Sieving for shortest vectors in lattices using (spherical) locality-sensitive hashing

Thijs Laarhoven, Benne de Weger

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

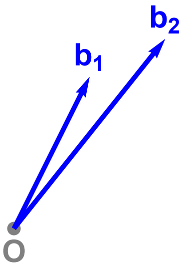
Latincrypt 2015, Guadalajara, Mexico
(August 24, 2015)

Lattices

What is a lattice?

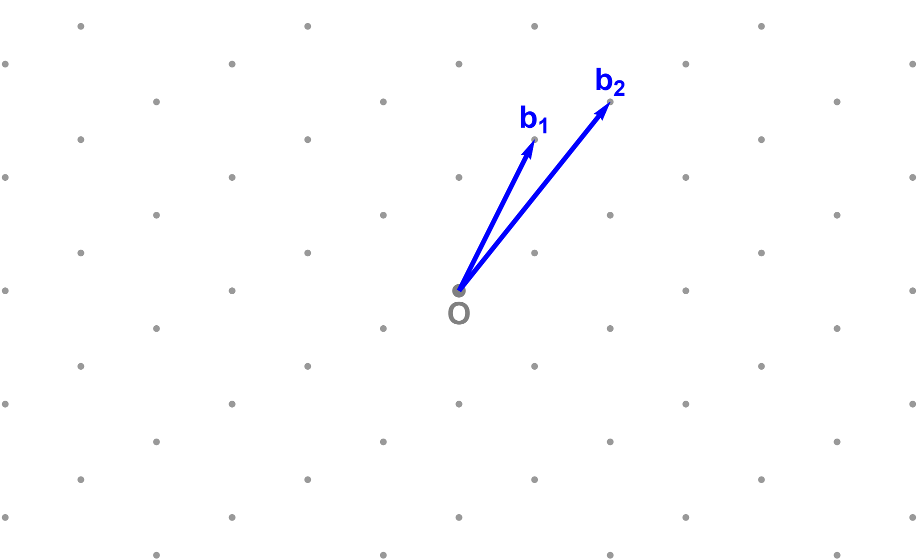
Lattices

What is a lattice?



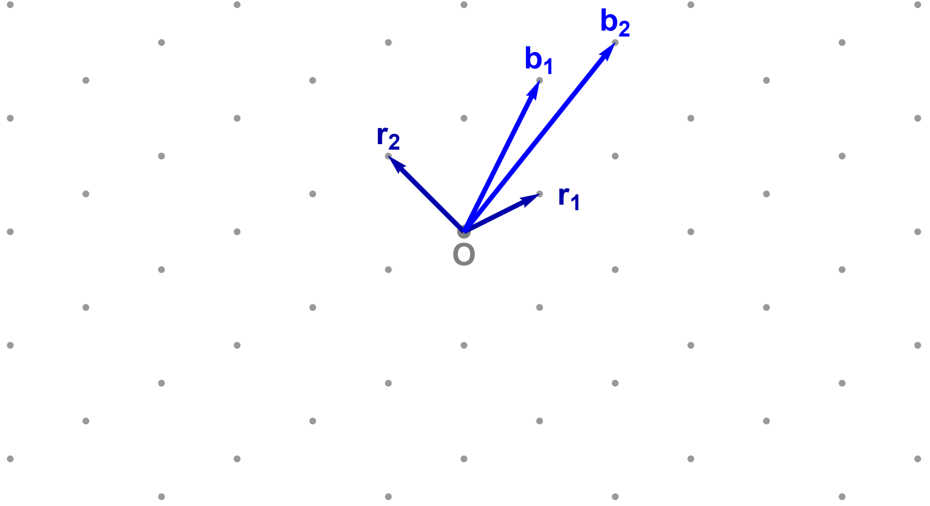
Lattices

What is a lattice?



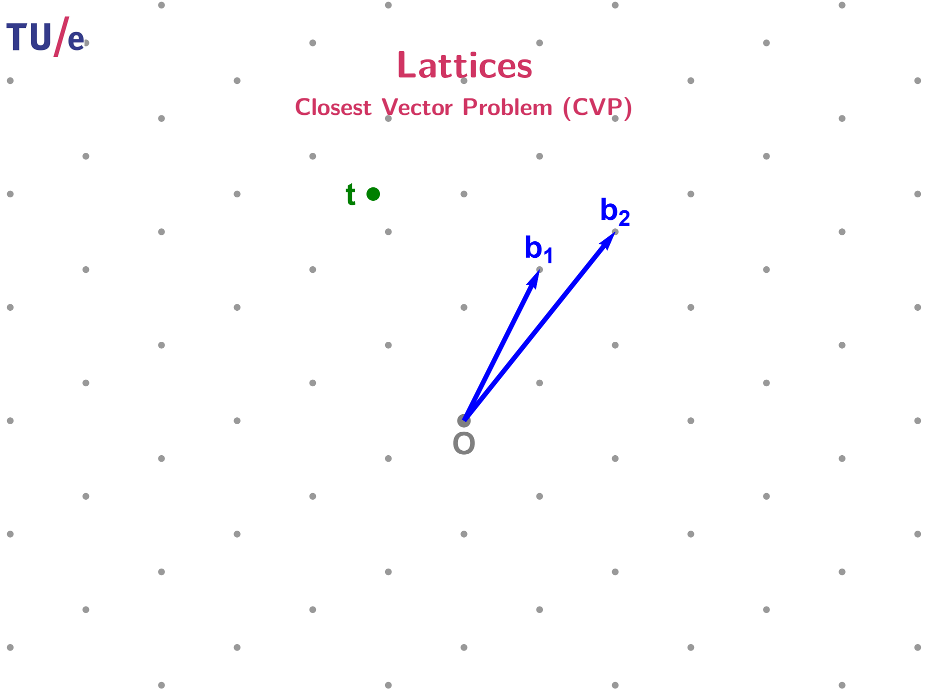
Lattices

Lattice basis reduction



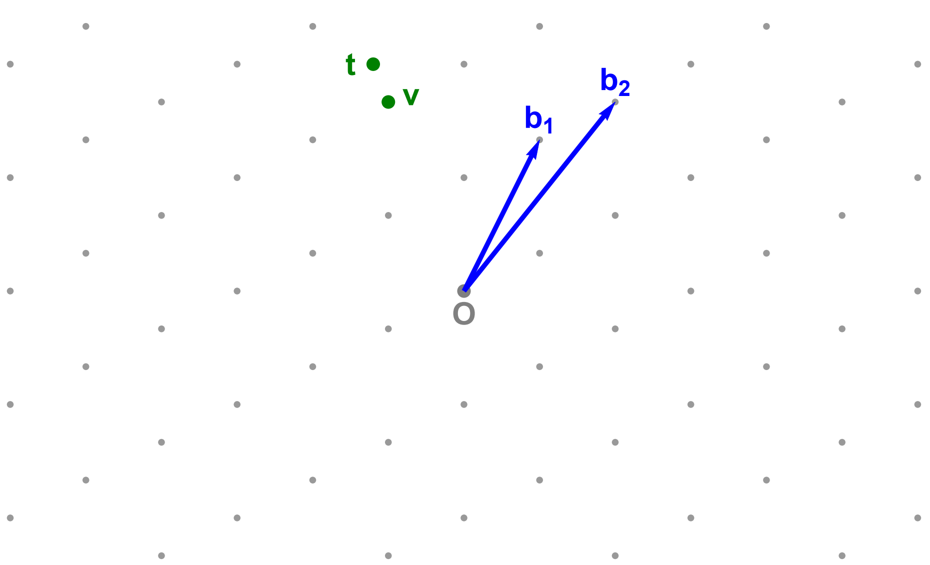
Lattices

Closest Vector Problem (CVP)



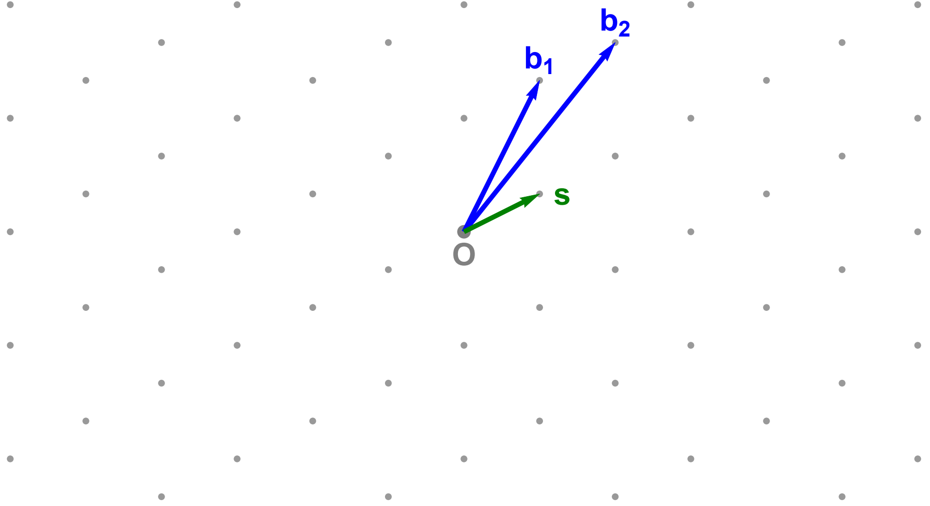
Lattices

Closest Vector Problem (CVP)



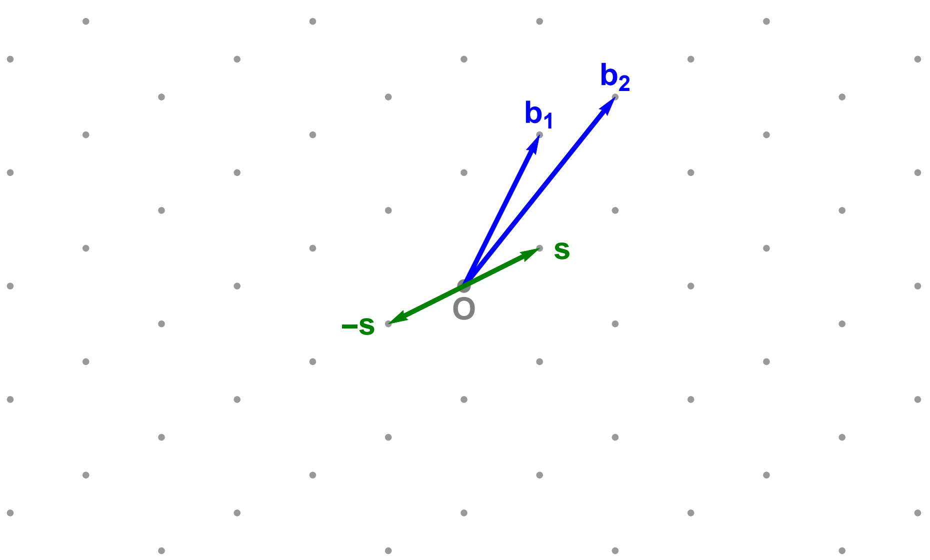
Lattices

Shortest Vector Problem (SVP)



Lattices

Shortest Vector Problem (SVP)



TU/e

Lattices

Exact SVP algorithms

	Algorithm	$\log_2$(Time)	$\log_2$(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]	$1.000n$	$1.000n$
Heuristic SVP	Nguyen-Vidick sieve [NV08]	$0.415n$	$0.208n$
	GaussSieve [MV10, . . . , IKMT14, BNvdP14]	$0.415n$?	$0.208n$
	Two-level sieve [WLTB11]	$0.384n$	$0.256n$
	Three-level sieve [ZPH13]	$0.3778n$	$0.283n$
	Overlattice sieving [BGJ14]	$0.3774n$	$0.293n$
	Hyperplane LSH [Laa15, MLB15]	$0.337n$	$0.208n$

TU/e

Lattices

Exact SVP algorithms

	Algorithm	$\log_2$(Time)	$\log_2$(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]	$1.000n$	$1.000n$
Heuristic SVP	Nguyen-Vidick sieve [NV08]	$0.415n$	$0.208n$
	GaussSieve [MV10, ..., IKMT14, BNvdP14]	$0.415n$?	$0.208n$
	Two-level sieve [WLTB11]	$0.384n$	$0.256n$
	Three-level sieve [ZPH13]	$0.3778n$	$0.283n$
	Overlattice sieving [BGJ14]	$0.3774n$	$0.293n$
	Hyperplane LSH [Laa15, MLB15]	$0.337n$	$0.208n$
	Spherical LSH **[LdW15]**	$0.298n$	$0.208n$

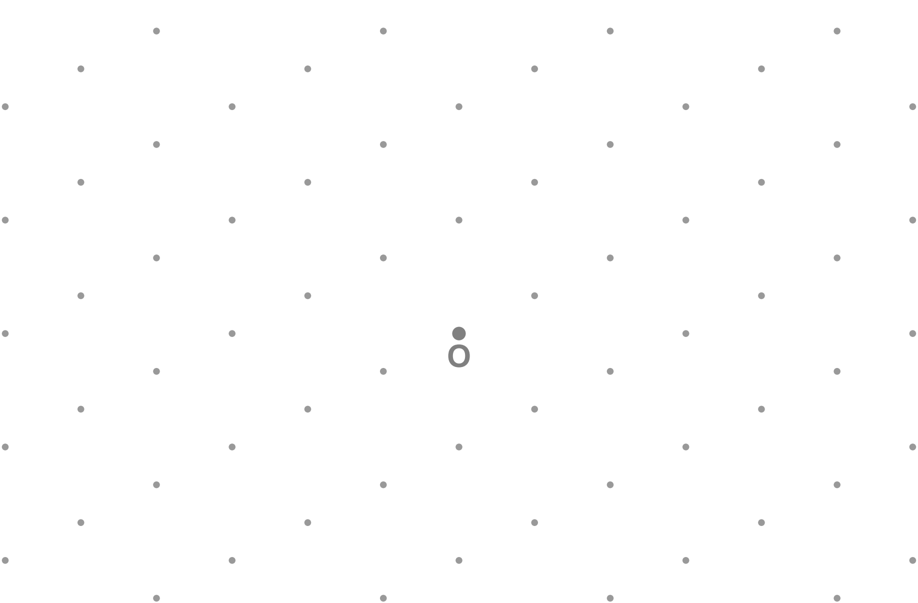
TU/e

Lattices

Exact SVP algorithms

	Algorithm	$\log_2$(Time)	$\log_2$(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]	$1.000n$	$1.000n$
Heuristic SVP	Nguyen-Vidick sieve [NV08]	$0.415n$	$0.208n$
	GaussSieve [MV10, . . . , IKMT14, BNvdP14]	$0.415n$?	$0.208n$
	Two-level sieve [WLTB11]	$0.384n$	$0.256n$
	Three-level sieve [ZPH13]	$0.3778n$	$0.283n$
	Overlattice sieving [BGJ14]	$0.3774n$	$0.293n$
	Hyperplane LSH [Laa15, MLB15]	$0.337n$	$0.208n$
	May and Ozerov's NNS method [BGJ15]	$0.311n$	$0.208n$
	Spherical LSH **[LdW15]**	$0.298n$	$0.208n$
	Cross-polytope LSH [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

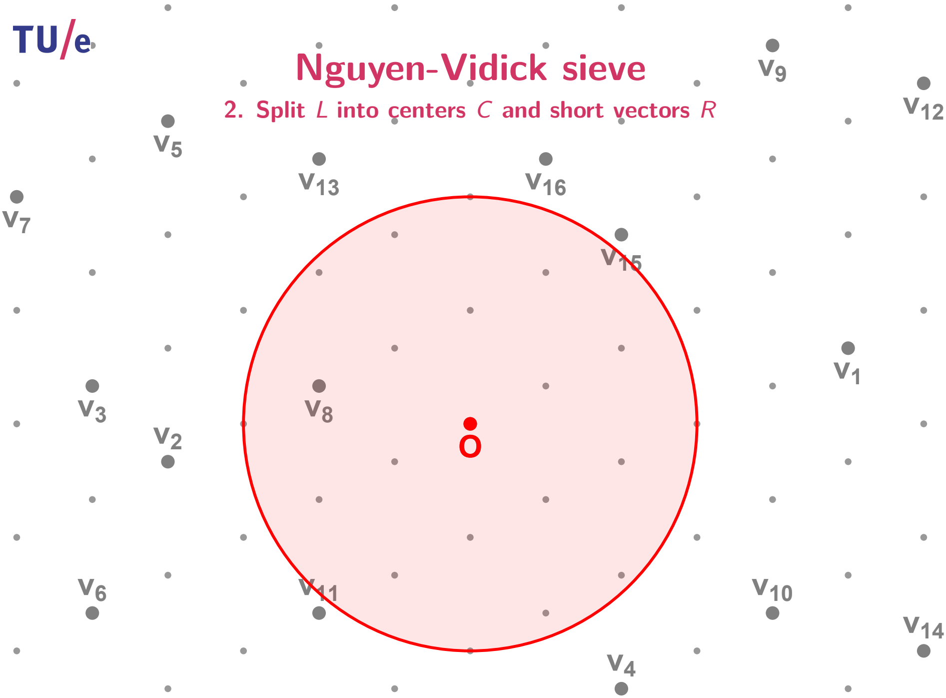
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R

TU/e

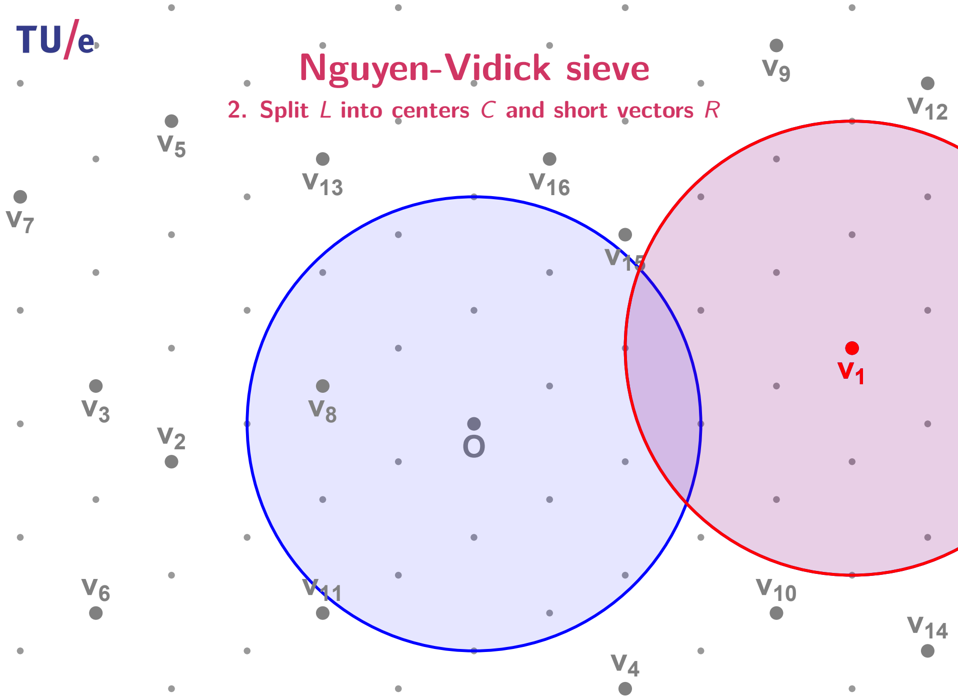
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



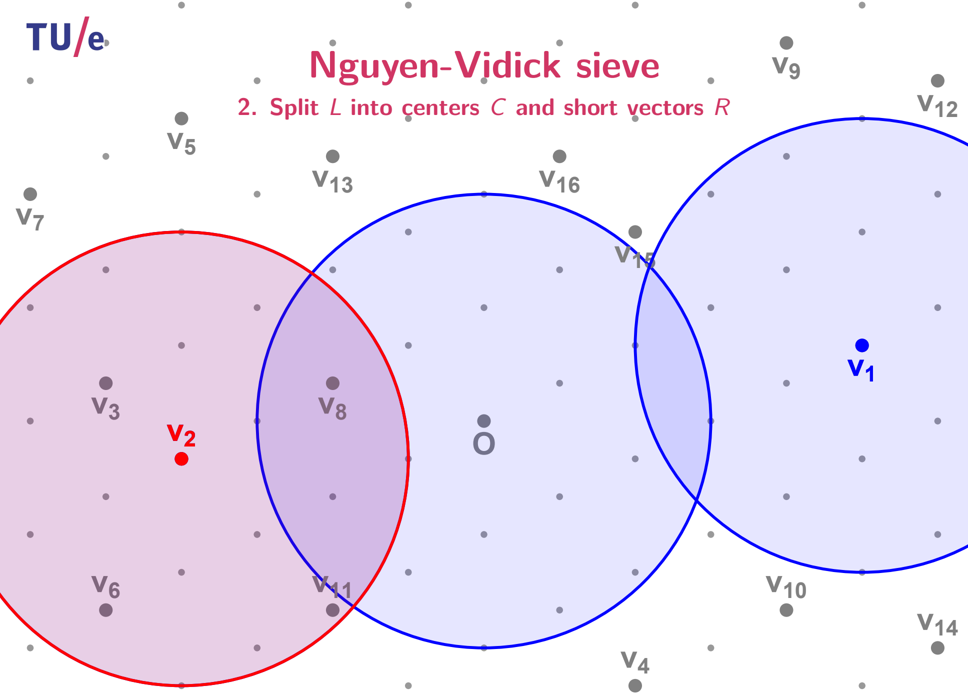
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



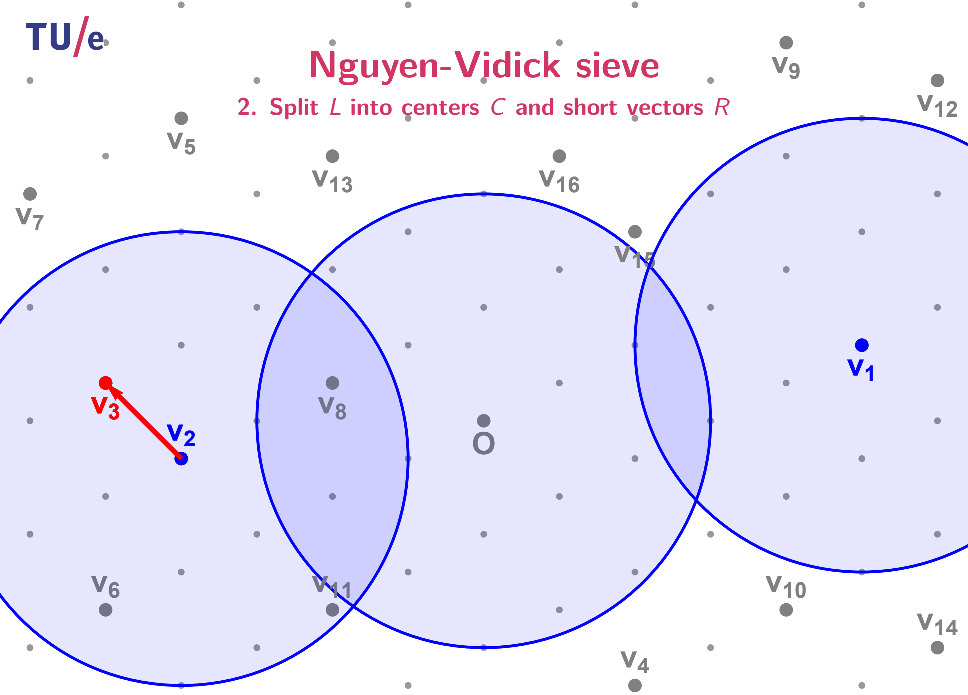
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



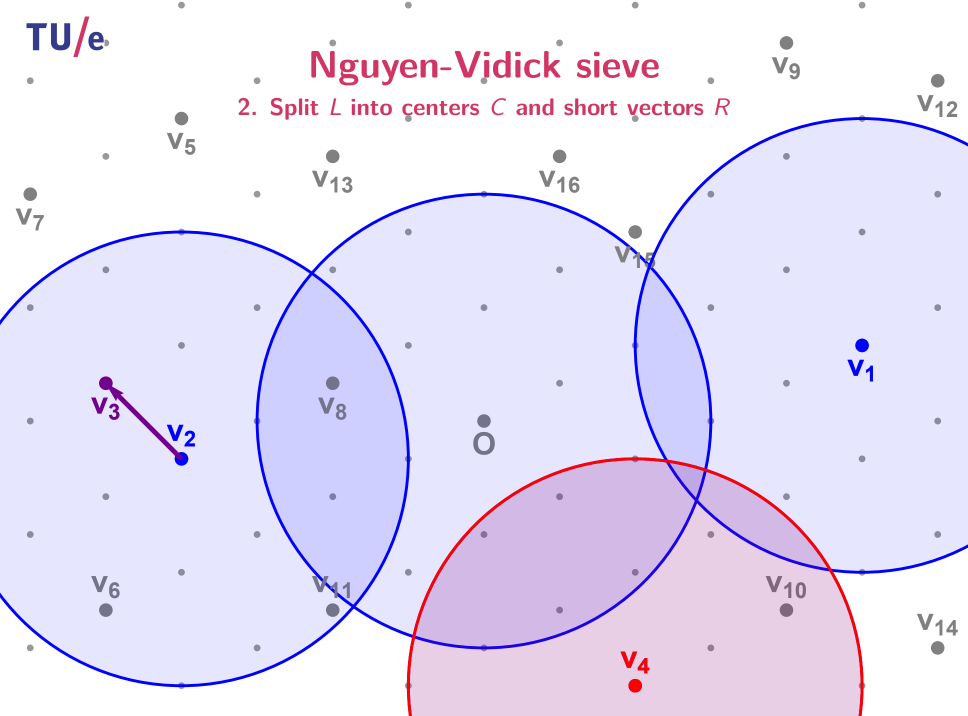
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



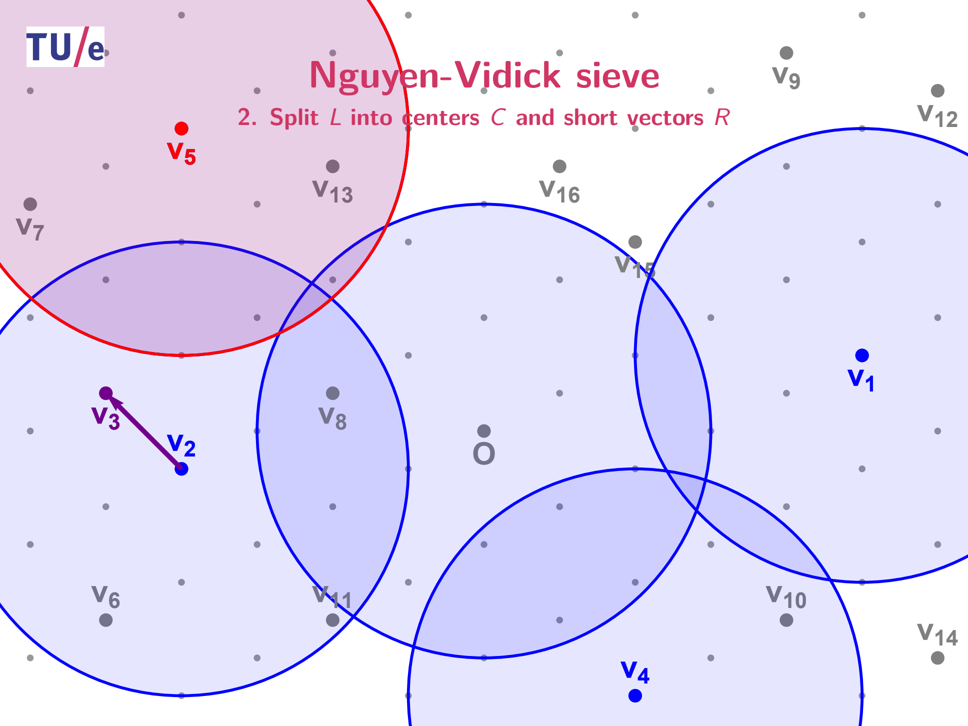
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



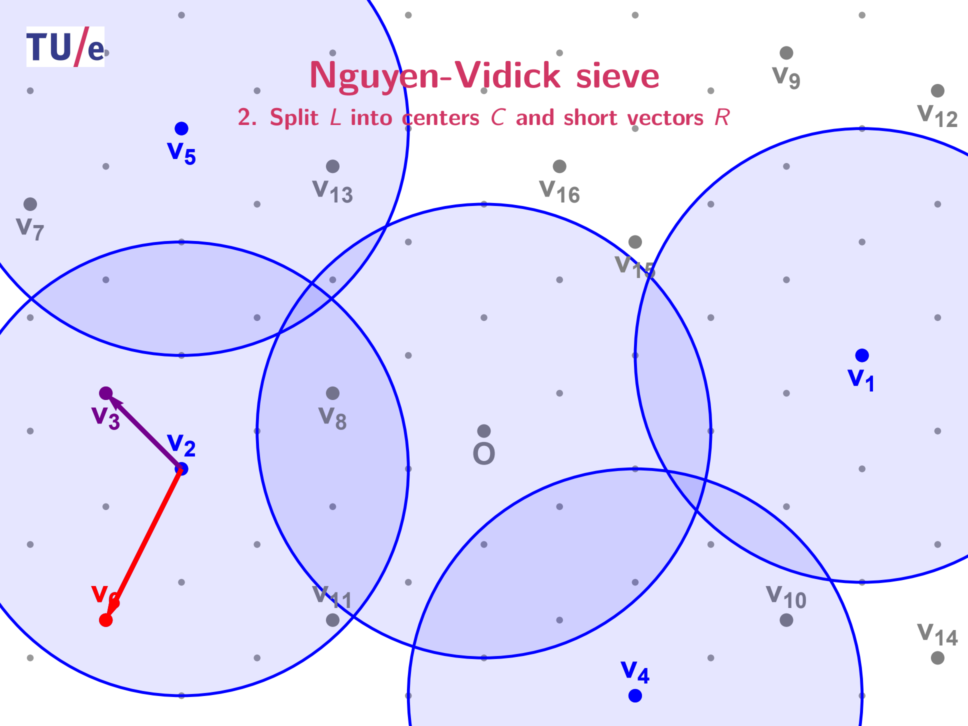
Nguyen-Vidick sieve

2. Split *L* into centers *C* and short vectors *R*



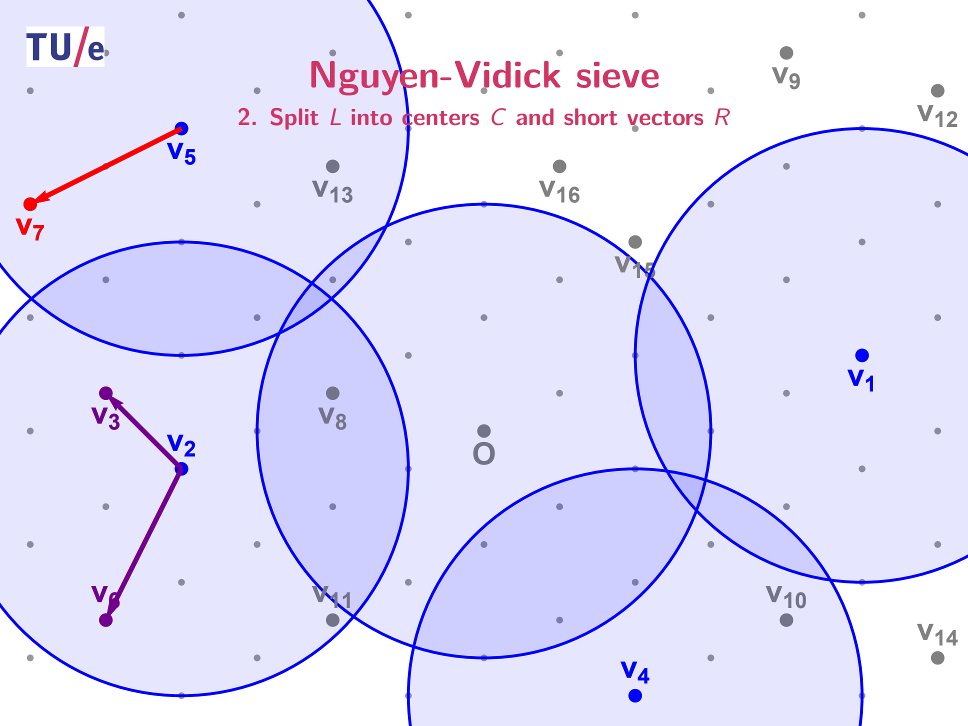
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



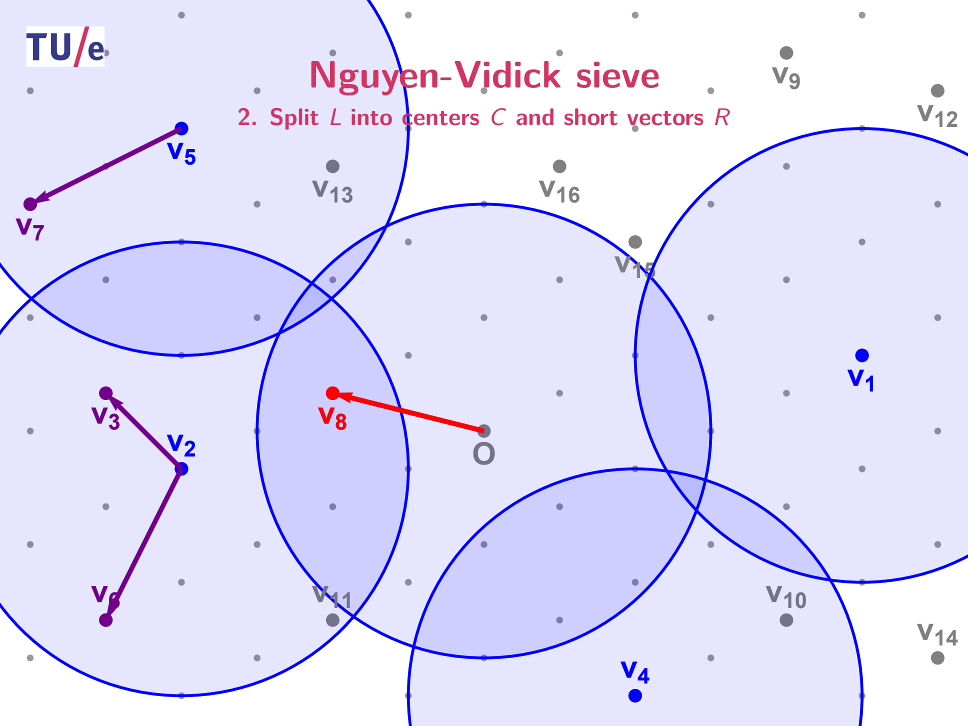
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



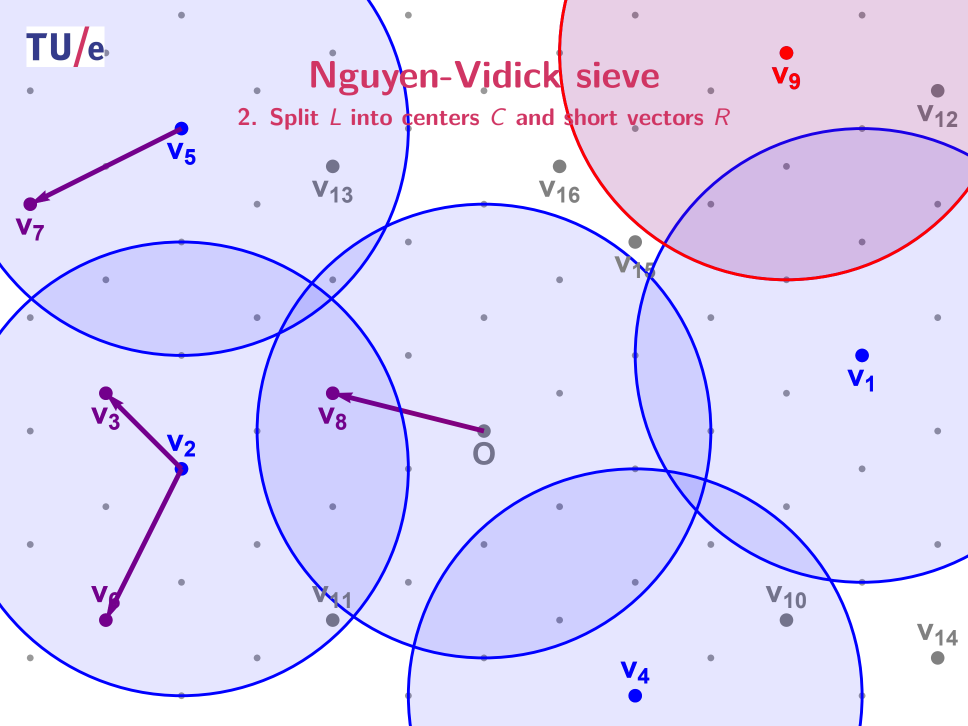
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



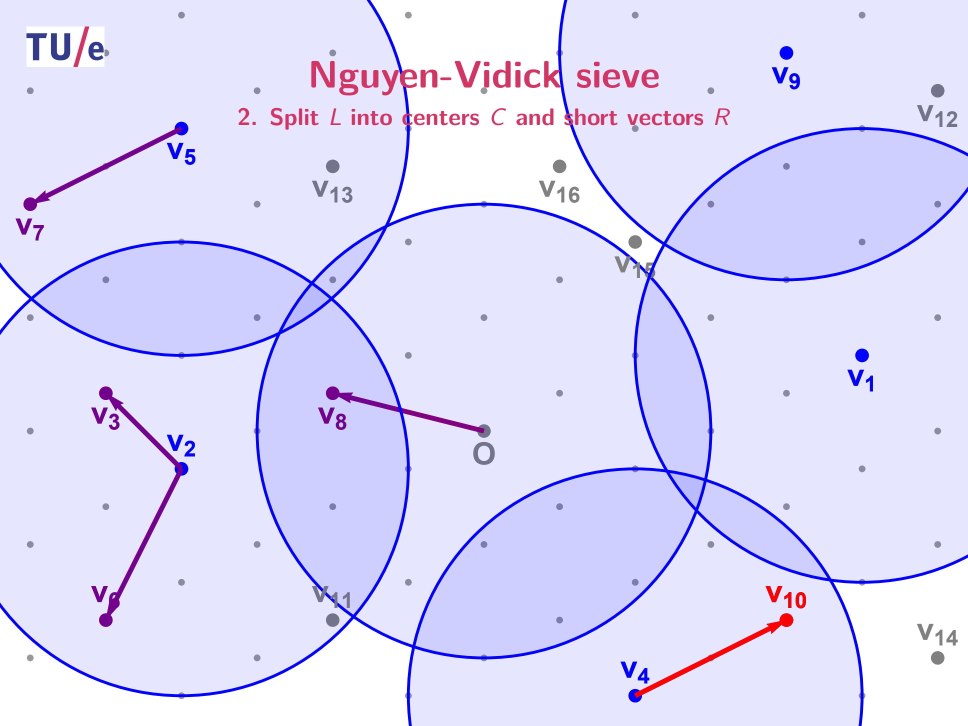
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



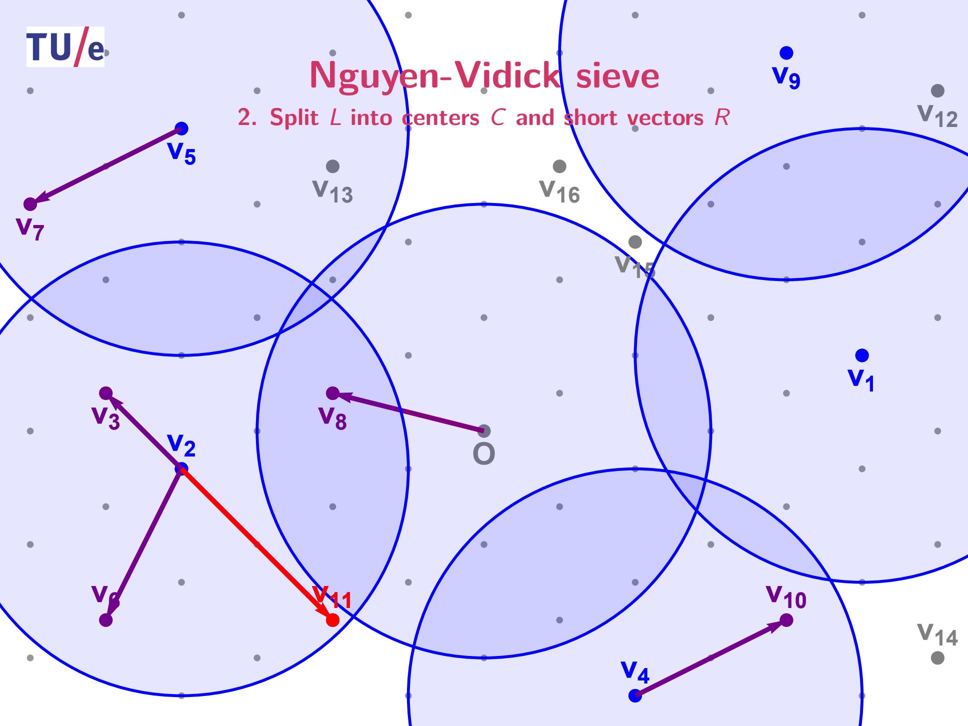
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



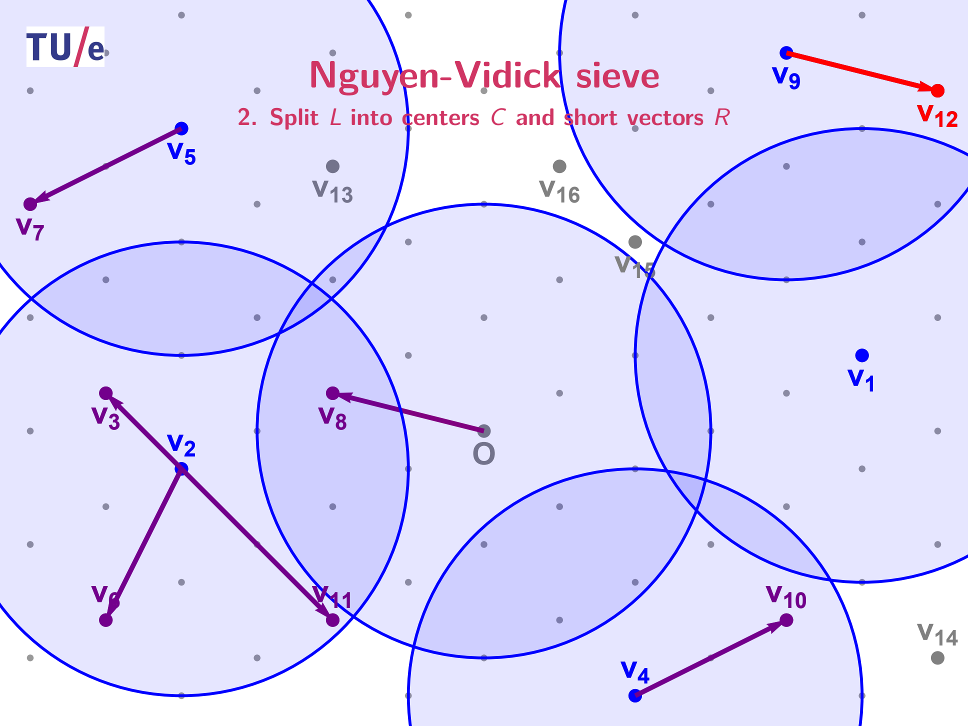
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



Nguyen-Vidick sieve

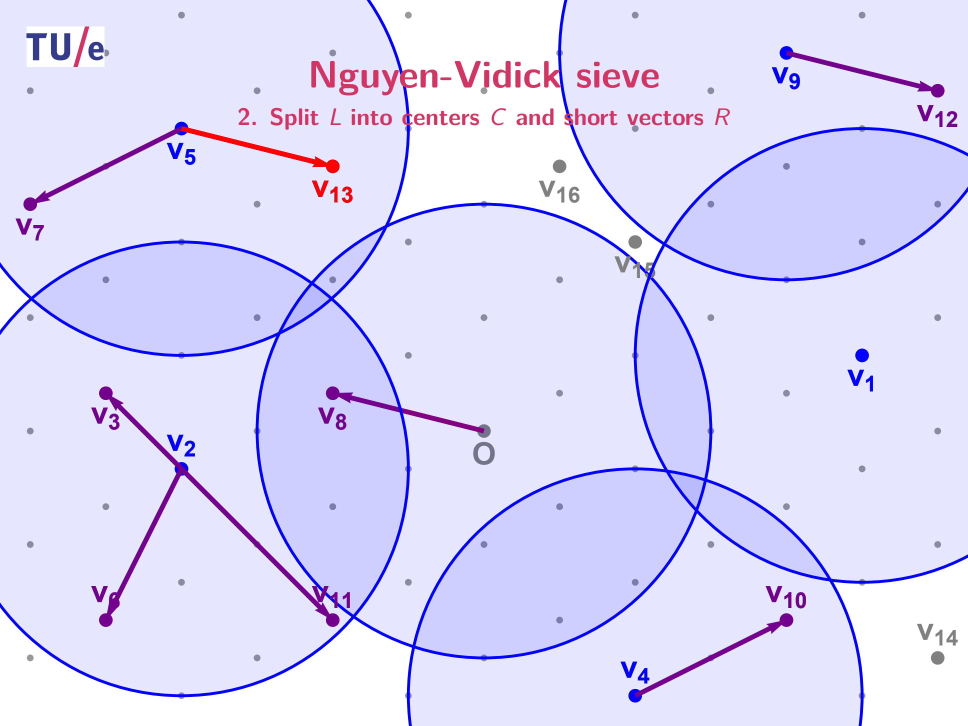
2. Split L into centers C and short vectors R



TU/e

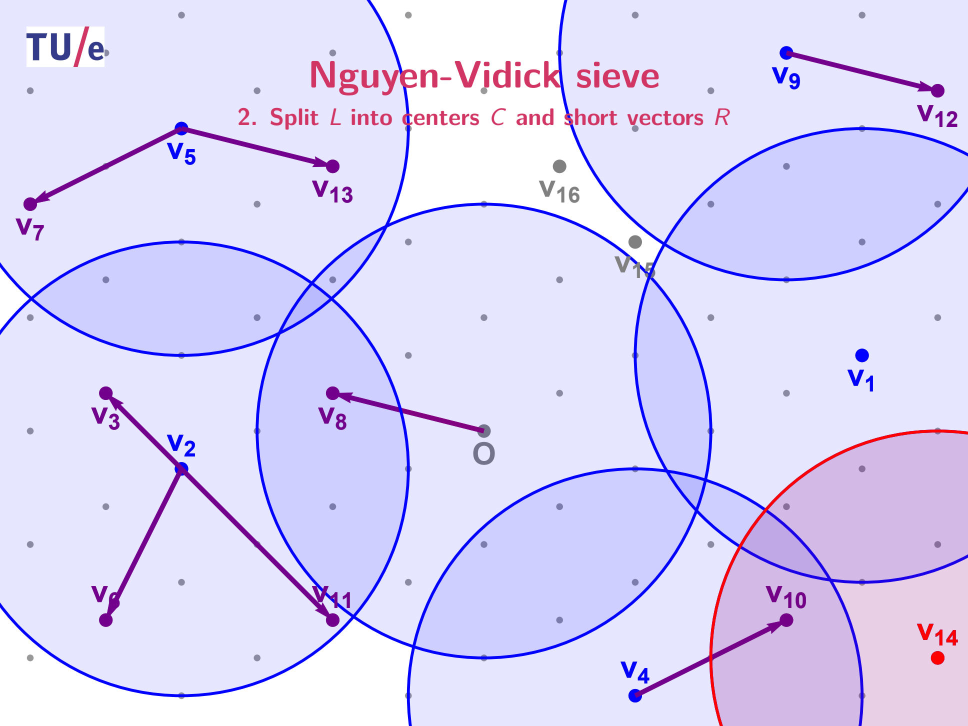
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



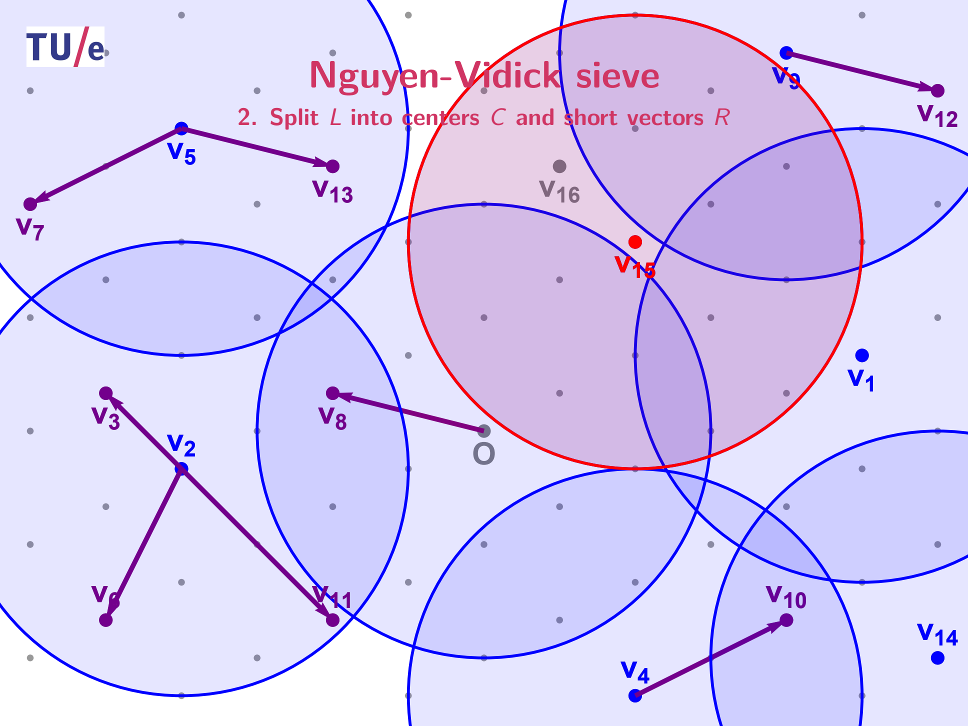
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



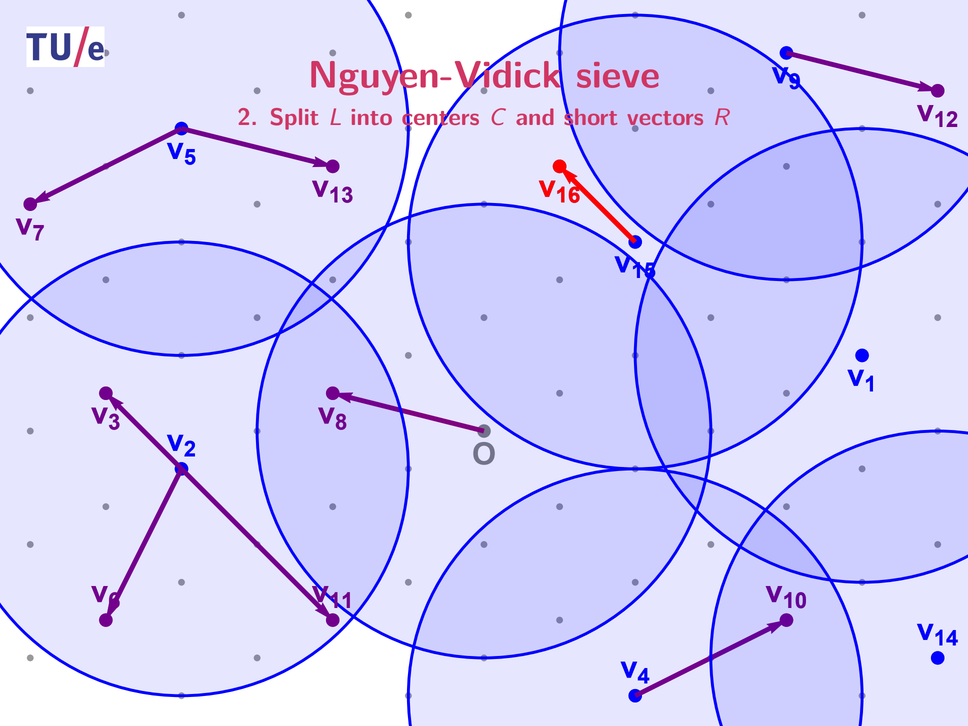
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



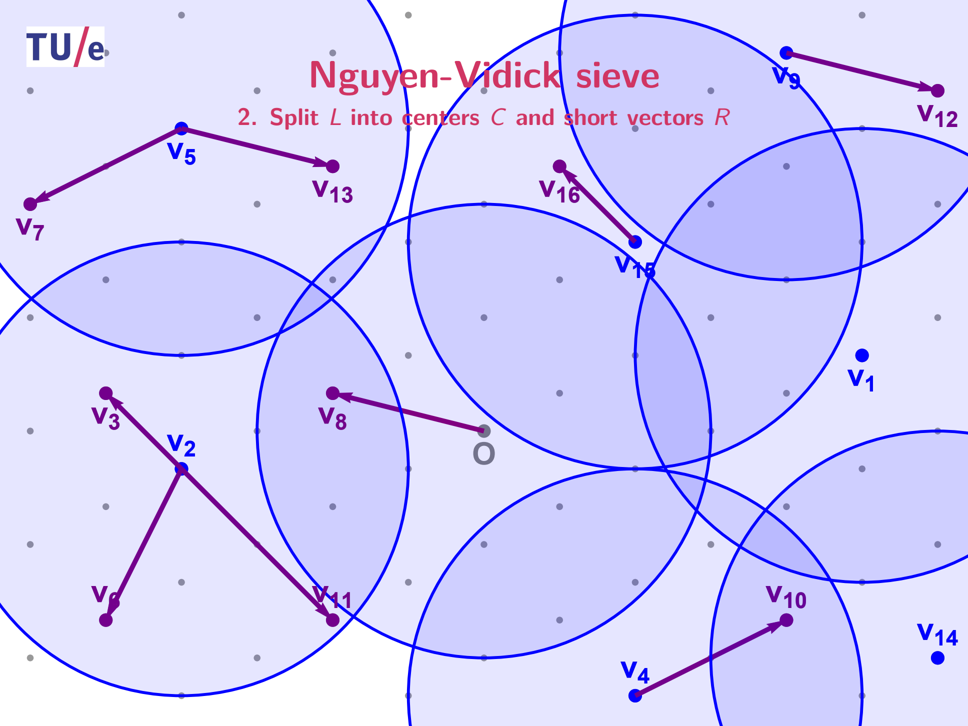
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



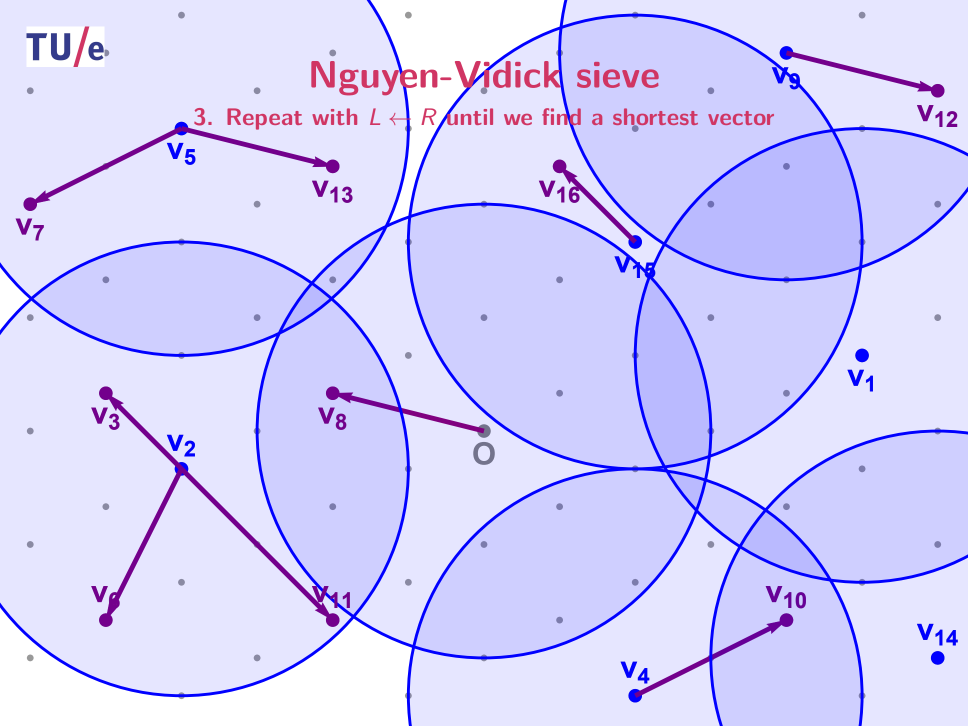
Nguyen-Vidick sieve

2. Split L into centers C and short vectors R



Nguyen-Vidick sieve

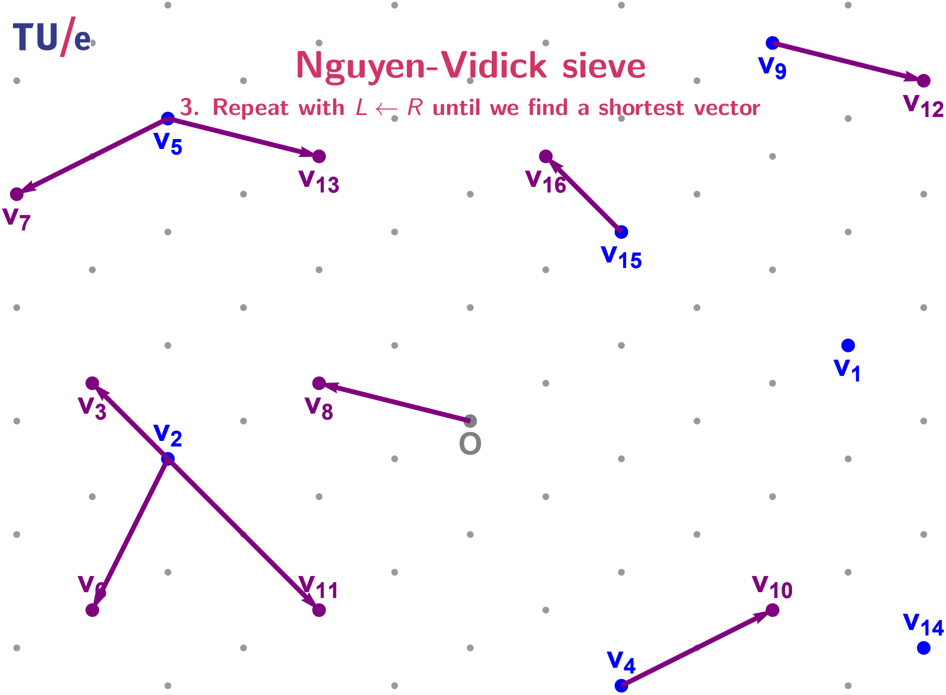
3. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

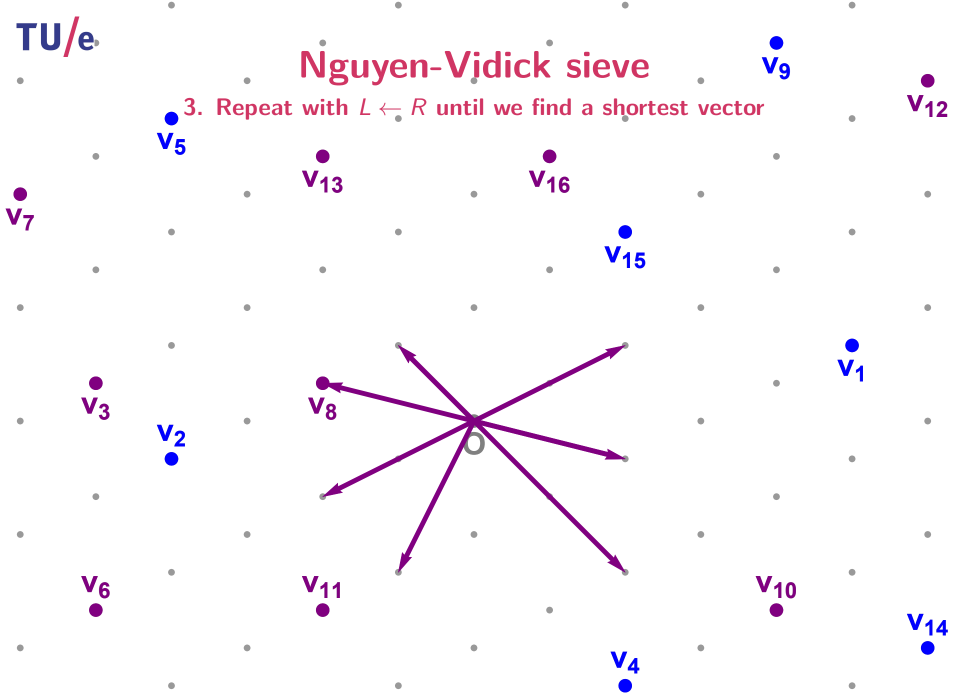
Nguyen-Vidick sieve

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

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

Nguyen-Vidick sieve

Overview

Nguyen-Vidick sieve

Overview

- **Heuristic assumption**: Normalized vectors are uniformly distributed on the unit sphere

Nguyen-Vidick sieve

Overview

- **Heuristic assumption**: Normalized vectors are uniformly distributed on the unit sphere
- Space complexity: $(\sqrt{4/3})^n \approx 2^{0.208n+o(n)}$ vectors
  - Each center covers $(\sin \frac{\pi}{3})^{-n} = (\sqrt{3/4})^n$ of the space
  - Need $(\sqrt{4/3})^{n+o(n)}$ vectors to cover all corners of $\mathbb{R}^n$

Nguyen-Vidick sieve

Overview

- **Heuristic assumption**: Normalized vectors are uniformly distributed on the unit sphere
- Space complexity: $(\sqrt{4/3})^n \approx 2^{0.208n+o(n)}$ vectors
  - Each center covers $(\sin \frac{\pi}{3})^{-n} = (\sqrt{3/4})^n$ of the space
  - Need $(\sqrt{4/3})^{n+o(n)}$ vectors to cover all corners of $\mathbb{R}^n$
- Time complexity: $(4/3)^n \approx 2^{0.415n+o(n)}$

Nguyen-Vidick sieve

Overview

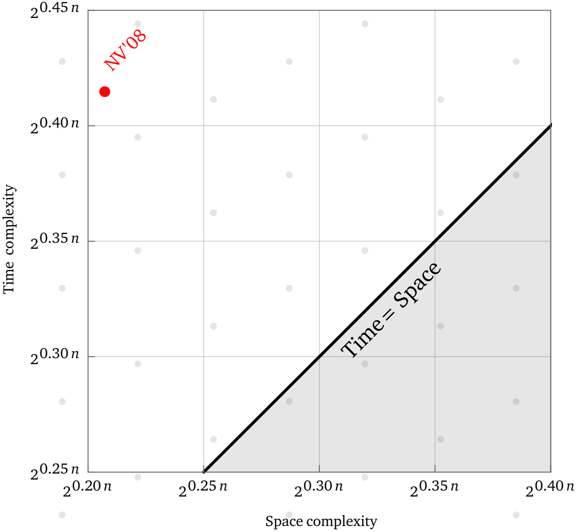
- **Heuristic assumption**: Normalized vectors are uniformly distributed on the unit sphere
- Space complexity: $(\sqrt{4/3})^n \approx 2^{0.208n+o(n)}$ vectors
  - Each center covers $(\sin \frac{\pi}{3})^{-n} = (\sqrt{3/4})^n$ of the space
  - Need $(\sqrt{4/3})^{n+o(n)}$ vectors to cover all corners of $\mathbb{R}^n$
- Time complexity: $(4/3)^n \approx 2^{0.415n+o(n)}$

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

The Nguyen-Vidick sieve heuristically solves SVP in time $2^{0.415n+o(n)}$ and space $2^{0.208n+o(n)}$.

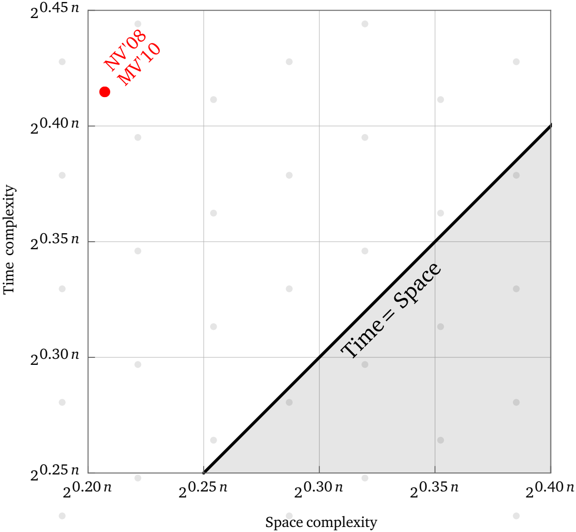
Nguyen-Vidick sieve

Space/time trade-off



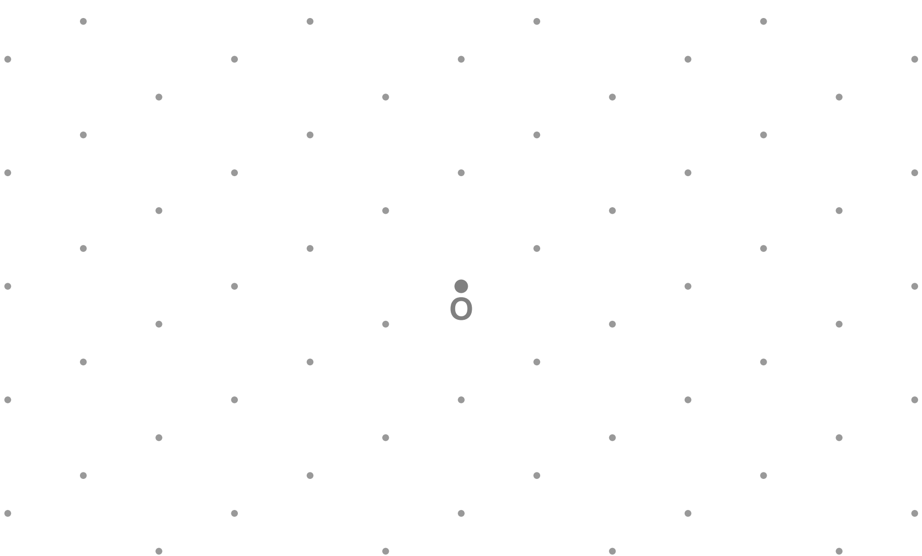
GaussSieve

Space/time trade-off



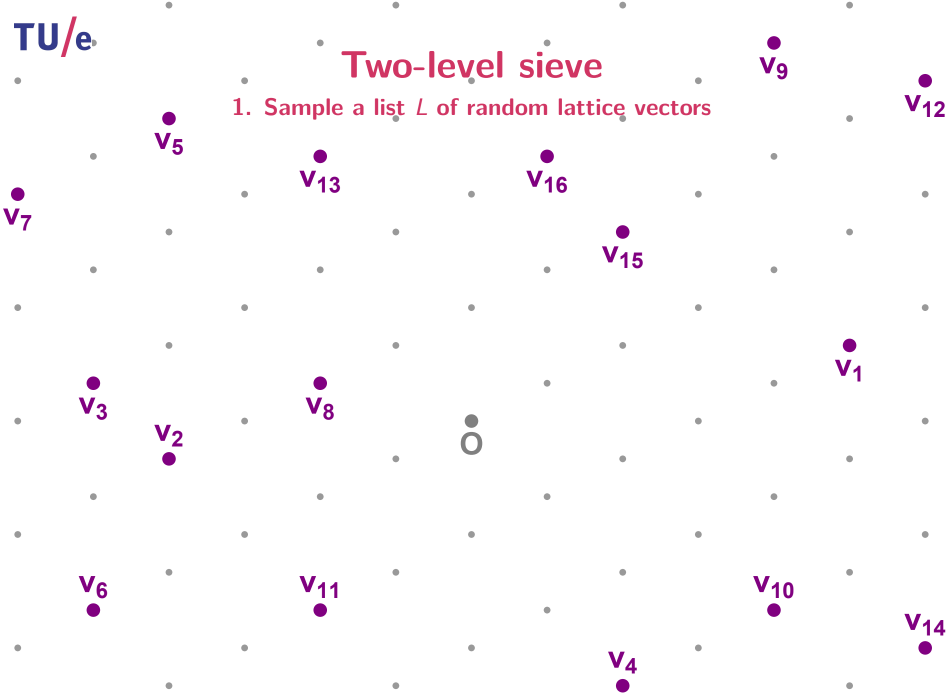
Two-level sieve

1. Sample a list L of random lattice vectors



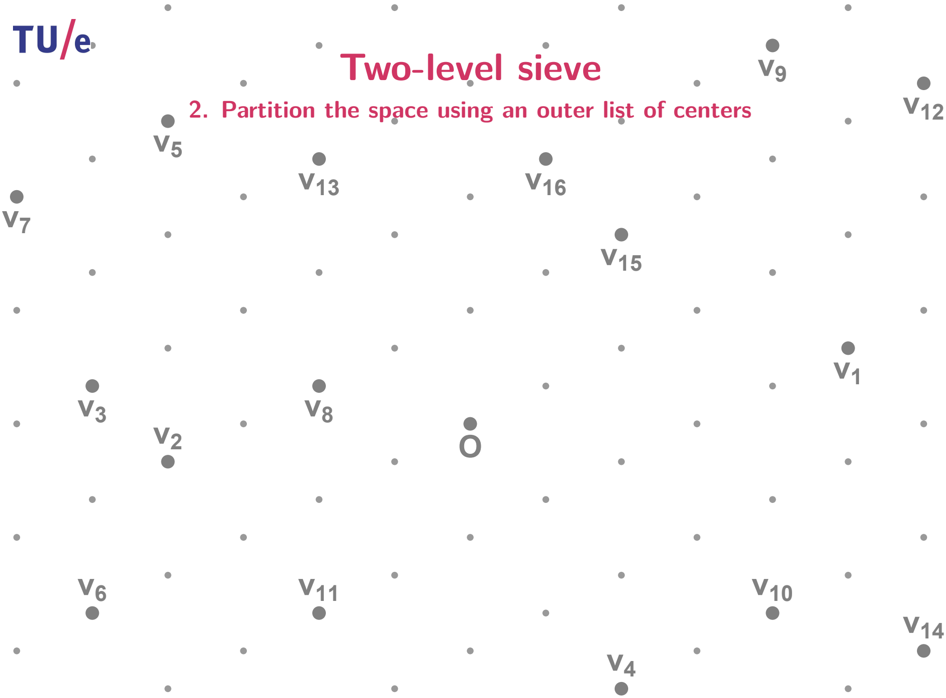
Two-level sieve

1. Sample a list L of random lattice vectors



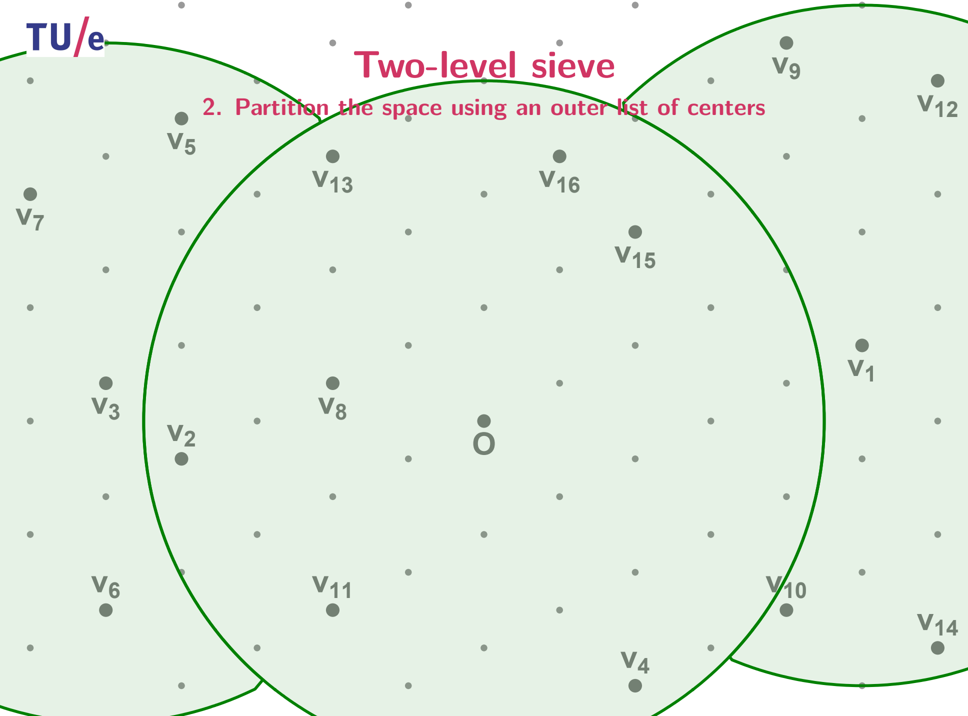
Two-level sieve

2. Partition the space using an outer list of centers



Two-level sieve

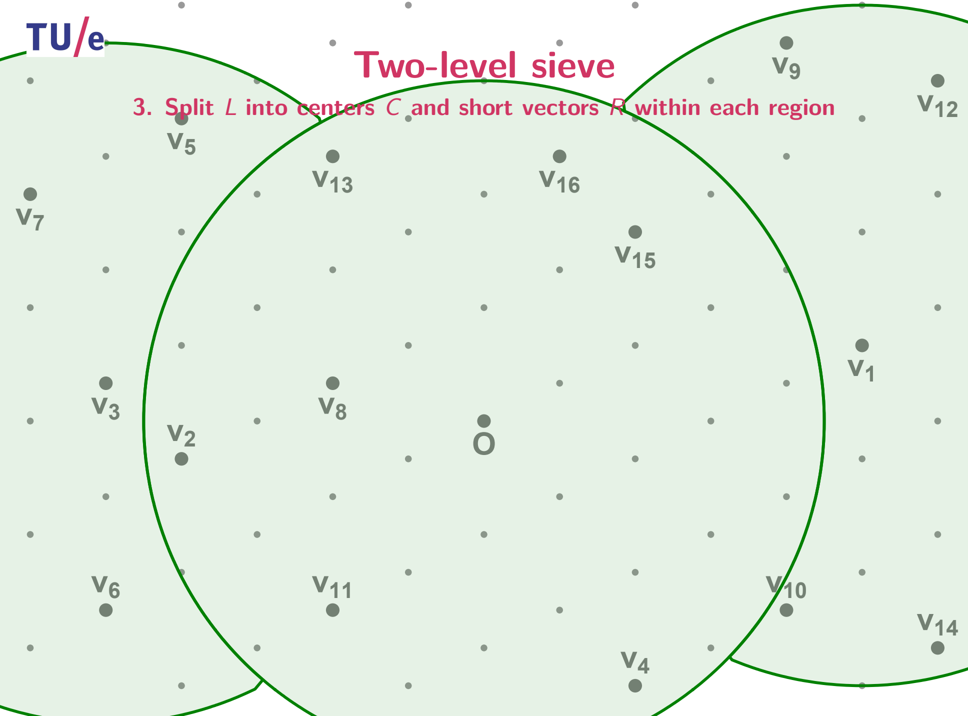
2. Partition the space using an outer list of centers



TU/e

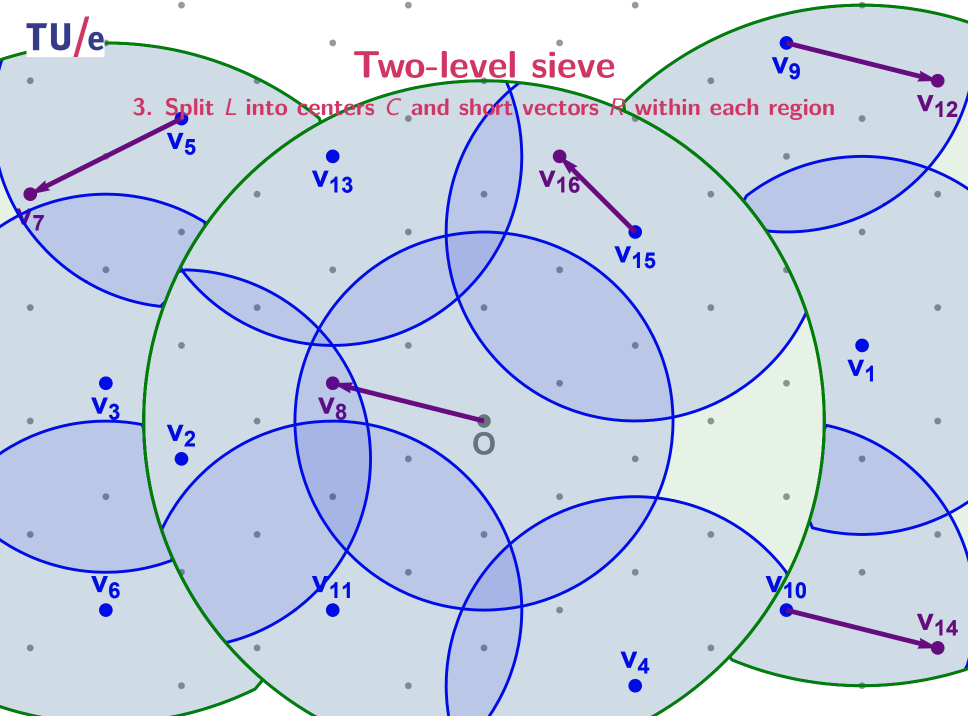
Two-level sieve

3. Split L into centers C and short vectors R within each region



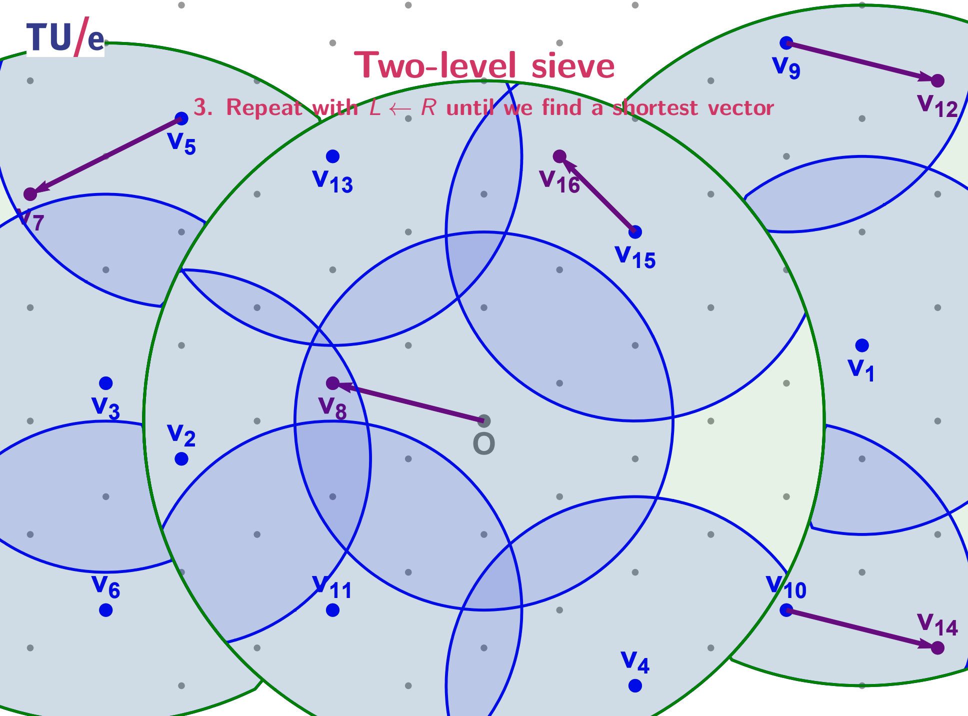
Two-level sieve

3. Split L into centers C and short vectors R within each region



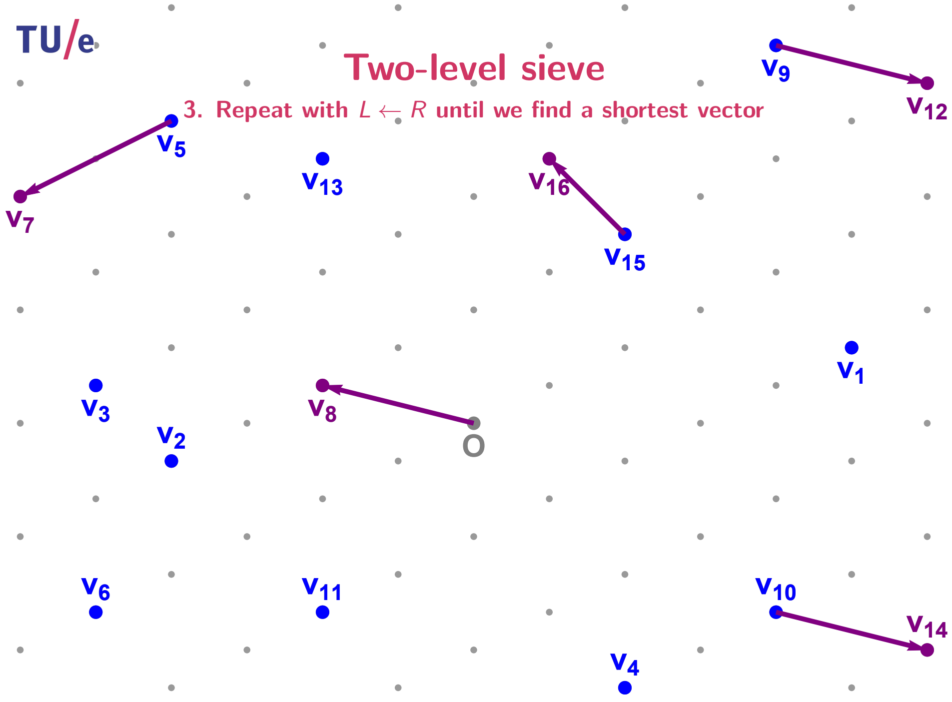
Two-level sieve

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



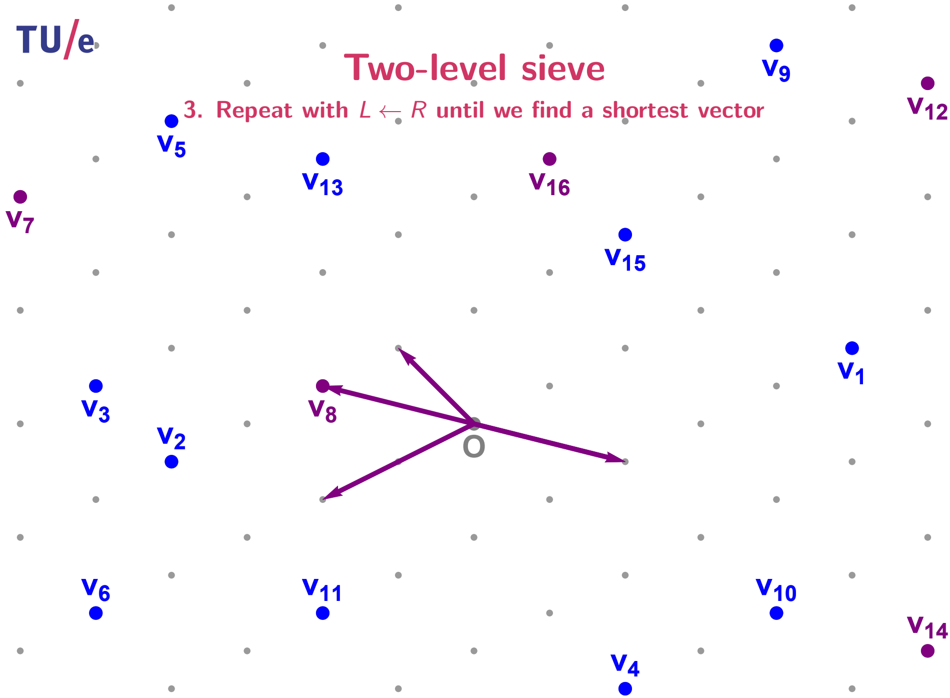
Two-level sieve

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



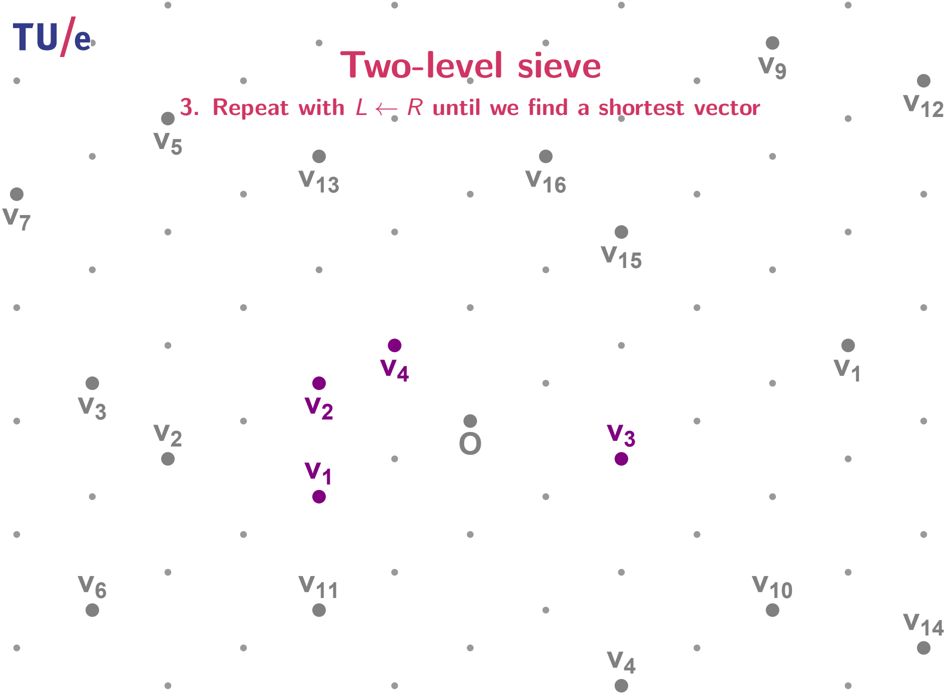
Two-level sieve

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



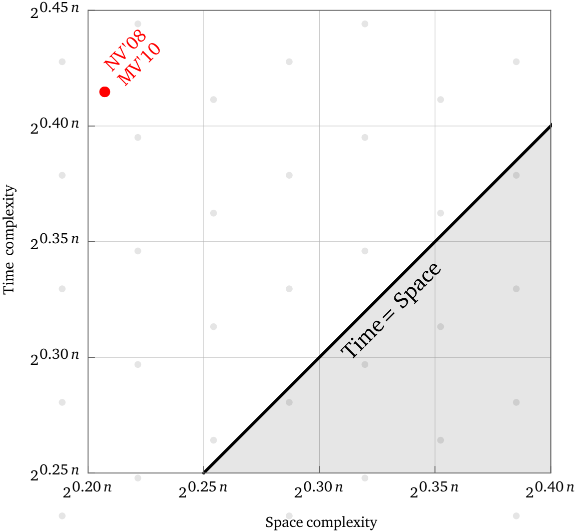
Two-level sieve

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



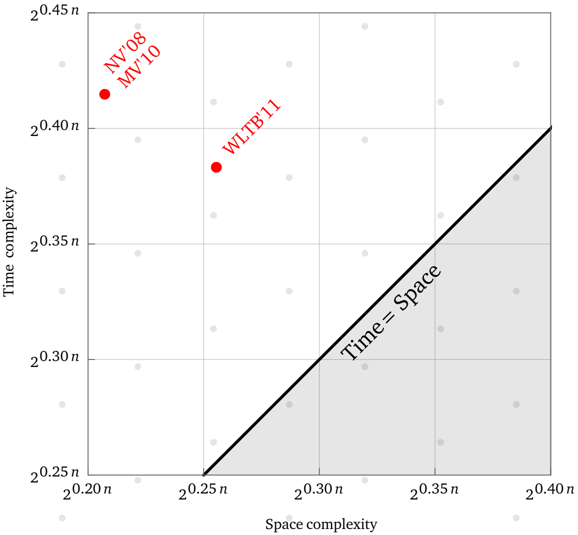
Two-level sieve

Space/time trade-off



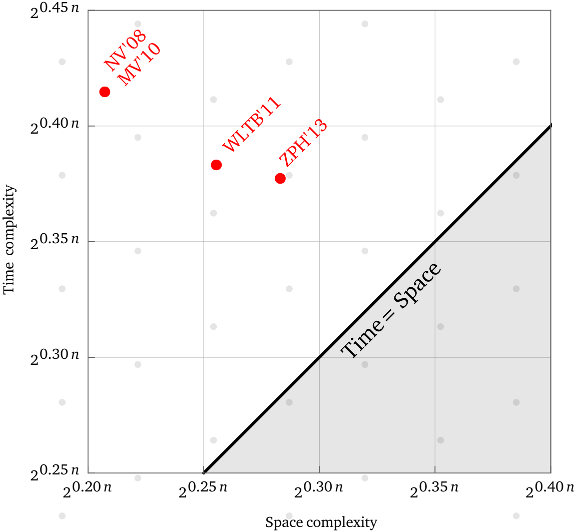
Two-level sieve

Space/time trade-off



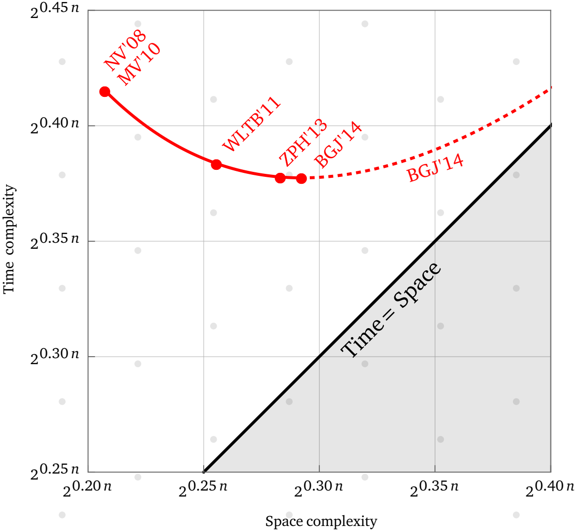
Three-level sieve

Space/time trade-off



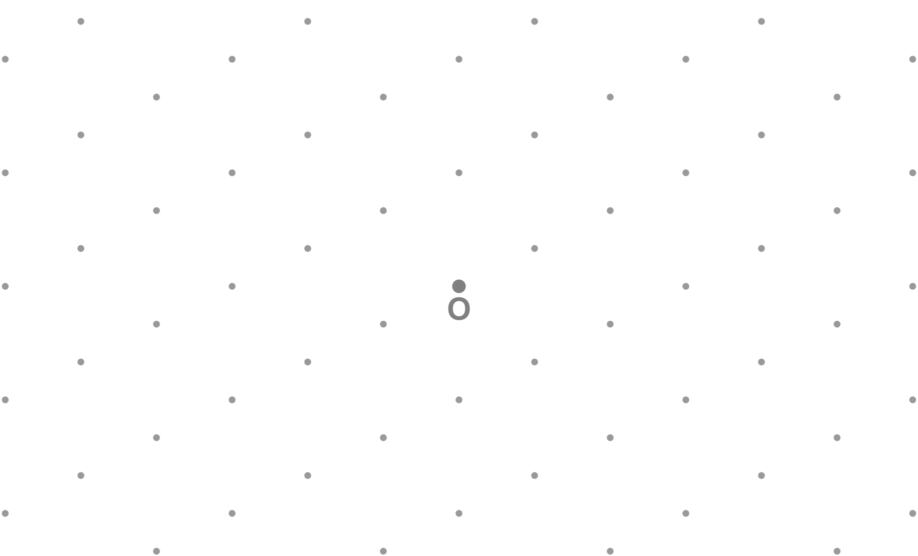
Overlattice sieving

Space/time trade-off



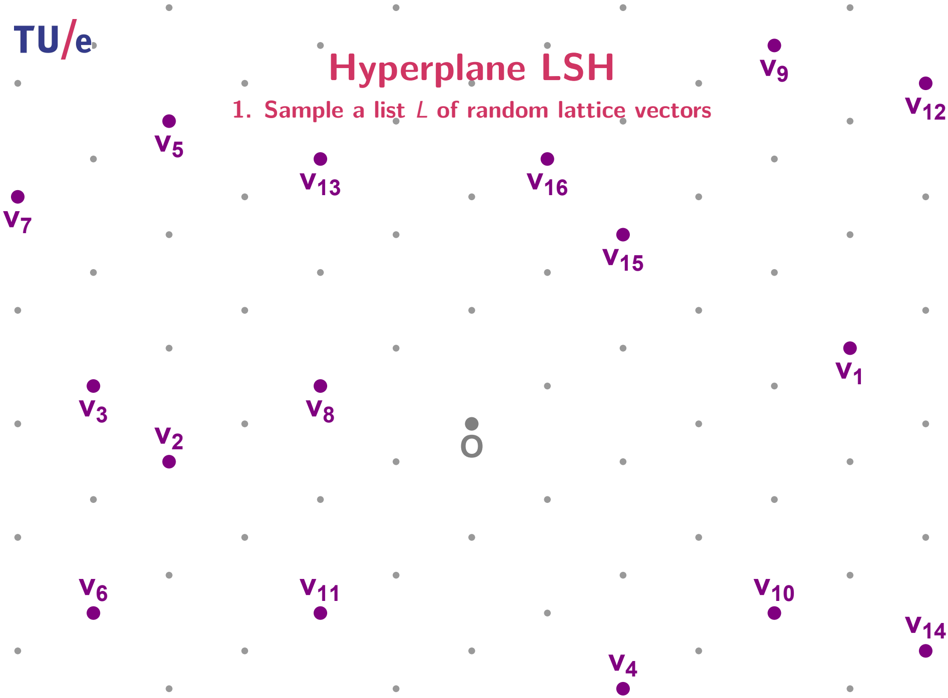
Hyperplane LSH

1. Sample a list L of random lattice vectors



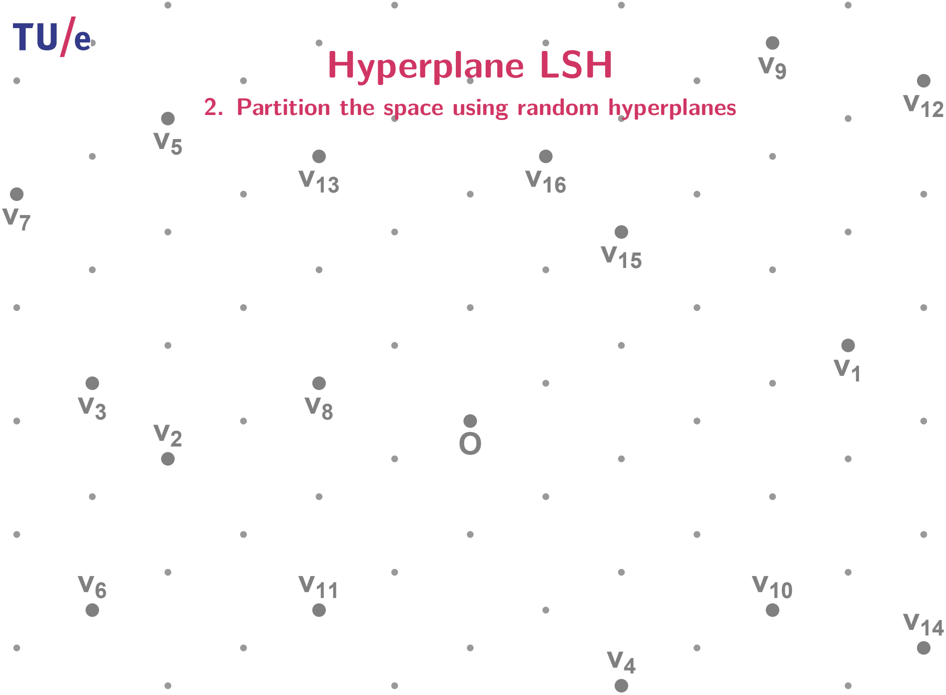
Hyperplane LSH

1. Sample a list L of random lattice vectors



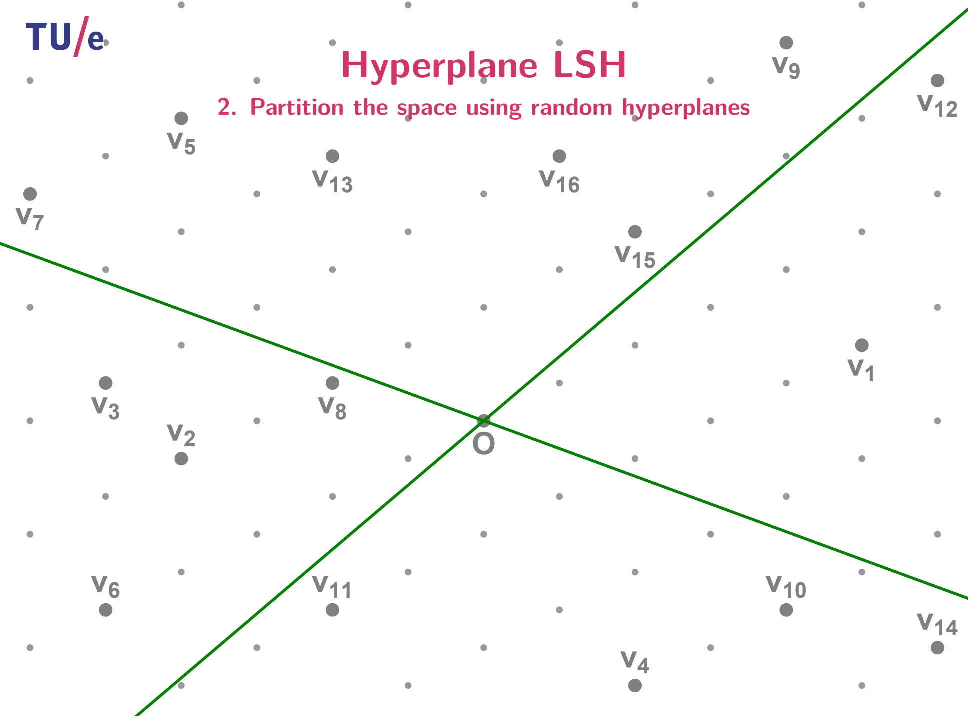
Hyperplane LSH

2. Partition the space using random hyperplanes



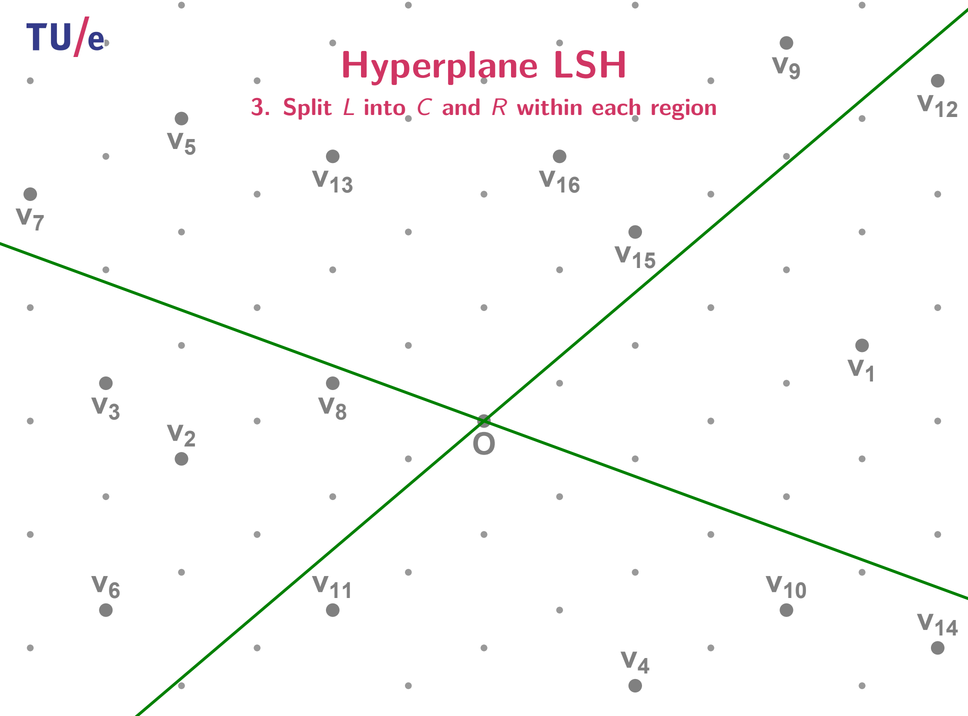
Hyperplane LSH

2. Partition the space using random hyperplanes



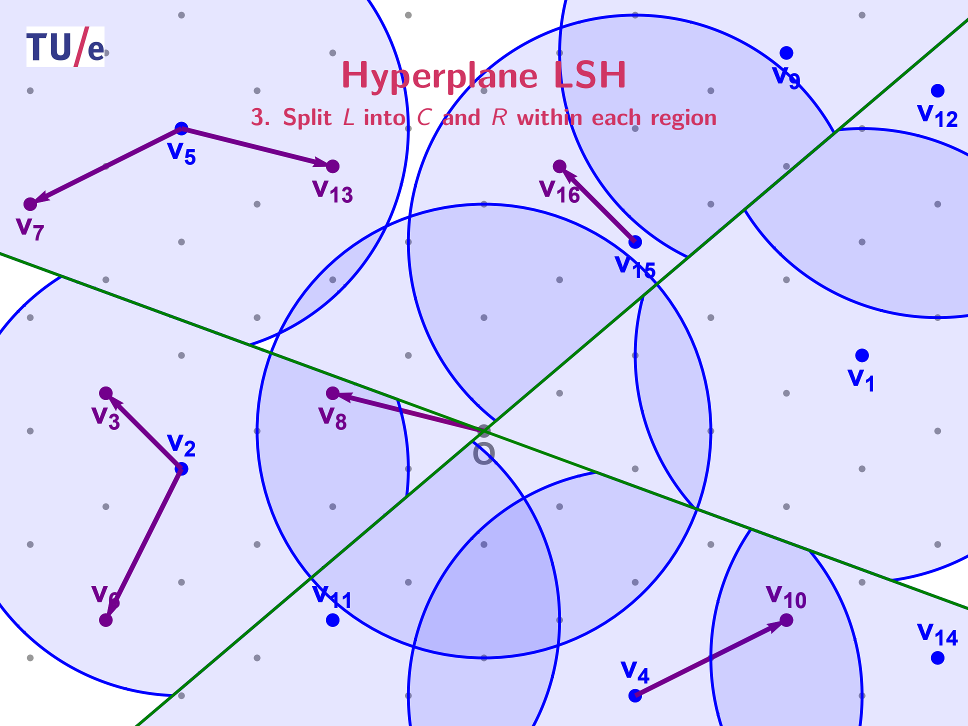
Hyperplane LSH

3. Split L into C and R within each region



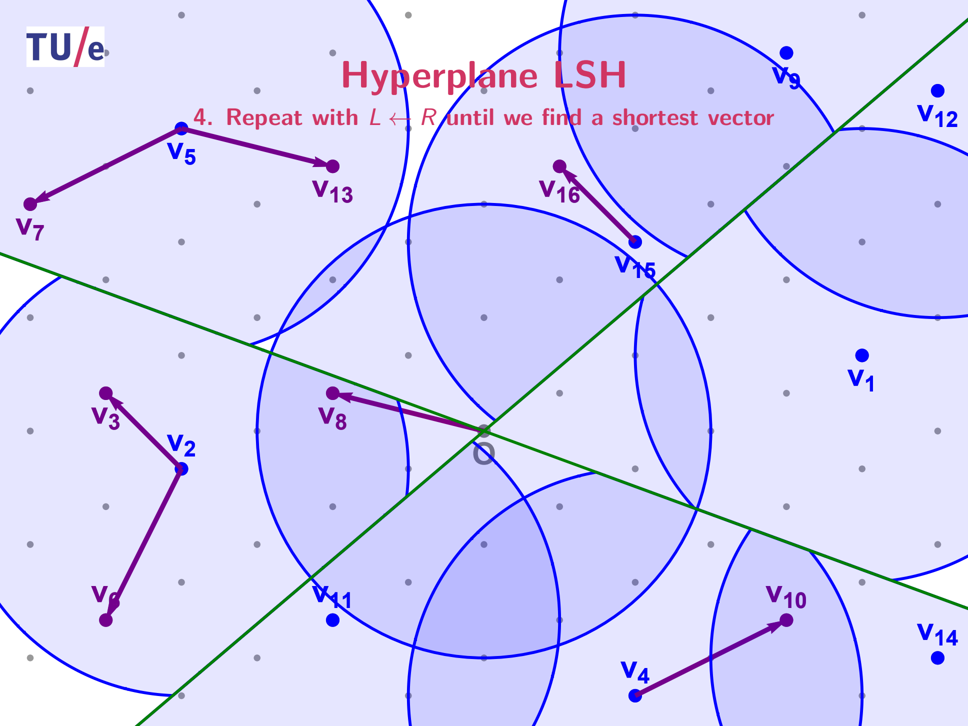
Hyperplane LSH

3. Split L into C and R within each region



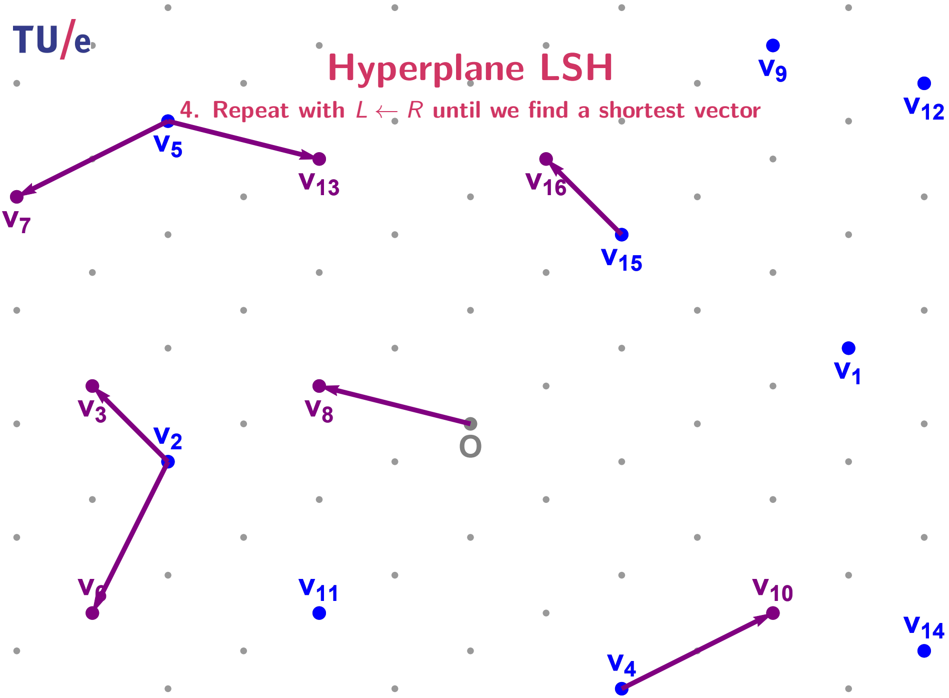
Hyperplane LSH

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



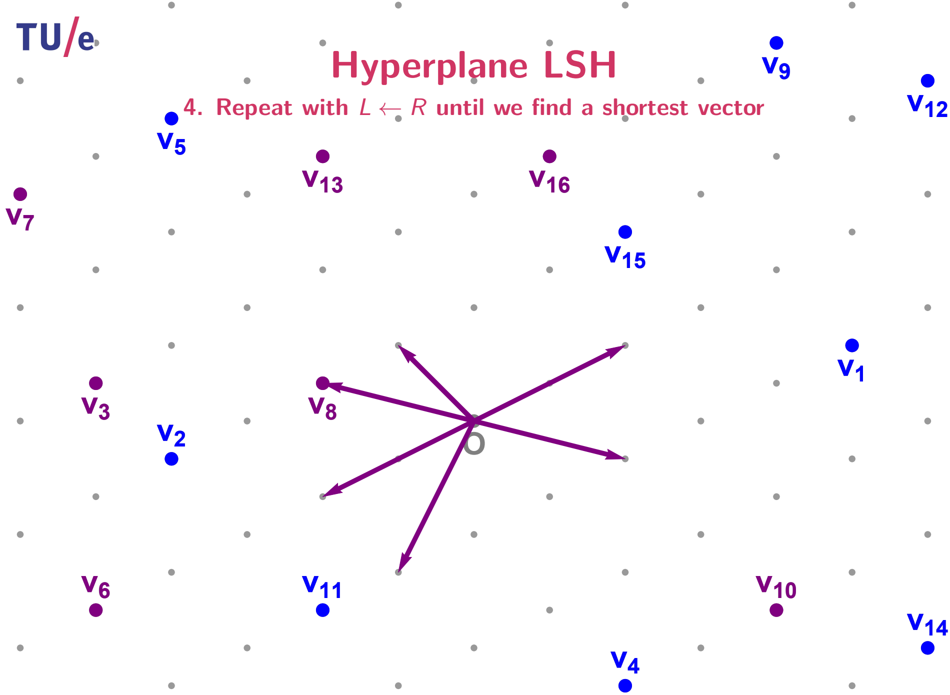
Hyperplane LSH

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



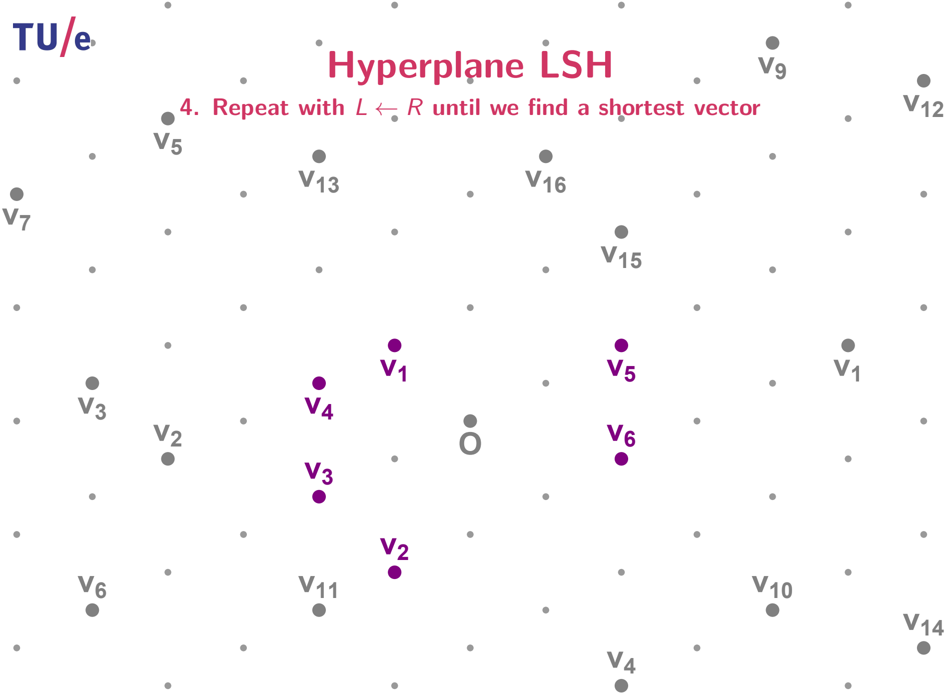
Hyperplane LSH

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



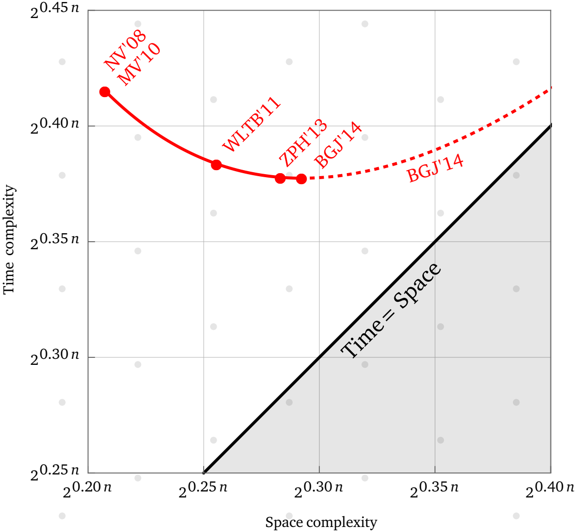
Hyperplane LSH

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



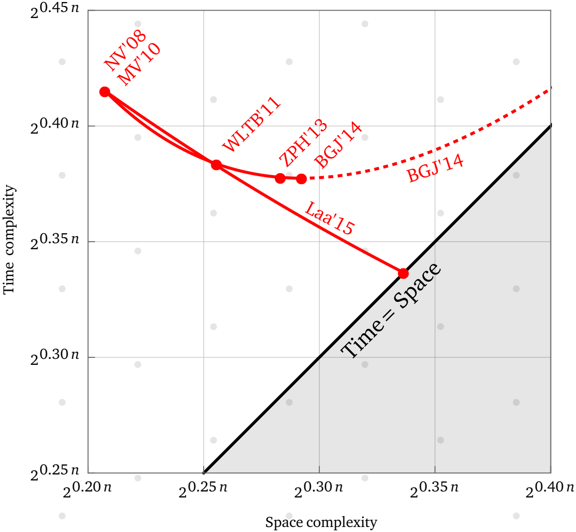
Hyperplane LSH

Space/time trade-off



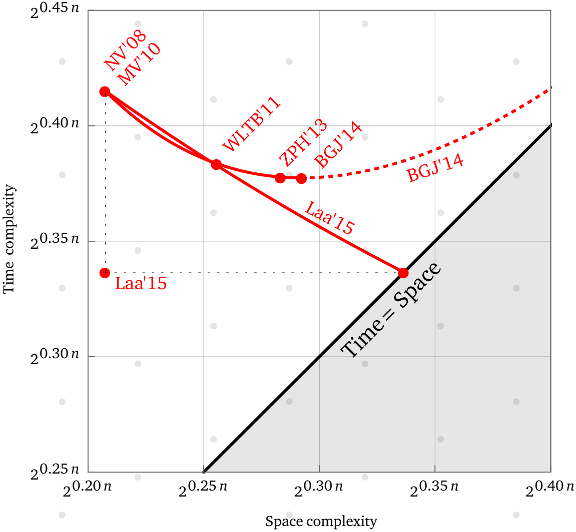
Hyperplane LSH

Space/time trade-off



Hyperplane LSH

Space/time trade-off



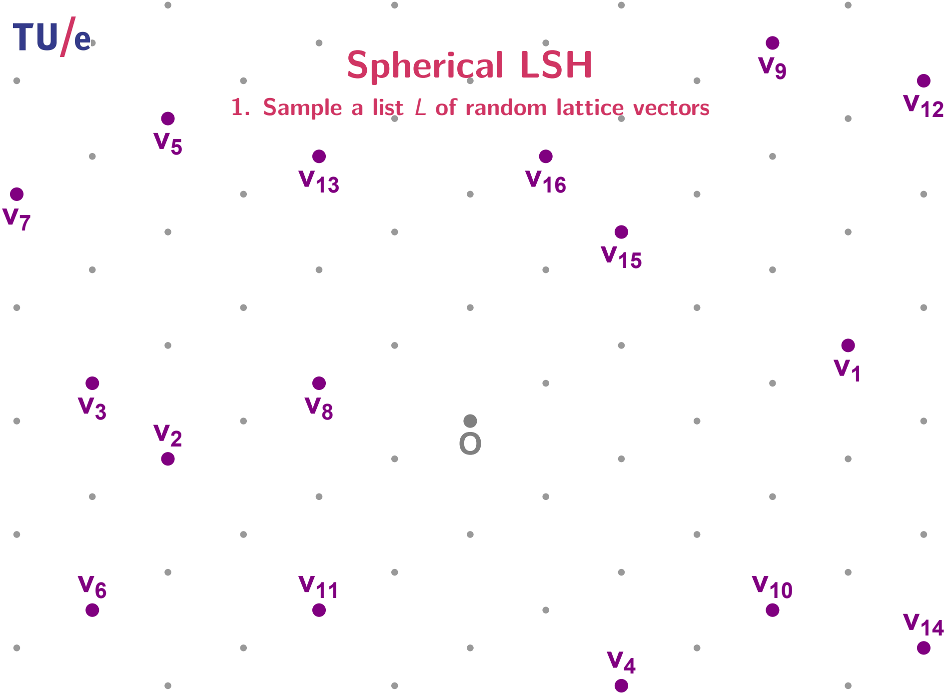
Spherical LSH

1. Sample a list L of random lattice vectors

TU/e

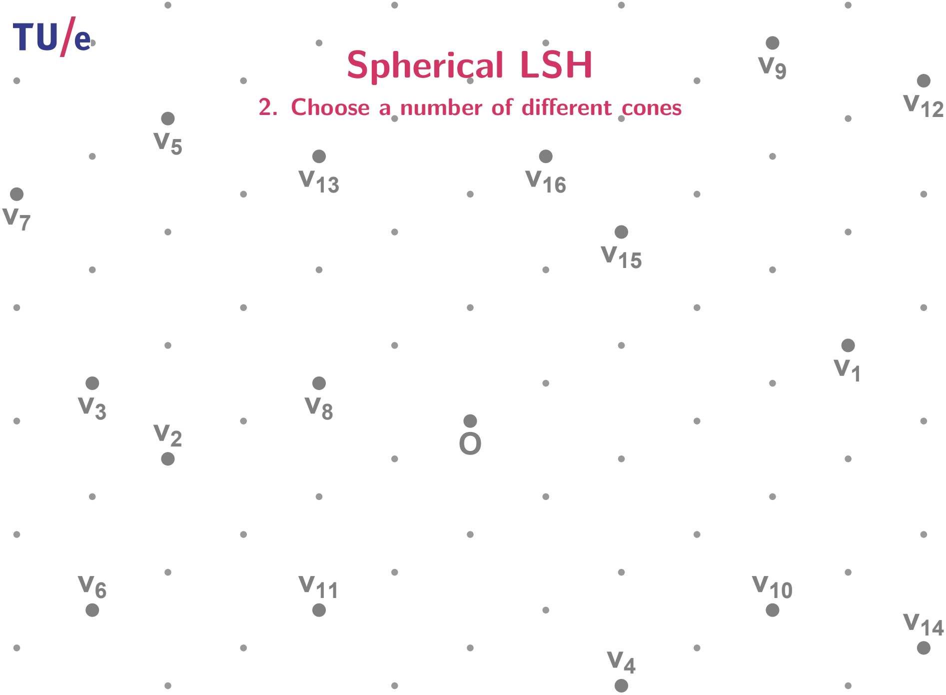
Spherical LSH

1. Sample a list L of random lattice vectors



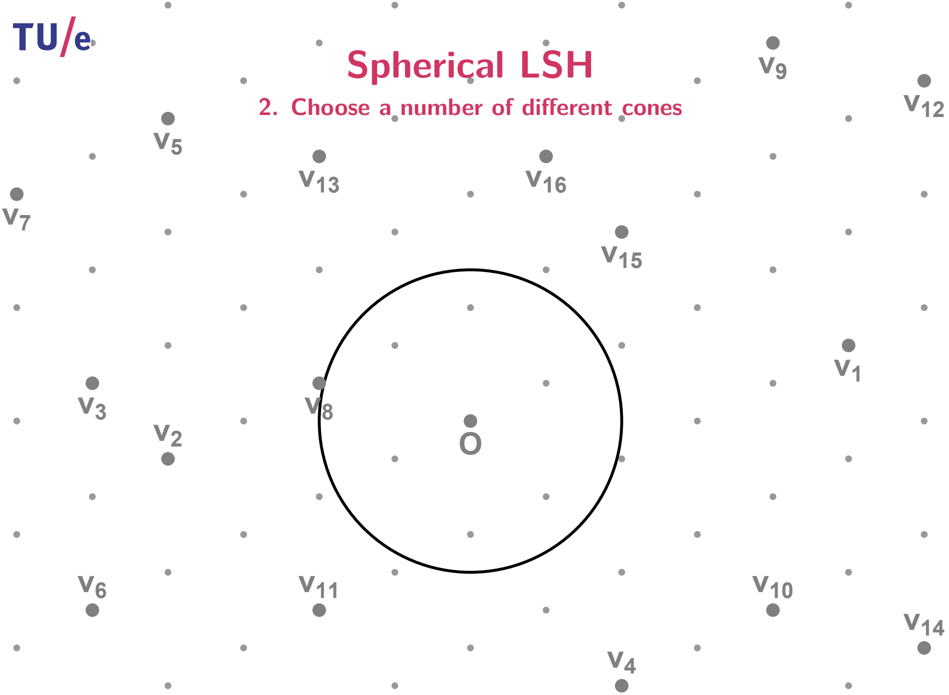
Spherical LSH

2. Choose a number of different cones



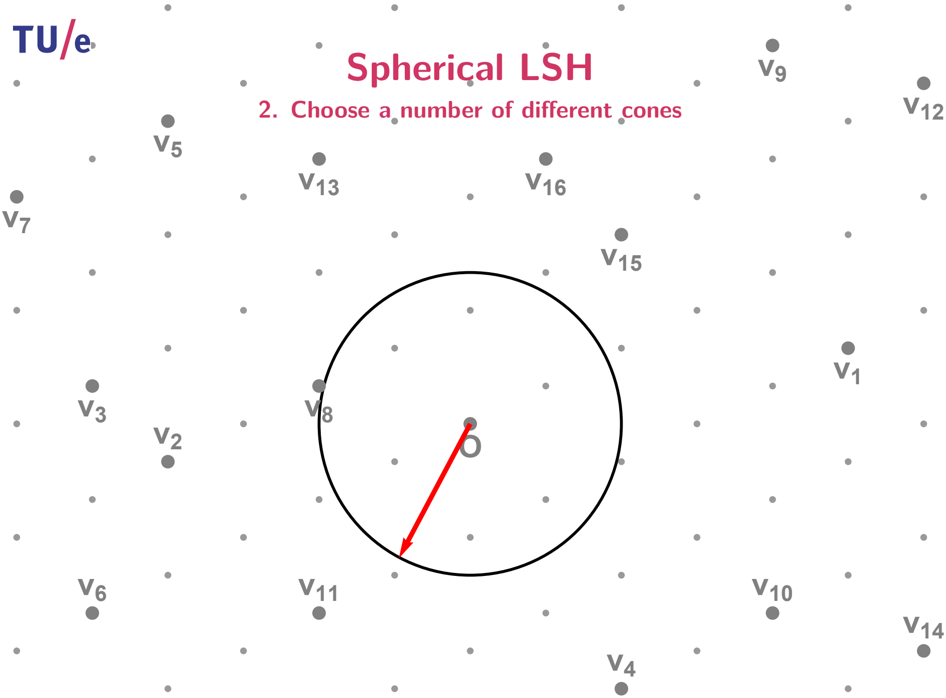
Spherical LSH

2. Choose a number of different cones



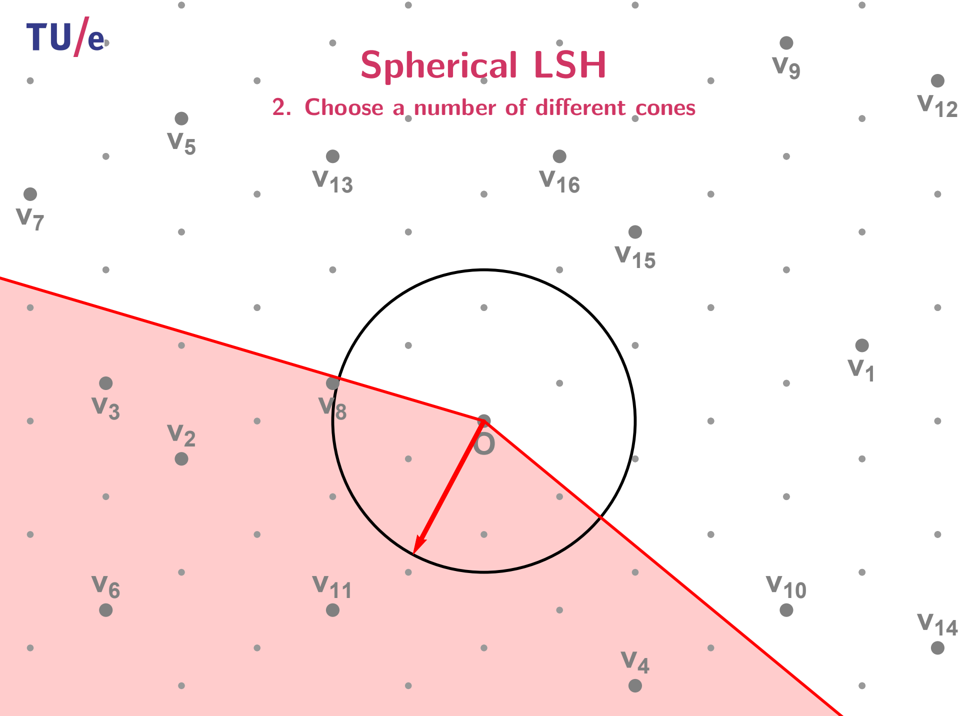
Spherical LSH

2. Choose a number of different cones



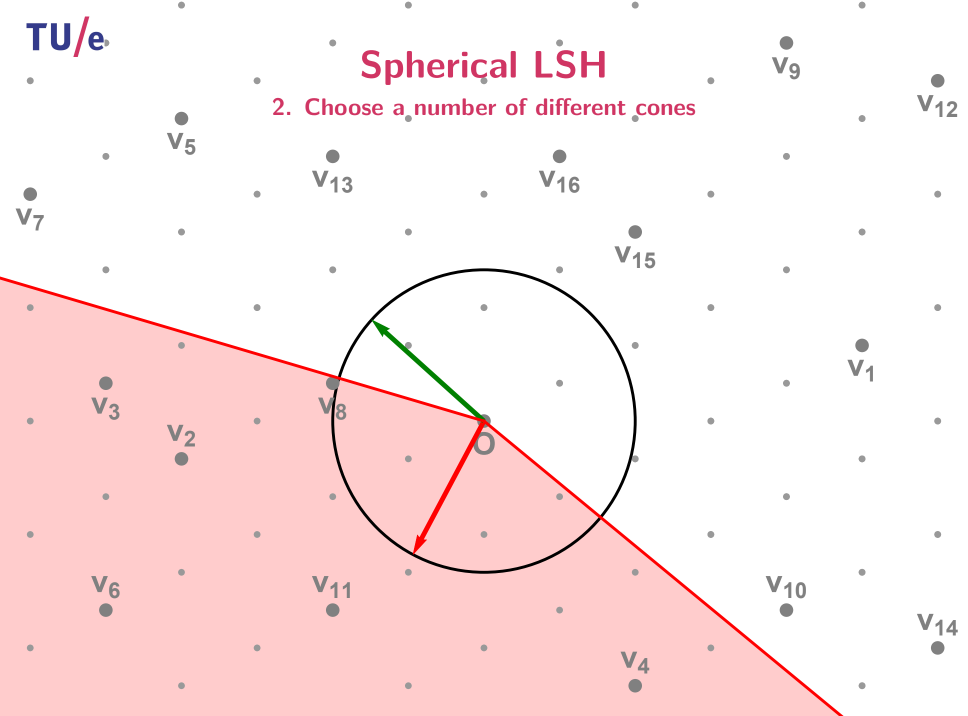
Spherical LSH

2. Choose a number of different cones



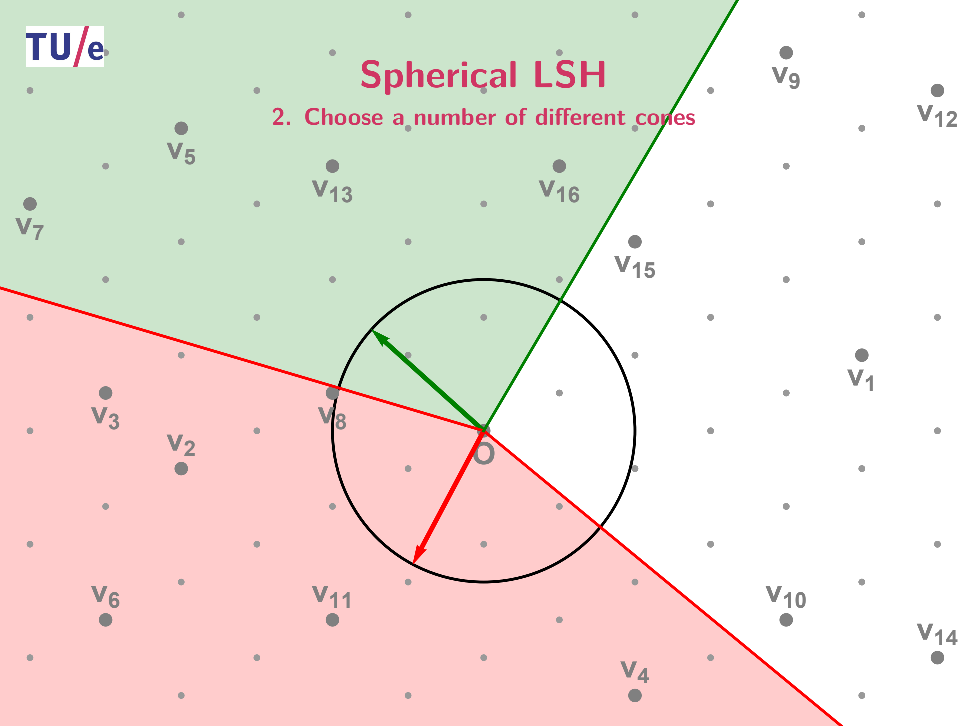
Spherical LSH

2. Choose a number of different cones



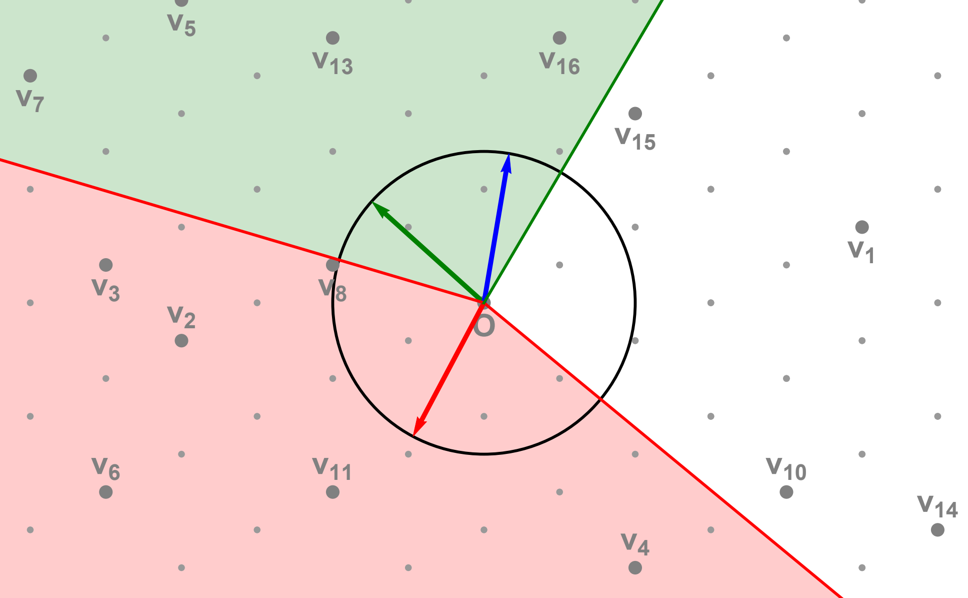
Spherical LSH

2. Choose a number of different cones



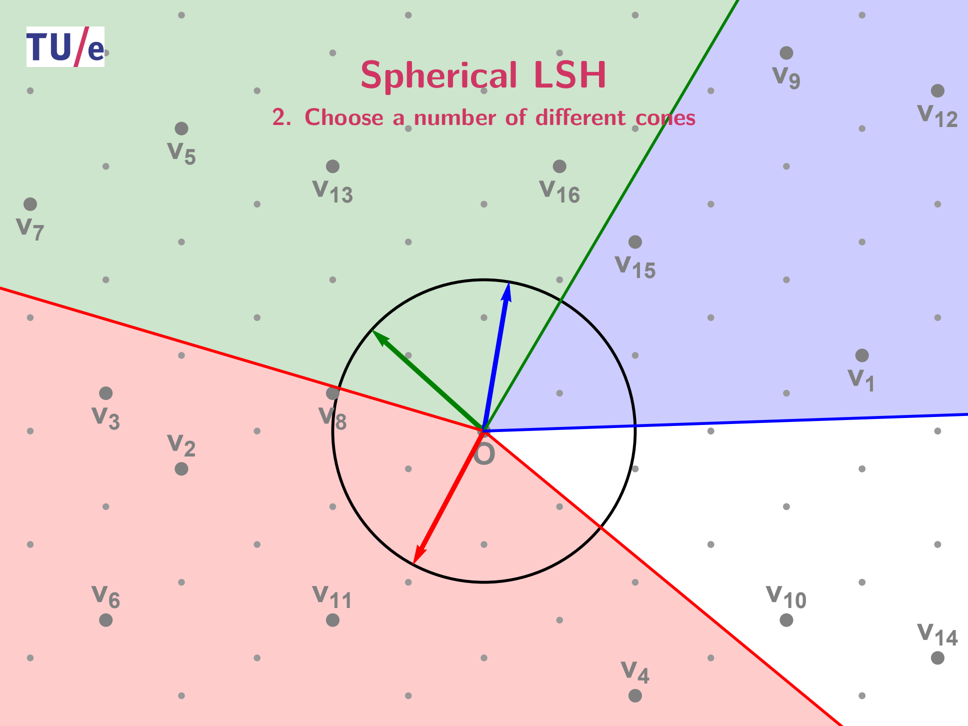
Spherical LSH

2. Choose a number of different cones



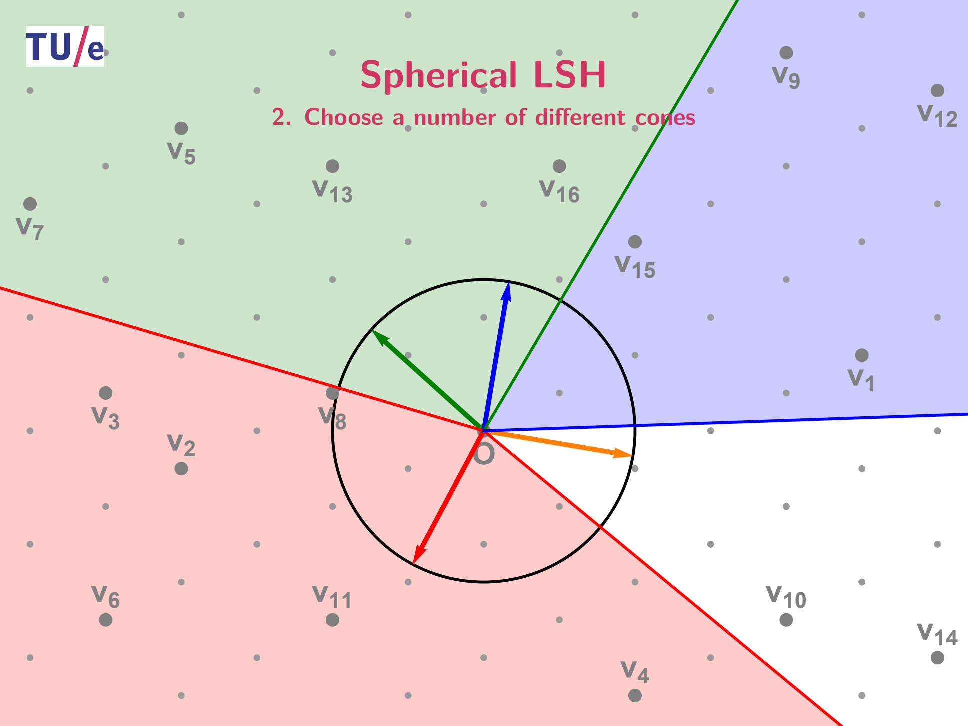
Spherical LSH

2. Choose a number of different cones



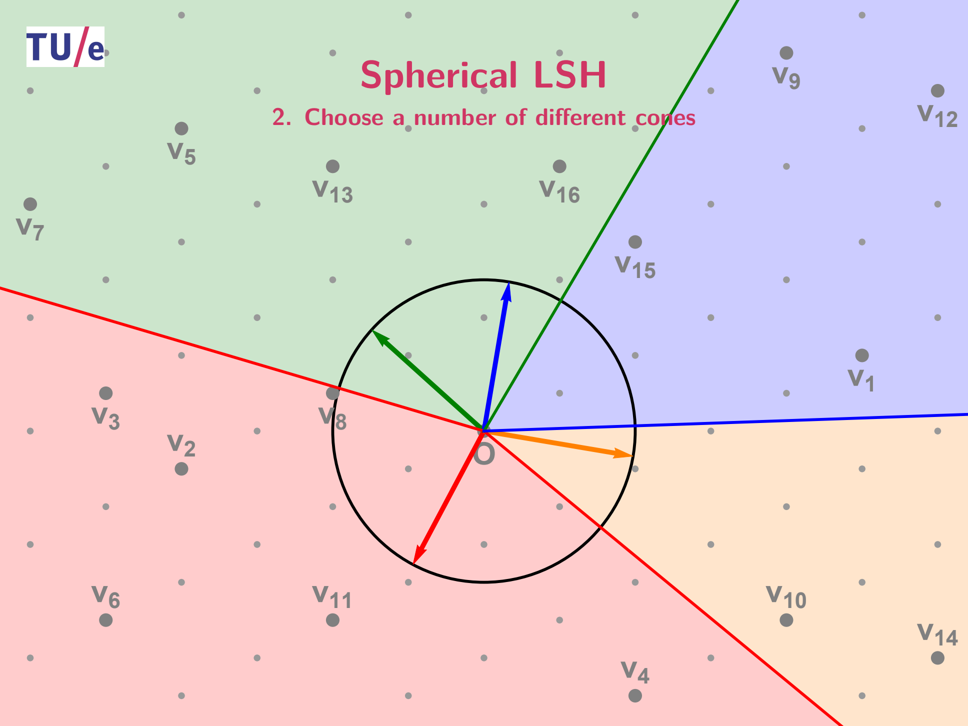
Spherical LSH

2. Choose a number of different cones



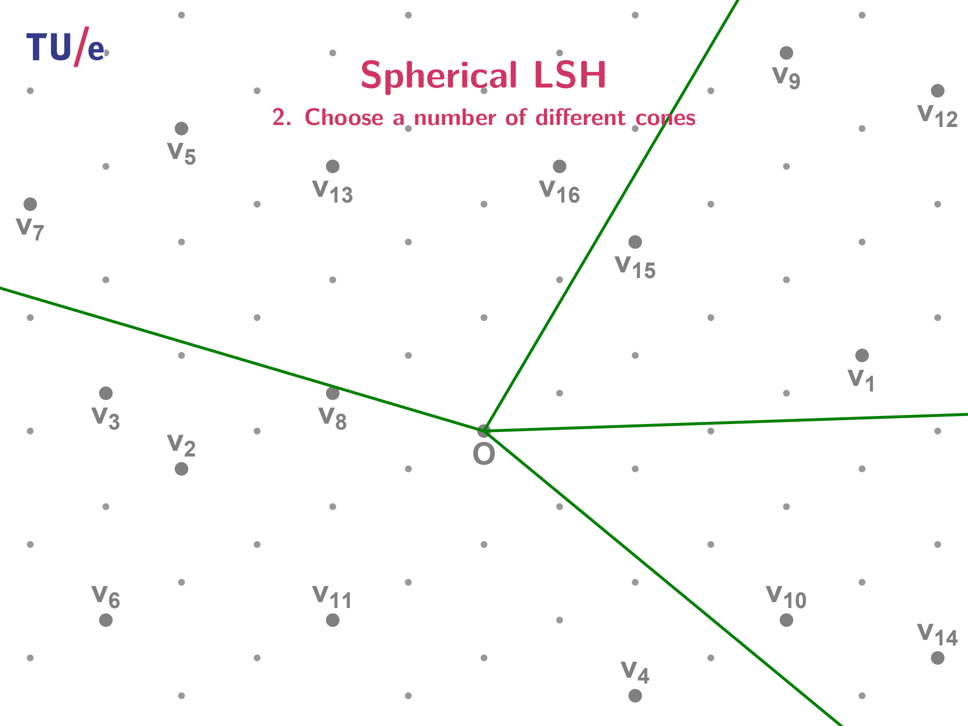
Spherical LSH

2. Choose a number of different cones



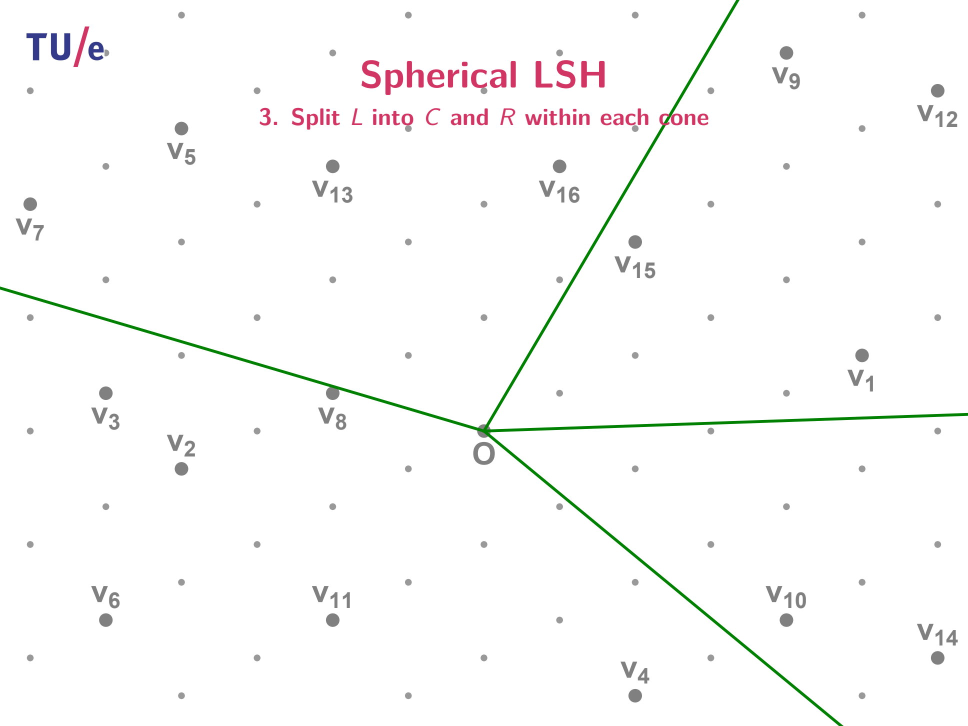
Spherical LSH

2. Choose a number of different cones



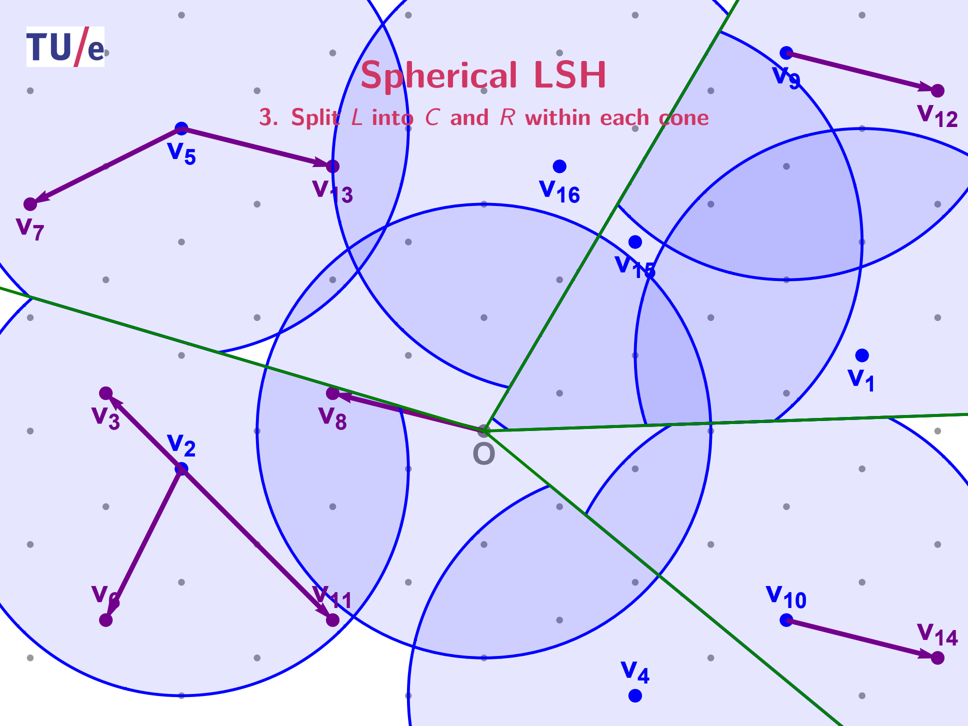
Spherical LSH

3. Split L into C and R within each cone



Spherical LSH

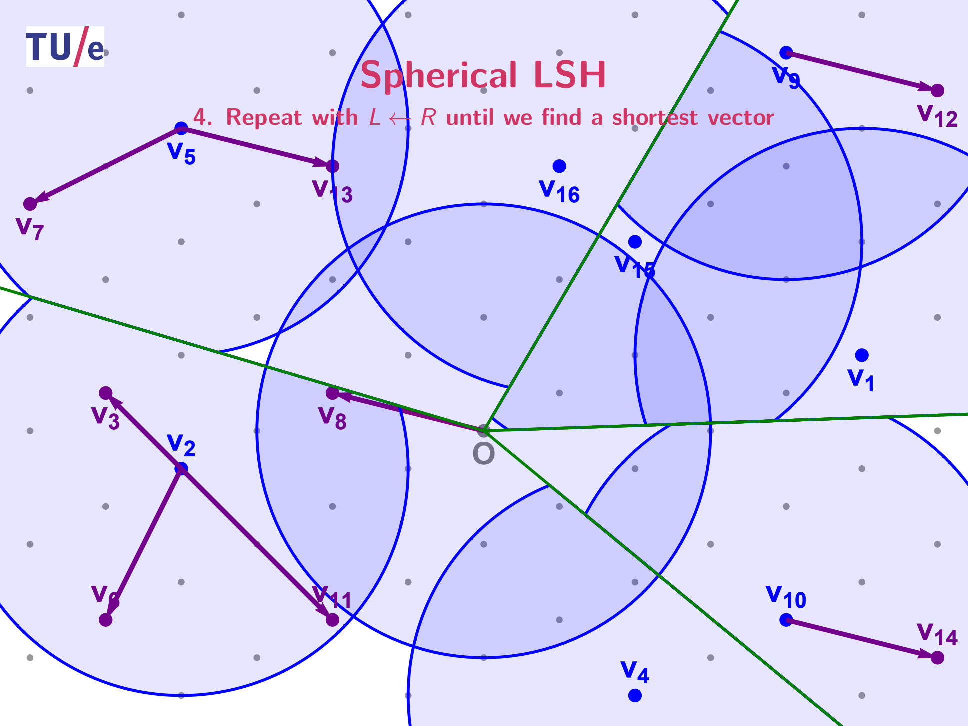
3. Split L into C and R within each cone



TU/e

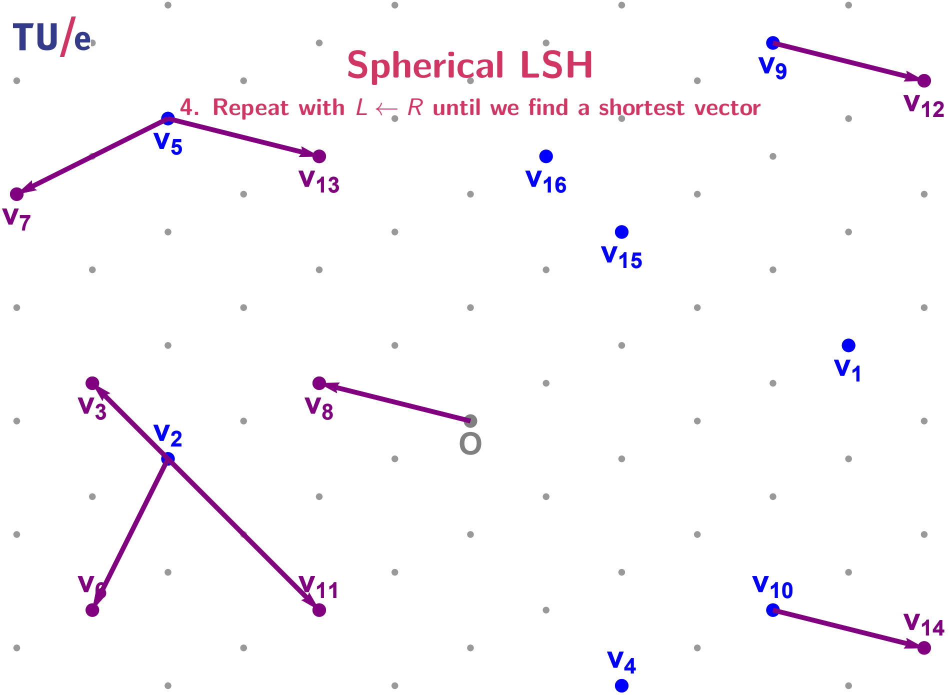
Spherical LSH

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



Spherical LSH

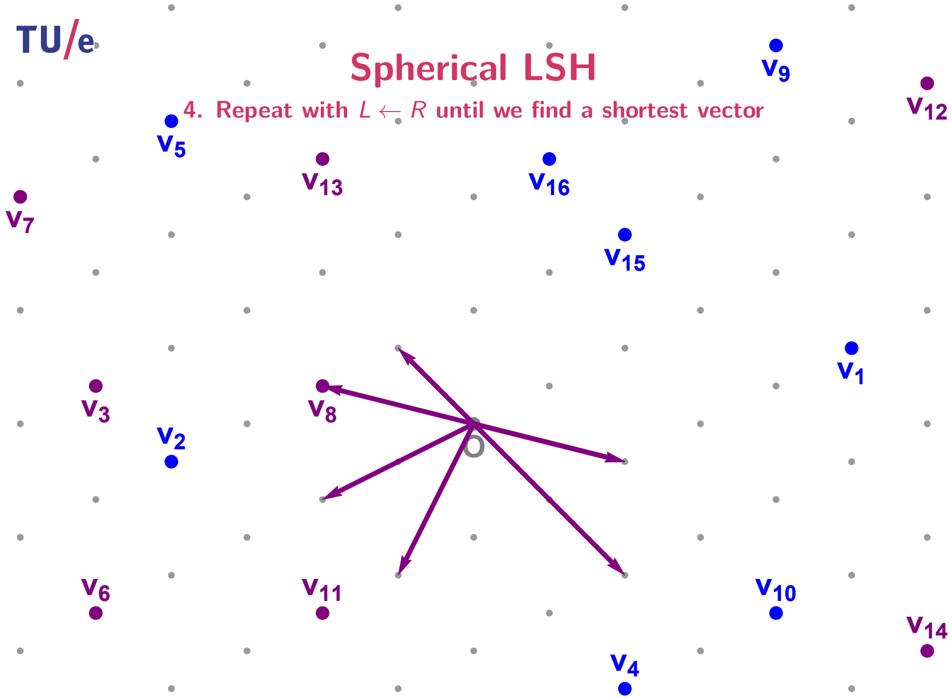
4. Repeat with $L \leftarrow R$ until we find a shortest vector



TU/e

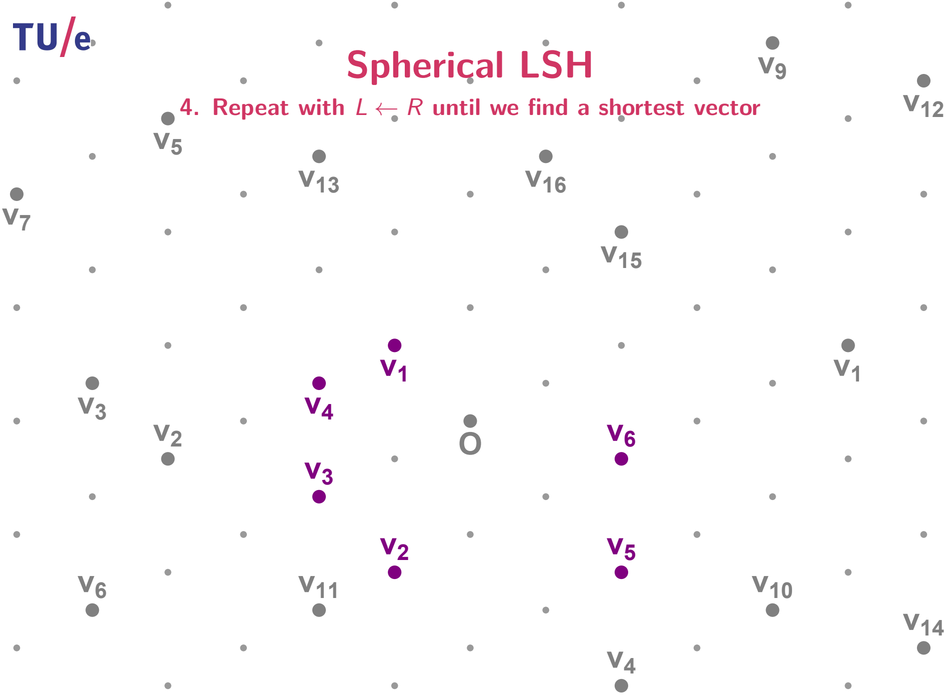
Spherical LSH

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



Spherical LSH

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



Spherical LSH

Overview

Spherical LSH

Overview

- Two parameters to tune
  - $k = O(\sqrt{n})$: Number of conic partitions per hash table
  - $t = 2^{O(n)}$: Number of different, independent hash tables

Spherical LSH

Overview

- Two parameters to tune
  - $k = O(\sqrt{n})$: Number of conic partitions per hash table
  - $t = 2^{O(n)}$: Number of different, independent hash tables
- Space complexity: $2^{0.298n+o(n)}$
  - Number of vectors: $2^{0.208n+o(n)}$
  - Number of hash tables: $2^{0.090n+o(n)}$
  - Each hash table contains all vectors

Spherical LSH

Overview

- Two parameters to tune
  - $k = O(\sqrt{n})$: Number of conic partitions per hash table
  - $t = 2^{O(n)}$: Number of different, independent hash tables
- Space complexity: $2^{0.298n+o(n)}$
  - Number of vectors: $2^{0.208n+o(n)}$
  - Number of hash tables: $2^{0.090n+o(n)}$
  - Each hash table contains all vectors
- Time complexity: $2^{0.298n+o(n)}$
  - Cost of computing hashes: $2^{0.090n+o(n)}$
  - Candidate nearest vectors: $2^{0.090n+o(n)}$
  - Repeat this for each list vector: $2^{0.208n+o(n)}$

Spherical LSH

Overview

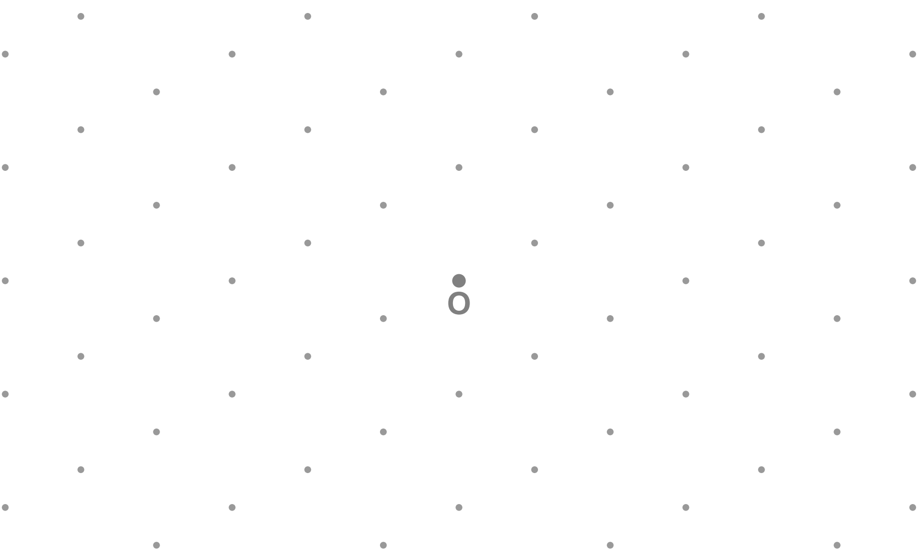
- Two parameters to tune
  - $k = O(\sqrt{n})$: Number of conic partitions per hash table
  - $t = 2^{O(n)}$: Number of different, independent hash tables
- Space complexity: $2^{0.298n+o(n)}$
  - Number of vectors: $2^{0.208n+o(n)}$
  - Number of hash tables: $2^{0.090n+o(n)}$
  - Each hash table contains all vectors
- Time complexity: $2^{0.298n+o(n)}$
  - Cost of computing hashes: $2^{0.090n+o(n)}$
  - Candidate nearest vectors: $2^{0.090n+o(n)}$
  - Repeat this for each list vector: $2^{0.208n+o(n)}$

Theorem

Sieving with spherical LSH heuristically solves SVP in time and space $2^{0.298n+o(n)}$.

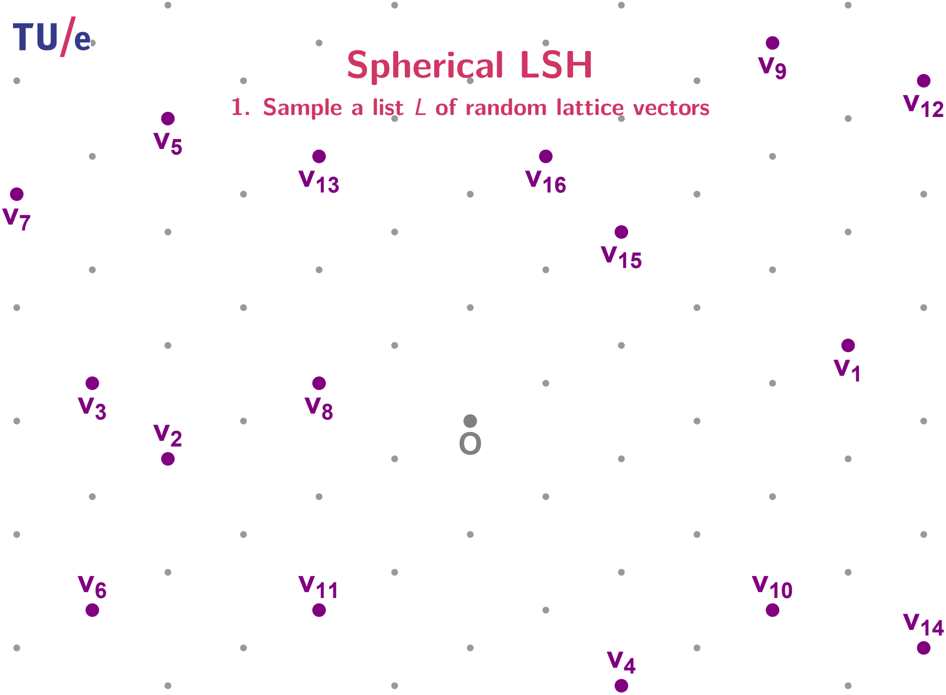
Spherical LSH

1. Sample a list L of random lattice vectors



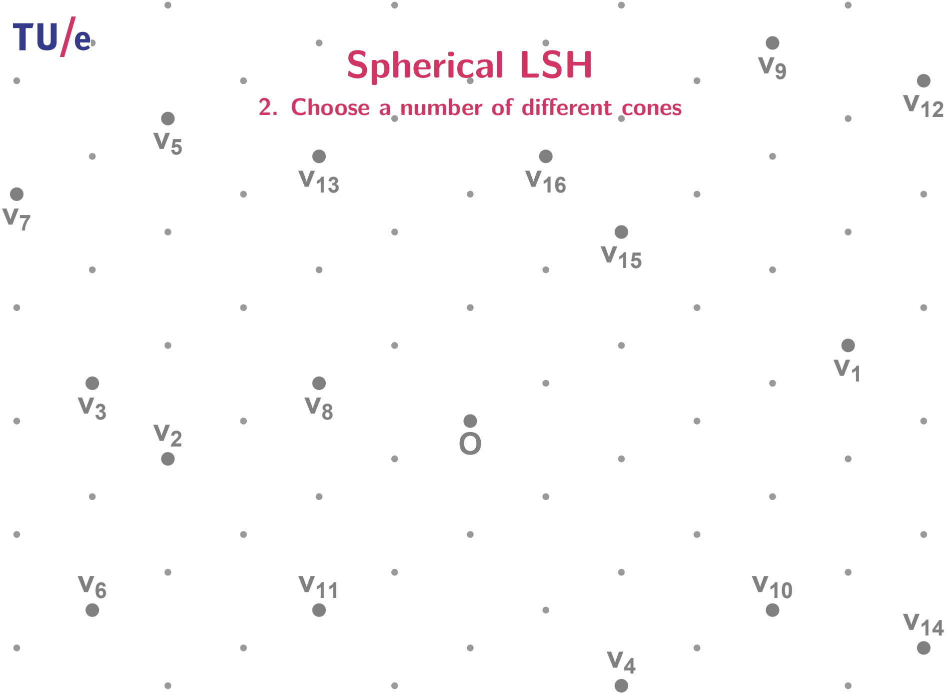
Spherical LSH

1. Sample a list L of random lattice vectors



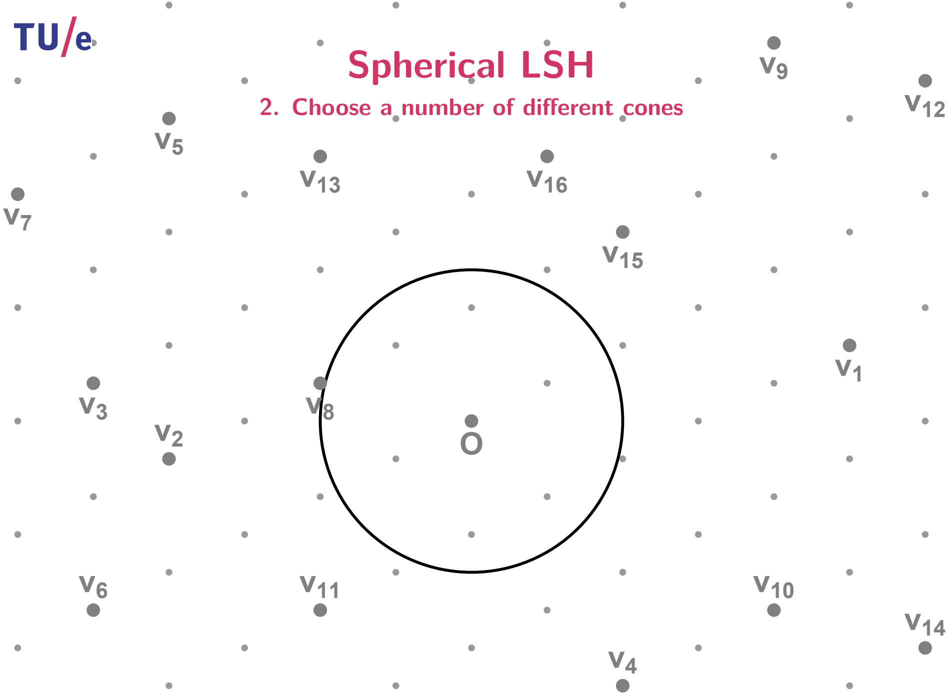
Spherical LSH

2. Choose a number of different cones



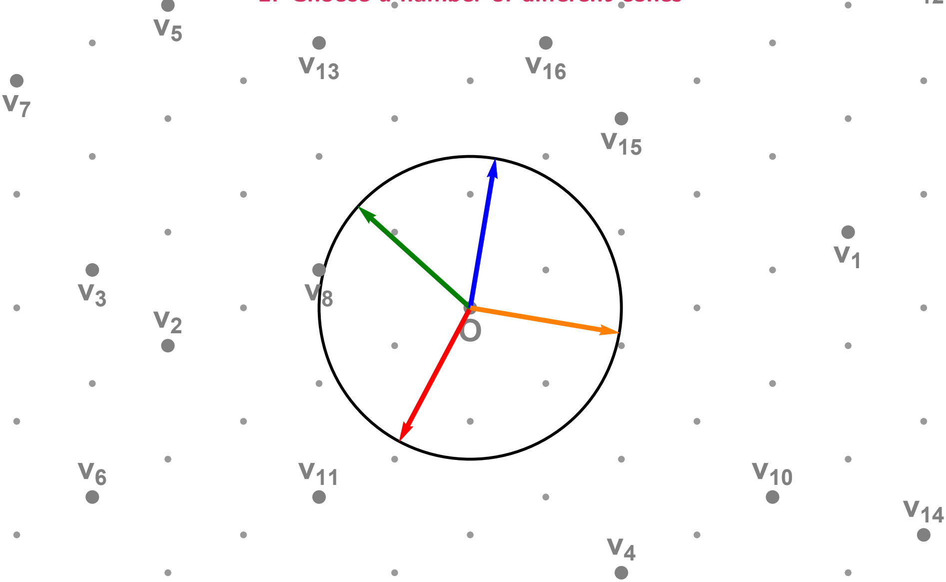
Spherical LSH

2. Choose a number of different cones



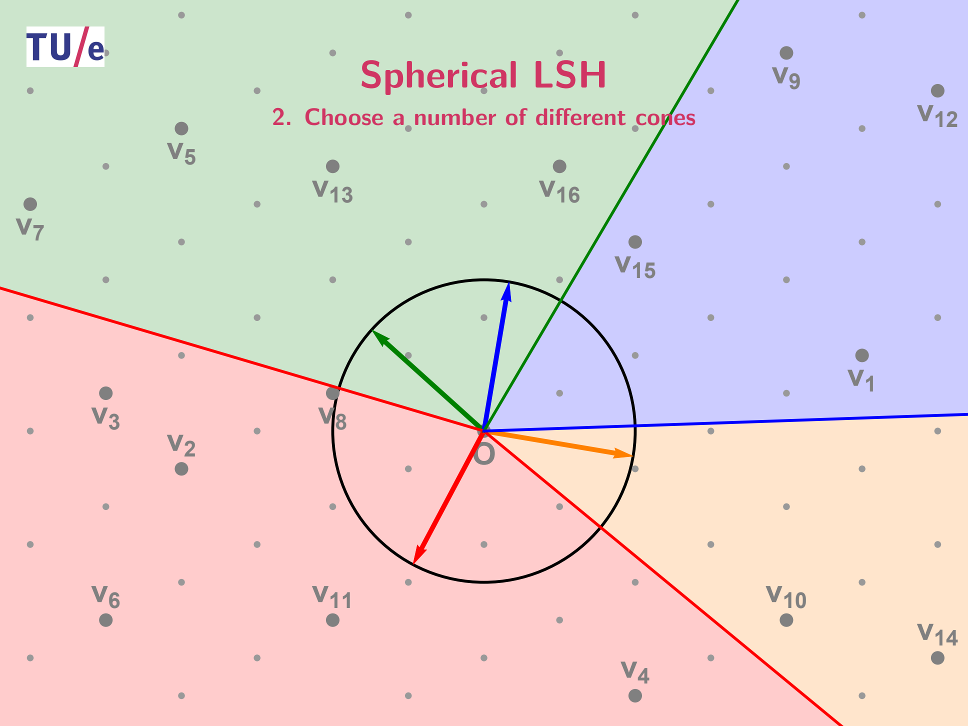
Spherical LSH

2. Choose a number of different cones



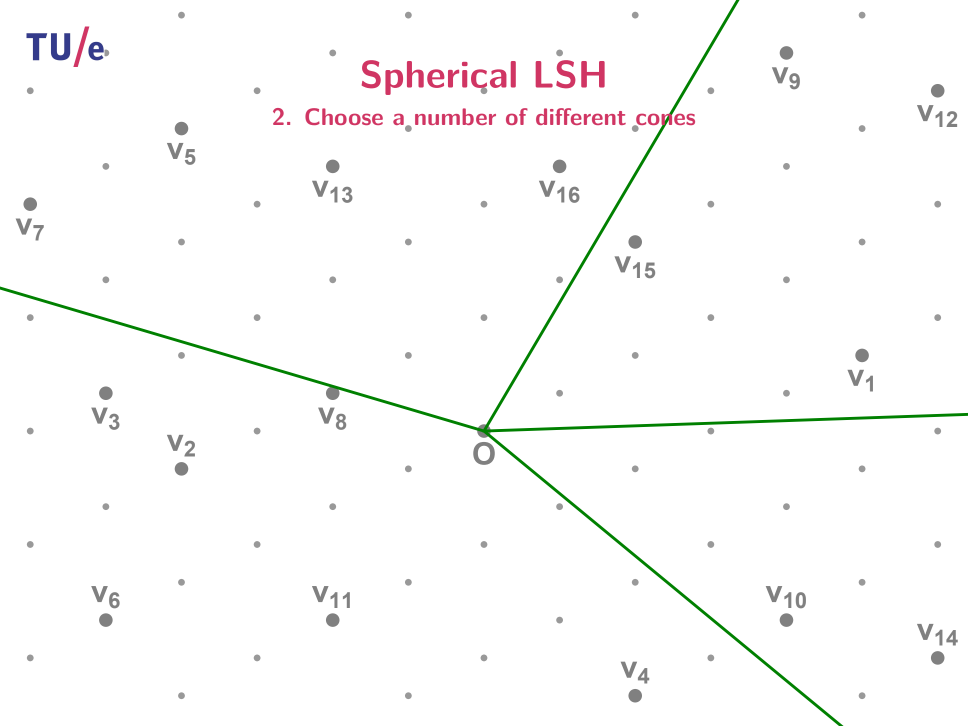
Spherical LSH

2. Choose a number of different cones



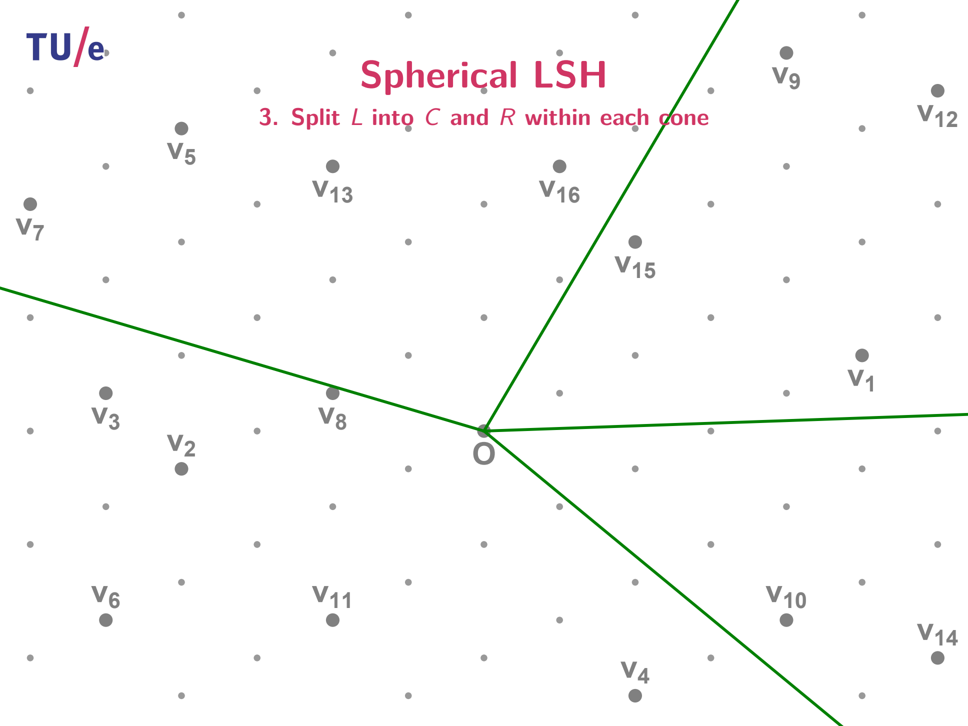
Spherical LSH

2. Choose a number of different cones



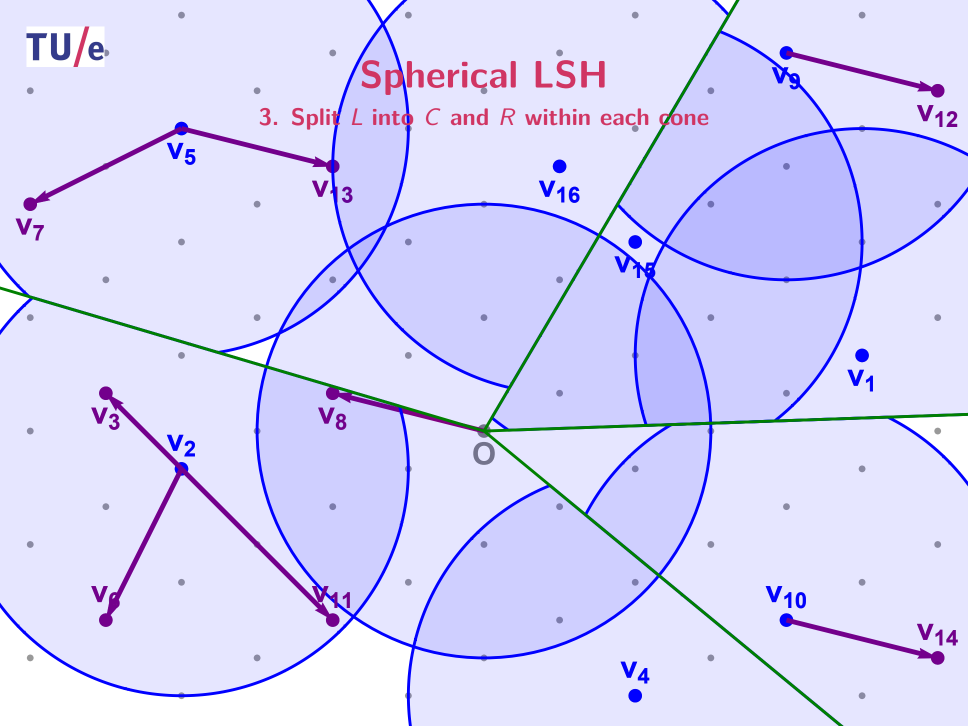
Spherical LSH

3. Split L into C and R within each cone



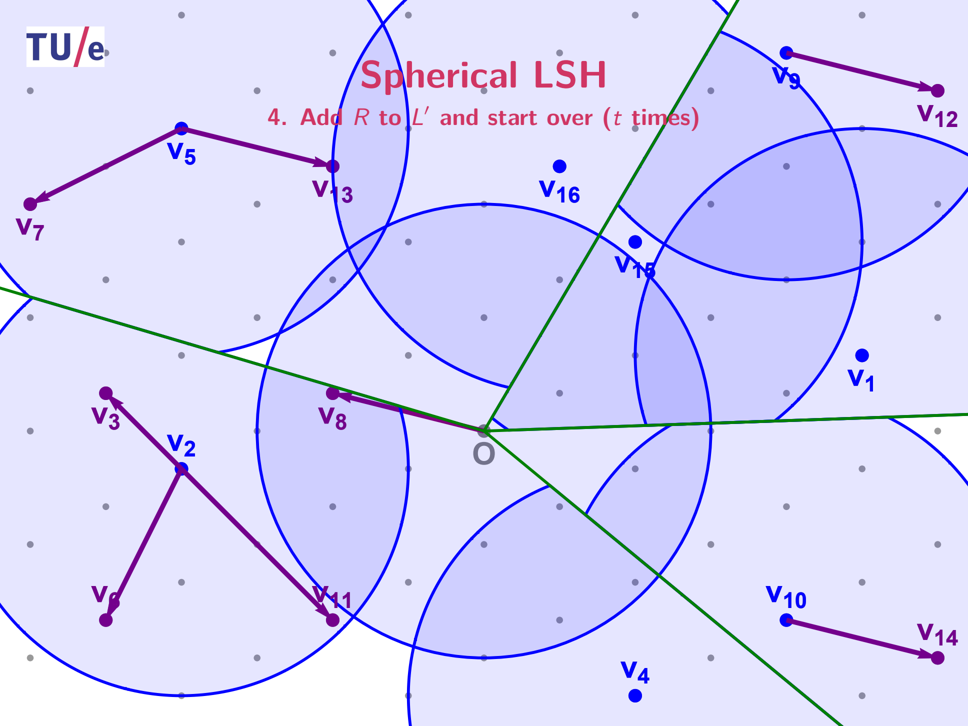
Spherical LSH

3. Split L into C and R within each cone



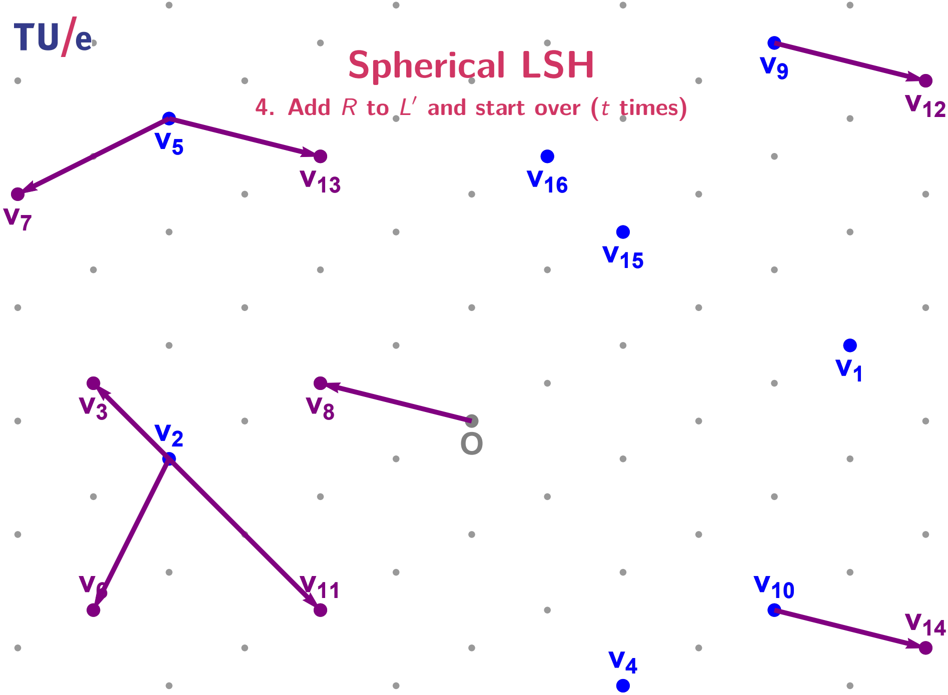
Spherical LSH

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



Spherical LSH

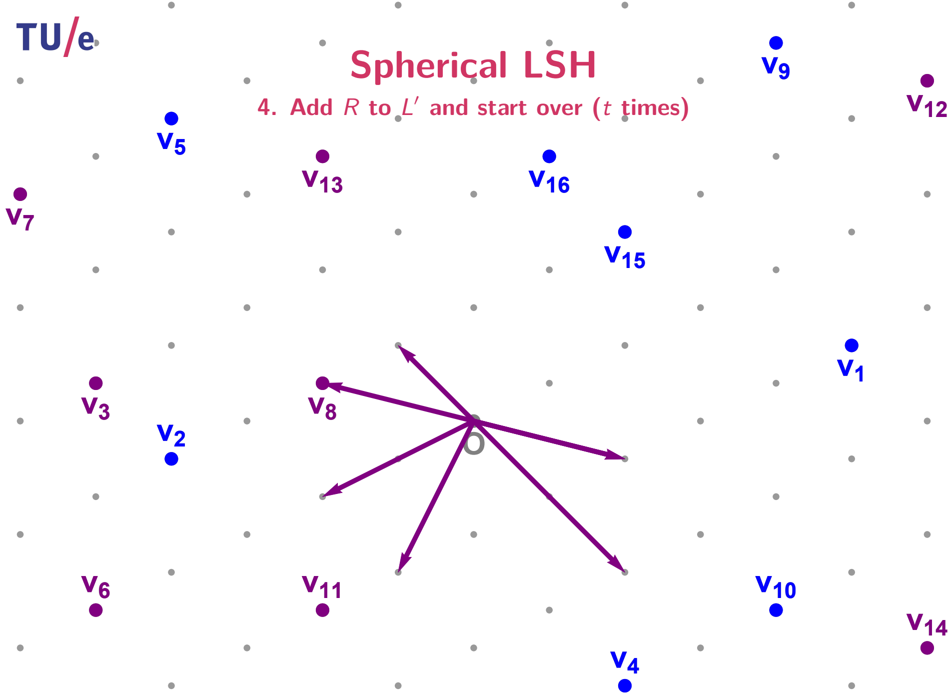
4. Add R to L' and start over (t times)



TU/e

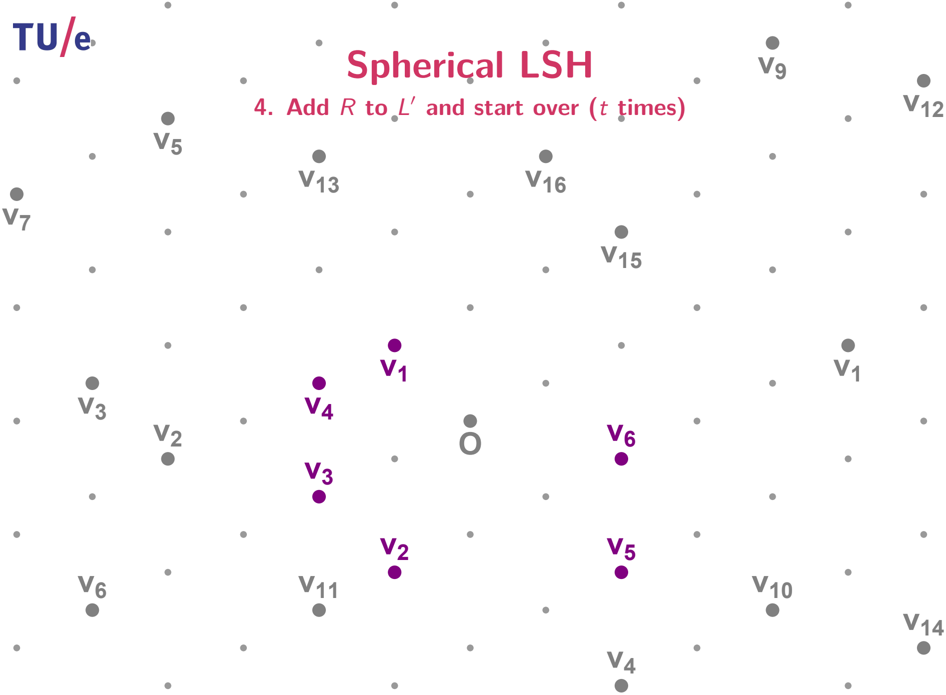
Spherical LSH

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



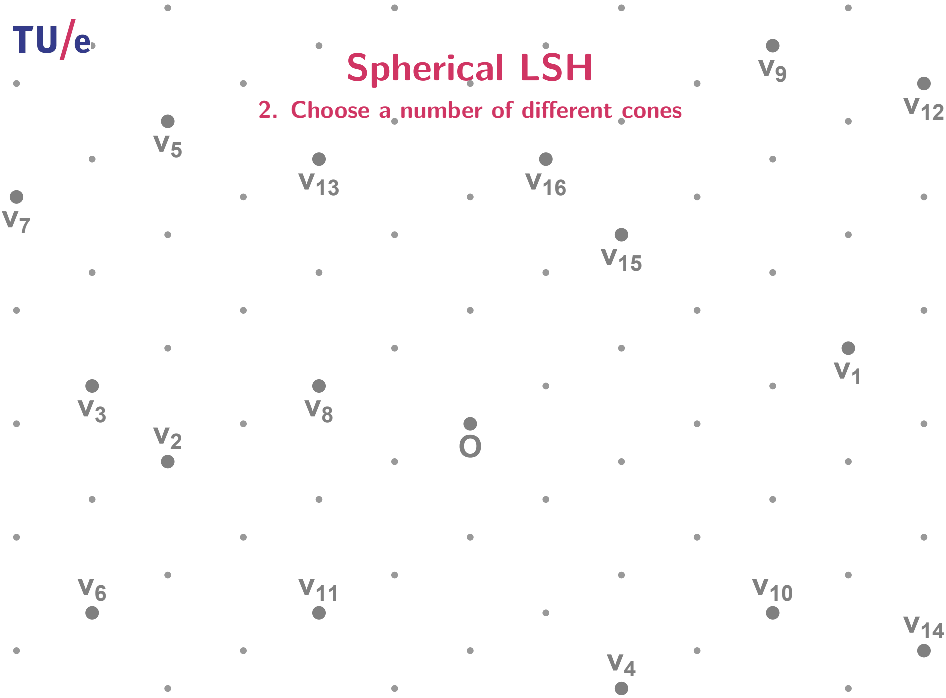
Spherical LSH

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



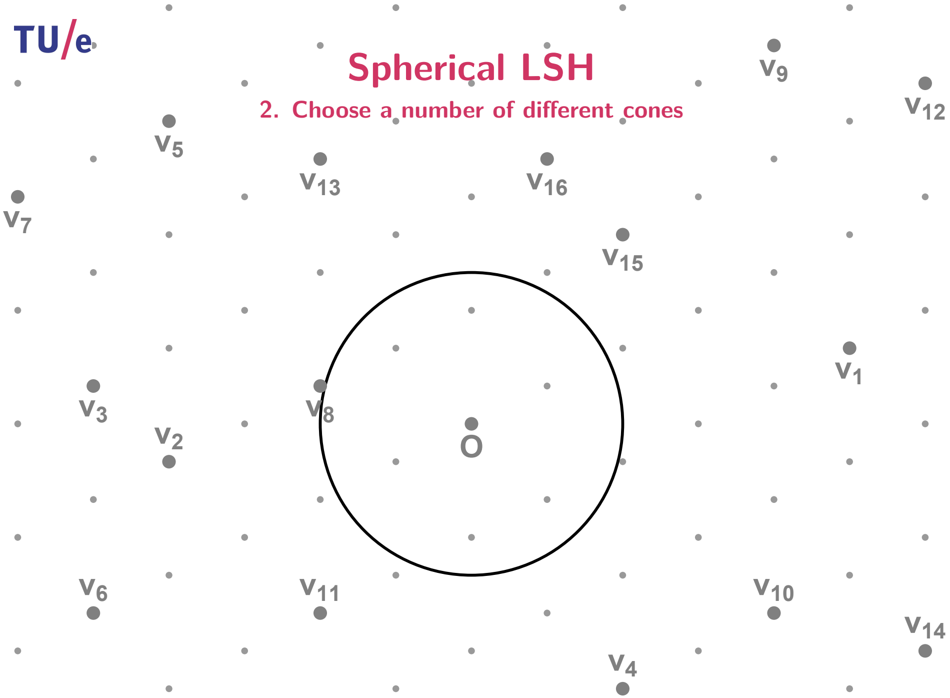
Spherical LSH

2. Choose a number of different cones



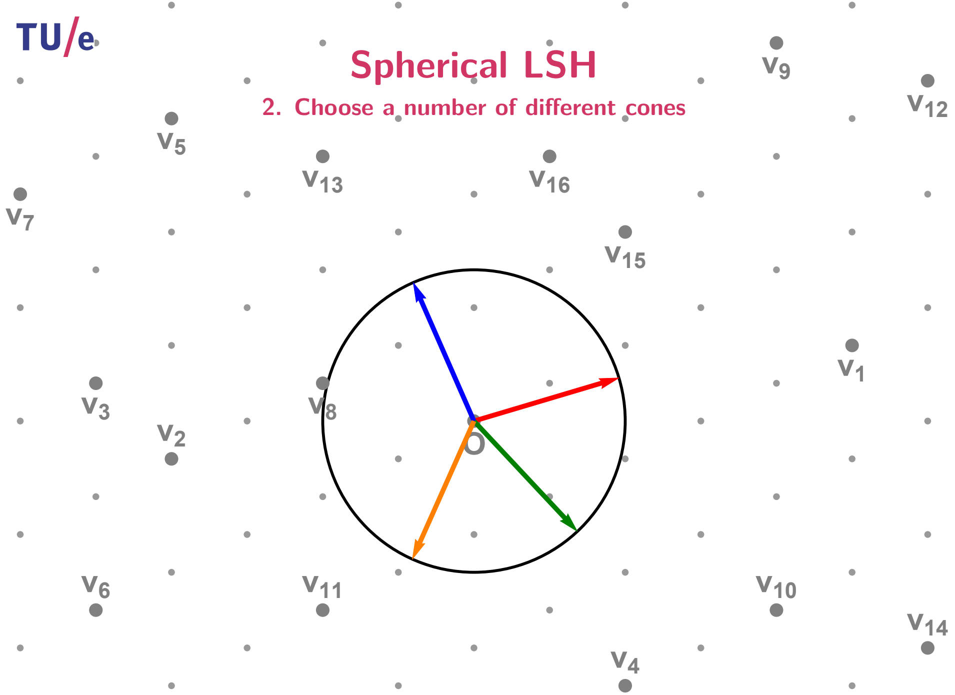
Spherical LSH

2. Choose a number of different cones



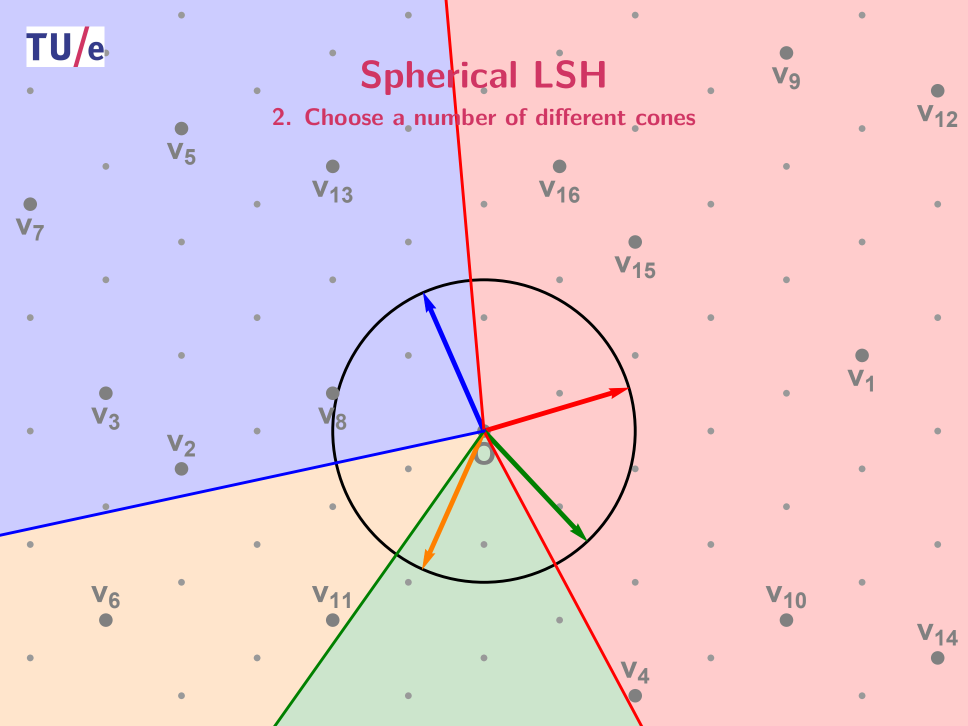
Spherical LSH

2. Choose a number of different cones



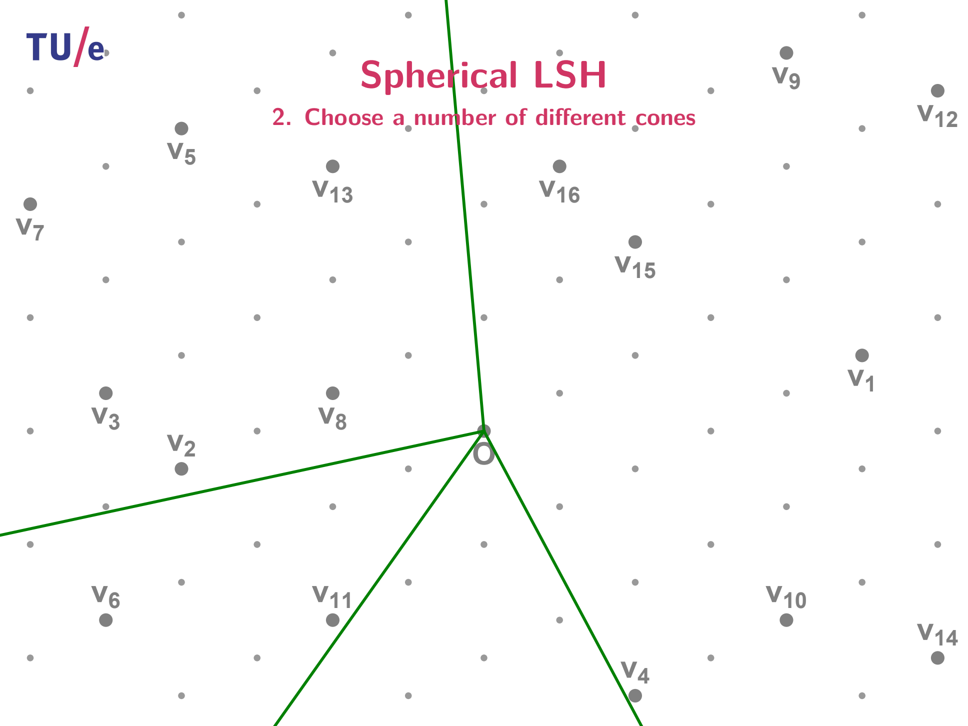
Spherical LSH

2. Choose a number of different cones



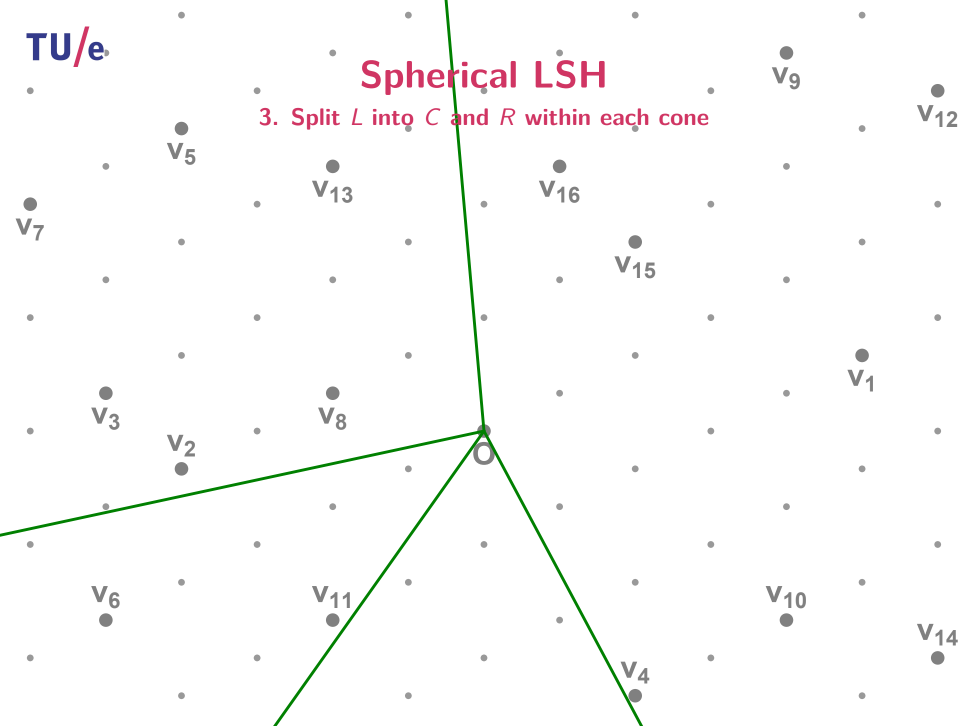
Spherical LSH

2. Choose a number of different cones



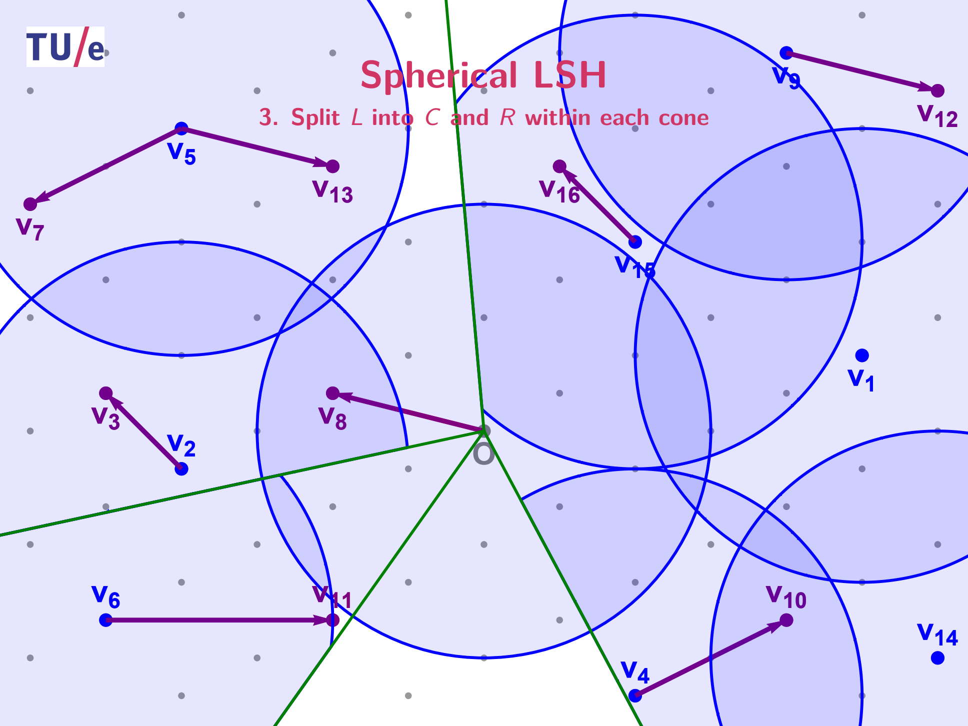
Spherical LSH

3. Split L into C and R within each cone



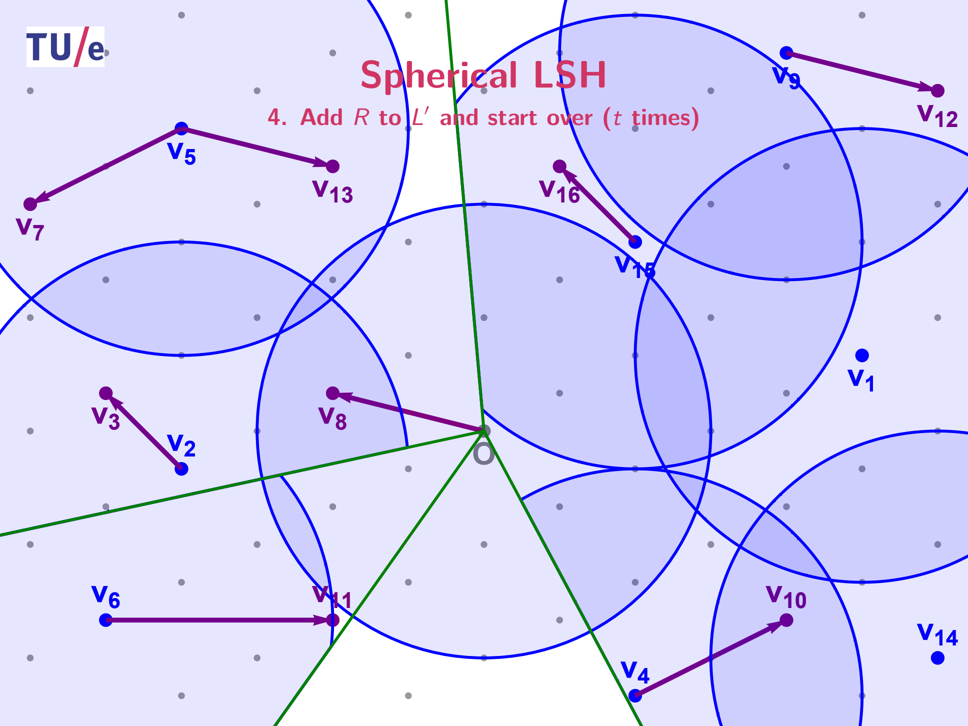
Spherical LSH

3. Split L into C and R within each cone



Spherical LSH

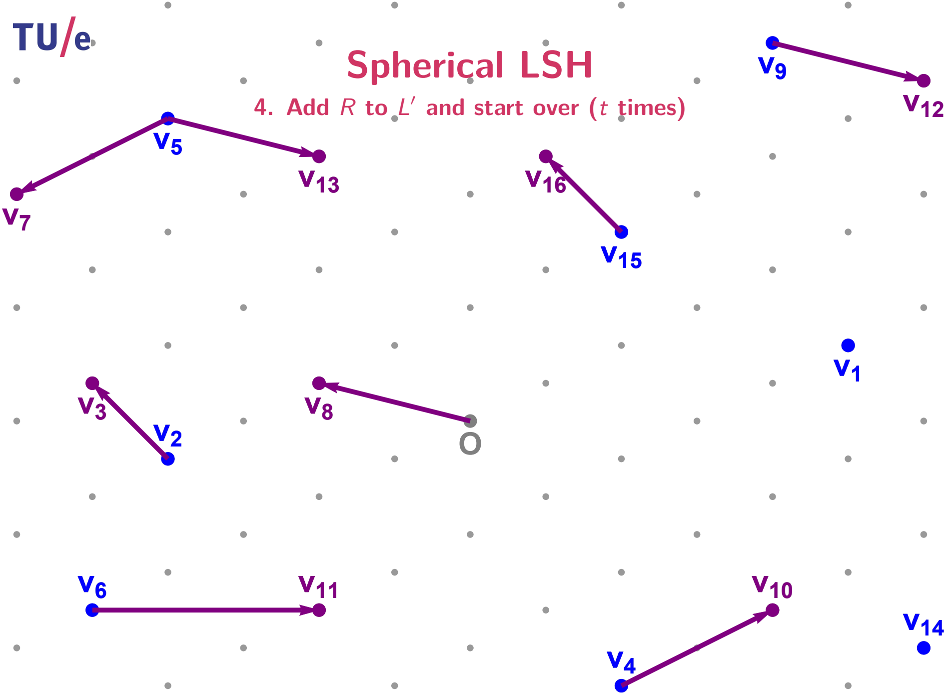
4. Add R to L' and start over (t times)



TU/e

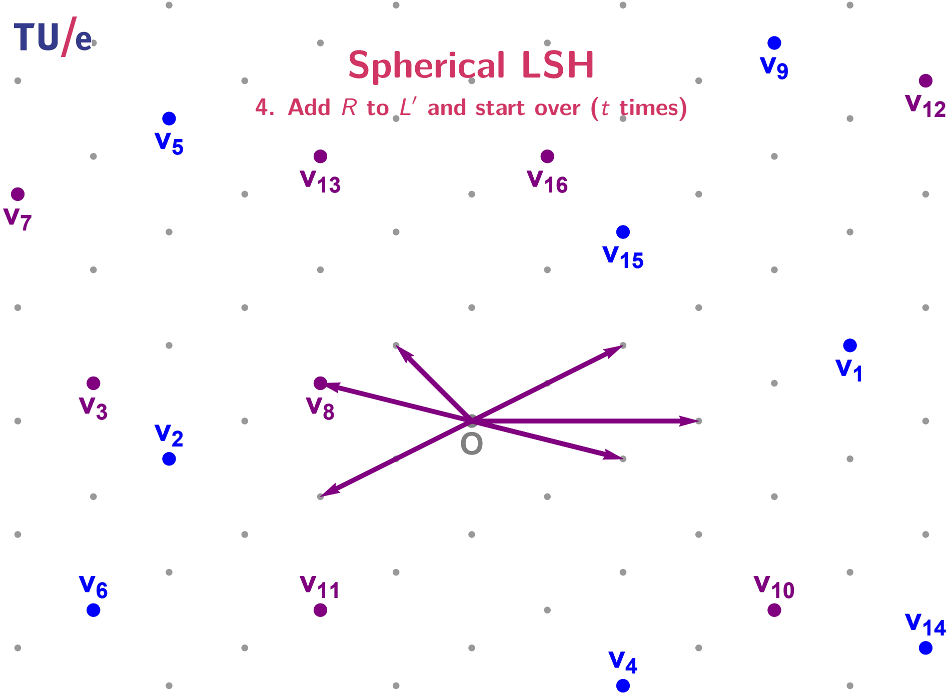
Spherical LSH

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



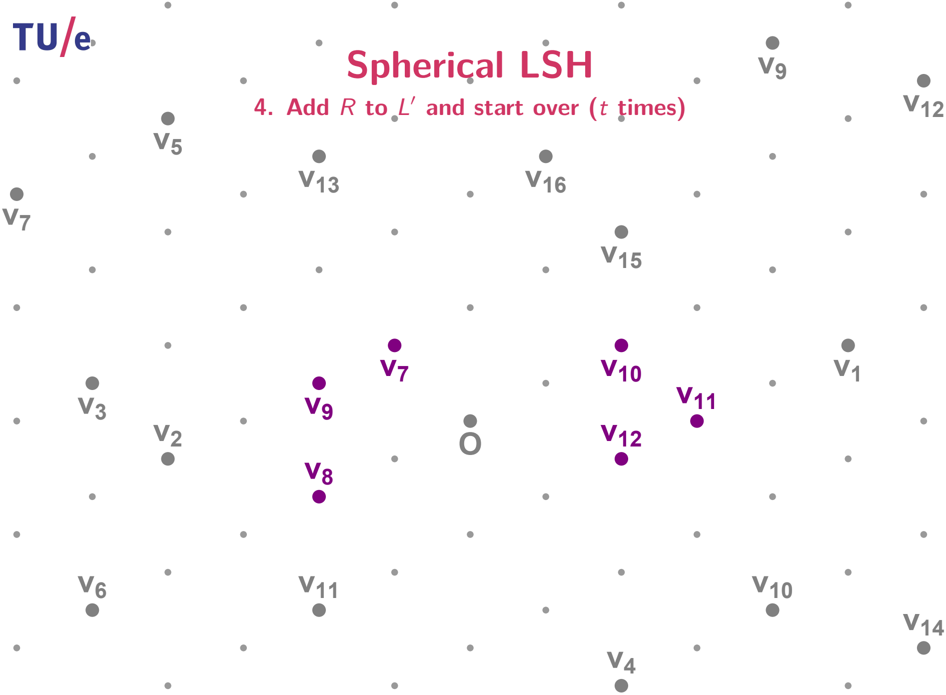
Spherical LSH

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



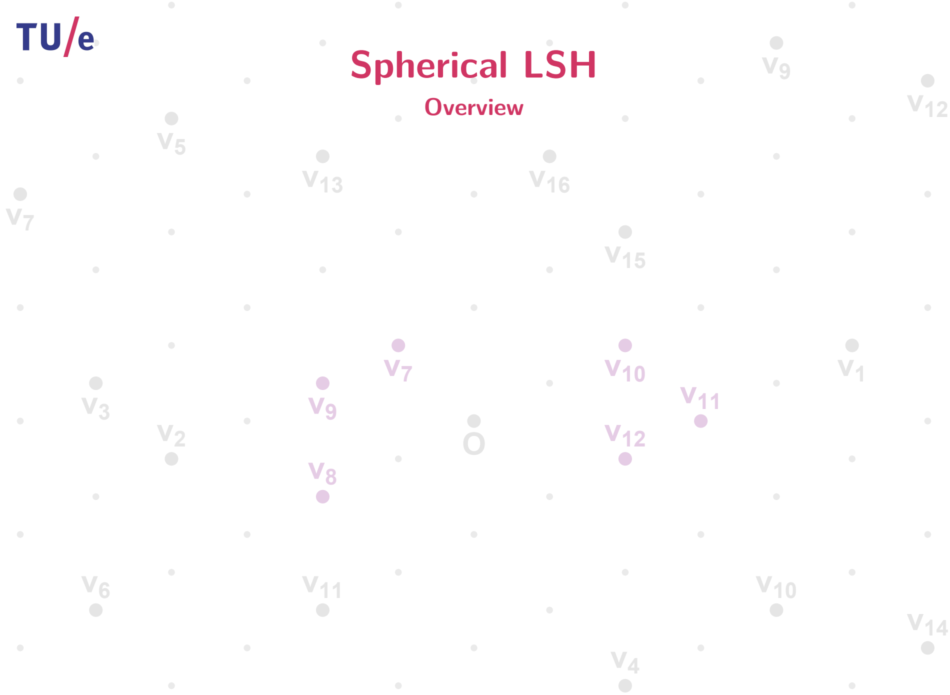
Spherical LSH

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



Spherical LSH

Overview



Spherical LSH

Overview

- Same k (conic partitions) and t (hash tables) as before

Spherical LSH

Overview

- Same k (conic partitions) and t (hash tables) as before
- Space complexity: $2^{0.208n+o(n)}$
  - Before: store $2^{0.090n}$ hash tables containing all $2^{0.208n}$ vectors
  - Now: process $2^{0.090n}$ hash tables one by one

Spherical LSH

Overview

- Same k (conic partitions) and t (hash tables) as before
- Space complexity: $2^{0.208n+o(n)}$
  - Before: store $2^{0.090n}$ hash tables containing all $2^{0.208n}$ vectors
  - Now: process $2^{0.090n}$ hash tables one by one
- Time complexity: $2^{0.298n+o(n)}$
  - Compute one hash, and go through $2^{o(n)}$ vectors
  - Repeat this for each of $2^{0.208n}$ vectors
  - Repeat this for each of $2^{0.090n}$ hash tables

Spherical LSH

Overview

- Same k (conic partitions) and t (hash tables) as before
- Space complexity: $2^{0.208n+o(n)}$
  - Before: store $2^{0.090n}$ hash tables containing all $2^{0.208n}$ vectors
  - Now: process $2^{0.090n}$ hash tables one by one
- Time complexity: $2^{0.298n+o(n)}$
  - Compute one hash, and go through $2^{o(n)}$ vectors
  - Repeat this for each of $2^{0.208n}$ vectors
  - Repeat this for each of $2^{0.090n}$ hash tables

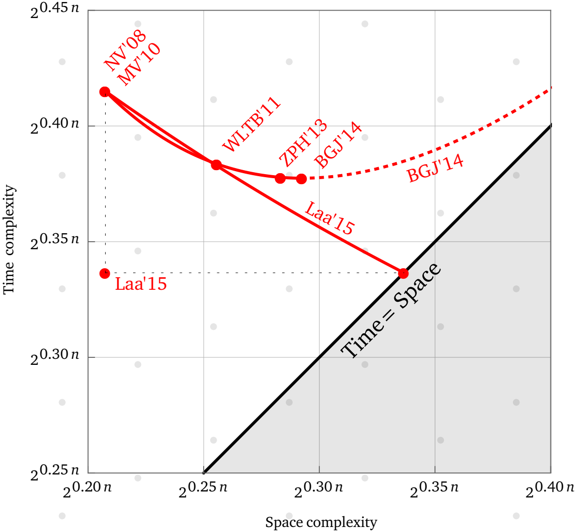
Heuristic

Sieving with spherical LSH heuristically solves SVP in time $2^{0.298n+o(n)}$ and space $2^{0.208n+o(n)}$.

TU/e

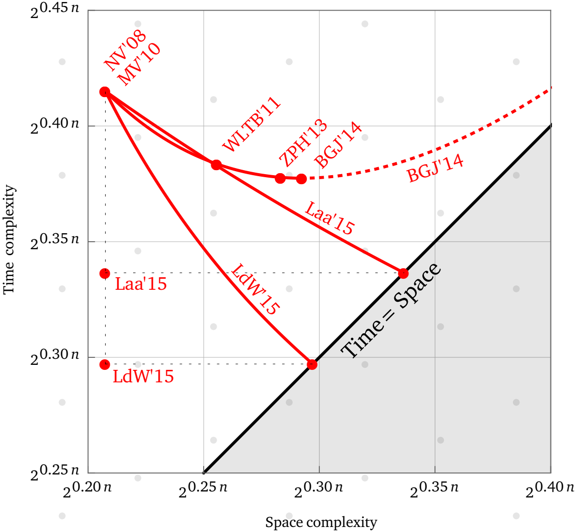
Spherical LSH

Space/time trade-off



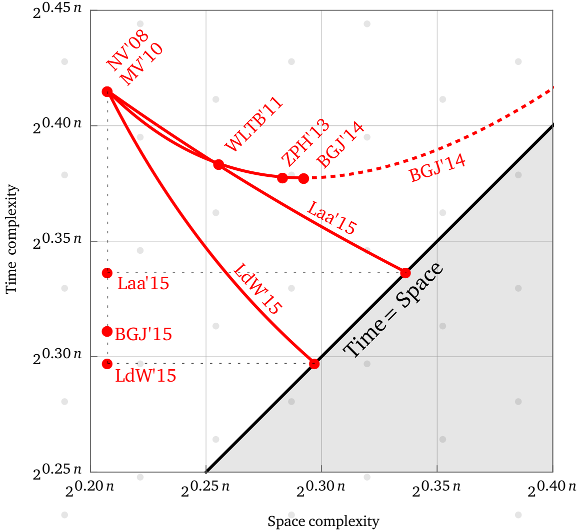
Spherical LSH

Space/time trade-off



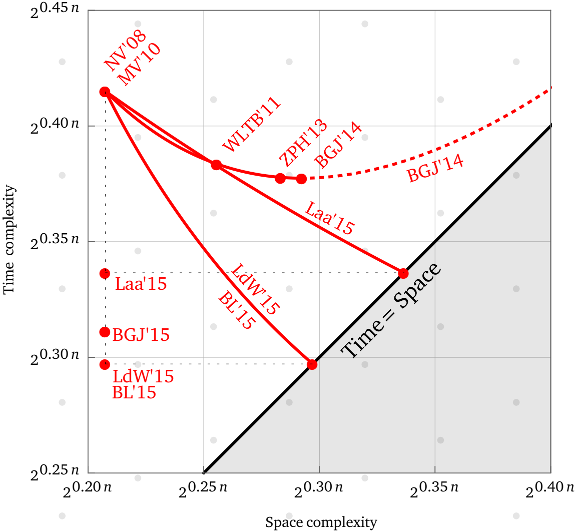
May and Ozerov's NNS method

Space/time trade-off



Cross-polytope LSH

Space/time trade-off



Questions?

[vdP'12]