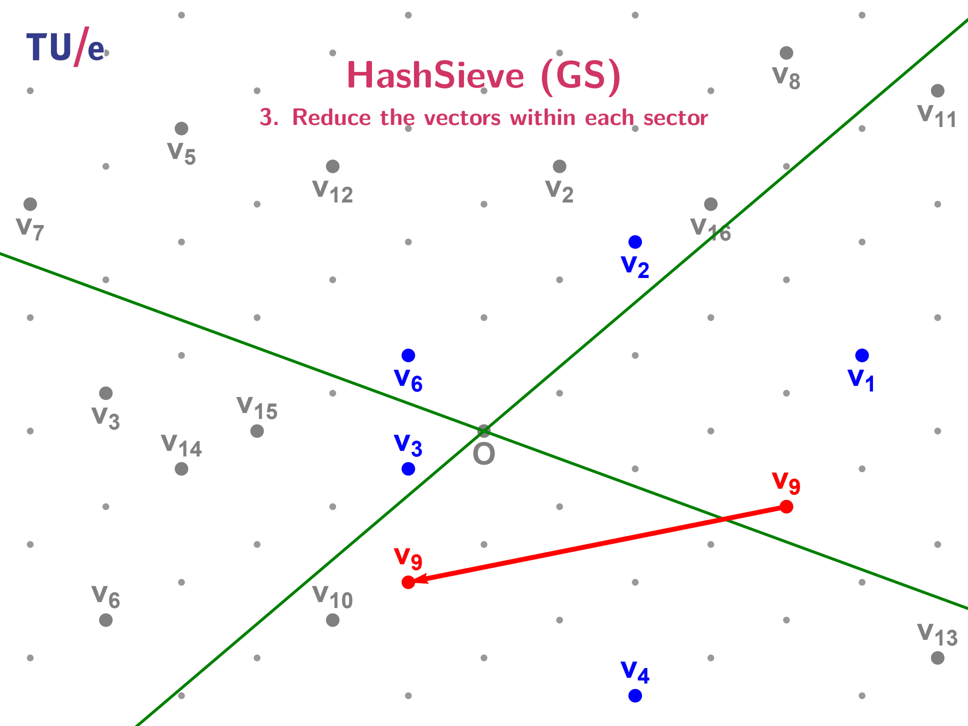
HashSieve (GS)

3. Reduce the vectors within each sector



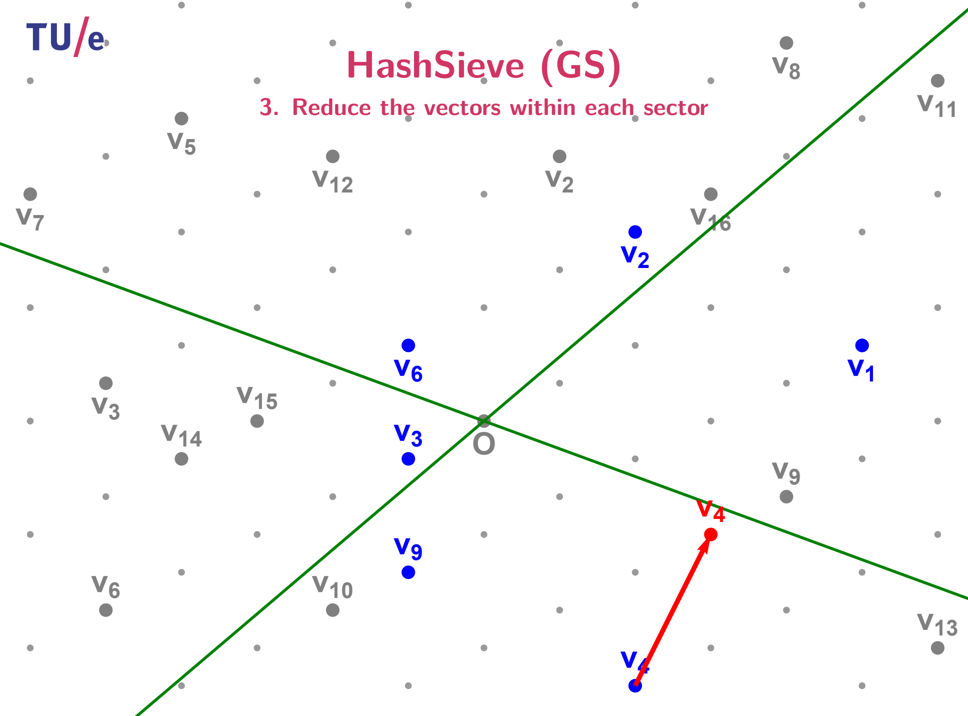
HashSieve (GS)

3. Reduce the vectors within each sector



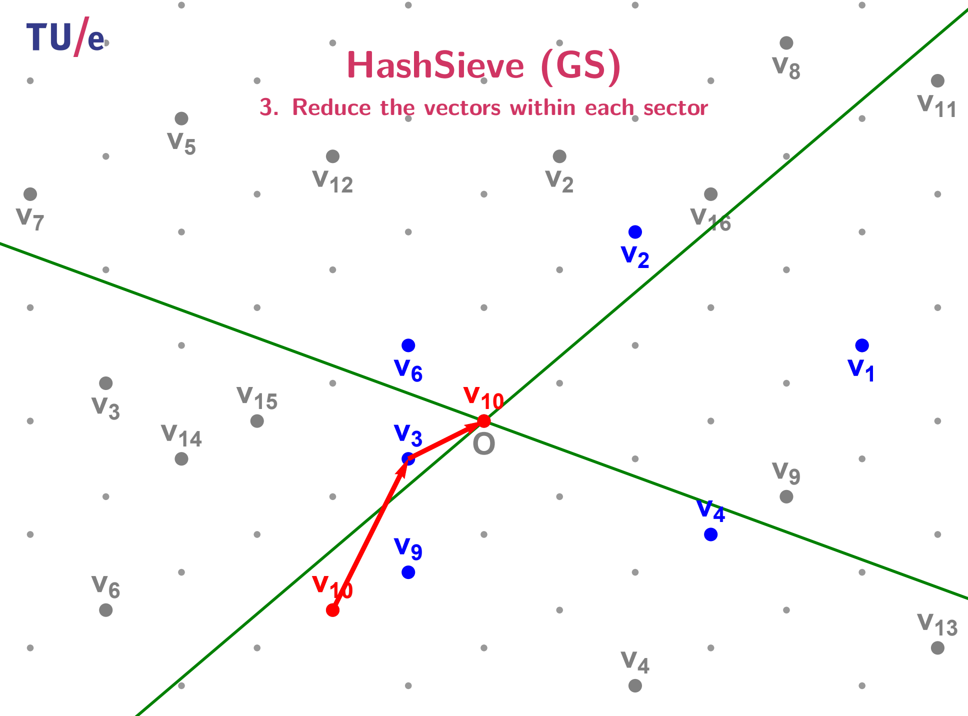
HashSieve (GS)

3. Reduce the vectors within each sector



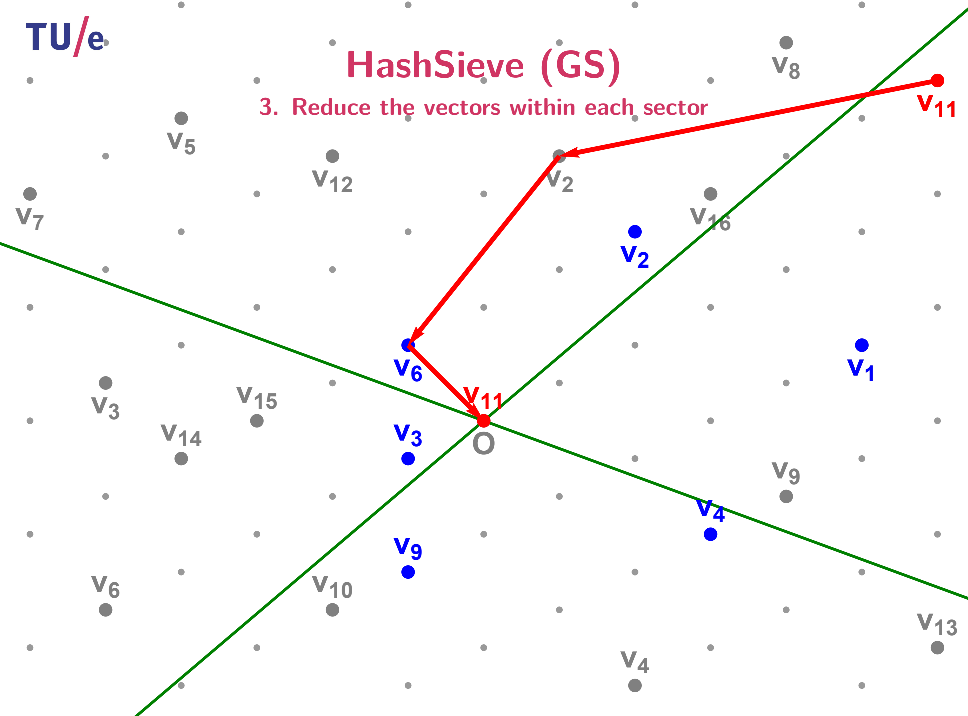
HashSieve (GS)

3. Reduce the vectors within each sector



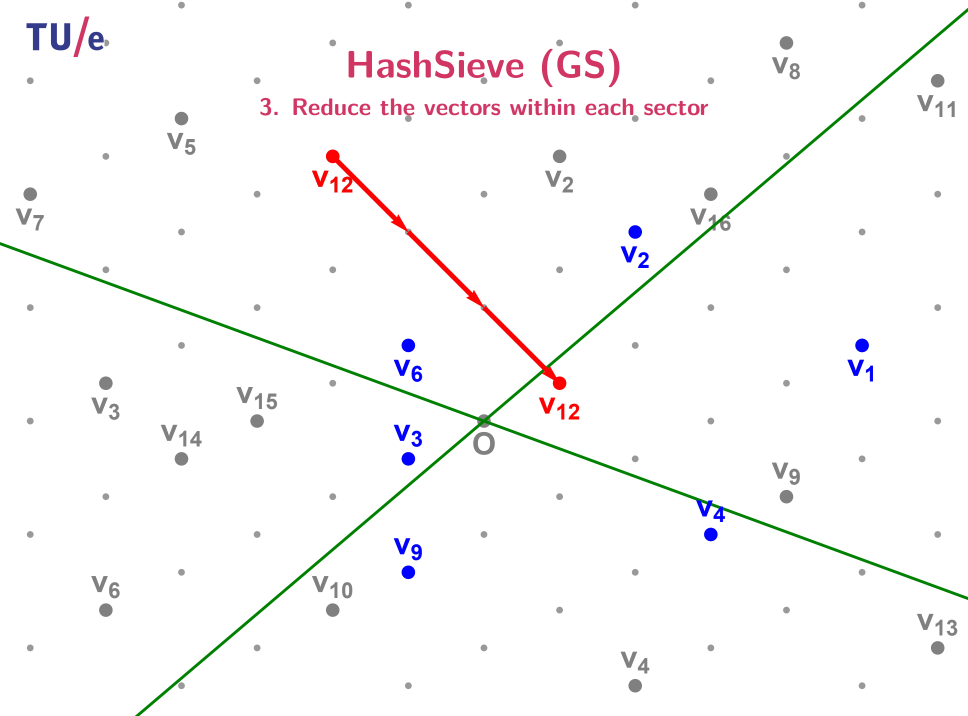
HashSieve (GS)

3. Reduce the vectors within each sector



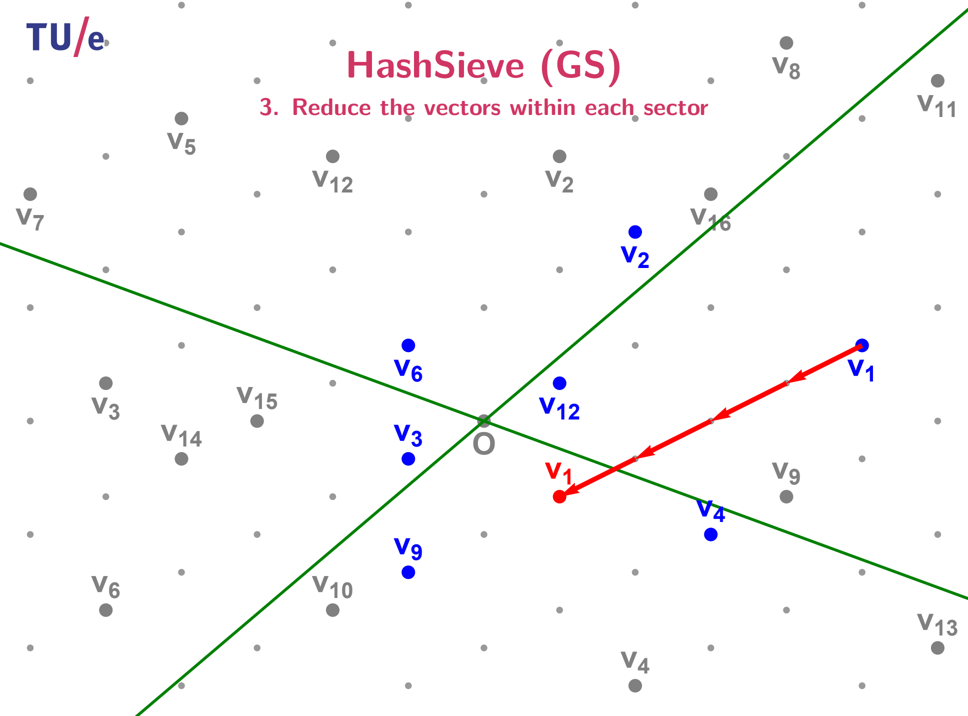
HashSieve (GS)

3. Reduce the vectors within each sector



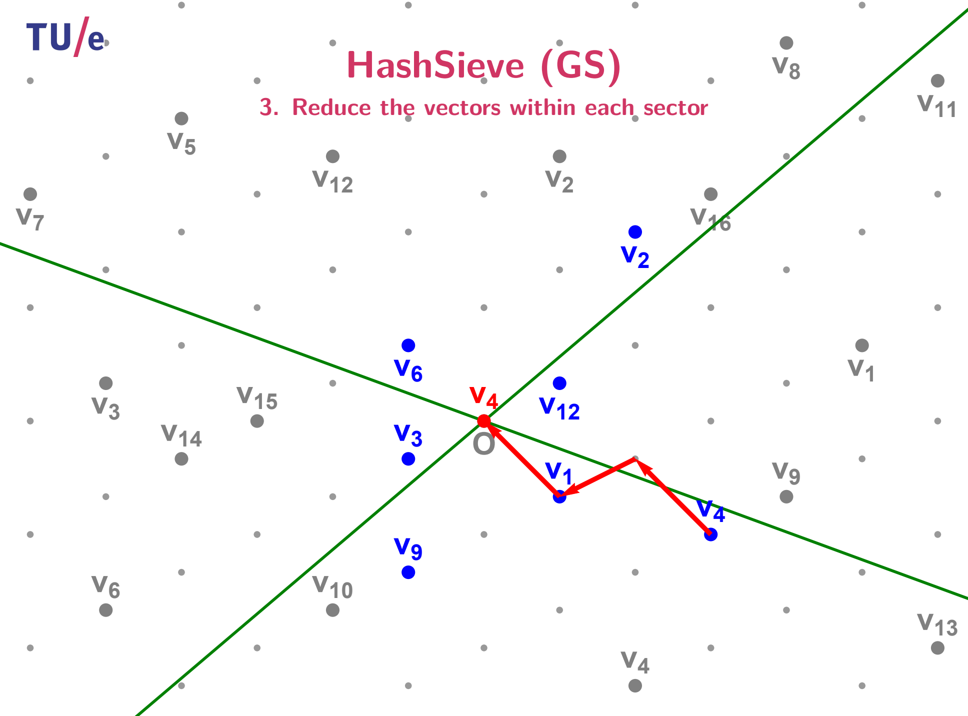
HashSieve (GS)

3. Reduce the vectors within each sector



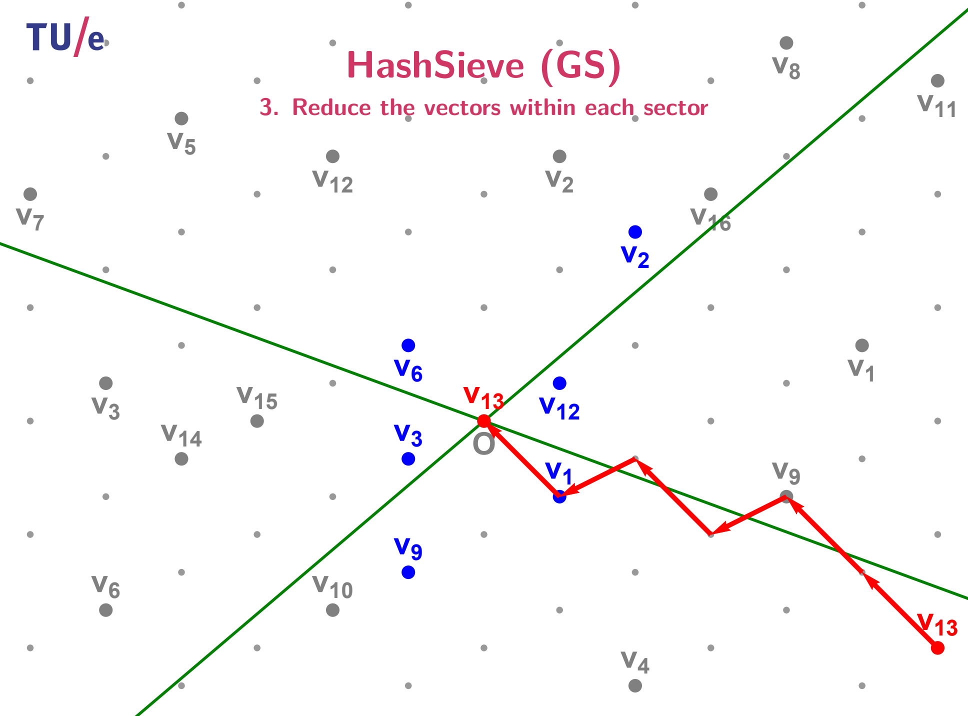
HashSieve (GS)

3. Reduce the vectors within each sector



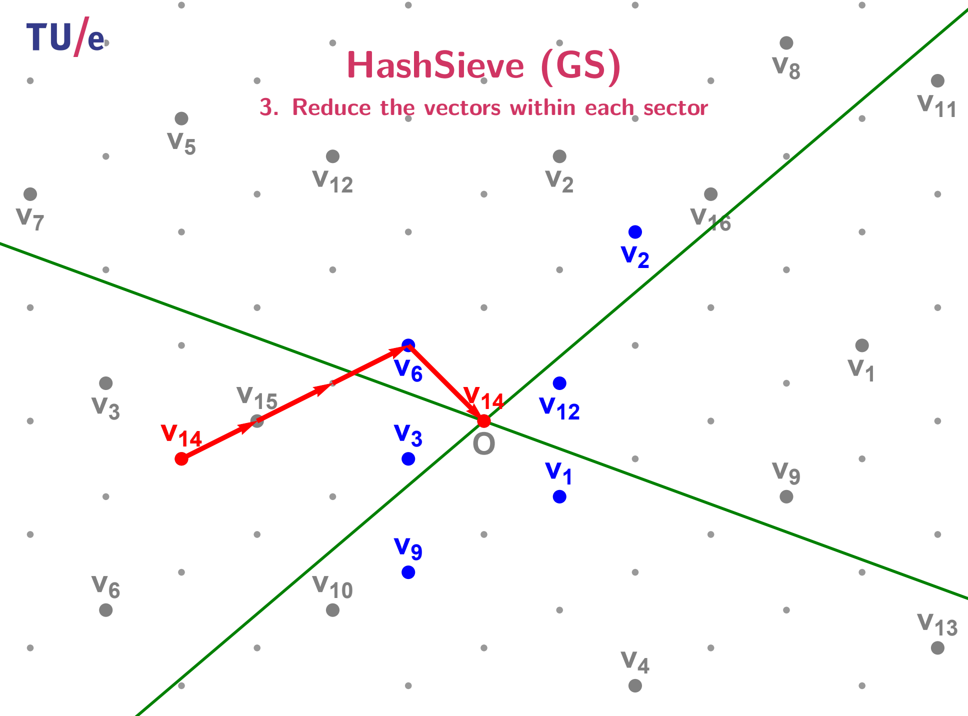
HashSieve (GS)

3. Reduce the vectors within each sector



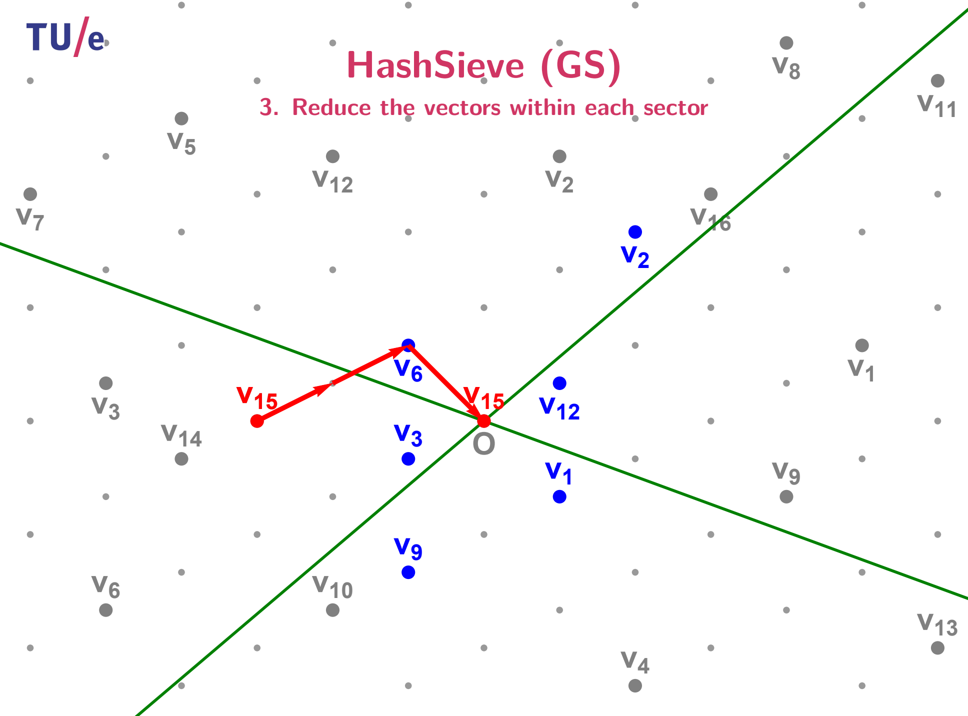
HashSieve (GS)

3. Reduce the vectors within each sector



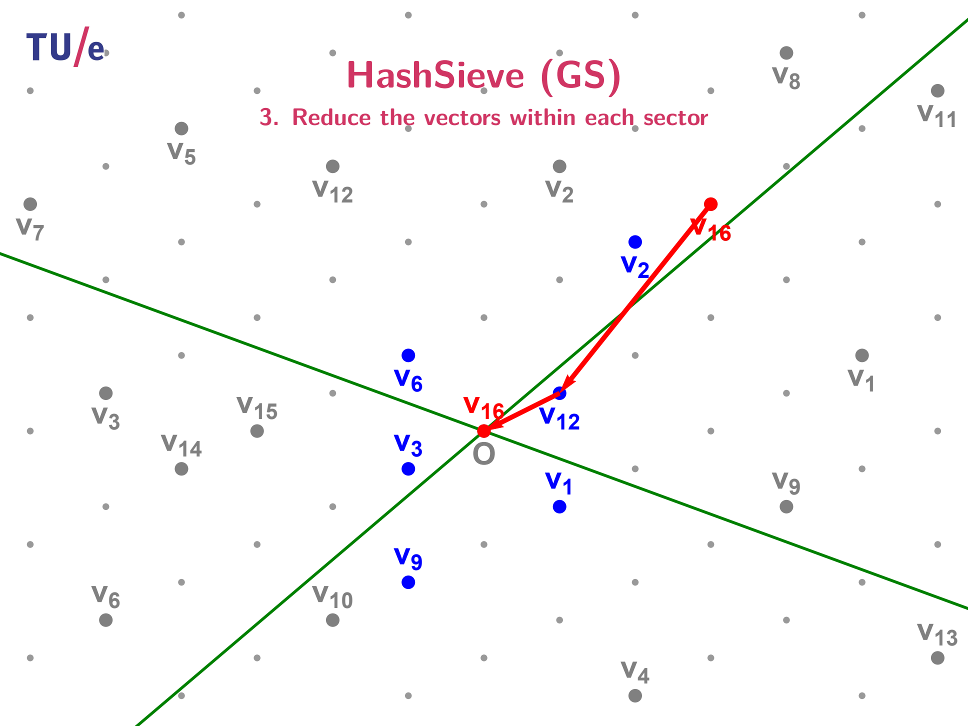
HashSieve (GS)

3. Reduce the vectors within each sector



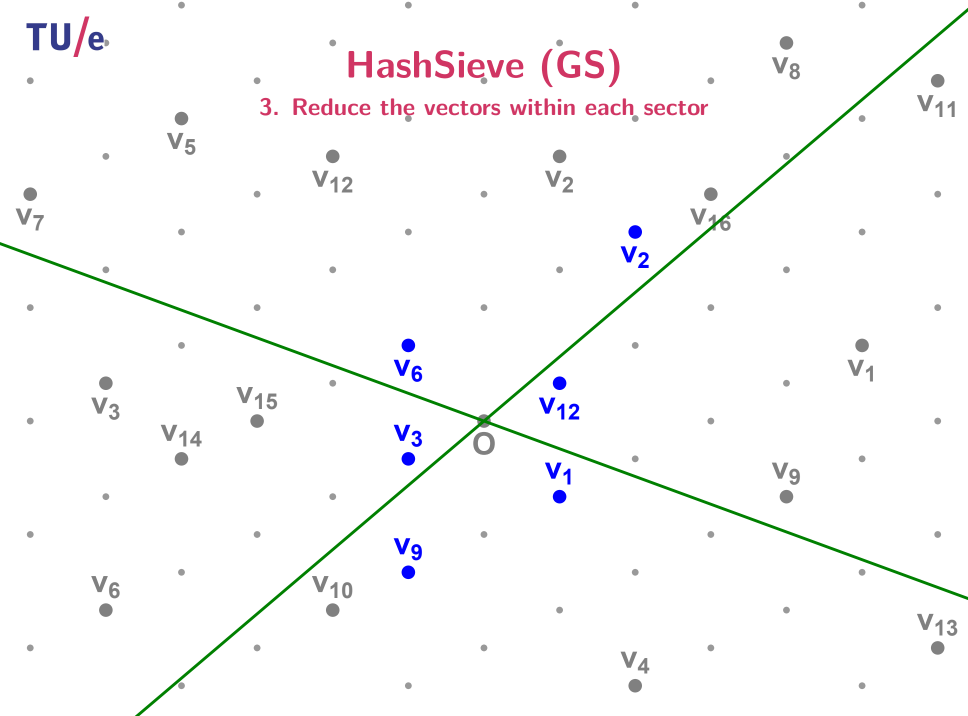
HashSieve (GS)

3. Reduce the vectors within each sector



HashSieve (GS)

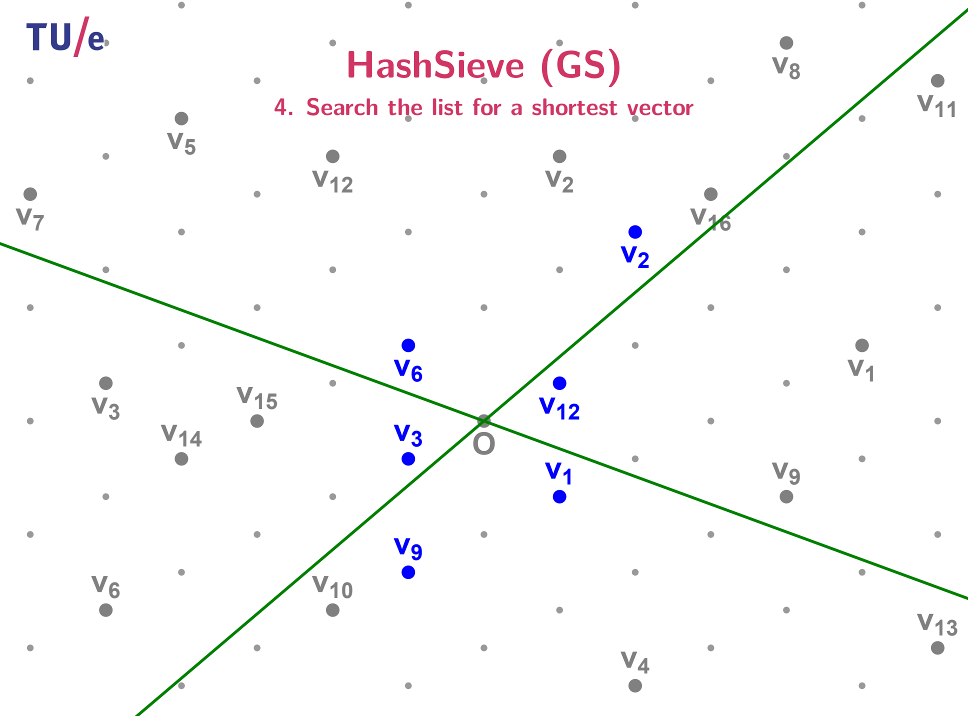
3. Reduce the vectors within each sector



TU/e

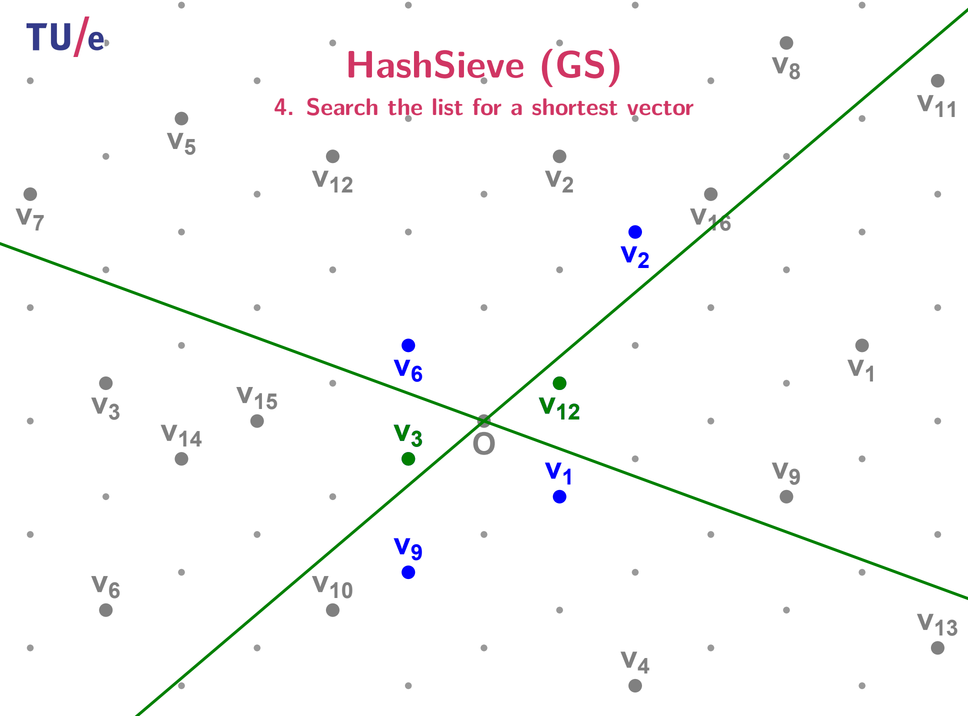
HashSieve (GS)

4. Search the list for a shortest vector



HashSieve (GS)

4. Search the list for a shortest vector



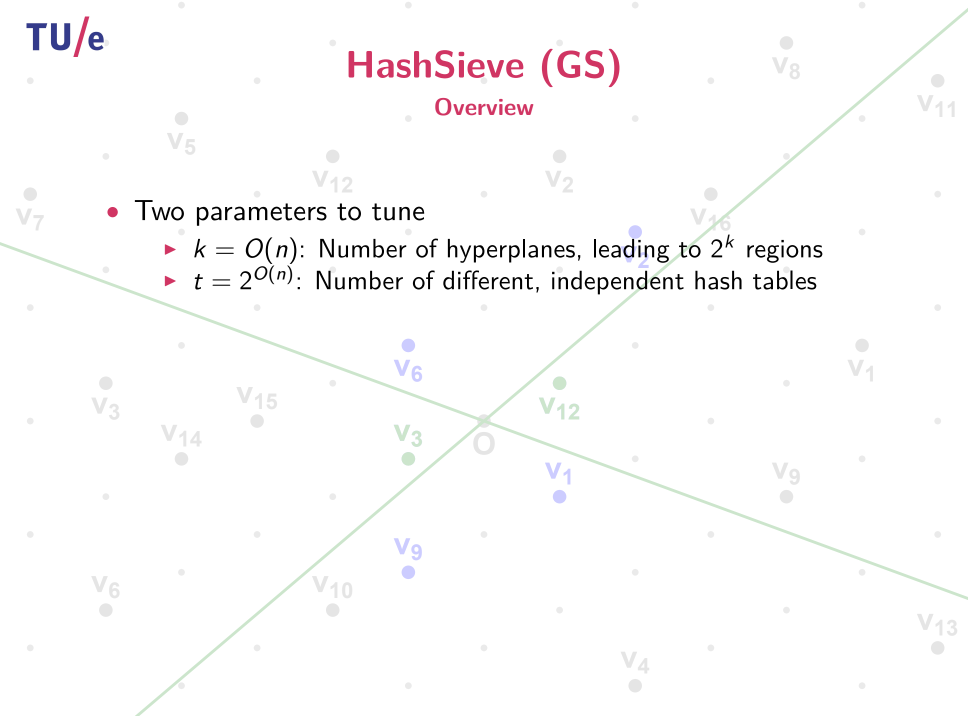
HashSieve (GS)

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

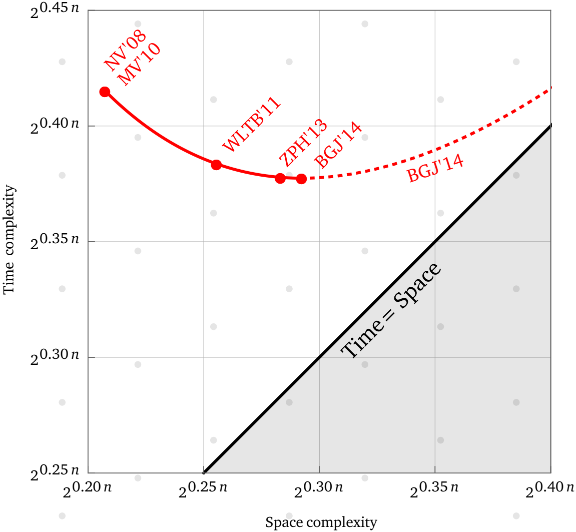
- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent hash tables
- Space complexity: $2^{0.34n+o(n)}$
  - Store $2^{0.13n}$ hash tables, each containing all $2^{0.21n}$ vectors
- Time complexity: $2^{0.34n+o(n)}$
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

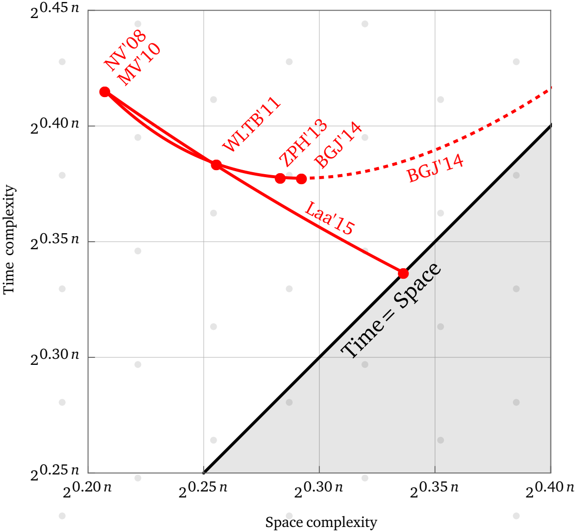
HashSieve (GS)

Space/time trade-off



HashSieve (GS)

Space/time trade-off



HashSieve (GS)

Experimental results

Dimension	**90**	**92**	**94**	**96**	**98**	**100**
Hyperplanes (k)	20	20	21	21	22	22
Hash tables (t)	3126	3738	4470	465	533	637
Probing?	N	N	N	Y	Y	Y
BKZ-β preproc.	34	34	34	34	40	40
Used vectors ($\cdot 10^6$)	2.43	3.00	4.50	5.57	7.05	10.05
Time (hours)	0.86	1.72	3.74	6.52	10.03	18.19
Memory (GB)	310	380	872	95	113	256

HashSieve (GS)

SVP challenge

22	118	2782	0	Kenji Kashiwabara and Masaharu Fukase	vec	Other	2013-08-12	1.00811
23	118	2868	8	Yuanmi Chen and Phong Nguyen	vec	ENUM	2013-02-13	1.04441
24	116	2743	0	Thorsten Kleinjung	vec	Sieving	2014-05-2	1.00492
25	116	2764	0	Kenji Kashiwabara and Masaharu Fukase	vec	Other	2014-03-21	1.01287
26	116	2786	0	Kenji Kashiwabara and Masaharu Fukase	vec	Other	2013-08-3	1.02075

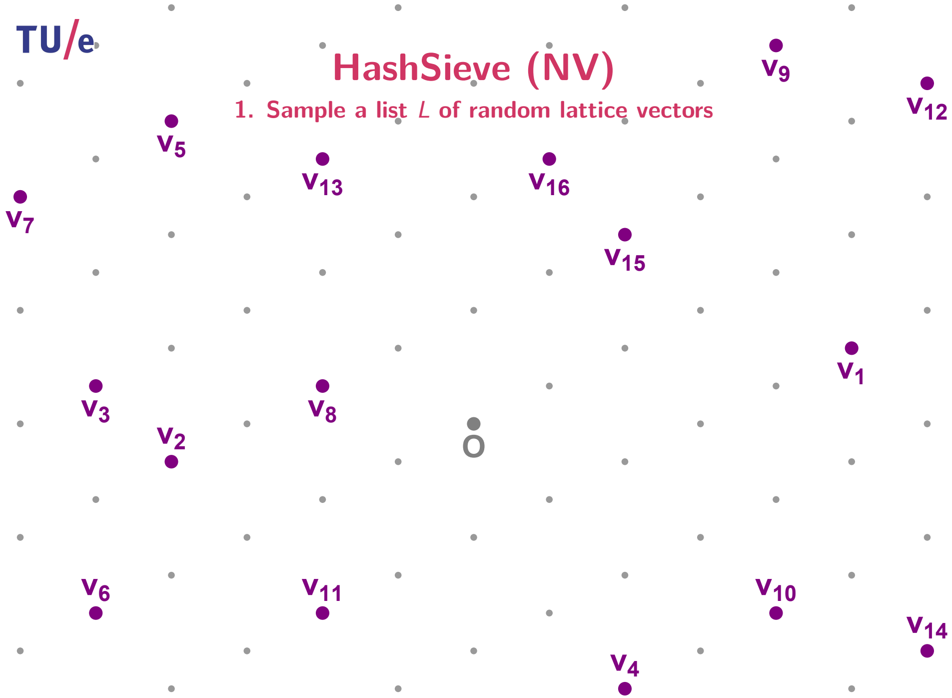
40	108	2508	0	Yuanmi Chen and Phong Nguyen	vec	Other	2010-06-16	0.95162
41	108	2755	0	Yuanmi Chen and Phong Nguyen	vec	Other	2010-05-30	1.04519
42	107	2626	0	Artur Mariano	vec	Sieving	2015-05-22	1.00056
43	107	2713	0	Artur Mariano	vec	Sieving	2015-05-22	1.03379
44	107	2716	0	Artur Mariano	vec	Sieving	2015-05-22	1.03490
45	107	2719	0	Artur Mariano	vec	Sieving	2015-05-22	1.03609
46	107	2720	0	Artur Mariano	vec	Sieving	2015-05-22	1.03657
47	107	2721	0	Artur Mariano	vec	Sieving	2015-05-22	1.03688
48	107	2722	0	Artur Mariano	vec	Sieving	2015-05-22	1.03721
49	107	2724	8	Po-Chun Kuo, Michael Schneider	vec	ENUM,BKZ	2011-03-12	1.03655

HashSieve (NV)

1. Sample a list L of random lattice vectors

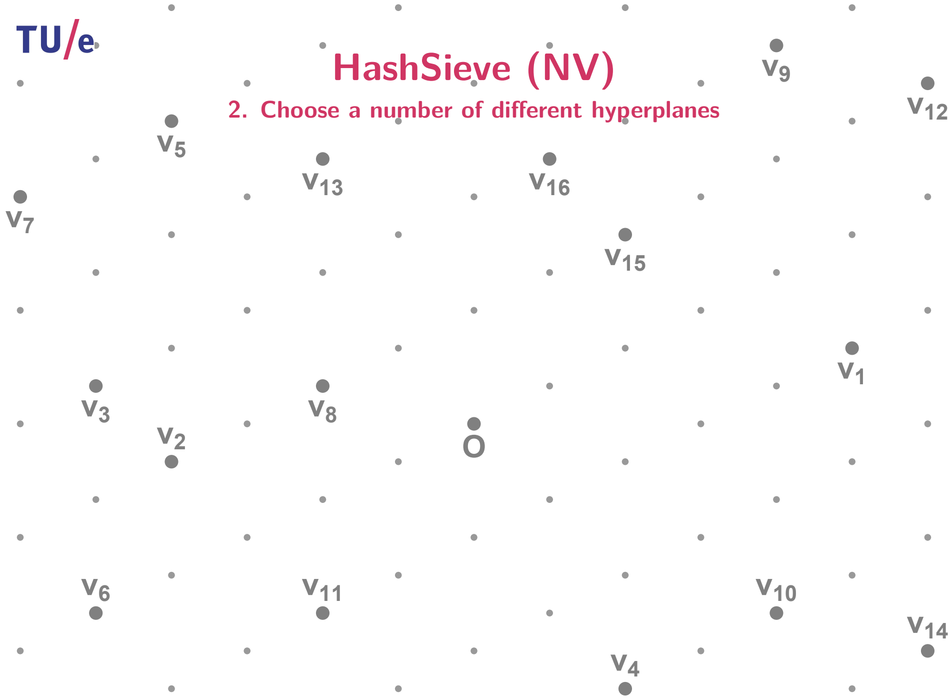
HashSieve (NV)

1. Sample a list L of random lattice vectors



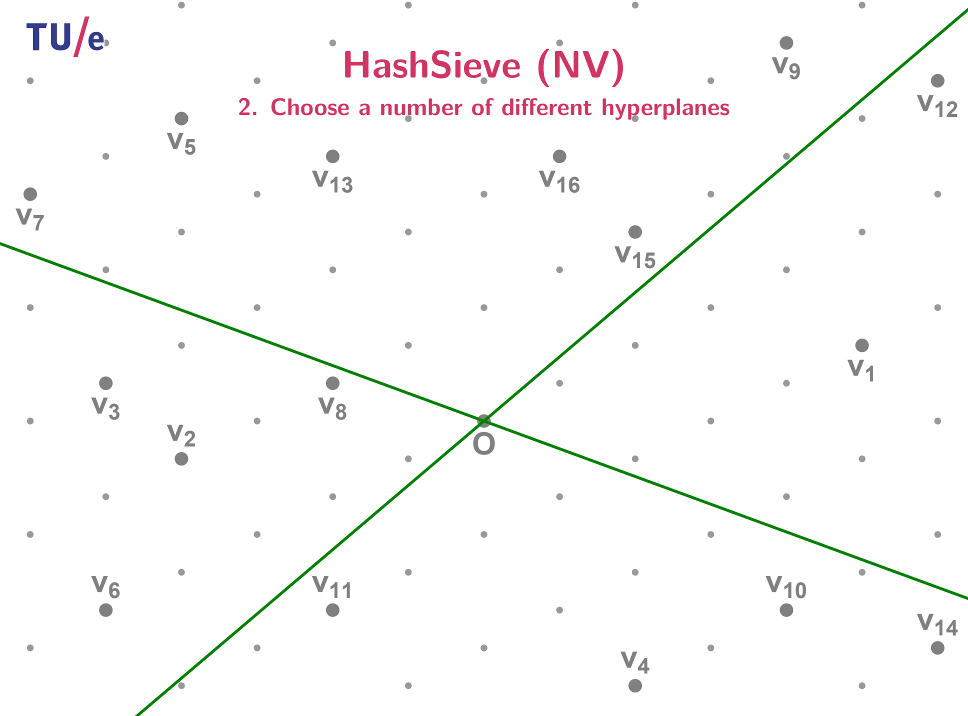
HashSieve (NV)

2. Choose a number of different hyperplanes



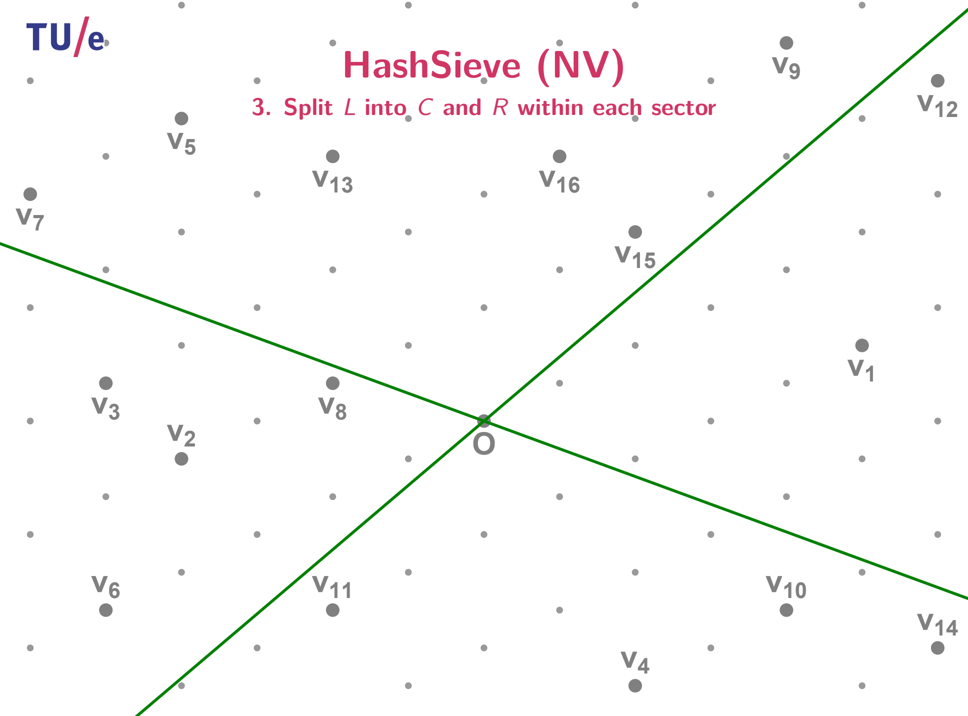
HashSieve (NV)

2. Choose a number of different hyperplanes



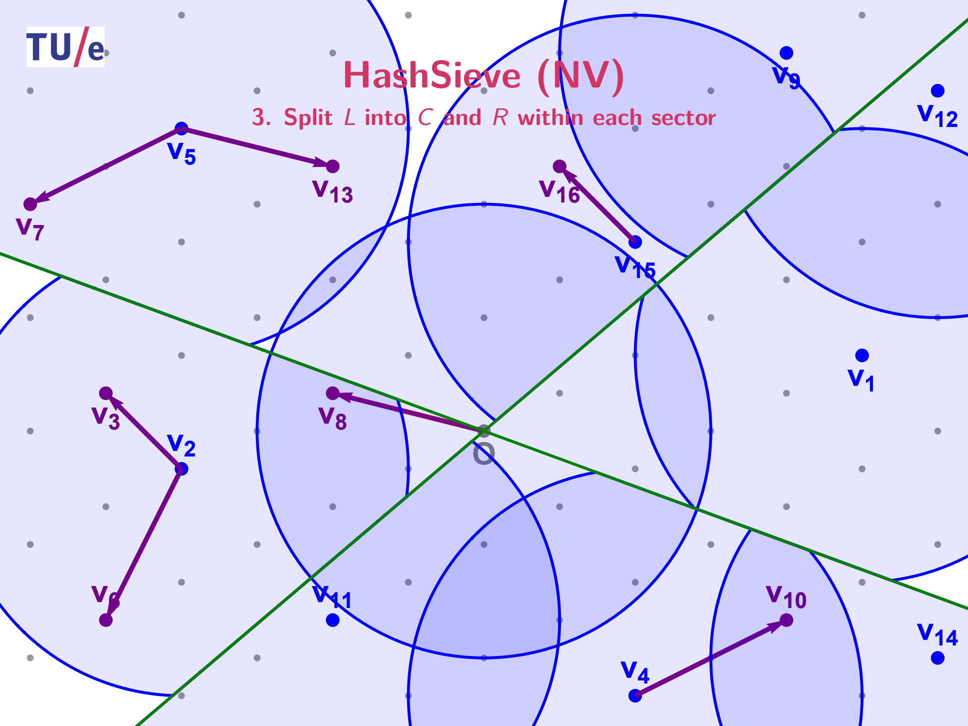
HashSieve (NV)

3. Split L into C and R within each sector



HashSieve (NV)

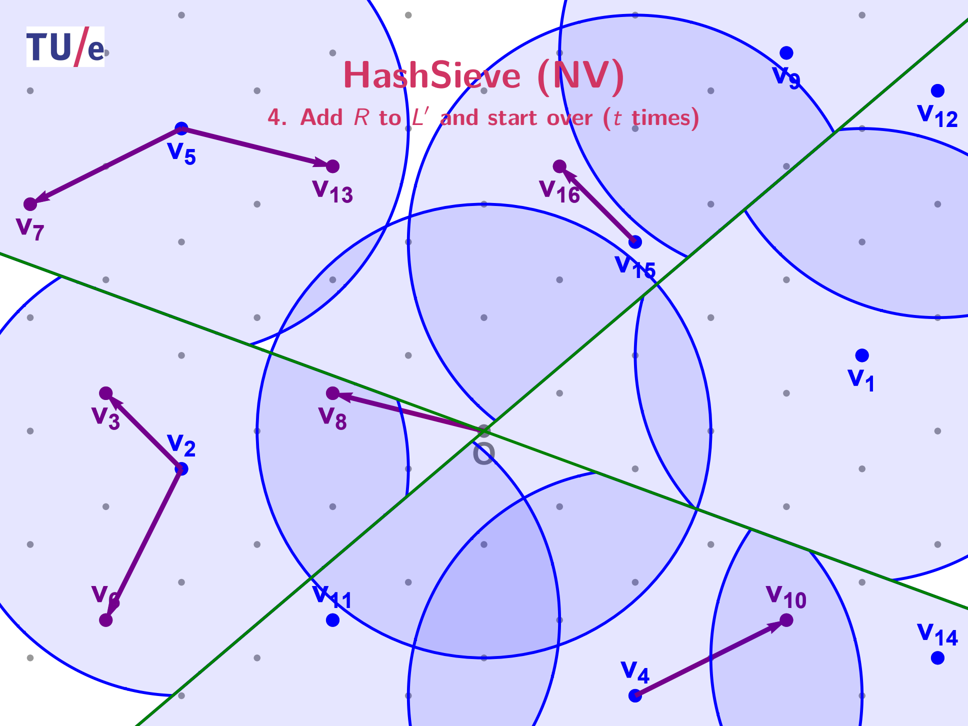
3. Split L into C and R within each sector



TU/e

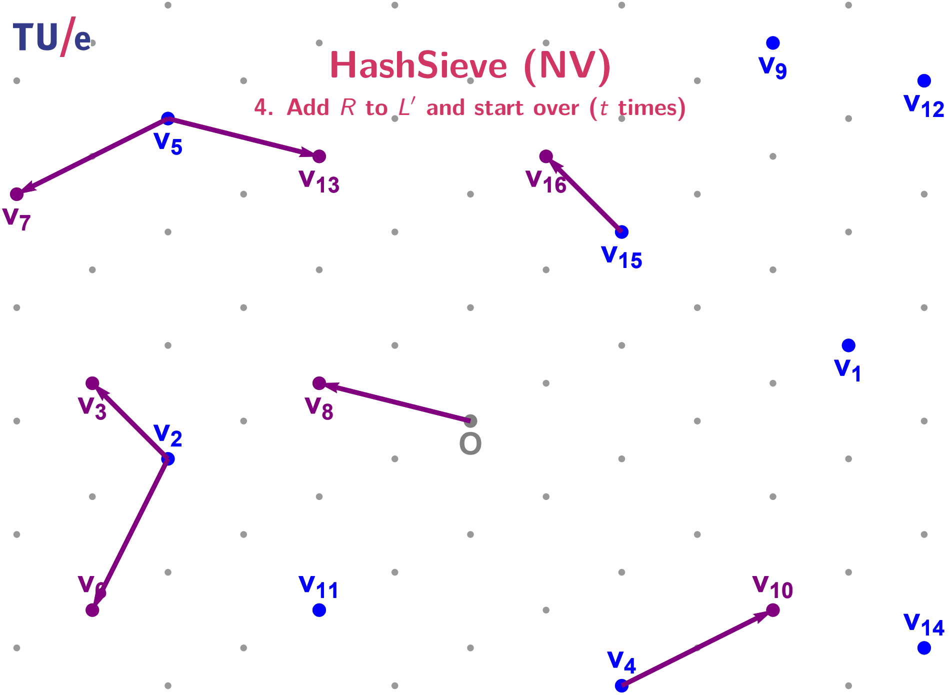
HashSieve (NV)

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



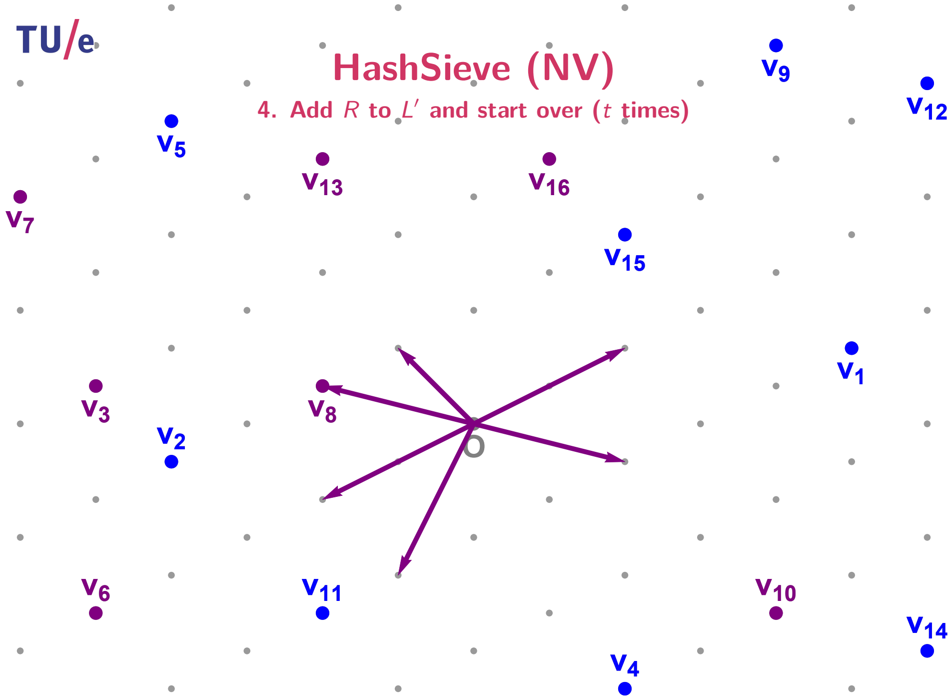
HashSieve (NV)

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



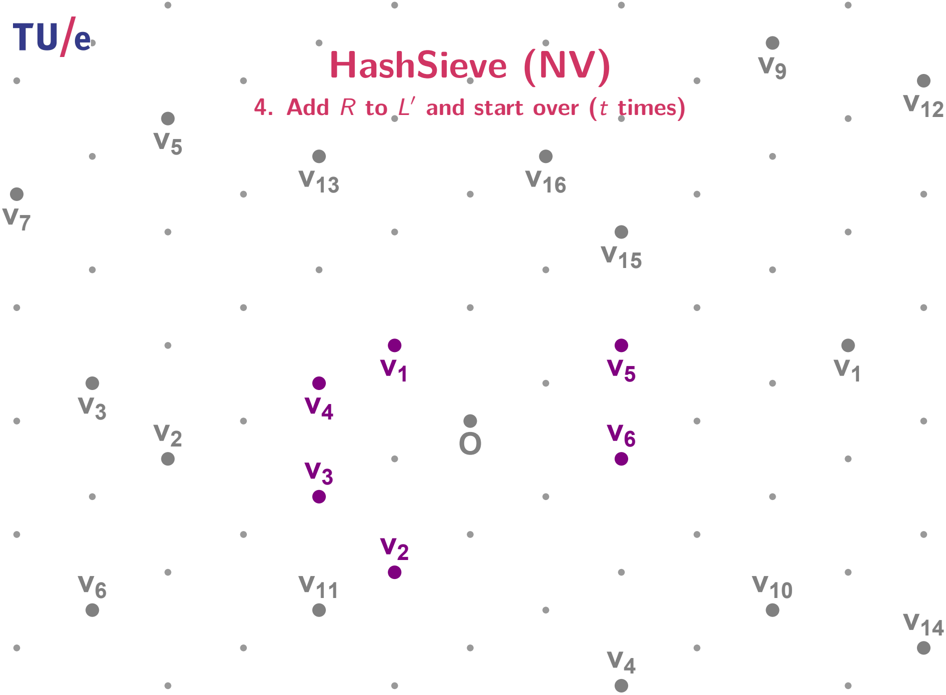
HashSieve (NV)

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



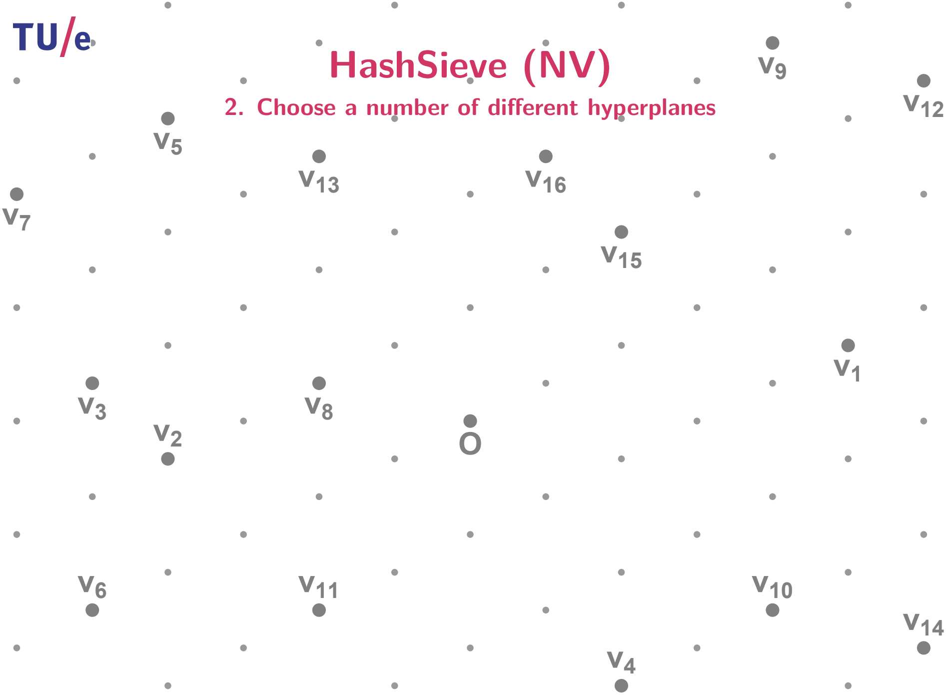
HashSieve (NV)

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



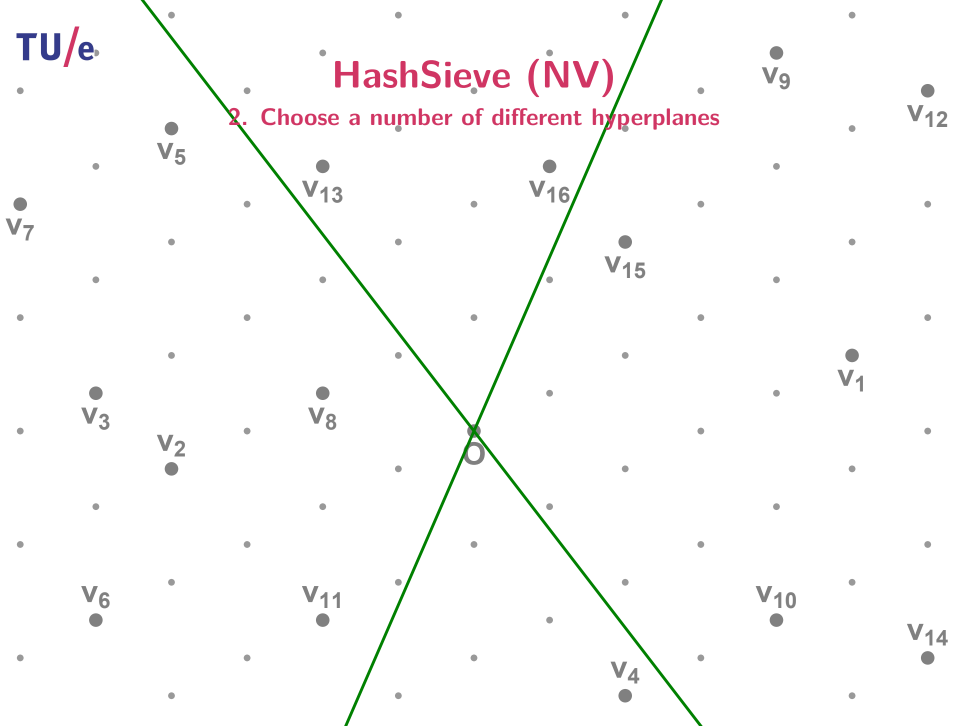
HashSieve (NV)

2. Choose a number of different hyperplanes



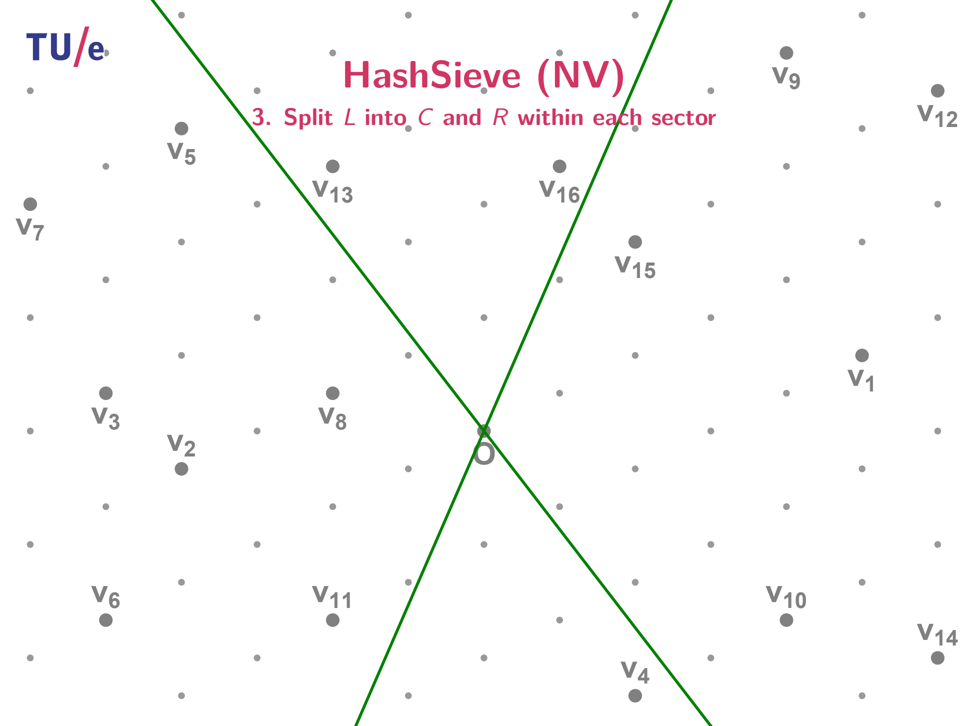
HashSieve (NV)

2. Choose a number of different hyperplanes



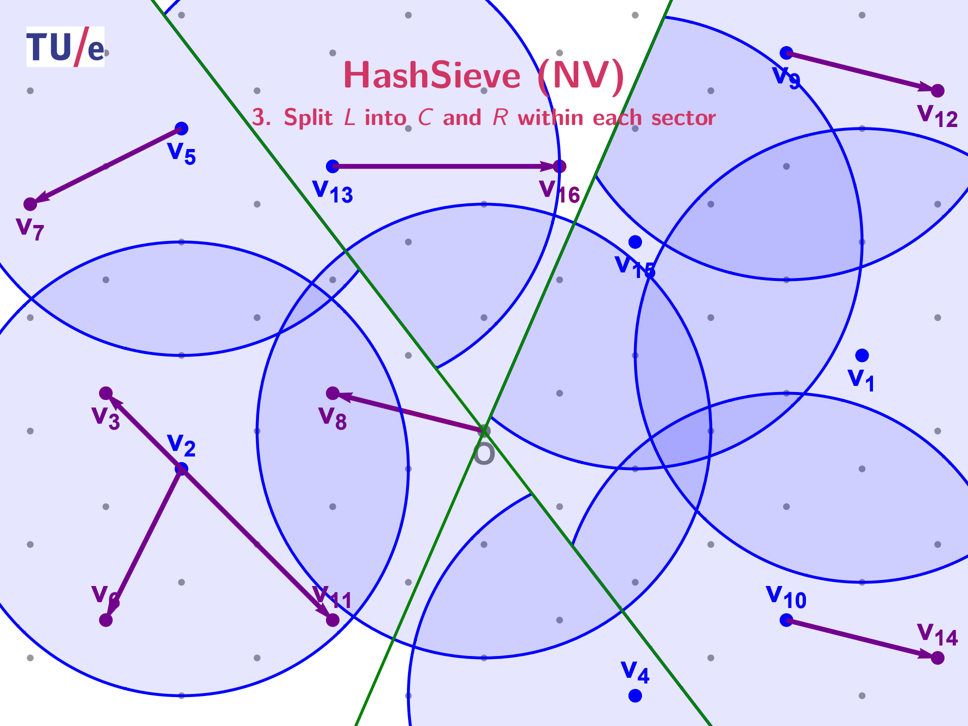
HashSieve (NV)

3. Split L into C and R within each sector



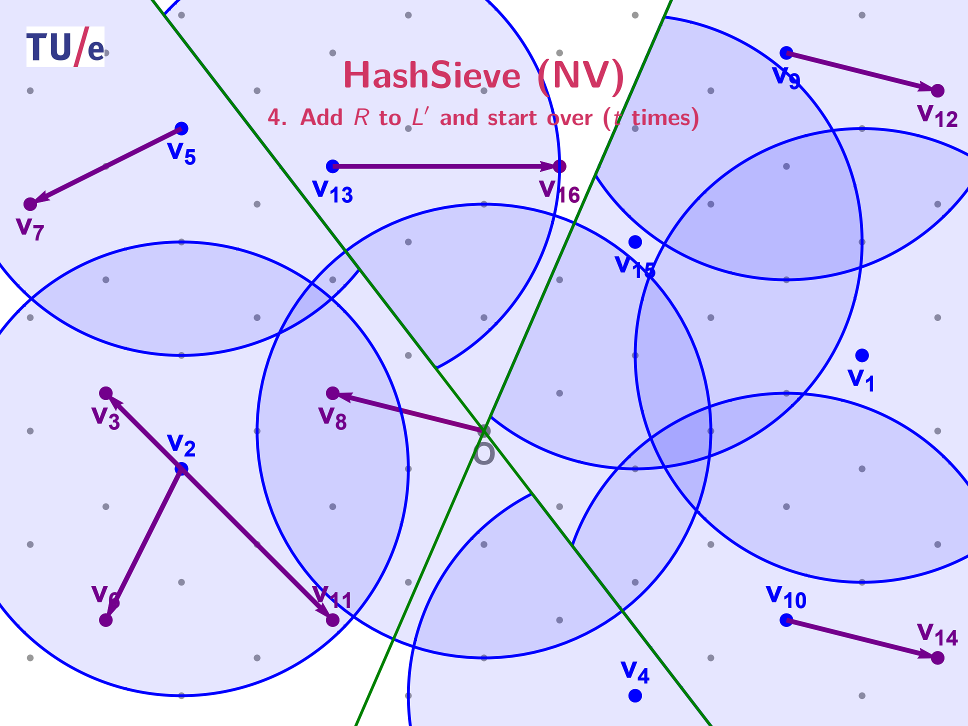
HashSieve (NV)

3. Split L into C and R within each sector



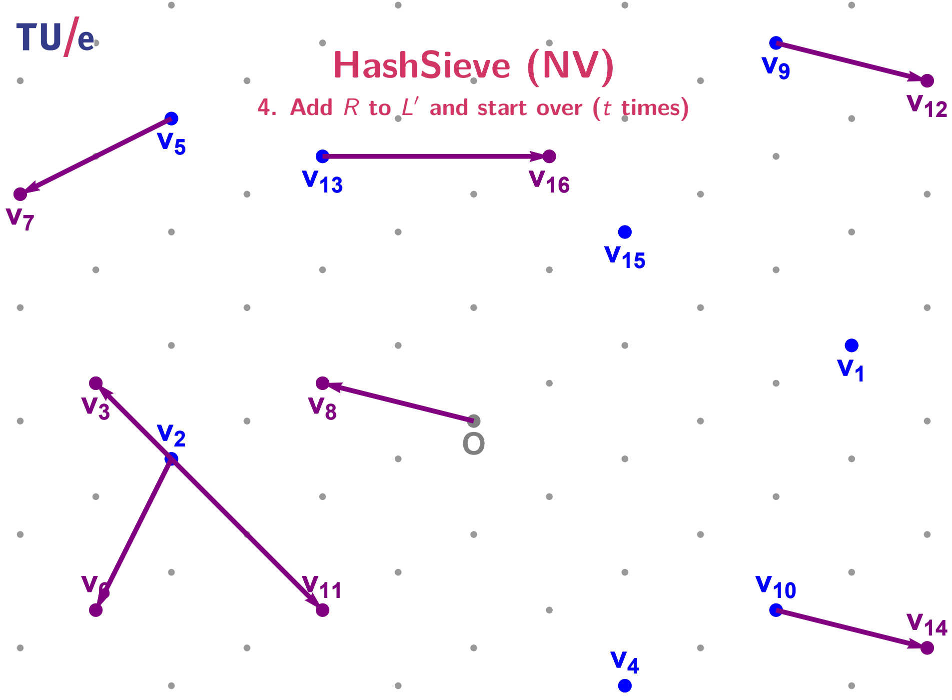
HashSieve (NV)

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



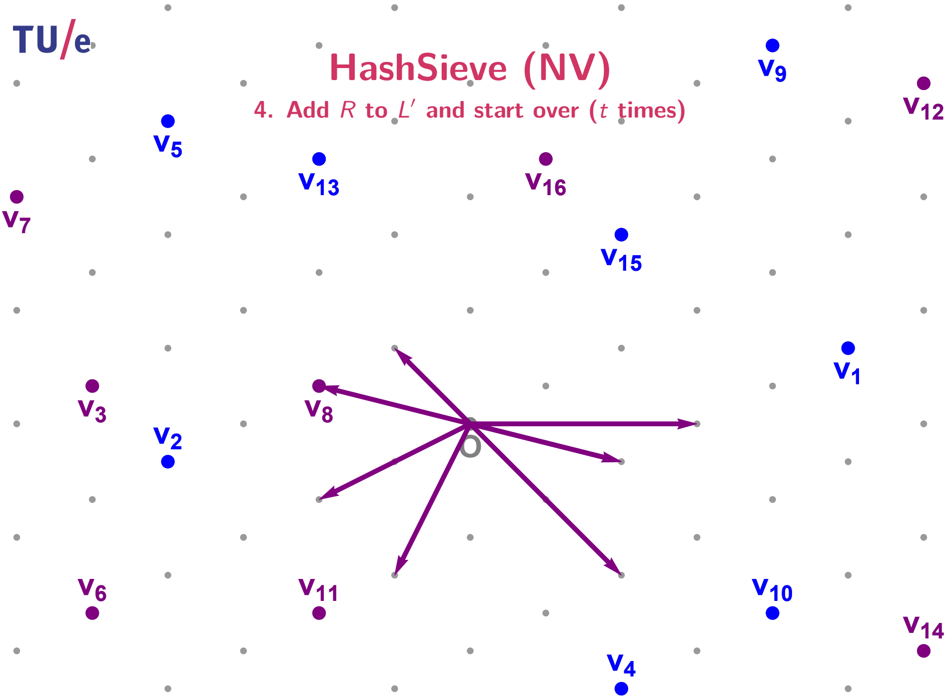
HashSieve (NV)

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



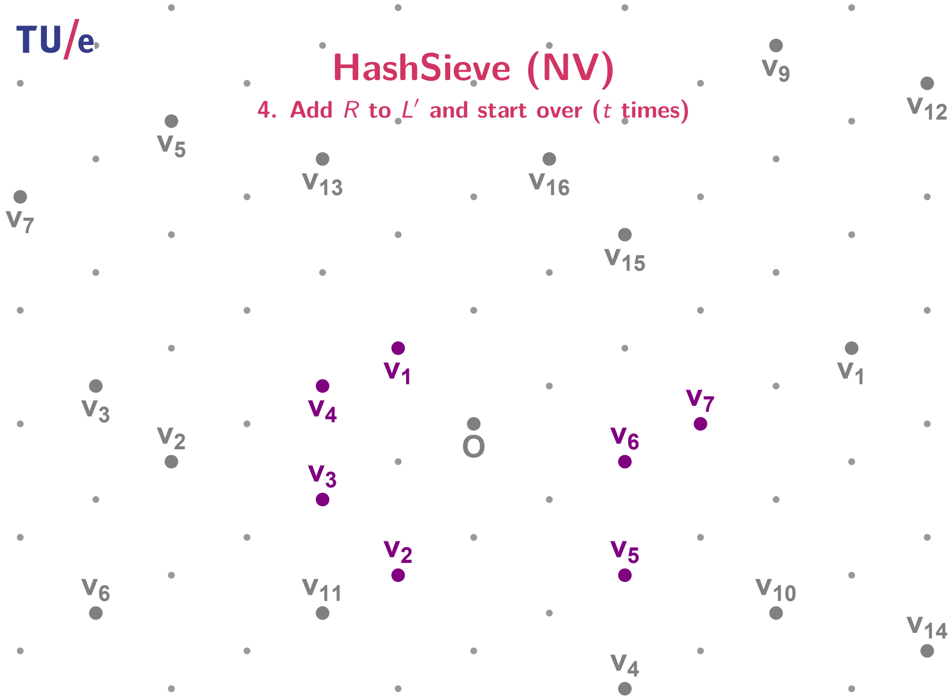
HashSieve (NV)

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

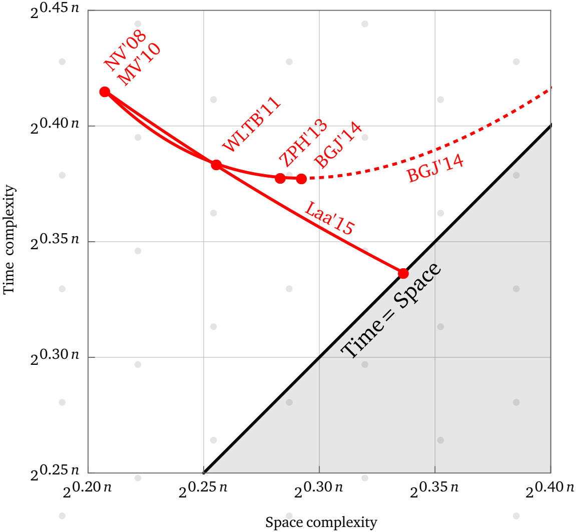
- Same k (hyperplanes) and t (hash tables) as before
- Space complexity: $2^{0.21n+o(n)}$
  - Before: store $2^{0.13n}$ hash tables containing all $2^{0.21n}$ vectors
  - Now: process $2^{0.13n}$ hash tables one by one
- Time complexity: $2^{0.34n+o(n)}$
  - Compute one hash, and go through $2^{o(n)}$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

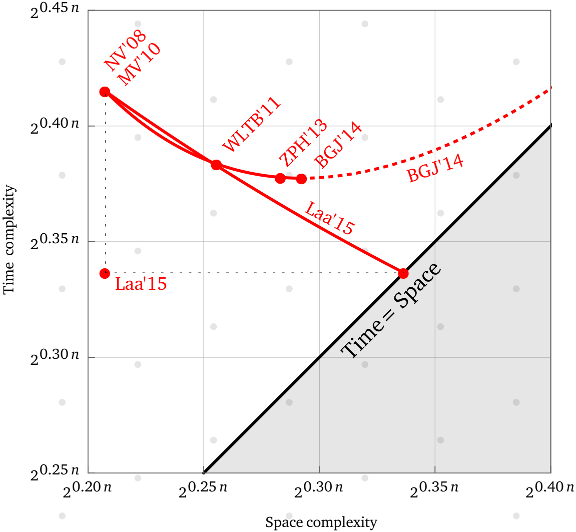
HashSieve (NV)

Space/time trade-off



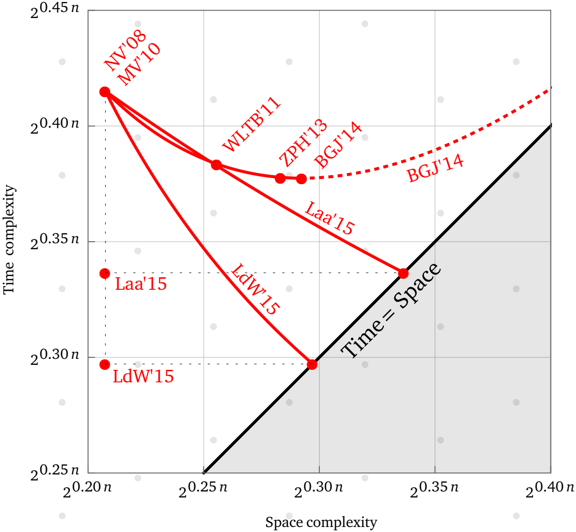
HashSieve (NV)

Space/time trade-off



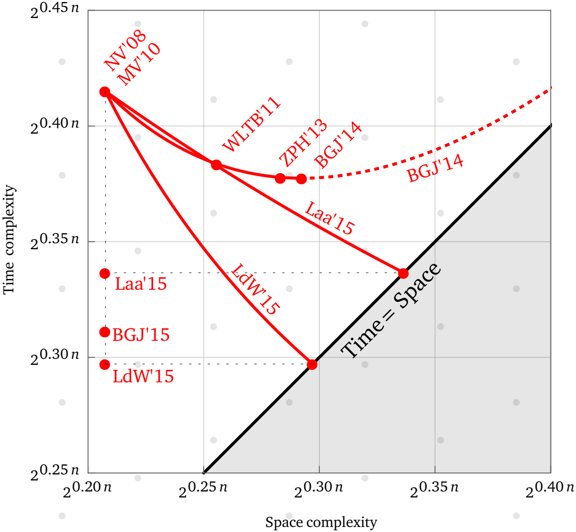
SphereSieve

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
- Observation: orthogonal hyperplanes better than random ones
  - Corresponds to "hypercube hashing"

CrossPolytopeSieve

Main ideas

- HashSieve: divide space into "equal" parts with hyperplanes
- Observation: orthogonal hyperplanes better than random ones
    - Corresponds to "hypercube hashing"
- Observation: hypercube not "most symmetric" object ($k \ll n$)

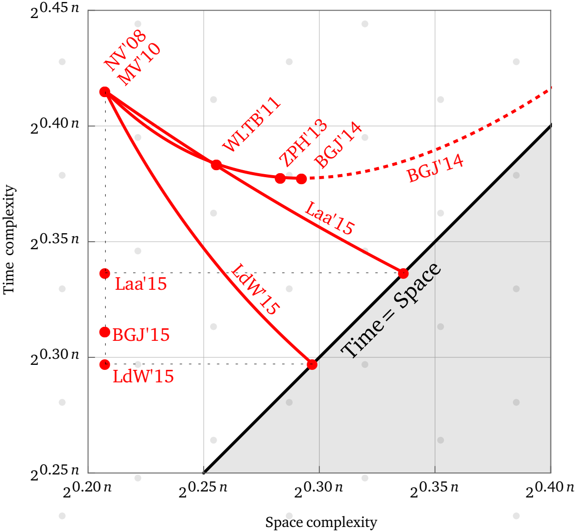
CrossPolytopeSieve

Main ideas

- HashSieve: divide space into "equal" parts with hyperplanes
- Observation: orthogonal hyperplanes better than random ones
  - Corresponds to "hypercube hashing"
- Observation: hypercube not "most symmetric" object ($k \ll n$)
- Idea: divide space into regions using regular polytopes
  - Hypercube $\implies$ HashSieve with orthogonal hyperplanes
  - Cross polytope $\implies$ CrossPolytopeSieve
  - Simplex $\implies$ similar to CrossPolytopeSieve

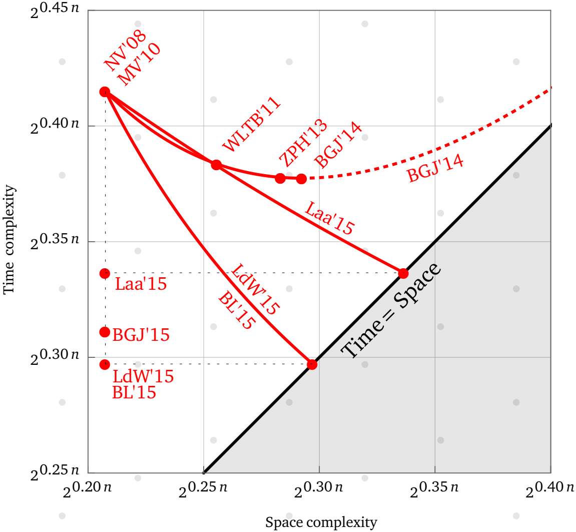
CrossPolytopeSieve

Space/time trade-off



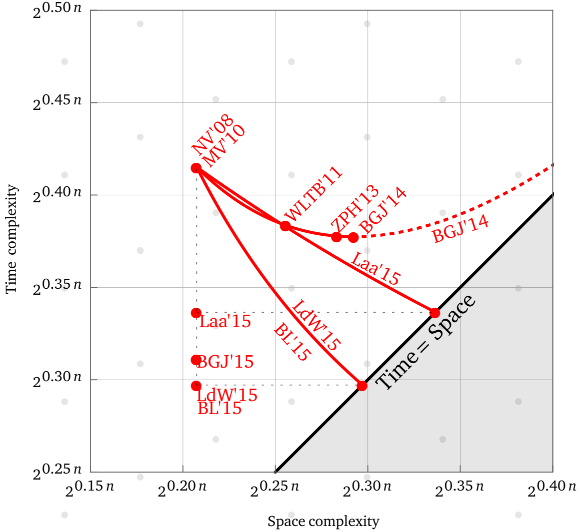
CrossPolytopeSieve

Space/time trade-off



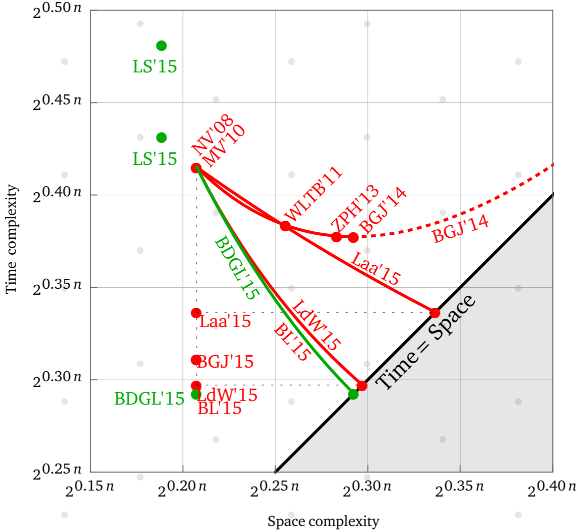
Ongoing work

Space/time trade-off



Ongoing work

Space/time trade-off



Questions