IBM Research

Algorithms for finding shortest lattice vectors

Thijs Laarhoven

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

HEAT workshop, Paris, France
(July 6, 2016)

Outline

Lattices

Enumeration algorithms
- Fincke–Pohst enumeration
- Kannan enumeration
- (Extreme) pruning

Constructing the Voronoi cell

Sieving algorithms
- Basic sieving
- Leveled sieving
- Nearest neighbor searching

Conclusion

Outline

Lattices

Enumeration algorithms
- Fincke–Pohst enumeration
- Kannan enumeration
- (Extreme) pruning

Constructing the Voronoi cell

Sieving algorithms
- Basic sieving
- Leveled sieving
- Nearest neighbor searching

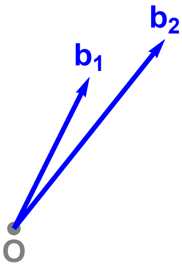
Conclusion

Lattices

What is a lattice?

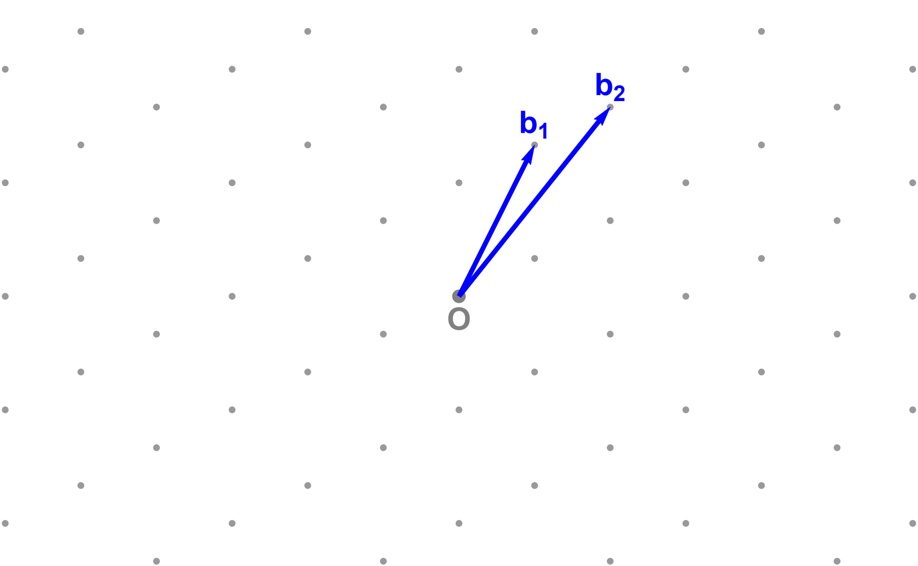
Lattices

What is a lattice?



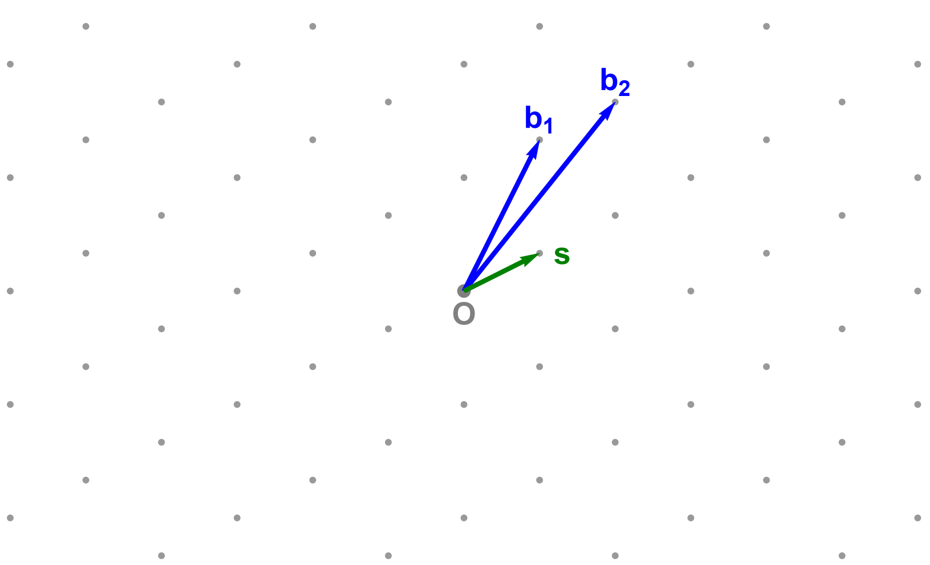
Lattices

What is a lattice?



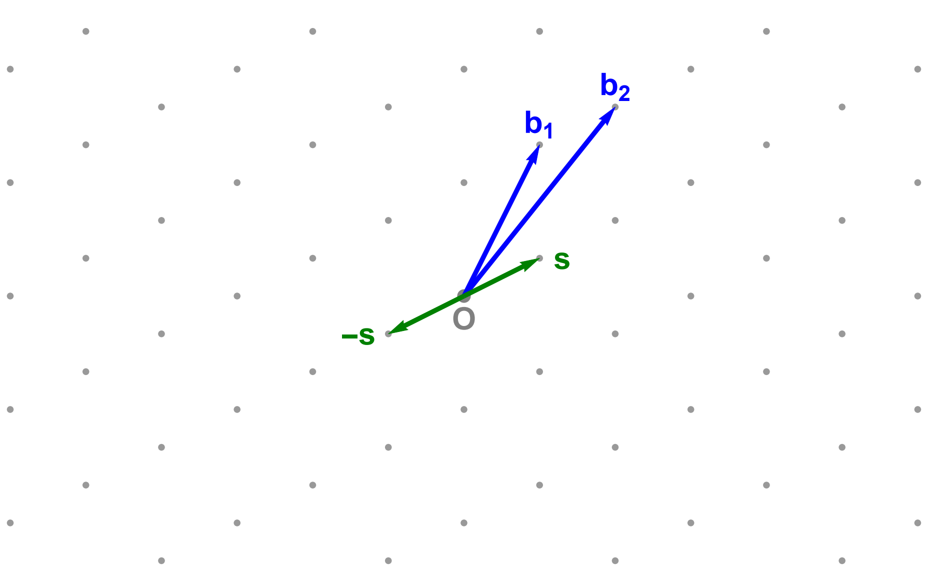
Lattices

Shortest Vector Problem (SVP)



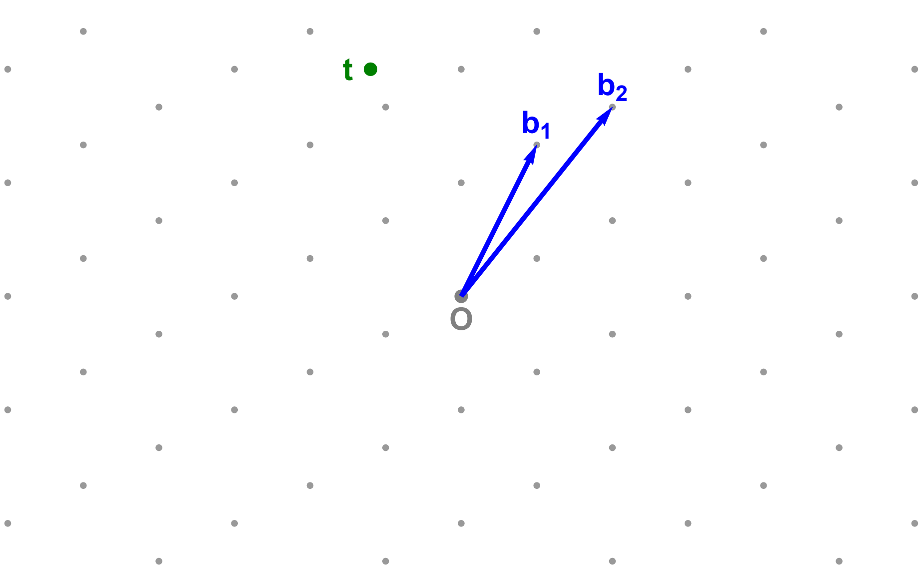
Lattices

Shortest Vector Problem (SVP)



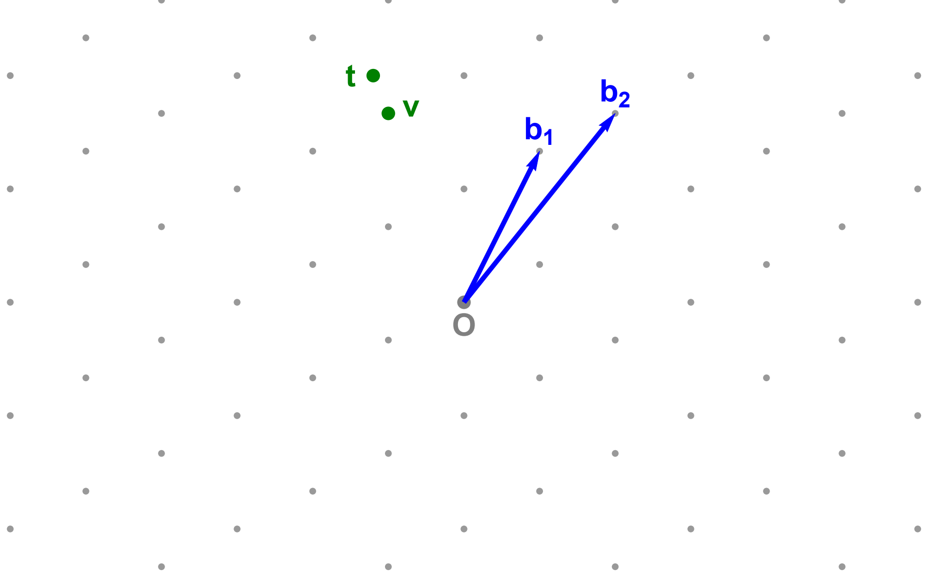
Lattices

Closest Vector Problem (CVP)



Lattices

Closest Vector Problem (CVP)



Lattices

Exact SVP algorithms

	Algorithm	$\log_2(\text{Time})$	$\log_2(\text{Space})$
Provable SVP	Enumeration [Poh81, Kan83, . . . , MW15]	$\Omega(n \log n)$	$O(\log n)$
	AKS-sieve [AKS01, NV08, MV10, HPS11]	$3.398n$	$1.985n$
	ListSieve [MV10, MDB14]	$3.199n$	$1.327n$
	AKS-sieve-birthday [PS09, HPS11]	$2.648n$	$1.324n$
	ListSieve-birthday [PS09]	$2.465n$	$1.233n$
	Voronoi cell algorithm [AEVZ02, MV10b]	$2.000n$	$1.000n$
	Discrete Gaussians [ADRS15, ADS15, Ste16]	$1.000n$	$1.000n$
Heuristic SVP	Nguyen–Vidick sieve [NV08]	$0.415n$	$0.208n$
	GaussSieve [MV10, . . . , IKMT14, BNvdP14]	$0.415n$	$0.208n$
	Two-level sieve [WLTB11]	$0.384n$	$0.256n$
	Three-level sieve [ZPH13]	$0.3778n$	$0.283n$
	Overlattice sieve [BGJ14]	$0.3774n$	$0.293n$
	Hyperplane LSH [Laa15, MLB15, Mar15]	$0.337n$	$0.208n$
	May and Ozerov's NNS method [BGJ15]	$0.311n$	$0.208n$
	Spherical LSH [LdW15]	$0.298n$	$0.208n$
	Cross-polytope LSH [BL15]	$0.298n$	$0.208n$
	Spherical filtering [BDGL16, Laa15, ML15]	$0.293n$	$0.208n$

Outline

Lattices

Enumeration algorithms
- Fincke–Pohst enumeration
- Kannan enumeration
- (Extreme) pruning

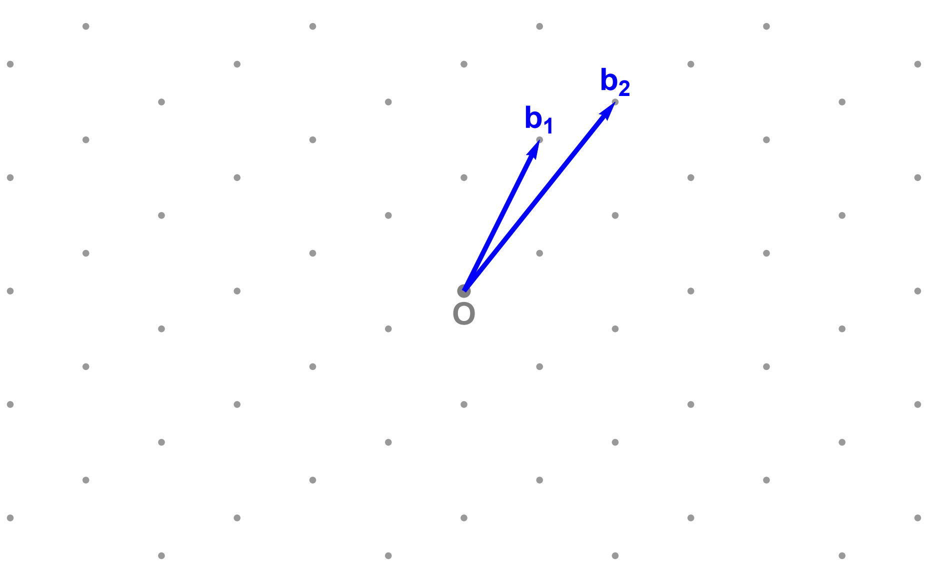
Constructing the Voronoi cell

Sieving algorithms
- Basic sieving
- Leveled sieving
- Nearest neighbor searching

Conclusion

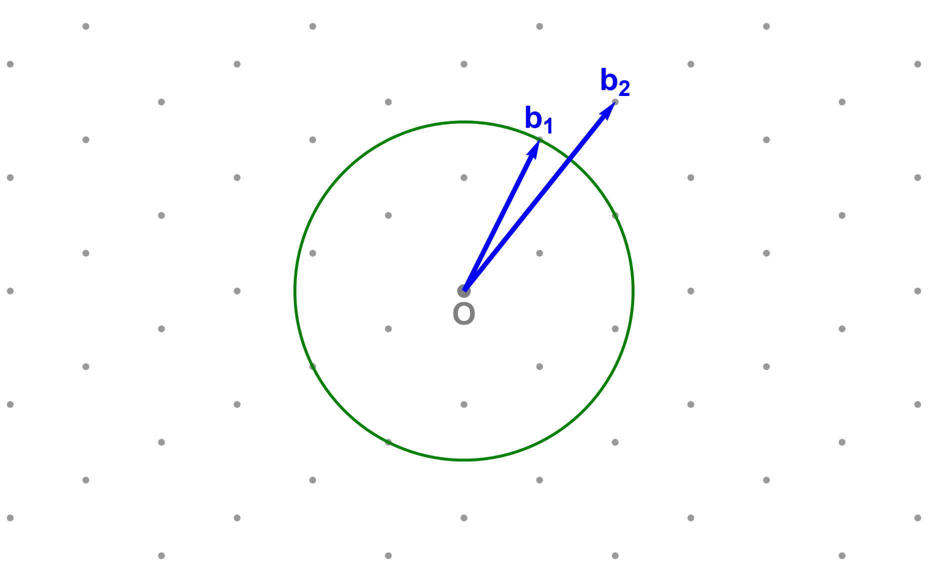
Fincke-Pohst enumeration

Determine possible coefficients of b_2



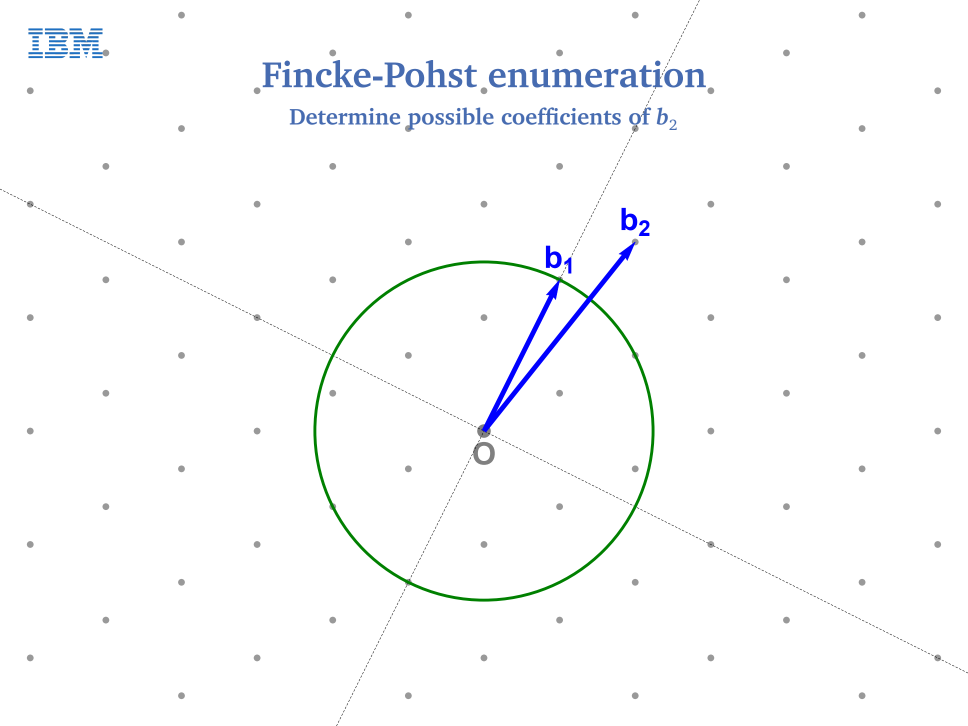
Fincke-Pohst enumeration

Determine possible coefficients of b_2



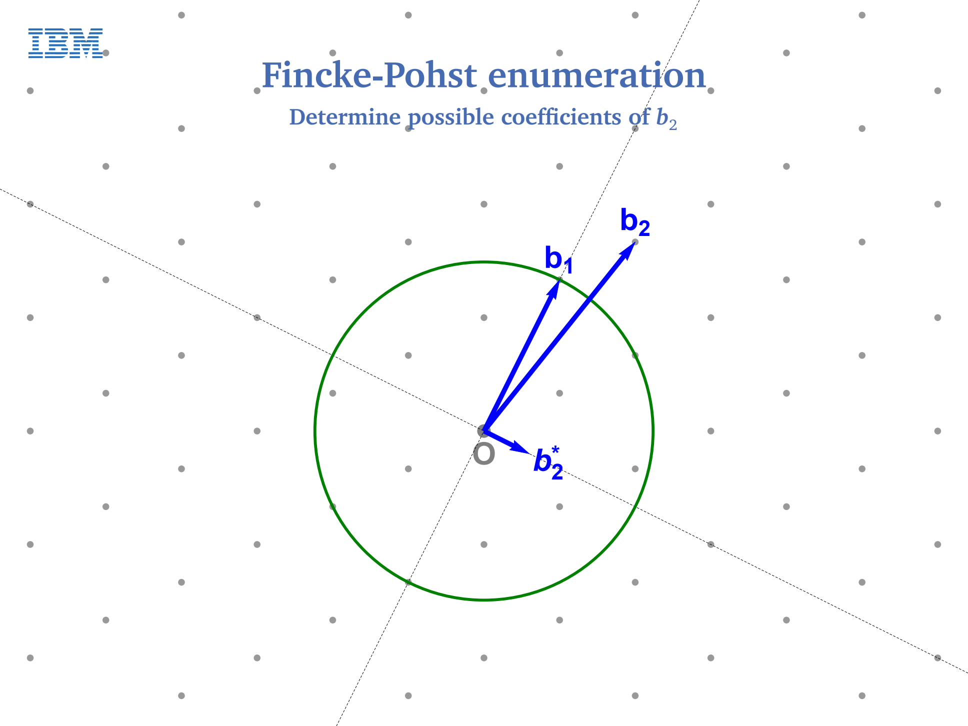
Fincke-Pohst enumeration

Determine possible coefficients of b_2



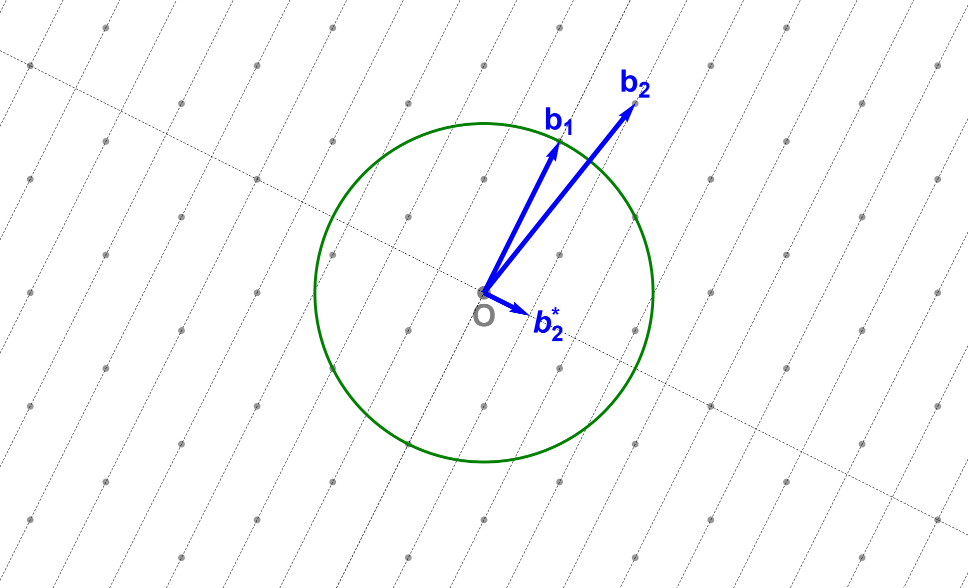
Fincke-Pohst enumeration

Determine possible coefficients of b_2



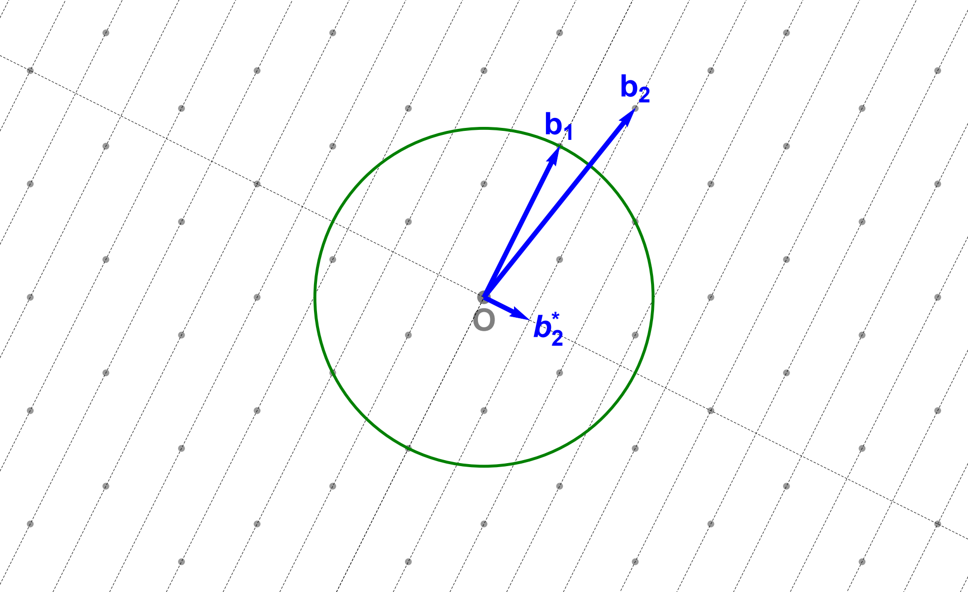
Fincke-Pohst enumeration

Determine possible coefficients of b_2



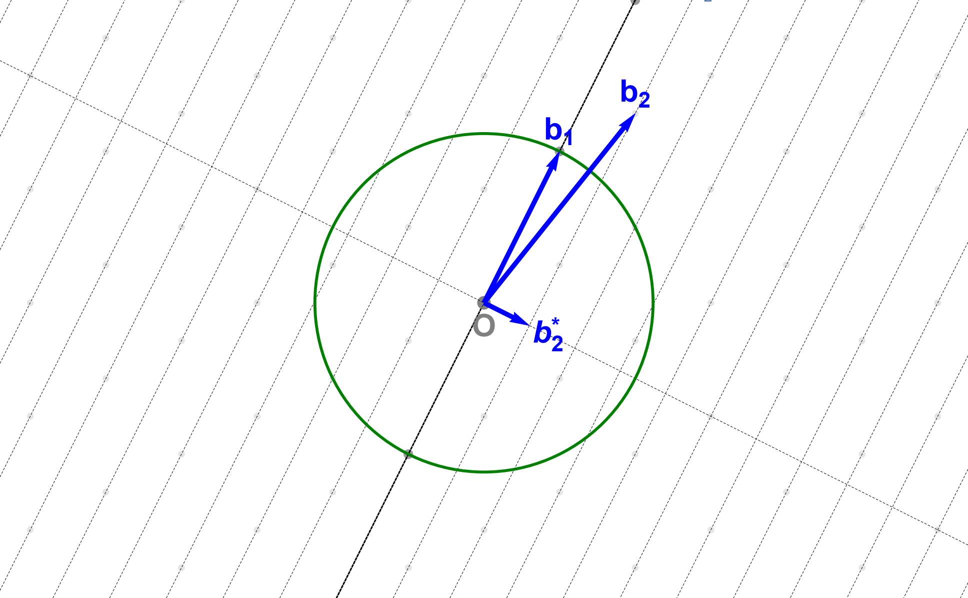
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



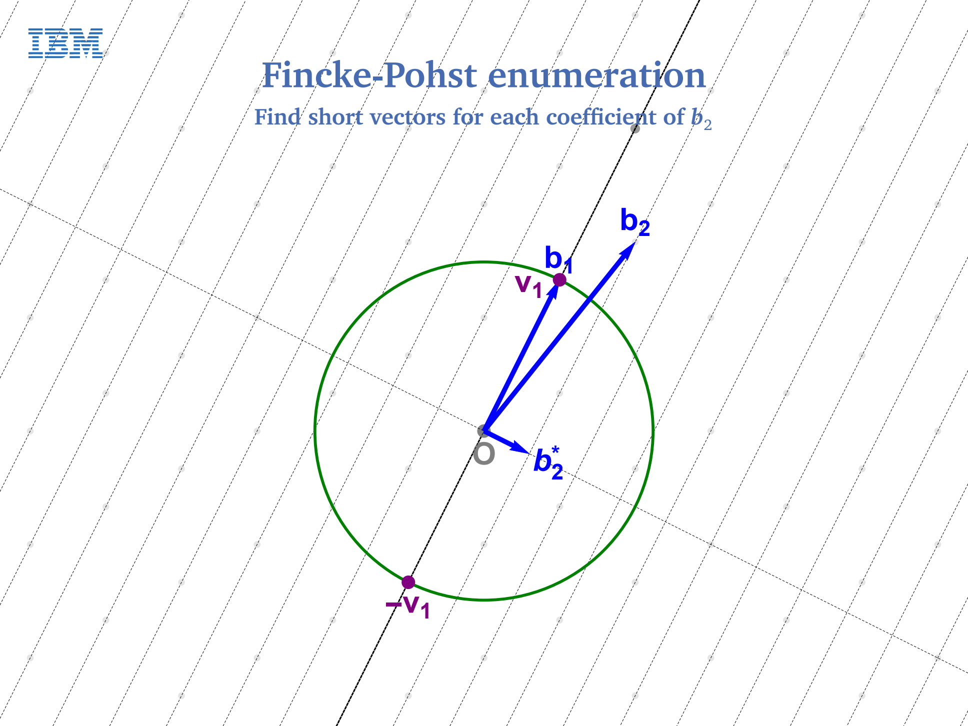
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



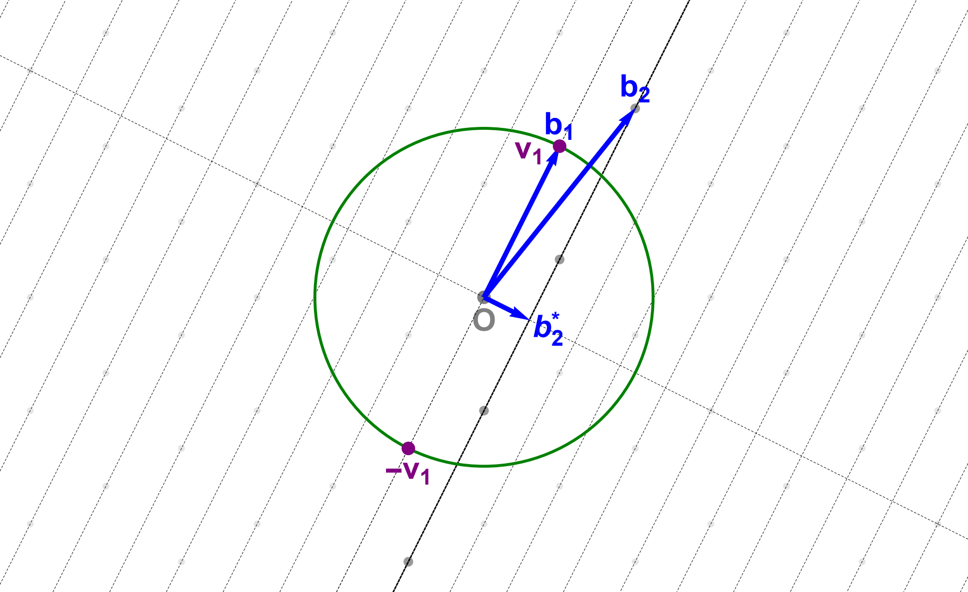
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



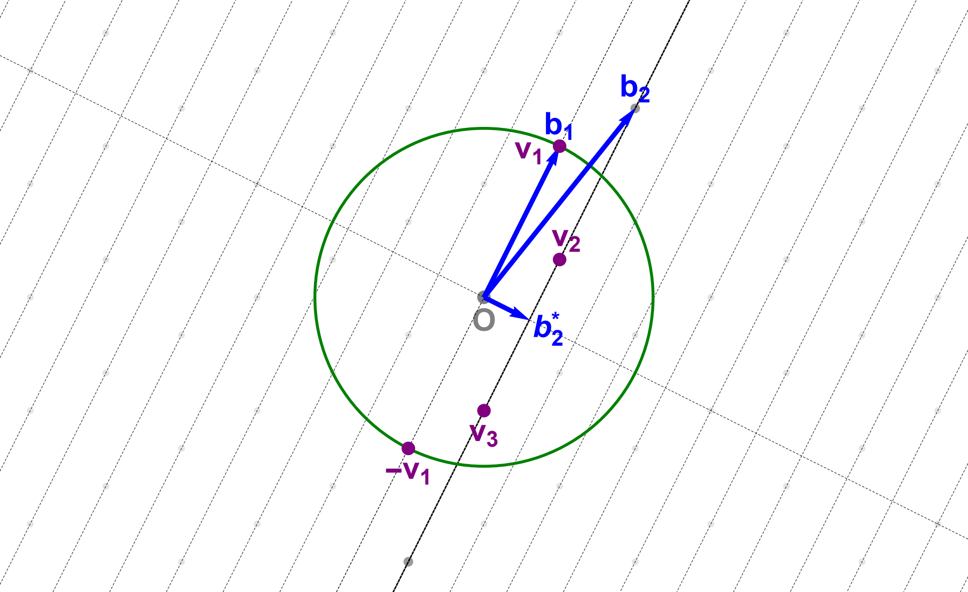
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



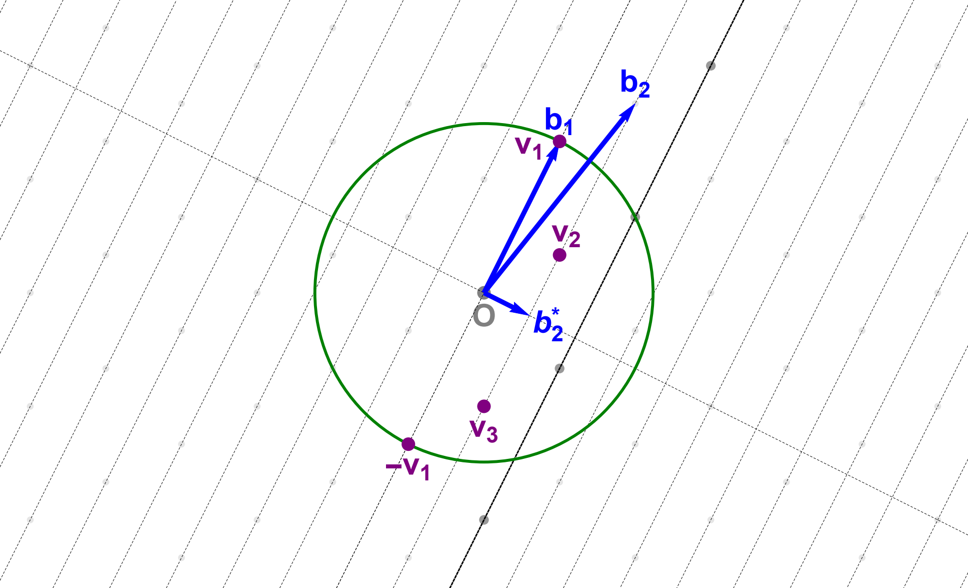
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



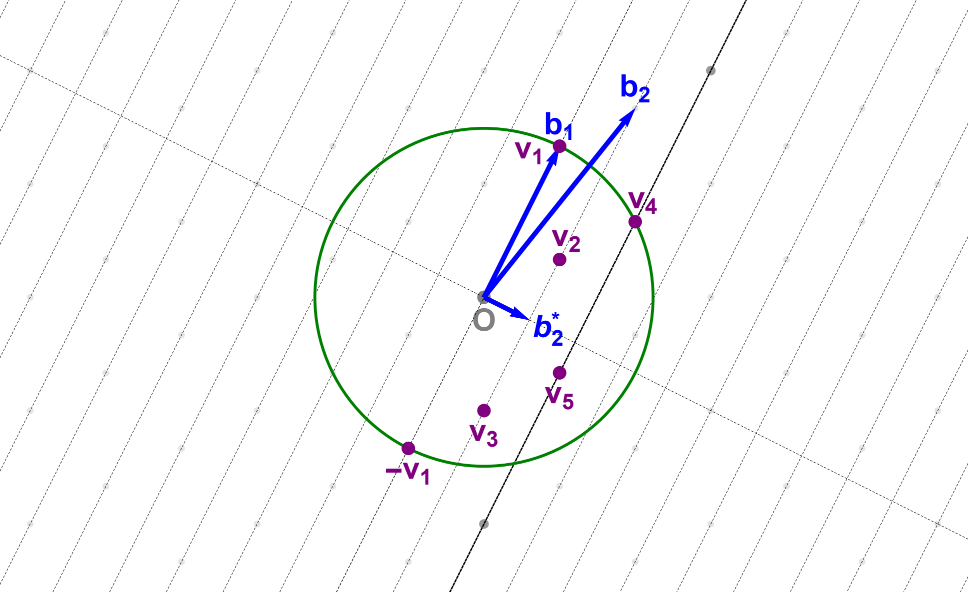
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



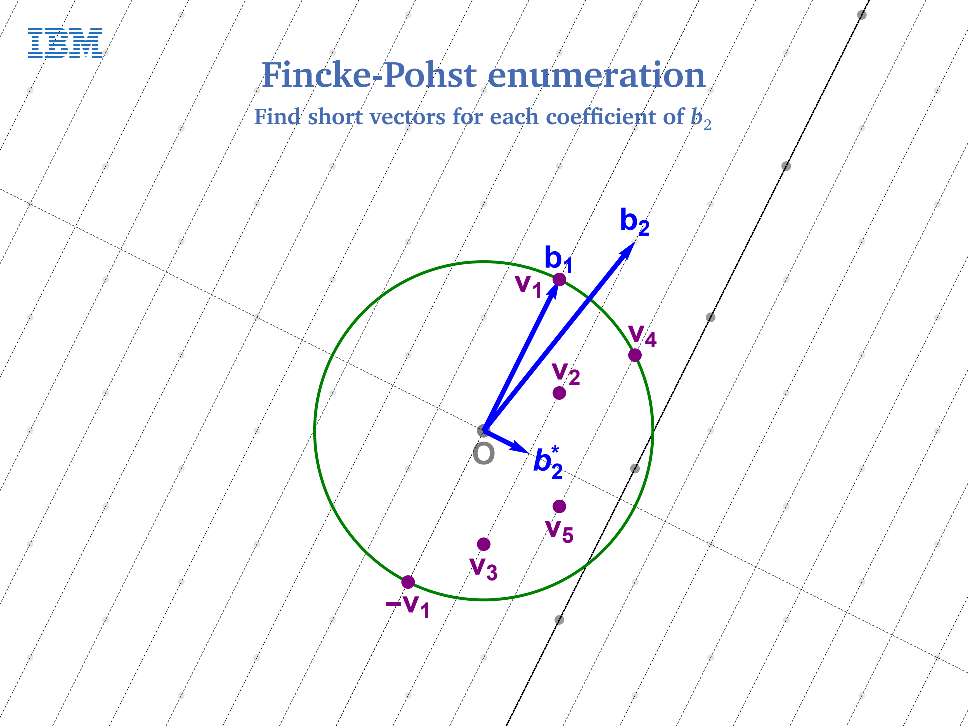
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



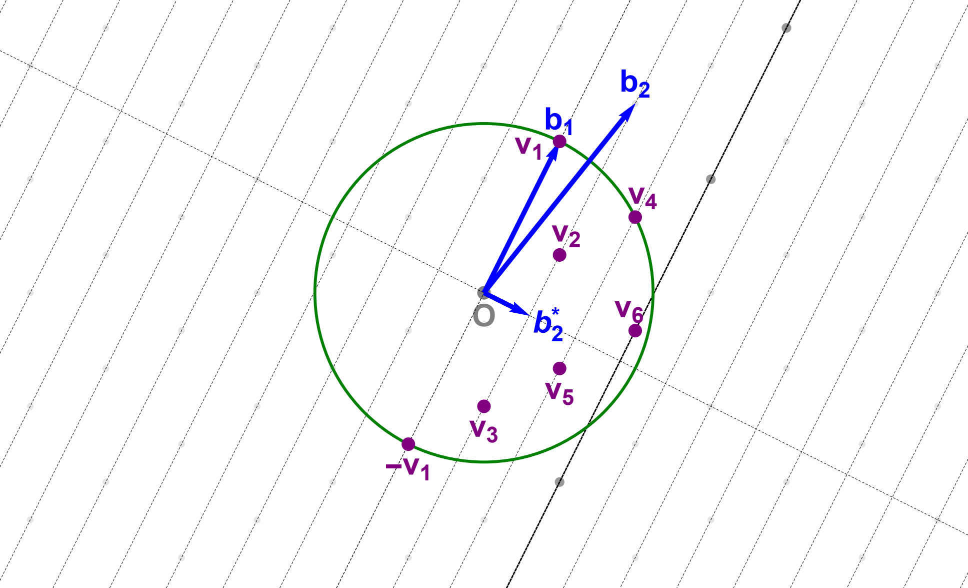
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



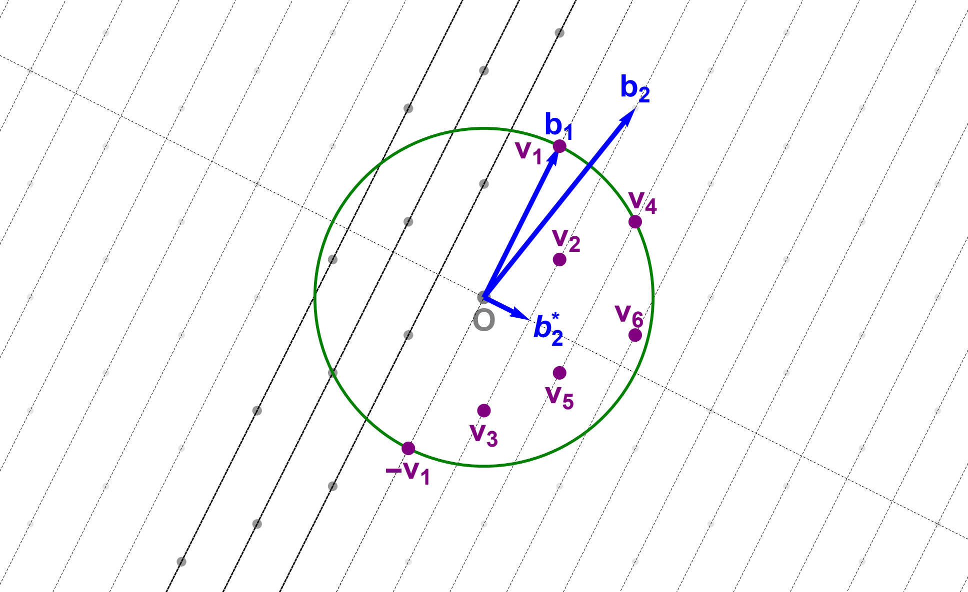
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



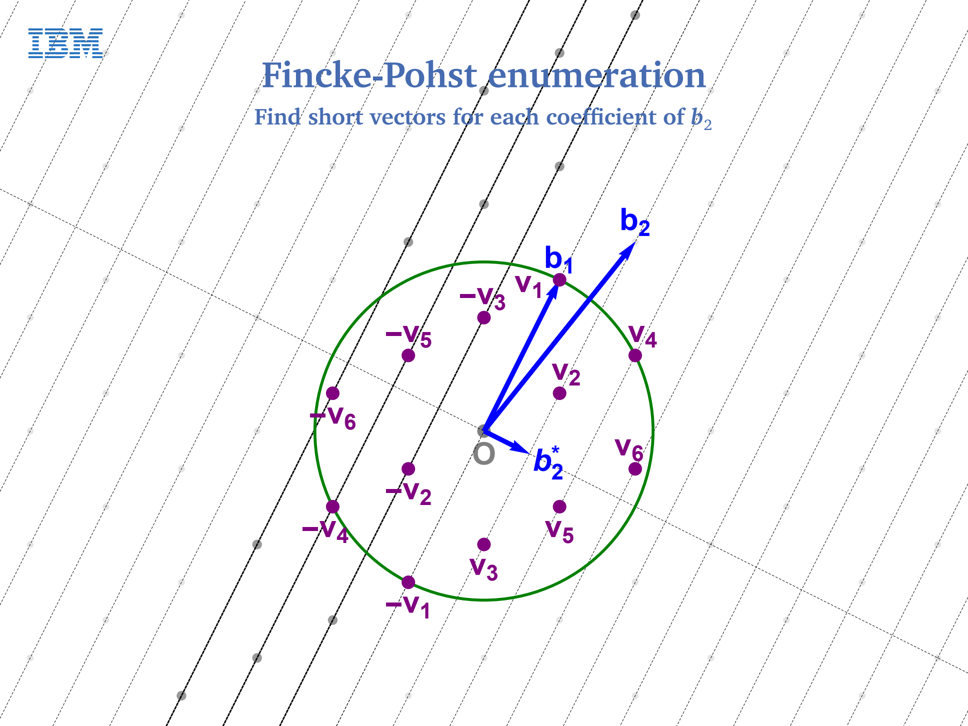
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



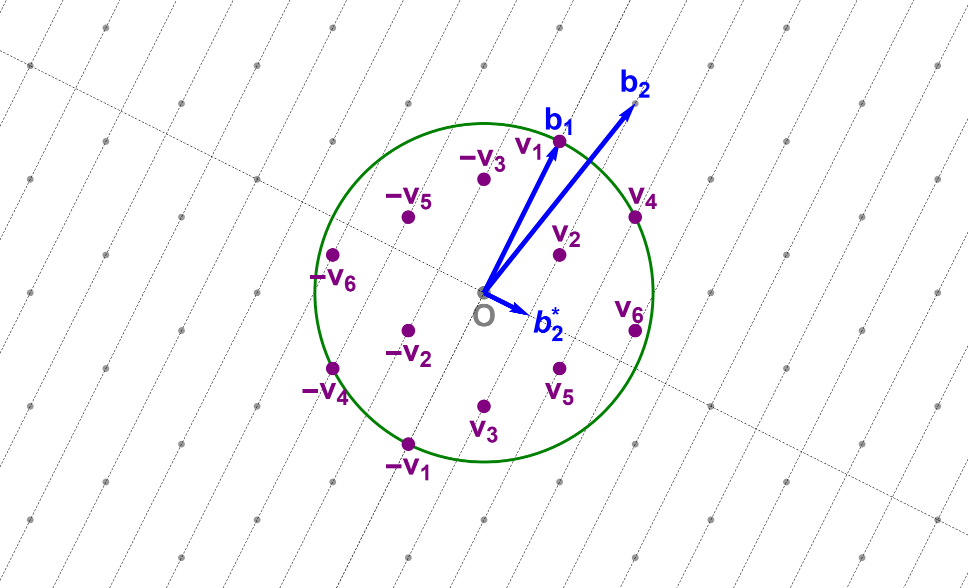
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



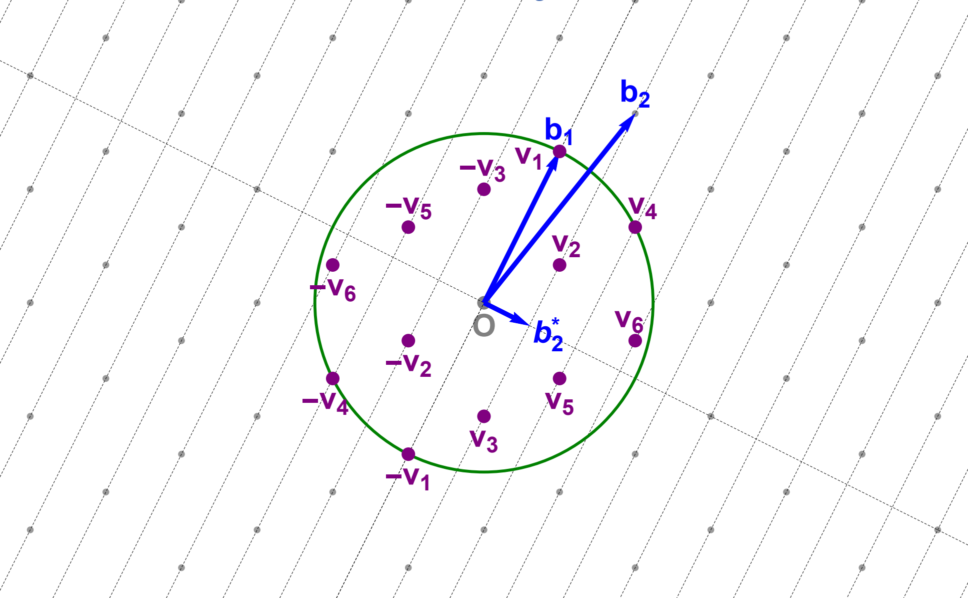
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



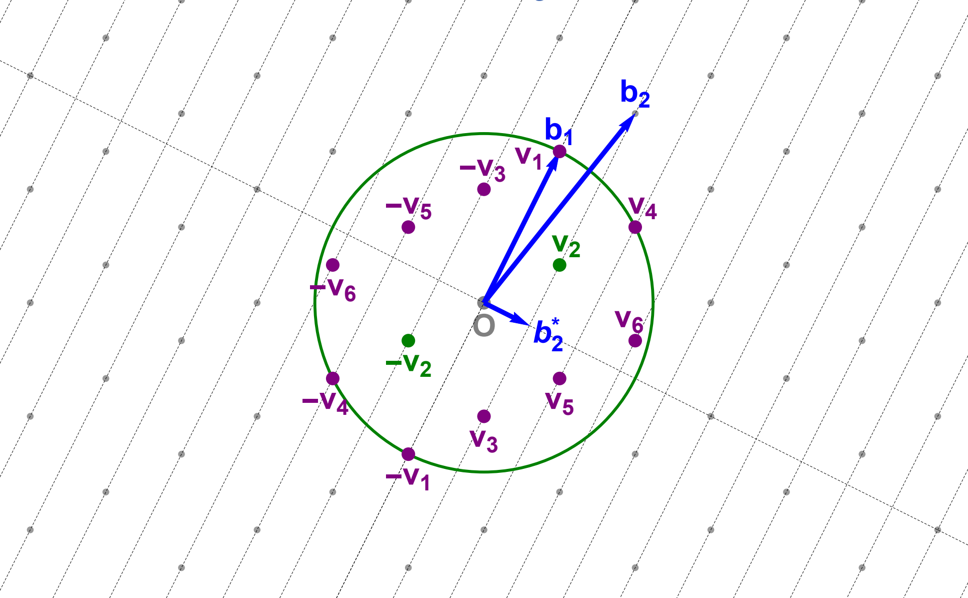
Fincke-Pohst enumeration

Find a shortest vector among all found vectors



Fincke-Pohst enumeration

Find a shortest vector among all found vectors



Fincke-Pohst enumeration

Overview

Fincke-Pohst enumeration

Overview

Theorem (Fincke–Pohst, Math. of Comp. '85)

Fincke-Pohst enumeration runs in time $2^{O(n^2)}$ *and space* $\mathrm{poly}(n)$.

Fincke-Pohst enumeration

Overview

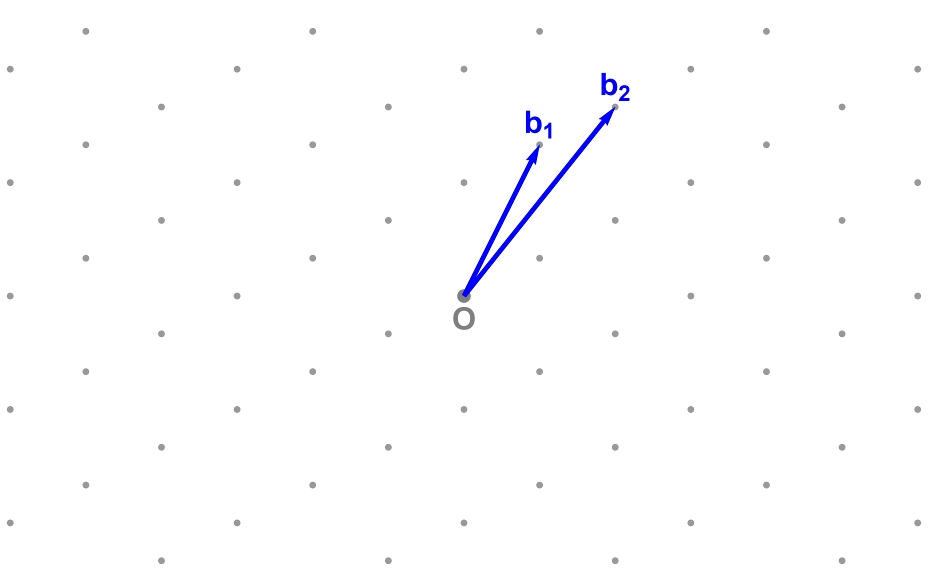
Theorem (Fincke–Pohst, Math. of Comp. '85)

Fincke-Pohst enumeration runs in time $2^{O(n^2)}$ *and space* $\text{poly}(n)$.

Essentially reduces SVP_n (CVP_n) to $2^{O(n)}$ instances of CVP_{n-1}

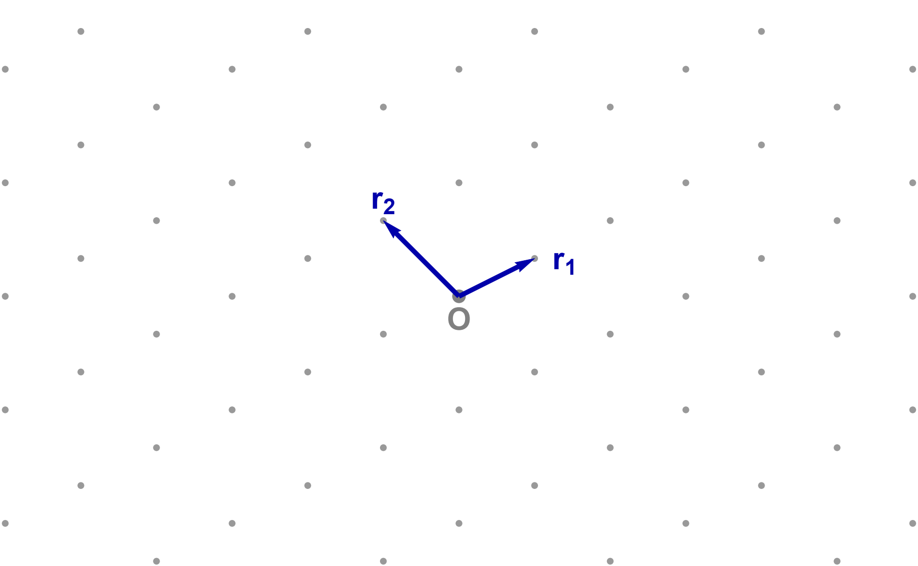
Kannan enumeration

Better bases



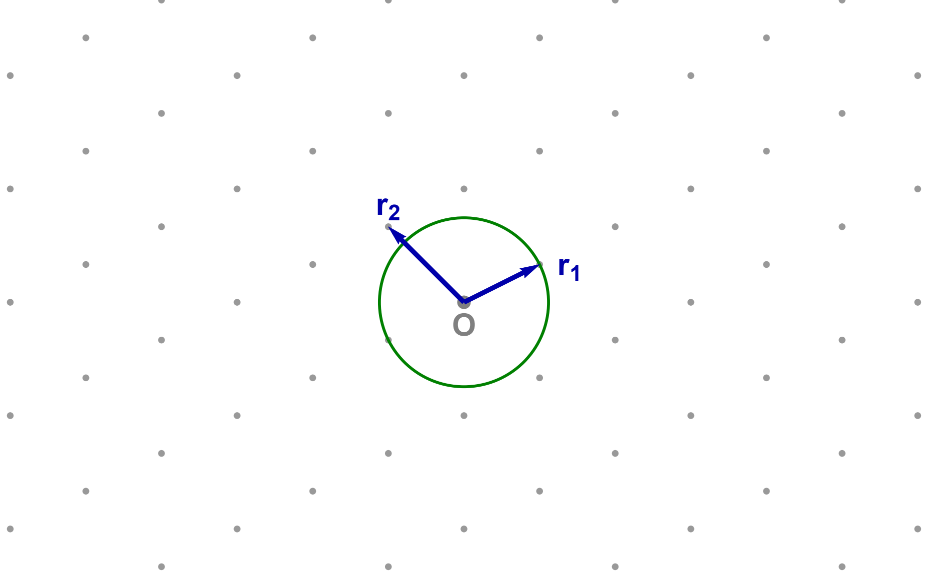
Kannan enumeration

Better bases



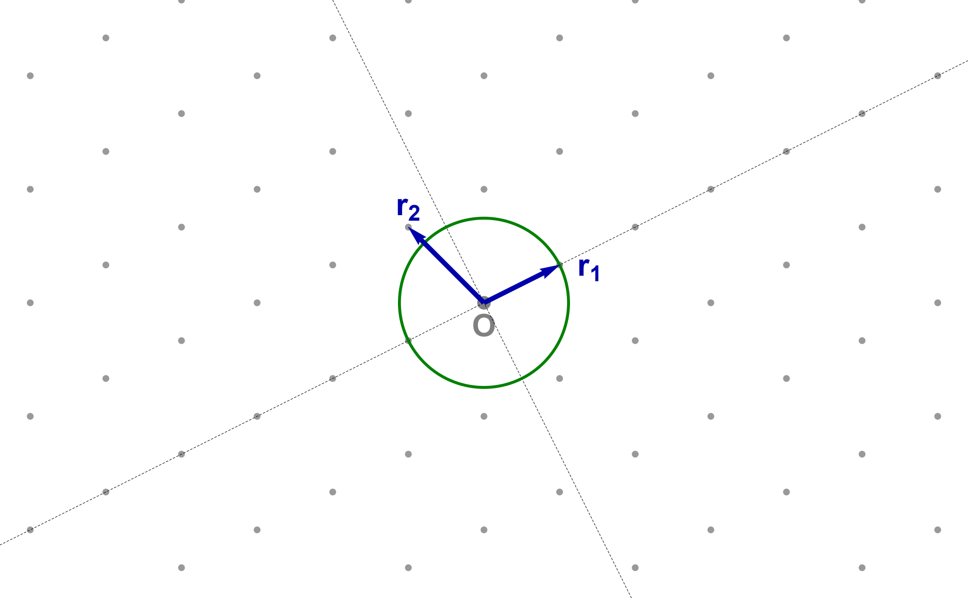
Kannan enumeration

Better bases



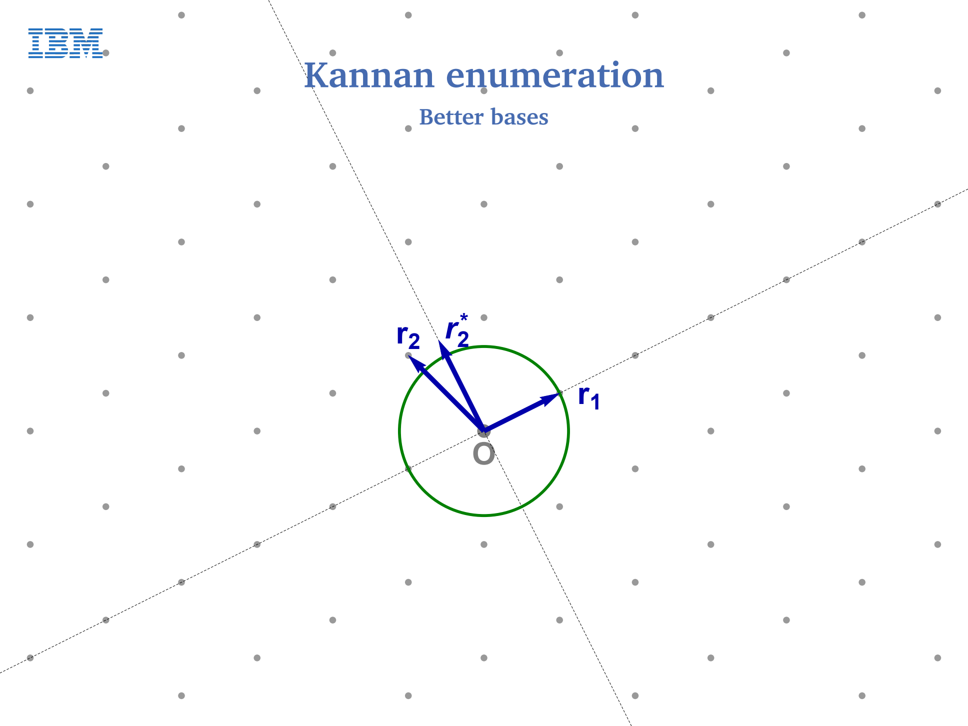
Kannan enumeration

Better bases



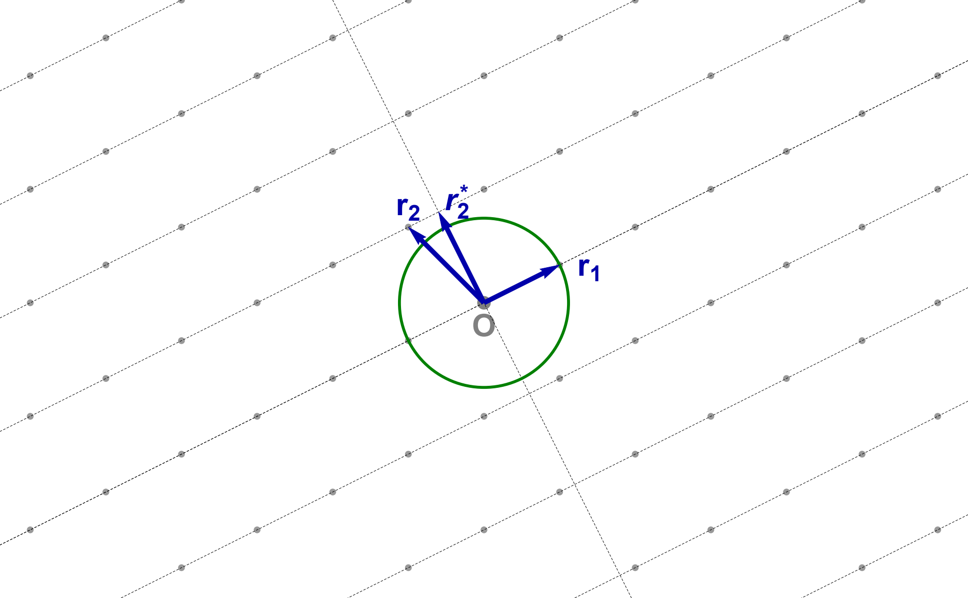
Kannan enumeration

Better bases



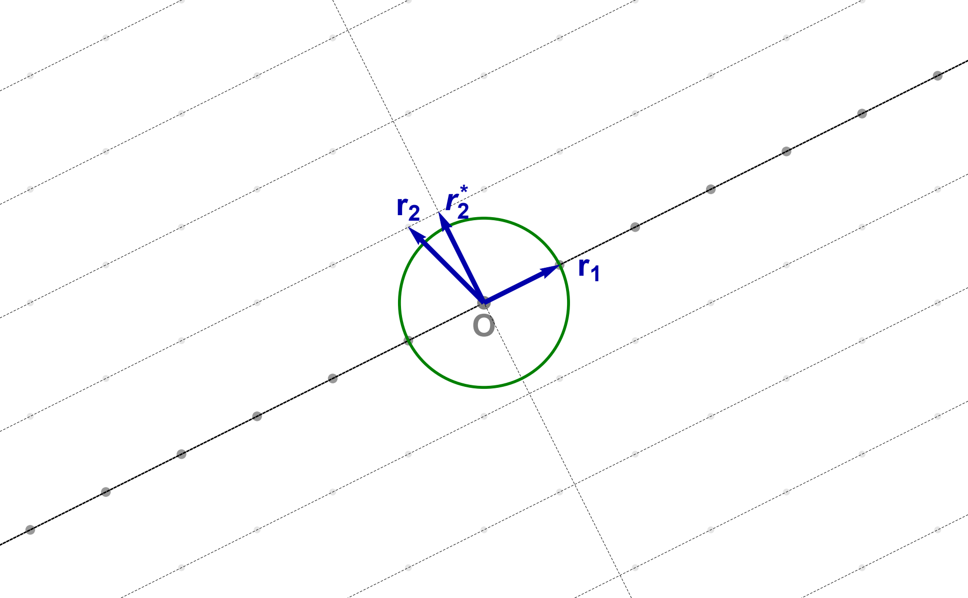
Kannan enumeration

Better bases



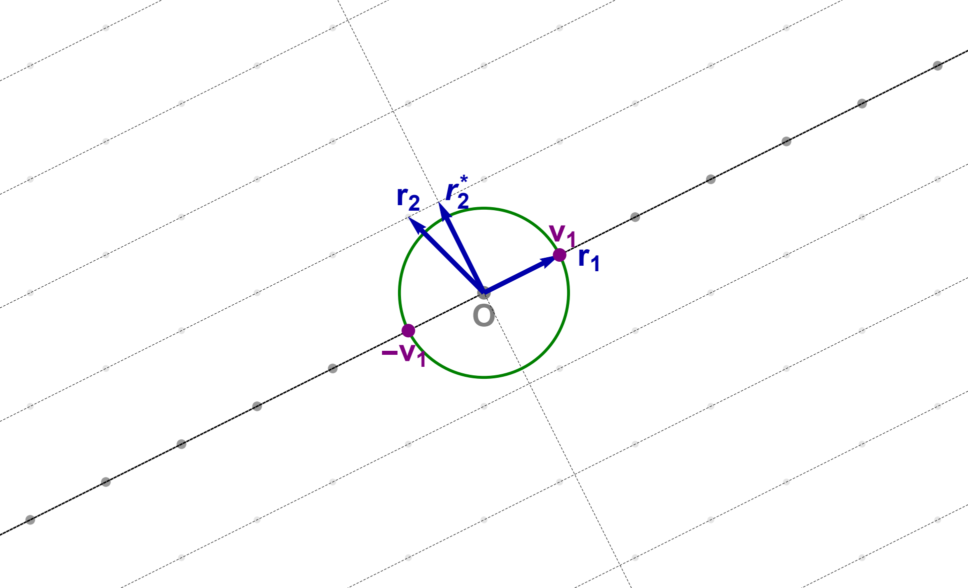
Kannan enumeration

Better bases



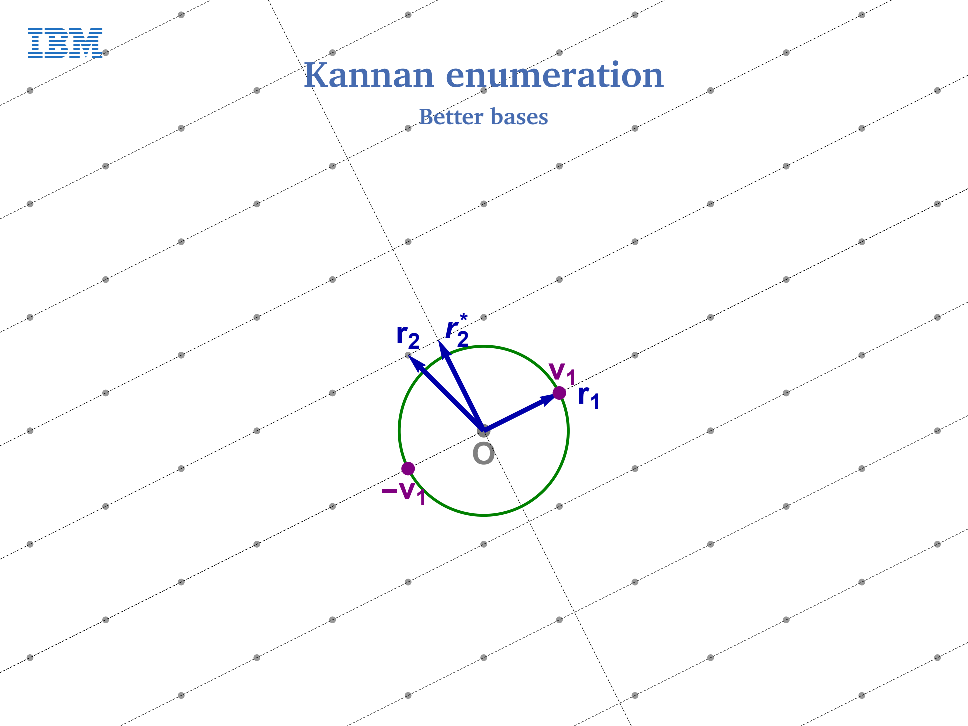
Kannan enumeration

Better bases



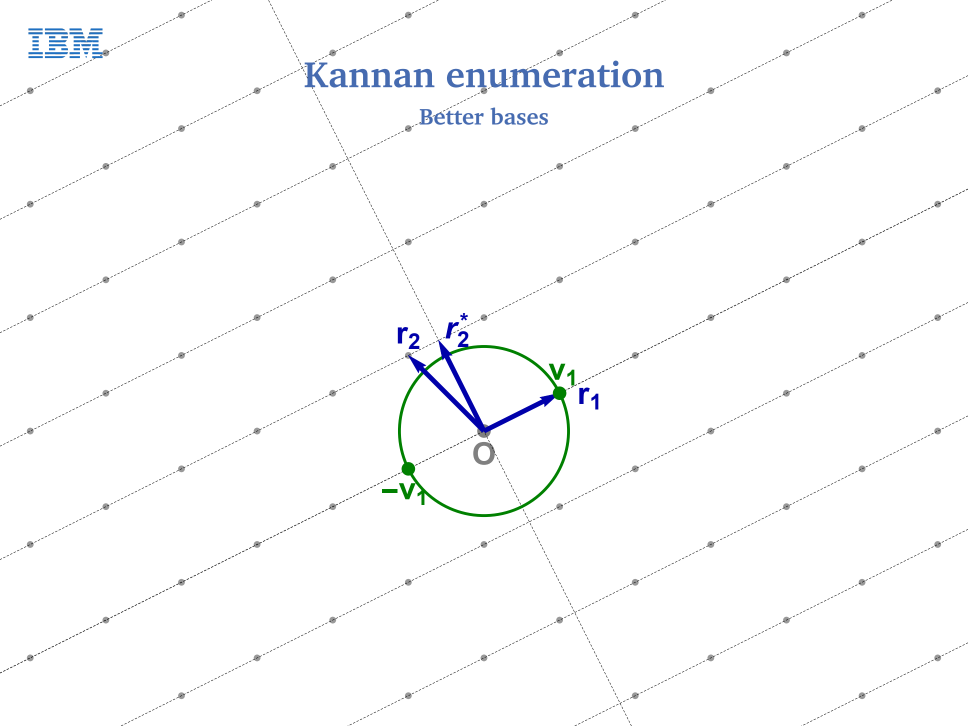
Kannan enumeration

Better bases



Kannan enumeration

Better bases



Kannan enumeration

Overview

Kannan enumeration

Overview

Theorem (Kannan, STOC'83)

Kannan enumeration runs in time $2^{O(n \log n)}$ *and space* $\mathrm{poly}(n)$.

Kannan enumeration

Overview

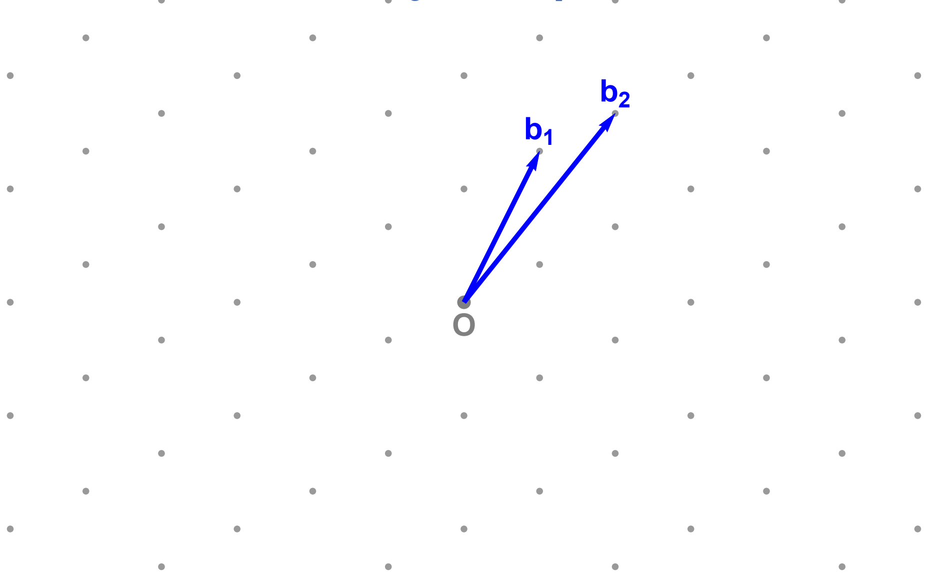
Theorem (Kannan, STOC'83)

Kannan enumeration runs in time $2^{O(n \log n)}$ *and space* $\mathrm{poly}(n)$.

> *"Our algorithm reduces an n-dimensional problem to polynomially many (instead of* $2^{O(n)}$*)* $(n-1)$*-dimensional problems. [...] The algorithm we propose, first finds a more orthogonal basis for a lattice in time* $2^{O(n \log n)}$*."*
>
> — Kannan, *STOC'83*

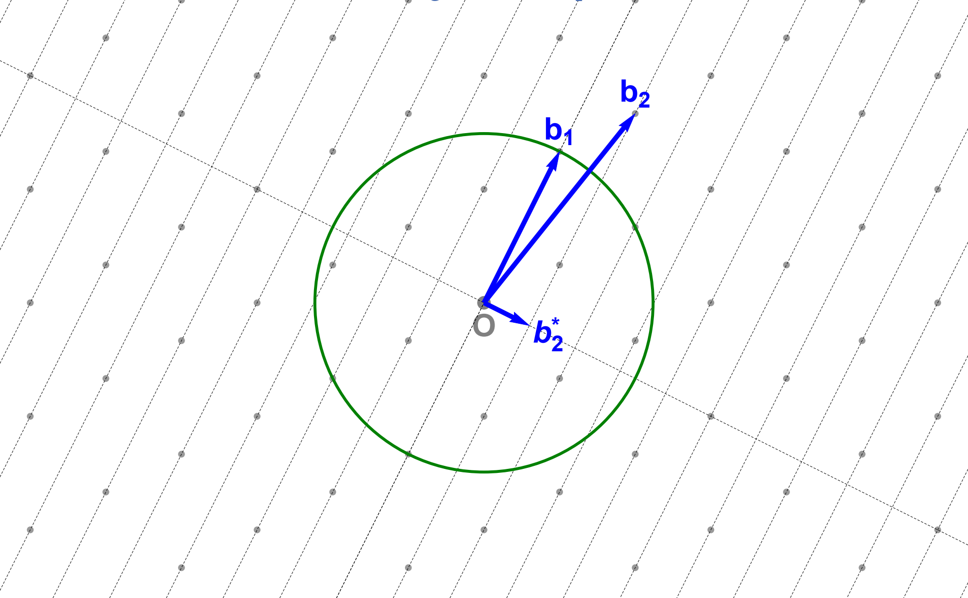
Pruned enumeration

Reducing the search space



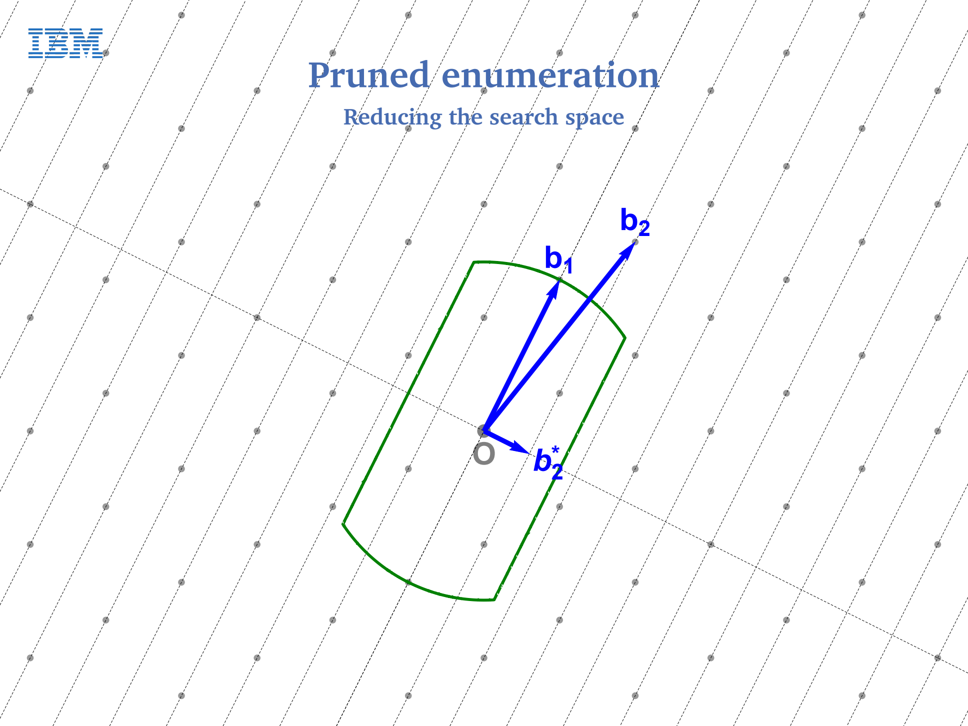
Pruned enumeration

Reducing the search space



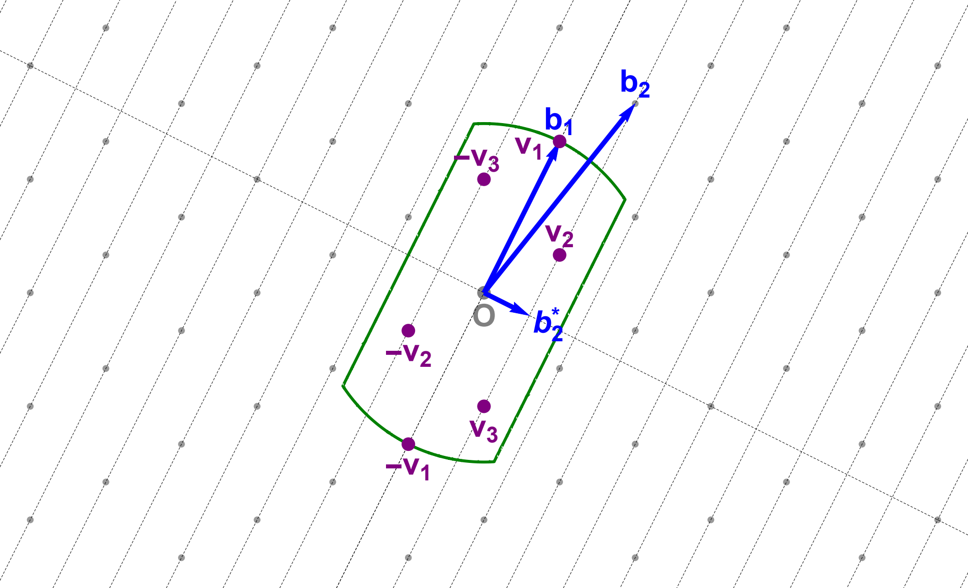
Pruned enumeration

Reducing the search space



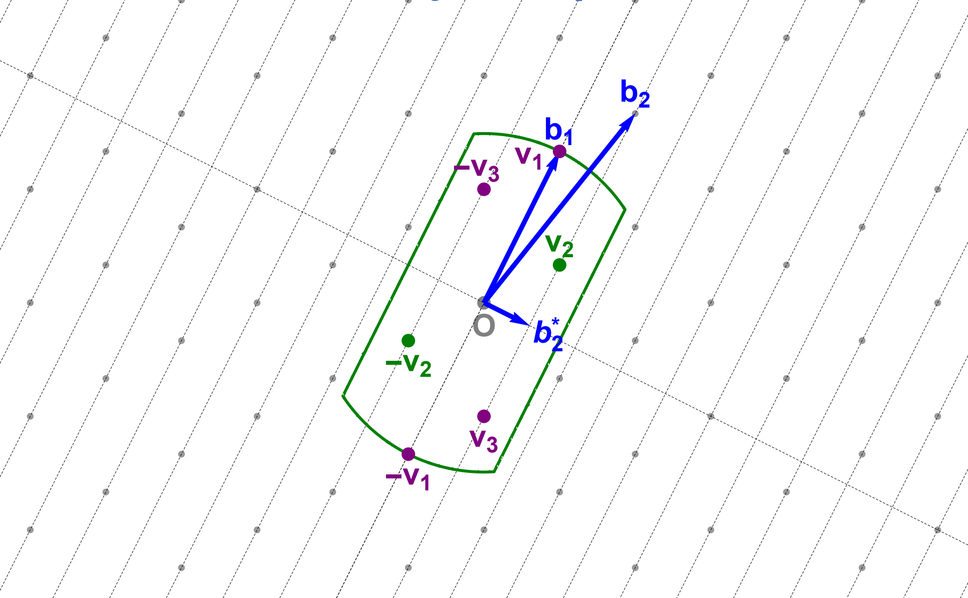
Pruned enumeration

Reducing the search space



Pruned enumeration

Reducing the search space



Pruned enumeration

Overview

Pruned enumeration

Overview

"Well-chosen bounding functions lead asymptotically to an exponential speedup of about $2^{n/4}$ over basic enumeration, maintaining a success probability $\geq$ 95%."

— Gama–Nguyen–Regev, *EUROCRYPT'10*

Pruned enumeration

Overview

"Well-chosen bounding functions lead asymptotically to an exponential speedup of about $2^{n/4}$ *over basic enumeration, maintaining a success probability* $\geq 95\%$*."*

— Gama–Nguyen–Regev, *EUROCRYPT'10*

"With extreme pruning, the probability of finding the desired vector is actually rather low (say, 0.1%), but surprisingly, the running time of the enumeration is reduced by a much more significant factor (say, much more than 1000)."

— Gama–Nguyen–Regev, *EUROCRYPT'10*

Outline

Lattices

Enumeration algorithms

- Fincke–Pohst enumeration
- Kannan enumeration
- (Extreme) pruning

Constructing the Voronoi cell

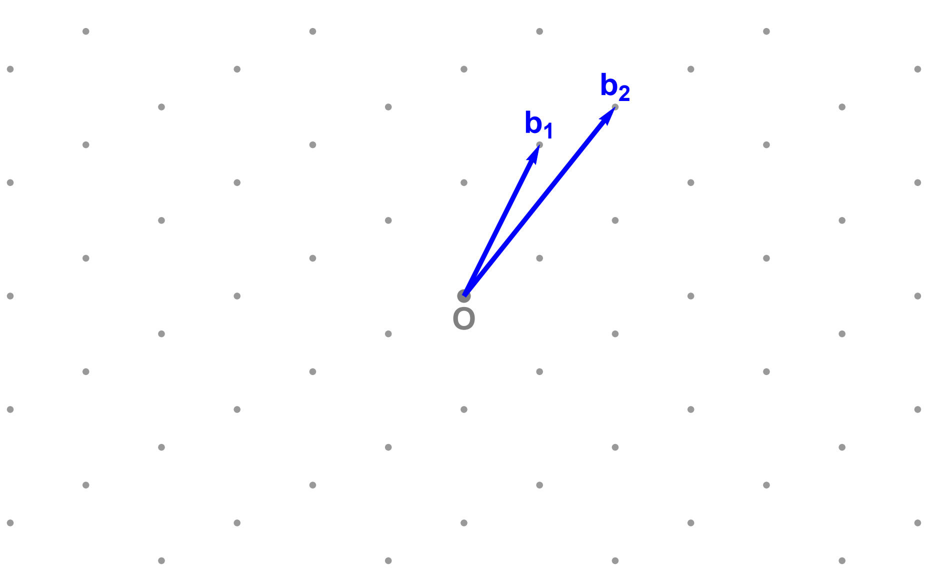
Sieving algorithms

- Basic sieving
- Leveled sieving
- Nearest neighbor searching

Conclusion

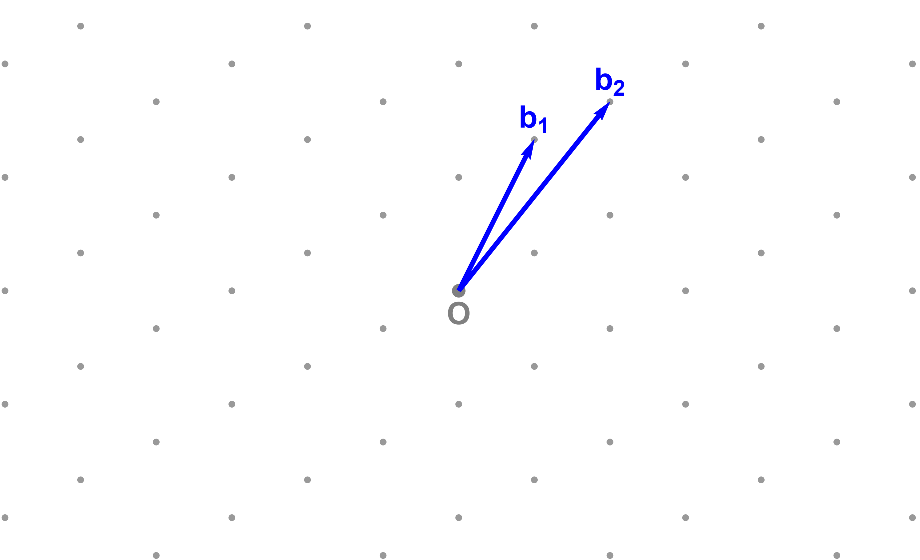
The Voronoi cell algorithm

Constructing the Voronoi cell of L



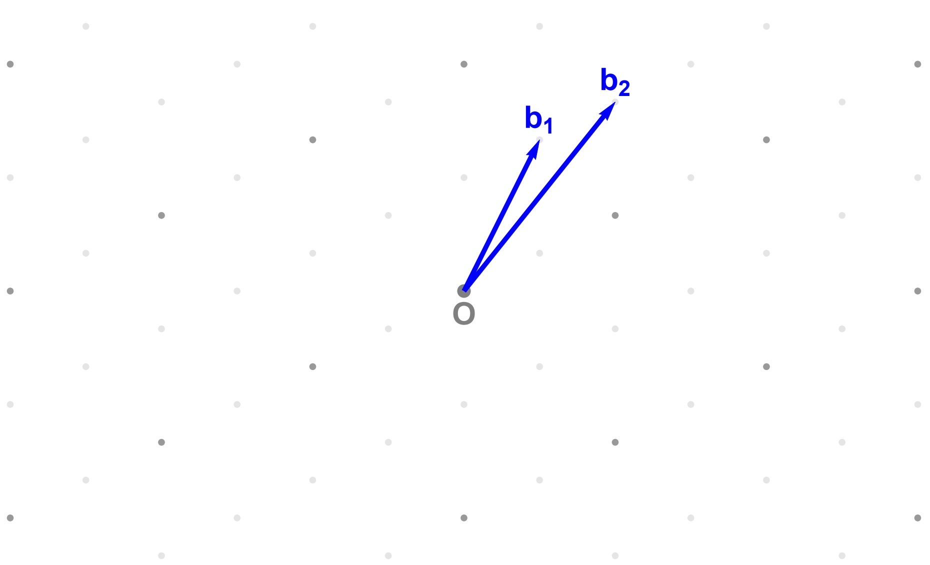
The Voronoi cell algorithm

Solve CVP in $2L$ for $\{0,1\}\mathbf{b}_1 + \cdots + \{0,1\}\mathbf{b}_n$



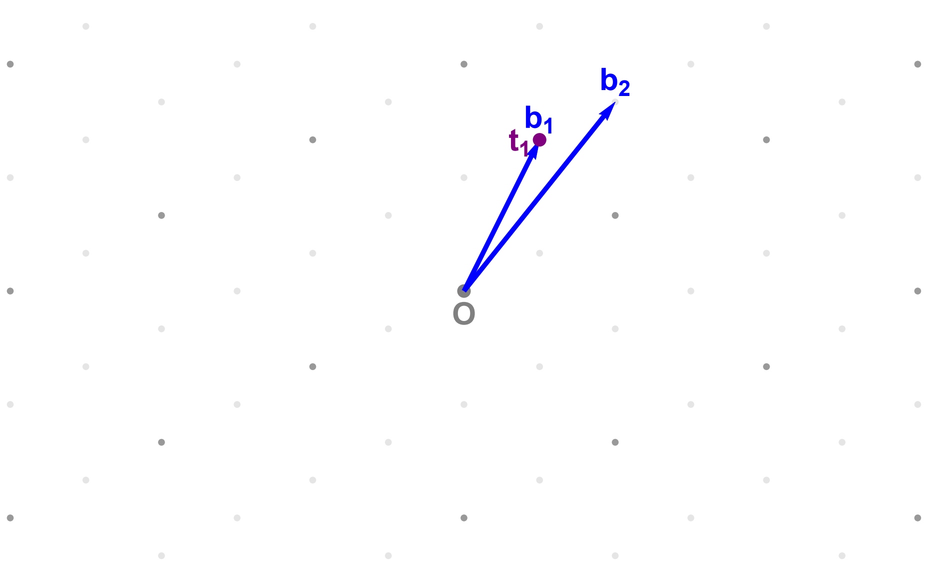
The Voronoi cell algorithm

Solve CVP in $2L$ for $\{0,1\}\boldsymbol{b}_1 + \cdots + \{0,1\}\boldsymbol{b}_n$



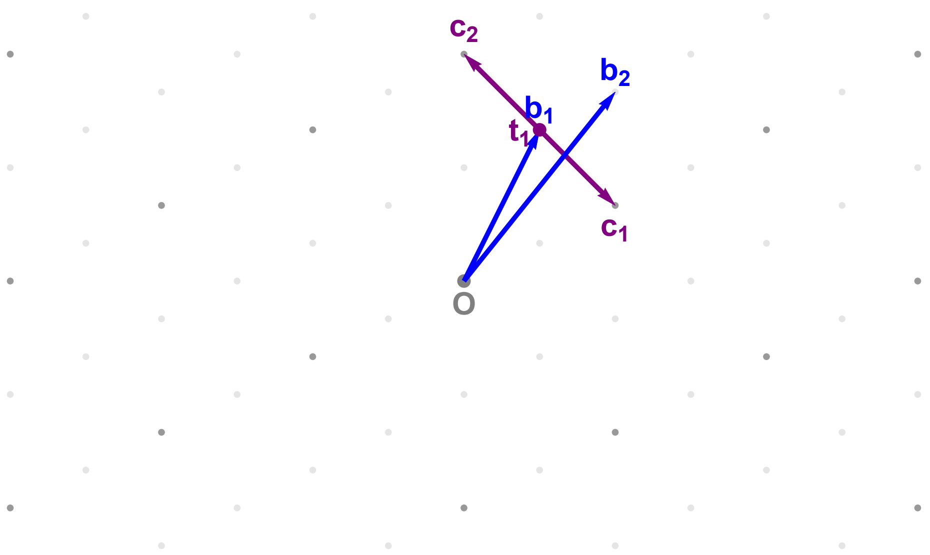
The Voronoi cell algorithm

Solve CVP in $2L$ for $\{0,1\}b_1 + \cdots + \{0,1\}b_n$



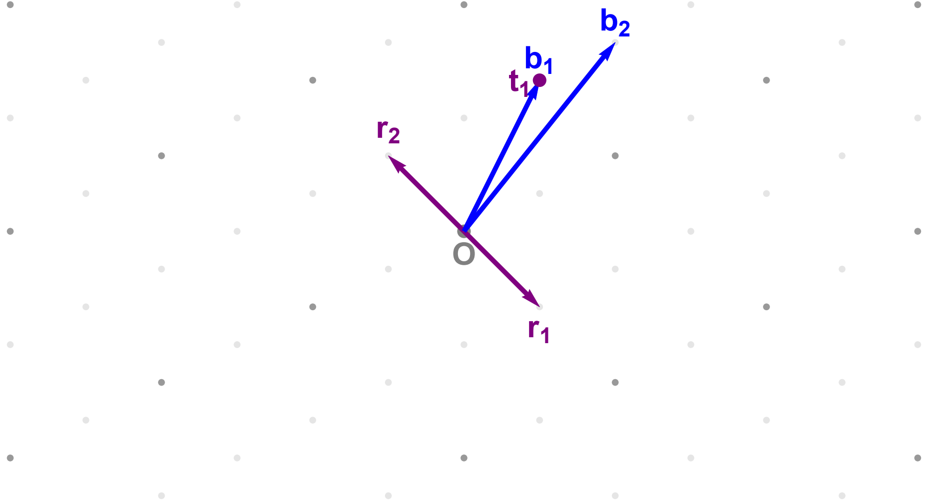
The Voronoi cell algorithm

Solve CVP in $2L$ for $\{0,1\}b_1 + \cdots + \{0,1\}b_n$



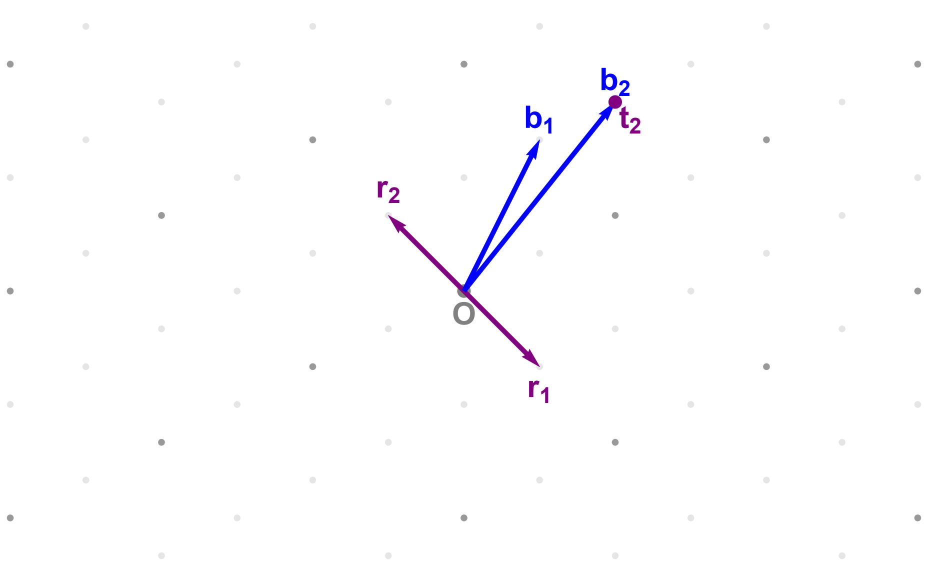
The Voronoi cell algorithm

Solve CVP in $2L$ for $\{0,1\}b_1 + \cdots + \{0,1\}b_n$



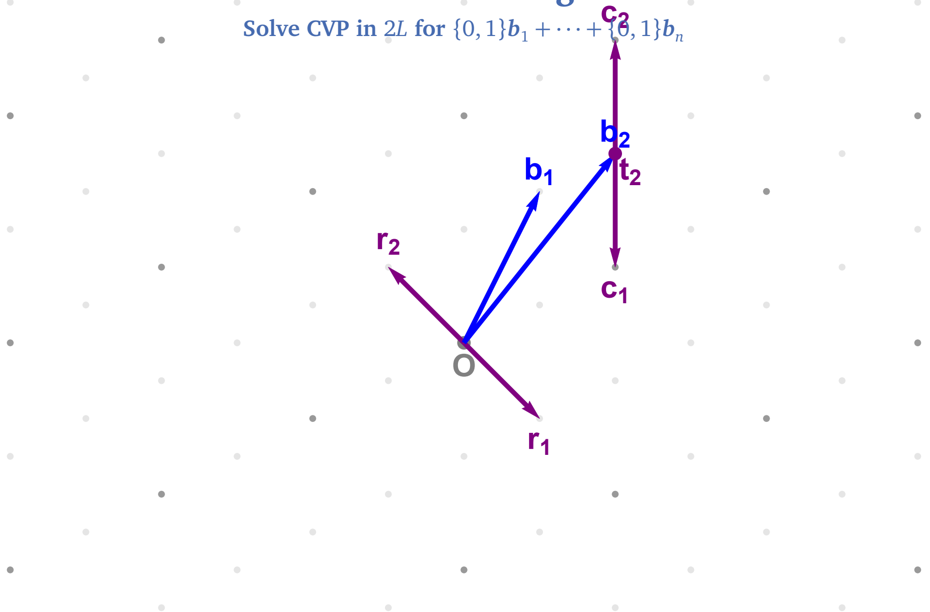
The Voronoi cell algorithm

Solve CVP in $2L$ for $\{0,1\}\boldsymbol{b}_1 + \cdots + \{0,1\}\boldsymbol{b}_n$



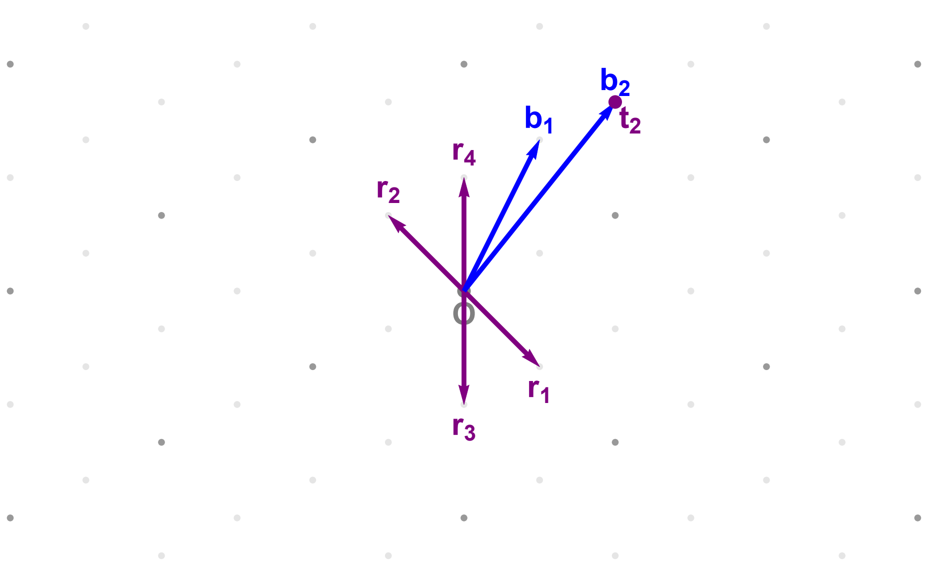
The Voronoi cell algorithm

Solve CVP in $2L$ for $\{0,1\}b_1 + \cdots + \{0,1\}b_n$



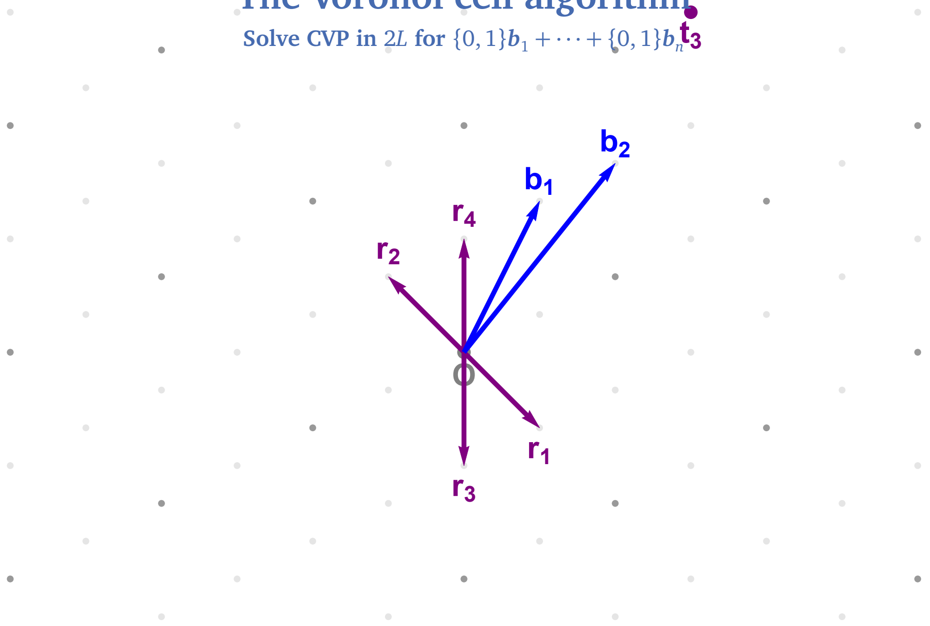
The Voronoi cell algorithm

Solve CVP in $2L$ for $\{0,1\}b_1 + \cdots + \{0,1\}b_n$



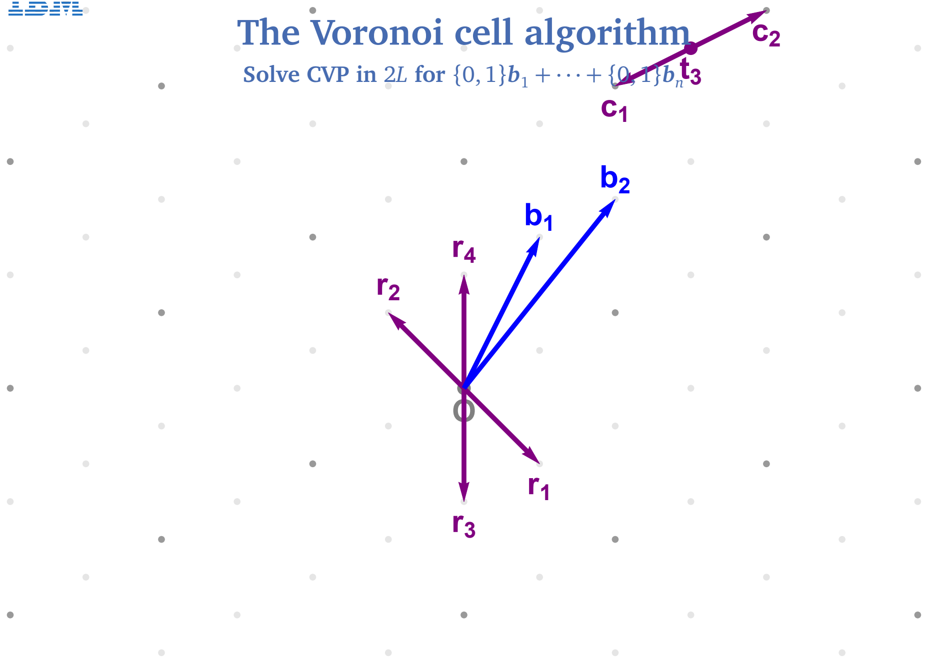
The Voronoi cell algorithm

Solve CVP in $2L$ for $\{0,1\}b_1 + \cdots + \{0,1\}b_n$



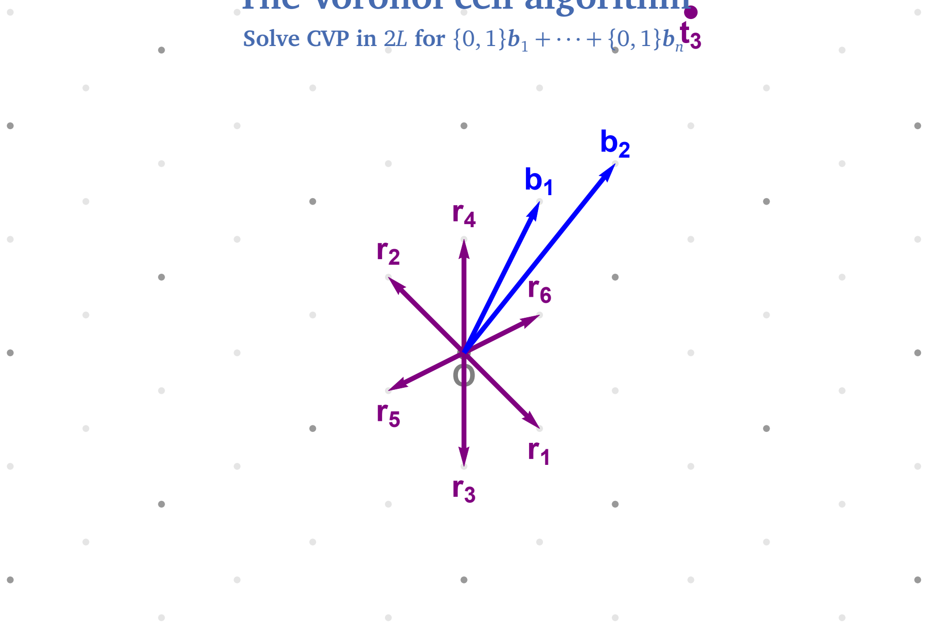
The Voronoi cell algorithm

Solve CVP in $2L$ for $\{0,1\}b_1 + \cdots + \{0,1\}b_n$



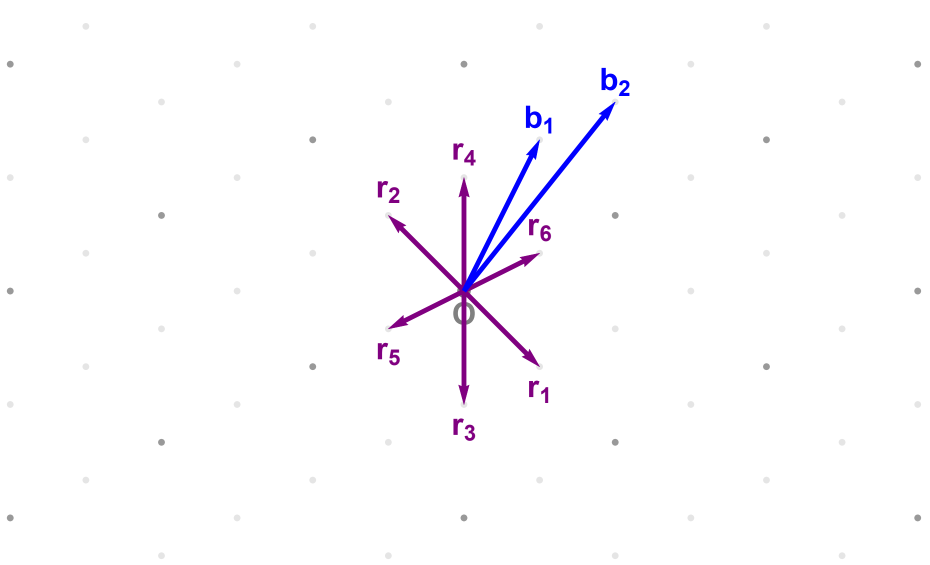
The Voronoi cell algorithm

Solve CVP in $2L$ for $\{0,1\}b_1 + \cdots + \{0,1\}b_n$



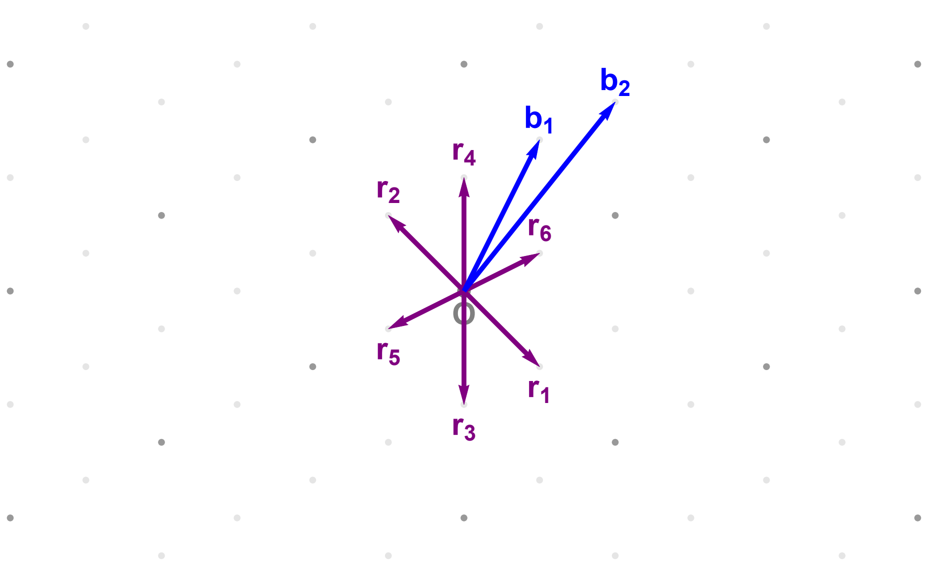
The Voronoi cell algorithm

Solve CVP in $2L$ for $\{0,1\}b_1 + \cdots + \{0,1\}b_n$



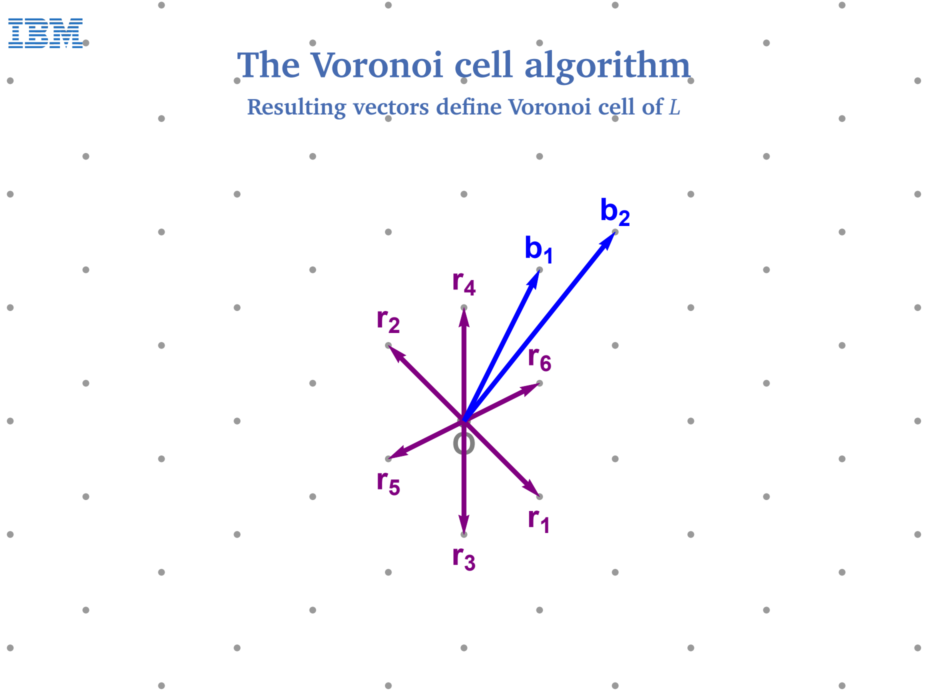
The Voronoi cell algorithm

Resulting vectors define Voronoi cell of L



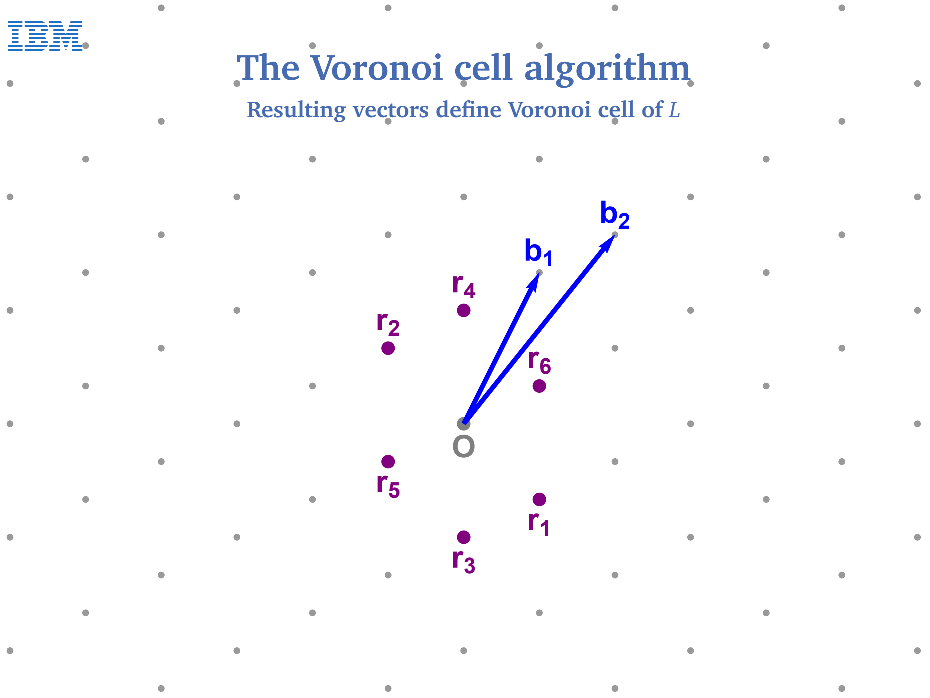
The Voronoi cell algorithm

Resulting vectors define Voronoi cell of L



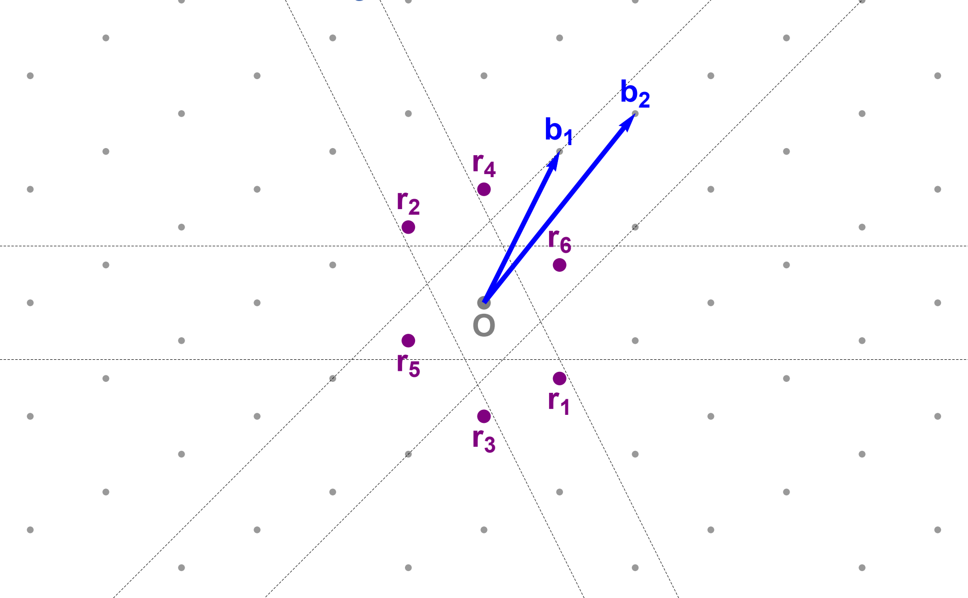
The Voronoi cell algorithm

Resulting vectors define Voronoi cell of L



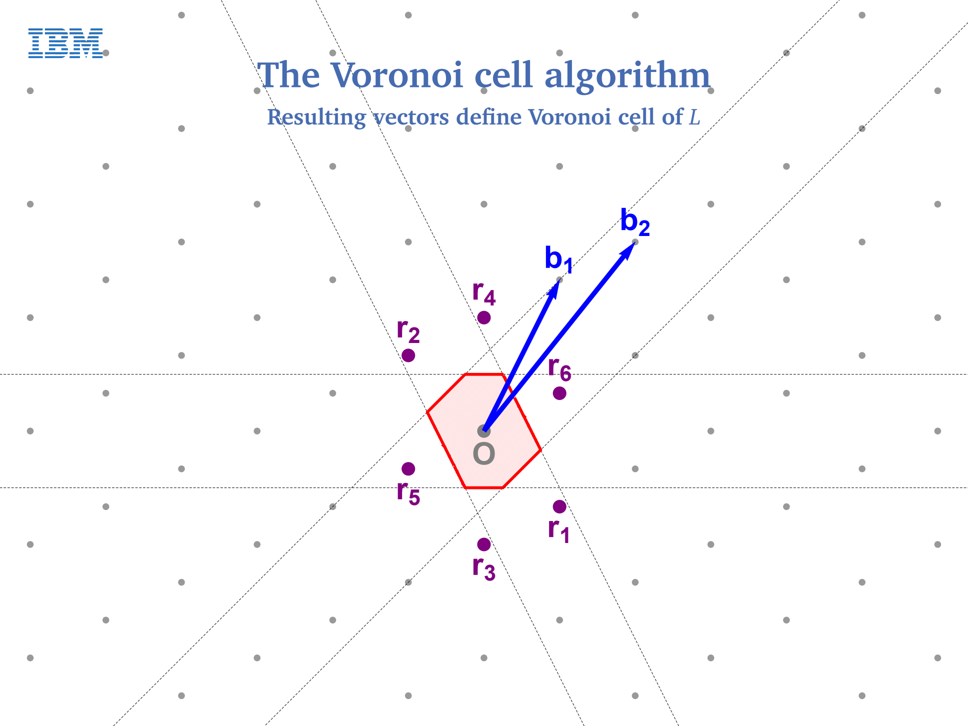
The Voronoi cell algorithm

Resulting vectors define Voronoi cell of L



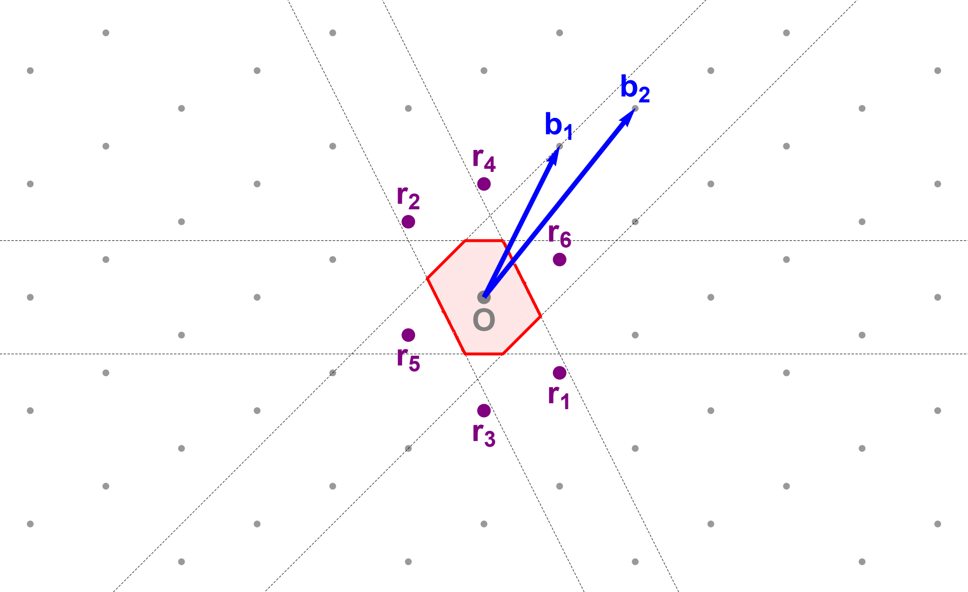
The Voronoi cell algorithm

Resulting vectors define Voronoi cell of L



The Voronoi cell algorithm

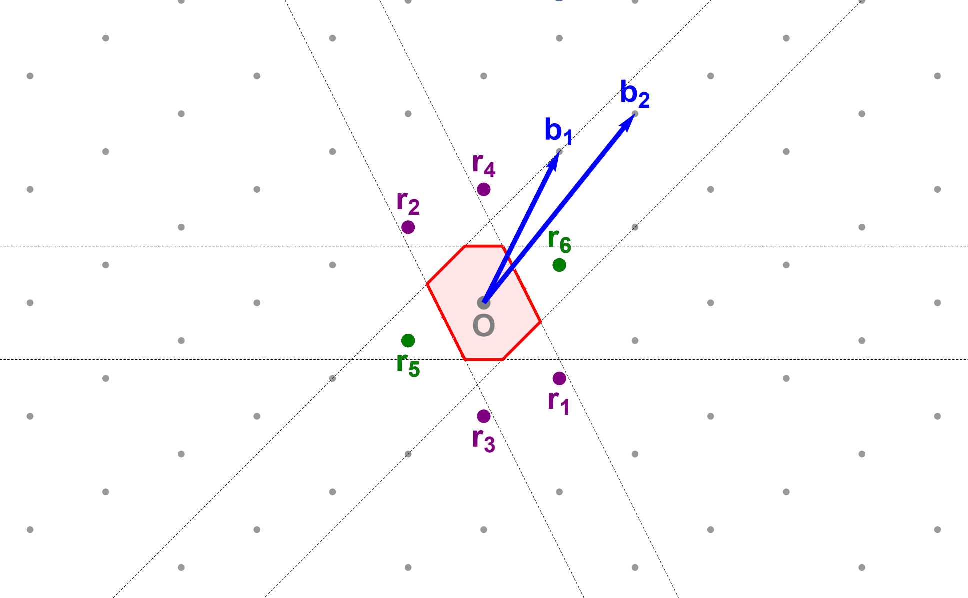
Find a shortest vector among these vectors



IBM

The Voronoi cell algorithm

Find a shortest vector among these vectors



The Voronoi cell algorithm

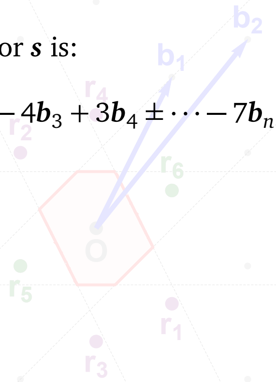
Analysis

The Voronoi cell algorithm

Analysis

Suppose the shortest vector $\boldsymbol{s}$ is:

$$\boldsymbol{s} = 5\boldsymbol{b}_1 + 2\boldsymbol{b}_2 - 4\boldsymbol{b}_3 + 3\boldsymbol{b}_4 \pm \cdots - 7\boldsymbol{b}_n$$



The Voronoi cell algorithm

Analysis

Suppose the shortest vector $\boldsymbol{s}$ is:

$$\boldsymbol{s} = 5\boldsymbol{b}_1 + 2\boldsymbol{b}_2 - 4\boldsymbol{b}_3 + 3\boldsymbol{b}_4 \pm \cdots - 7\boldsymbol{b}_n$$

Let the vector $\boldsymbol{t} \in \{0,1\}^n \cdot B$ be such that $\boldsymbol{s} + \boldsymbol{t} \in 2L$:

$$\begin{aligned} \boldsymbol{t} &= 1\boldsymbol{b}_1 + 0\boldsymbol{b}_2 + 0\boldsymbol{b}_3 + 1\boldsymbol{b}_4 \pm \cdots + 1\boldsymbol{b}_n \\ \boldsymbol{s} + \boldsymbol{t} &= 6\boldsymbol{b}_1 + 2\boldsymbol{b}_2 - 4\boldsymbol{b}_3 + 4\boldsymbol{b}_4 \pm \cdots - 6\boldsymbol{b}_n \qquad (\in 2L) \end{aligned}$$

The Voronoi cell algorithm

Analysis

Suppose the shortest vector $\boldsymbol{s}$ is:

$$\boldsymbol{s} = 5\boldsymbol{b}_1 + 2\boldsymbol{b}_2 - 4\boldsymbol{b}_3 + 3\boldsymbol{b}_4 \pm \cdots - 7\boldsymbol{b}_n$$

Let the vector $\boldsymbol{t} \in \{0,1\}^n \cdot B$ be such that $\boldsymbol{s} + \boldsymbol{t} \in 2L$:

$$\begin{aligned} \boldsymbol{t} &= 1\boldsymbol{b}_1 + 0\boldsymbol{b}_2 + 0\boldsymbol{b}_3 + 1\boldsymbol{b}_4 \pm \cdots + 1\boldsymbol{b}_n \\ \boldsymbol{s} + \boldsymbol{t} &= 6\boldsymbol{b}_1 + 2\boldsymbol{b}_2 - 4\boldsymbol{b}_3 + 4\boldsymbol{b}_4 \pm \cdots - 6\boldsymbol{b}_n \qquad (\in 2L) \end{aligned}$$

The vectors closest to $\boldsymbol{t}$ in the sublattice $2L$ are $\boldsymbol{t} \pm \boldsymbol{s}$.

The Voronoi cell algorithm

Overview

Theorem (Micciancio–Voulgaris, SODA'10)

The Voronoi cell algorithm can solve SVP in time $2^{2n+o(n)}$ *and space* $2^{n+o(n)}$.

The Voronoi cell algorithm

Overview

Theorem (Micciancio–Voulgaris, SODA'10)

The Voronoi cell algorithm can solve SVP in time $2^{2n+o(n)}$ *and space* $2^{n+o(n)}$.

Essentially reduces SVP_n (CVP_n) to CVP_{n-1} with $2^{O(n)}$ overhead

Outline

Lattices

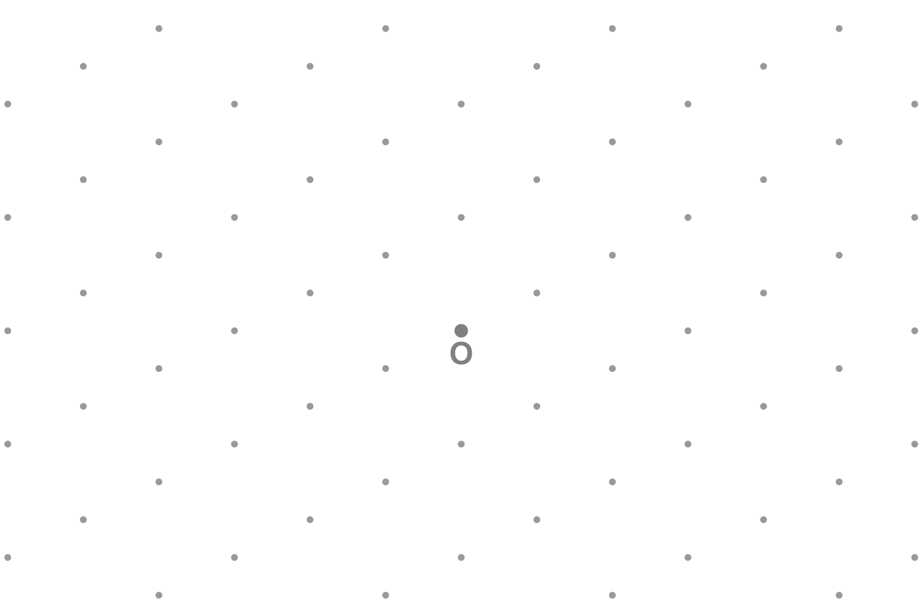
Enumeration algorithms
- Fincke–Pohst enumeration
- Kannan enumeration
- (Extreme) pruning

Constructing the Voronoi cell

Sieving algorithms
- Basic sieving
- Leveled sieving
- Nearest neighbor searching

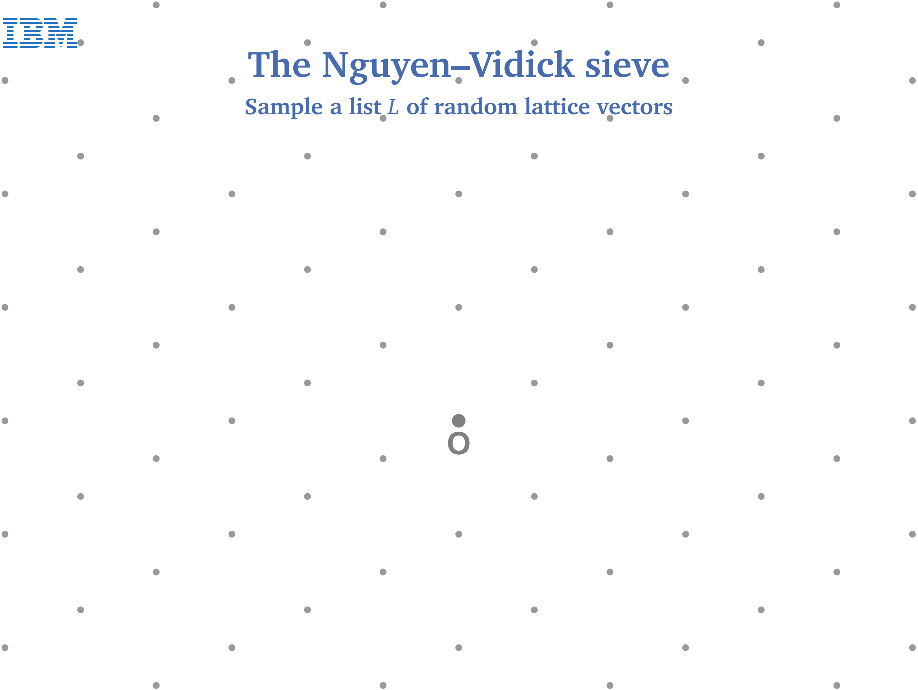
Conclusion

The Nguyen–Vidick sieve



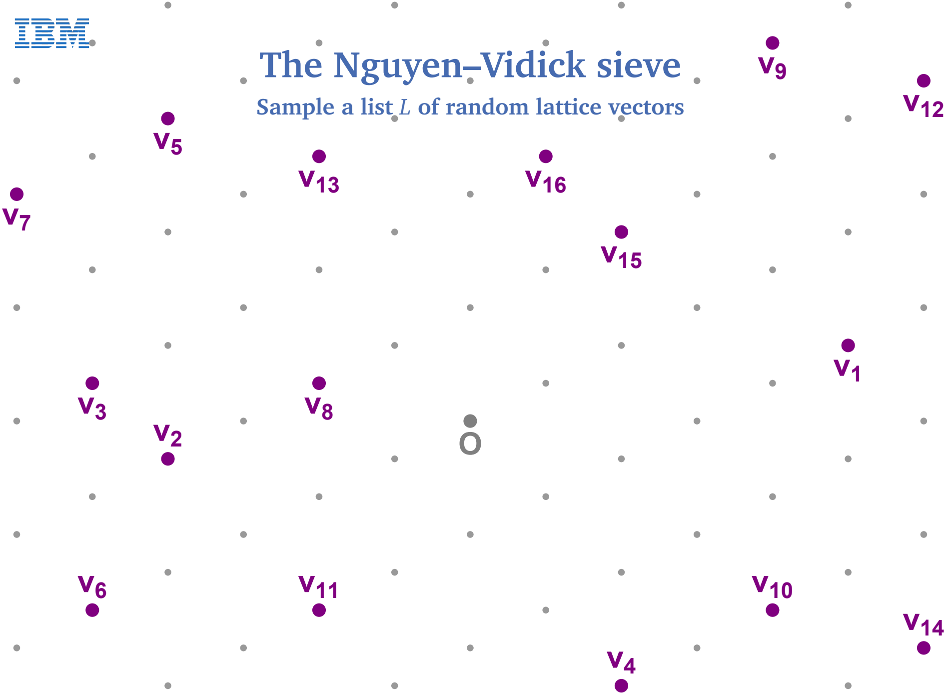
The Nguyen–Vidick sieve

Sample a list L of random lattice vectors



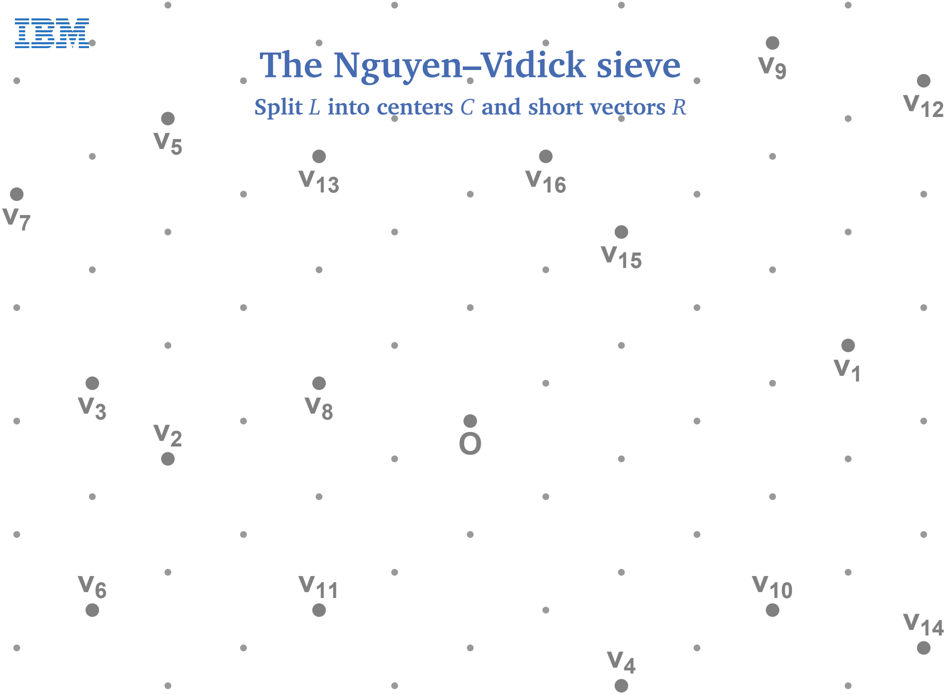
The Nguyen–Vidick sieve

Sample a list L of random lattice vectors



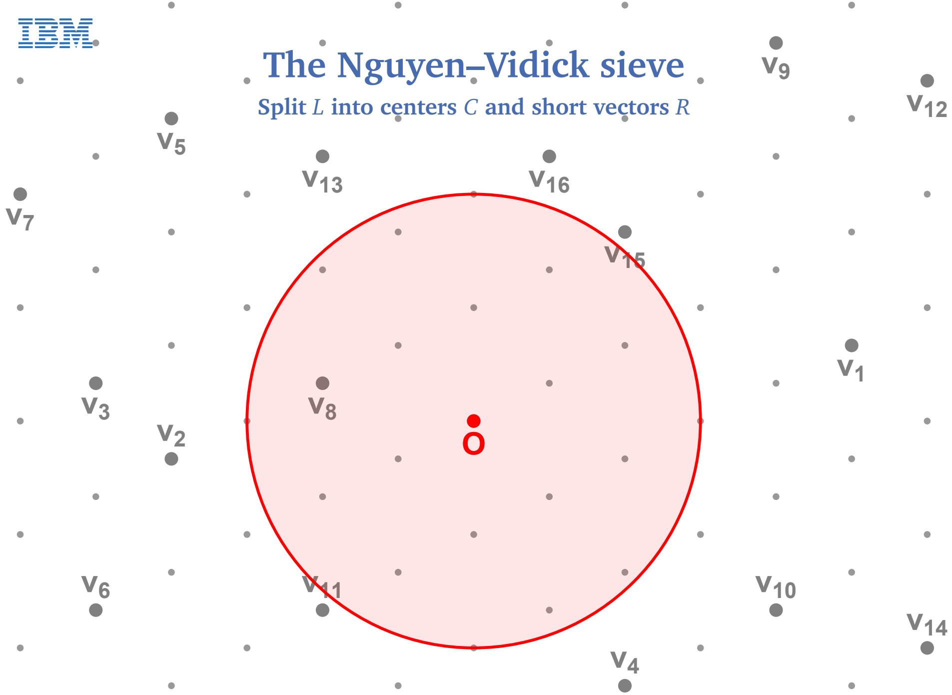
The Nguyen–Vidick sieve

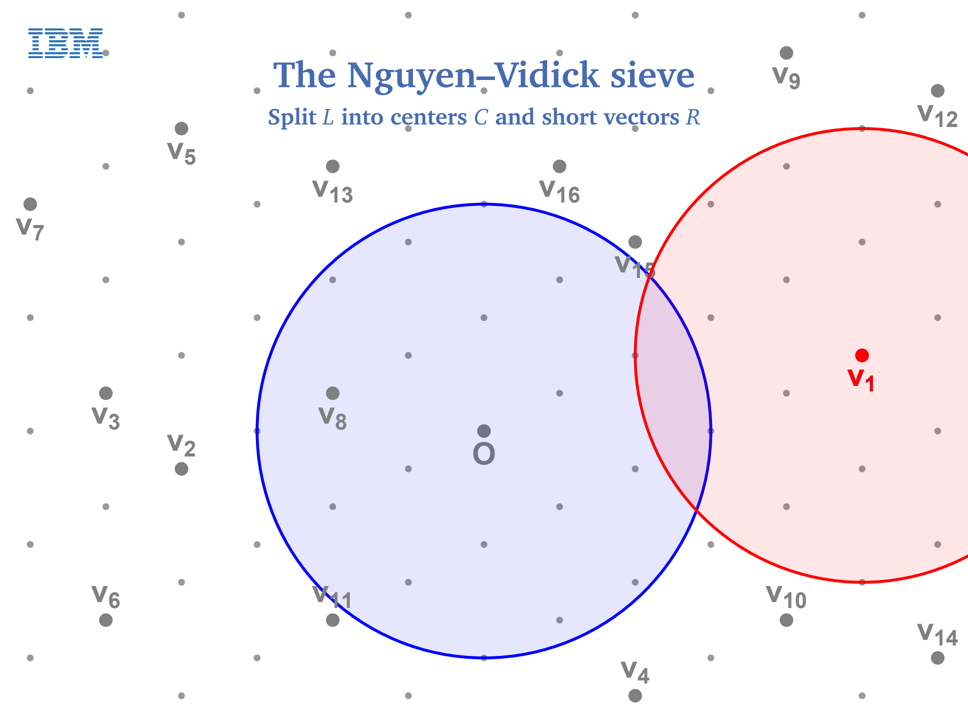
Split L into centers C and short vectors R

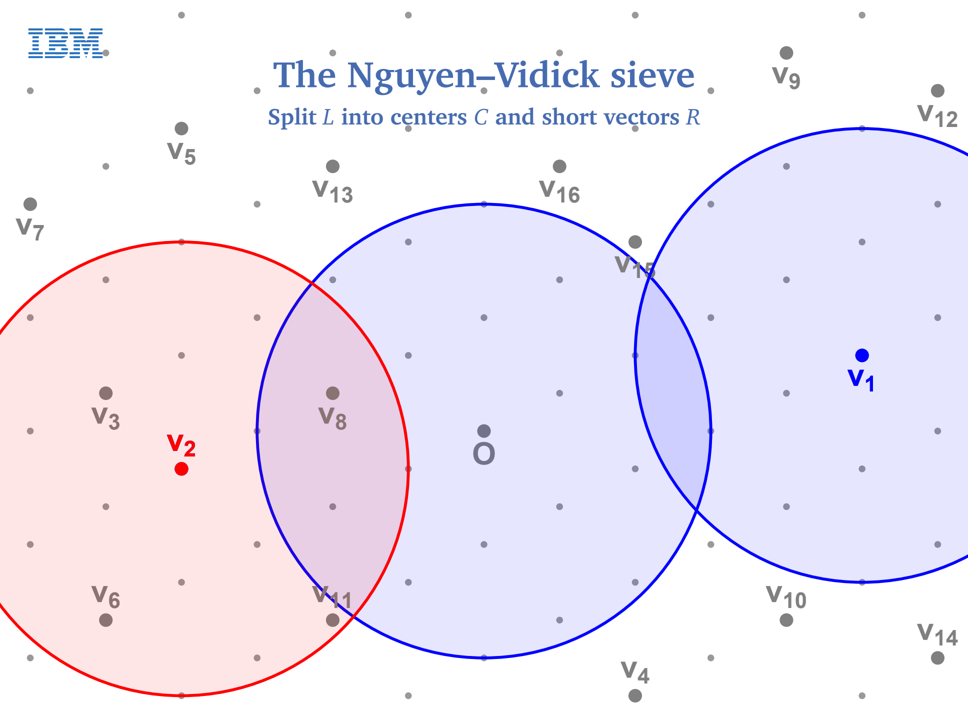


The Nguyen–Vidick sieve

Split L into centers C and short vectors R

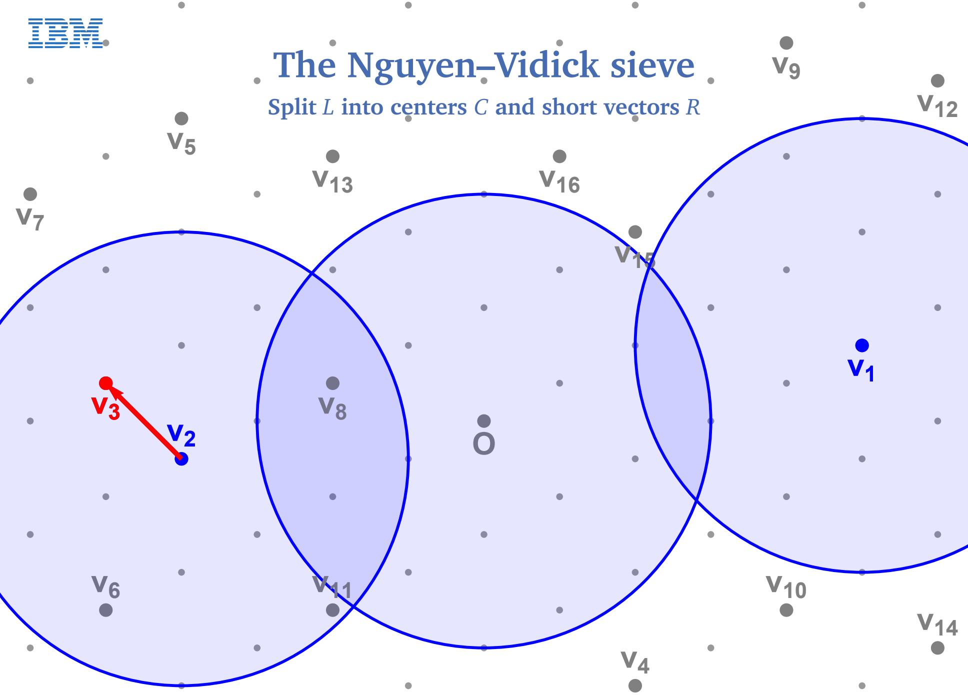



IBM
The Nguyen–Vidick sieve
Split L into centers C and short vectors R
v9
v12
v5
v13
v16
v7
v15
v1
v3
v8
O
v2
v6
v11
v10
v14
v4


The Nguyen–Vidick sieve
Split L into centers C and short vectors R
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
v_{10}
v_{11}
v_{12}
v_{13}
v_{14}
v_{15}
v_{16}
O

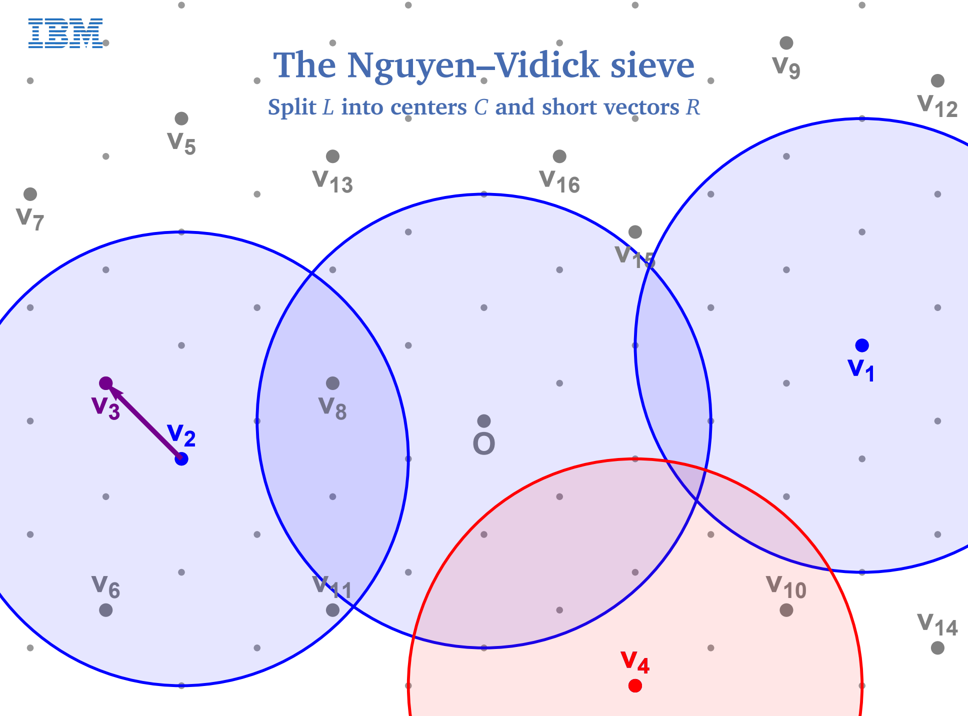
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



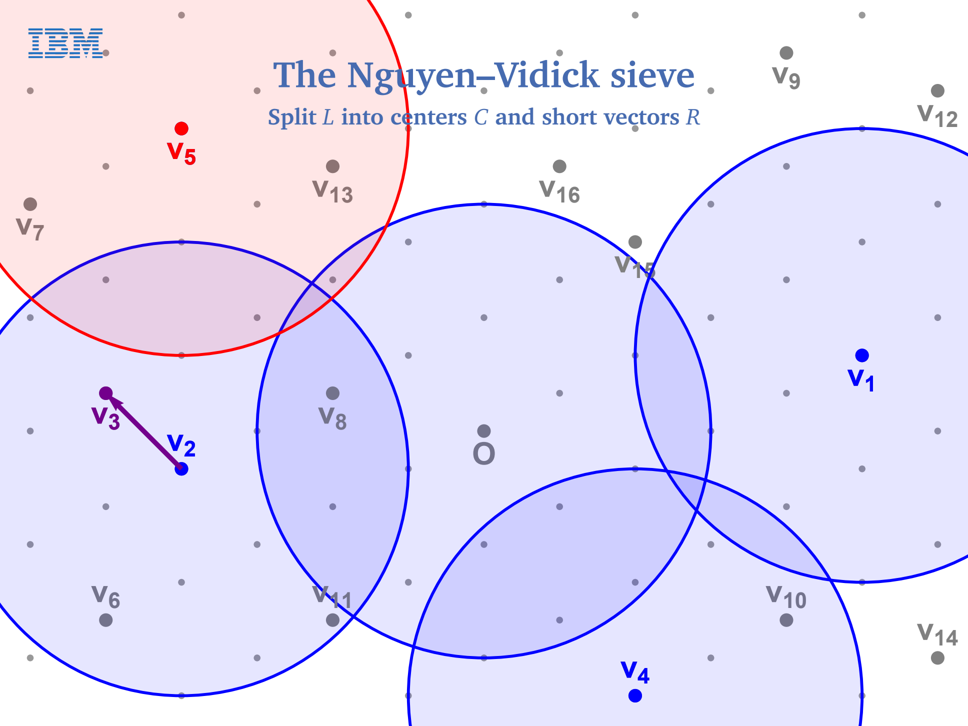
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



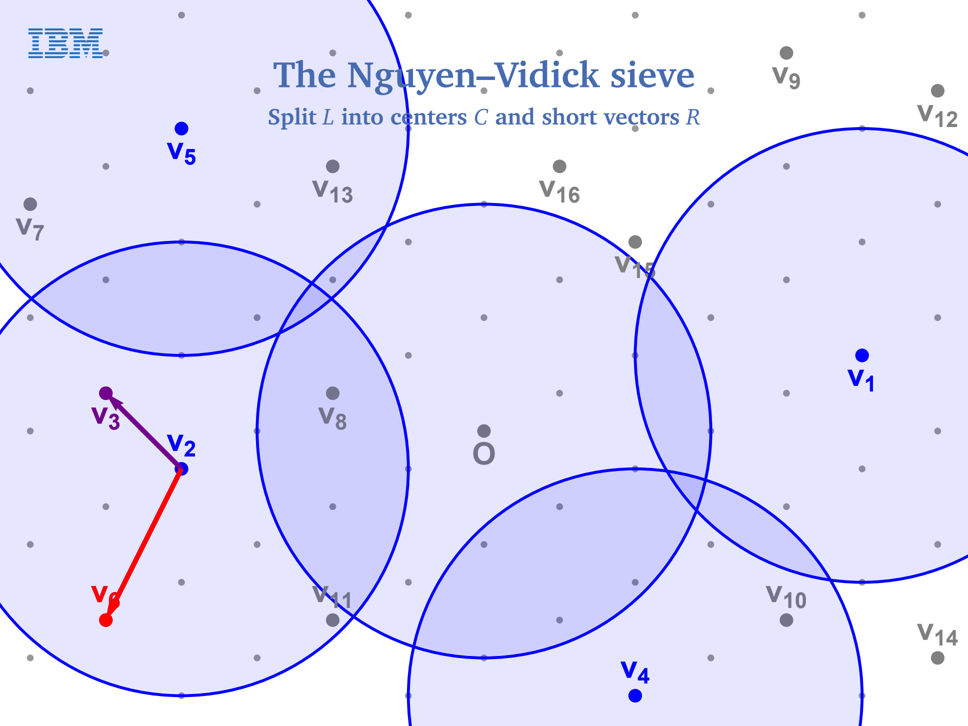
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



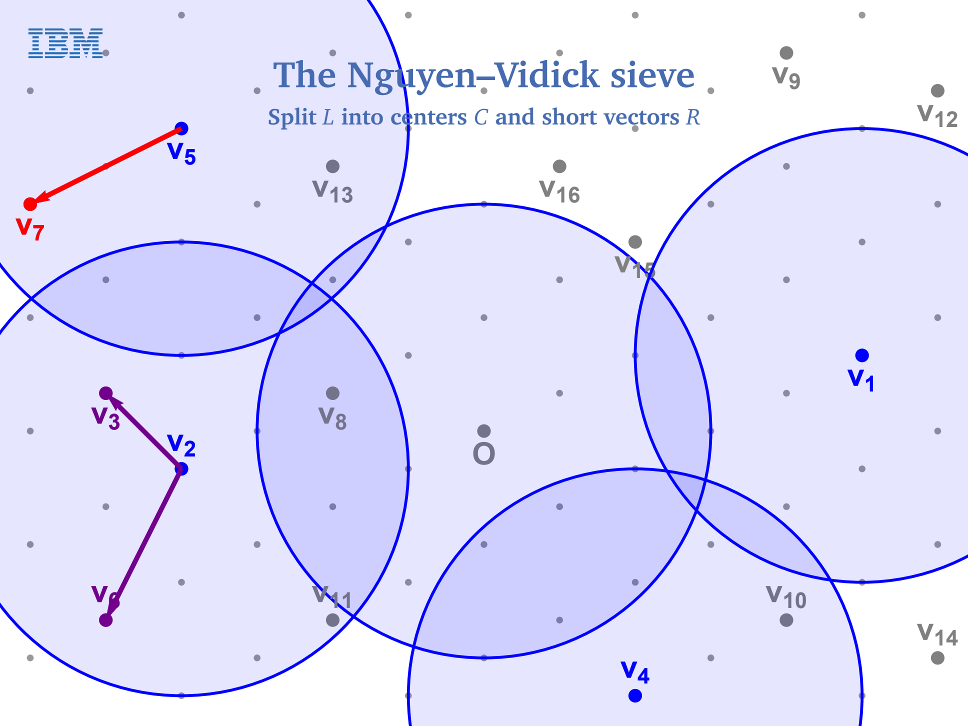
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



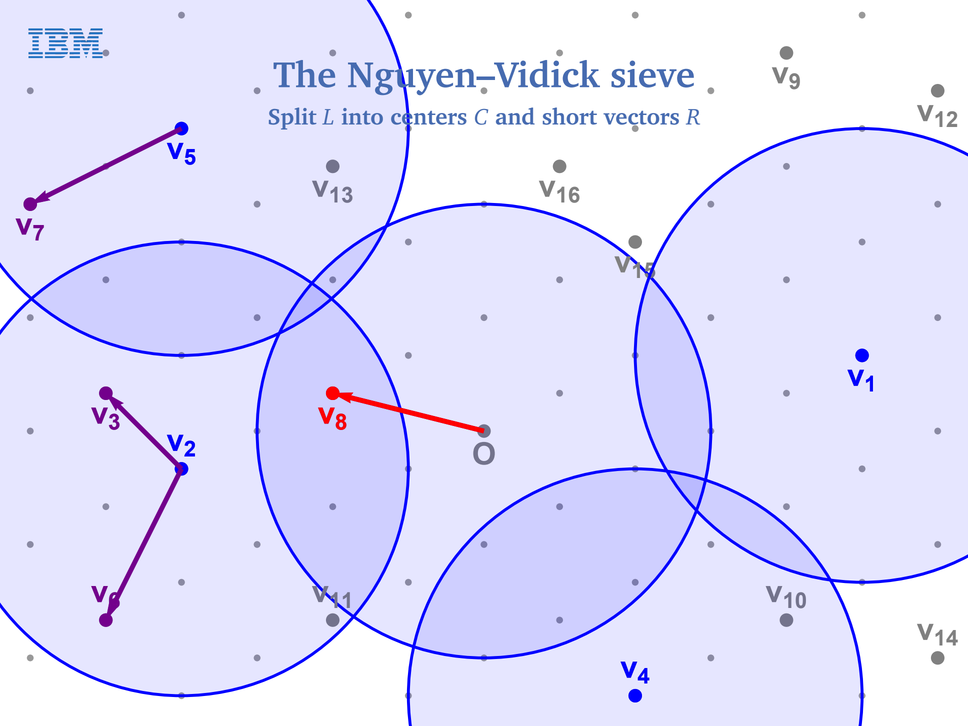
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



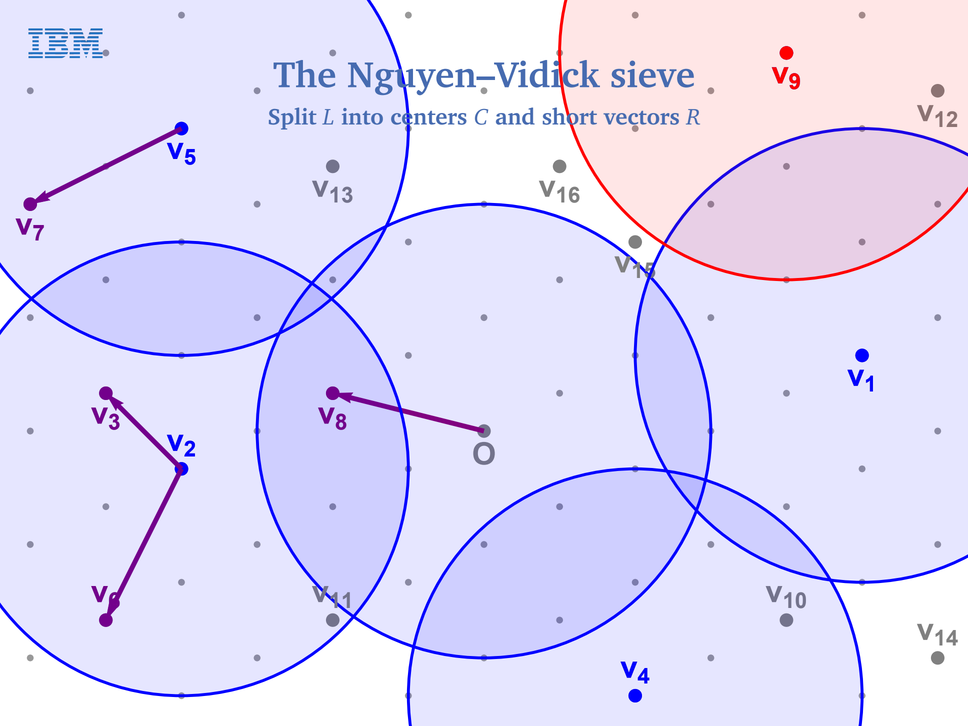
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



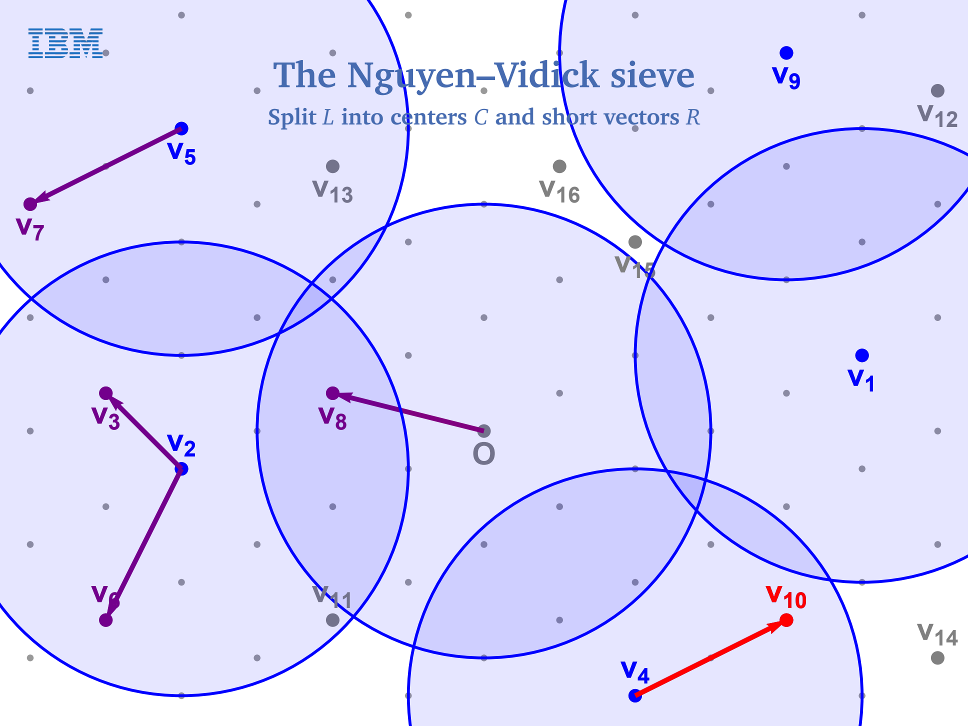
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



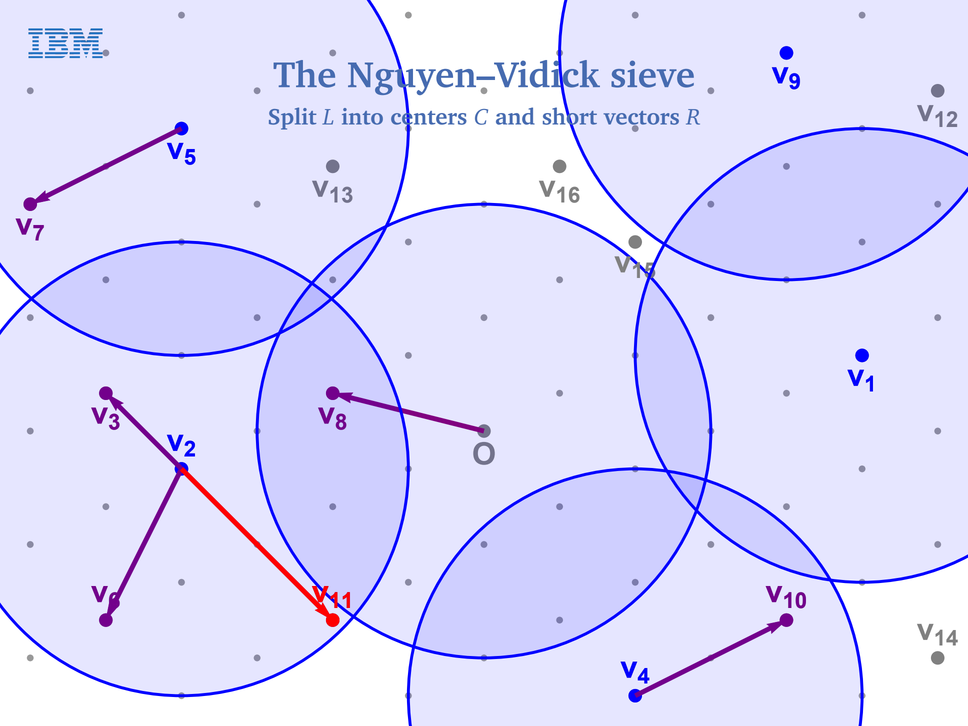
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



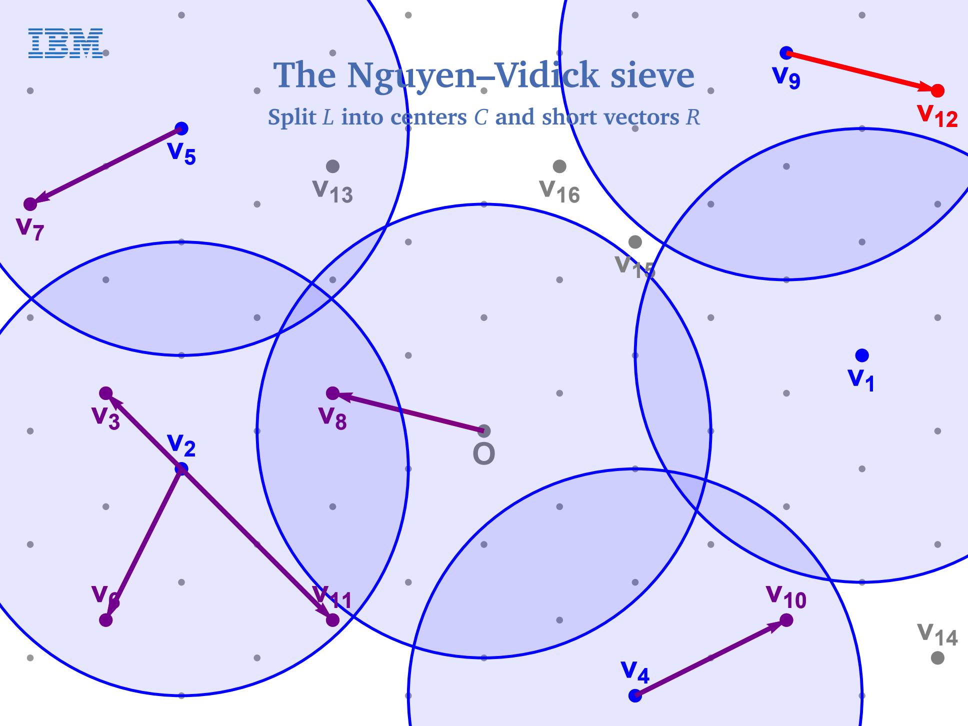
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



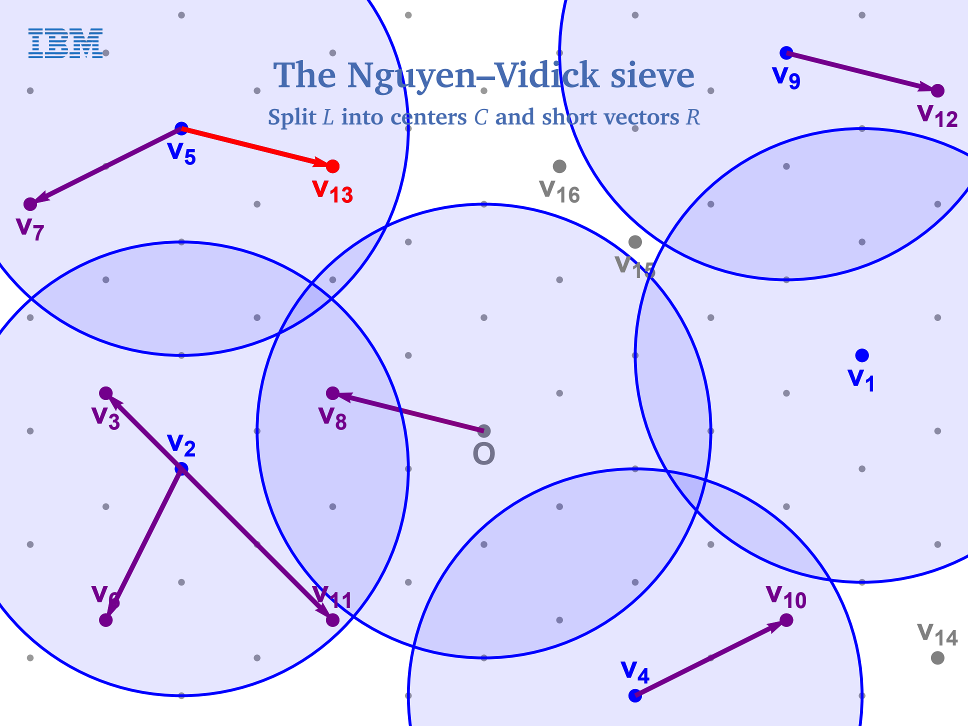
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



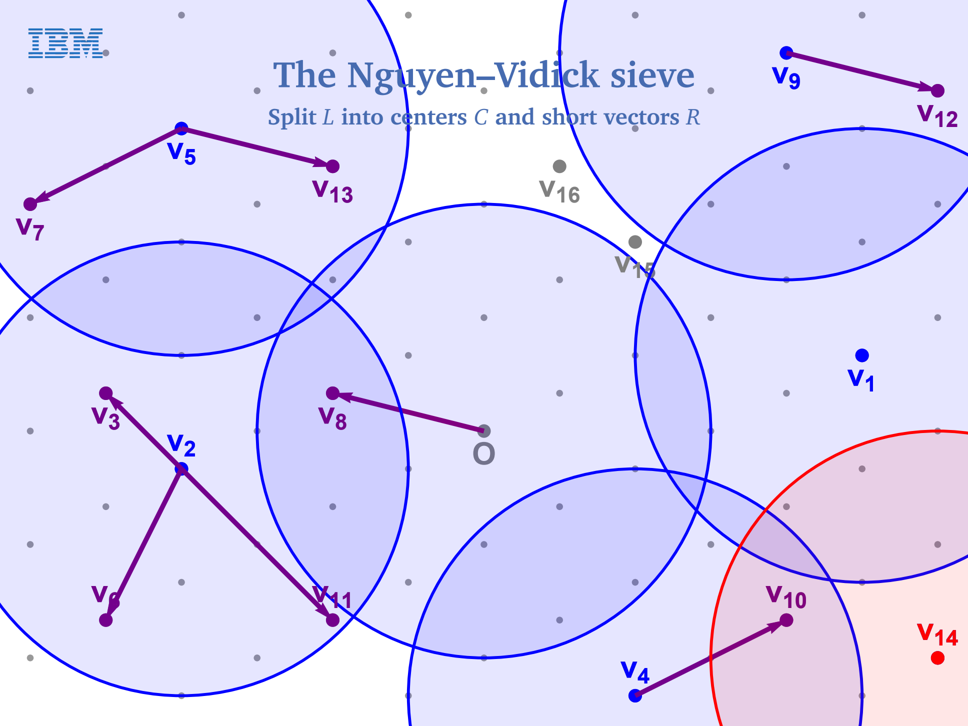
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



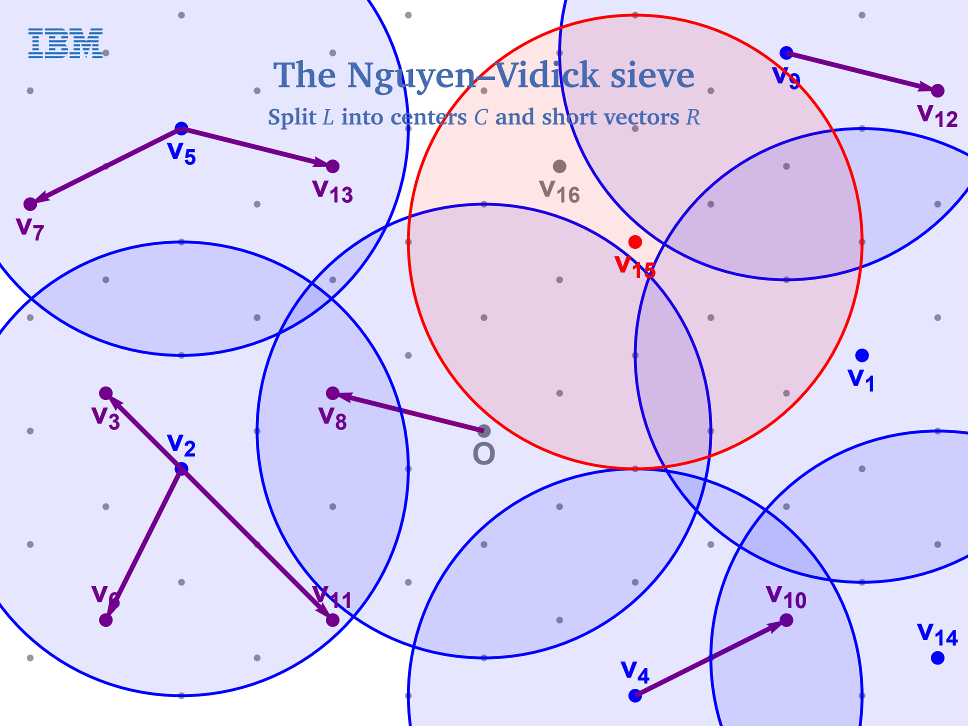
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



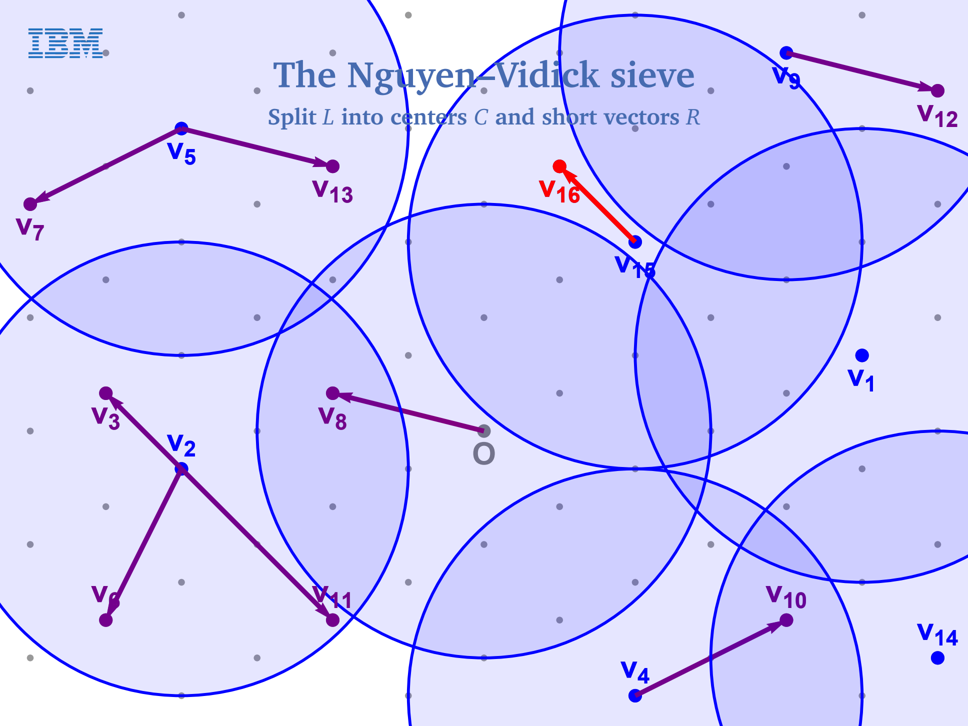
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



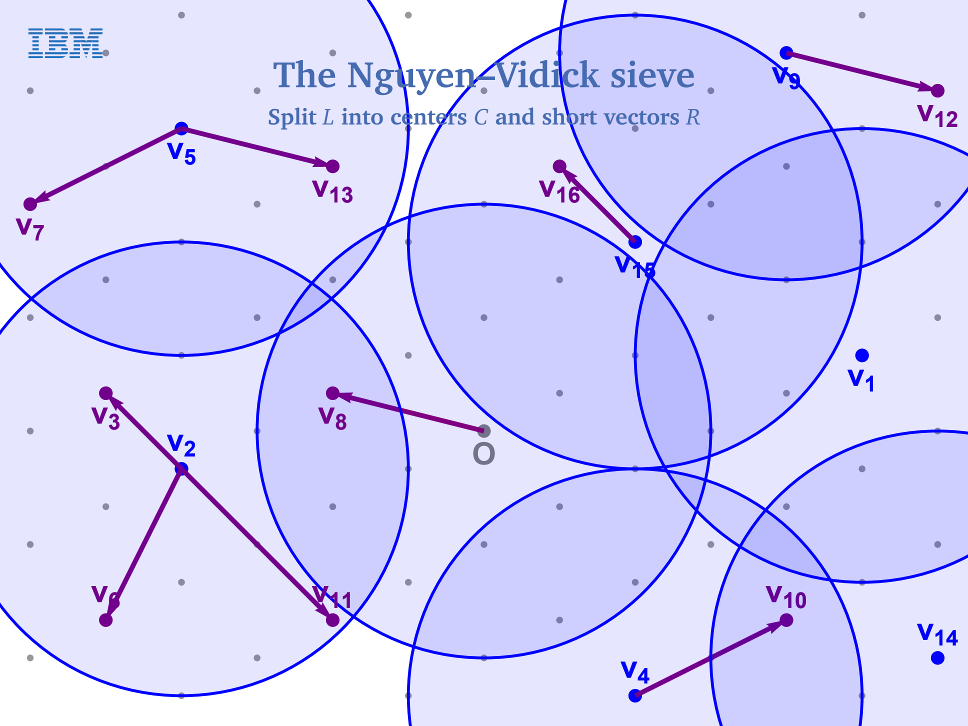
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



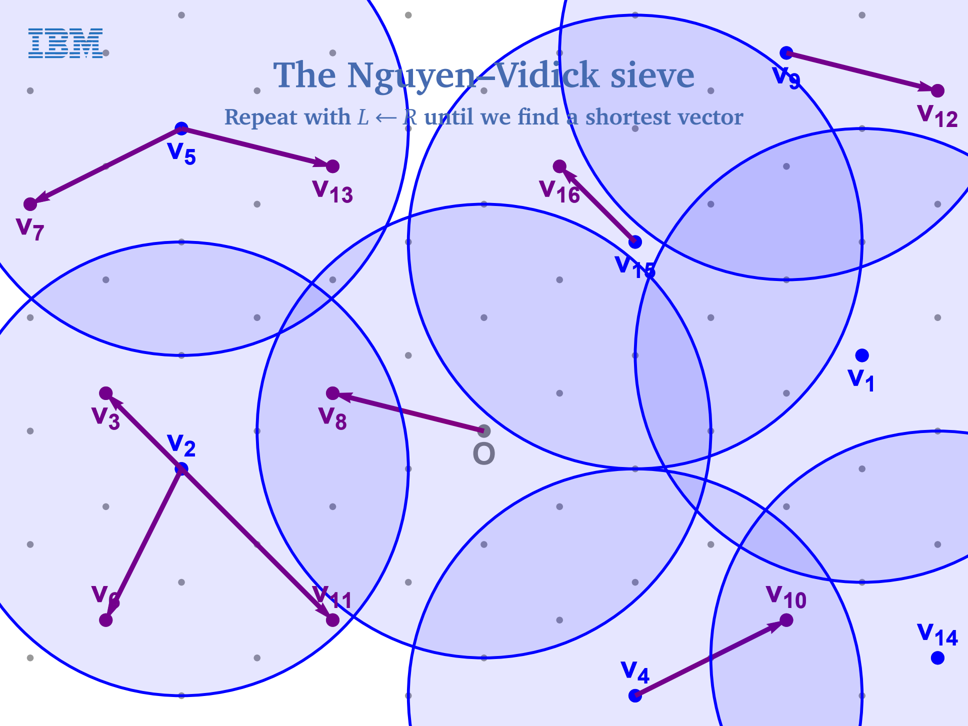
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



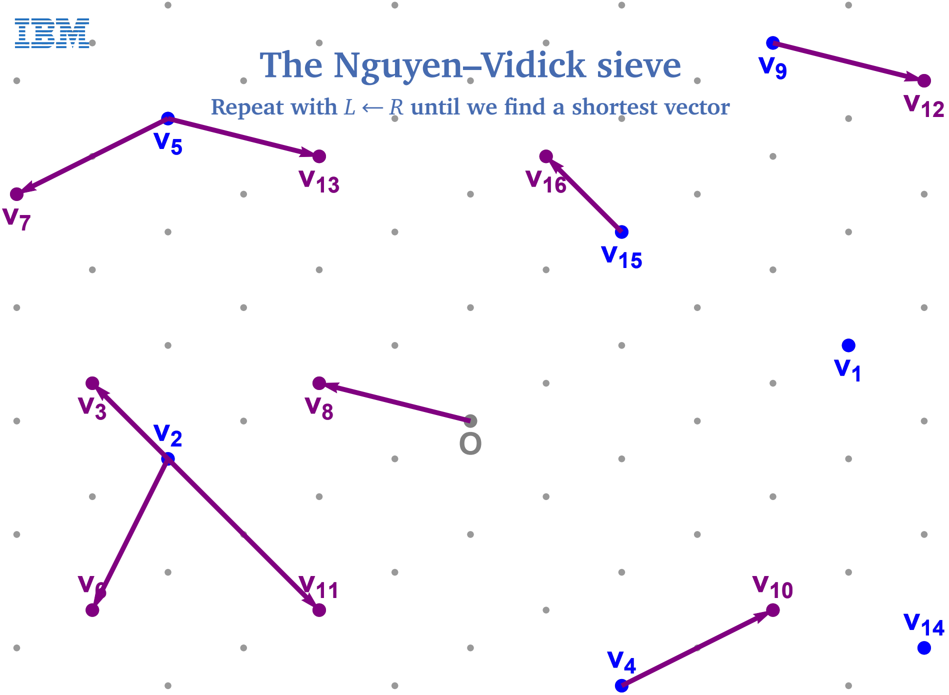
The Nguyen–Vidick sieve

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



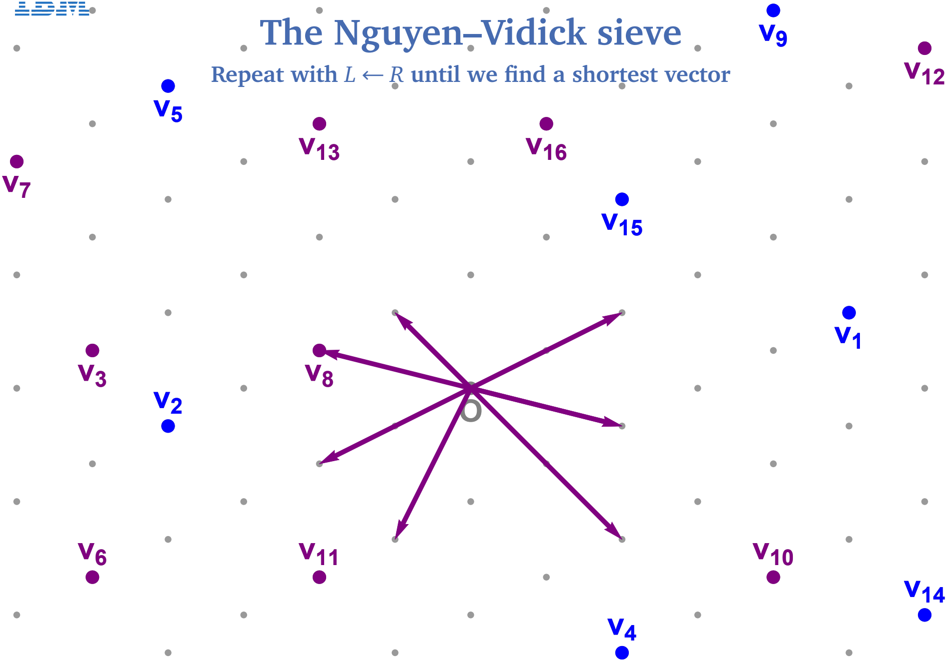
The Nguyen–Vidick sieve

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



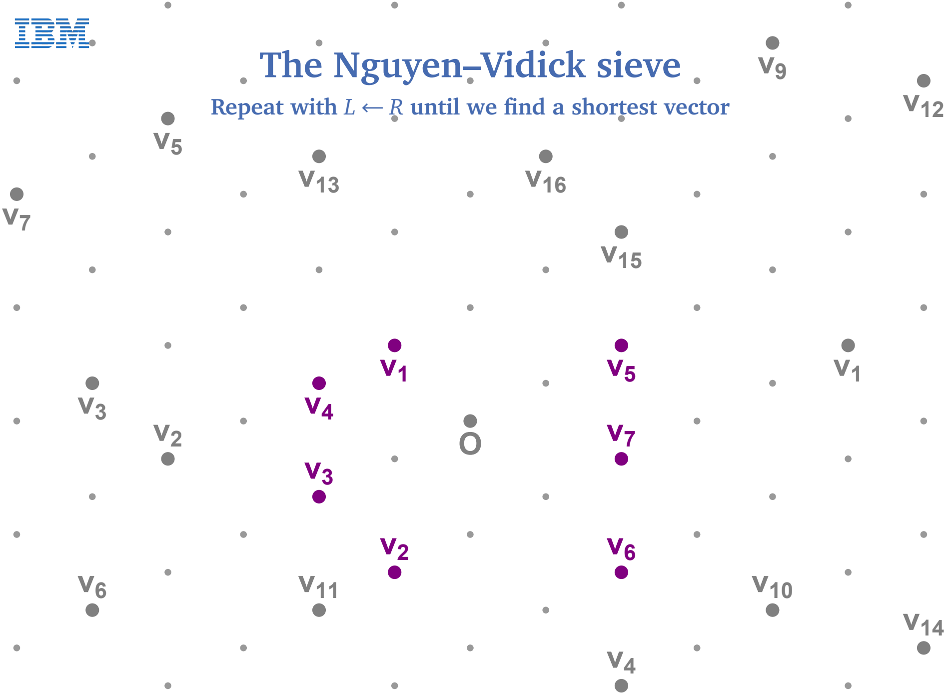
The Nguyen–Vidick sieve

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



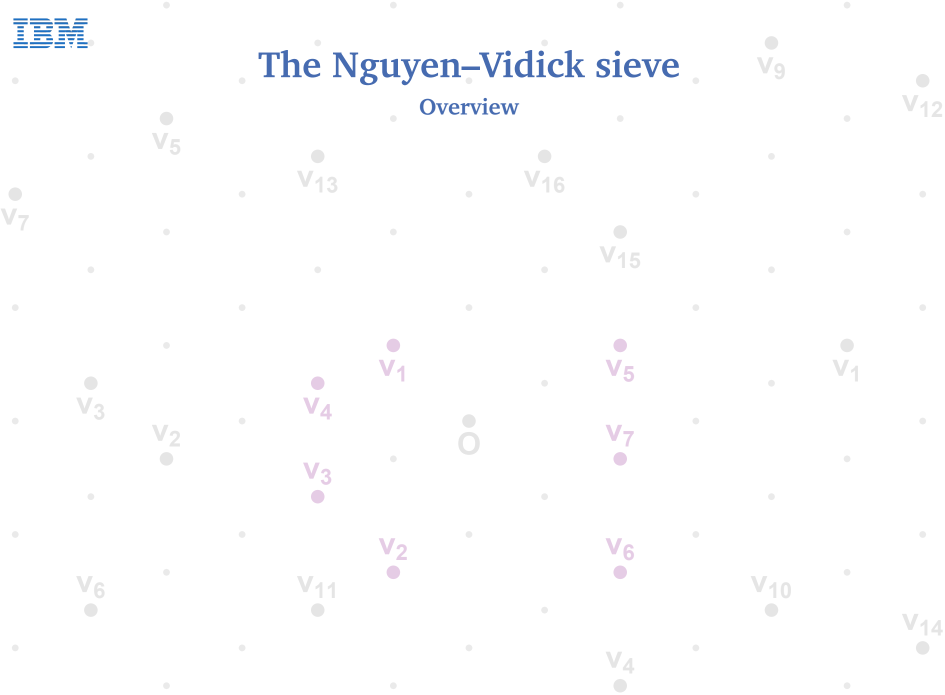
The Nguyen–Vidick sieve

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



The Nguyen–Vidick sieve

Overview



The 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 $\mathbb{R}^n$

The 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: $\mathrm{poly}(n)$

The Nguyen–Vidick sieve

Overview

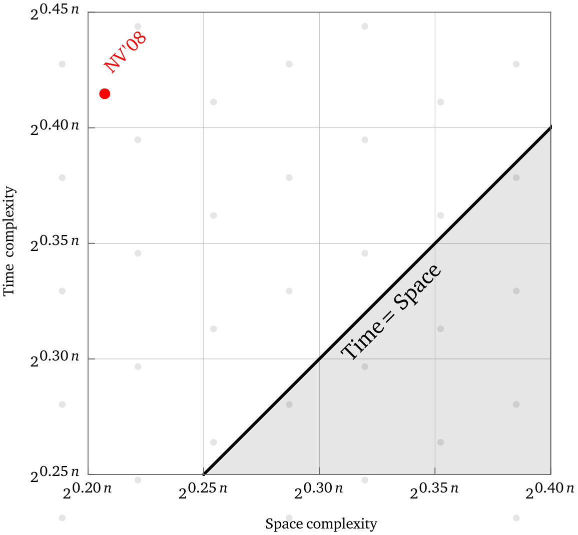
- Space complexity: $\sqrt{4/3}^n \approx 2^{0.21n+o(n)}$ vectors
  - Need $\sqrt{4/3}^n$ vectors to cover all corners of $\mathbb{R}^n$
- Time complexity: $(4/3)^n \approx 2^{0.42n+o(n)}$
  - Comparing a target vector to all centers: $2^{0.21n+o(n)}$
  - Repeating this for each list vector: $2^{0.21n+o(n)}$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

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

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

The Nguyen–Vidick sieve

Space/time trade-off

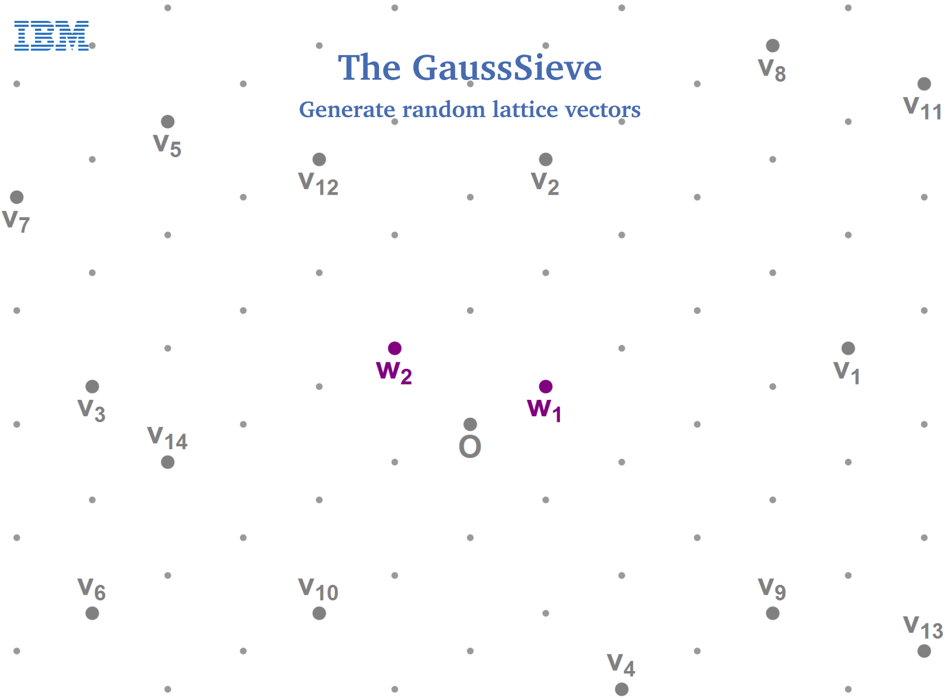


The GaussSieve

Generate random lattice vectors

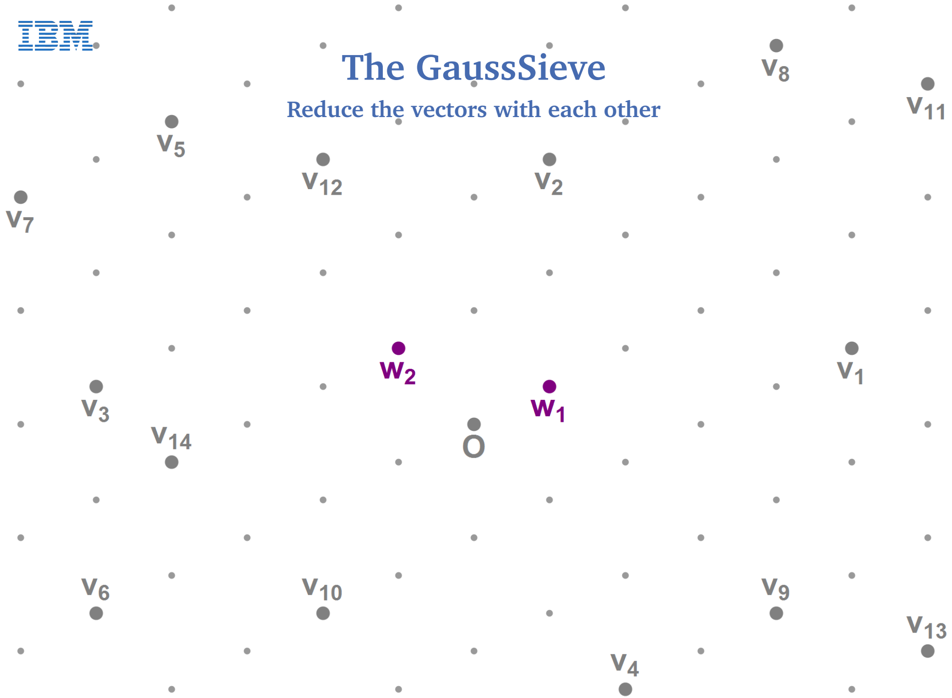
The GaussSieve

Generate random lattice vectors



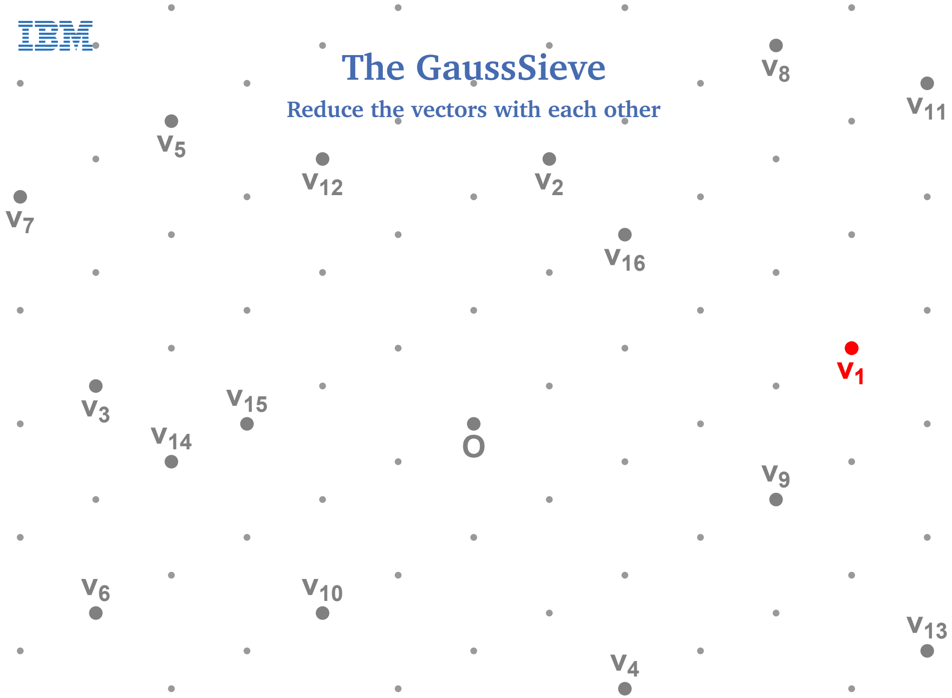
The GaussSieve

Reduce the vectors with each other



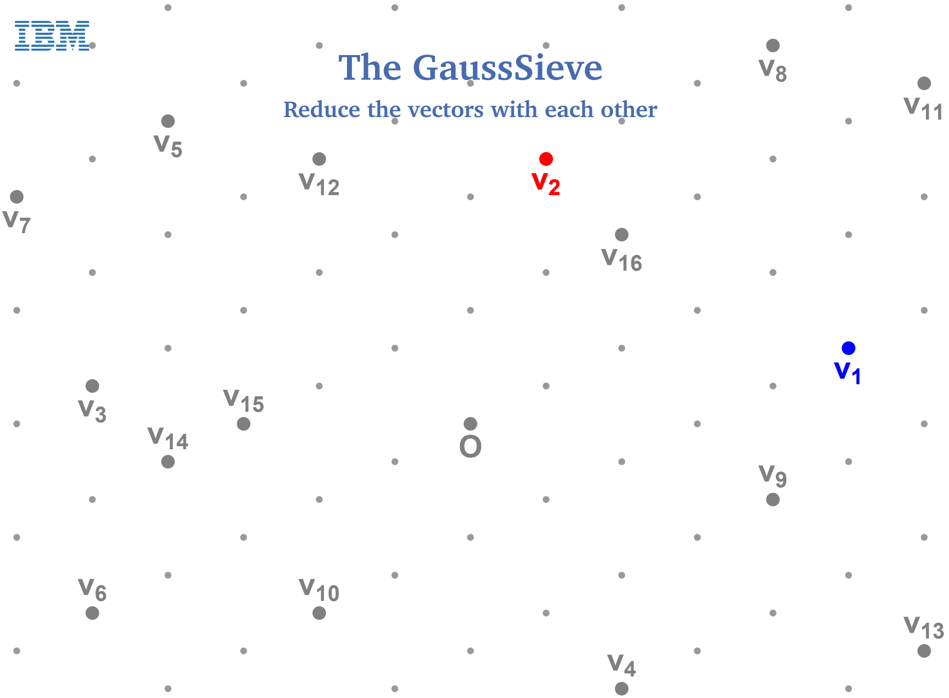
The GaussSieve

Reduce the vectors with each other



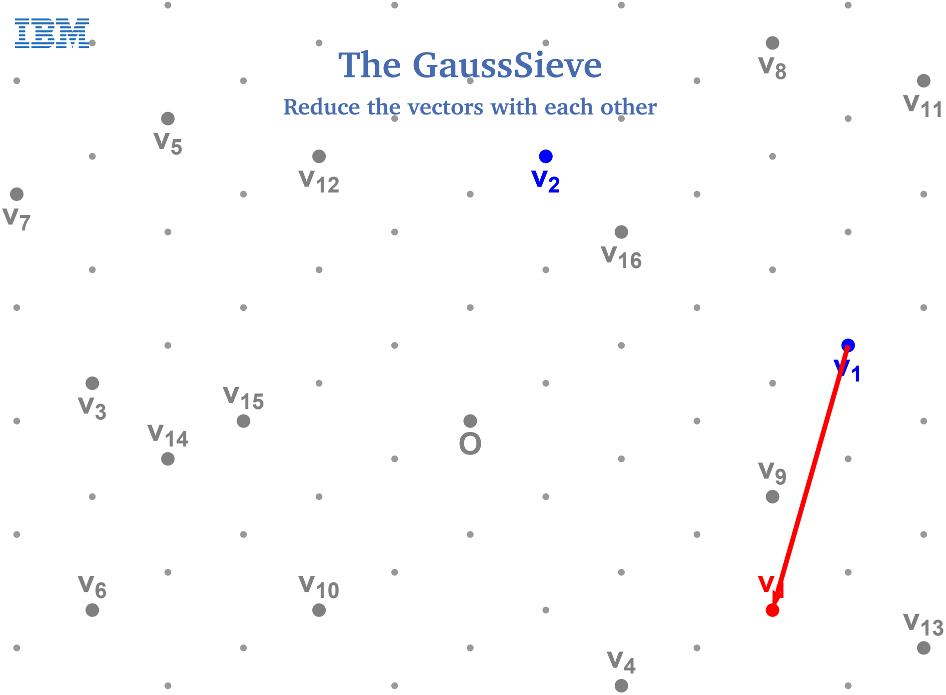
The GaussSieve

Reduce the vectors with each other



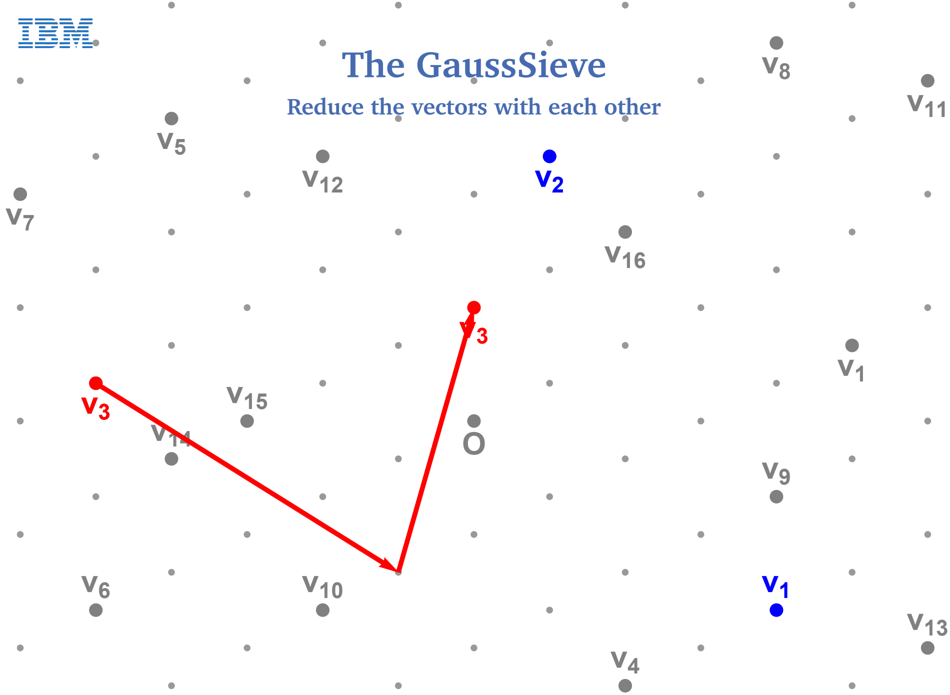
The GaussSieve

Reduce the vectors with each other



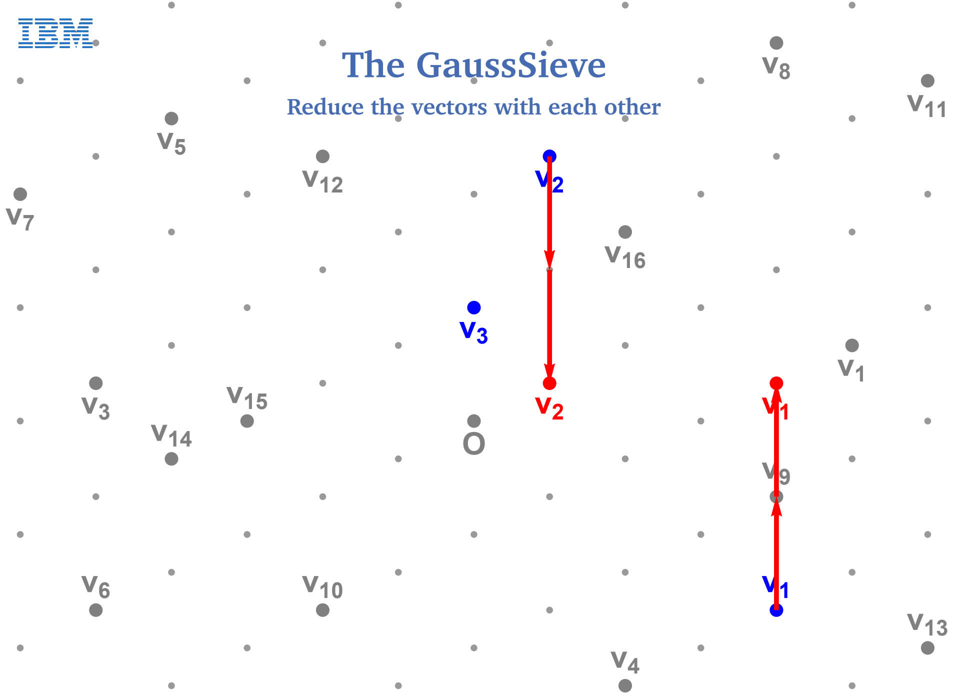
The GaussSieve

Reduce the vectors with each other



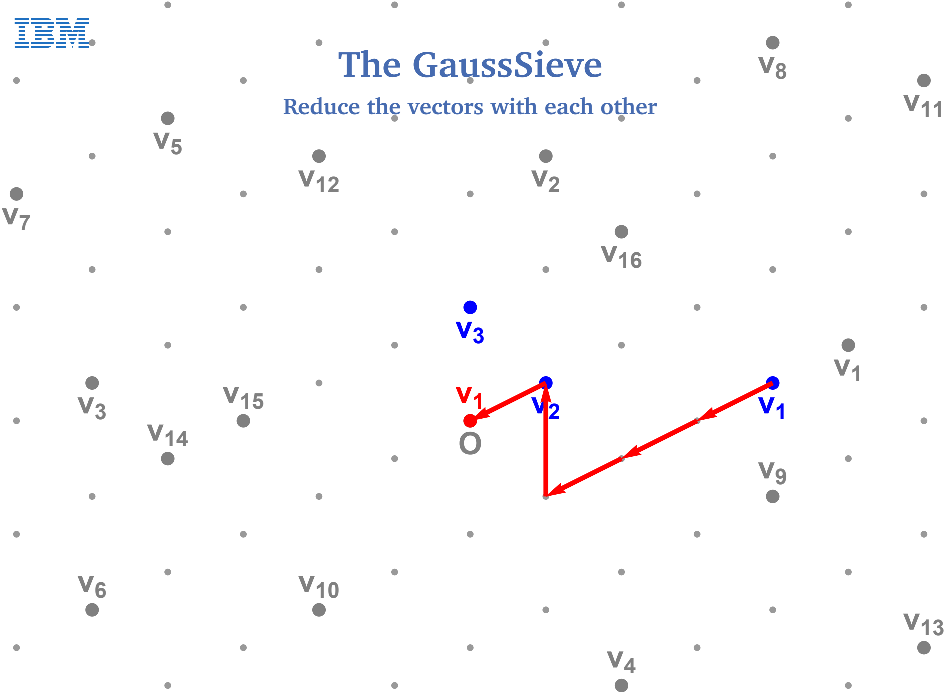
The GaussSieve

Reduce the vectors with each other



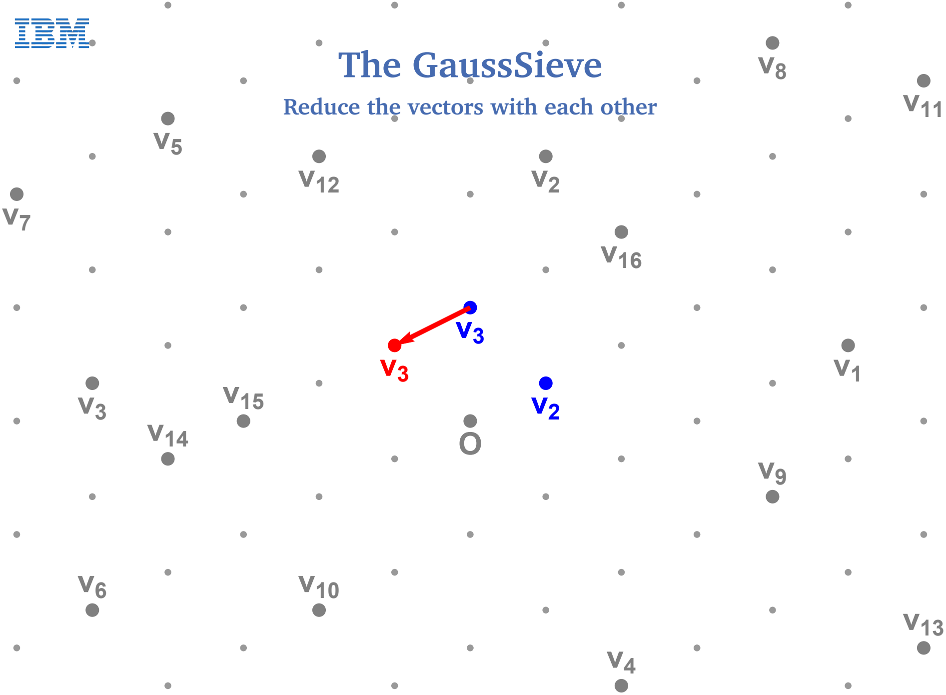
The GaussSieve

Reduce the vectors with each other



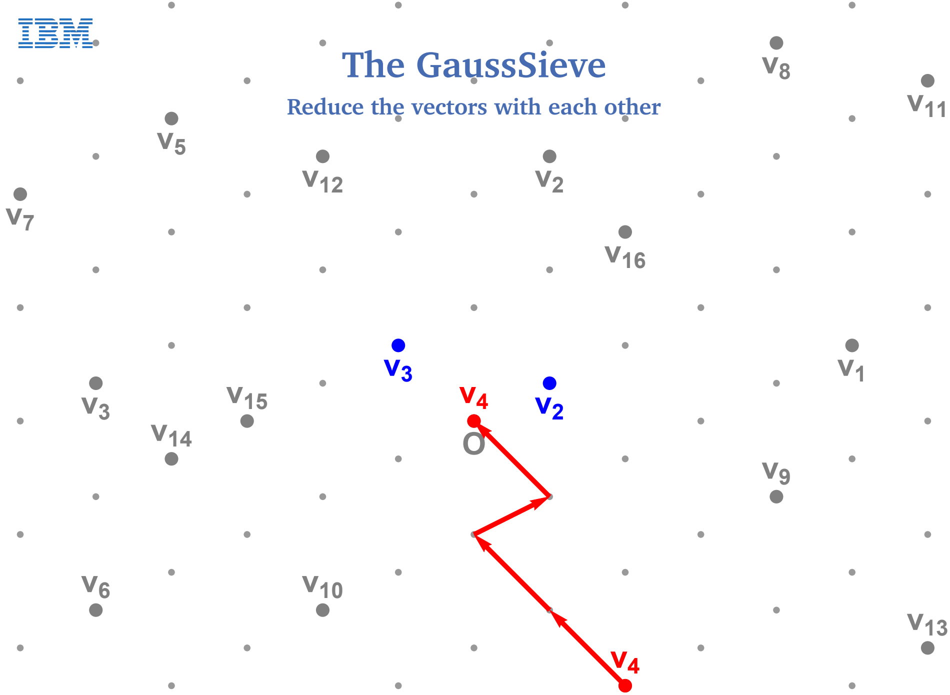
The GaussSieve

Reduce the vectors with each other



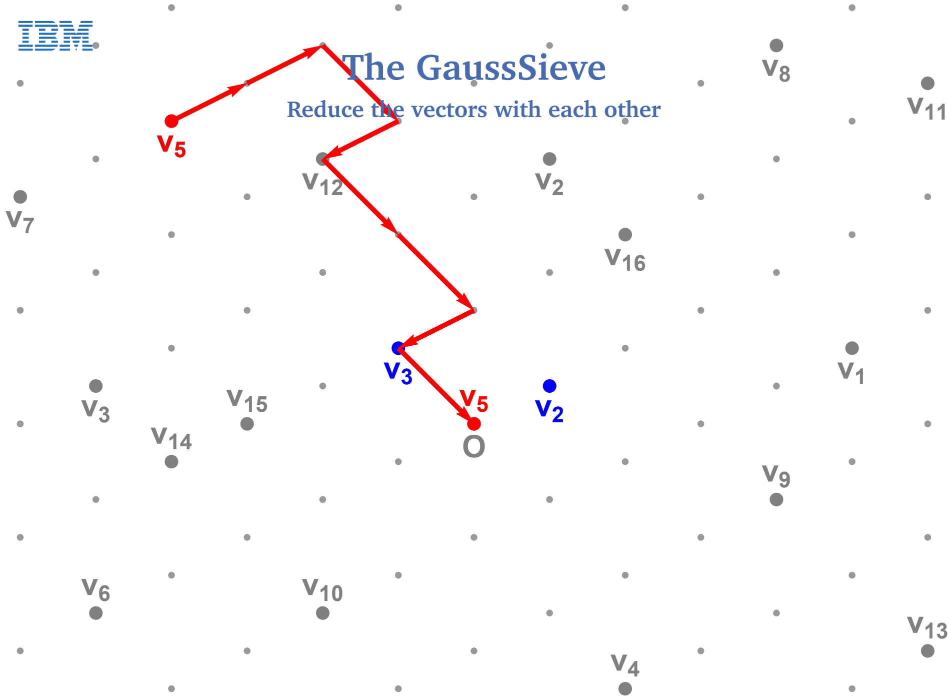
The GaussSieve

Reduce the vectors with each other



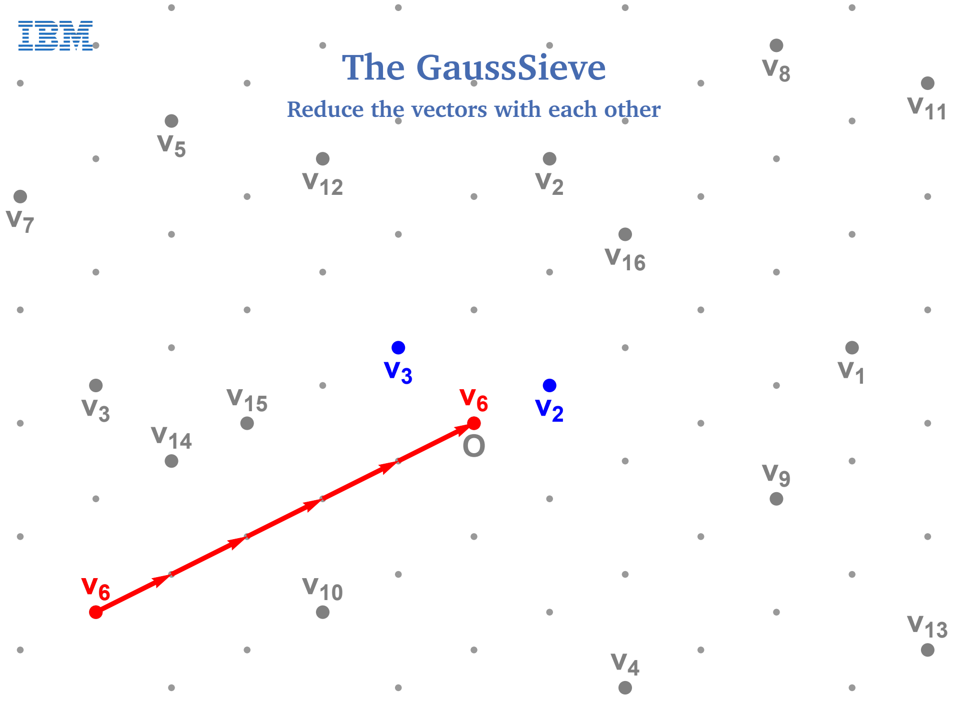
The GaussSieve

Reduce the vectors with each other



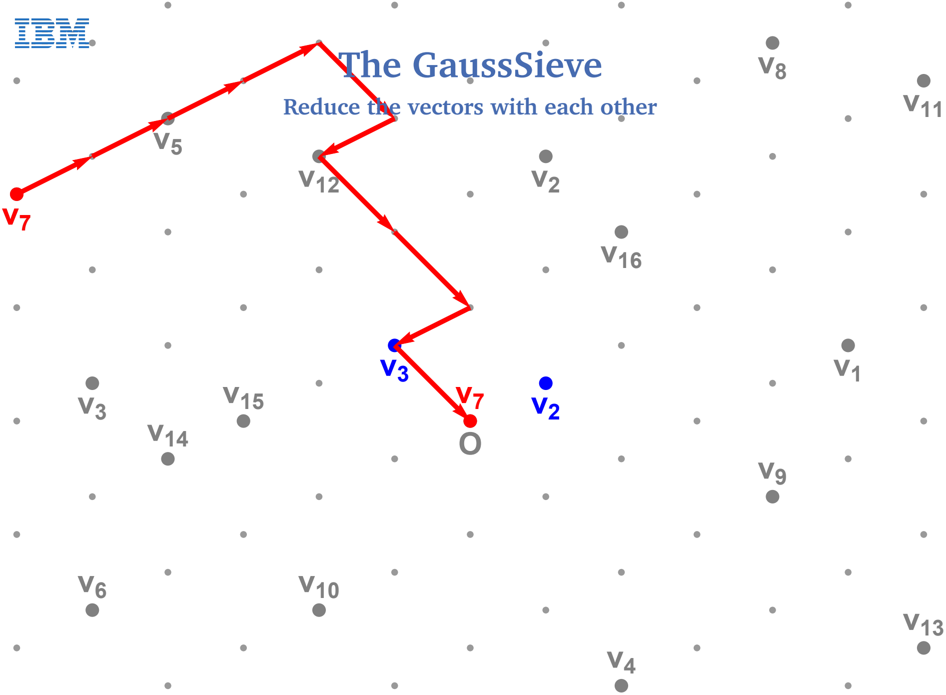
The GaussSieve

Reduce the vectors with each other



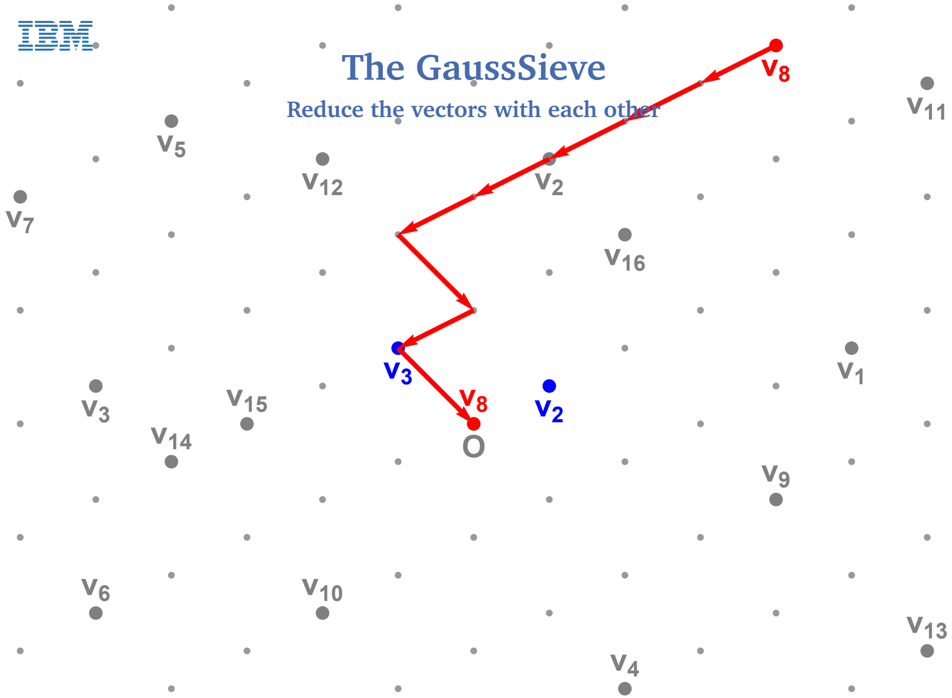
The GaussSieve

Reduce the vectors with each other



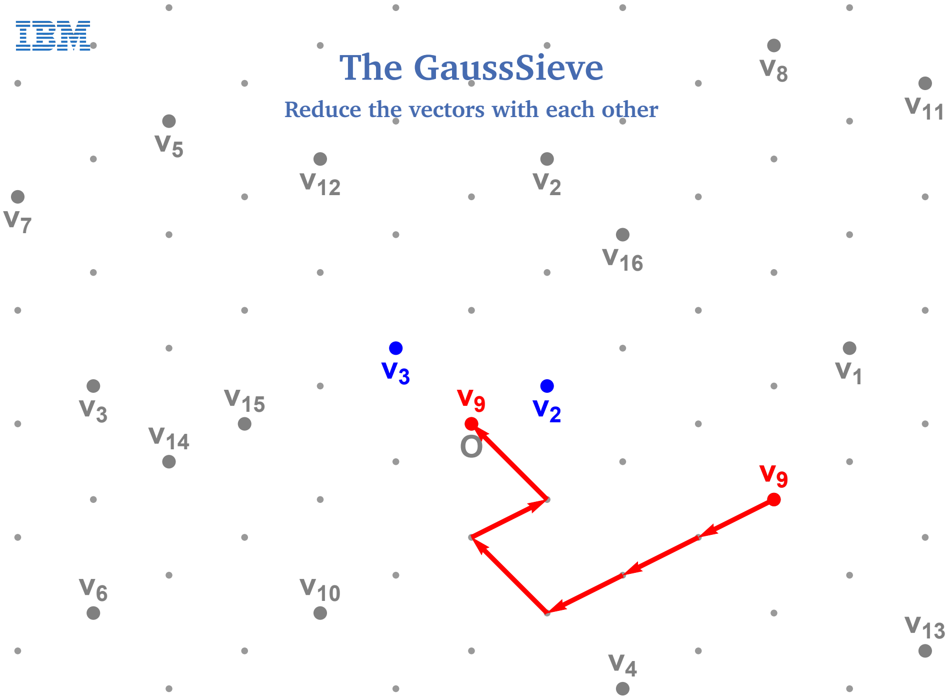
The GaussSieve

Reduce the vectors with each other



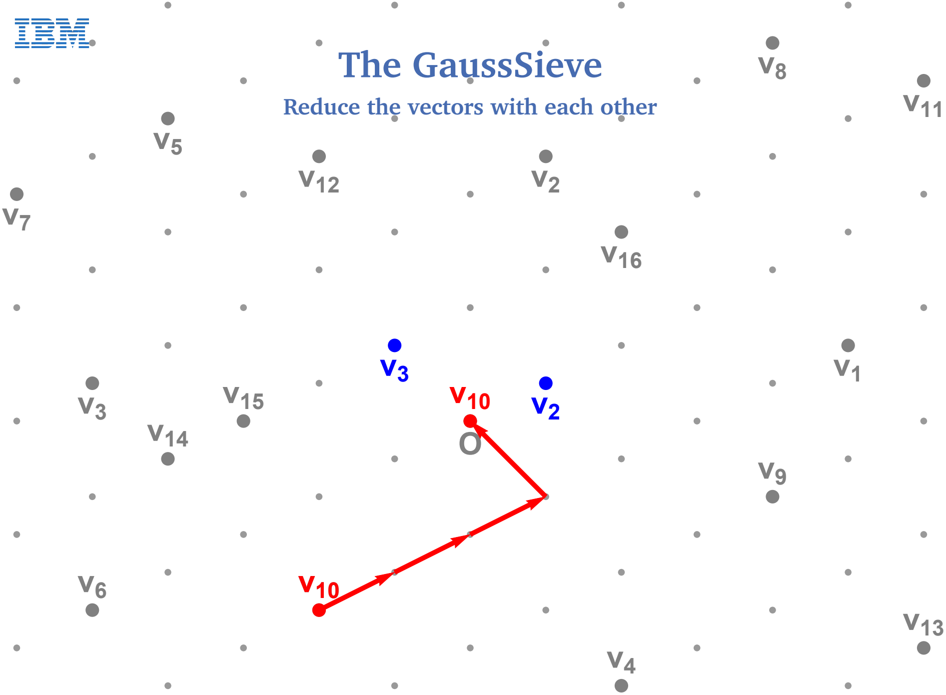
The GaussSieve

Reduce the vectors with each other



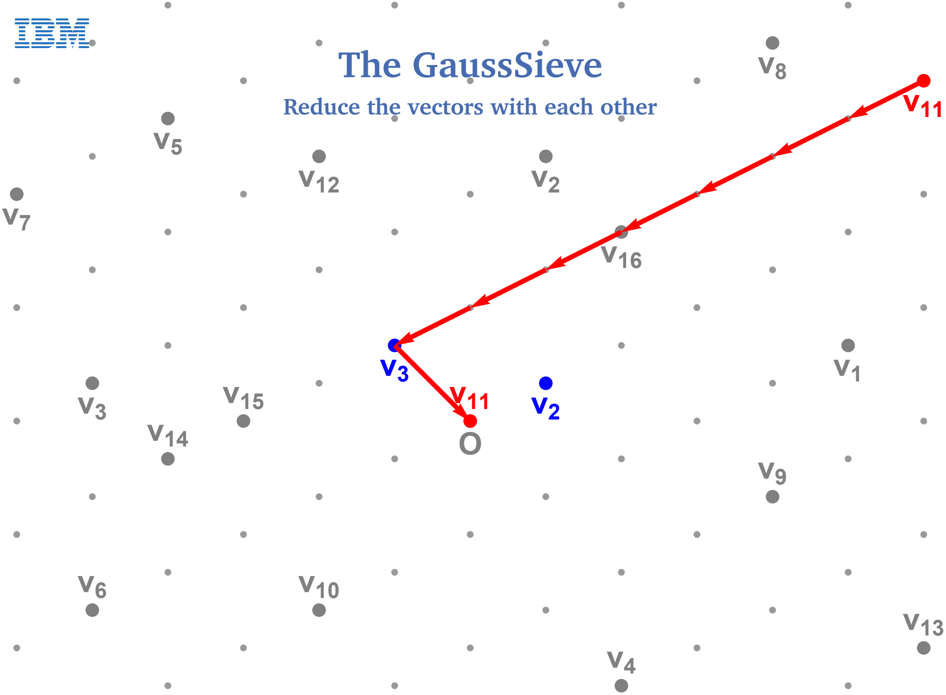
The GaussSieve

Reduce the vectors with each other



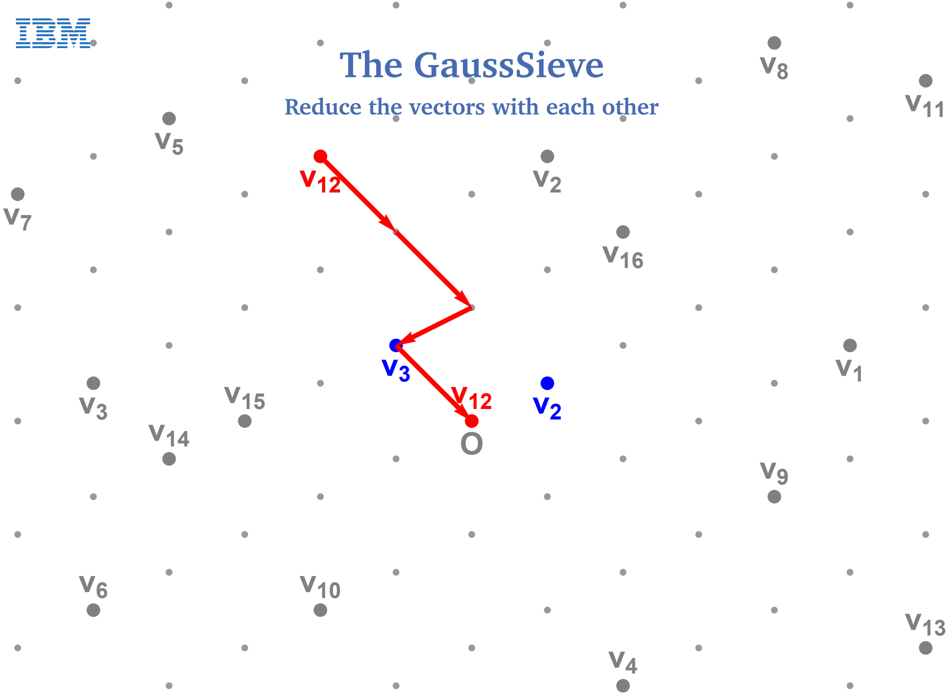
The GaussSieve

Reduce the vectors with each other



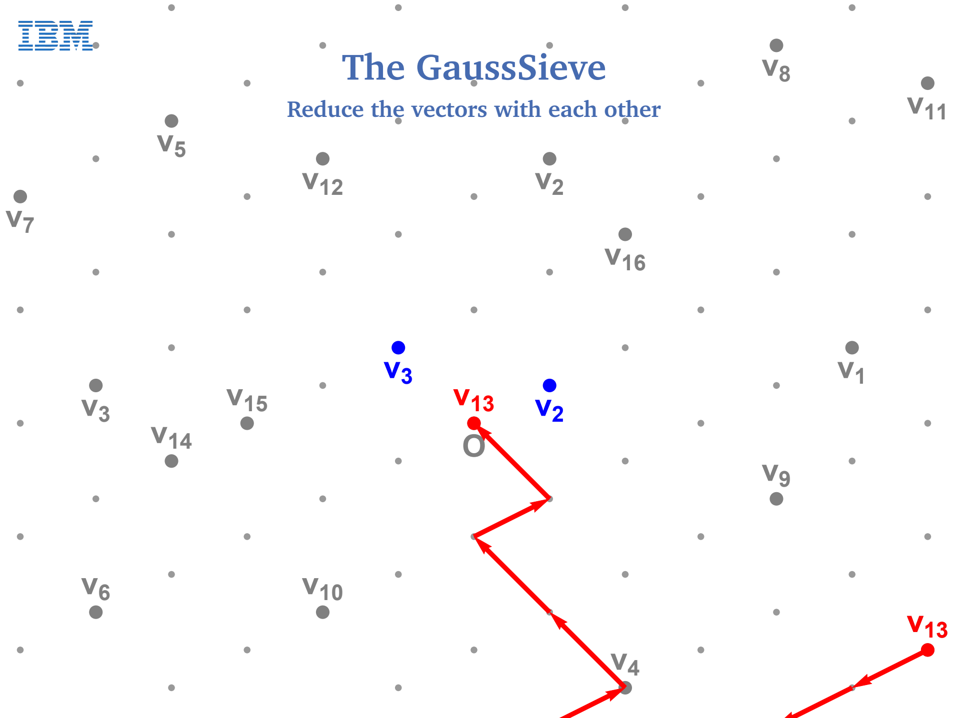
The GaussSieve

Reduce the vectors with each other



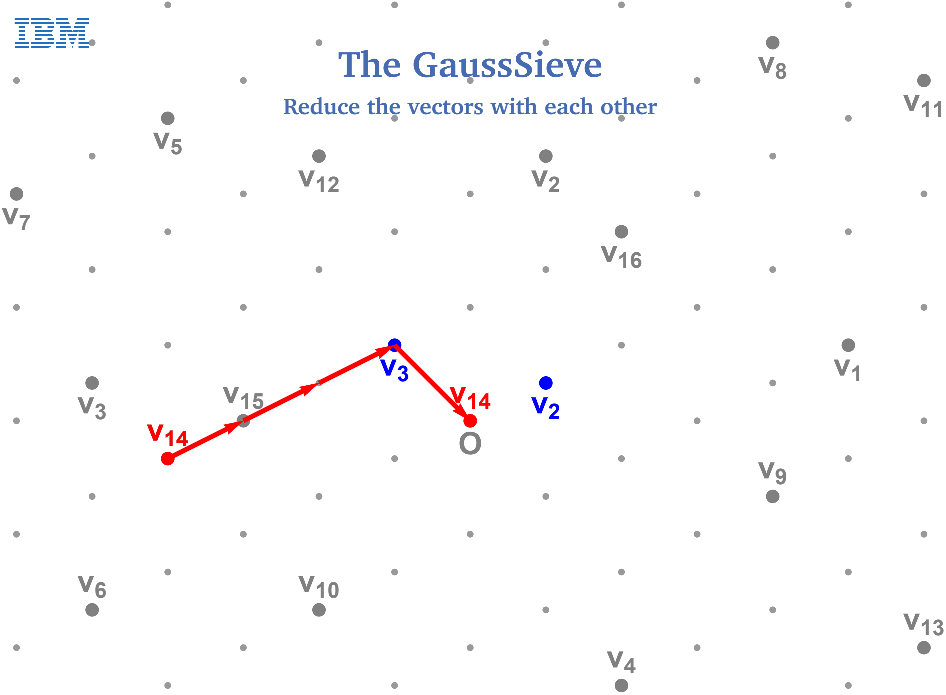
The GaussSieve

Reduce the vectors with each other



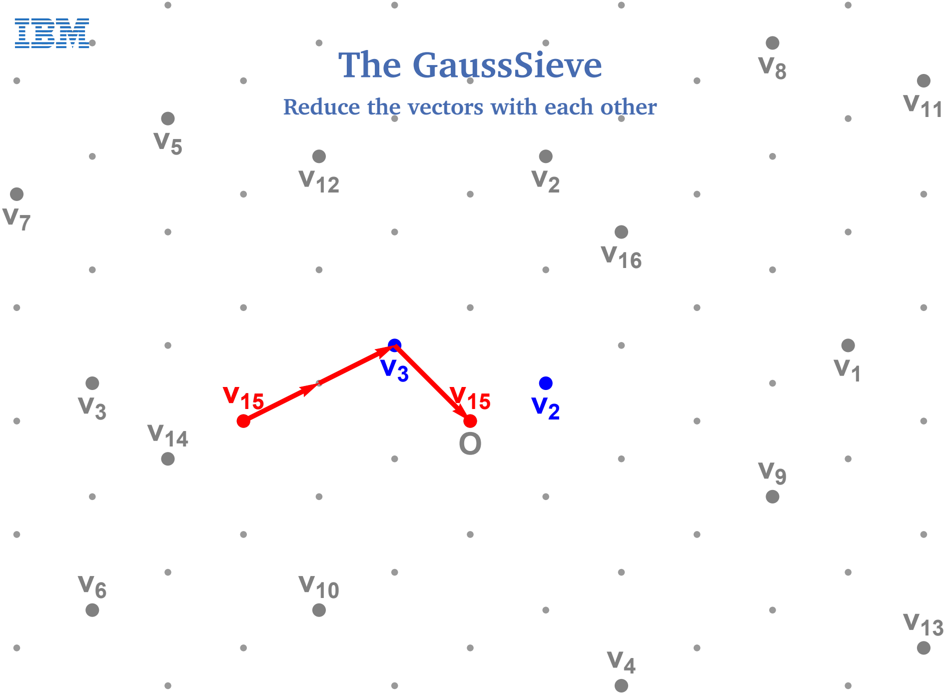
The GaussSieve

Reduce the vectors with each other



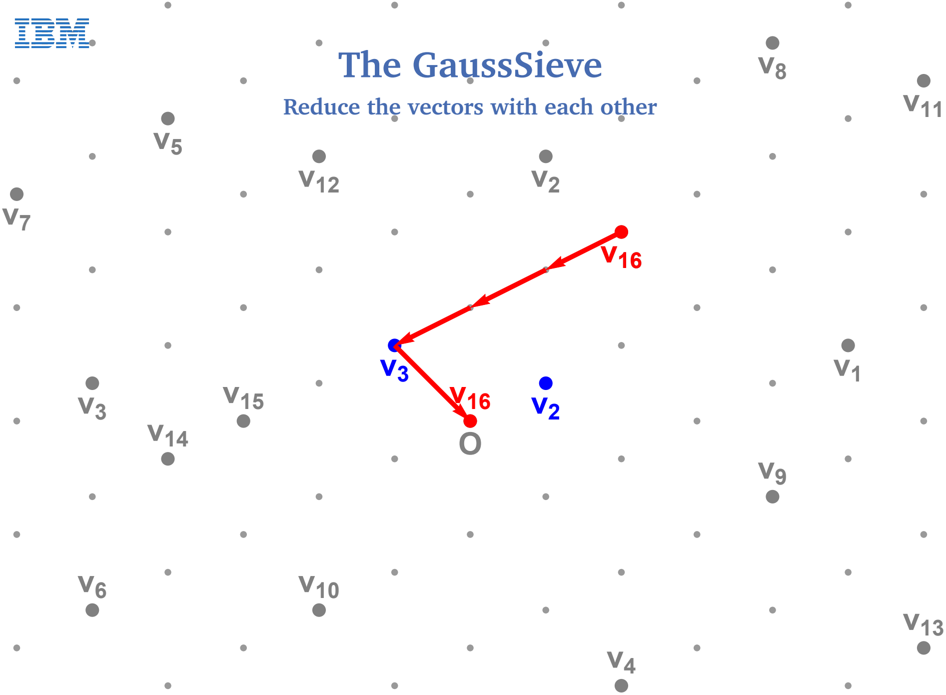
The GaussSieve

Reduce the vectors with each other



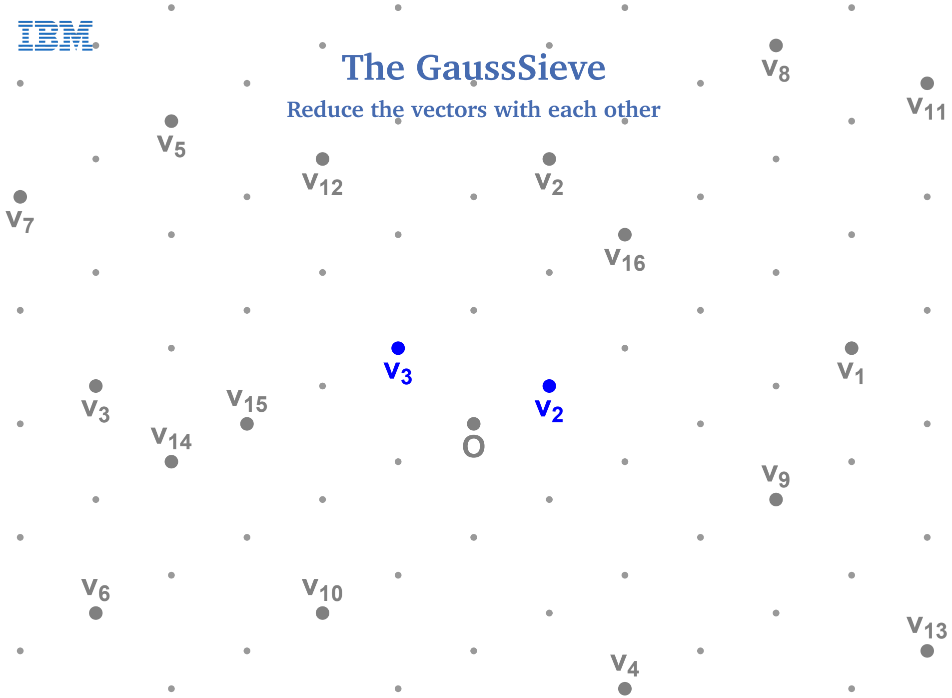
The GaussSieve

Reduce the vectors with each other



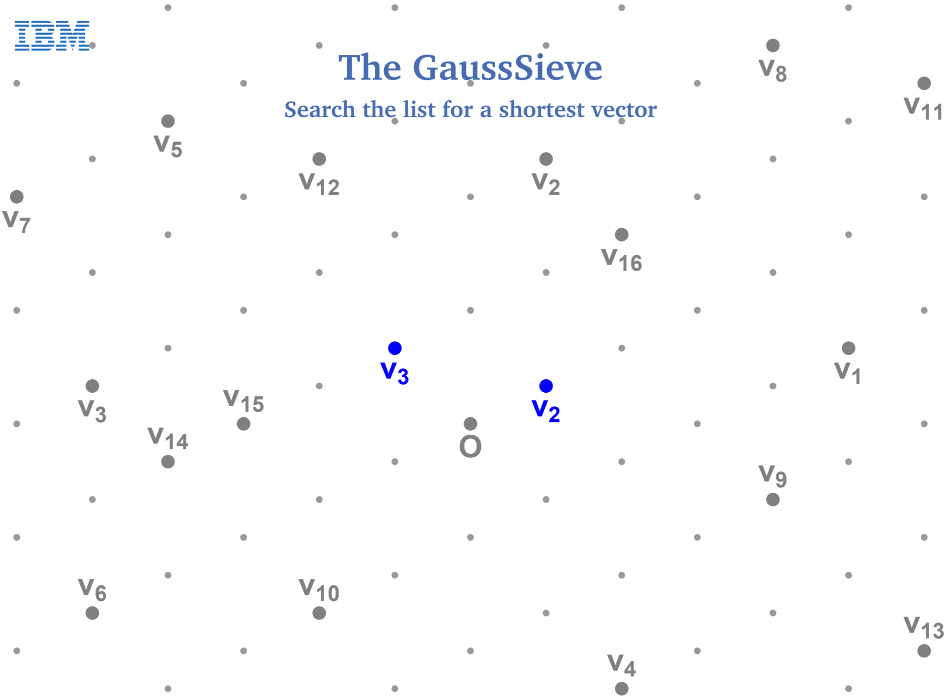
The GaussSieve

Reduce the vectors with each other



The GaussSieve

Search the list for a shortest vector

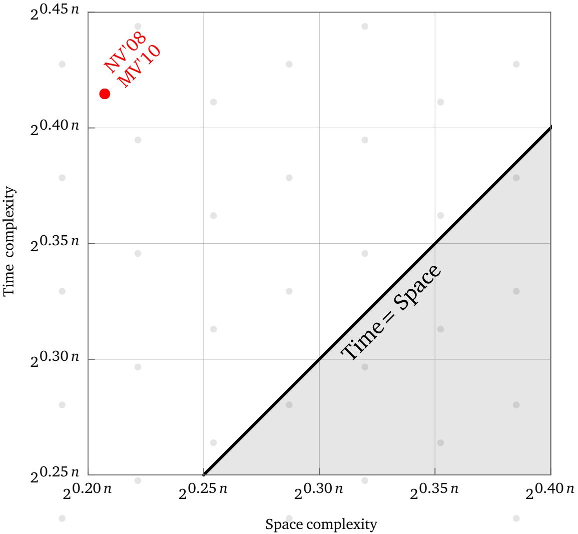


The GaussSieve

Search the list for a shortest vector

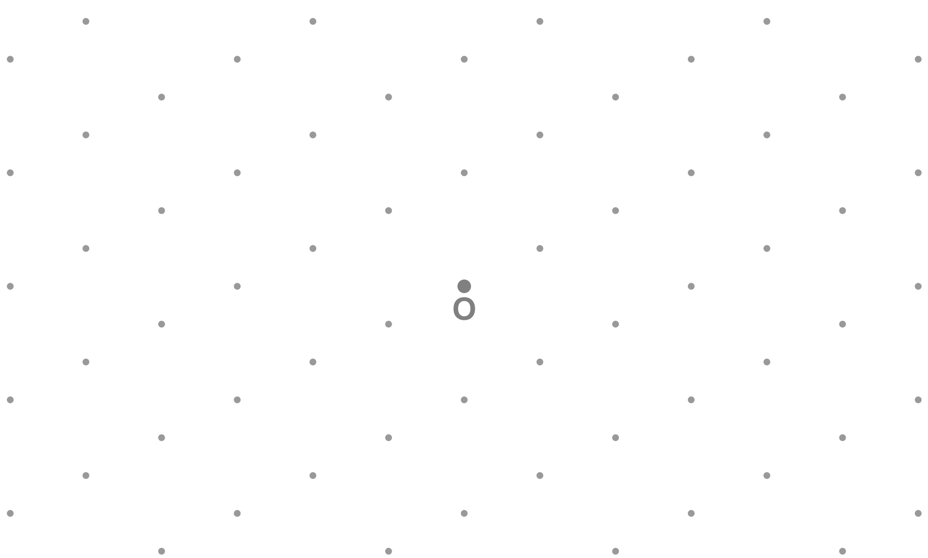
The GaussSieve

Space/time trade-off



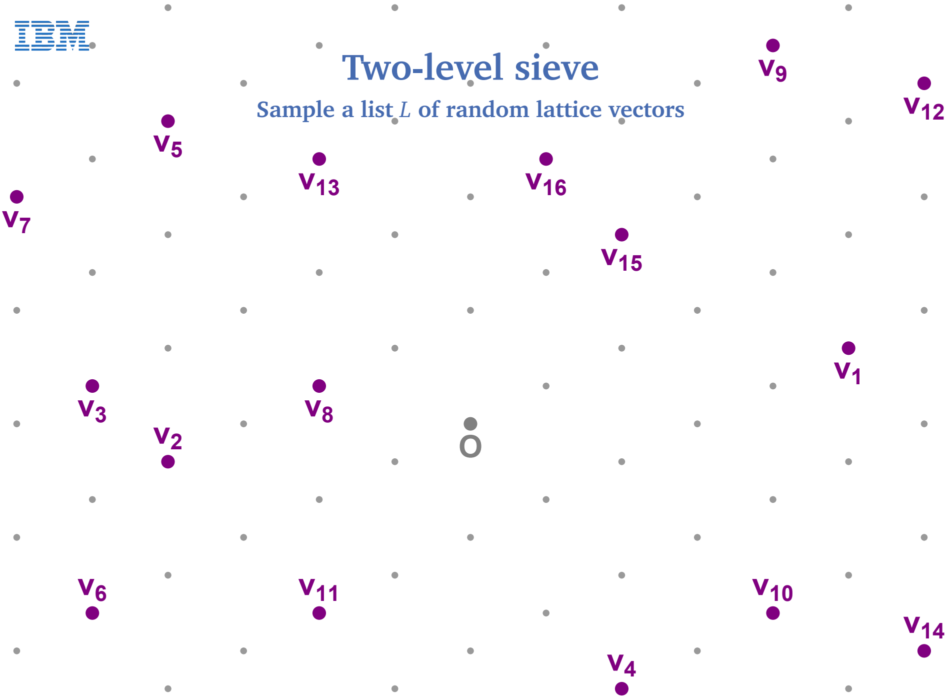
Two-level sieve

Sample a list L of random lattice vectors



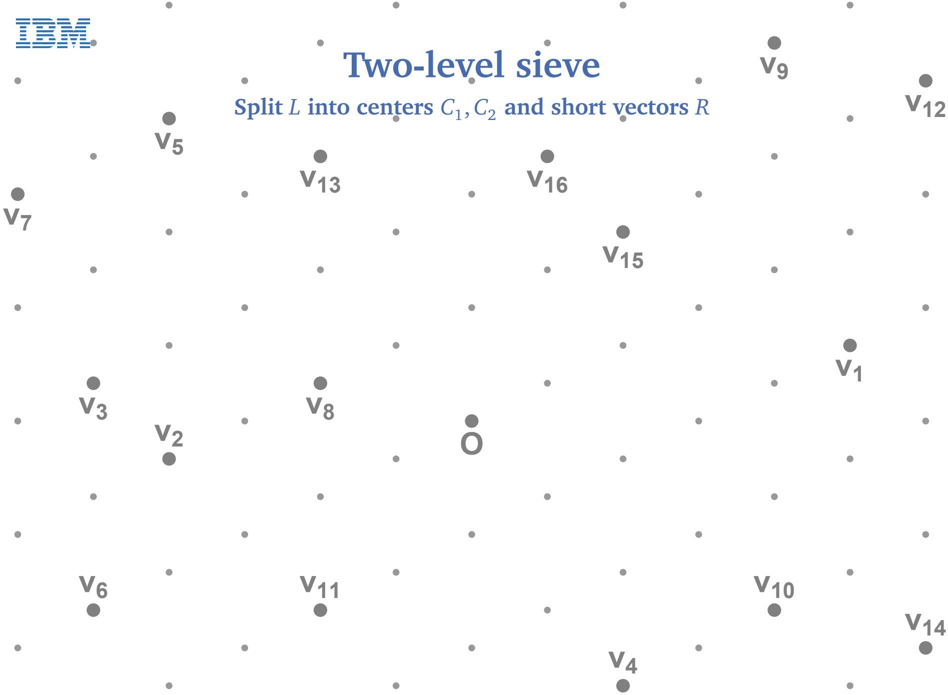
Two-level sieve

Sample a list L of random lattice vectors



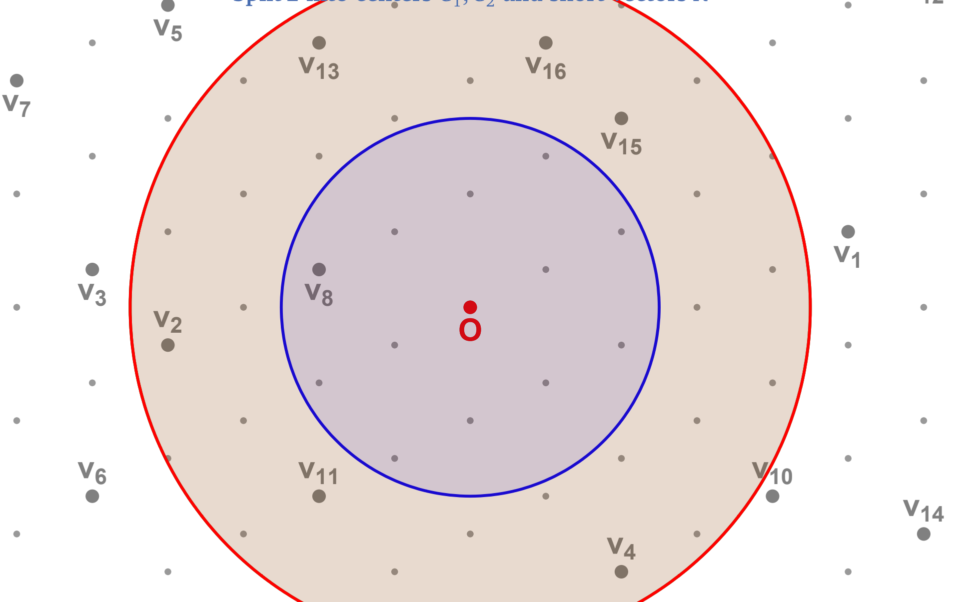
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



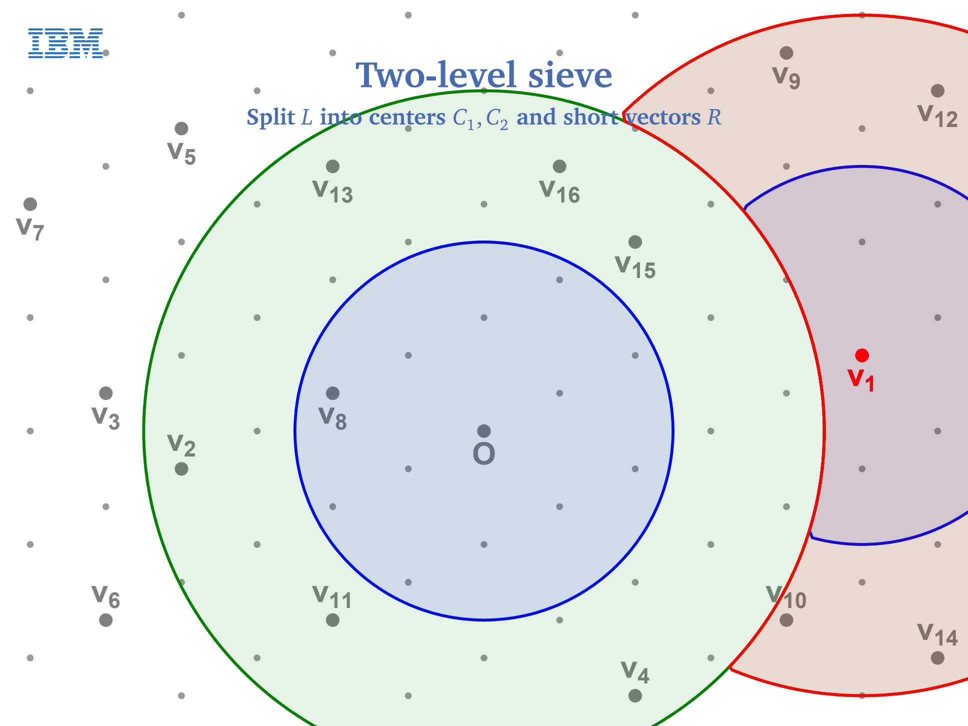
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



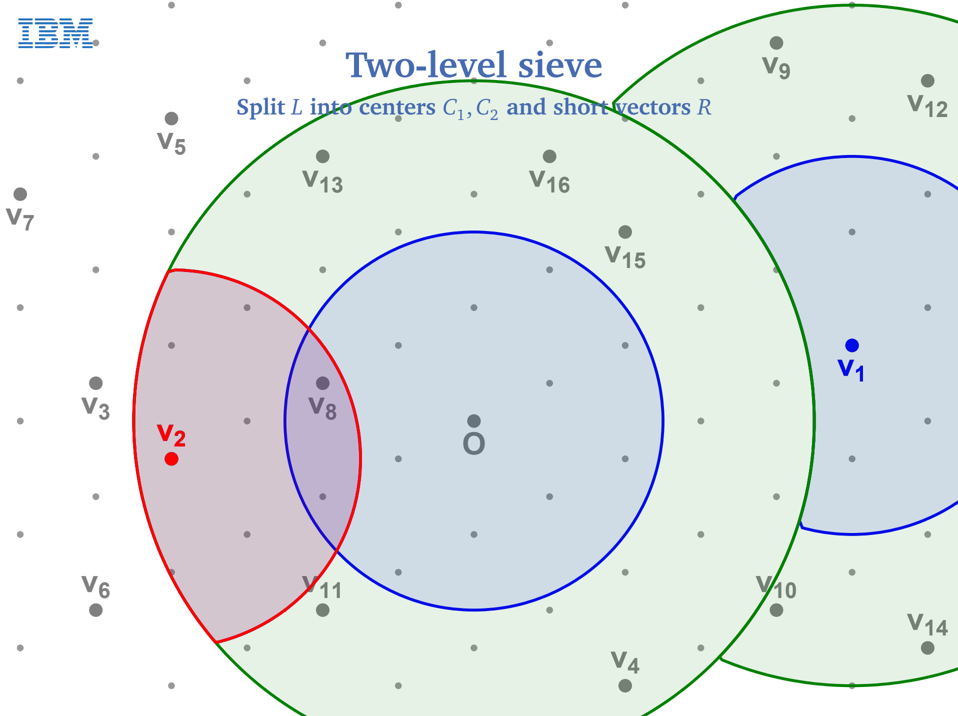
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



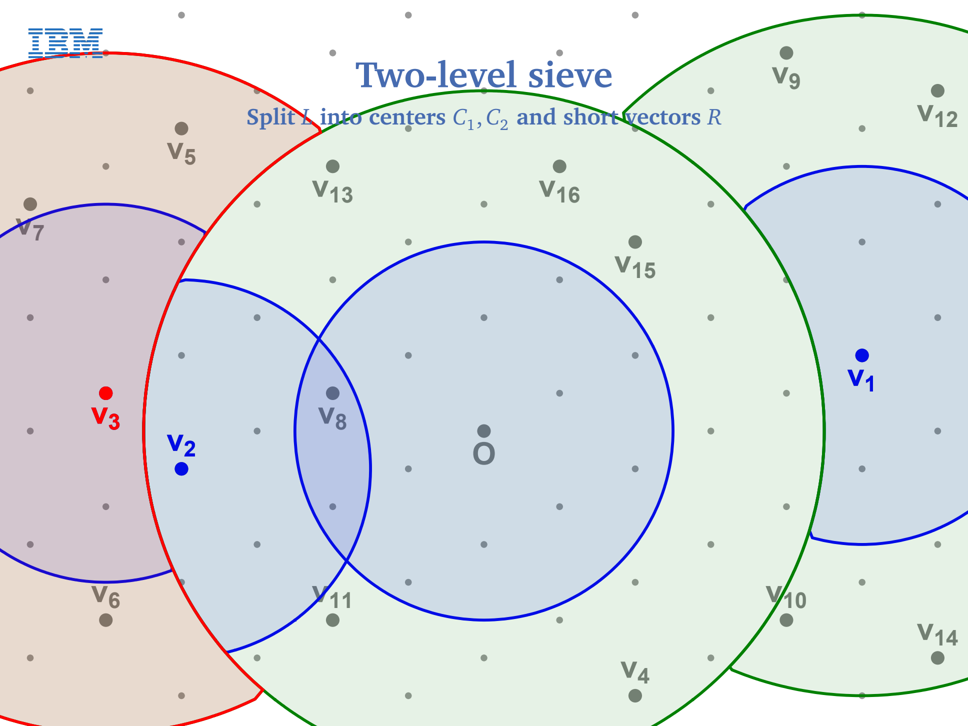
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



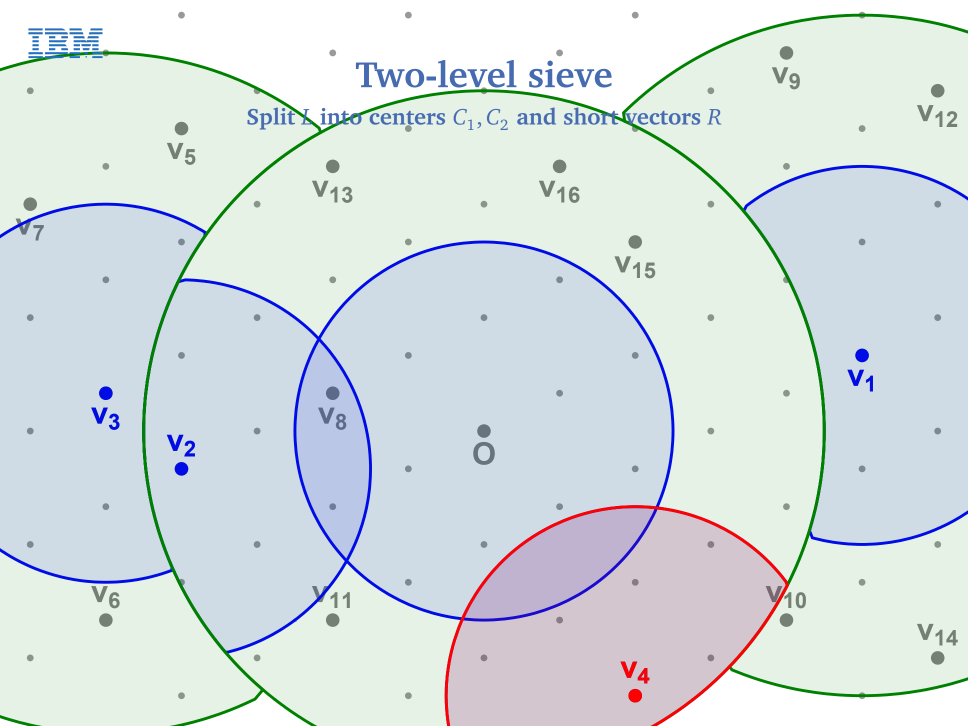
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



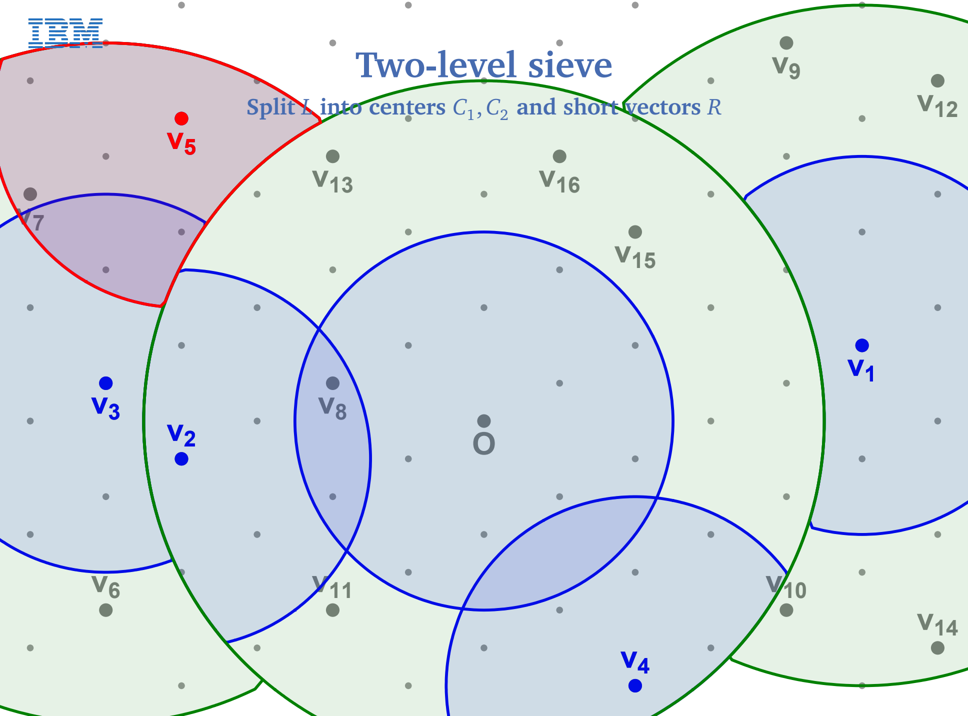
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



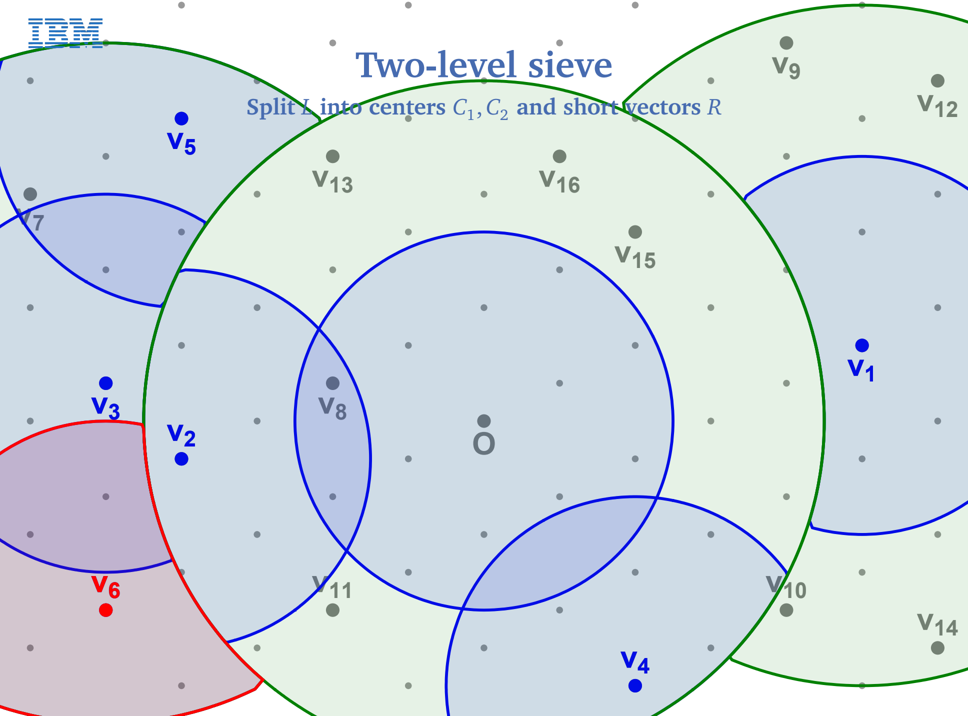
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



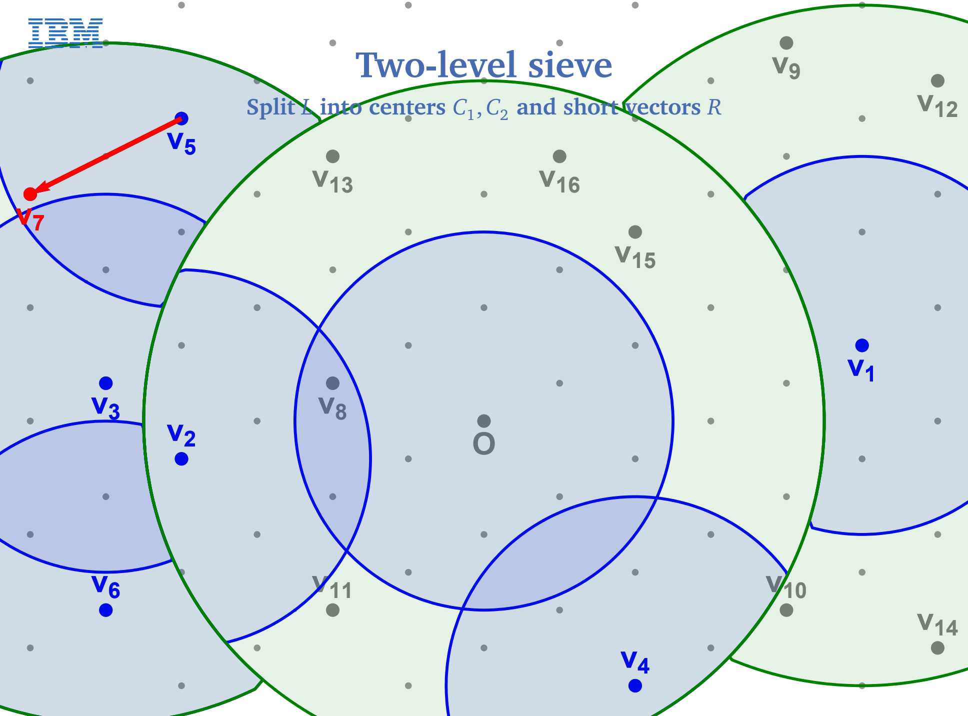
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



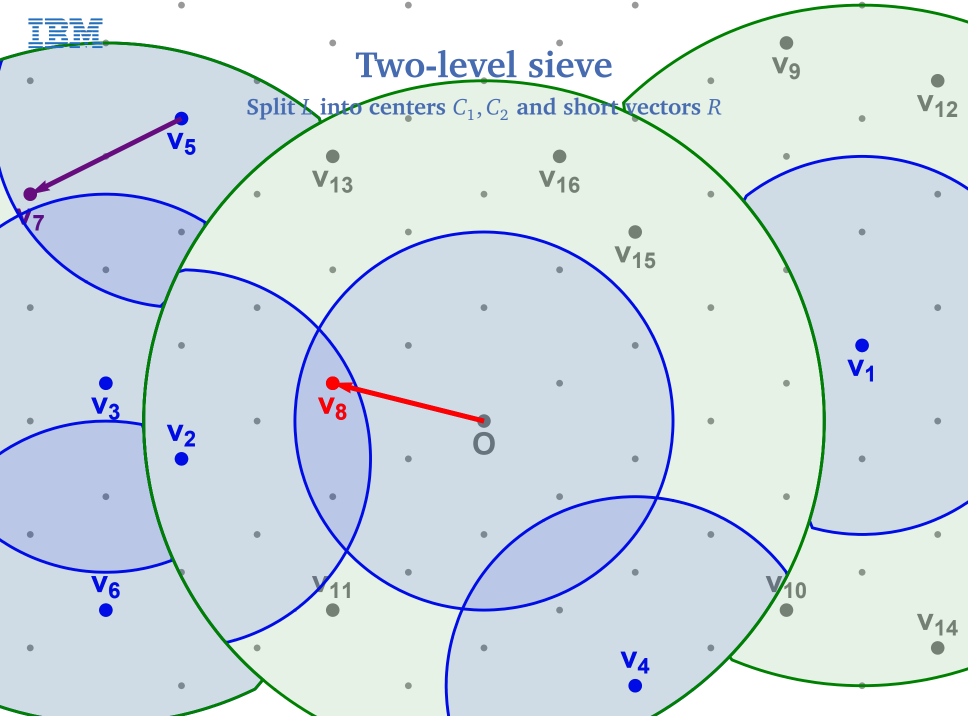
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



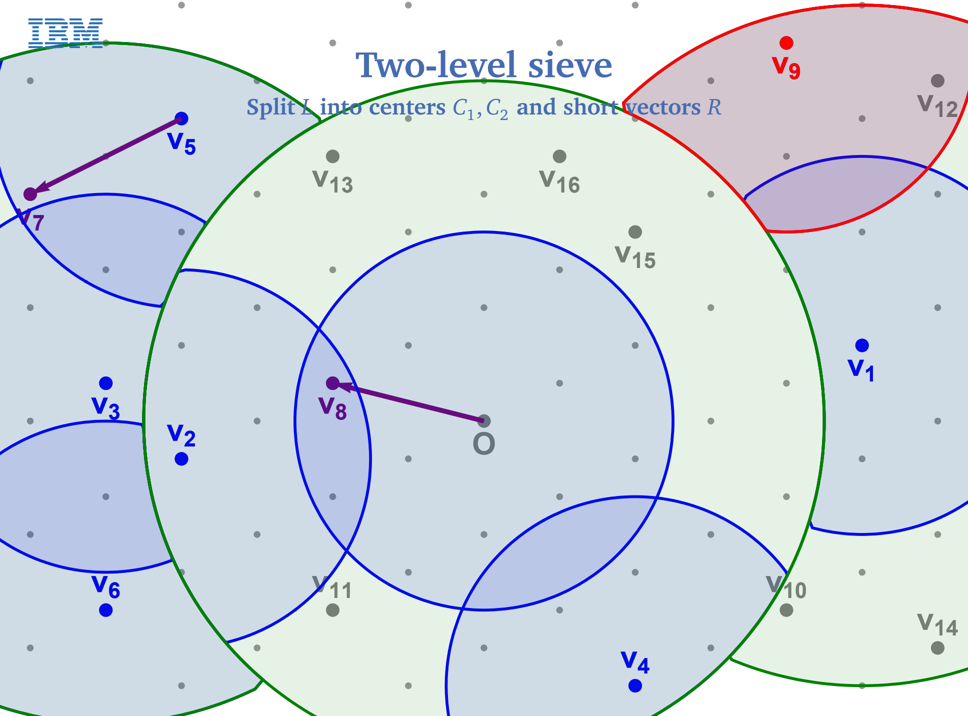
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



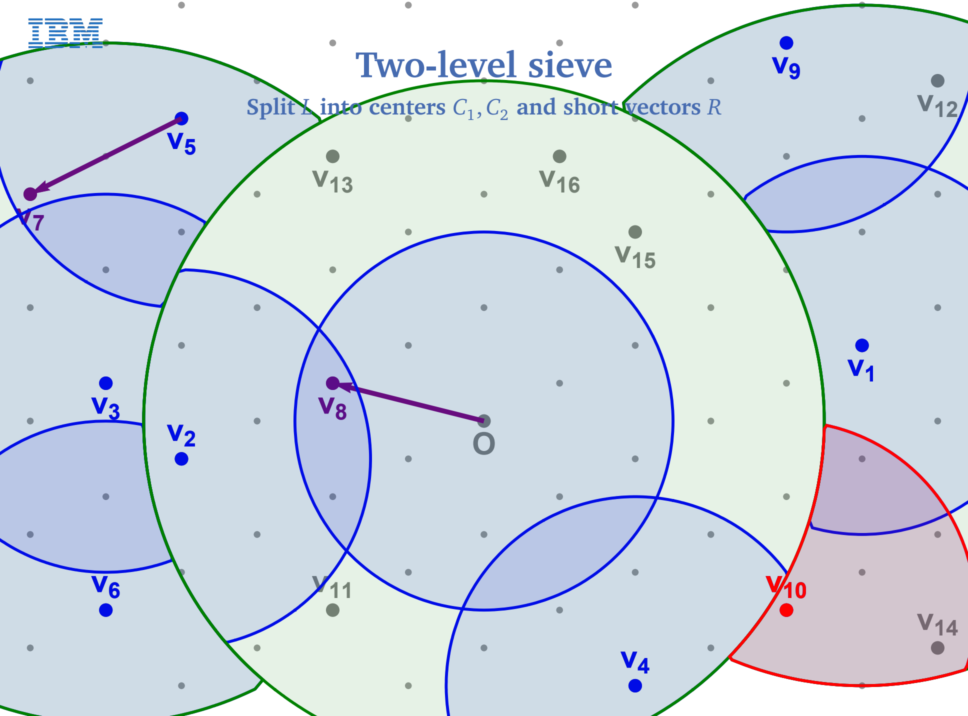
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



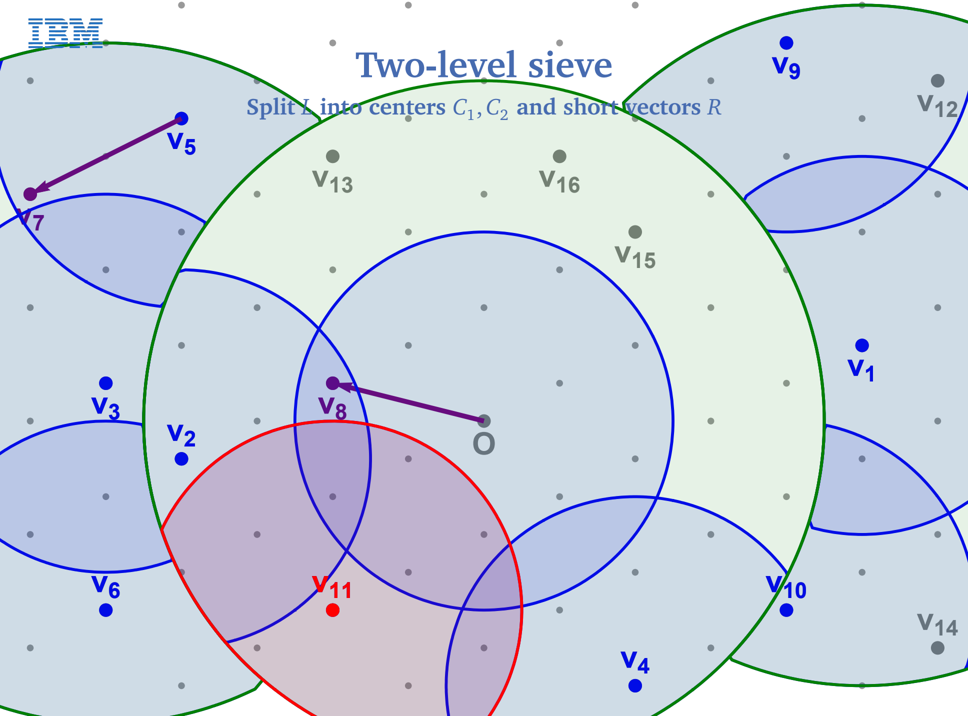
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



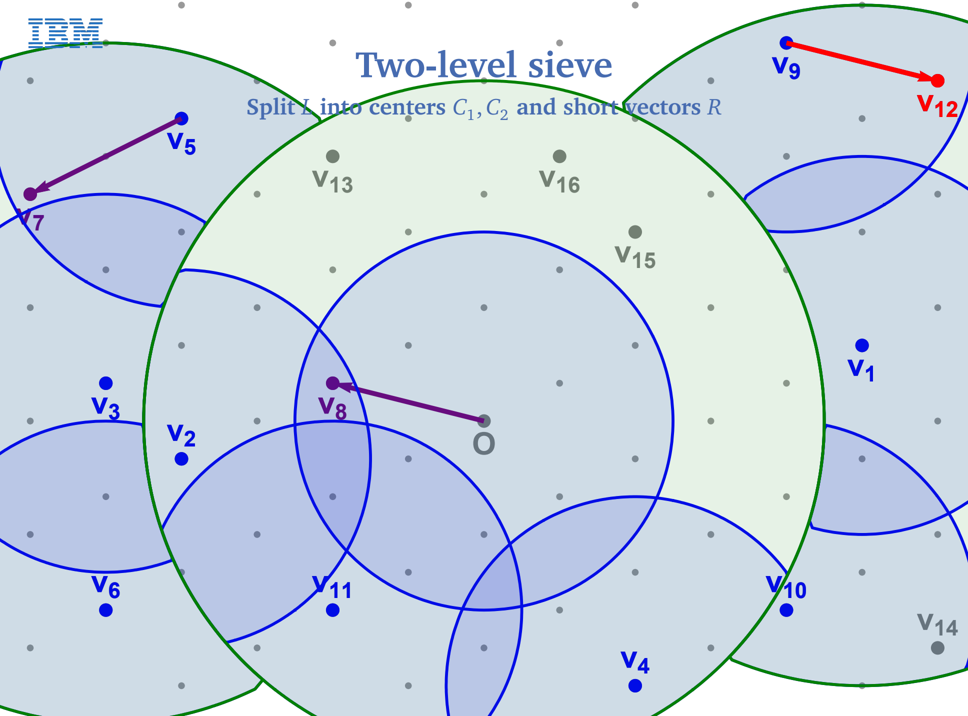
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



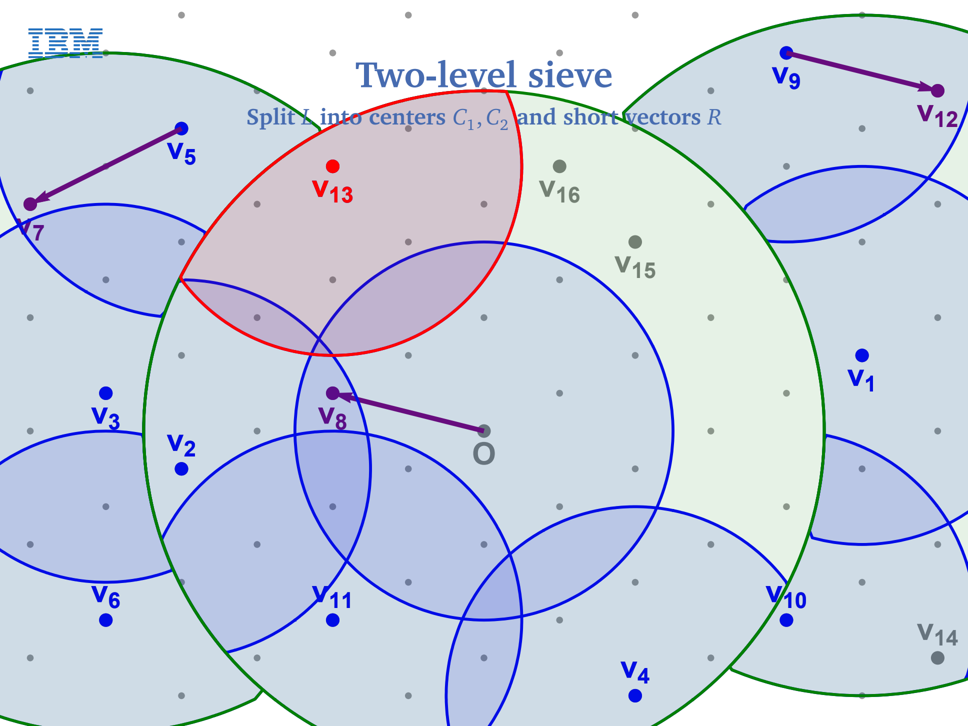
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



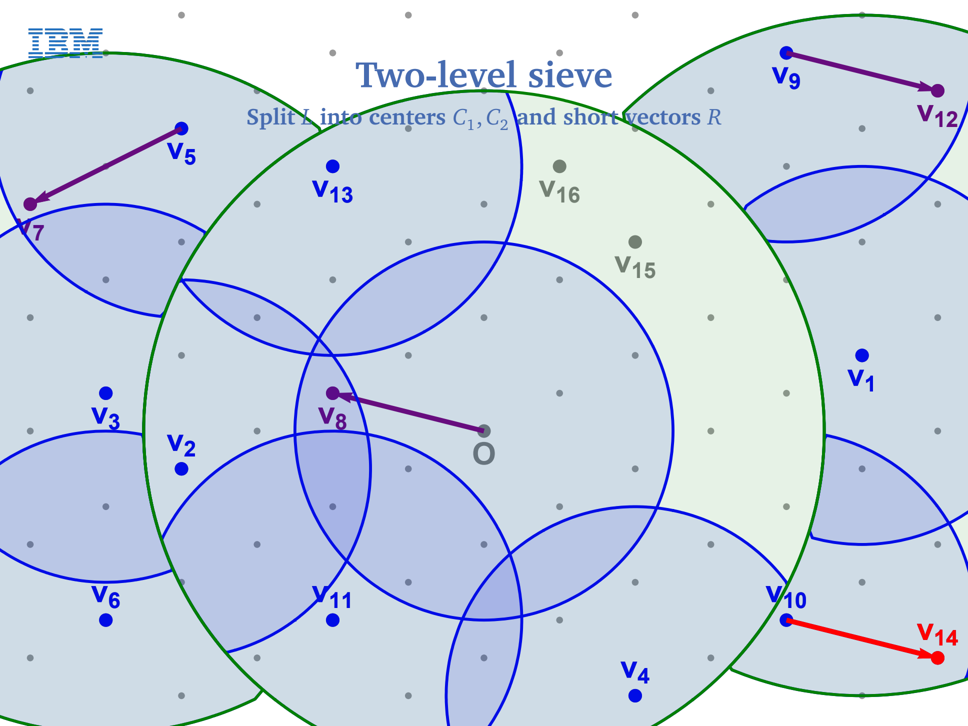
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



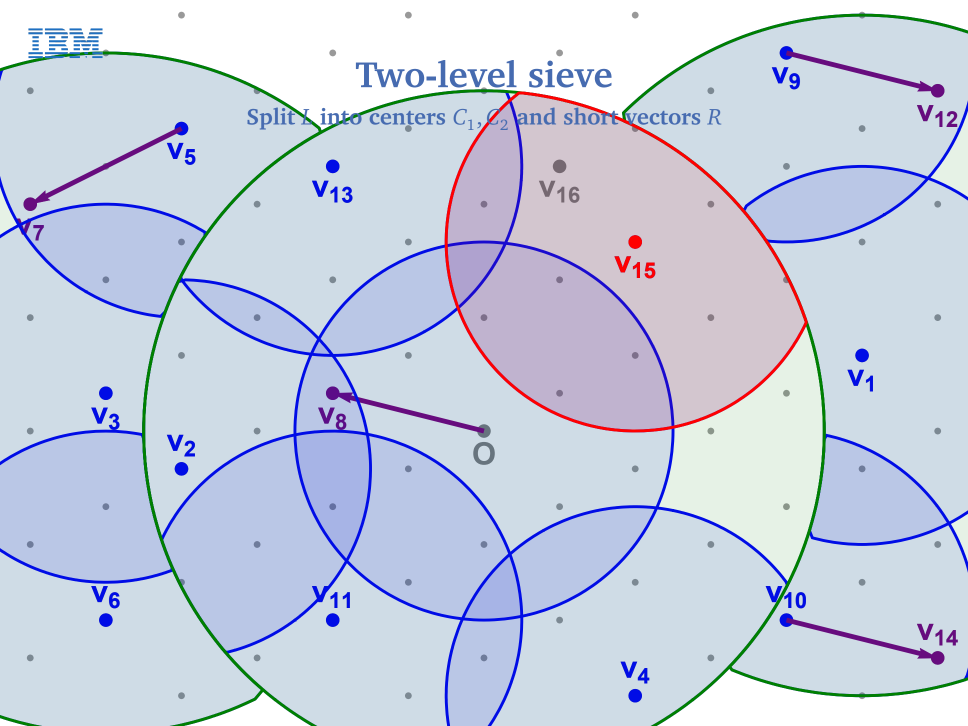
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



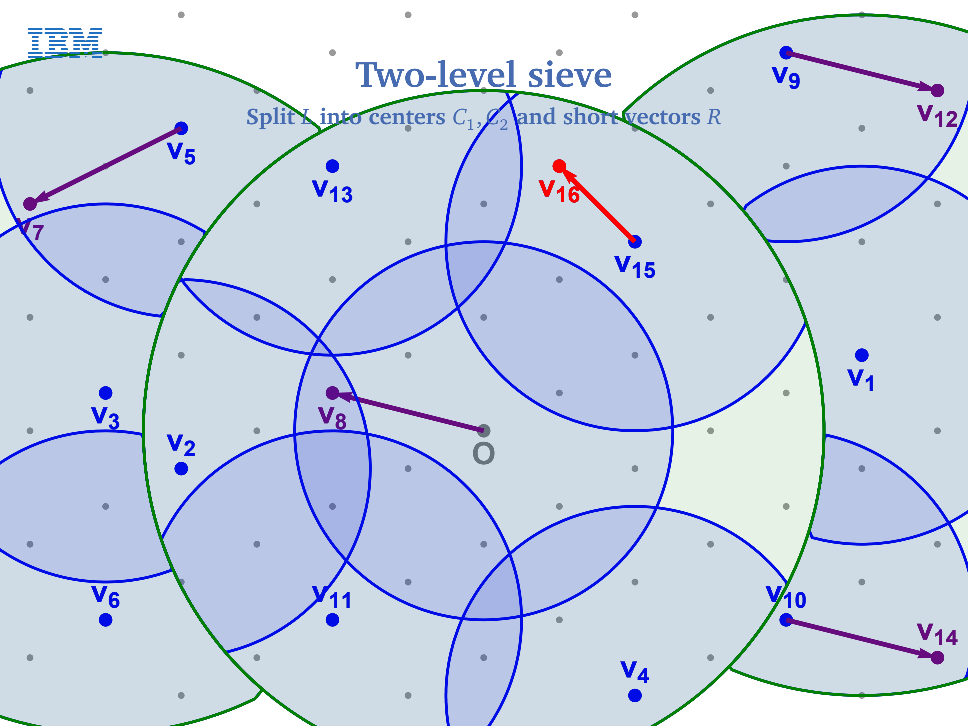
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



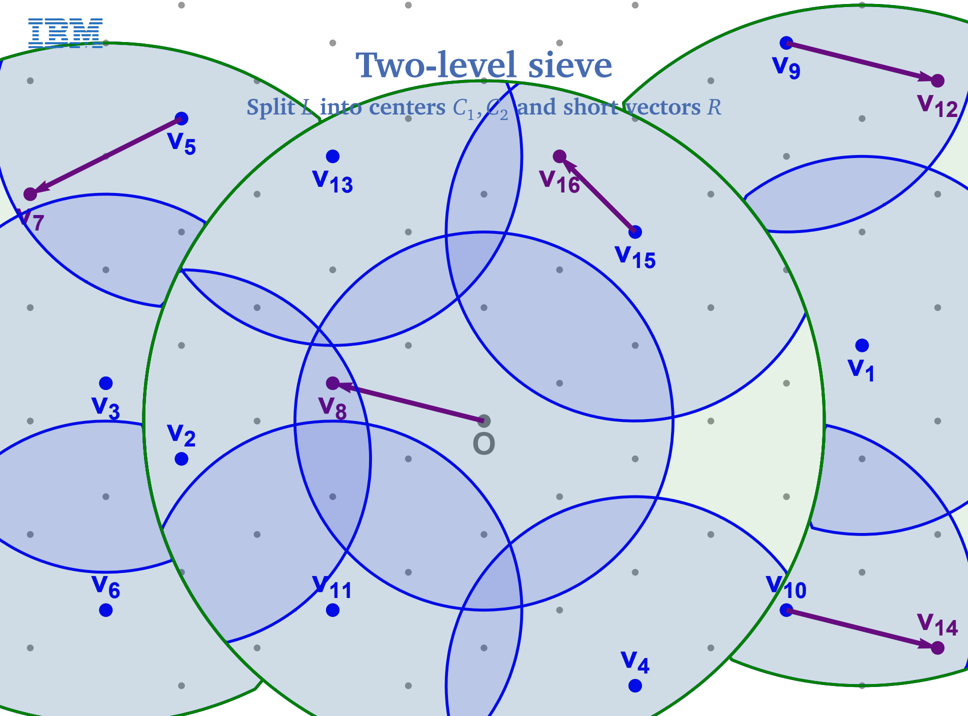
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



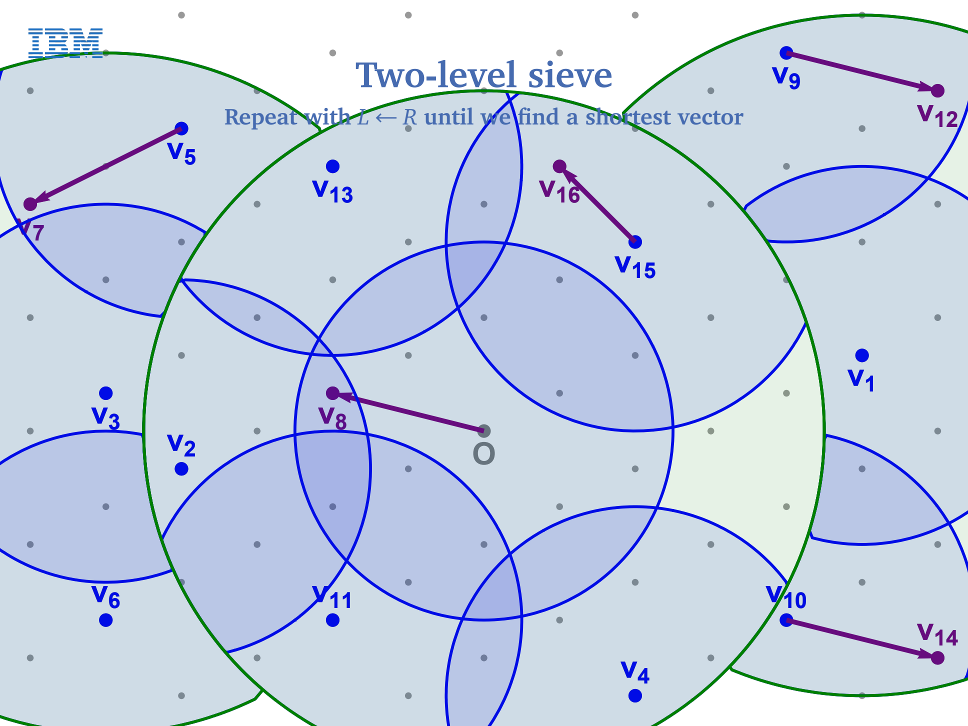
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



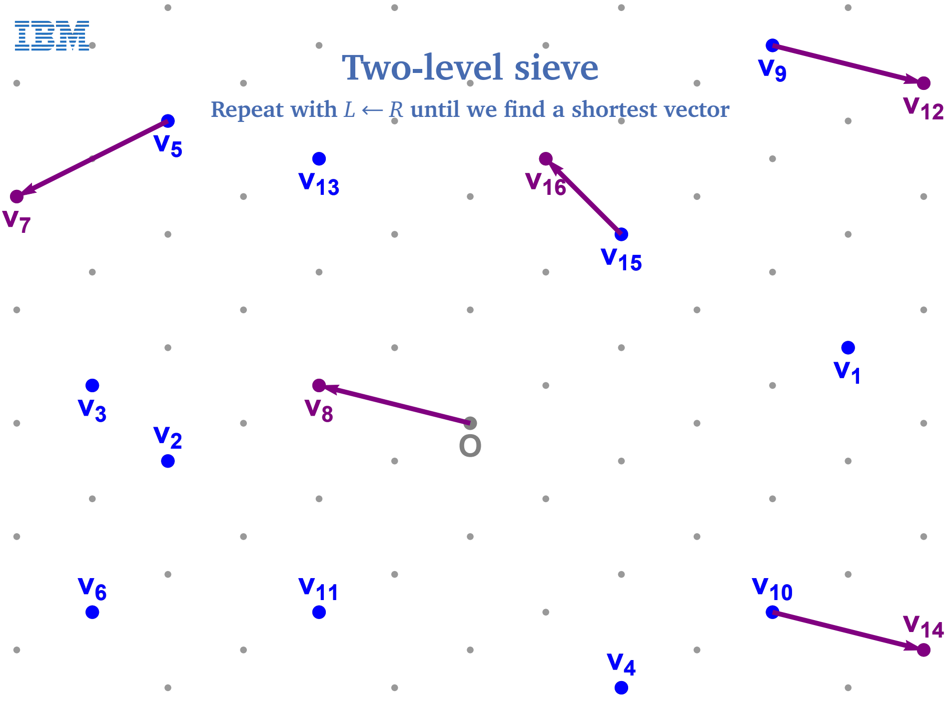
Two-level sieve

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



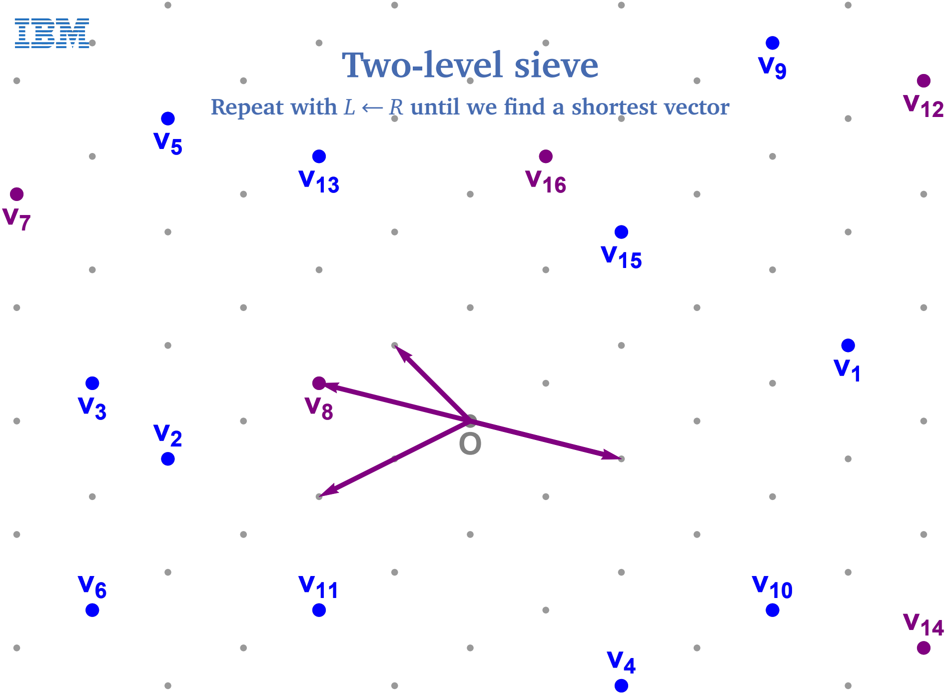
Two-level sieve

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



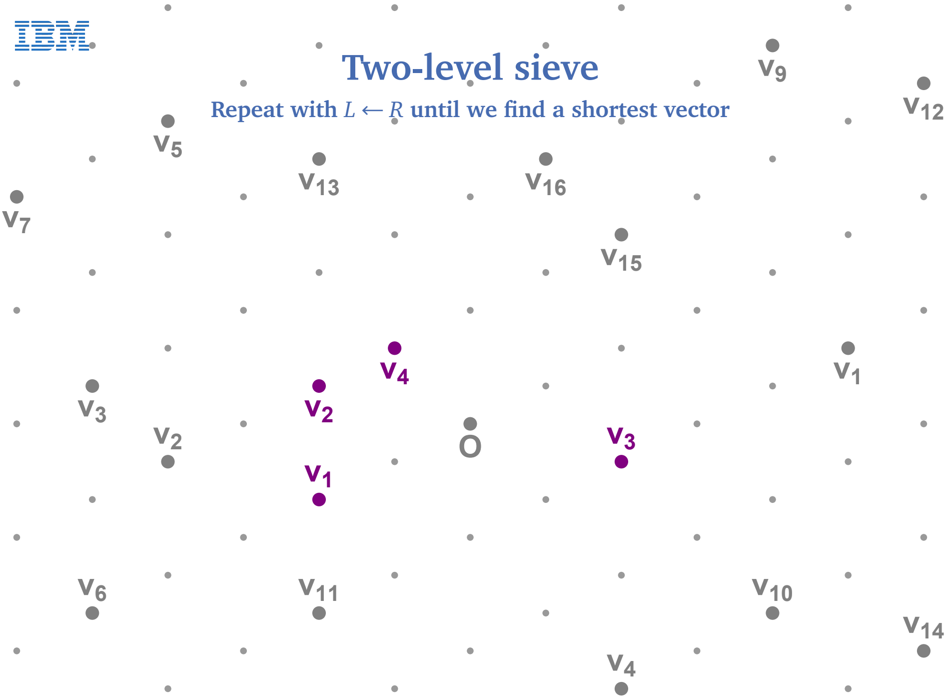
Two-level sieve

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



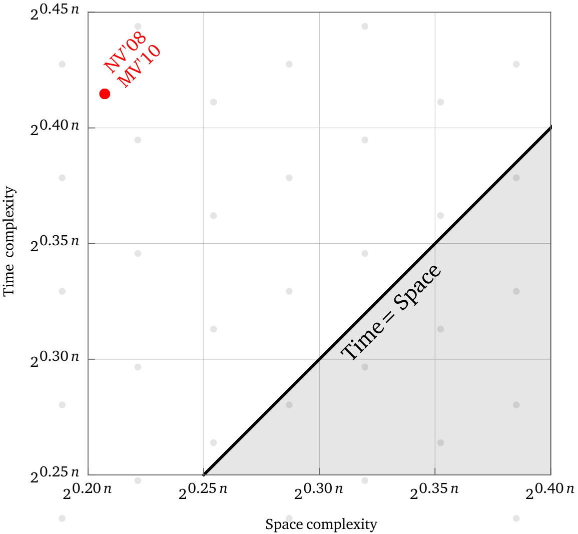
Two-level sieve

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



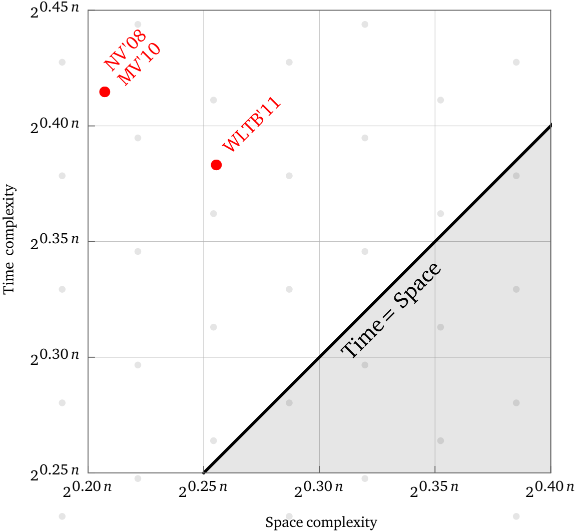
Two-level sieve

Space/time trade-off



Two-level sieve

Space/time trade-off



Three-level sieve

Overview

Heuristic result (Nguyen–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 result (Nguyen–Vidick, J. Math. Crypt. ’08)

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

Heuristic result (Wang–Liu–Tian–Bi, ASIACCS’11)

The two-level sieve runs in time $2^{0.3836n}$ *and space* $2^{0.2557n}$.

Three-level sieve

Overview

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

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

Heuristic result (Wang–Liu–Tian–Bi, ASIACCS'11)

The two-level sieve runs in time $2^{0.3836n}$ *and space* $2^{0.2557n}$.

Heuristic result (Zhang–Pan–Hu, SAC'13)

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

Three-level sieve

Overview

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

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

Heuristic result (Wang–Liu–Tian–Bi, ASIACCS'11)

The two-level sieve runs in time $2^{0.3836n}$ *and space* $2^{0.2557n}$.

Heuristic result (Zhang–Pan–Hu, SAC'13)

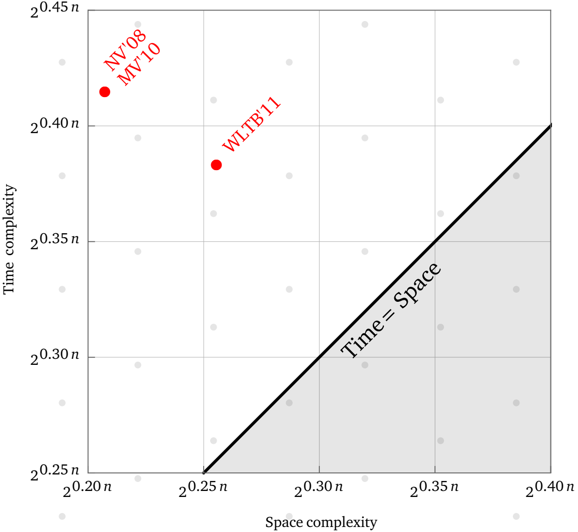
The three-level sieve runs in time $2^{0.3778n}$ *and space* $2^{0.2833n}$.

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.*

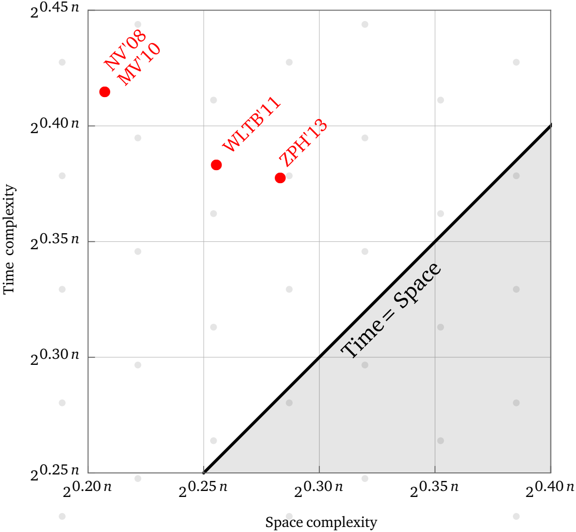
Three-level sieve

Space/time trade-off



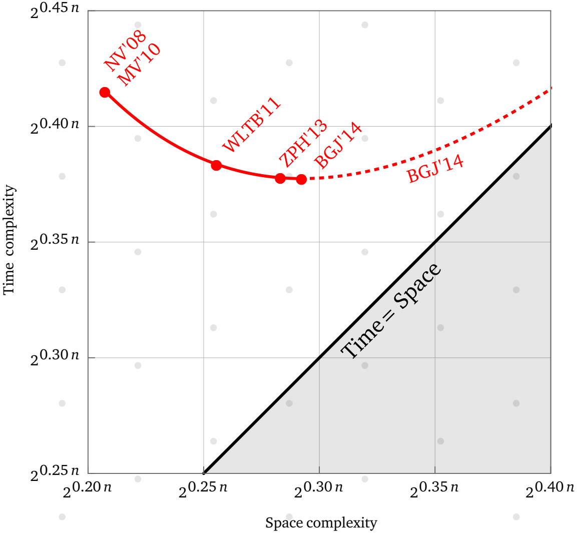
Three-level sieve

Space/time trade-off



Decomposition approach

Space/time trade-off



Locality-sensitive hashing

Introduction

Problem: Given a high-dimensional data set $D \subset \mathbb{R}^n$, preprocess it such that when later given a target $\boldsymbol{t} \in \mathbb{R}^n$, we can quickly find a nearby vector to $\boldsymbol{t}$ in D.

Locality-sensitive hashing

Introduction

Problem: Given a high-dimensional data set $D \subset \mathbb{R}^n$, preprocess it such that when later given a target $\boldsymbol{t} \in \mathbb{R}^n$, we can quickly find a nearby vector to $\boldsymbol{t}$ in D.

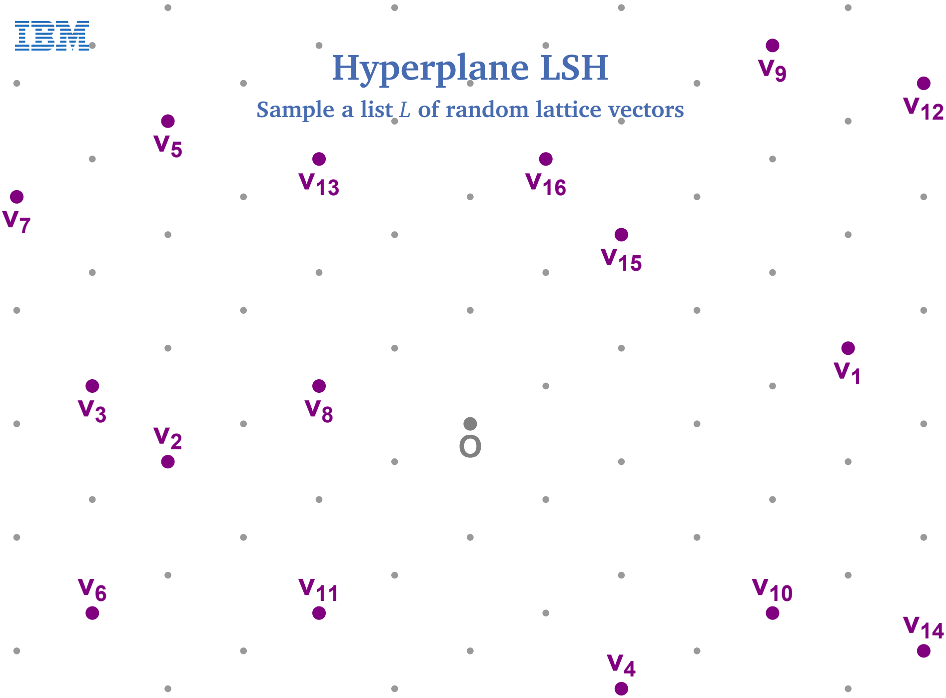
> *"The key idea is to use hash functions such that the probability of collision is much higher for objects that are close to each other than for those that are far apart."*
>
> — Indyk–Motwani, *STOC'98*

Hyperplane LSH

Sample a list L of random lattice vectors

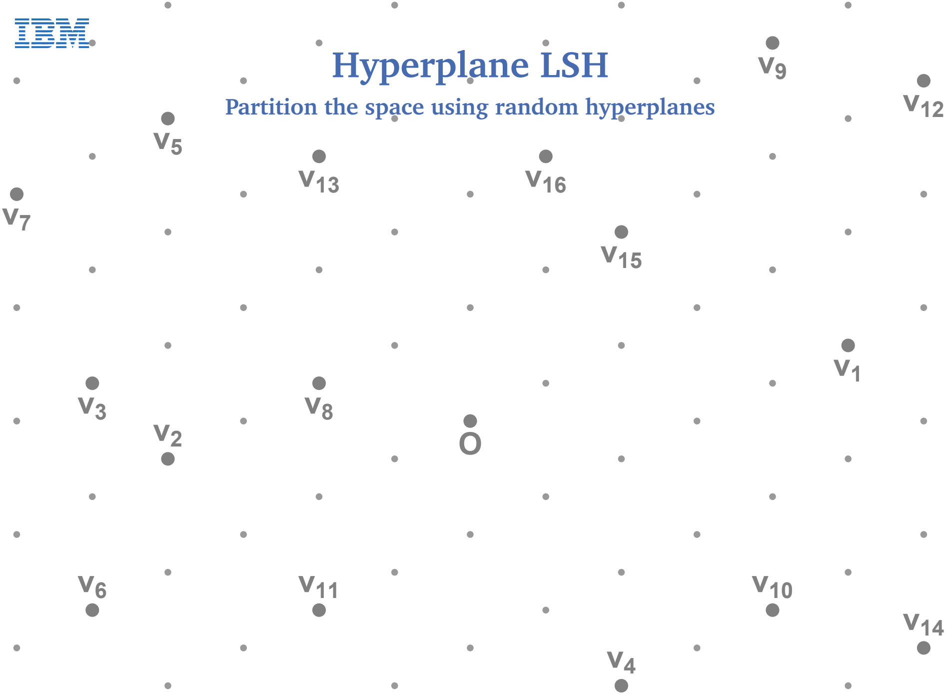
Hyperplane LSH

Sample a list L of random lattice vectors



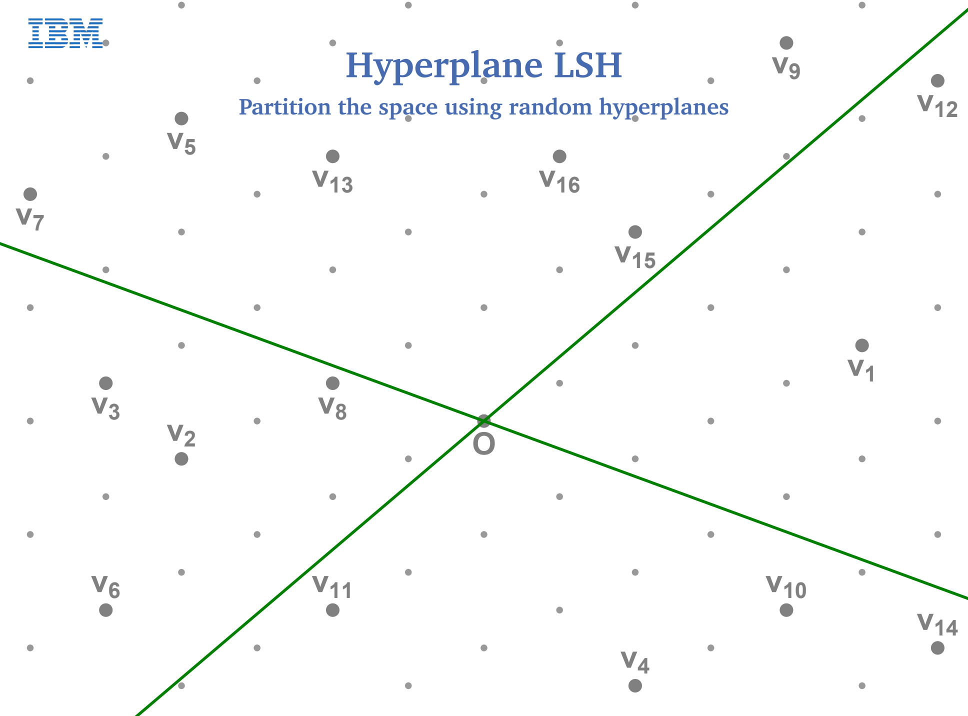
Hyperplane LSH

Partition the space using random hyperplanes



Hyperplane LSH

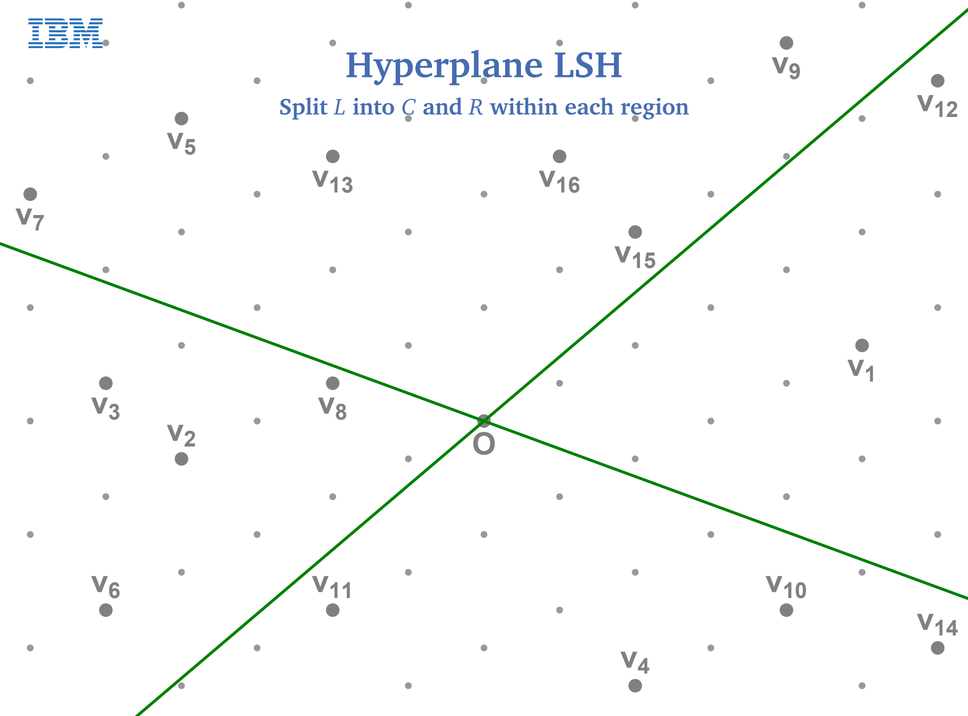
Partition the space using random hyperplanes



IBM

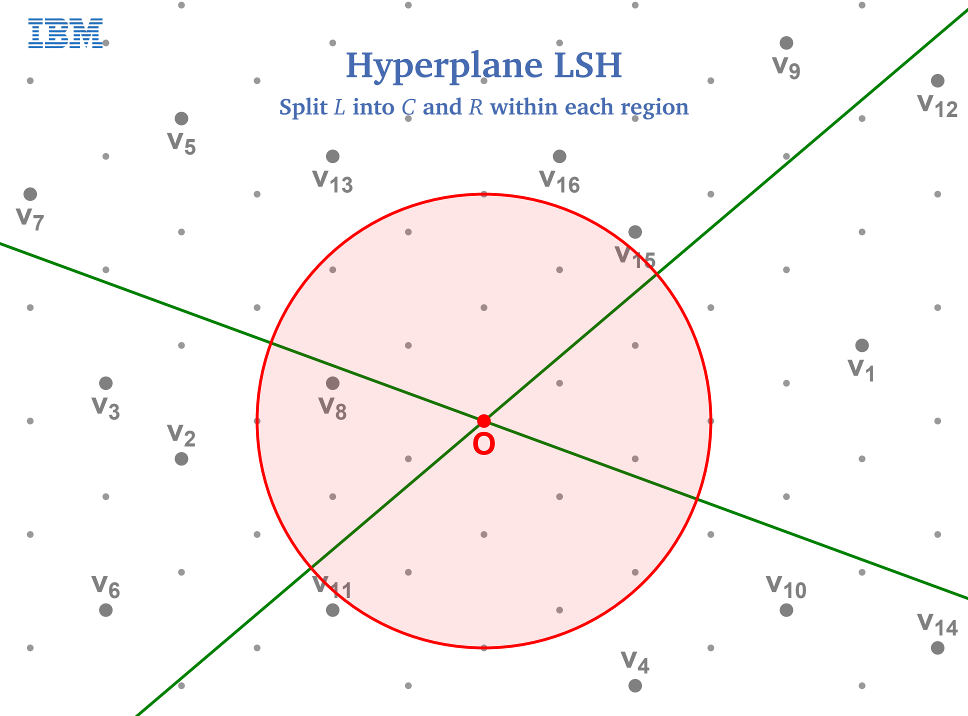
Hyperplane LSH

Split L into C and R within each region



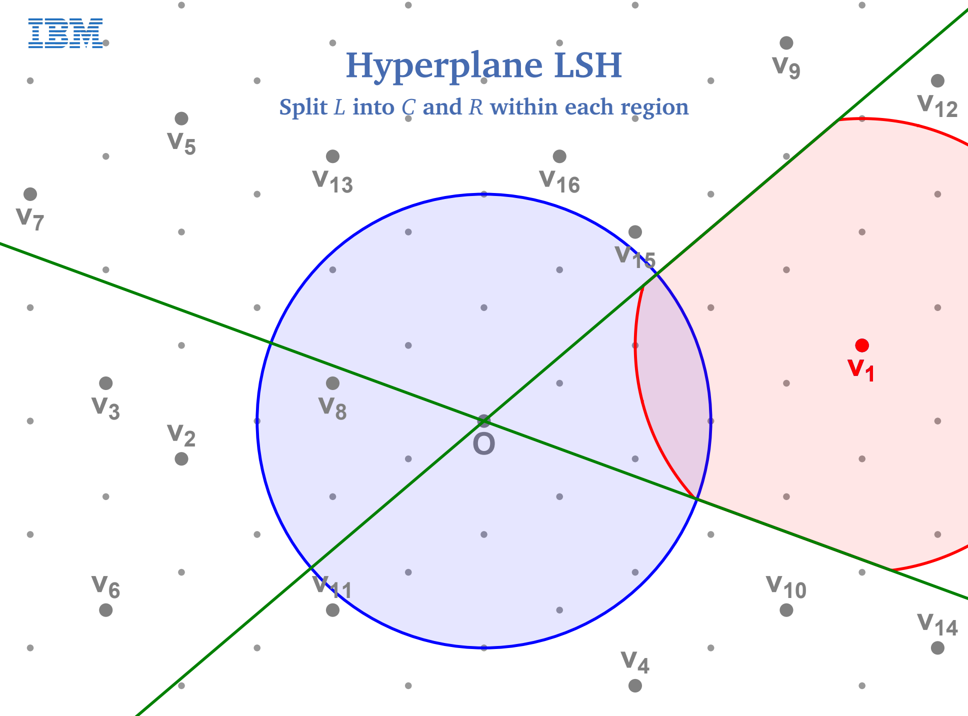
Hyperplane LSH

Split L into C and R within each region



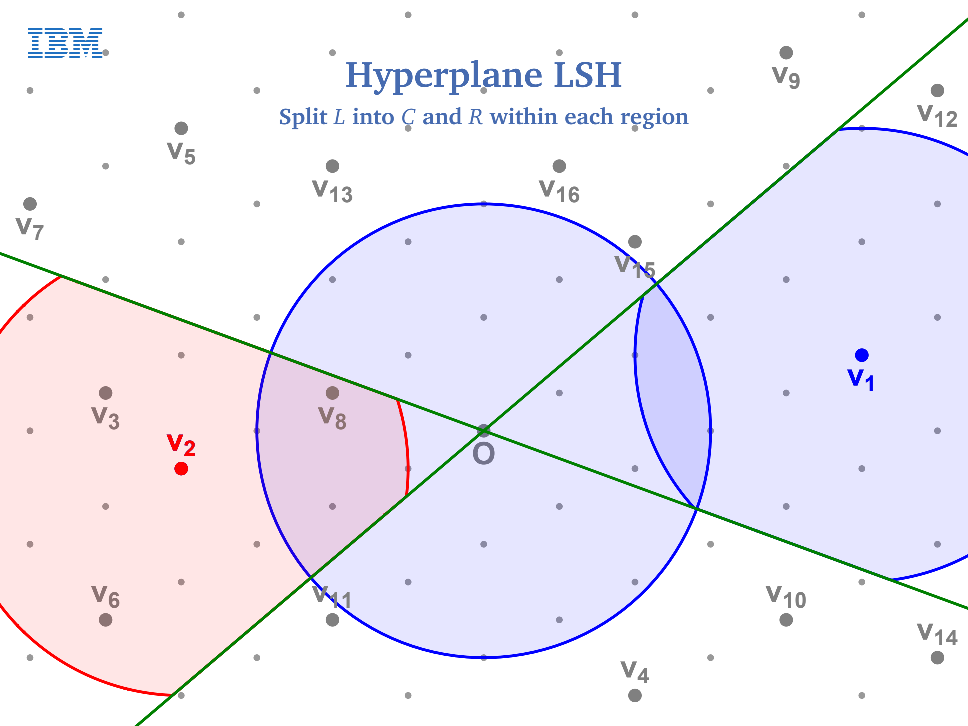
Hyperplane LSH

Split *L* into *C* and *R* within each region



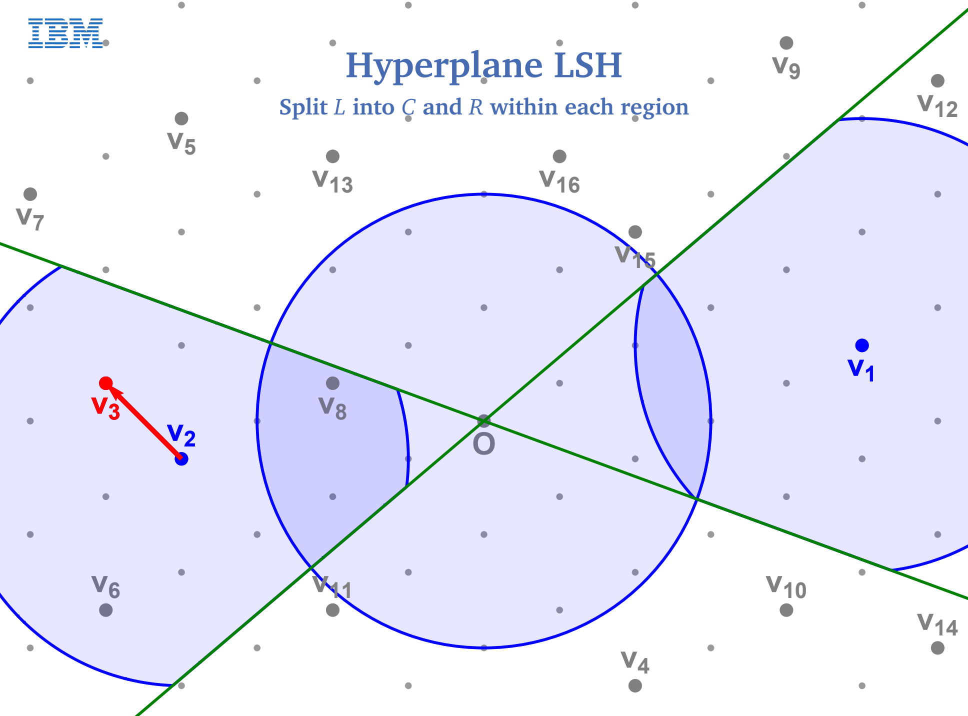
Hyperplane LSH

Split L into C and R within each region



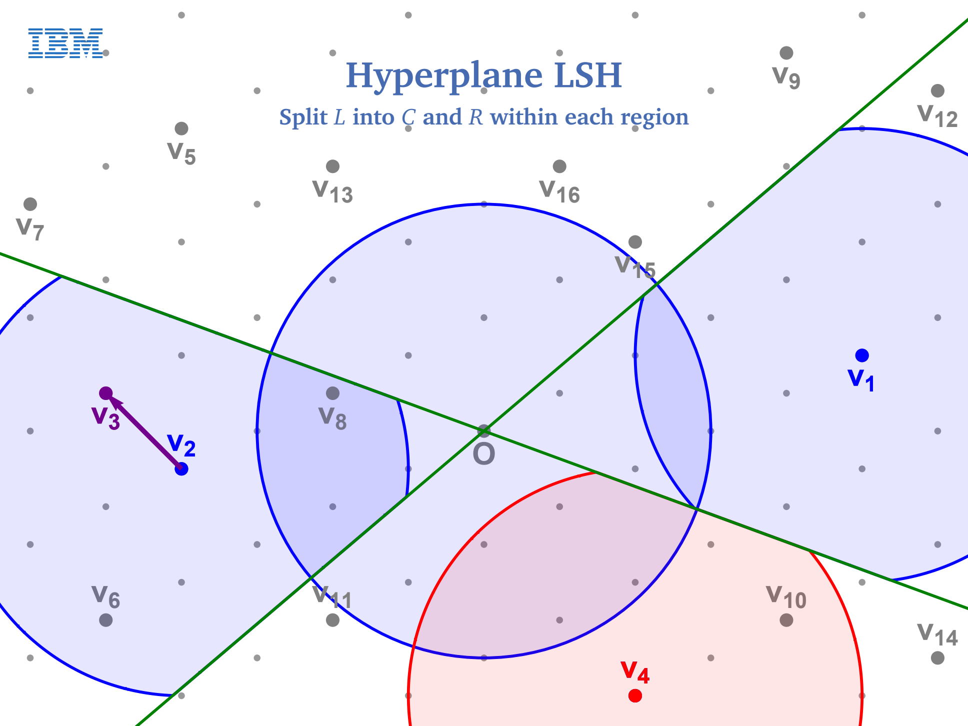
Hyperplane LSH

Split L into C and R within each region



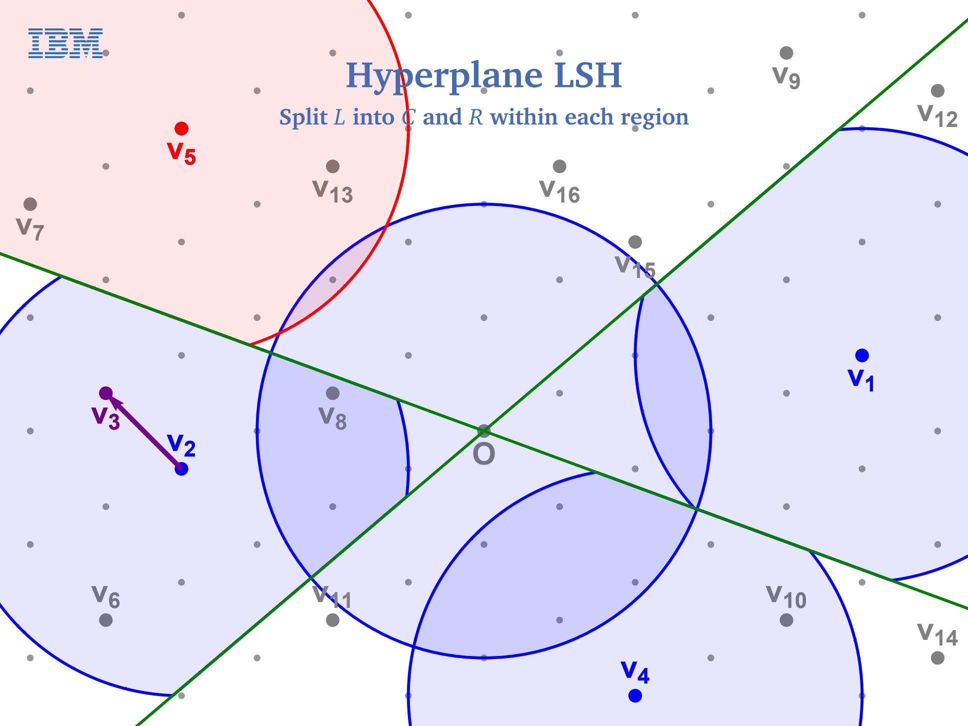
Hyperplane LSH

Split L into C and R within each region



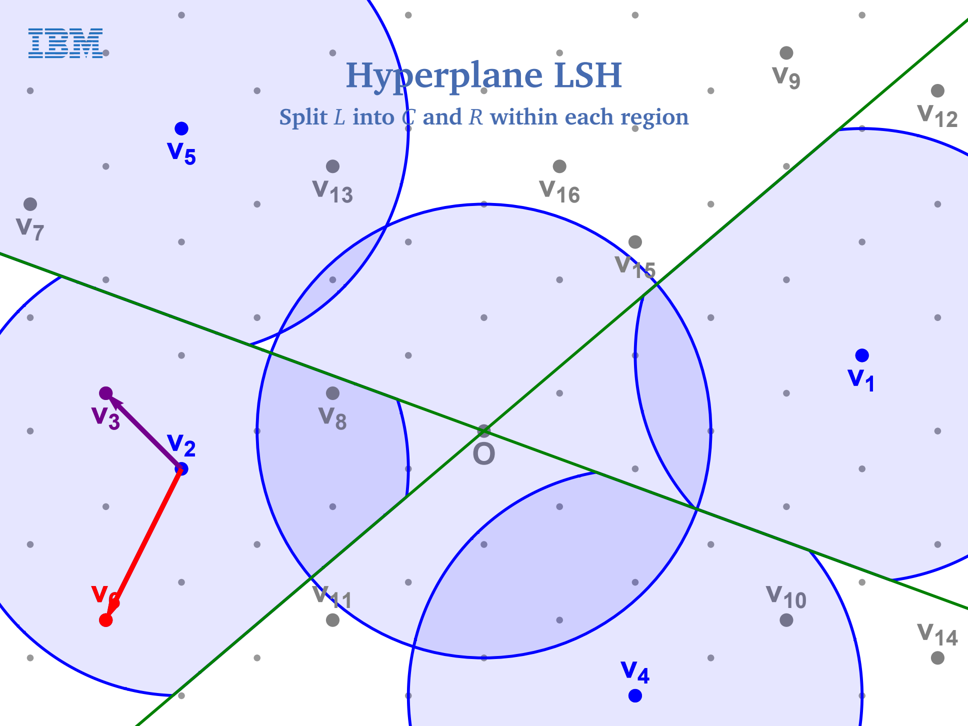
Hyperplane LSH

Split L into C and R within each region



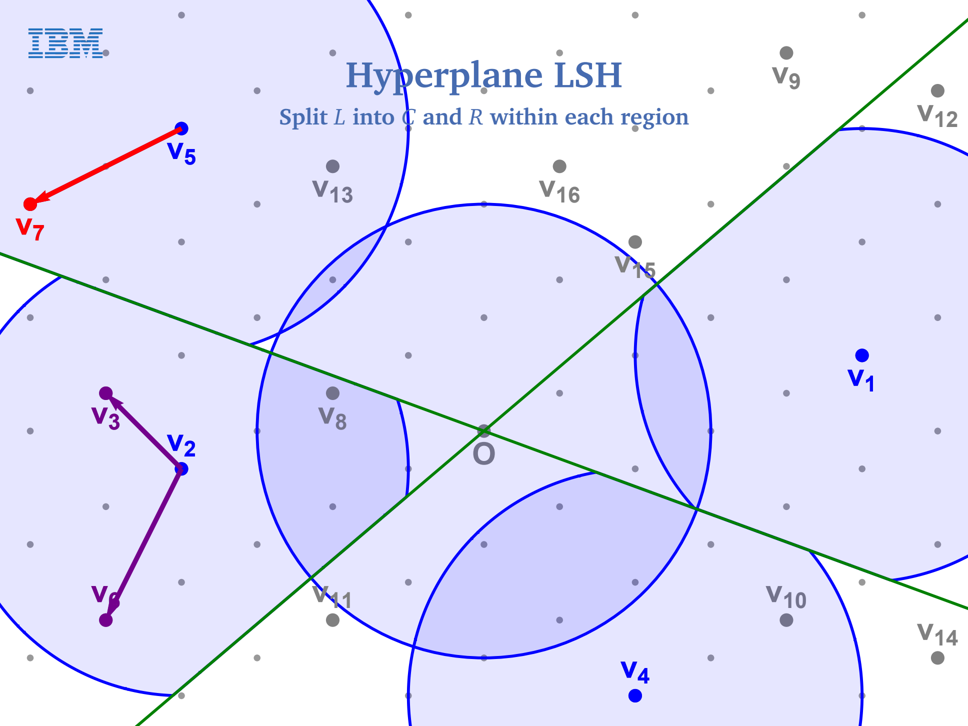
Hyperplane LSH

Split L into C and R within each region



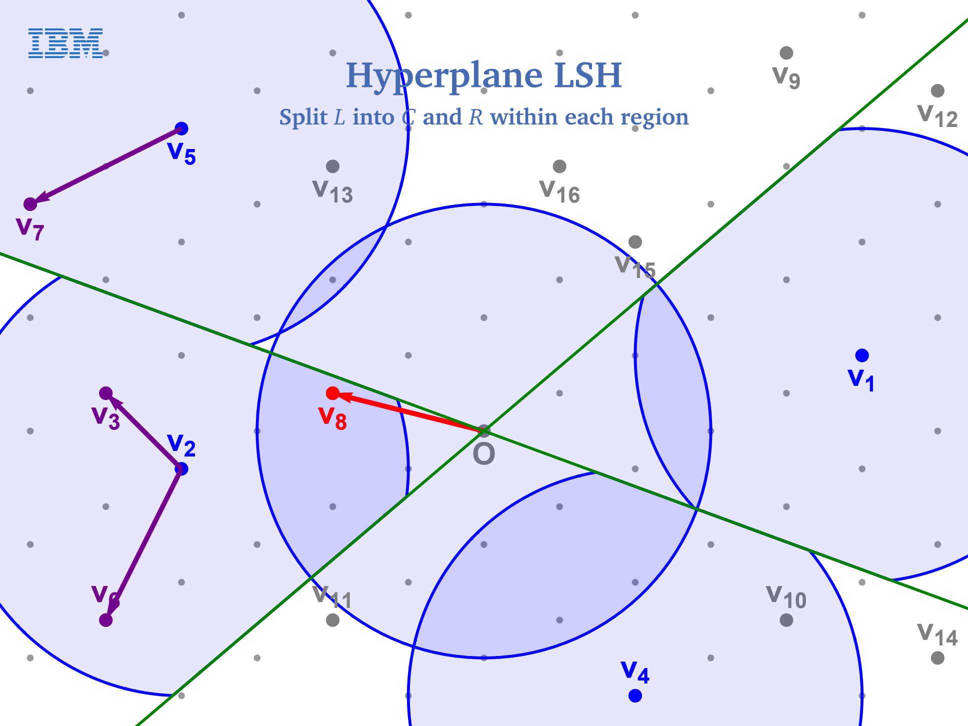
Hyperplane LSH

Split L into C and R within each region



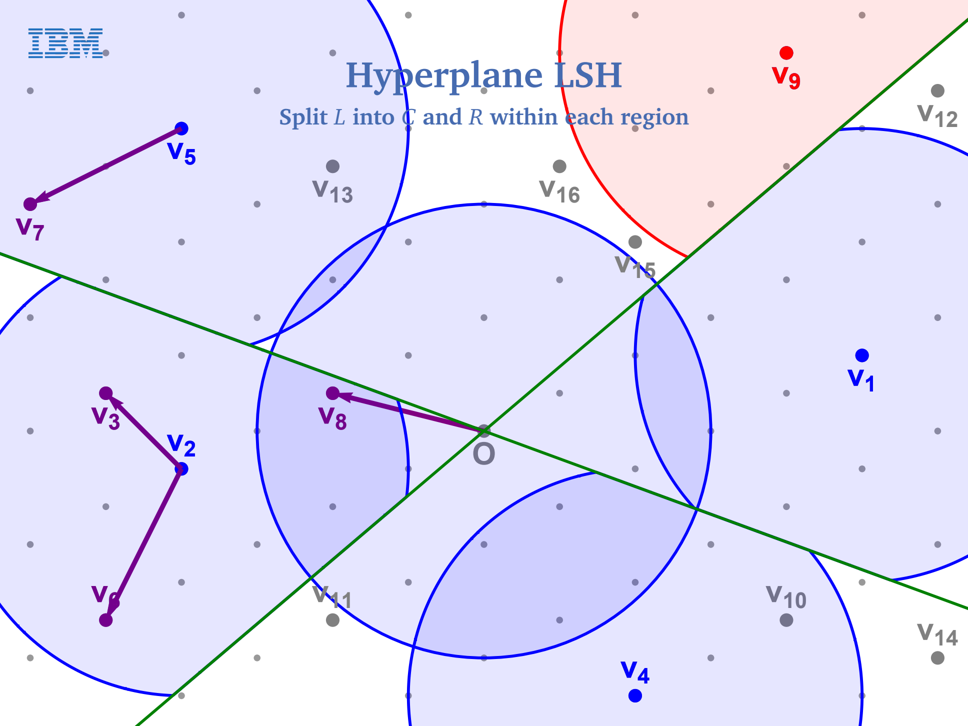
Hyperplane LSH

Split L into C and R within each region



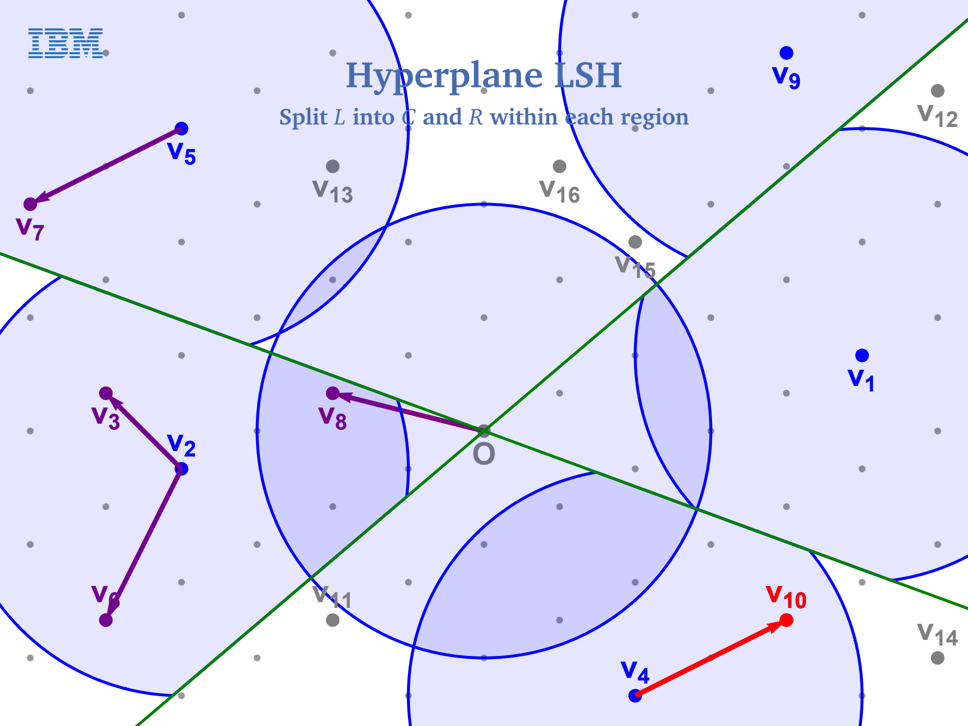
Hyperplane LSH

Split L into C and R within each region



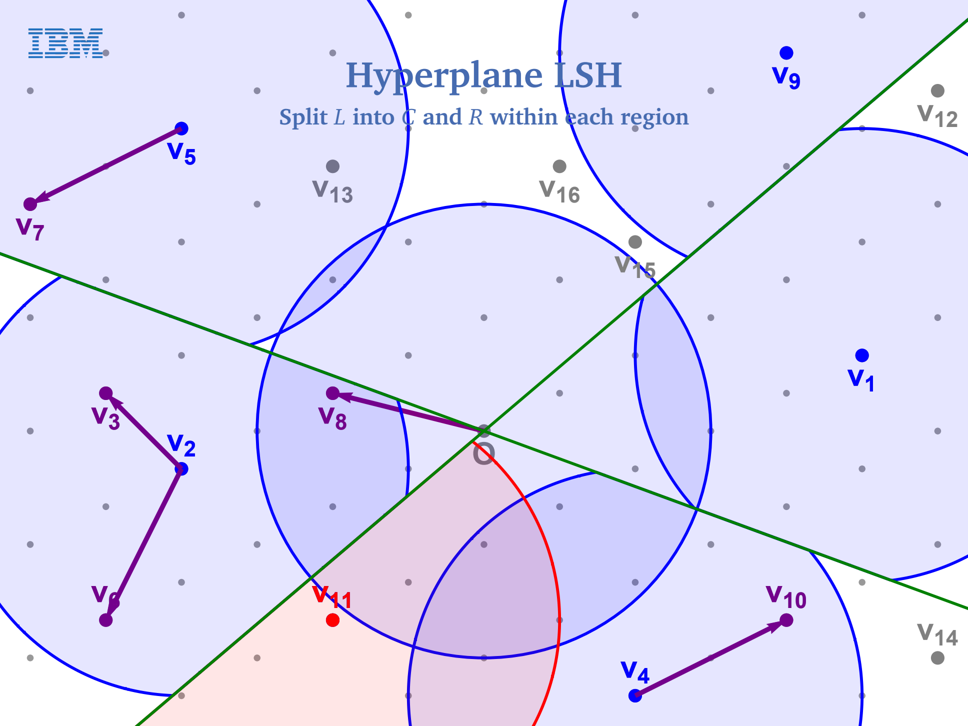
Hyperplane LSH

Split L into C and R within each region



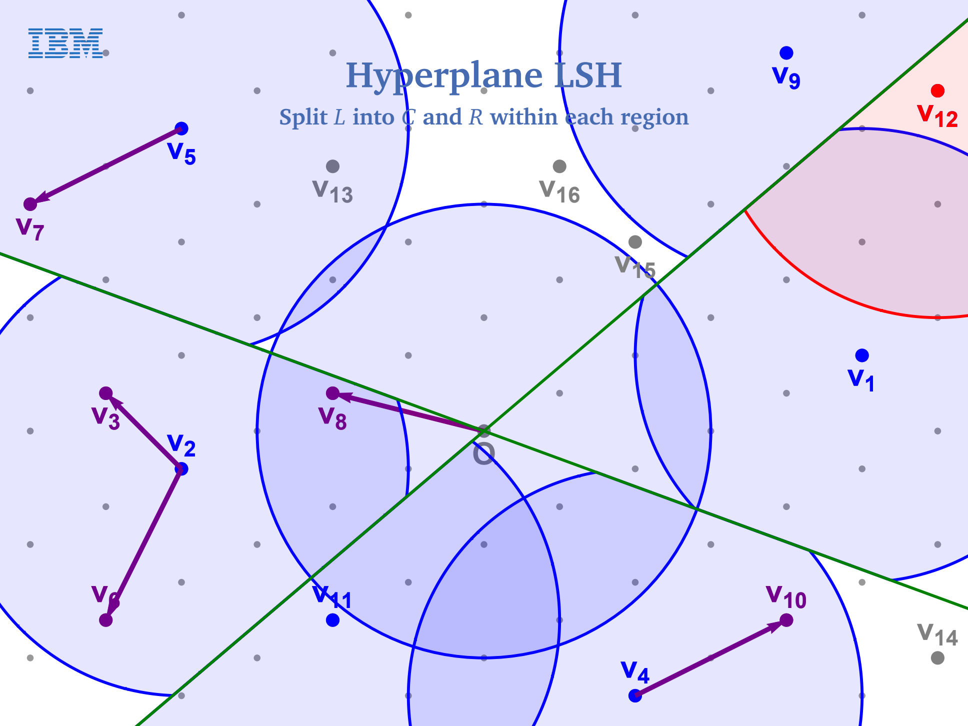
Hyperplane LSH

Split L into C and R within each region



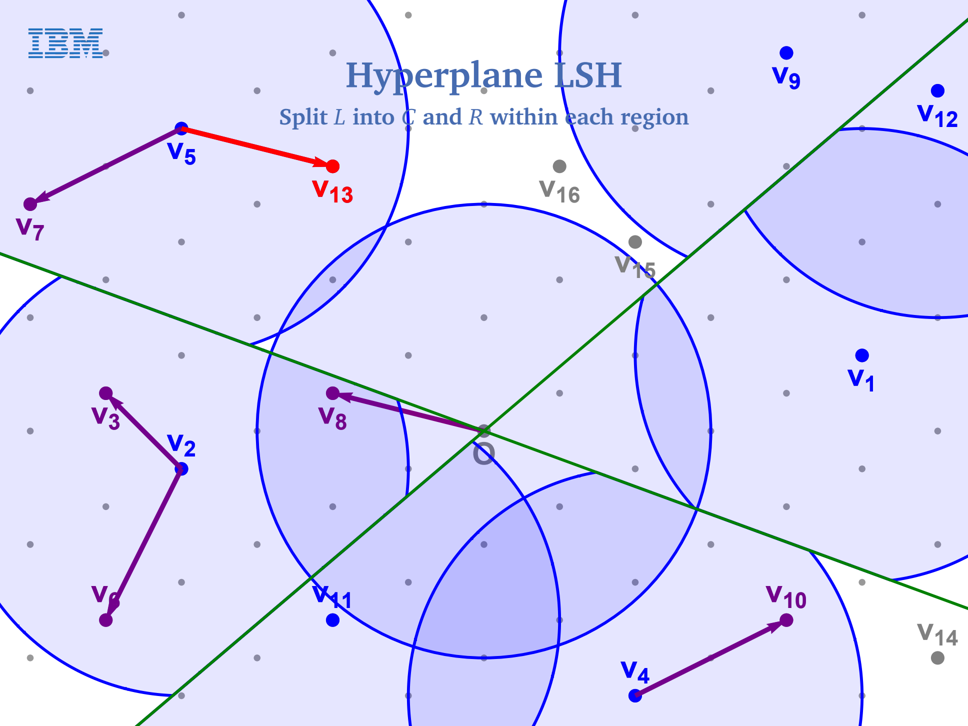
Hyperplane LSH

Split L into C and R within each region



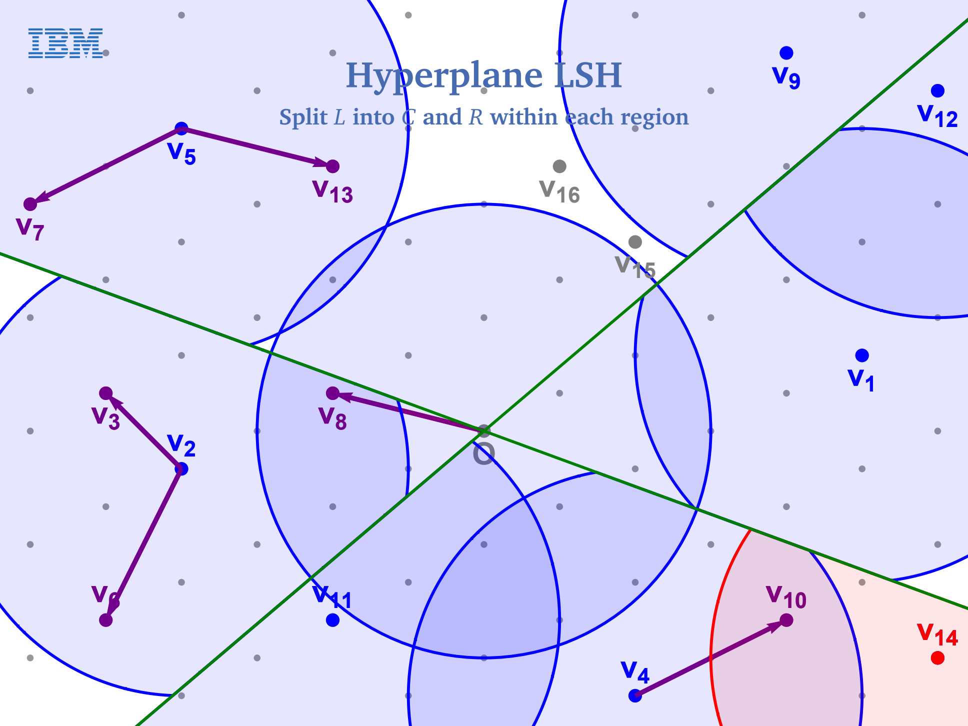
Hyperplane LSH

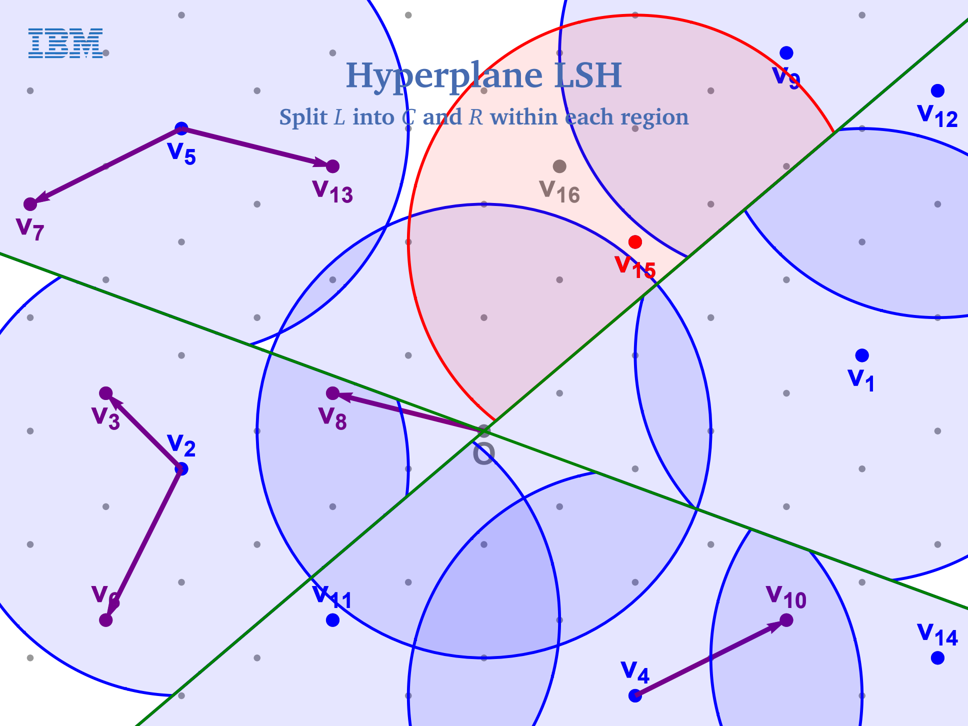
Split L into C and R within each region



Hyperplane LSH

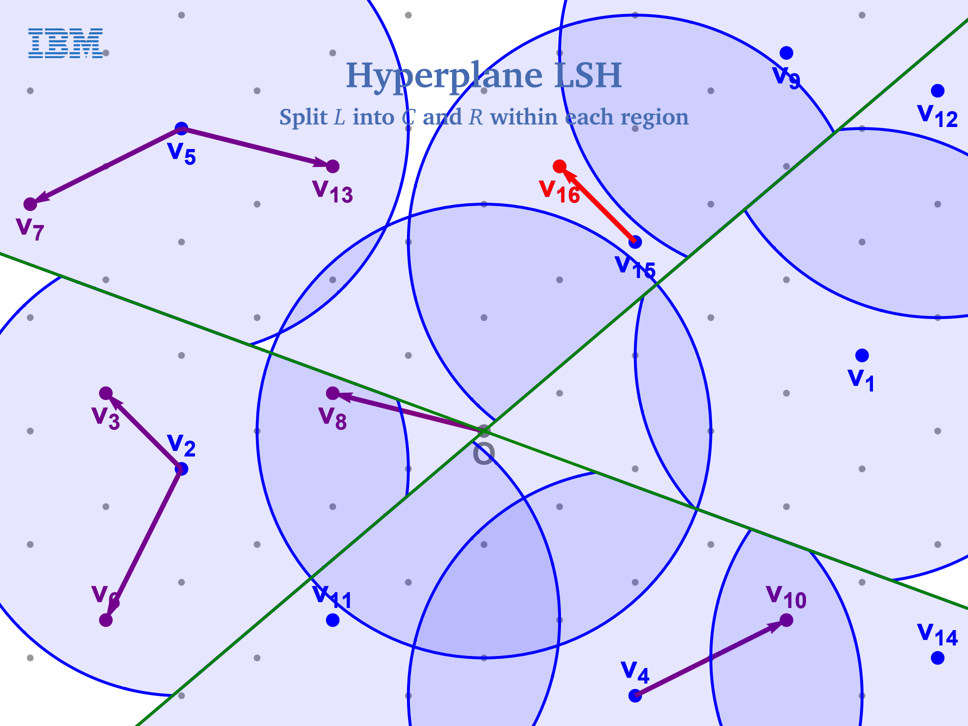
Split L into C and R within each region




Hyperplane LSH
Split L into C and R within each region
v_9
v_{12}
v_5
v_{13}
v_{16}
v_7
v_{15}
v_1
v_3
v_8
v_2
O
v_{11}
v_{10}
v_{14}
v_4

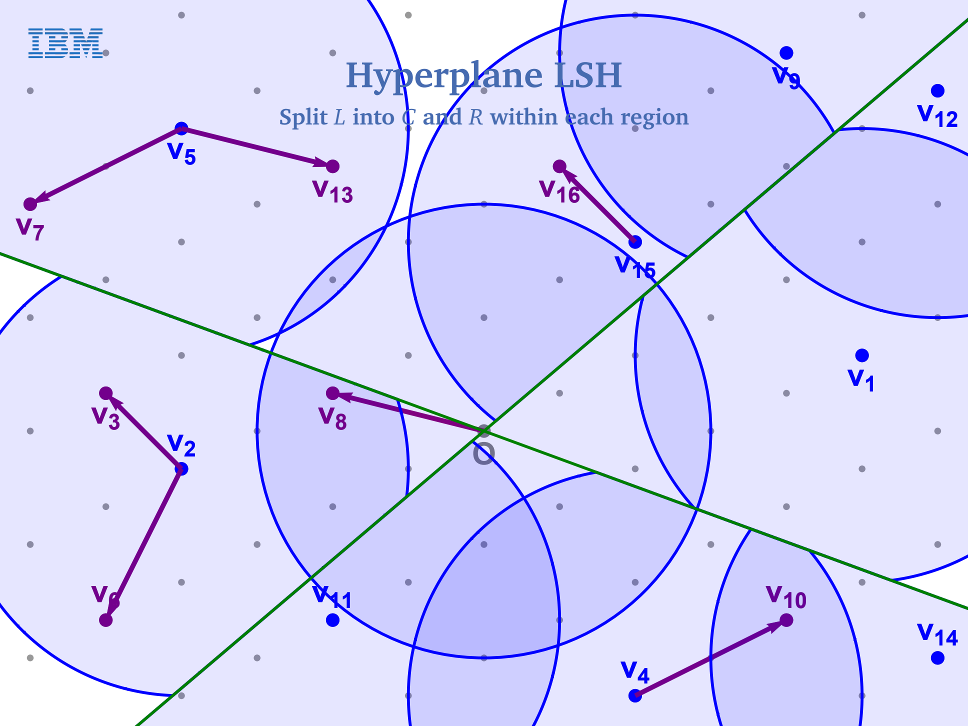
Hyperplane LSH

Split L into C and R within each region



Hyperplane LSH

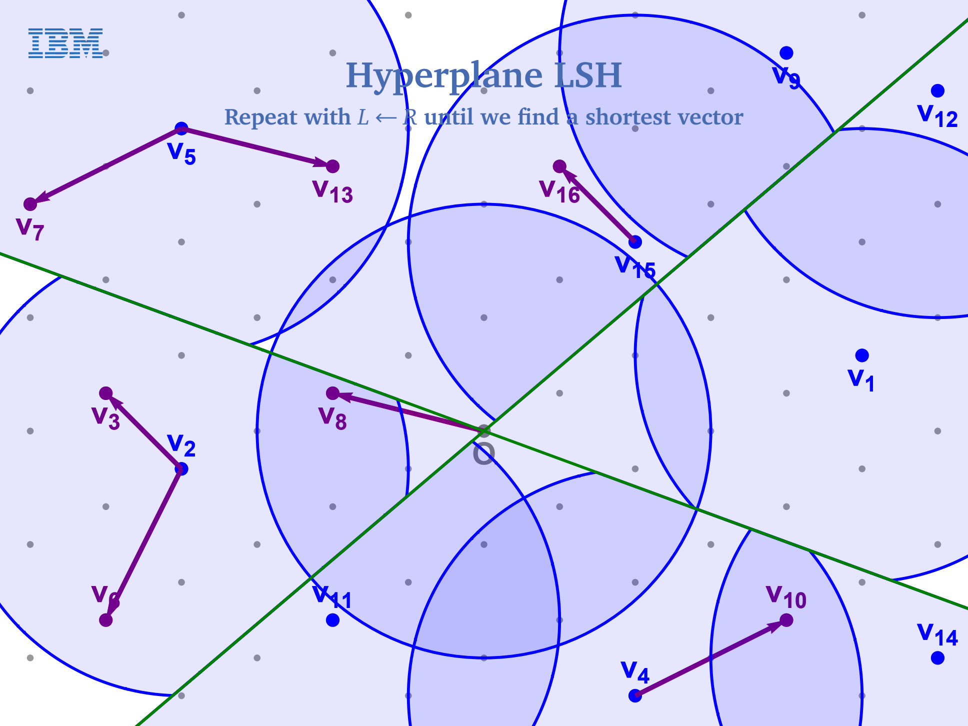
Split L into C and R within each region



IBM

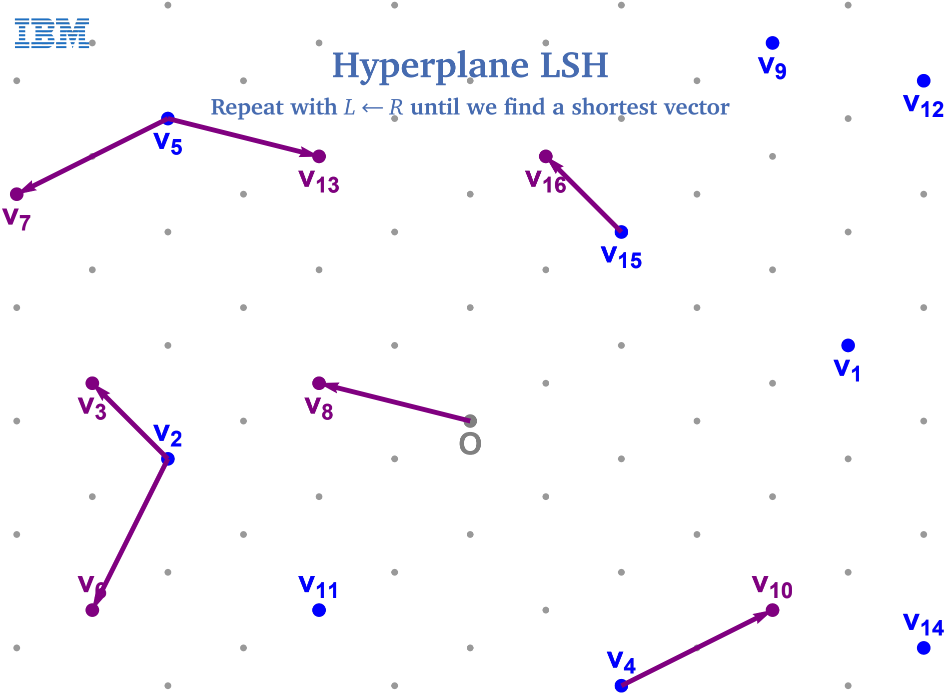
Hyperplane LSH

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



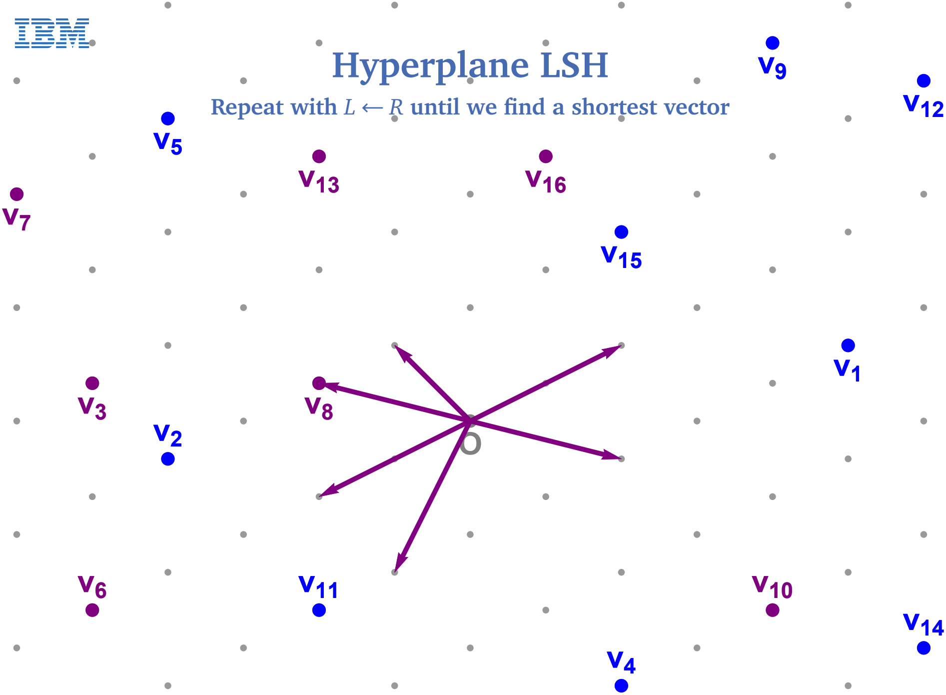
Hyperplane LSH

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



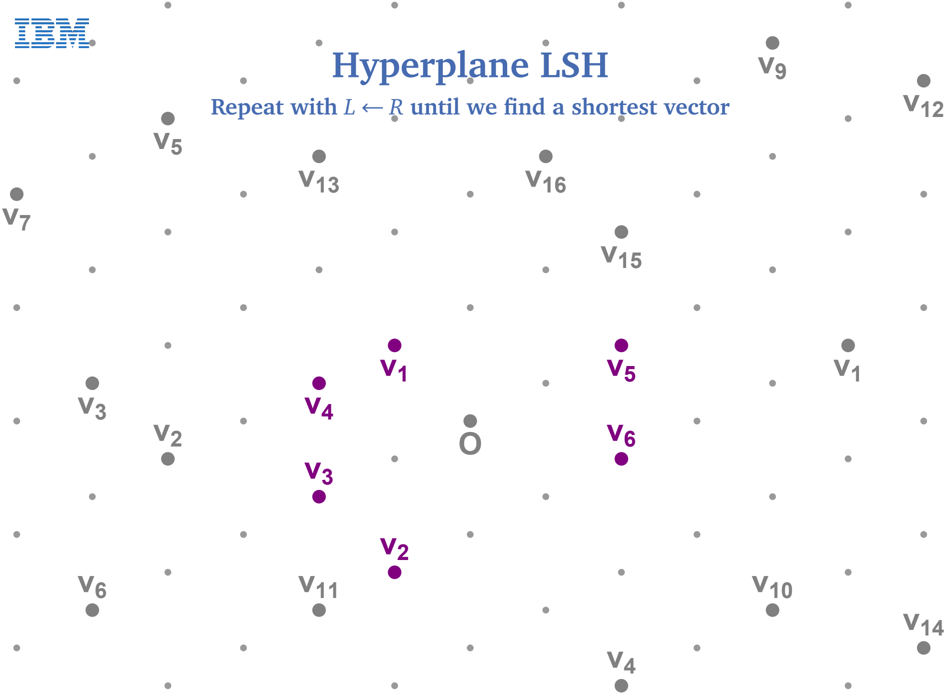
Hyperplane LSH

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



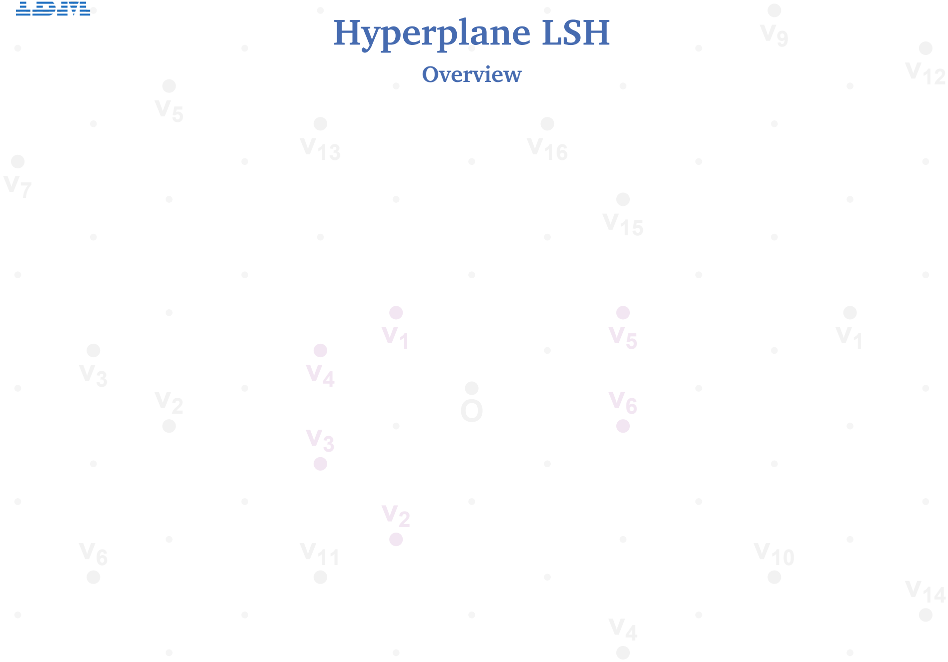
Hyperplane LSH

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



Hyperplane LSH

Overview



Hyperplane LSH

Overview

- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent “hash tables”

Hyperplane LSH

Overview

- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent "hash tables"
- Space complexity: $2^{0.337n+o(n)}$
  - Number of vectors: $2^{0.208n+o(n)}$
  - Number of hash tables: $2^{0.129n+o(n)}$
  - Each hash table contains all vectors

Hyperplane LSH

Overview

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

Hyperplane LSH

Overview

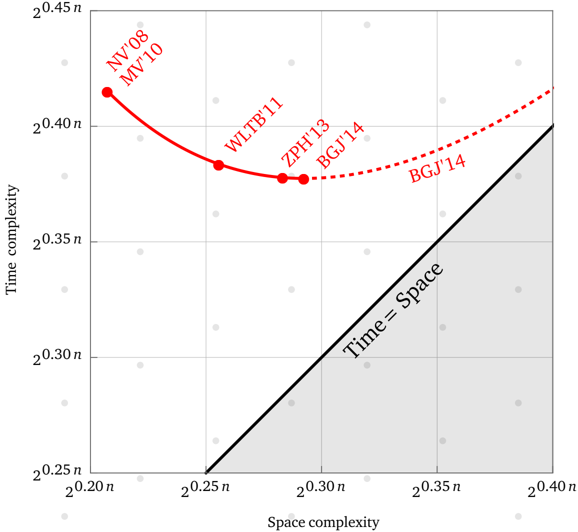
- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent "hash tables"
- Space complexity: $2^{0.337n+o(n)}$
  - Number of vectors: $2^{0.208n+o(n)}$
  - Number of hash tables: $2^{0.129n+o(n)}$
  - Each hash table contains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

Heuristic result (Laarhoven, CRYPTO'15)

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

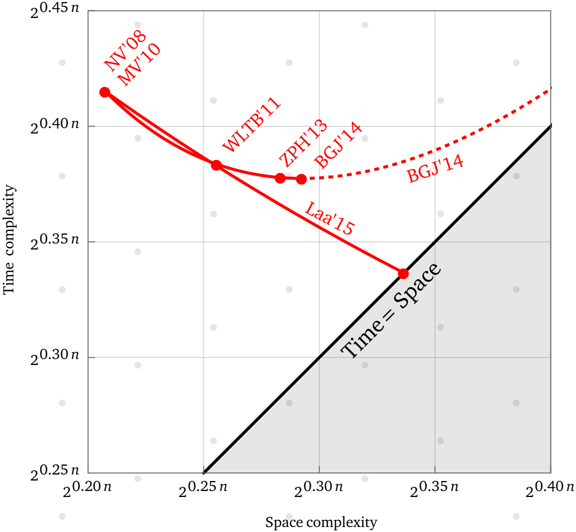
Hyperplane LSH

Space/time trade-off



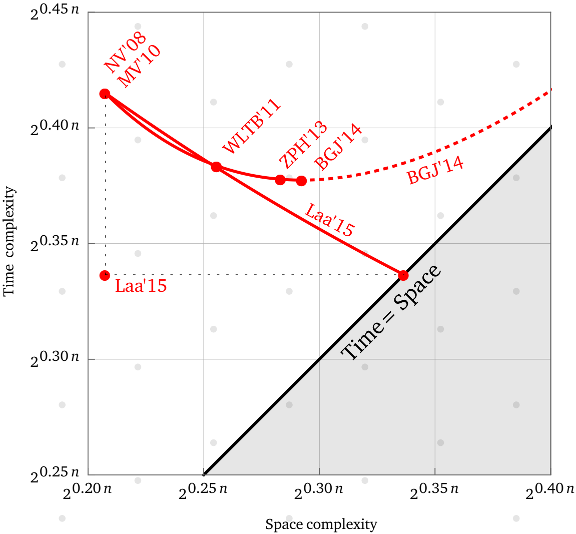
Hyperplane LSH

Space/time trade-off



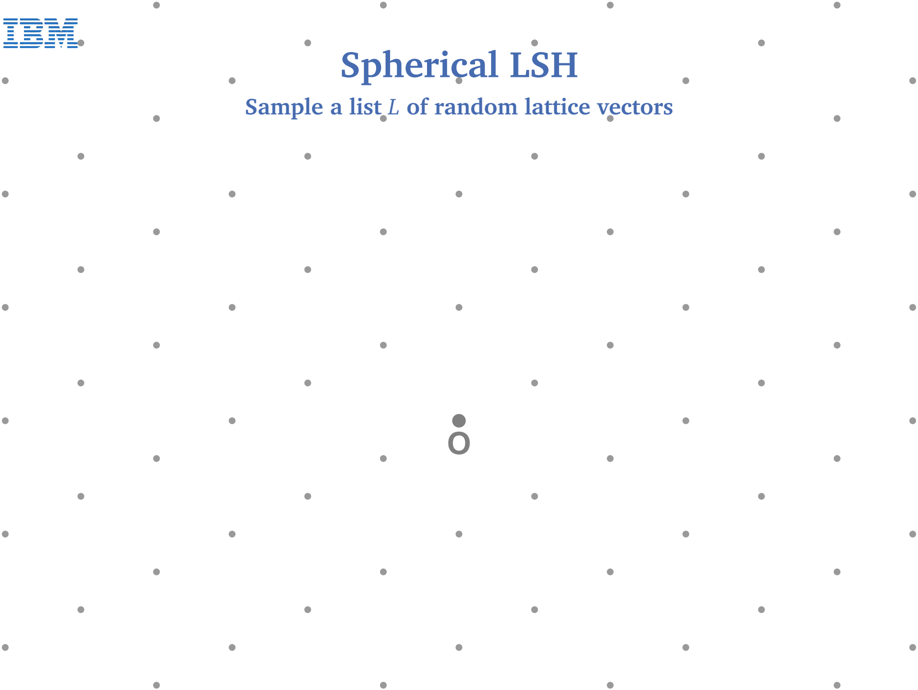
Hyperplane LSH

Space/time trade-off



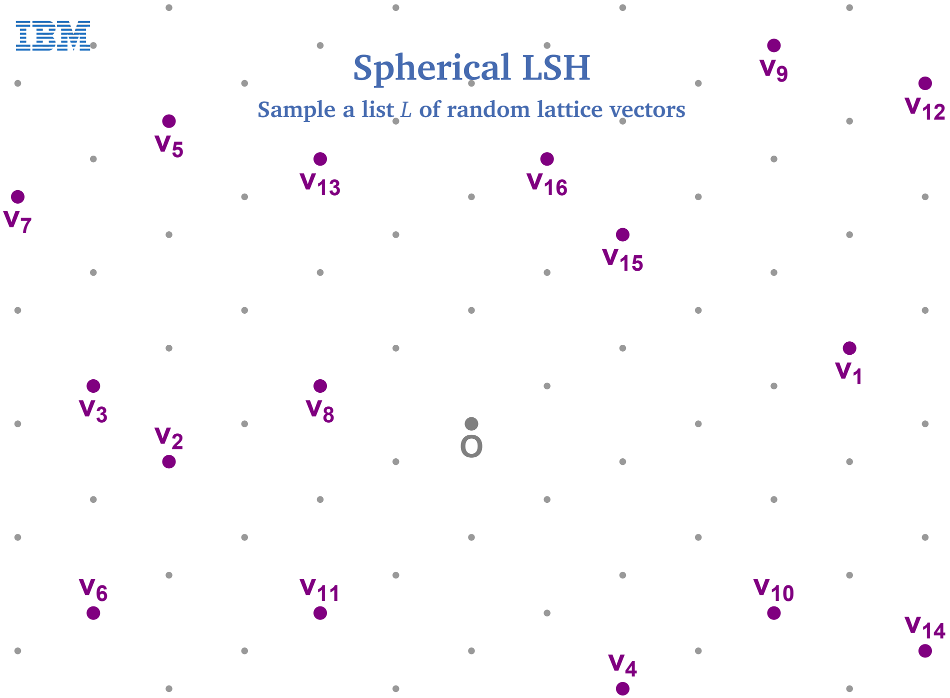
Spherical LSH

Sample a list L of random lattice vectors



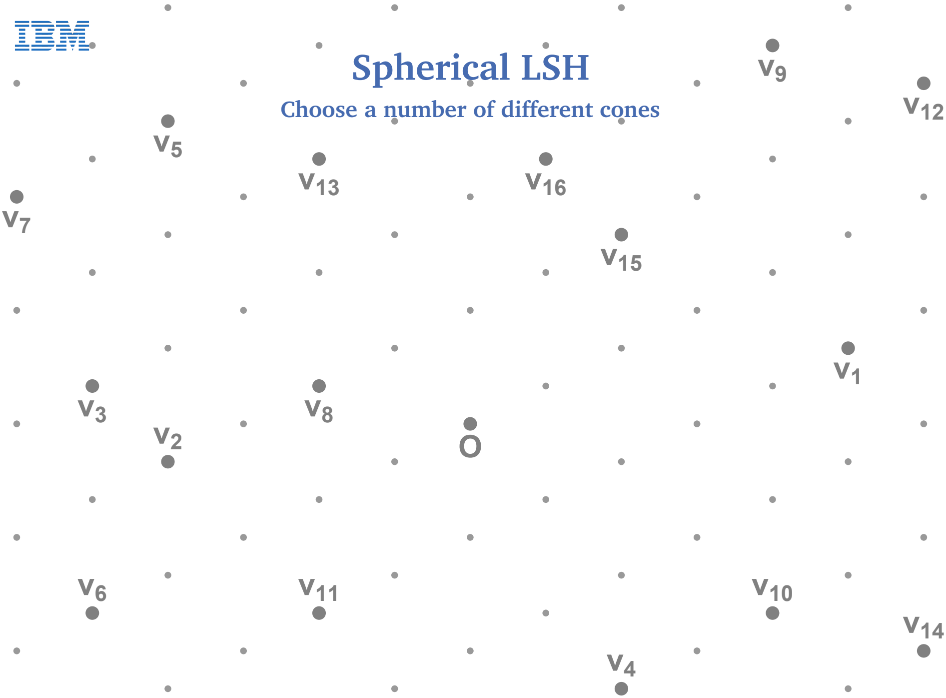
Spherical LSH

Sample a list L of random lattice vectors



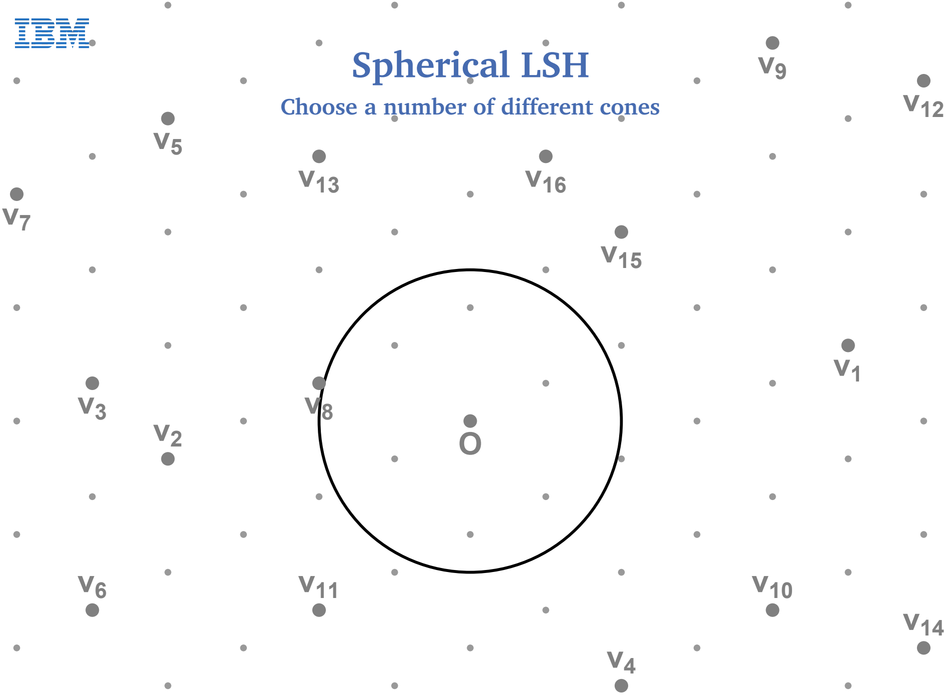
Spherical LSH

Choose a number of different cones



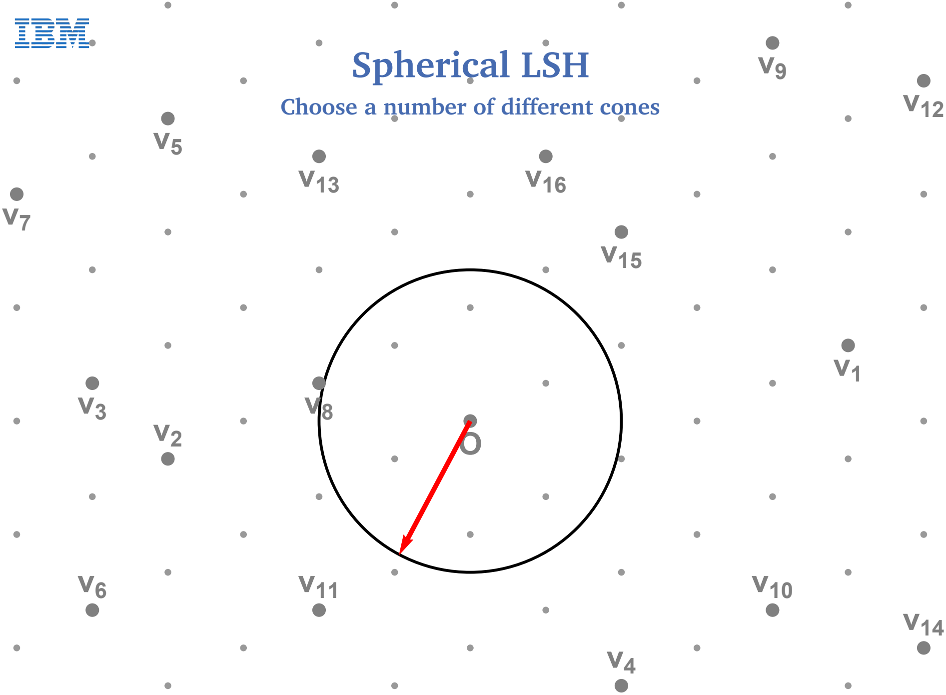
Spherical LSH

Choose a number of different cones



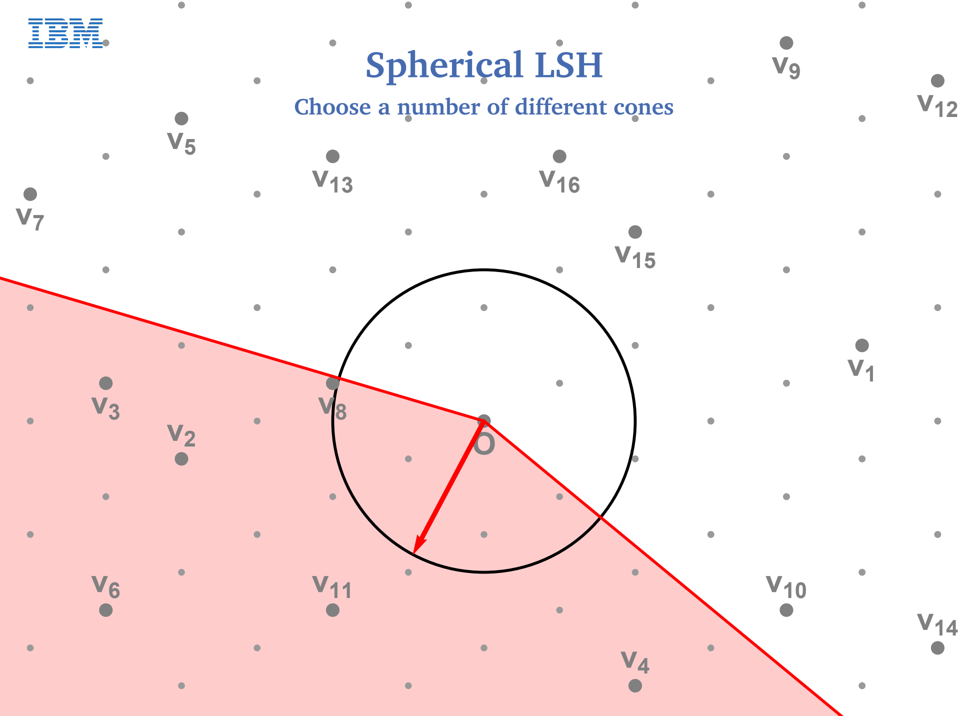
Spherical LSH

Choose a number of different cones



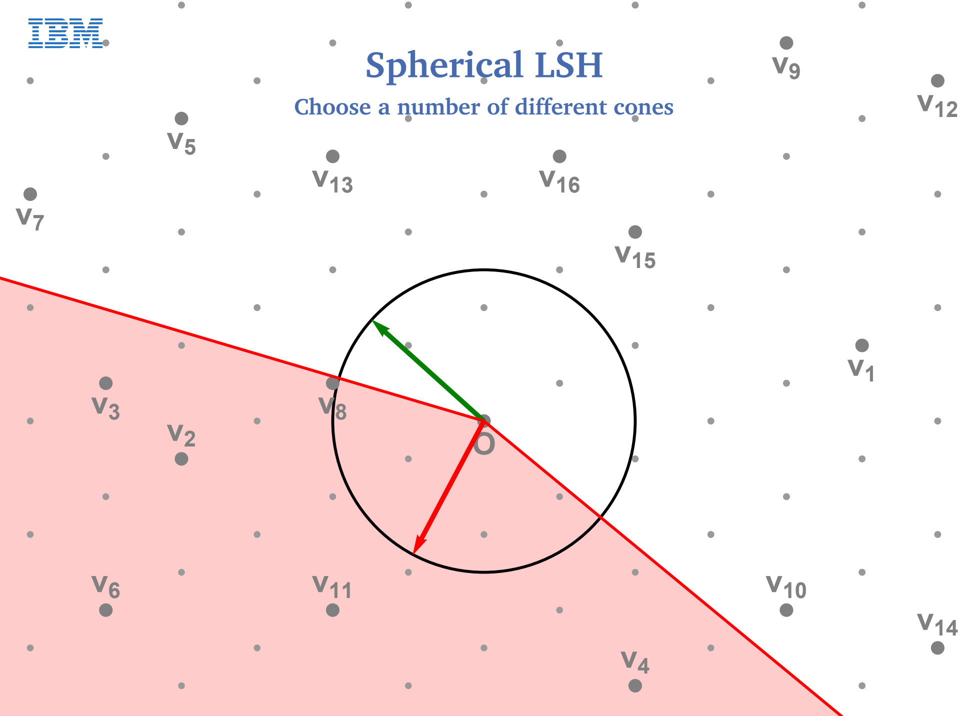
Spherical LSH

Choose a number of different cones



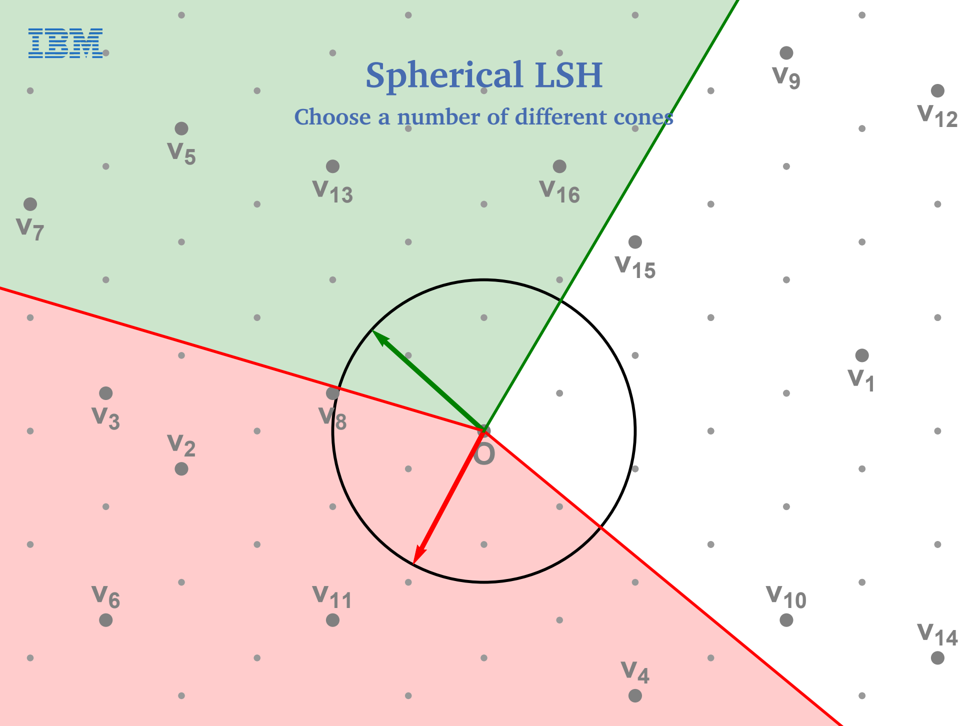
Spherical LSH

Choose a number of different cones



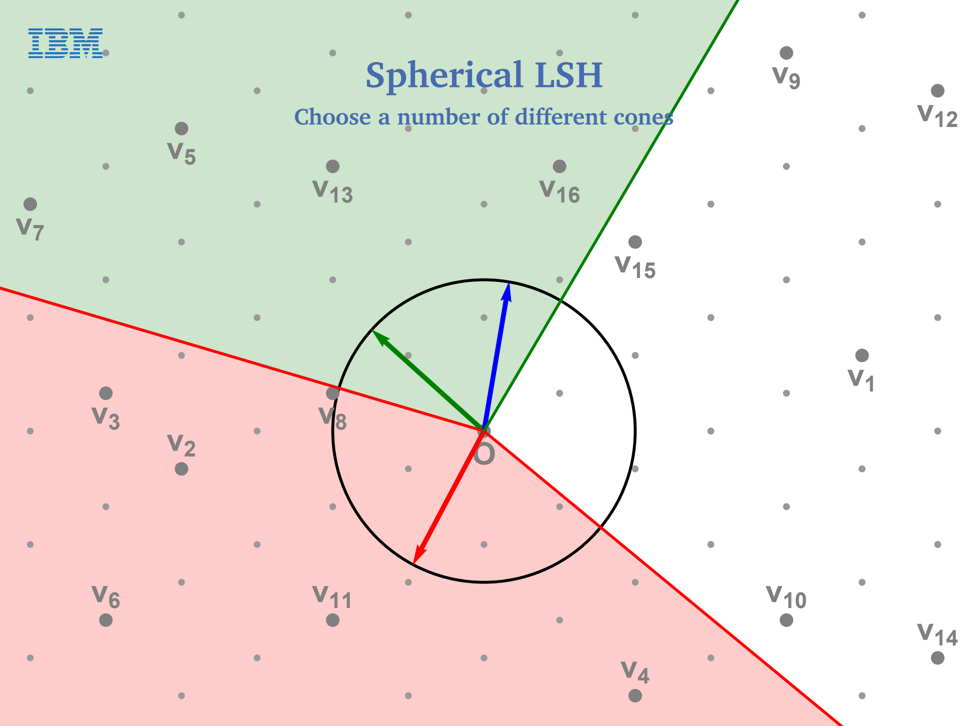
Spherical LSH

Choose a number of different cones



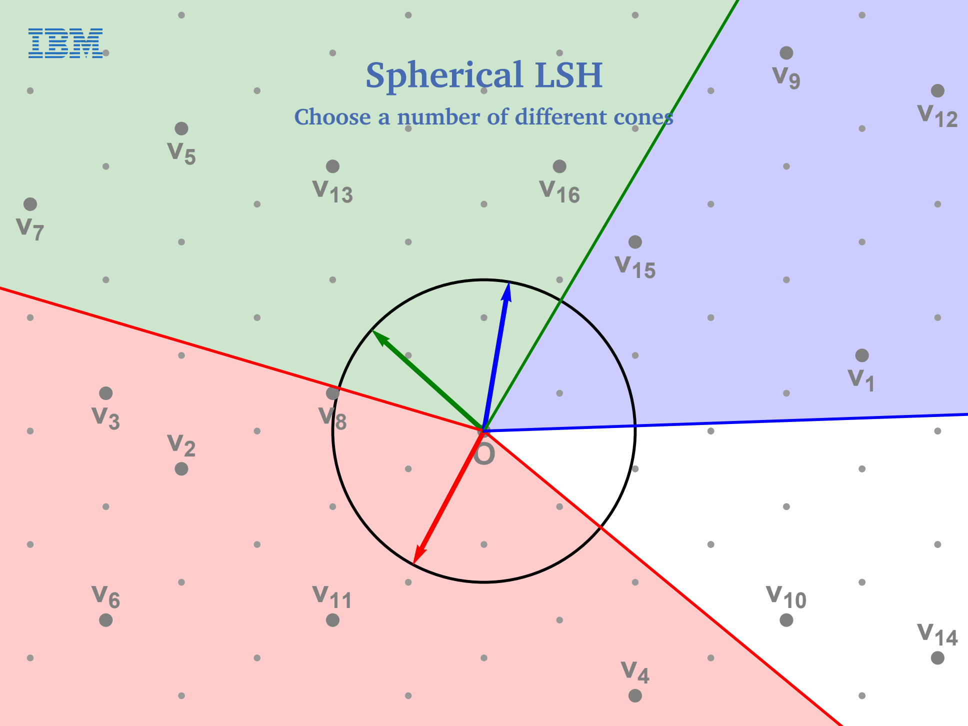
Spherical LSH

Choose a number of different cones



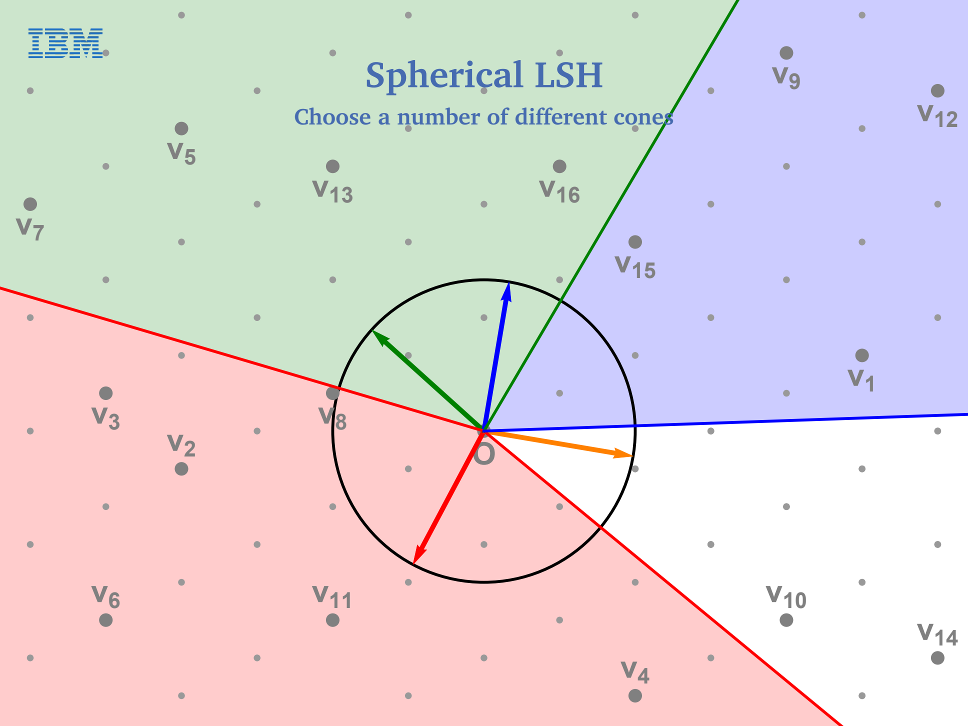
Spherical LSH

Choose a number of different cones



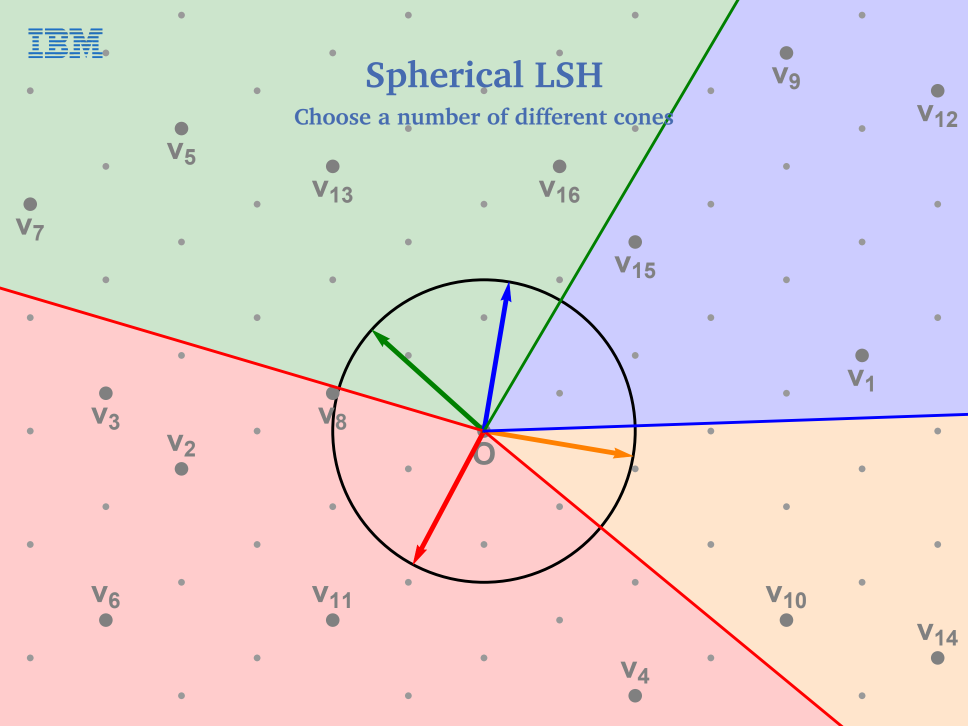
Spherical LSH

Choose a number of different cones



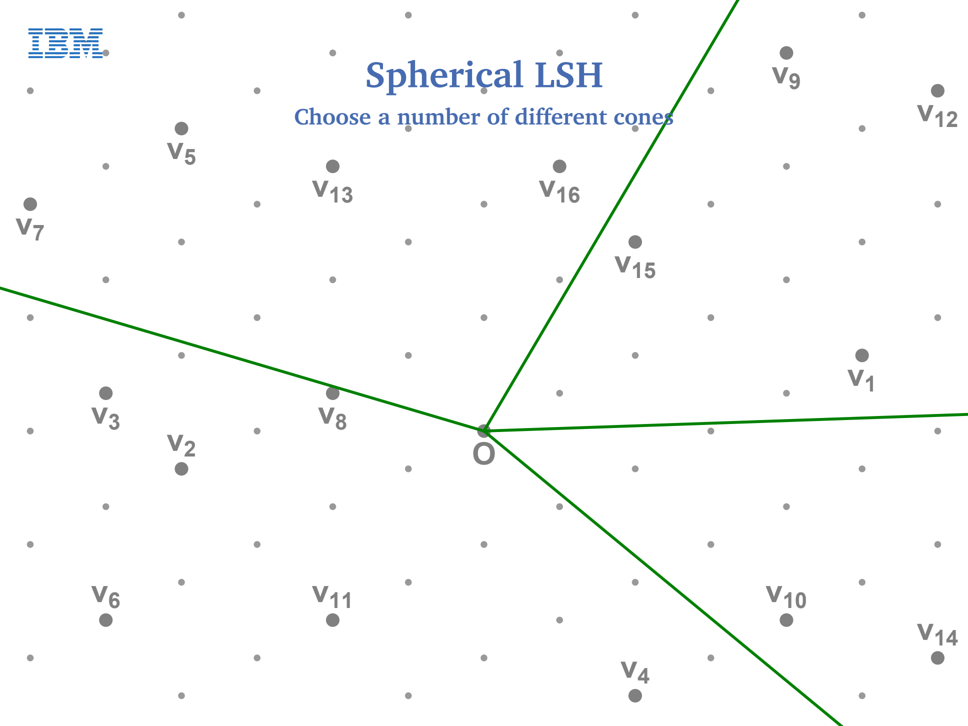
Spherical LSH

Choose a number of different cones



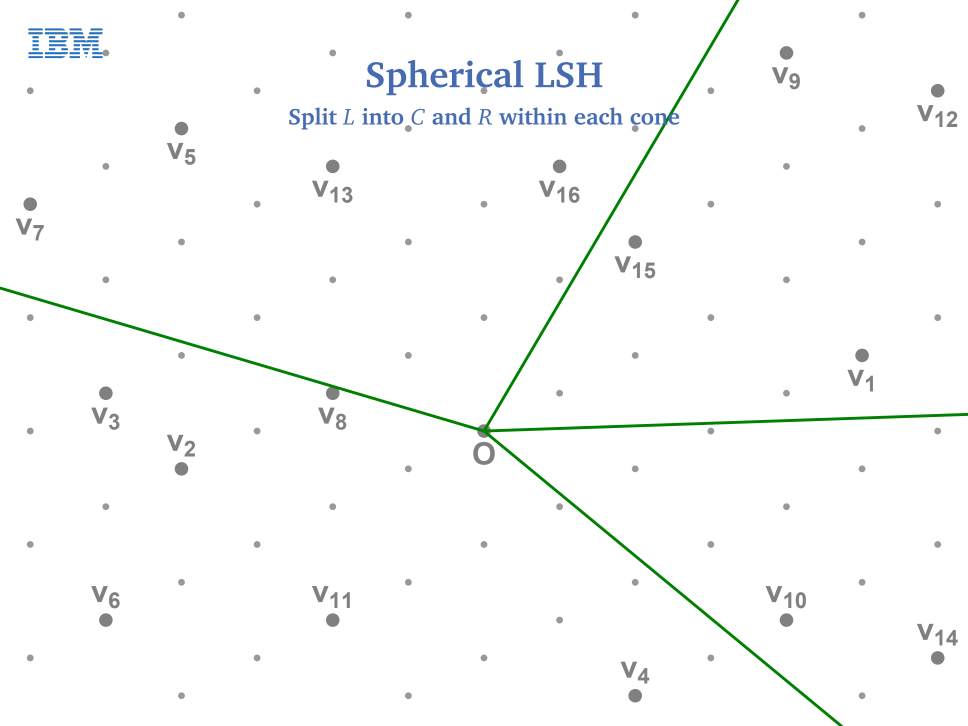
Spherical LSH

Choose a number of different cones



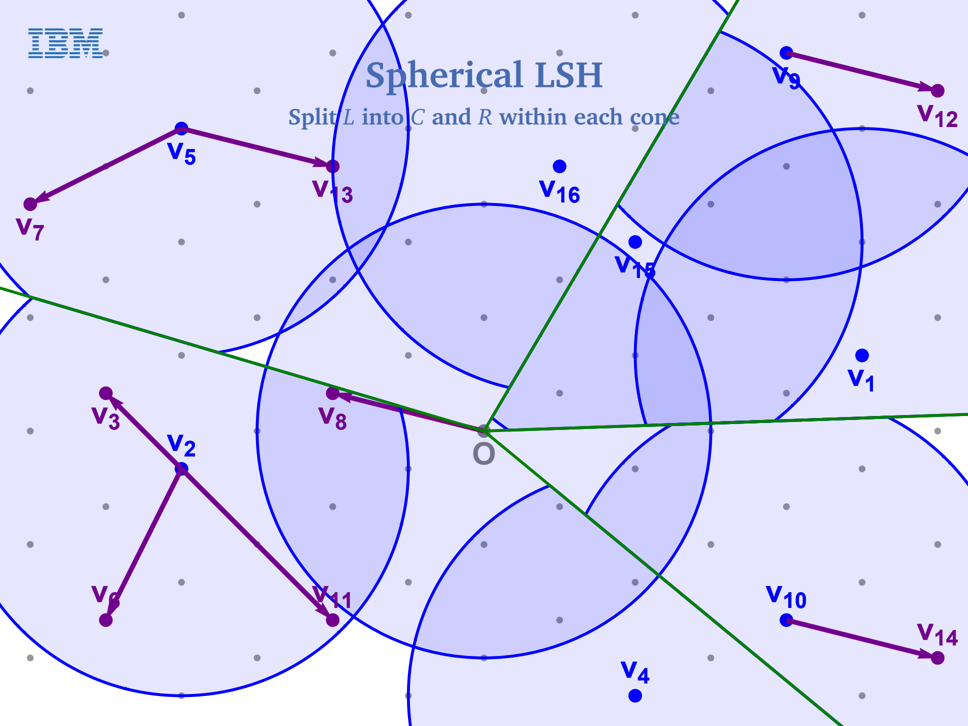
Spherical LSH

Split L into C and R within each cone



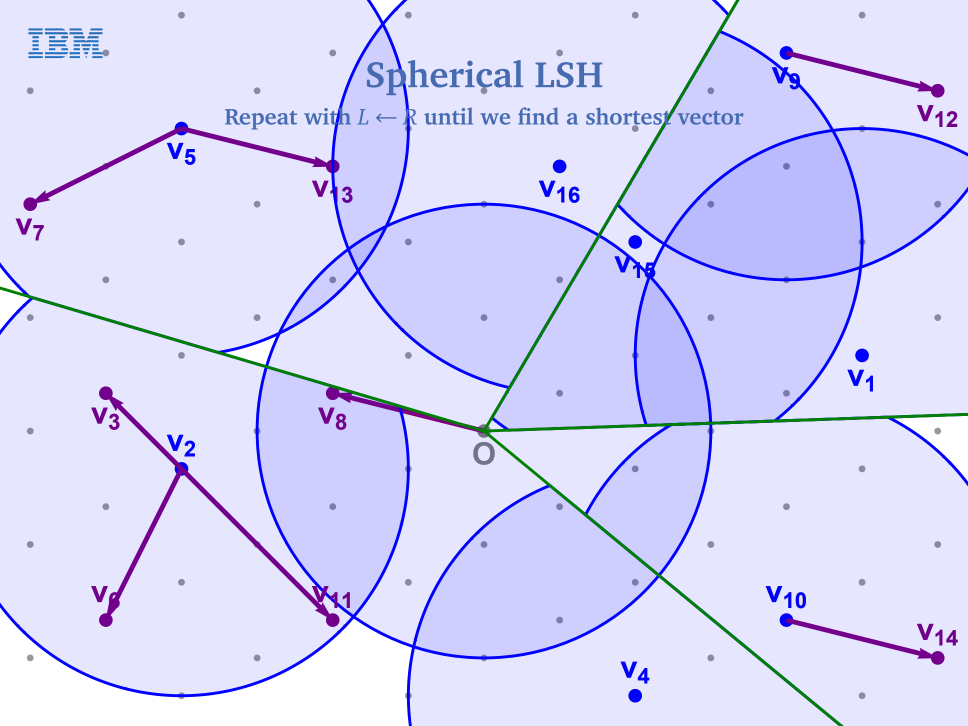
Spherical LSH

Split L into C and R within each cone



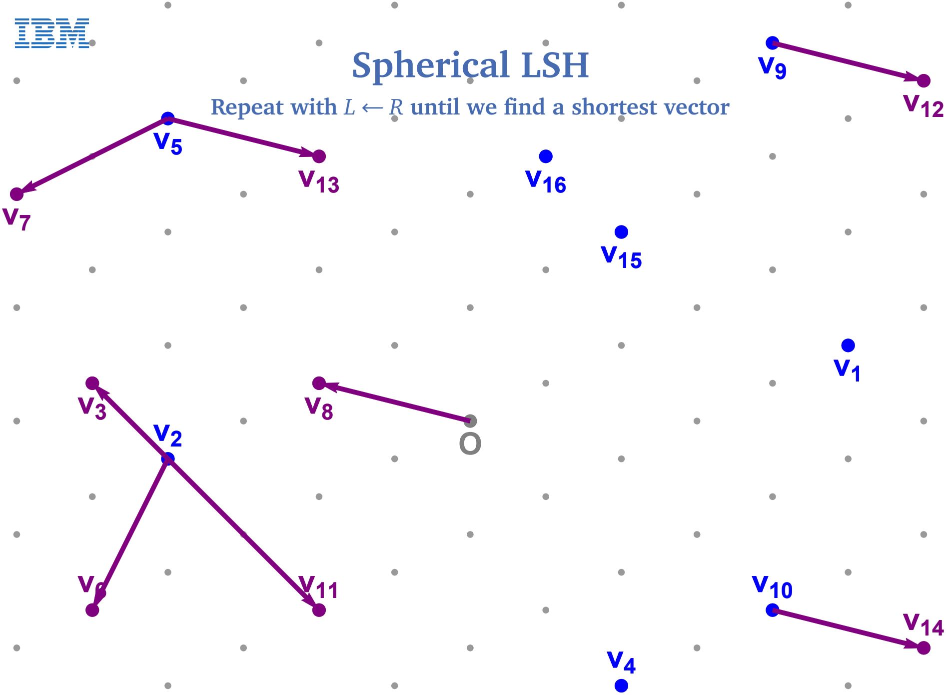
Spherical LSH

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



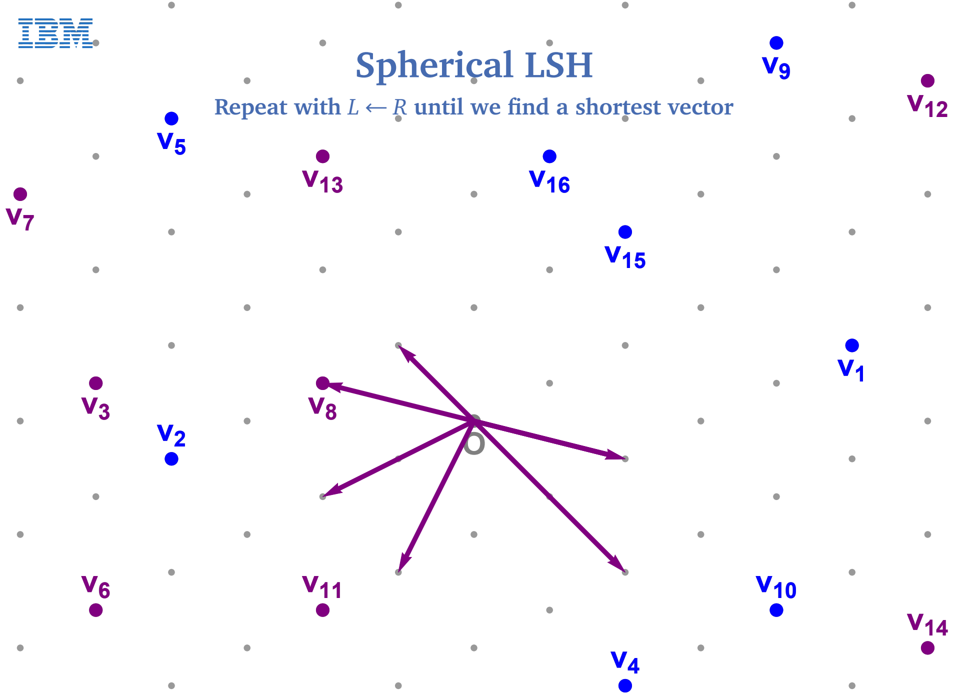
Spherical LSH

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



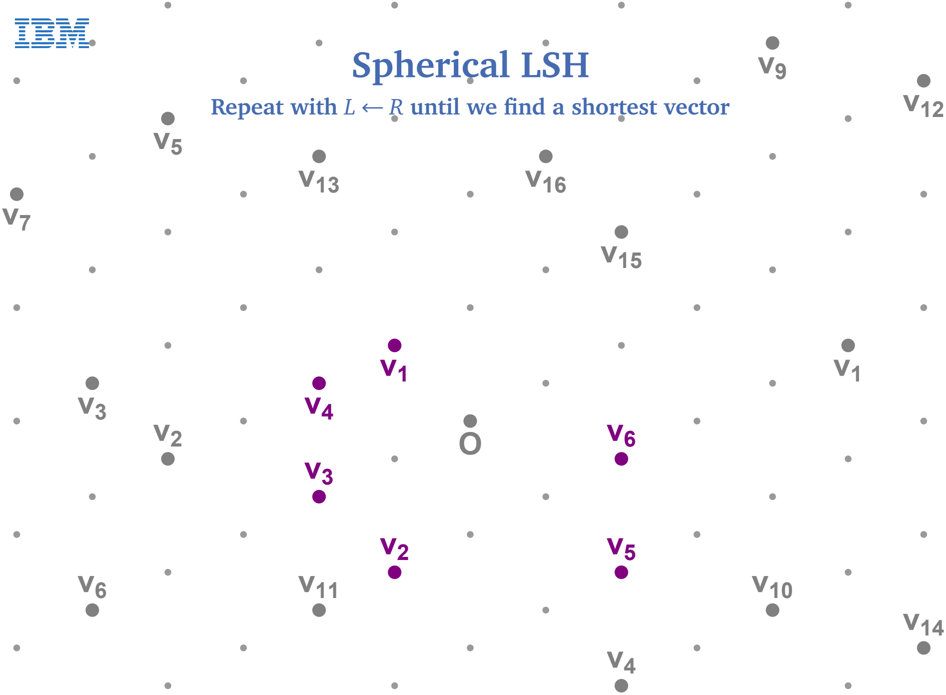
Spherical LSH

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



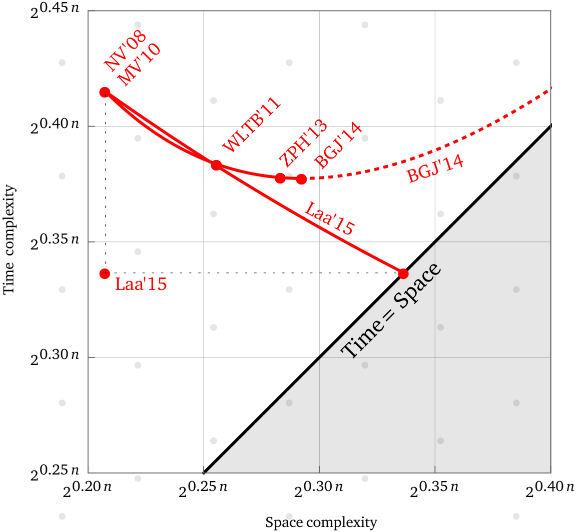
Spherical LSH

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



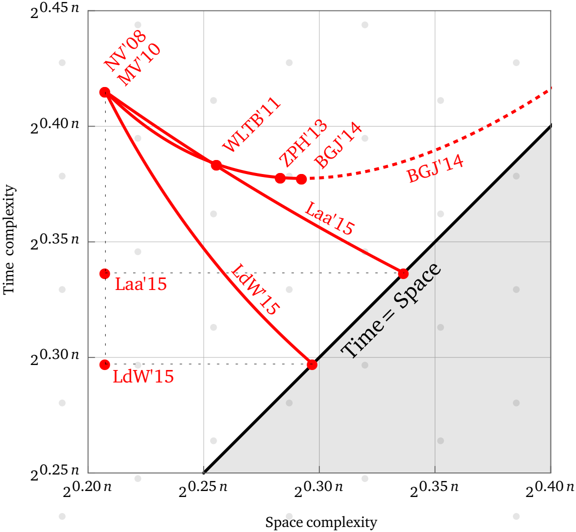
Spherical LSH

Space/time trade-off



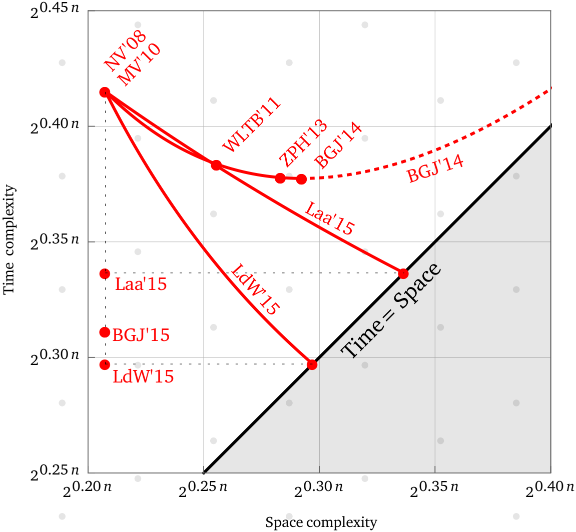
Spherical LSH

Space/time trade-off



May and Ozerov's NNS method

Space/time trade-off

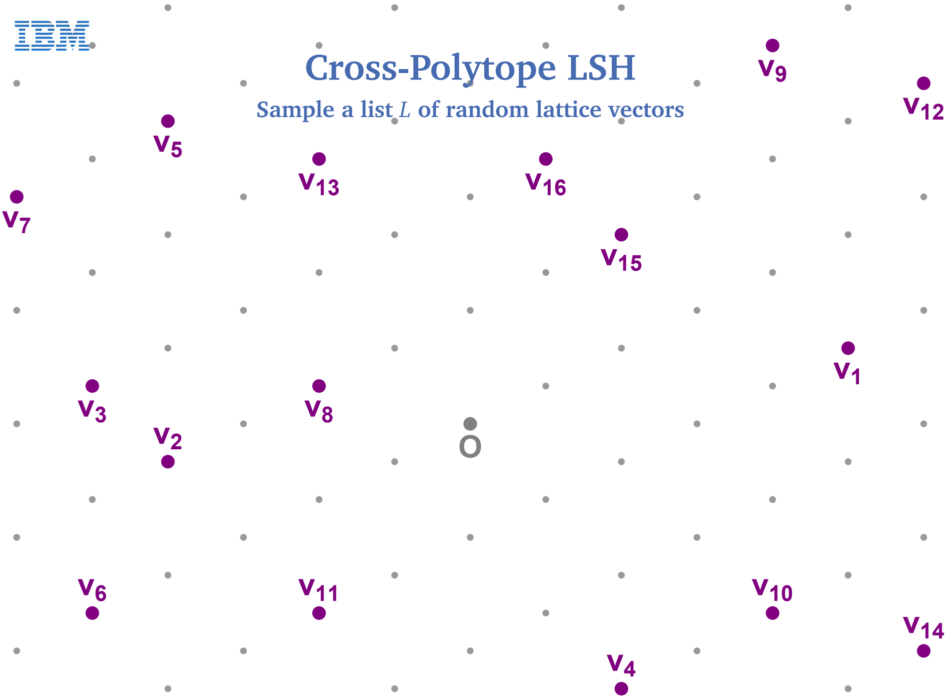


Cross-Polytope LSH

Sample a list L of random lattice vectors

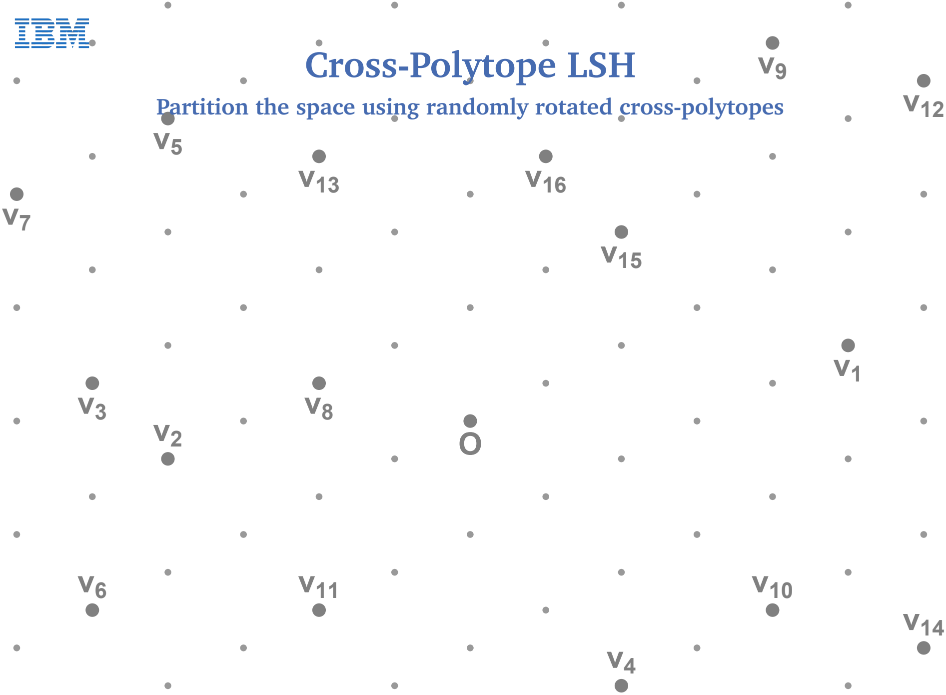
Cross-Polytope LSH

Sample a list L of random lattice vectors



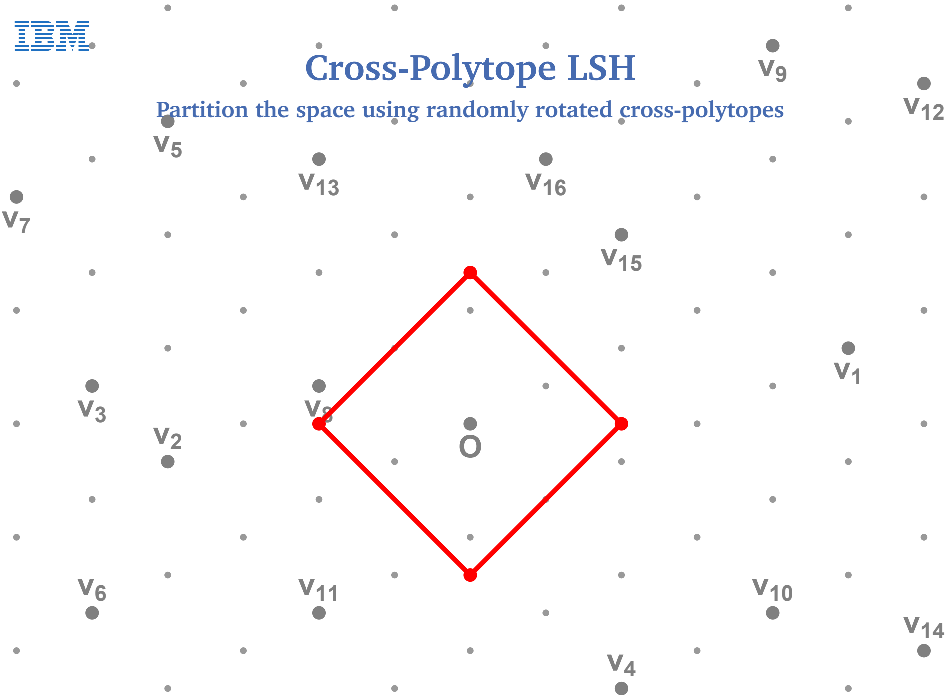
Cross-Polytope LSH

Partition the space using randomly rotated cross-polytopes



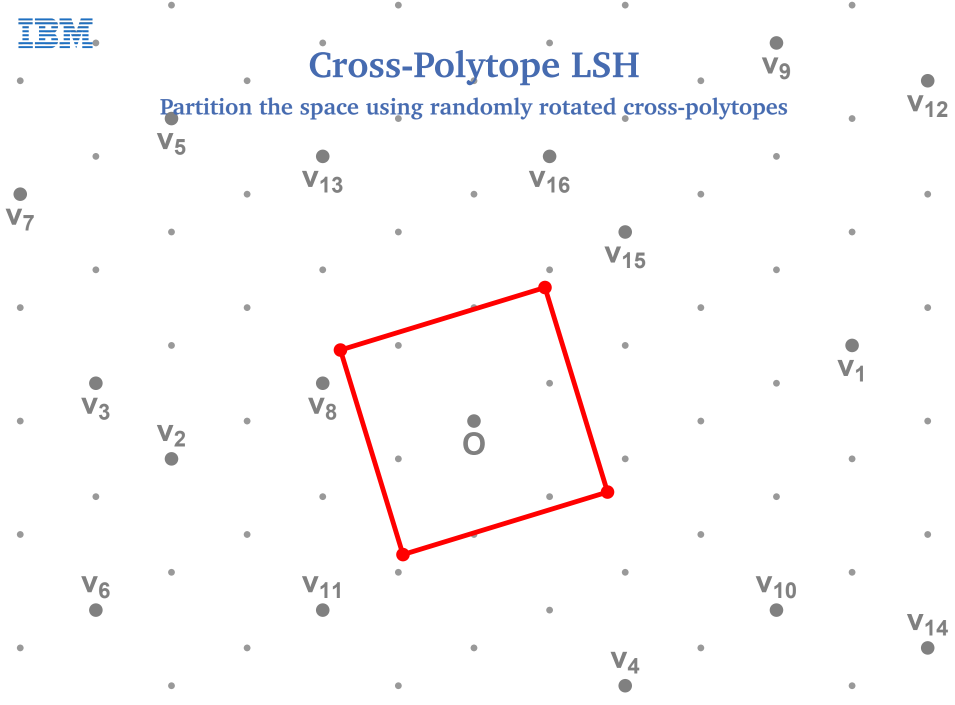
Cross-Polytope LSH

Partition the space using randomly rotated cross-polytopes



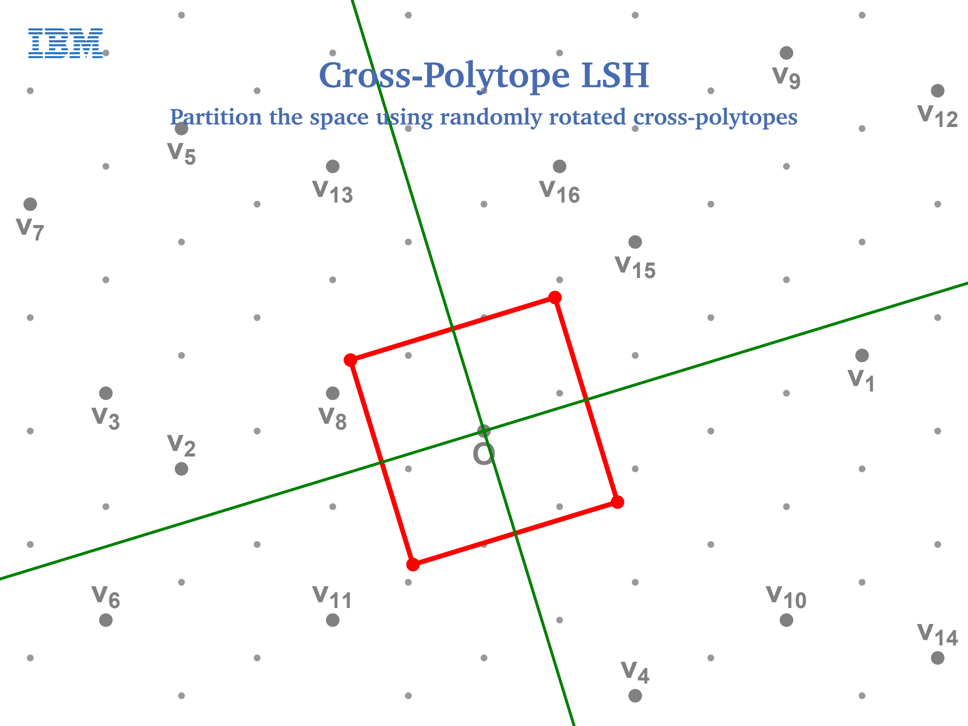
Cross-Polytope LSH

Partition the space using randomly rotated cross-polytopes



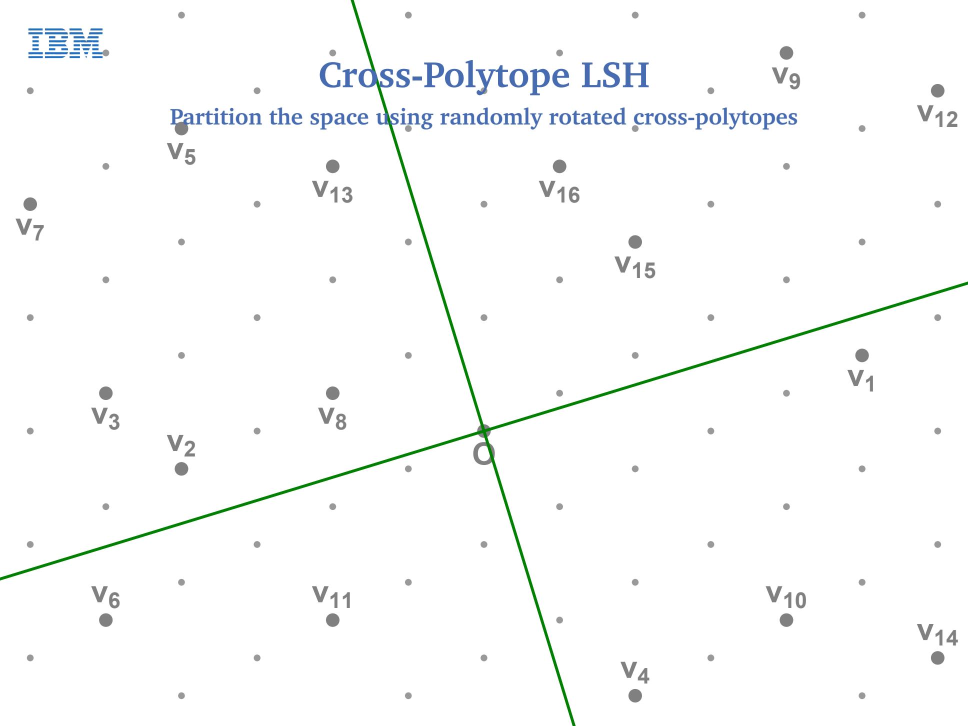
Cross-Polytope LSH

Partition the space using randomly rotated cross-polytopes



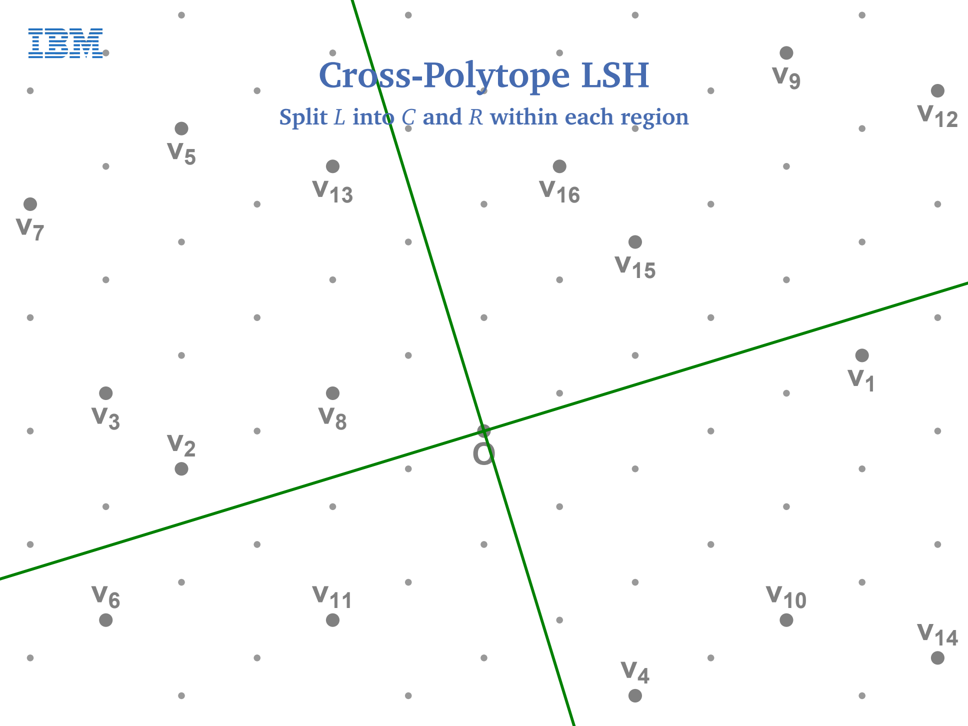
Cross-Polytope LSH

Partition the space using randomly rotated cross-polytopes



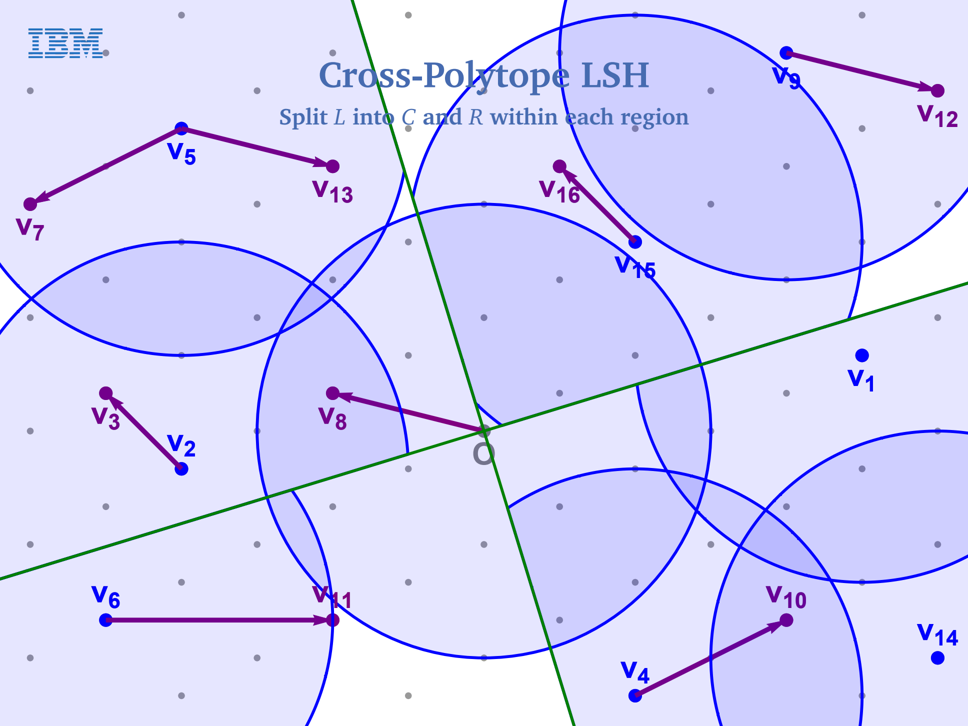
Cross-Polytope LSH

Split L into C and R within each region



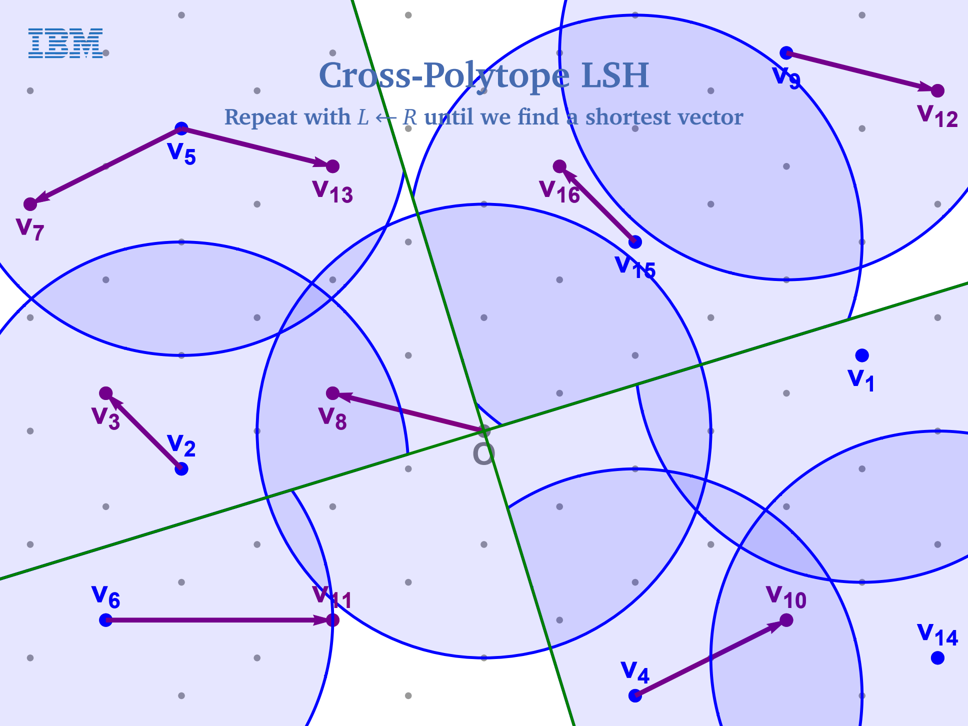
Cross-Polytope LSH

Split L into C and R within each region



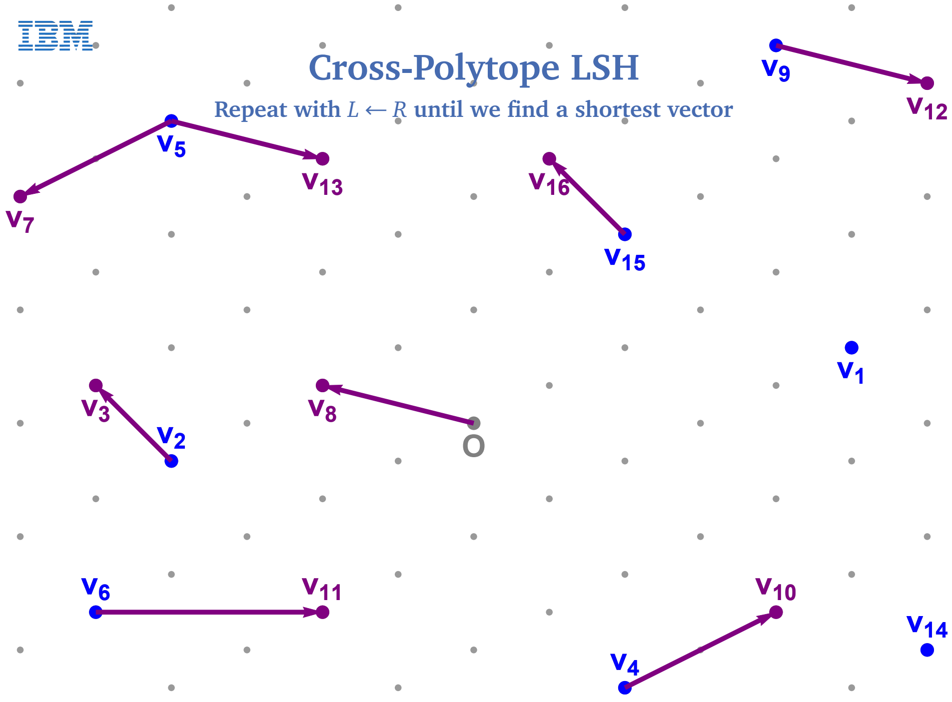
Cross-Polytope LSH

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



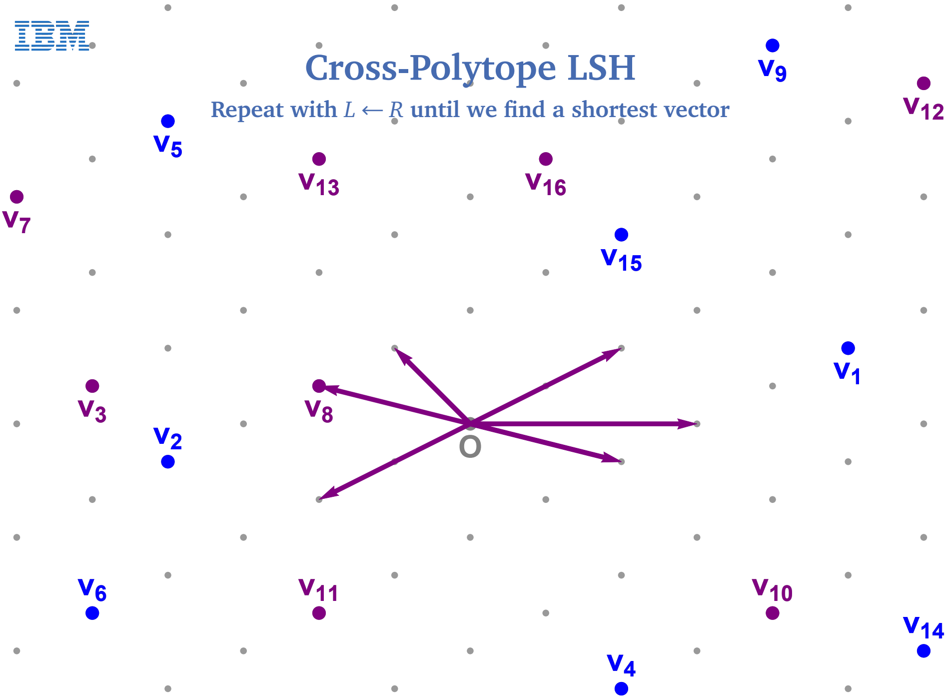
Cross-Polytope LSH

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



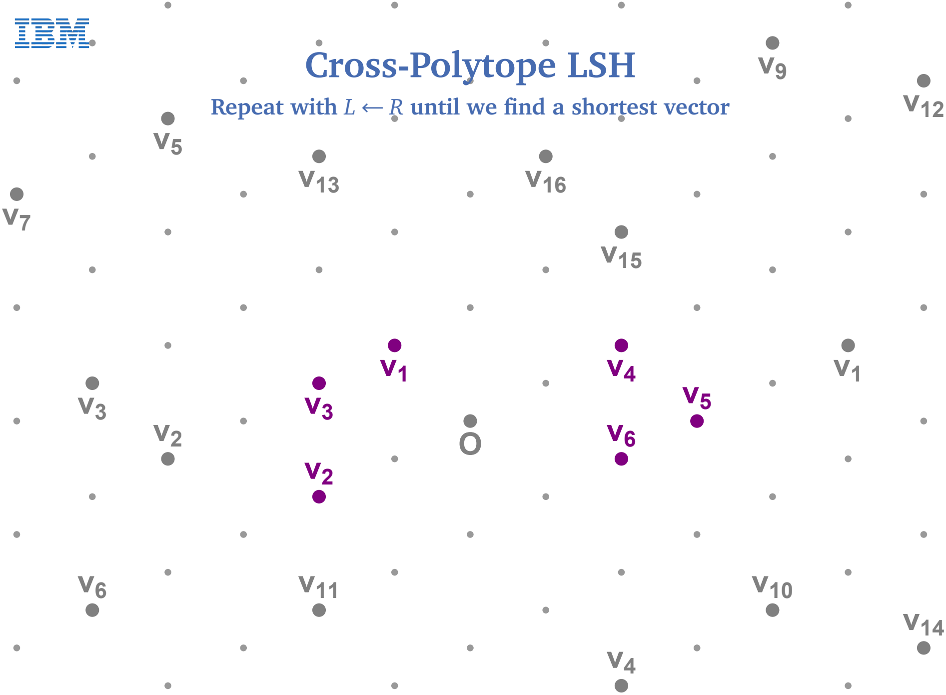
Cross-Polytope LSH

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



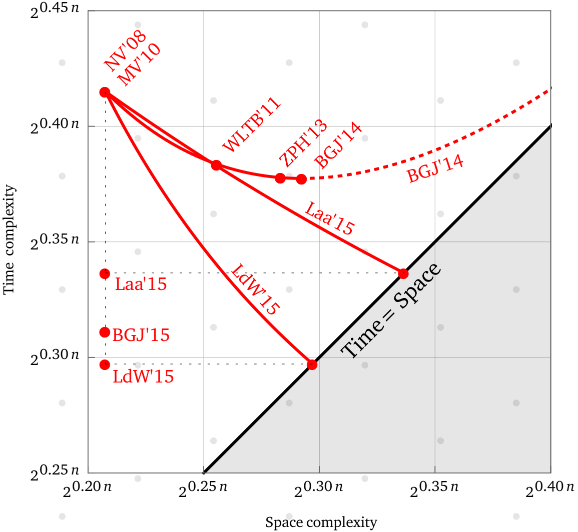
Cross-Polytope LSH

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



Cross-Polytope LSH

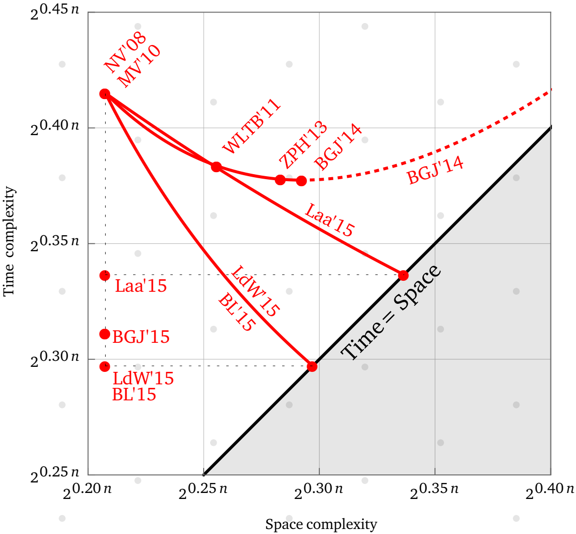
Space/time trade-off



IBM

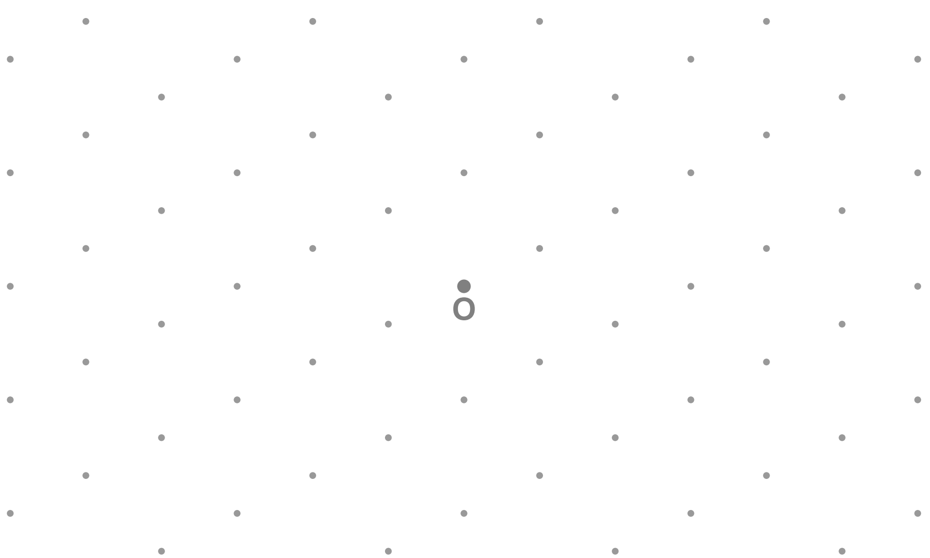
Cross-Polytope LSH

Space/time trade-off



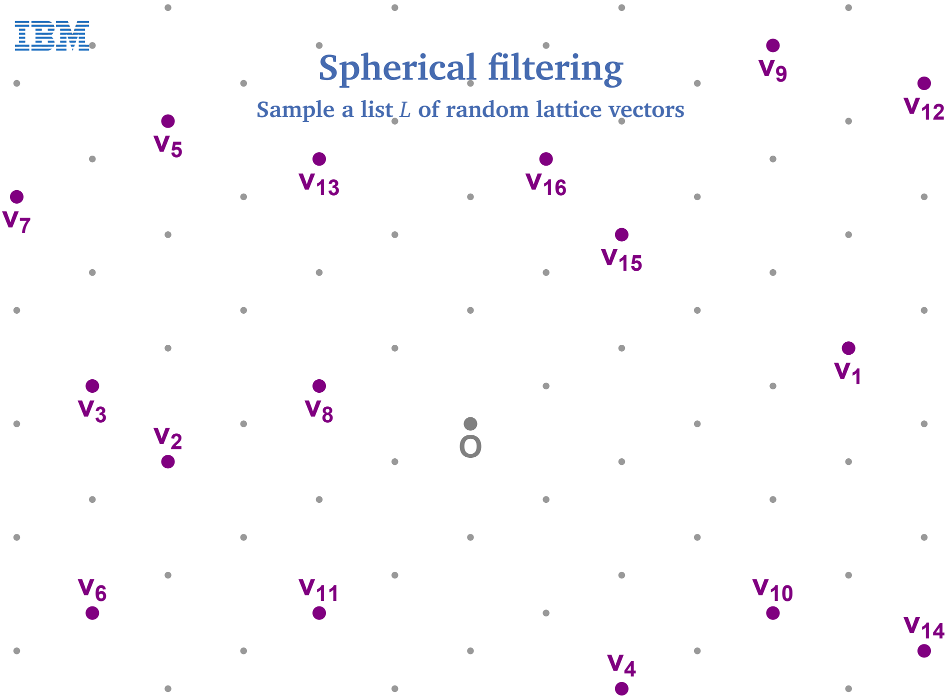
Spherical filtering

Sample a list L of random lattice vectors



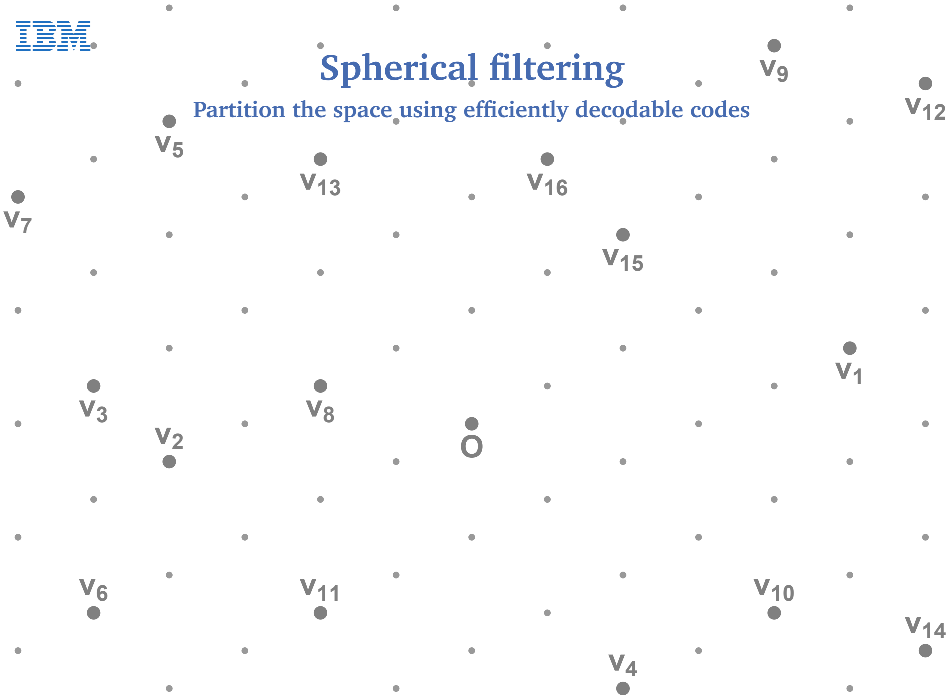
Spherical filtering

Sample a list L of random lattice vectors



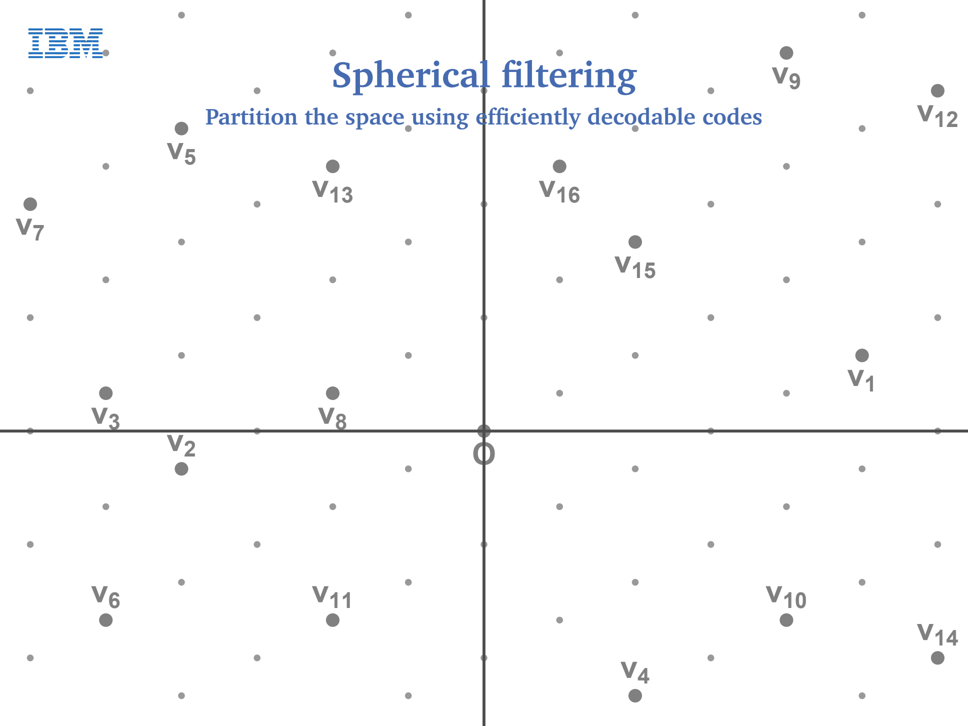
Spherical filtering

Partition the space using efficiently decodable codes



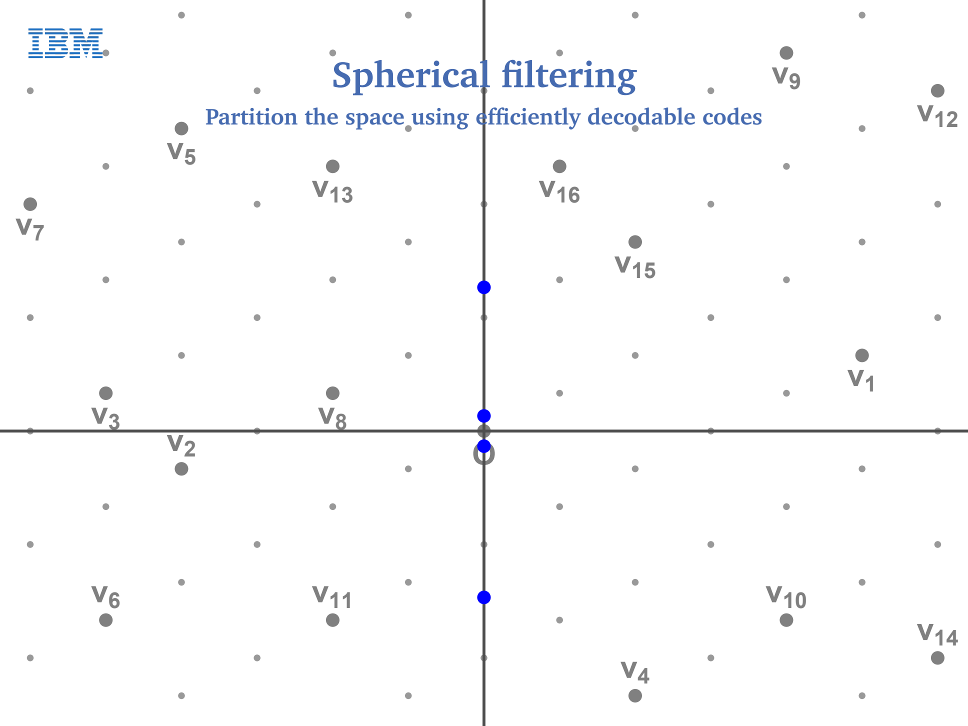
Spherical filtering

Partition the space using efficiently decodable codes



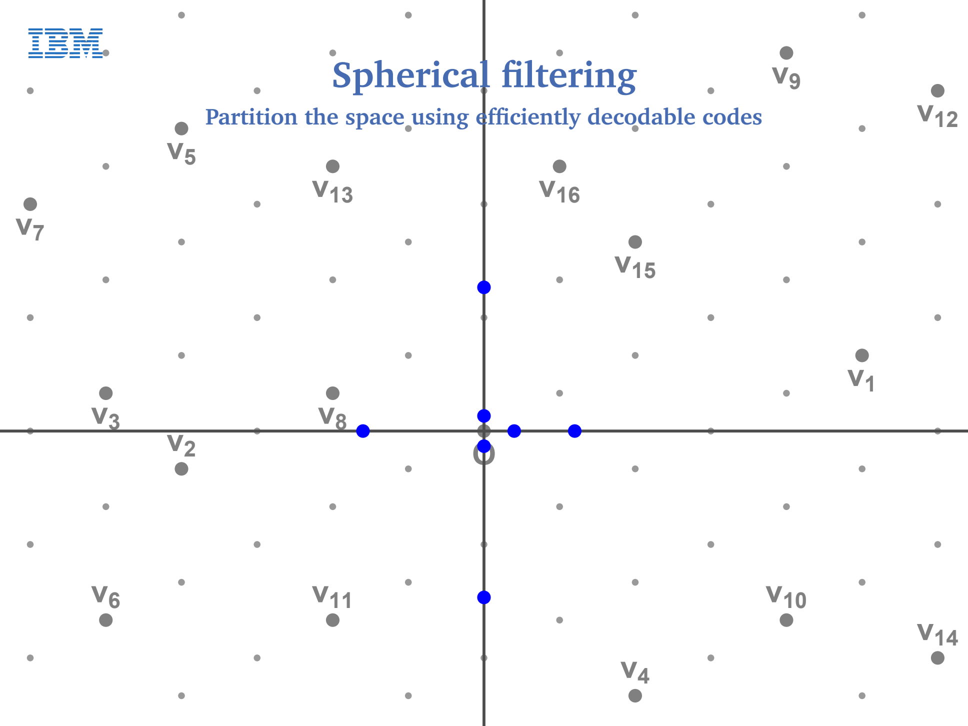
Spherical filtering

Partition the space using efficiently decodable codes



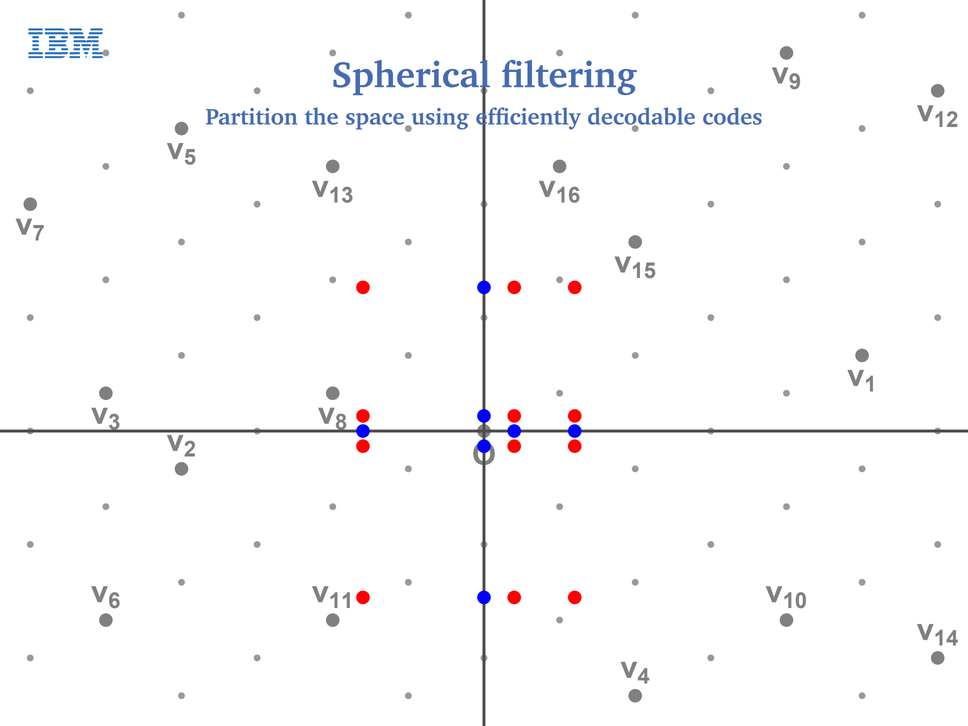
Spherical filtering

Partition the space using efficiently decodable codes



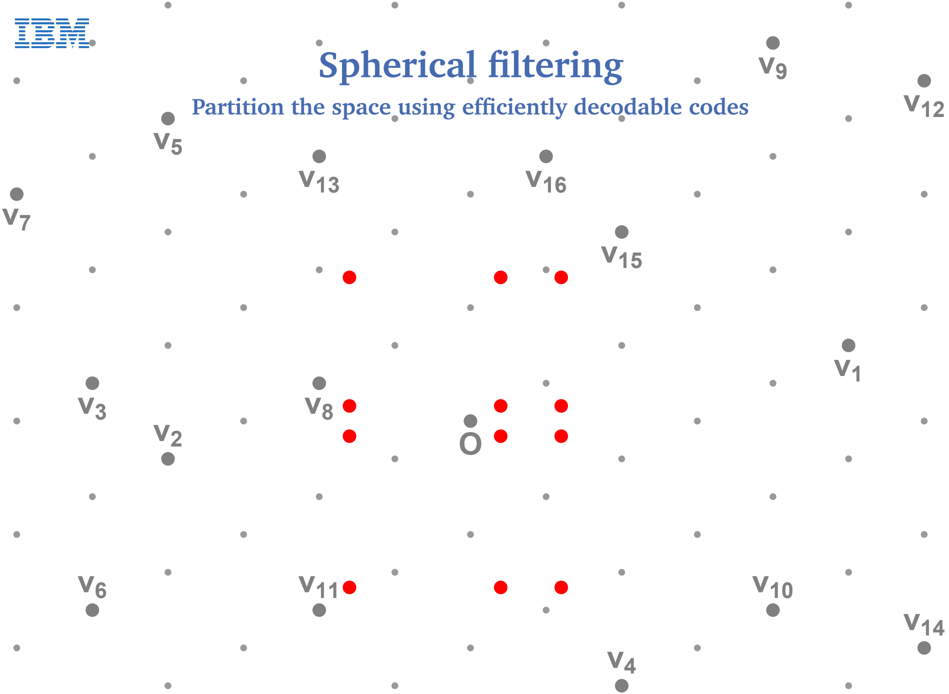
Spherical filtering

Partition the space using efficiently decodable codes



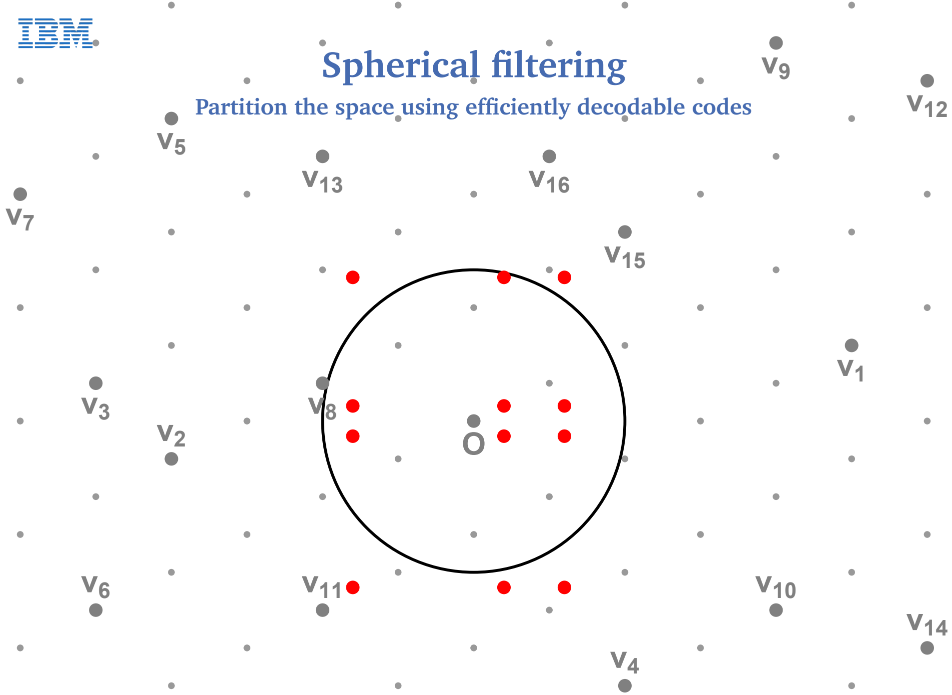
Spherical filtering

Partition the space using efficiently decodable codes



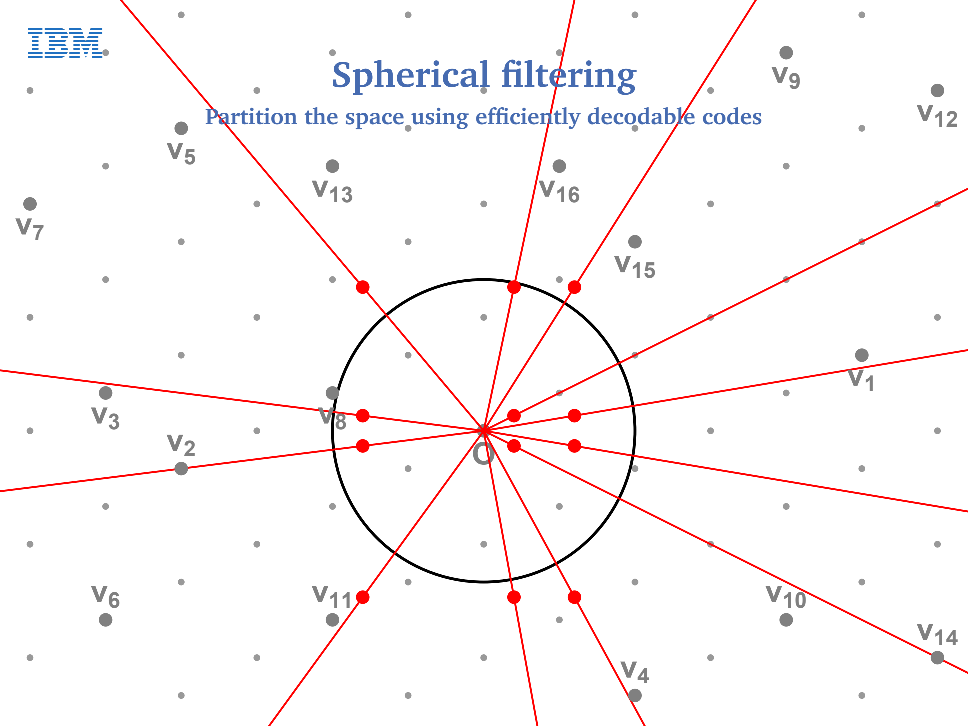
Spherical filtering

Partition the space using efficiently decodable codes



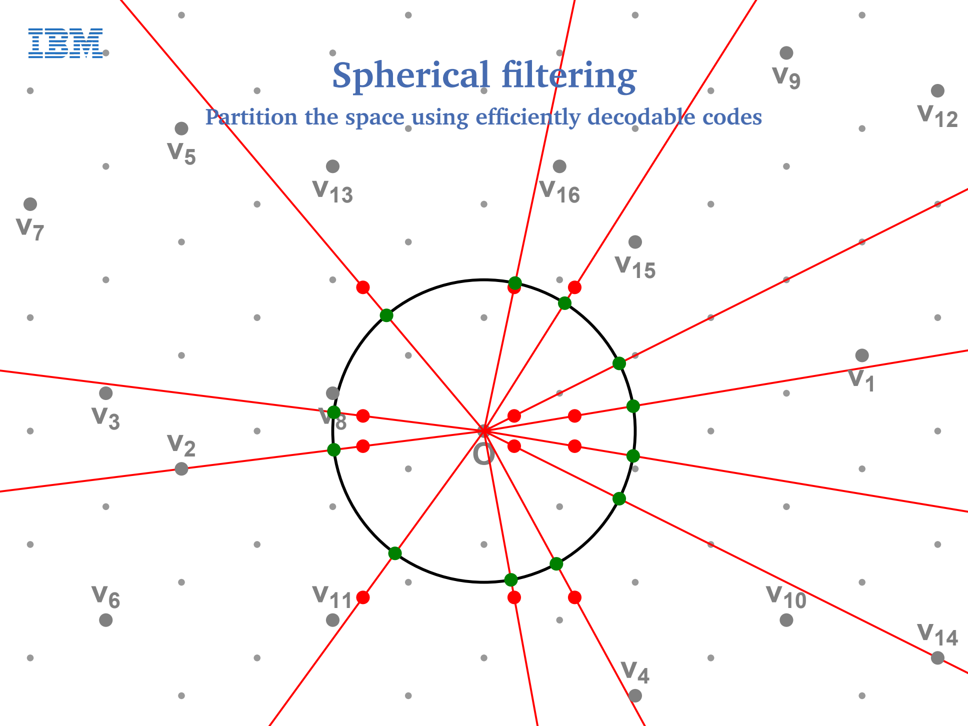
Spherical filtering

Partition the space using efficiently decodable codes



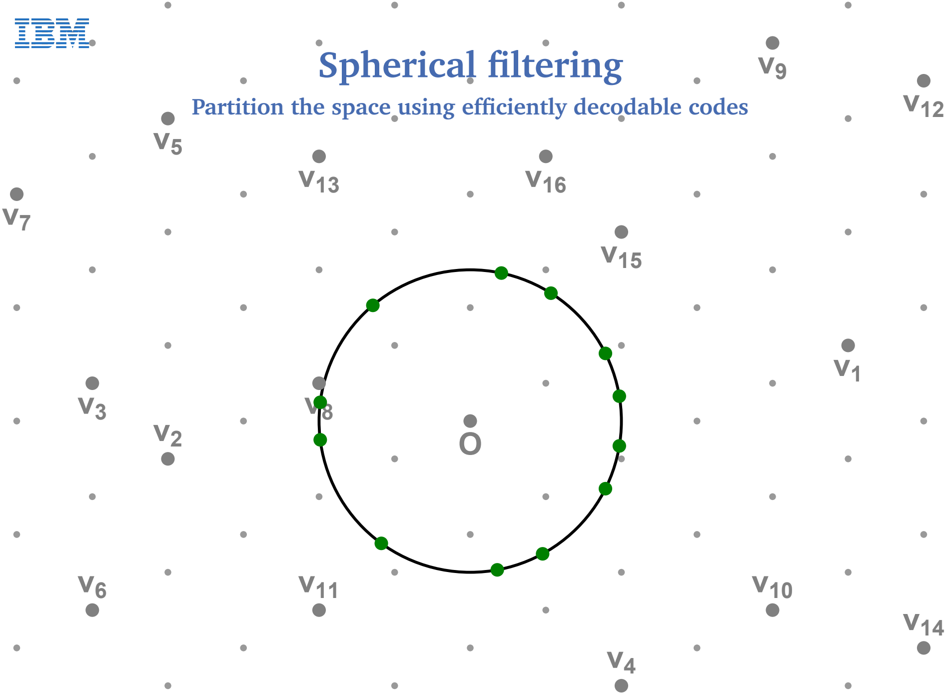
Spherical filtering

Partition the space using efficiently decodable codes



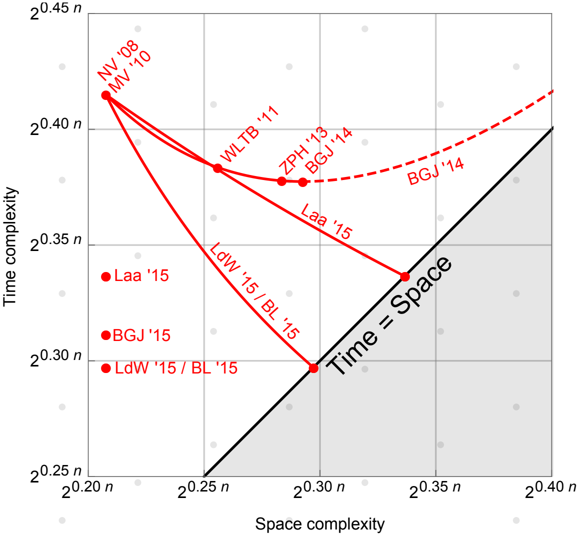
Spherical filtering

Partition the space using efficiently decodable codes



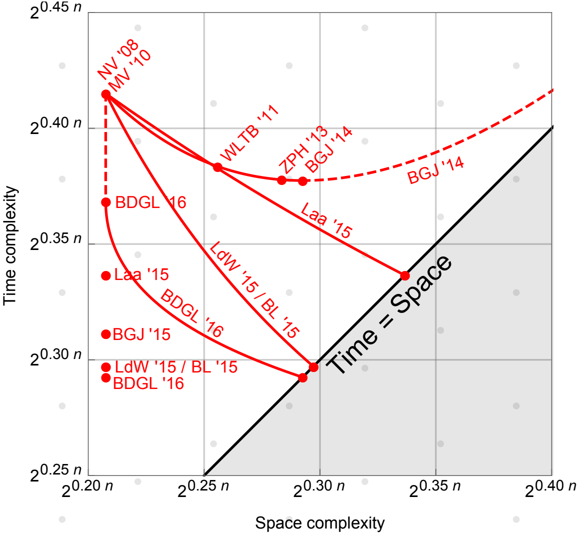
Spherical filtering

Space/time trade-off



Spherical filtering

Space/time trade-off



SVP in practice

"We expect our [enumeration] algorithm to be more efficient than lattice sieving up to dimension $n = 1895$."

— Micciancio–Walter, *SODA'15*

SVP in practice

"We expect our [enumeration] algorithm to be more efficient than lattice sieving up to dimension $n = 1895$."

— Micciancio–Walter, *SODA'15*

"As far as I know, everyone who has tried sieving as a BKZ subroutine in place of enumeration has concluded that sieving is much too slow to be useful—the cutoff is beyond cryptographically relevant sizes."

— Bernstein, *Google groups '16*

SVP in practice

"We expect our [enumeration] algorithm to be more efficient than lattice sieving up to dimension $n = 1895$."

— Micciancio–Walter, *SODA'15*

"As far as I know, everyone who has tried sieving as a BKZ subroutine in place of enumeration has concluded that sieving is much too slow to be useful—the cutoff is beyond cryptographically relevant sizes."

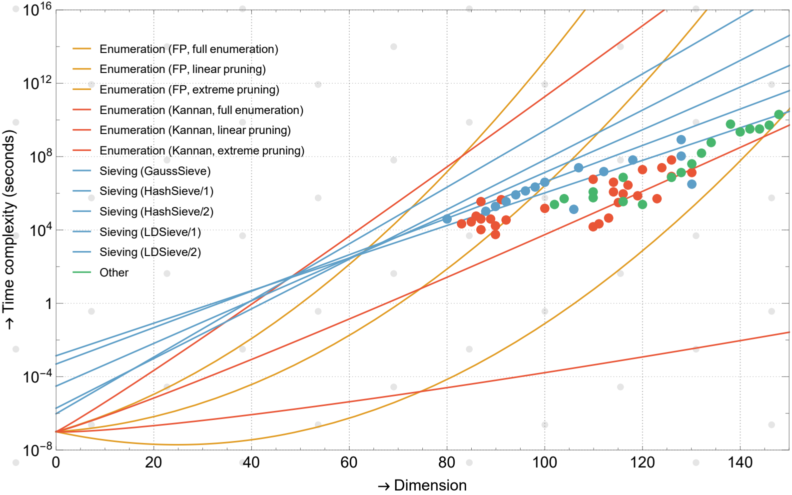
— Bernstein, *Google groups '16*

"I compute a cross-over point [between enumeration and the HashSieve] at dimension $b = 217$."

— Ducas, *Google groups '16*

IBM

SVP in practice



Open problems

- Exponential time, polynomial space for SVP(?)
- Effects of pruning on Kannan enumeration
- Close gap between provable and heuristic sieving
- Close gap between provable and heuristic enumeration
- Mixing/interpolating between different methods
- Sieving as BKZ subroutine
- Lower bounds on SVP complexity

Questions?