IBM Research

Sieving for shortest lattice vectors using near neighbor techniques

Thijs Laarhoven

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

Coding & Crypto seminar, Zürich, Switzerland
(April 26, 2017)

Outline

Lattices
- Basics
- Cryptography

Enumeration algorithms
- Fincke–Pohst enumeration
- Kannan enumeration
- Pruned enumeration

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

Practical comparison

Outline

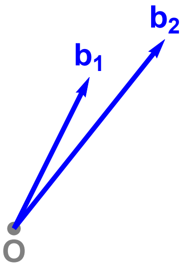
- Lattices
  - Basics
  - Cryptography
- Enumeration algorithms
  - Fincke–Pohst enumeration
  - Kannan enumeration
  - Pruned enumeration
- Sieving algorithms
  - Basic sieving
  - Leveled sieving
  - Near neighbor searching
- Practical comparison

Lattices

What is a lattice?

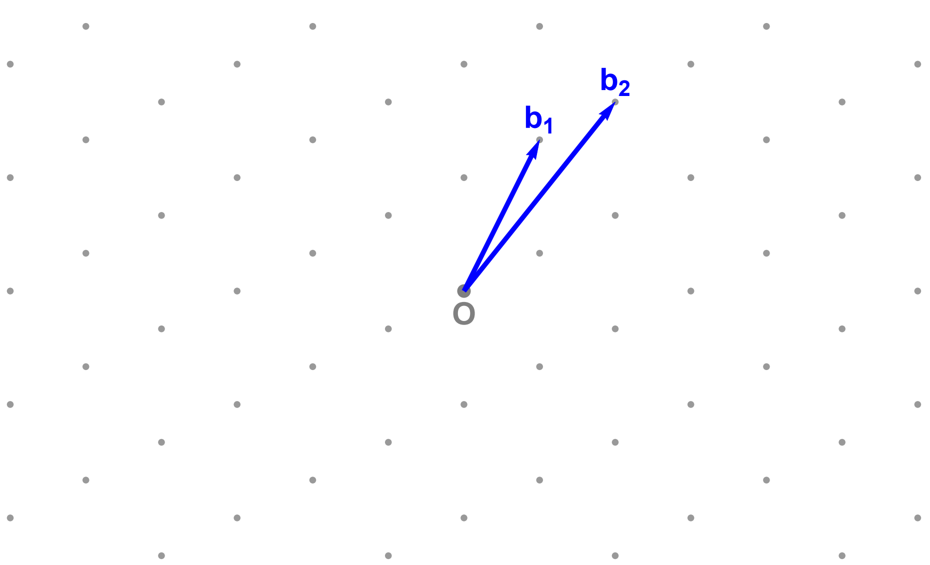
Lattices

What is a lattice?



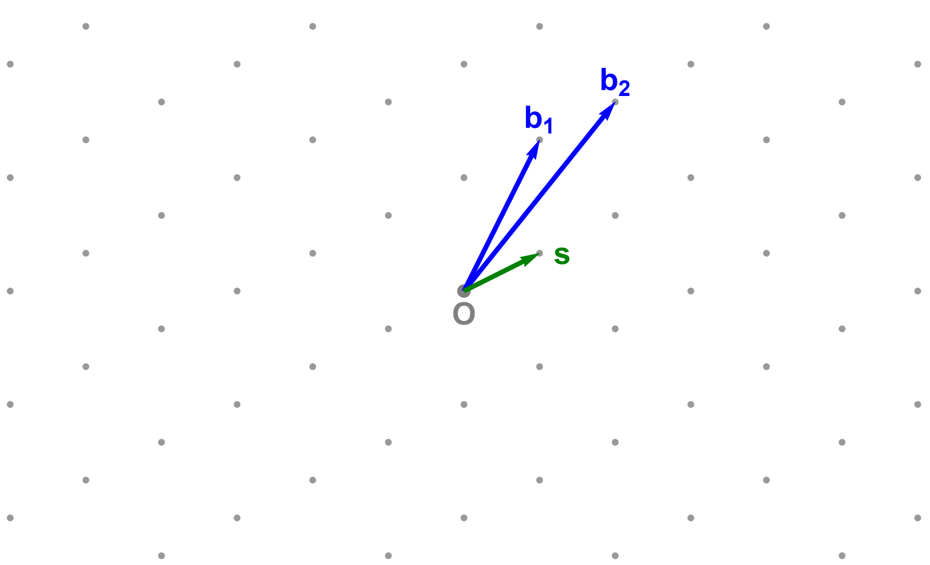
Lattices

What is a lattice?



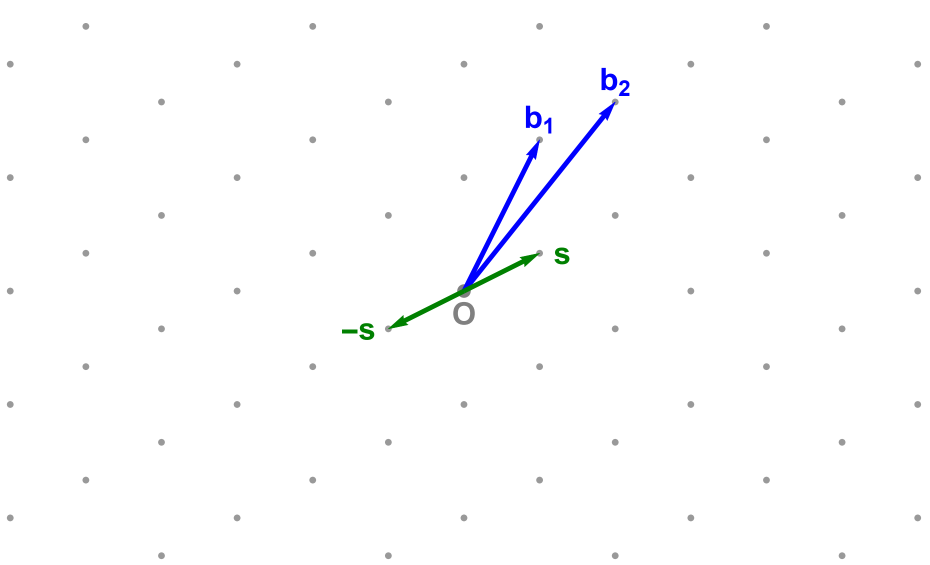
Lattices

Shortest Vector Problem (SVP)



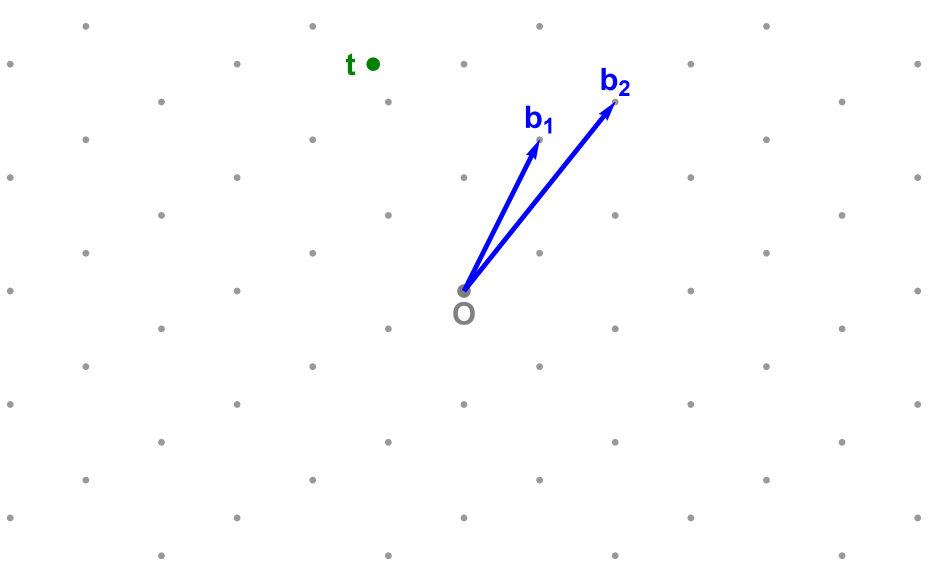
Lattices

Shortest Vector Problem (SVP)



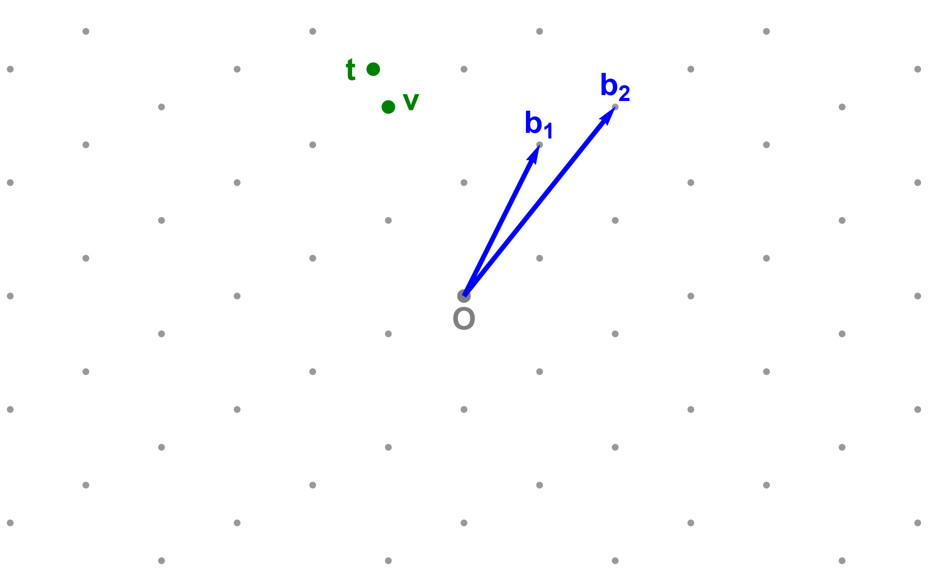
Lattices

Closest Vector Problem (CVP)



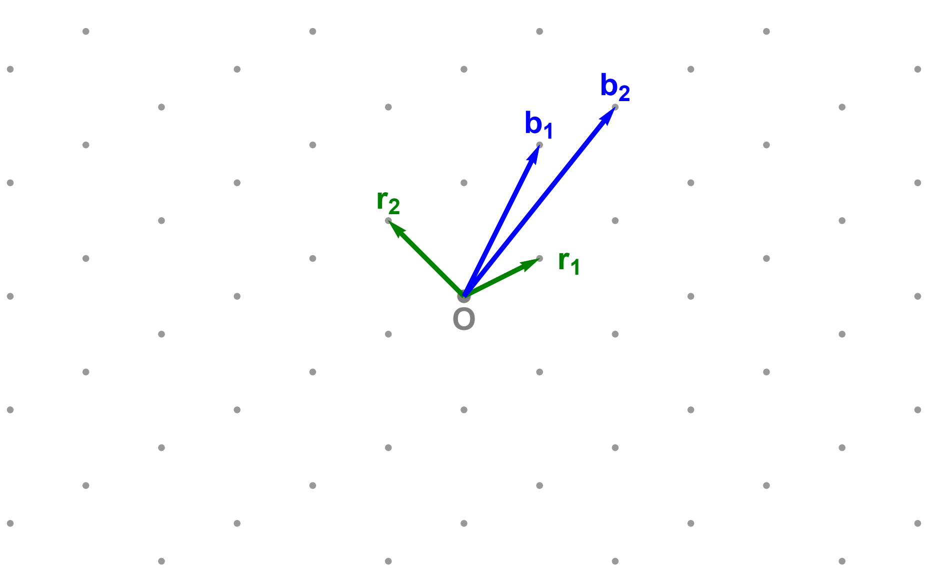
Lattices

Closest Vector Problem (CVP)



Lattices

Lattice basis reduction



Cryptography

GGH cryptosystem [GGH97]

Private key: $R = \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

Encrypt $\boldsymbol{m}$:

$$\boldsymbol{v} = \boldsymbol{m}B$$
$$\boldsymbol{c} = \boldsymbol{v} + \boldsymbol{e}$$

Decrypt $\boldsymbol{c}$:

$$\boldsymbol{v}' = \lfloor \boldsymbol{c}R^{-1} \rceil R$$
$$\boldsymbol{m}' = \boldsymbol{v}'B^{-1}$$

Cryptography

Private key

Private key: $R = \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

Encrypt $\boldsymbol{m}$:

$$\boldsymbol{v} = \boldsymbol{m}B$$
$$\boldsymbol{c} = \boldsymbol{v} + \boldsymbol{e}$$

Decrypt $\boldsymbol{c}$:

$$\boldsymbol{v}' = \lfloor \boldsymbol{c}R^{-1} \rceil R$$
$$\boldsymbol{m}' = \boldsymbol{v}'B^{-1}$$

Cryptography

Private key

Private key: $R = \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{pmatrix}$

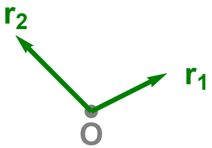
Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

Encrypt $\boldsymbol{m}$:

$$\boldsymbol{v} = \boldsymbol{m}B$$
$$\boldsymbol{c} = \boldsymbol{v} + \boldsymbol{e}$$

Decrypt $\boldsymbol{c}$:

$$\boldsymbol{v}' = \lfloor \boldsymbol{c}R^{-1} \rceil R$$
$$\boldsymbol{m}' = \boldsymbol{v}'B^{-1}$$



Cryptography

Public key

Private key: $R = \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{pmatrix}$

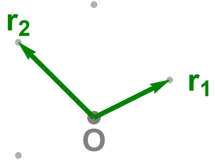
Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

Encrypt $\boldsymbol{m}$:

$$\boldsymbol{v} = \boldsymbol{m}B$$
$$\boldsymbol{c} = \boldsymbol{v} + \boldsymbol{e}$$

Decrypt $\boldsymbol{c}$:

$$\boldsymbol{v}' = \lfloor \boldsymbol{c}R^{-1} \rceil R$$
$$\boldsymbol{m}' = \boldsymbol{v}'B^{-1}$$



Cryptography

Public key

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

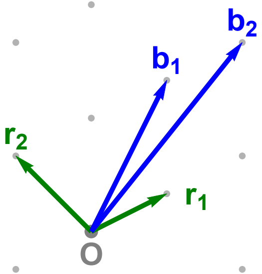
Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

$$v = mB$$
$$c = v + e$$

Decrypt c:

$$v' = \lfloor cR^{-1} \rceil R$$
$$m' = v'B^{-1}$$



Cryptography

Encryption

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

Encrypt $\boldsymbol{m}$:

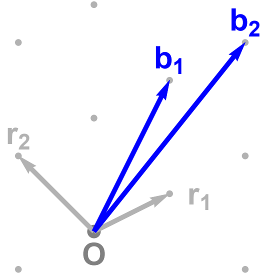
$$\boldsymbol{v} = \boldsymbol{m}B$$

$$\boldsymbol{c} = \boldsymbol{v} + \boldsymbol{e}$$

Decrypt $\boldsymbol{c}$:

$$\boldsymbol{v}' = \lfloor \boldsymbol{c}R^{-1} \rceil R$$

$$\boldsymbol{m}' = \boldsymbol{v}'B^{-1}$$



Cryptography

Encryption

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

Encrypt $\boldsymbol{m}$:

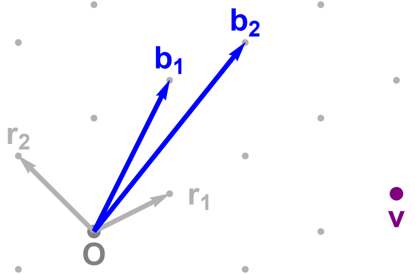
$$\boldsymbol{v} = \boldsymbol{m}B$$

$$\boldsymbol{c} = \boldsymbol{v} + \boldsymbol{e}$$

Decrypt $\boldsymbol{c}$:

$$\boldsymbol{v}' = \lfloor \boldsymbol{c}R^{-1} \rceil R$$

$$\boldsymbol{m}' = \boldsymbol{v}'B^{-1}$$



Cryptography

Encryption

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

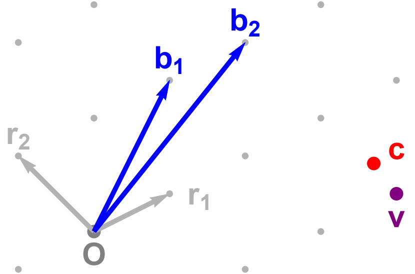
Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

Encrypt $\boldsymbol{m}$:

$$\boldsymbol{v} = \boldsymbol{m}B$$
$$\boldsymbol{c} = \boldsymbol{v} + \boldsymbol{e}$$

Decrypt $\boldsymbol{c}$:

$$v' = \lfloor \boldsymbol{c}R^{-1} \rceil R$$
$$m' = v'B^{-1}$$



Cryptography

Decryption with good basis

Private key: $R = \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{pmatrix}$

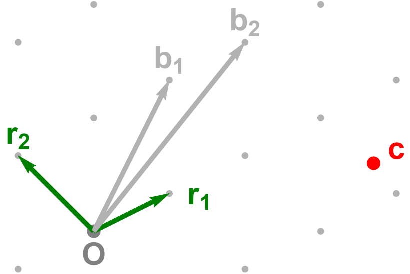
Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

$$v = mB$$
$$c = v + e$$

Decrypt $\boldsymbol{c}$:

$$\boldsymbol{v}' = \lfloor \boldsymbol{c}R^{-1} \rceil R$$
$$\boldsymbol{m}' = \boldsymbol{v}'B^{-1}$$



Cryptography

Decryption with good basis

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

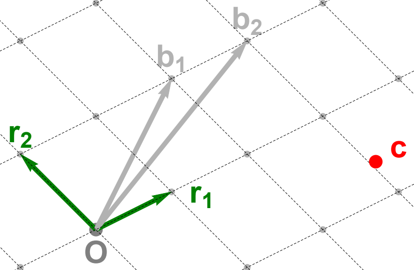
Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

$$v = mB$$
$$c = v + e$$

Decrypt c:

$$v' = \lfloor cR^{-1} \rceil R$$
$$m' = v'B^{-1}$$



Cryptography

Decryption with good basis

Private key: $R = \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{pmatrix}$

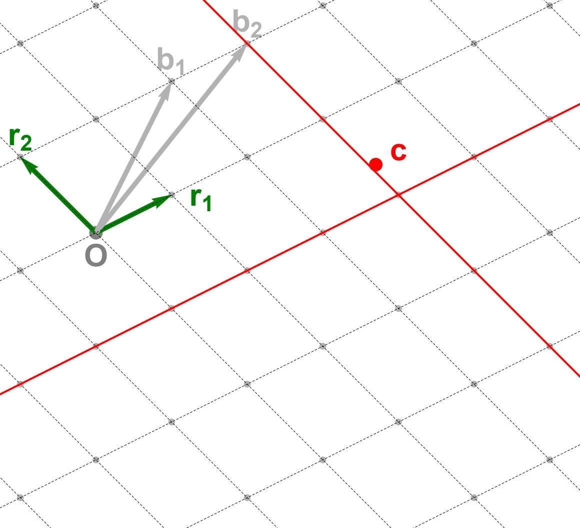
Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt $\boldsymbol{m}$:

$$\boldsymbol{v} = \boldsymbol{m}B$$
$$\boldsymbol{c} = \boldsymbol{v} + \boldsymbol{e}$$

Decrypt $\boldsymbol{c}$:

$$\boldsymbol{v}' = \lfloor \boldsymbol{c}R^{-1} \rceil R$$
$$\boldsymbol{m}' = \boldsymbol{v}'B^{-1}$$



IBM

Cryptography

Decryption with good basis

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

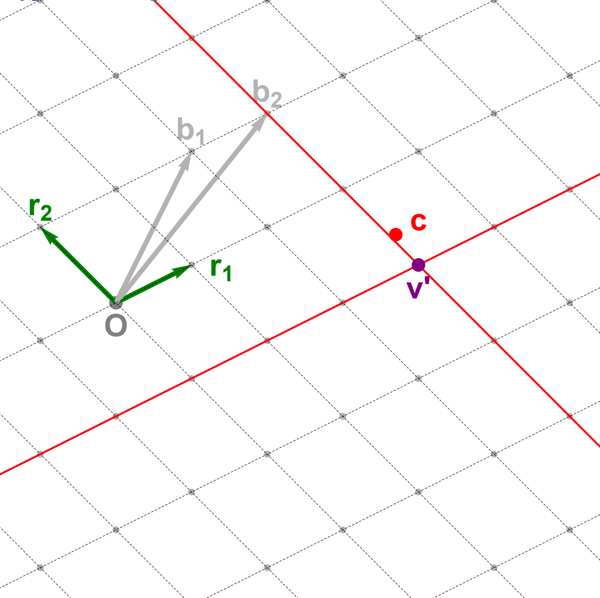
$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$



Cryptography

Decryption with bad basis

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

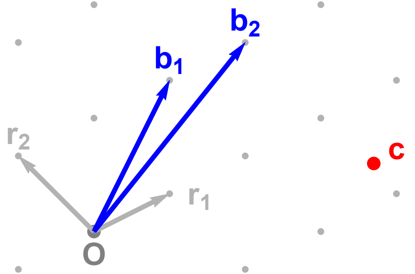
Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

Encrypt m:

$$v = mB$$
$$c = v + e$$

Decrypt $\boldsymbol{c}$:

$$\boldsymbol{v}' = \lfloor \boldsymbol{c}R^{-1} \rceil R$$
$$\boldsymbol{m}' = \boldsymbol{v}'B^{-1}$$



IBM

Cryptography

Decryption with bad basis

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

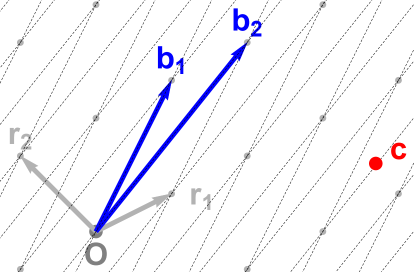
Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

$$v = mB$$
$$c = v + e$$

Decrypt c:

$$v' = \lfloor cR^{-1} \rceil R$$
$$m' = v' B^{-1}$$



Cryptography

Decryption with bad basis

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

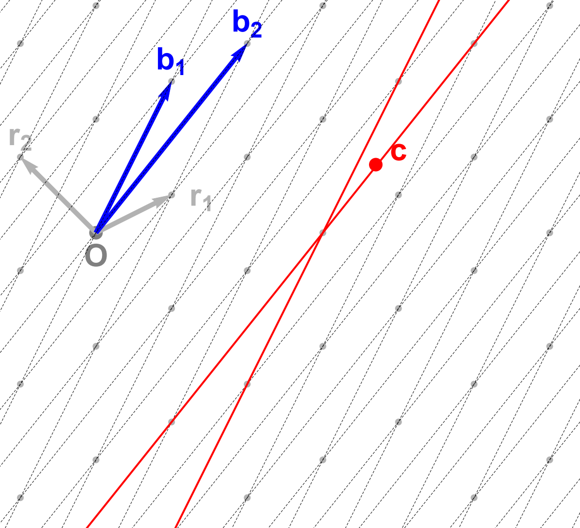
$v = mB$

$c = v + e$

Decrypt c:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v'B^{-1}$



Cryptography

Decryption with bad basis

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Encrypt m:

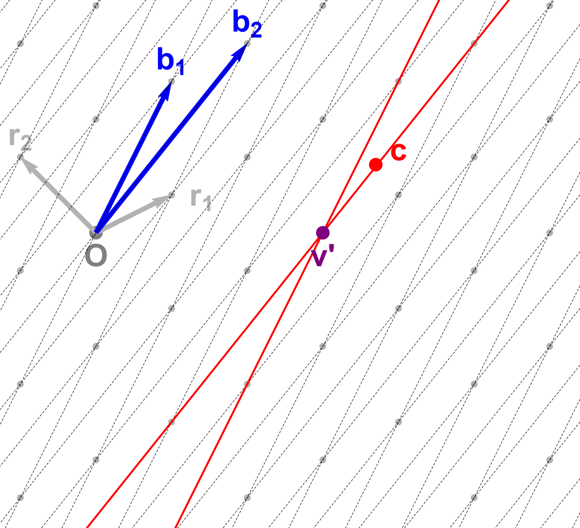
$v = mB$

$c = v + e$

Decrypt **c**:

$v' = \lfloor cR^{-1} \rceil R$

$m' = v' B^{-1}$



Cryptography

Overview

Private key: $R = \begin{pmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}$

Encrypt $\mathbf{m}$:

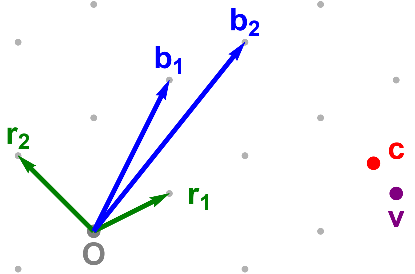
$$\mathbf{v} = \mathbf{m}B$$

$$\mathbf{c} = \mathbf{v} + \mathbf{e}$$

Decrypt $\mathbf{c}$:

$$\mathbf{v}' = \lfloor \mathbf{c}R^{-1} \rceil R$$

$$\mathbf{m}' = \mathbf{v}'B^{-1}$$



Cryptography

GGH signatures

Private key: $R = \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

Sign $\boldsymbol{m}$:

$\boldsymbol{c} = H(\boldsymbol{m})$

$\boldsymbol{s} = \lfloor \boldsymbol{c} R^{-1} \rceil R$

Verify $(\boldsymbol{m}, \boldsymbol{s})$:

$\boldsymbol{s}$ lies on the lattice

$\|\boldsymbol{s} - H(\boldsymbol{m})\|$ is small

Cryptography

Private and public keys

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Sign m:

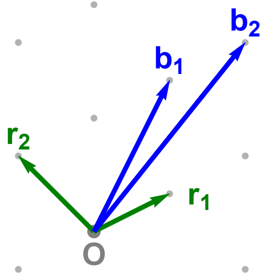
$c = H(m)$

$s = \lfloor cR^{-1} \rceil R$

Verify (m, s):

s lies on the lattice

$\|s - H(m)\|$ is small



IBM

Cryptography

Signing messages

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Sign m:

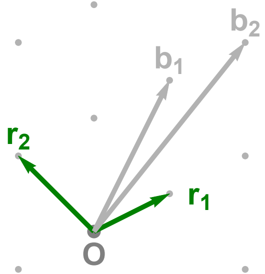
$c = H(m)$

$s = \lfloor cR^{-1} \rceil R$

Verify (m, s):

s lies on the lattice

$\|s - H(m)\|$ is small



Cryptography

Signing messages

Private key: $R = \begin{pmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}$

Sign $\mathbf{m}$:

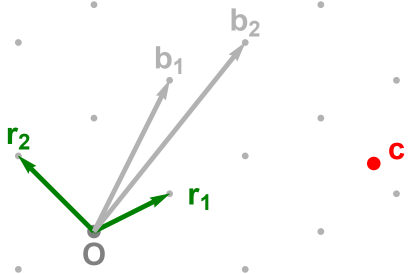
$\mathbf{c} = H(\mathbf{m})$

$\mathbf{s} = \lfloor \mathbf{c}R^{-1} \rceil R$

Verify $(\mathbf{m}, \mathbf{s})$:

$\mathbf{s}$ lies on the lattice

$\|\mathbf{s} - H(\mathbf{m})\|$ is small



Cryptography

Signing messages

Private key: $R = \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

Sign $\boldsymbol{m}$:

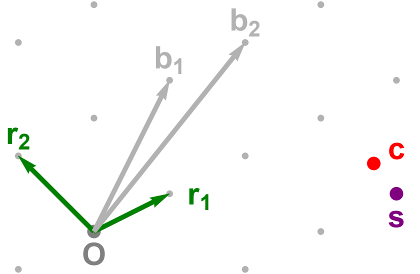
$\boldsymbol{c} = H(\boldsymbol{m})$

$\boldsymbol{s} = \lfloor \boldsymbol{c} R^{-1} \rceil R$

Verify $(\boldsymbol{m}, \boldsymbol{s})$:

$\boldsymbol{s}$ lies on the lattice

$\|\boldsymbol{s} - H(\boldsymbol{m})\|$ is small



Cryptography

Verifying signatures

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{pmatrix}$

Sign $\boldsymbol{m}$:

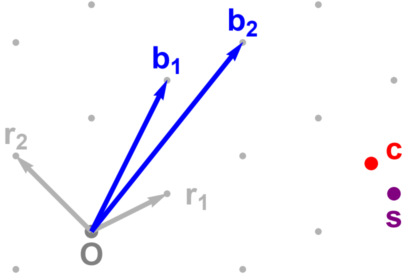
$c = H(m)$

$s = \lfloor cR^{-1} \rceil R$

Verify $(\boldsymbol{m}, \boldsymbol{s})$:

$\boldsymbol{s}$ lies on the lattice

$\|\boldsymbol{s} - H(\boldsymbol{m})\|$ is small



IBM

Cryptography

Overview

Private key: $R = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$

Public key: $B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Sign m:

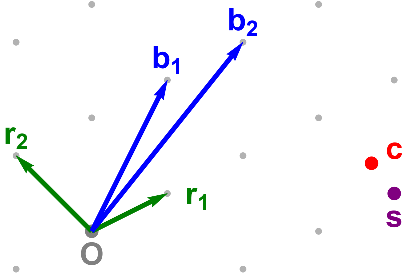
$c = H(m)$

$s = \lfloor cR^{-1} \rceil R$

Verify (m, s):

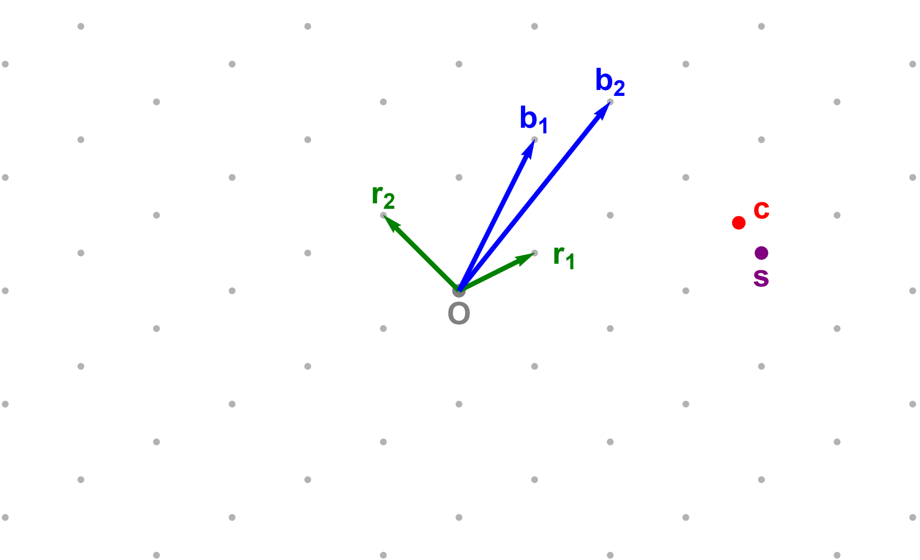
s lies on the lattice

$\|s - H(m)\|$ is small



Cryptography

Breaking the scheme [NR06]

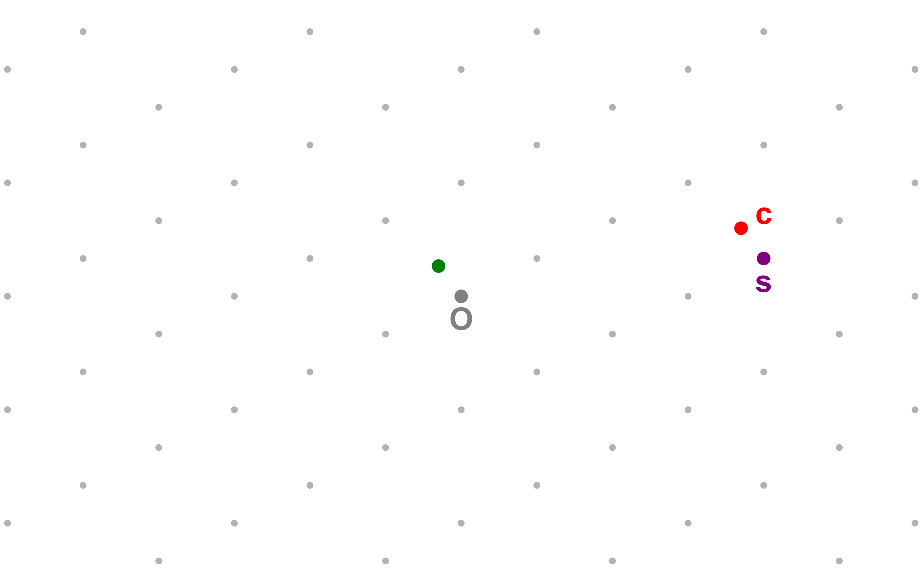


Cryptography

Breaking the scheme [NR06]

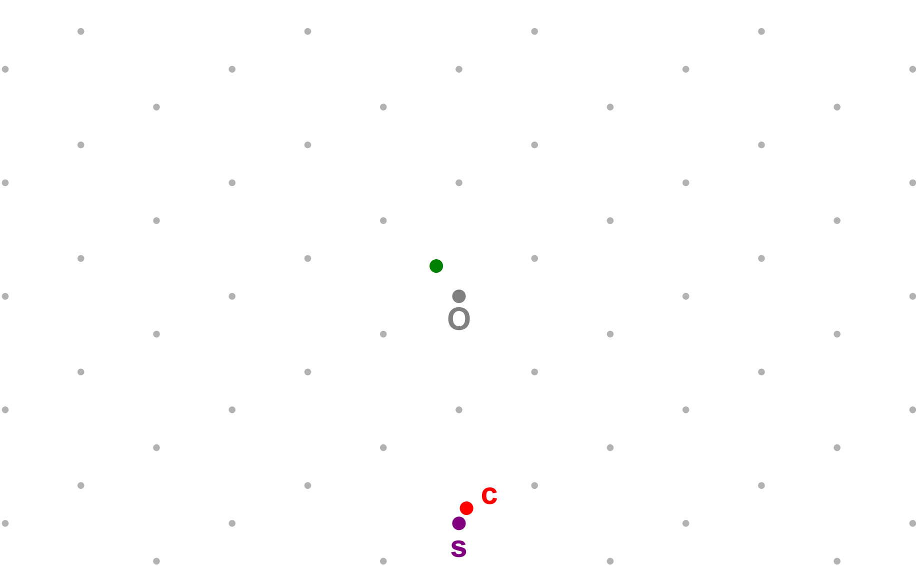
Cryptography

Breaking the scheme [NR06]



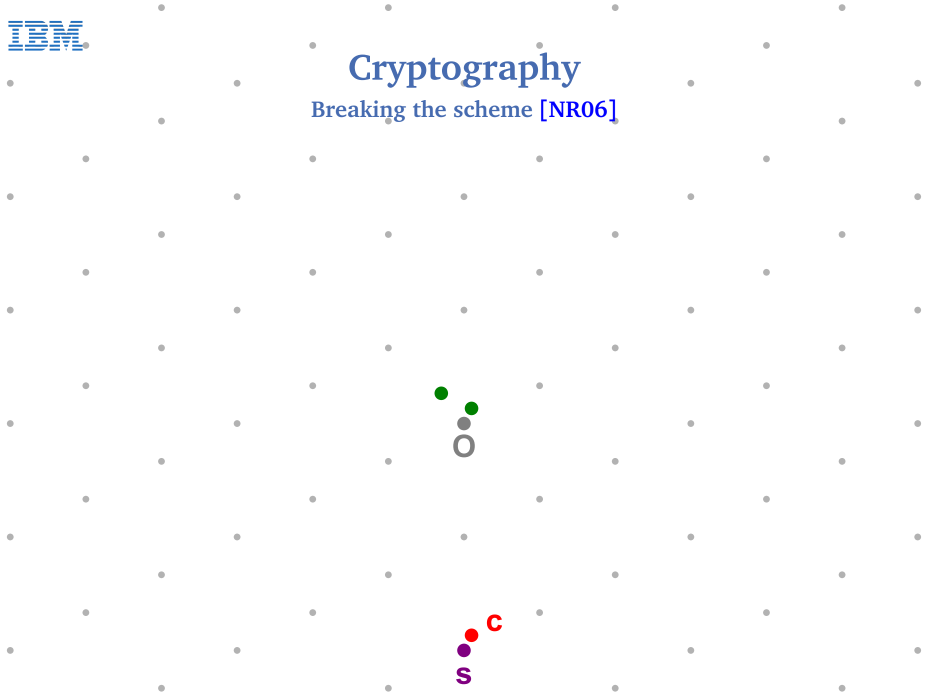
Cryptography

Breaking the scheme [NR06]



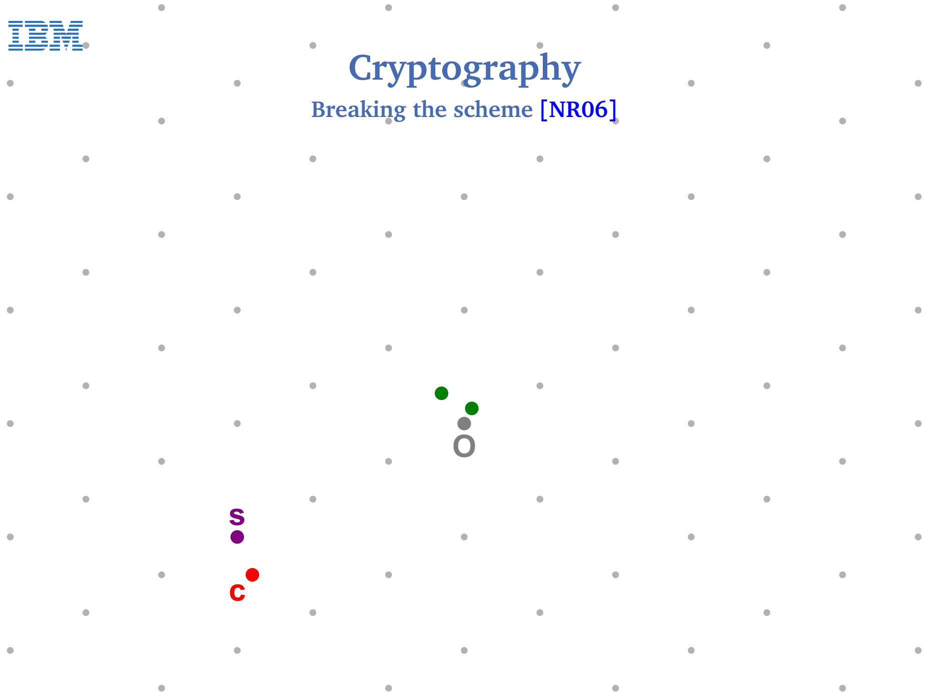
Cryptography

Breaking the scheme [NR06]



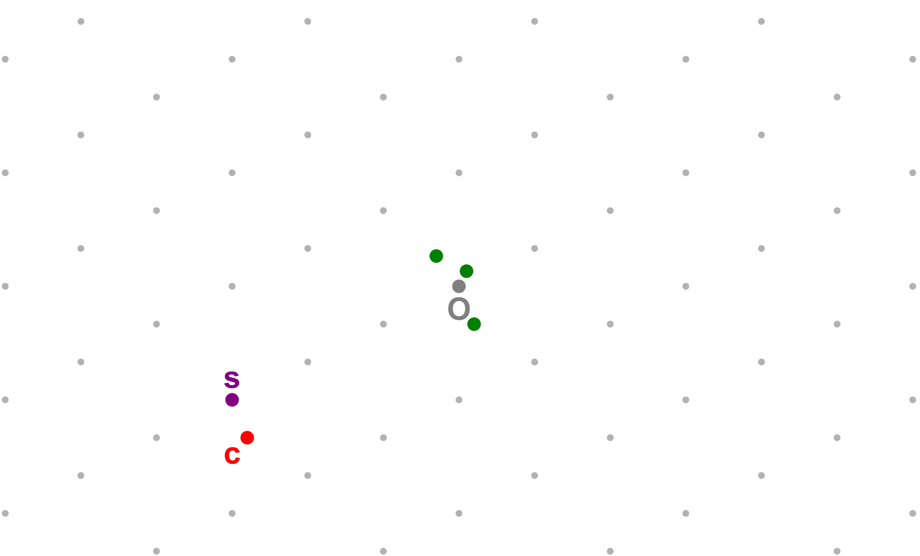
Cryptography

Breaking the scheme [NR06]



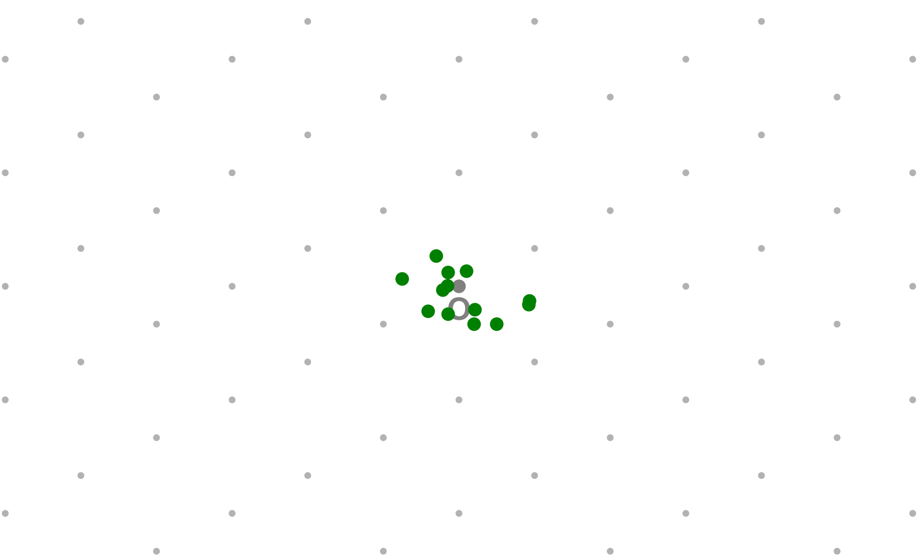
Cryptography

Breaking the scheme [NR06]



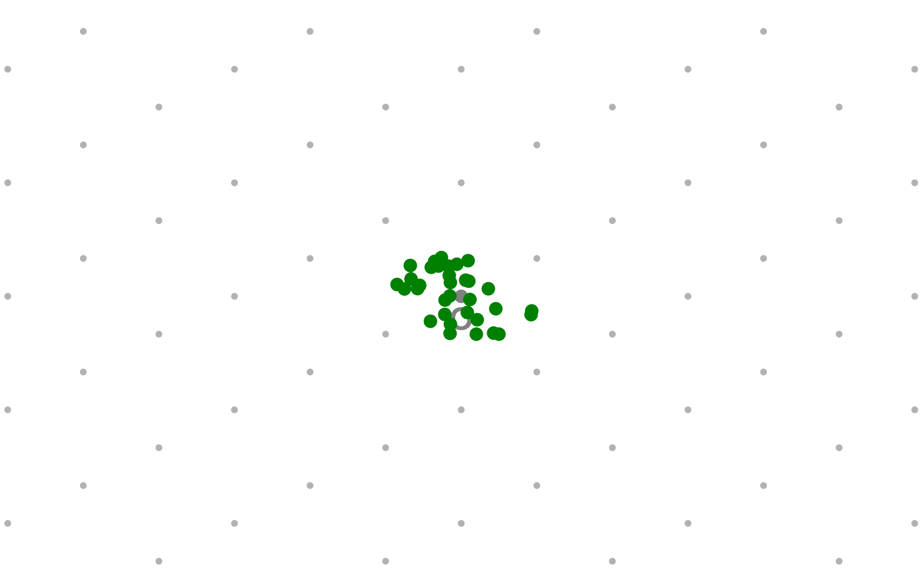
Cryptography

Breaking the scheme [NR06]



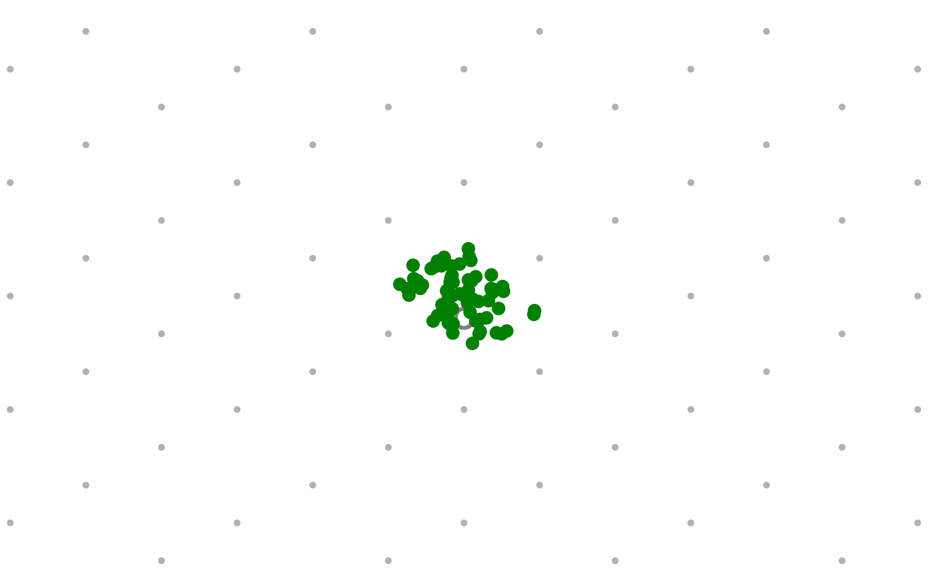
Cryptography

Breaking the scheme [NR06]



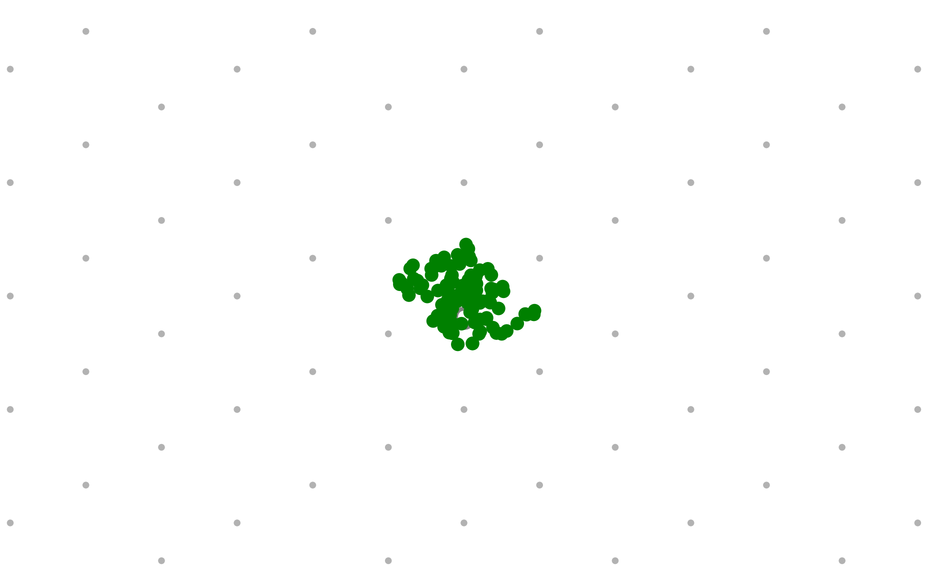
Cryptography

Breaking the scheme [NR06]



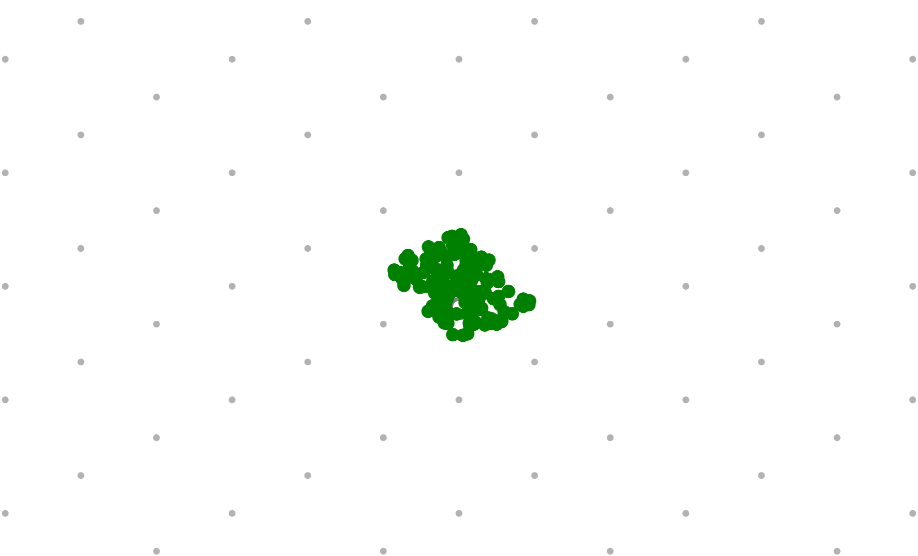
Cryptography

Breaking the scheme [NR06]



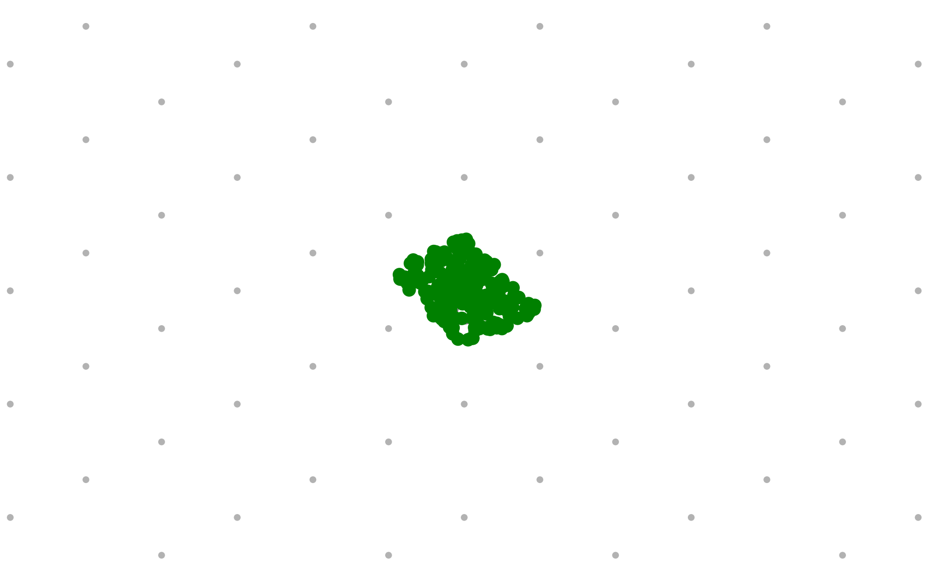
Cryptography

Breaking the scheme [NR06]



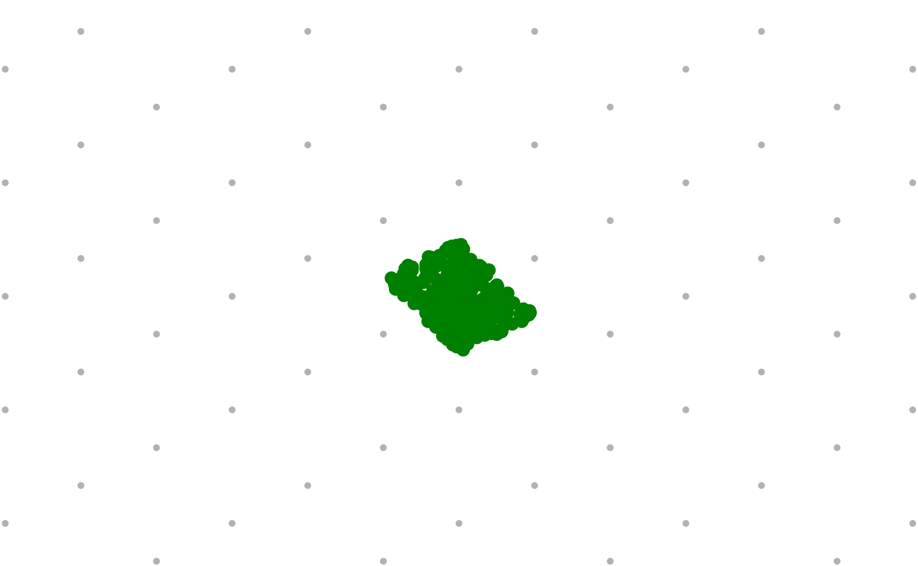
Cryptography

Breaking the scheme [NR06]



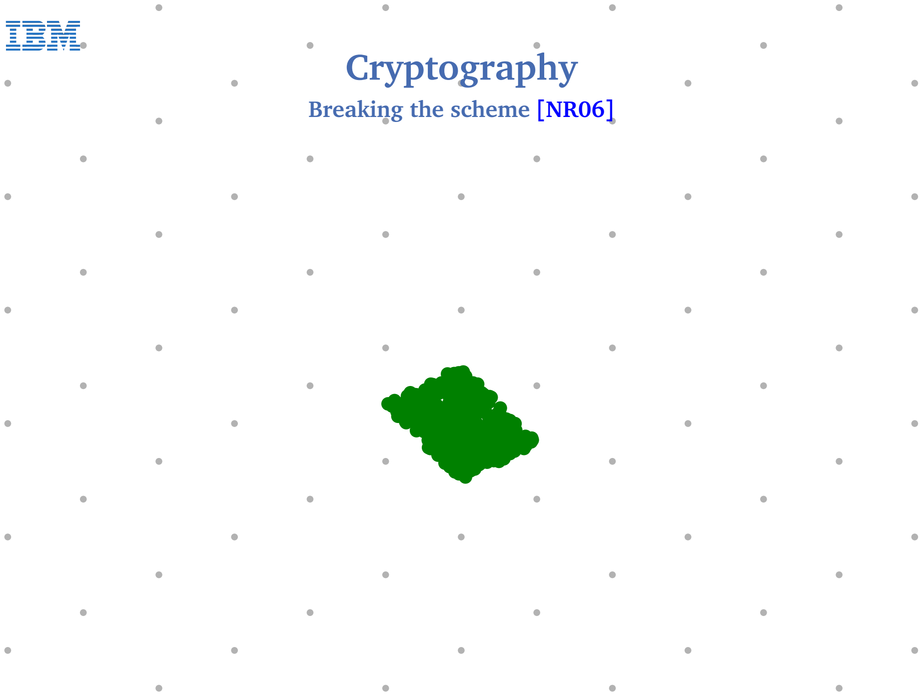
Cryptography

Breaking the scheme [NR06]



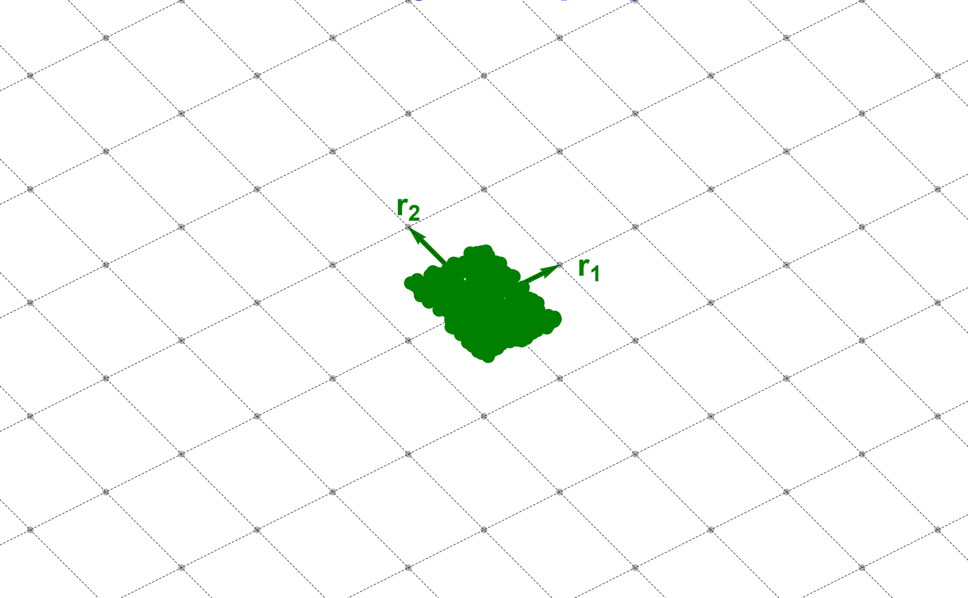
Cryptography

Breaking the scheme [NR06]



Cryptography

Breaking the scheme [NR06]



Cryptography

Security analysis

- Finding short bases implies breaking these schemes

Cryptography

Security analysis

- Finding short bases implies breaking these schemes
- Estimate hardness based on state-of-the-art basis reduction
  - LLL [LLL83] - fast, but poor quality in high dimensions
  - BKZ [Sch87, SE94] - arbitrary time/quality tradeoff
  - Variants of BKZ [..., MW16, AWHT16] - best in practice

Cryptography

Security analysis

- Finding short bases implies breaking these schemes
- Estimate hardness based on state-of-the-art basis reduction
  - LLL [LLL83] - fast, but poor quality in high dimensions
  - BKZ [Sch87, SE94] - arbitrary time/quality tradeoff
  - Variants of BKZ [..., MW16, AWHT16] - best in practice
- Complexity of BKZ dominated by SVP in projected lattices
  - ex. NewHope [ADPS16]: BKZ $\approx$ one call to SVP subroutine

Cryptography

Security analysis

- Finding short bases implies breaking these schemes
- Estimate hardness based on state-of-the-art basis reduction
  - LLL [LLL83] - fast, but poor quality in high dimensions
  - BKZ [Sch87, SE94] - arbitrary time/quality tradeoff
  - Variants of BKZ [..., MW16, AWHT16] - best in practice
- Complexity of BKZ dominated by SVP in projected lattices
  - ex. NewHope [ADPS16]: BKZ $\approx$ one call to SVP subroutine
- Question: What is the computational cost of exact SVP?

Lattices

Exact SVP algorithms

	Algorithm	$\log_2(\text{Time})$	$\log_2(\text{Space})$
Provable SVP	Enumeration [Poh81, Kan83, ..., MW15, AN17]	$O(n \log n)$	$O(\log n)$
	AKS-sieve [AKS01, NV08, MV10, HPS11]	$3.398n$	$1.985n$
	ListSieve [MV10, MDB14]	$3.199n$	$1.327n$
	Birthday sieves [PS09, HPS11]	$2.465n$	$1.233n$
	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]	$0.415n$	$0.208n$
	Leveled sieving [WLTB11, 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/cross-polytope LSH [LdW15, BL16]	$0.298n$	$0.208n^*$
	Spherical filtering [BDGL16, MLB17]	$0.293n$	$0.208n^*$
	Triple sieve [BLS16, HK17, Laa17]	$0.359n$	$0.188n$

Outline

Lattices
- Basics
- Cryptography

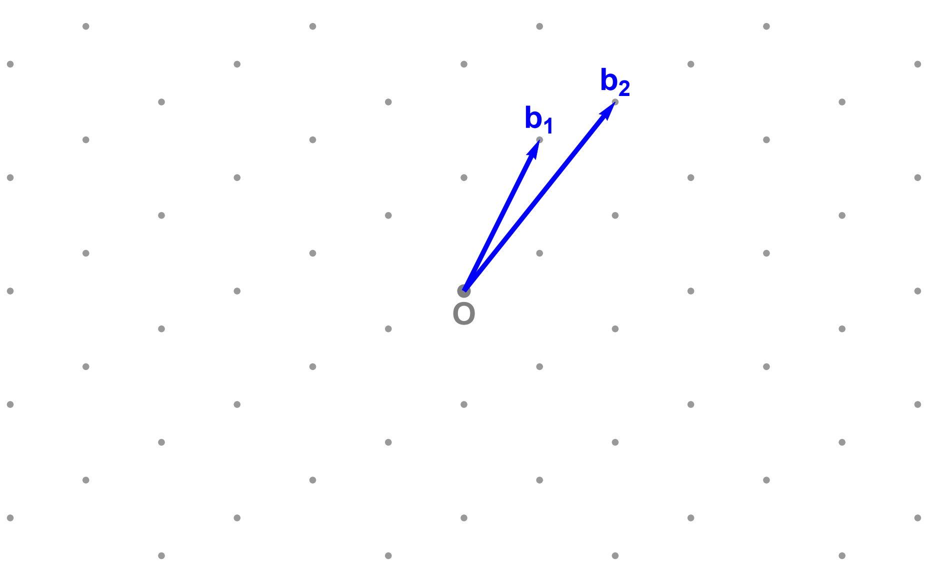
Enumeration algorithms
- Fincke–Pohst enumeration
- Kannan enumeration
- Pruned enumeration

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

Practical comparison

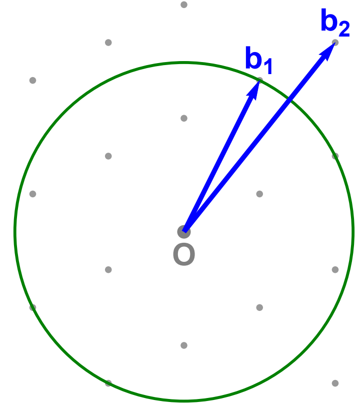
Fincke-Pohst enumeration

Determine possible coefficients of b_2



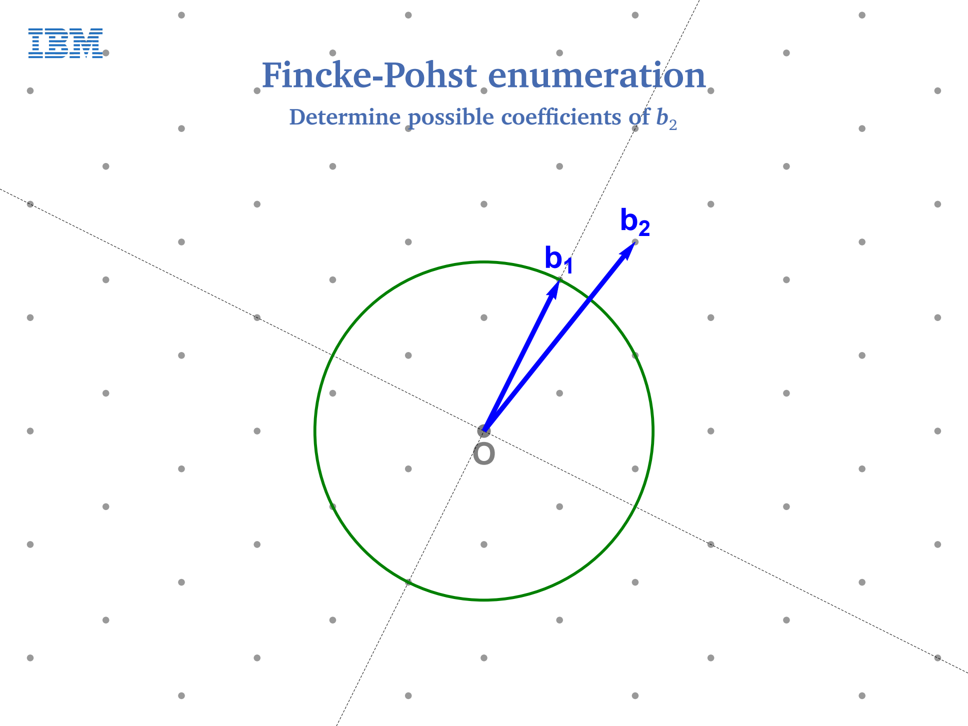
Fincke-Pohst enumeration

Determine possible coefficients of b_2



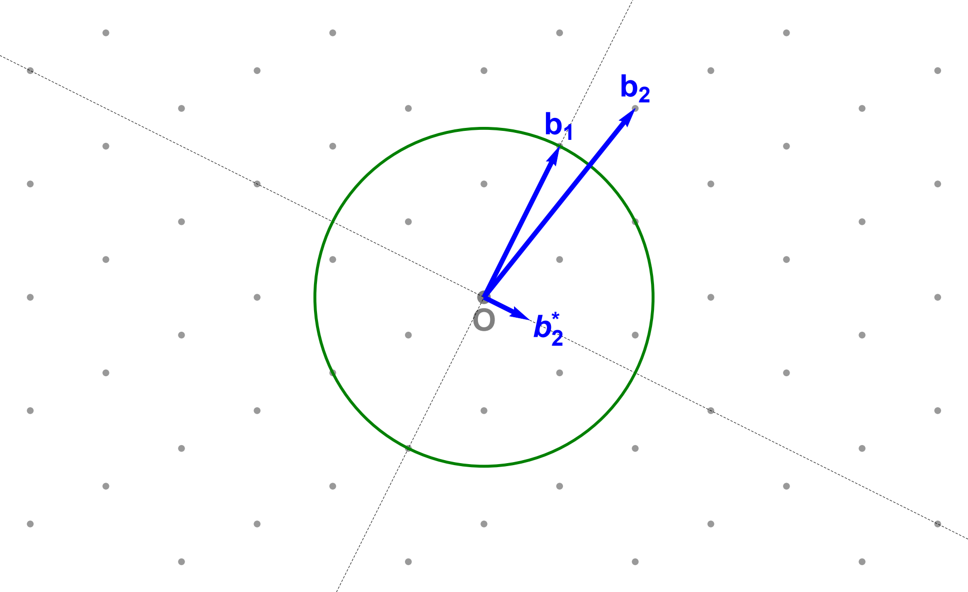
Fincke-Pohst enumeration

Determine possible coefficients of b_2



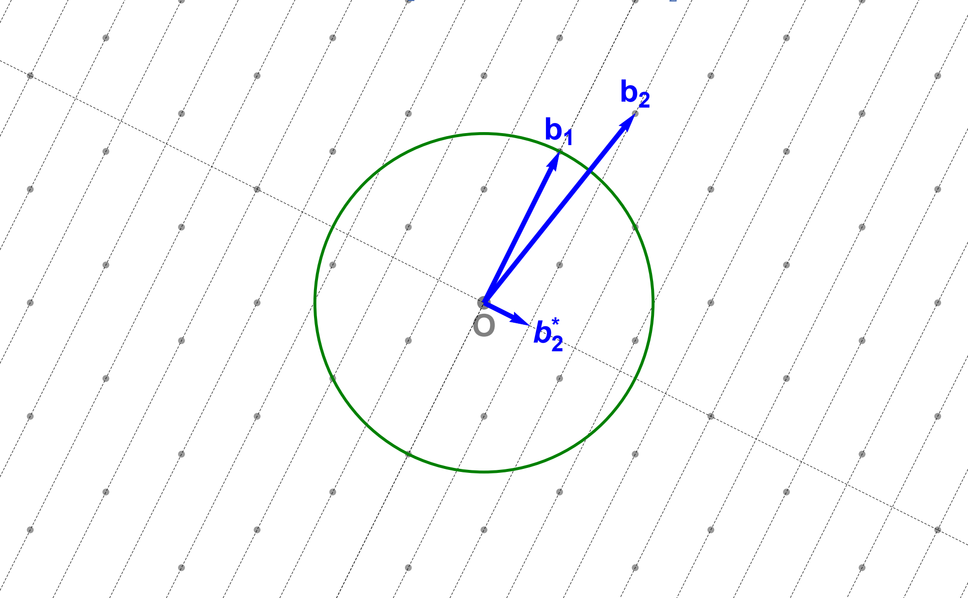
Fincke-Pohst enumeration

Determine possible coefficients of b_2



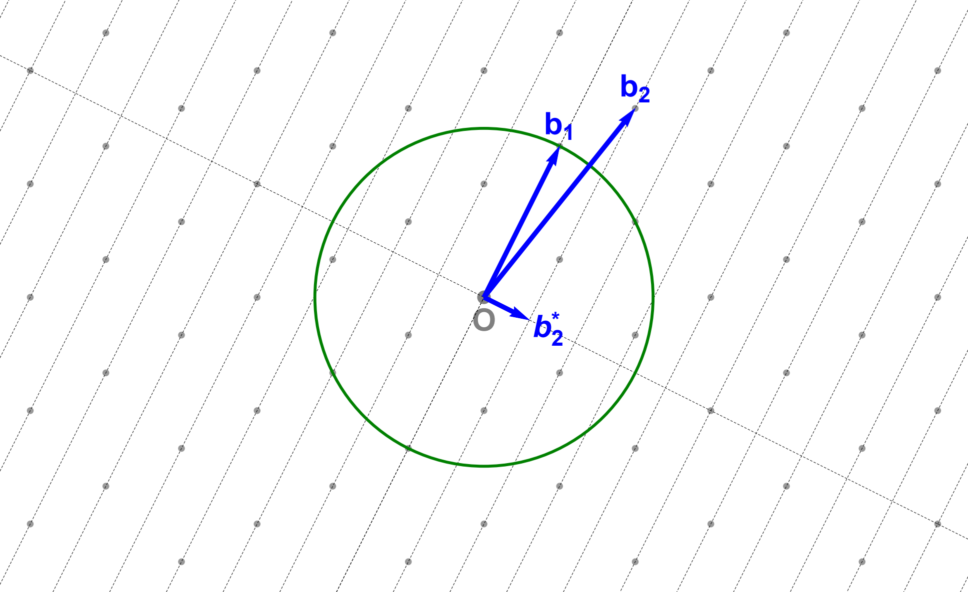
Fincke-Pohst enumeration

Determine possible coefficients of b_2



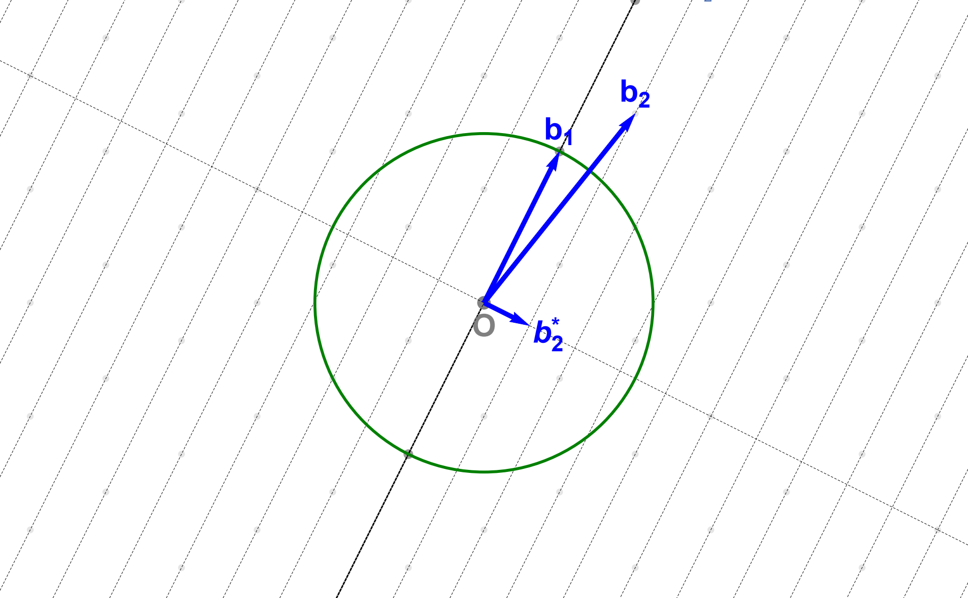
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



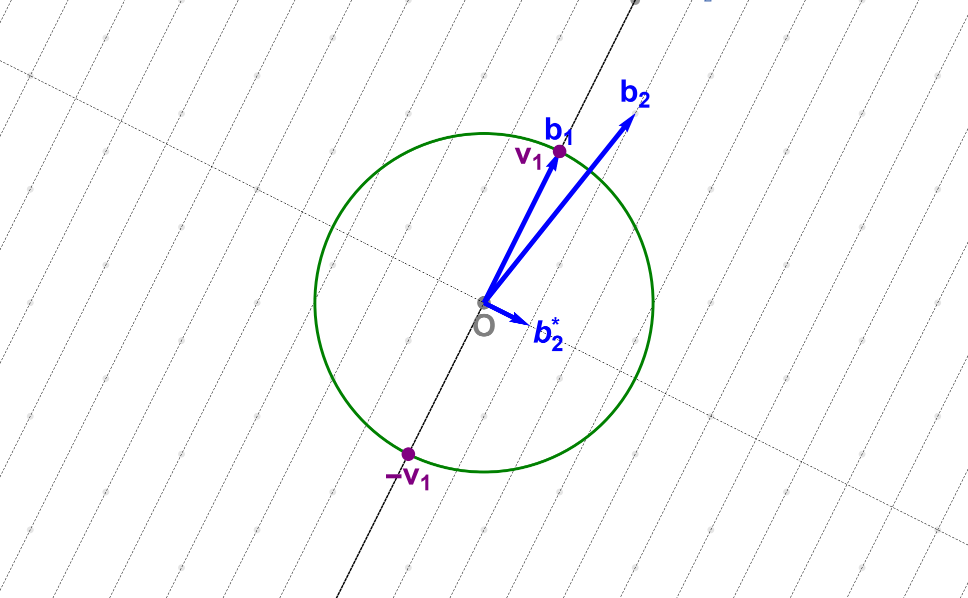
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



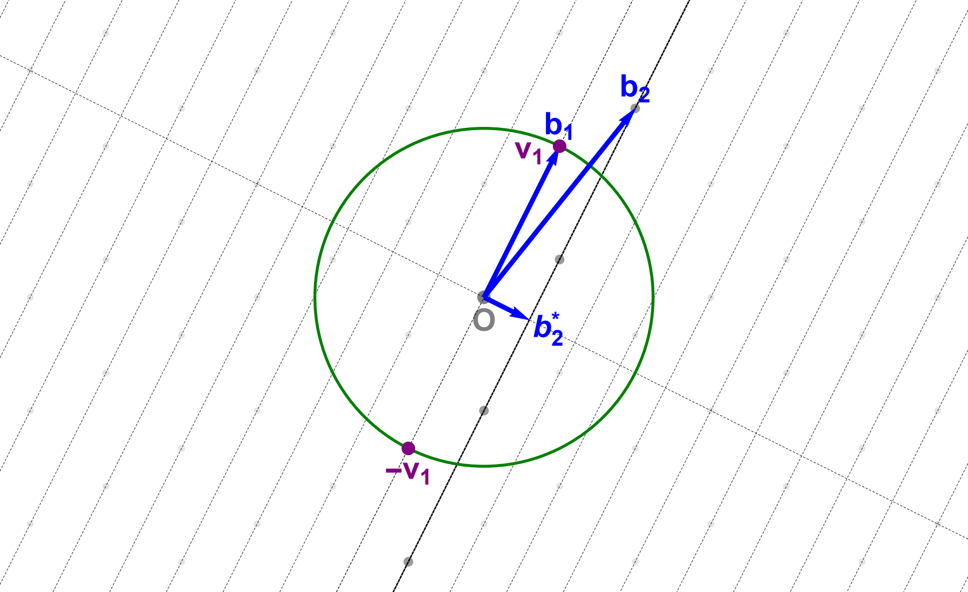
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



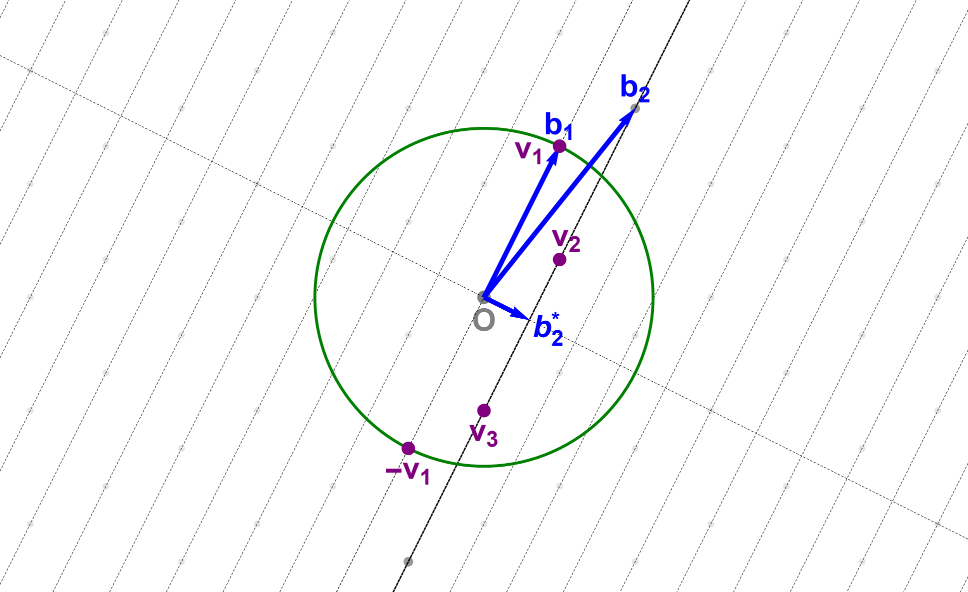
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



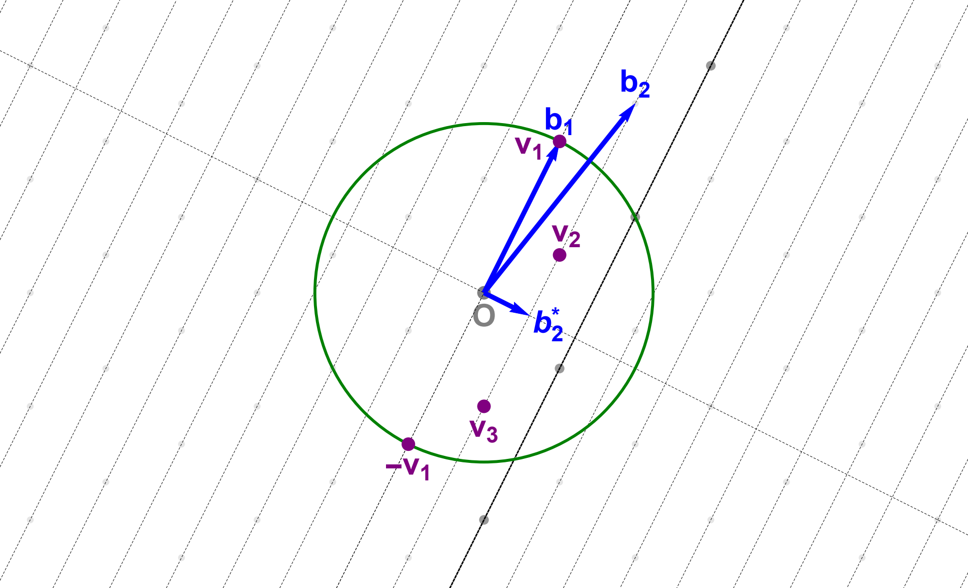
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



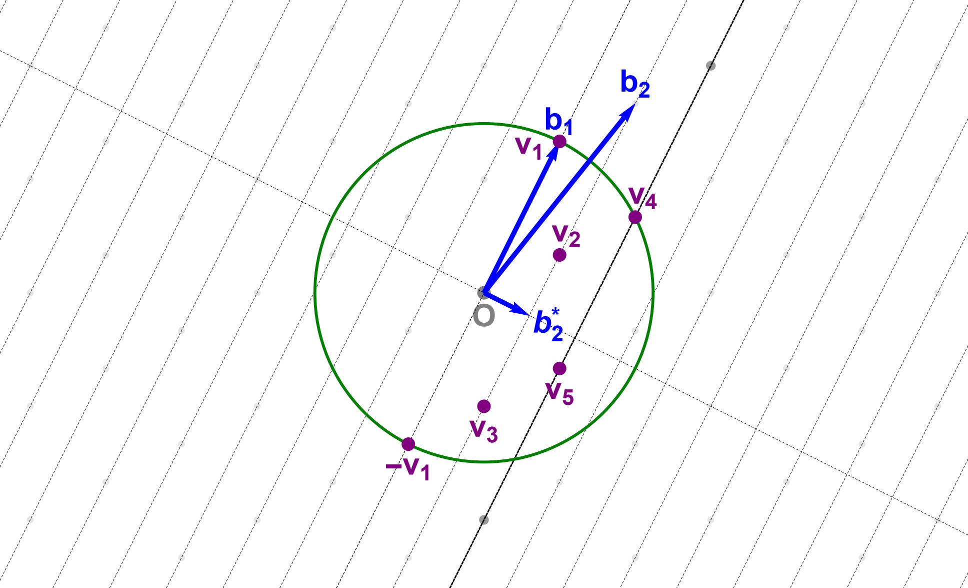
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



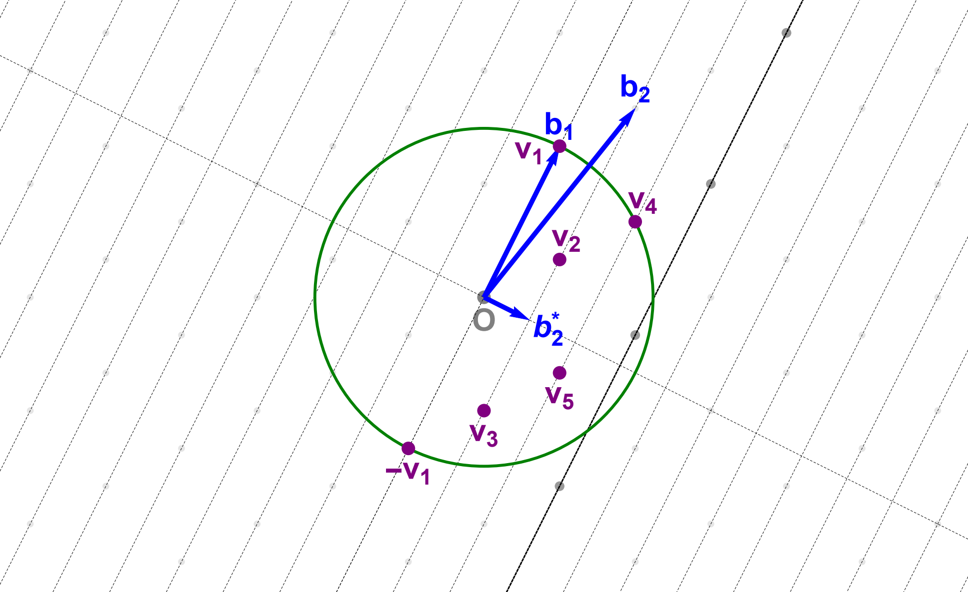
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



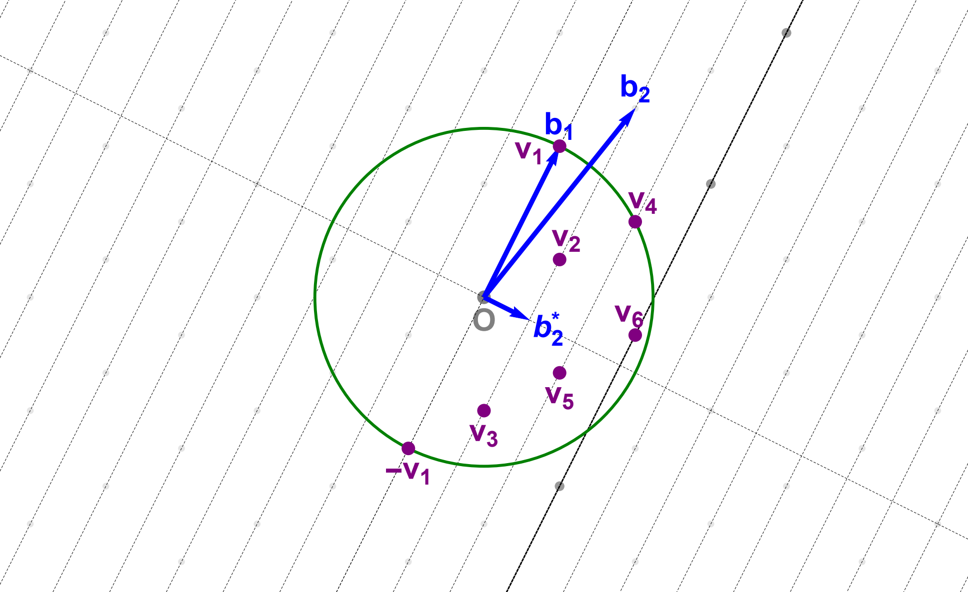
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



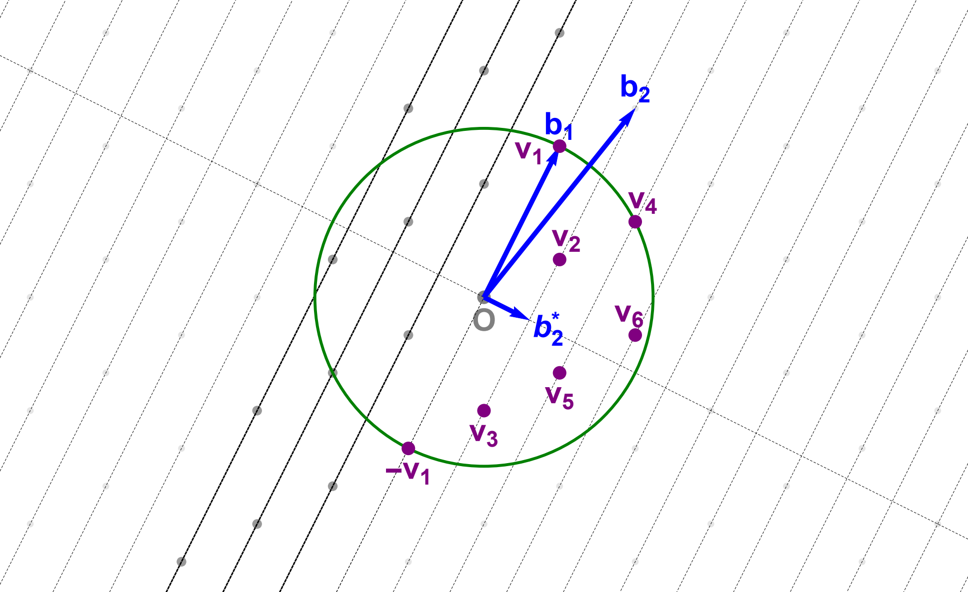
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



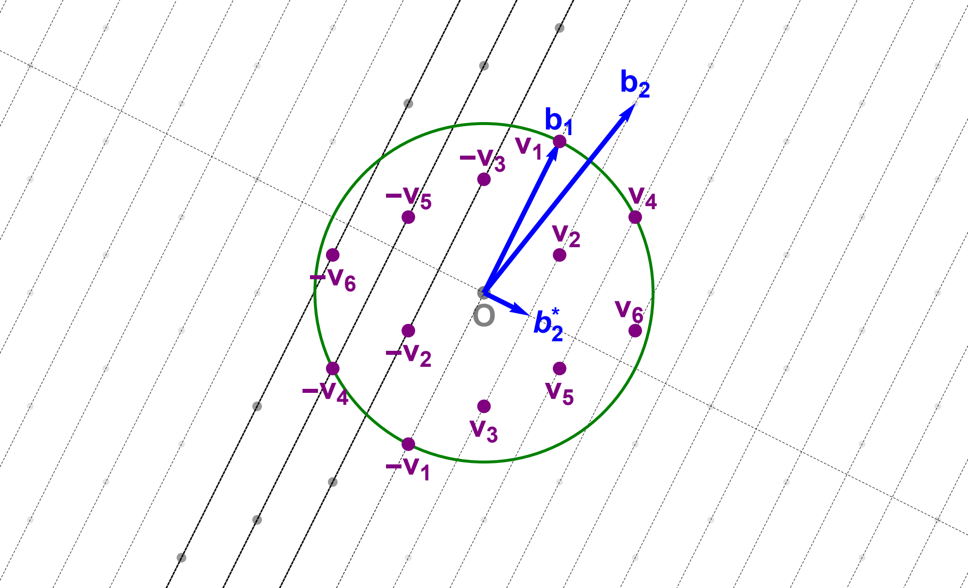
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



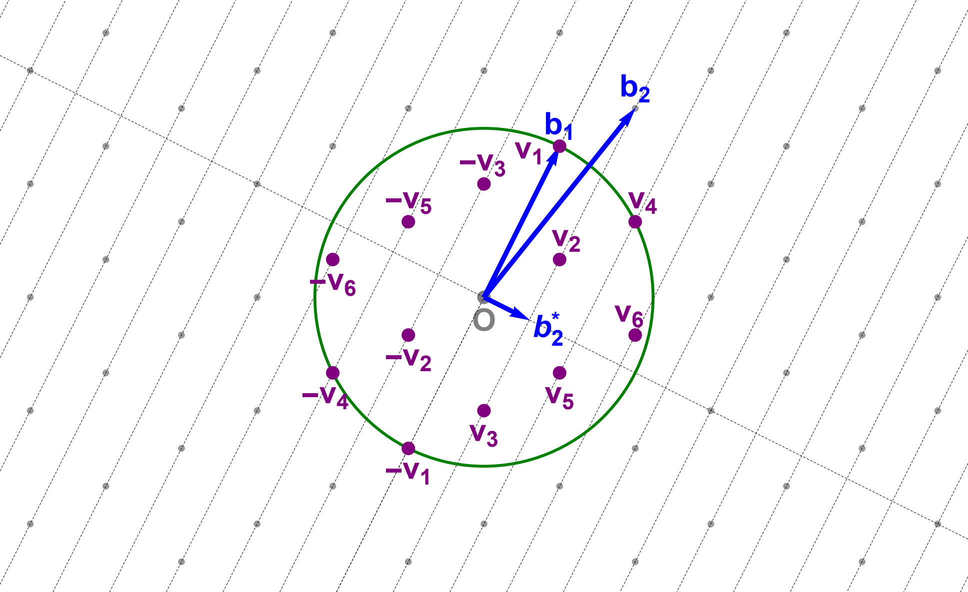
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



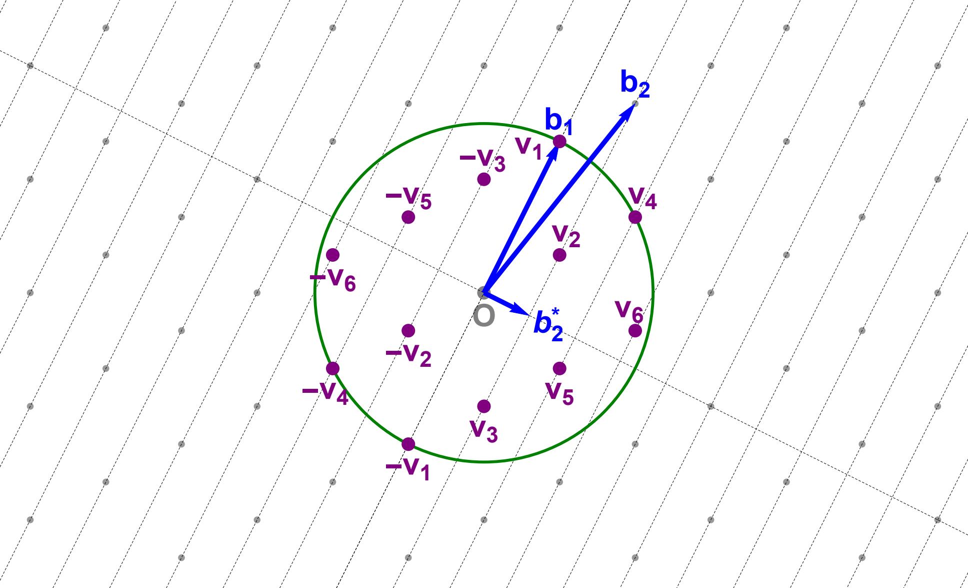
Fincke-Pohst enumeration

Find short vectors for each coefficient of b_2



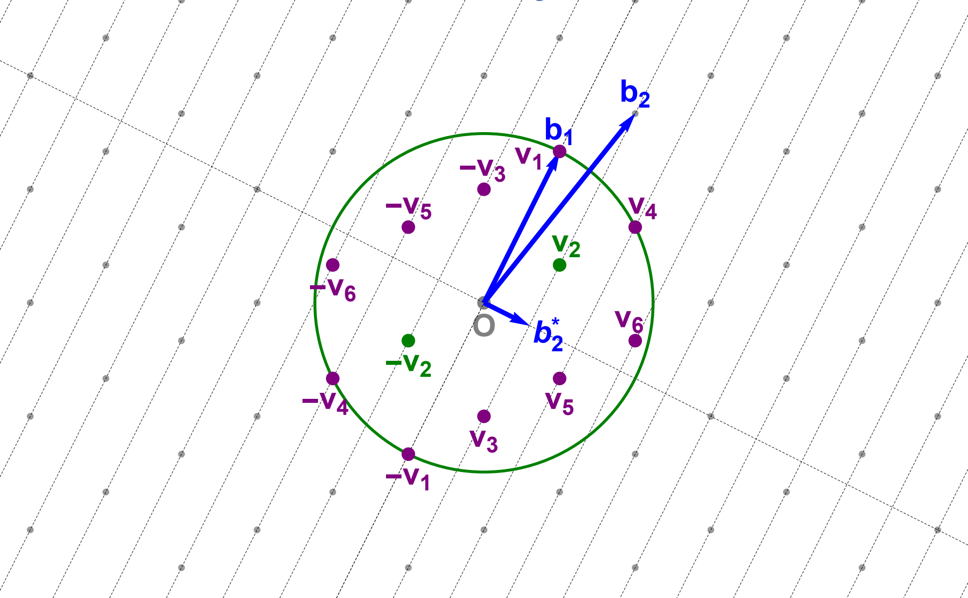
Fincke-Pohst enumeration

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

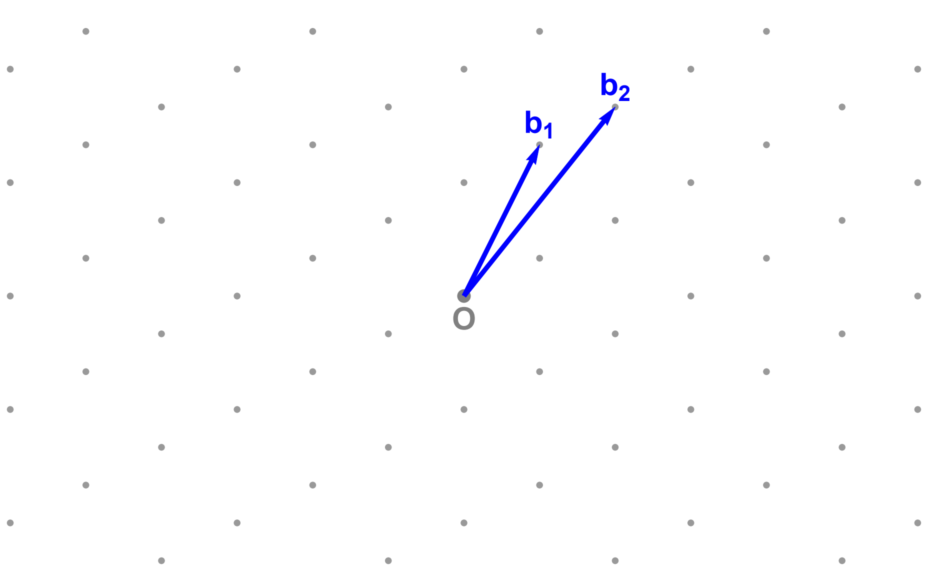
Theorem (Fincke–Pohst, Math. of Comp. '85)

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}

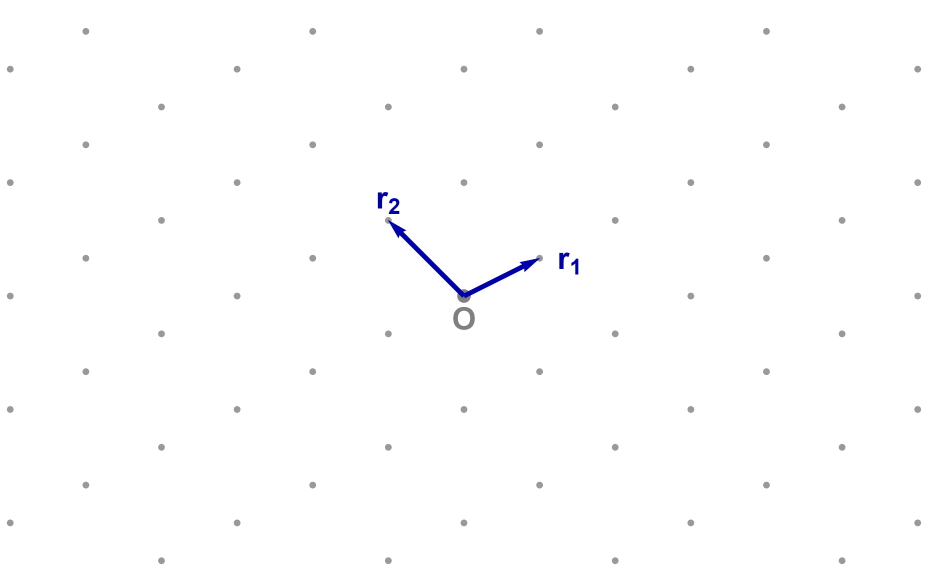
Kannan enumeration

Better bases



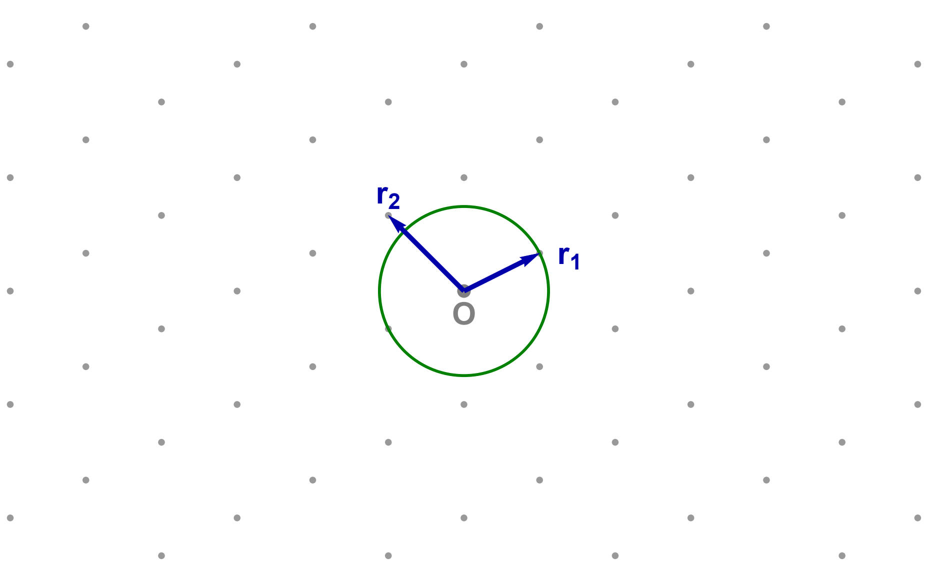
Kannan enumeration

Better bases



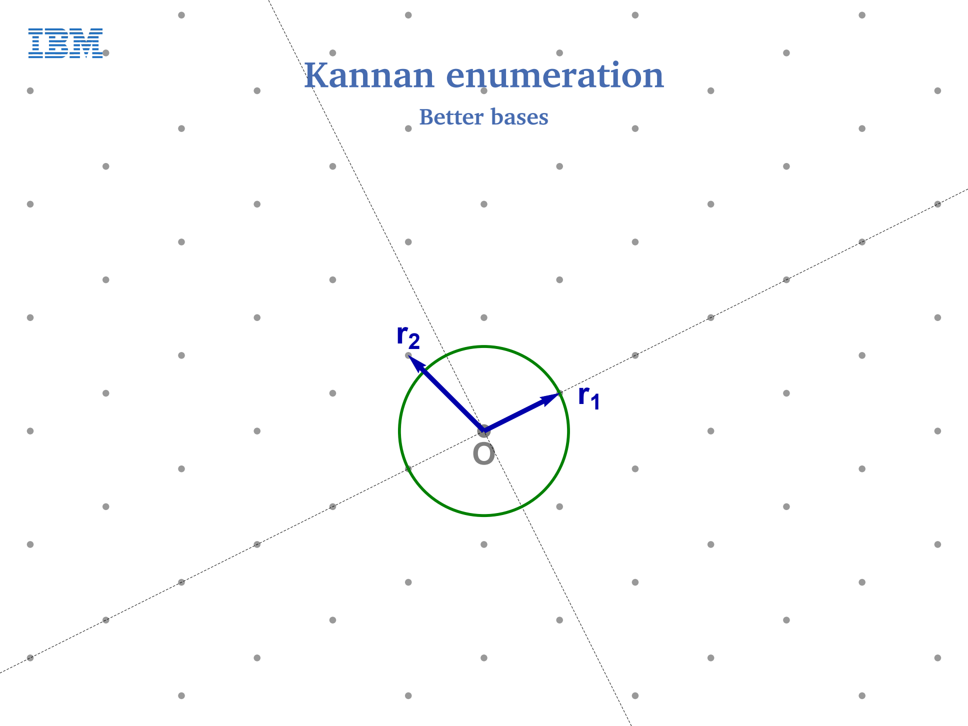
Kannan enumeration

Better bases



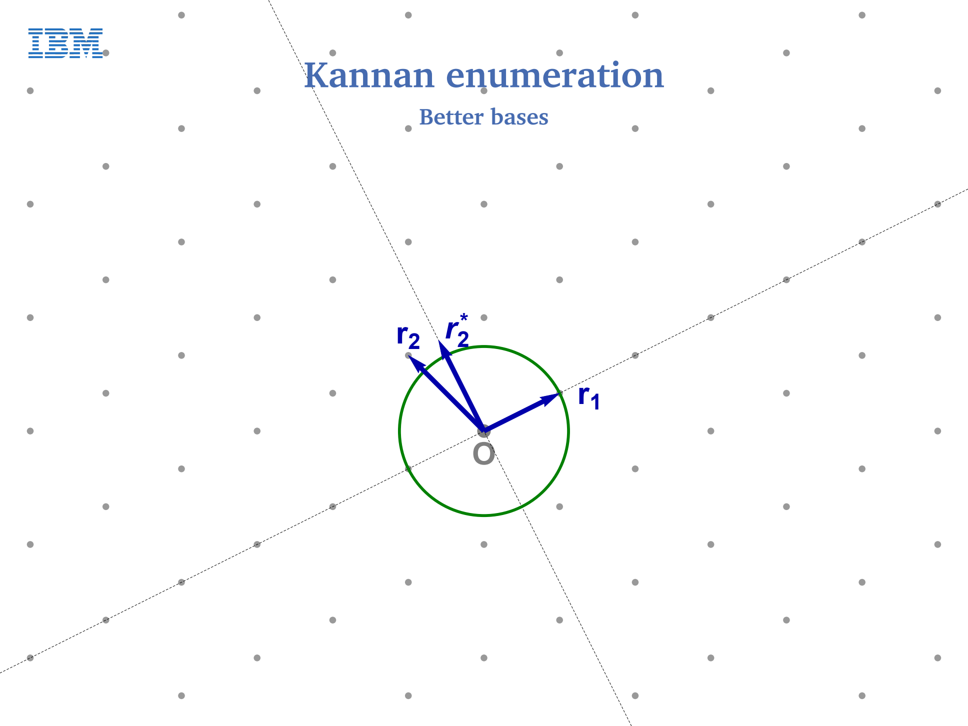
Kannan enumeration

Better bases



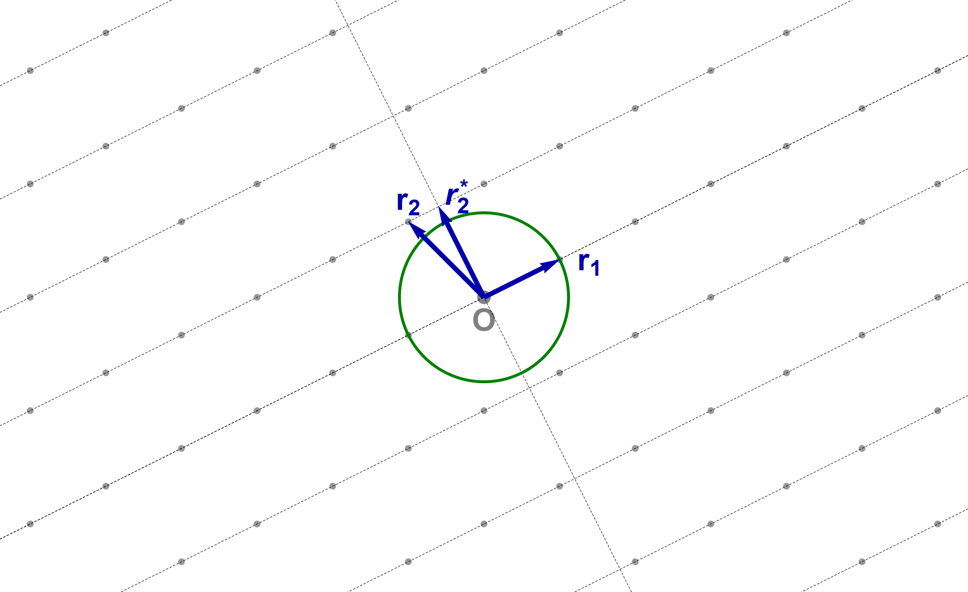
Kannan enumeration

Better bases



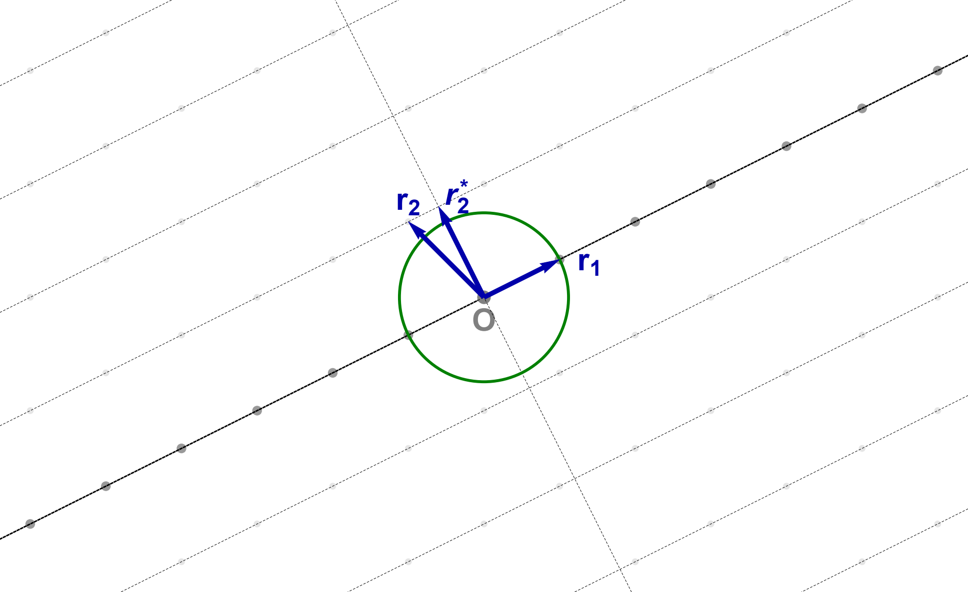
Kannan enumeration

Better bases



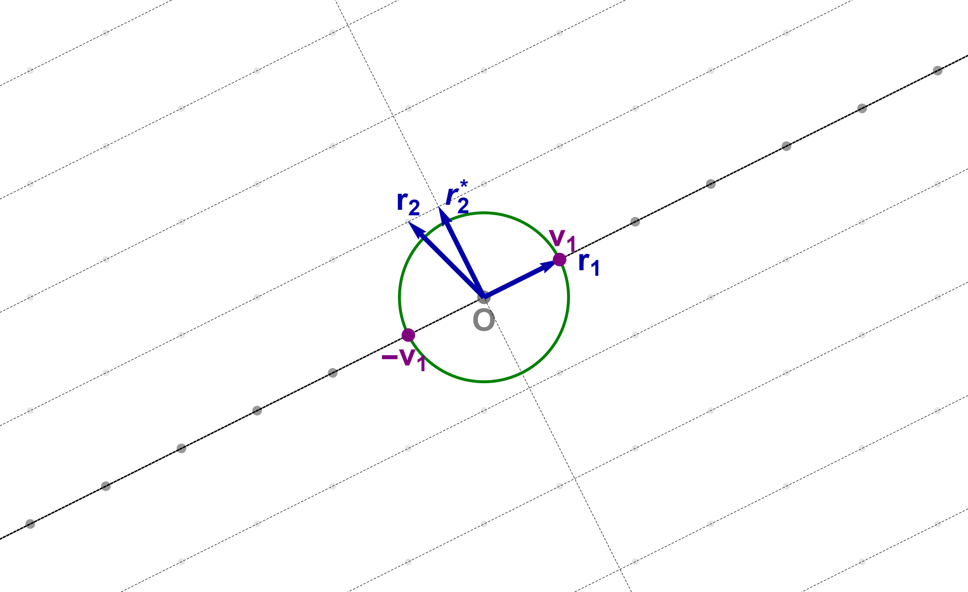
Kannan enumeration

Better bases



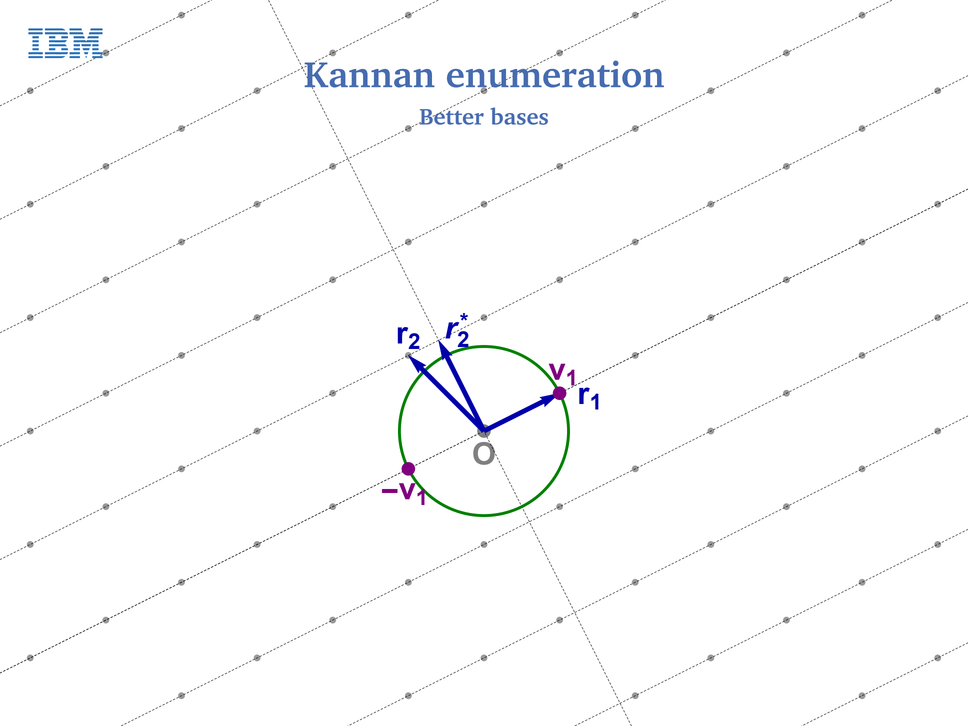
Kannan enumeration

Better bases



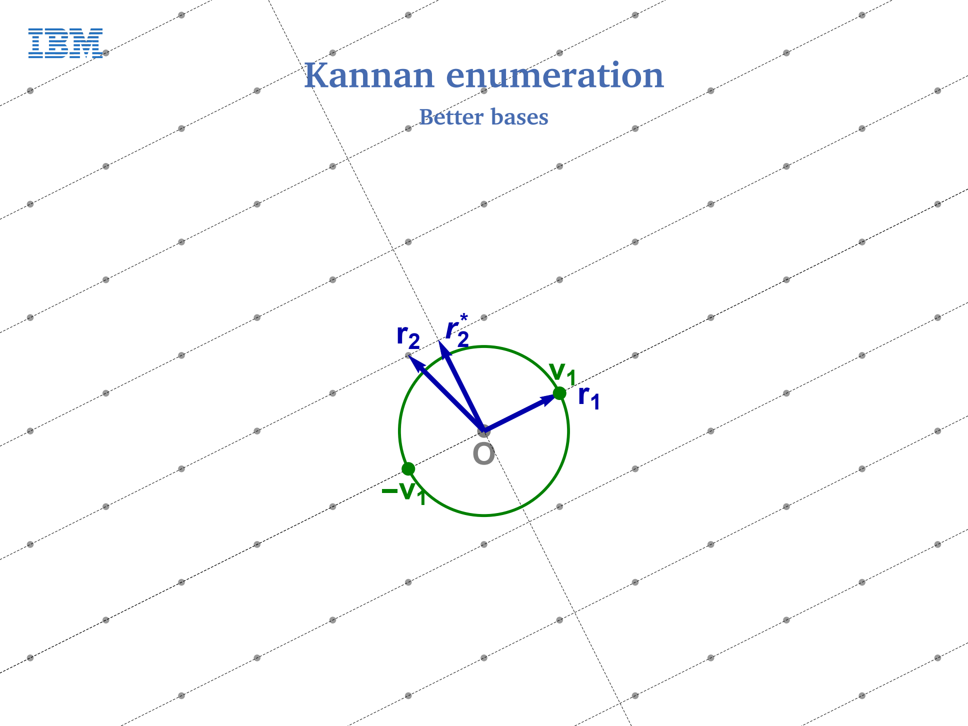
Kannan enumeration

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

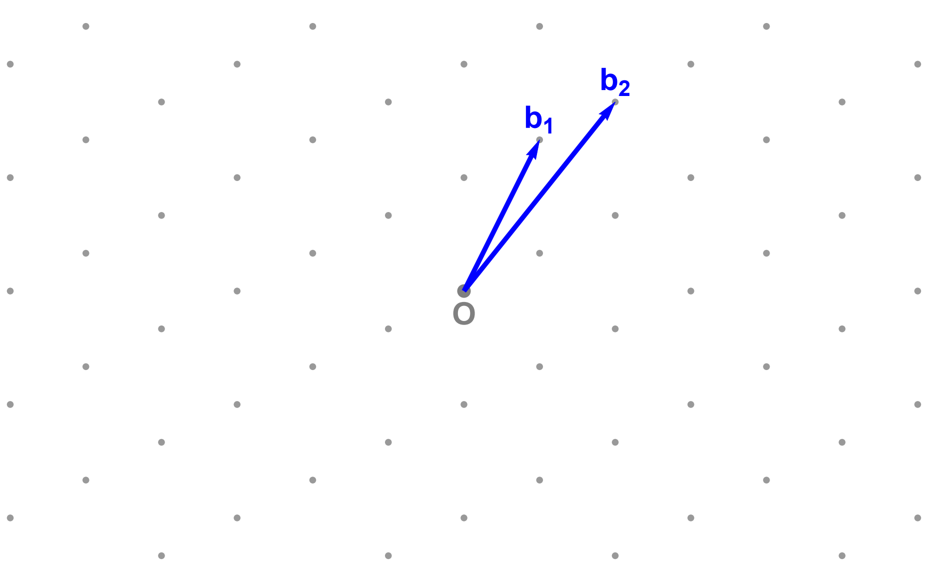
Theorem (Kannan, STOC'83)

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*

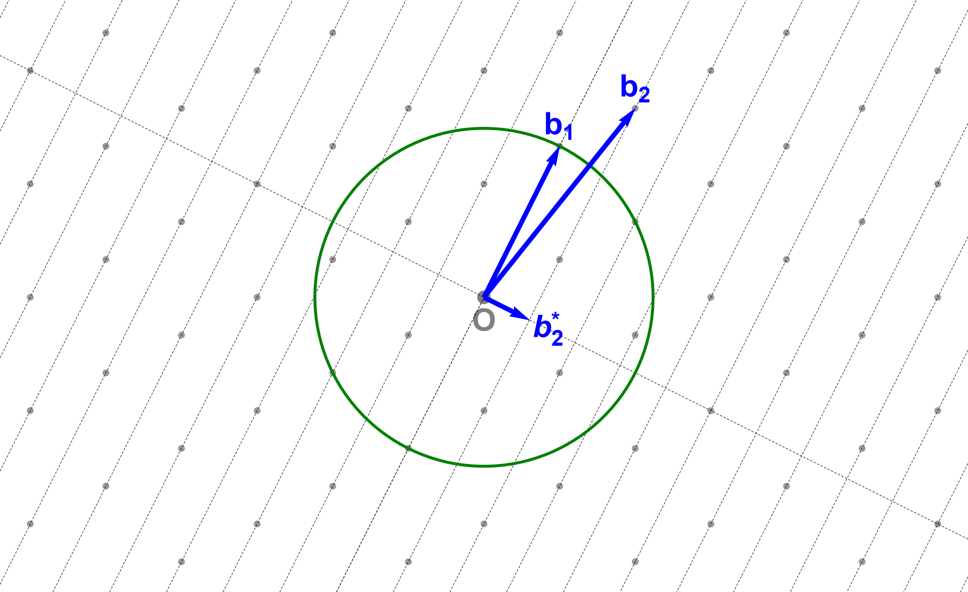
Pruned enumeration

Reducing the search space



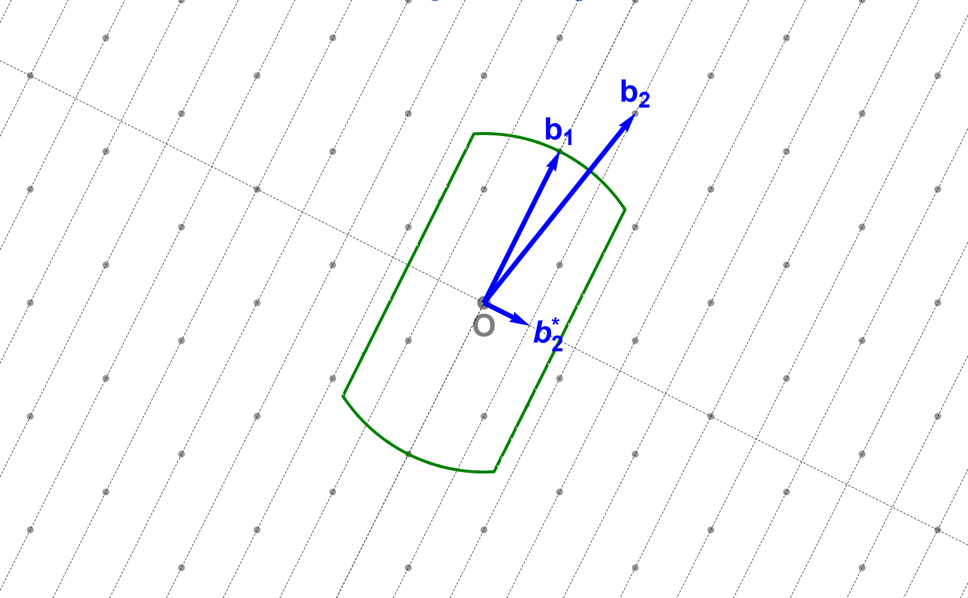
Pruned enumeration

Reducing the search space



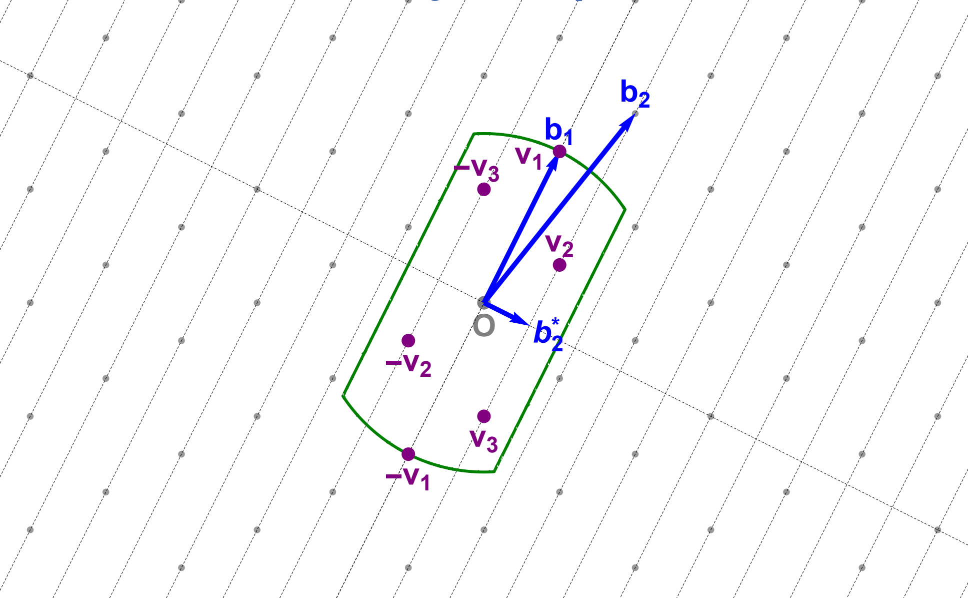
Pruned enumeration

Reducing the search space



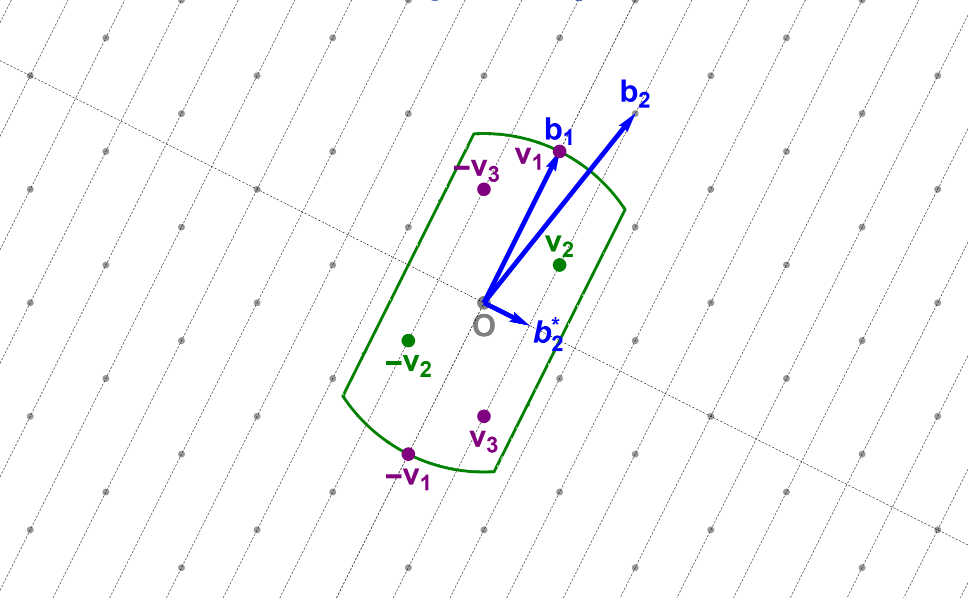
Pruned enumeration

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*

Pruned enumeration

Overview

- Pruning framework: Enumerate $\mathscr{L} \cap \mathscr{B} \cap P$ for well-chosen P
- Continuous pruning [GNR10]: P is a cylinder intersection.
- Discrete pruning [AN17]: P is a union of boxes.

Pruned enumeration

Overview

- Pruning framework: Enumerate $\mathscr{L} \cap \mathscr{B} \cap P$ for well-chosen P
- Continuous pruning [GNR10]: P is a cylinder intersection.
- Discrete pruning [AN17]: P is a union of boxes.

"We now know continuous pruning and discrete pruning [...] but a theoretical asymptotical comparison is not easy. Can a combination of both, or another form of pruning be more efficient?"

— Aono–Nguyen, *EUROCRYPT'17*

Outline

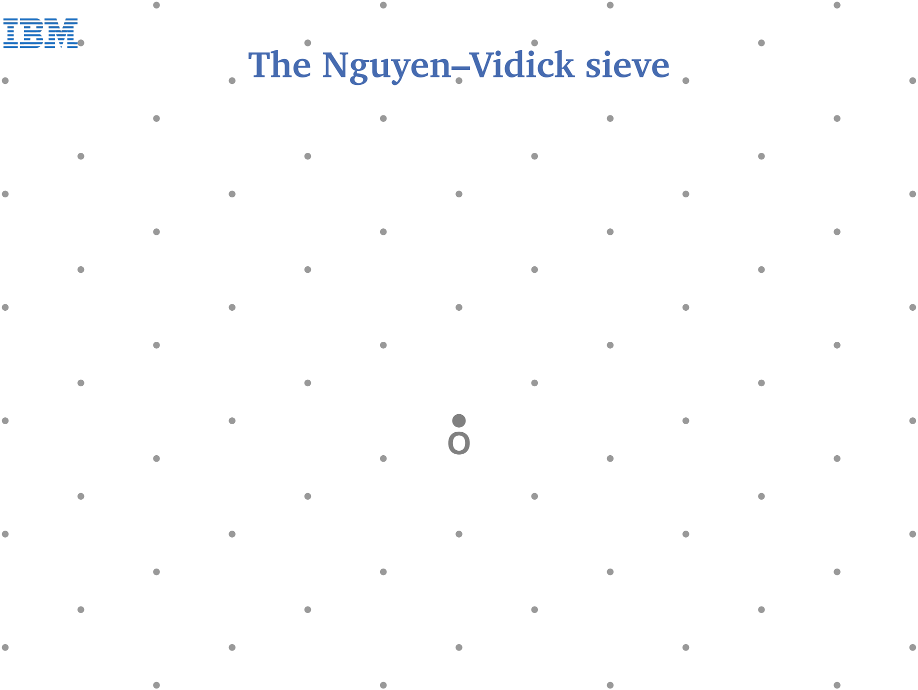
Lattices
- Basics
- Cryptography

Enumeration algorithms
- Fincke–Pohst enumeration
- Kannan enumeration
- Pruned enumeration

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

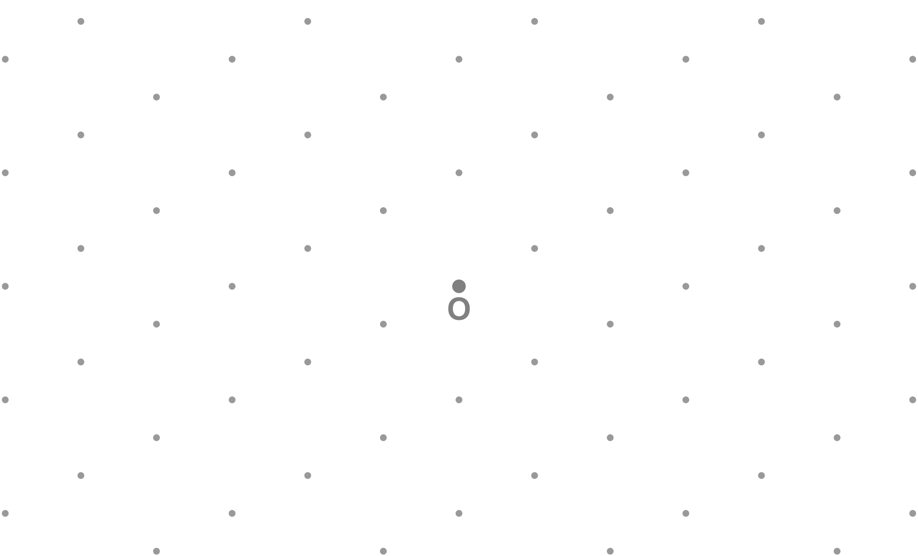
Practical comparison

The Nguyen–Vidick sieve



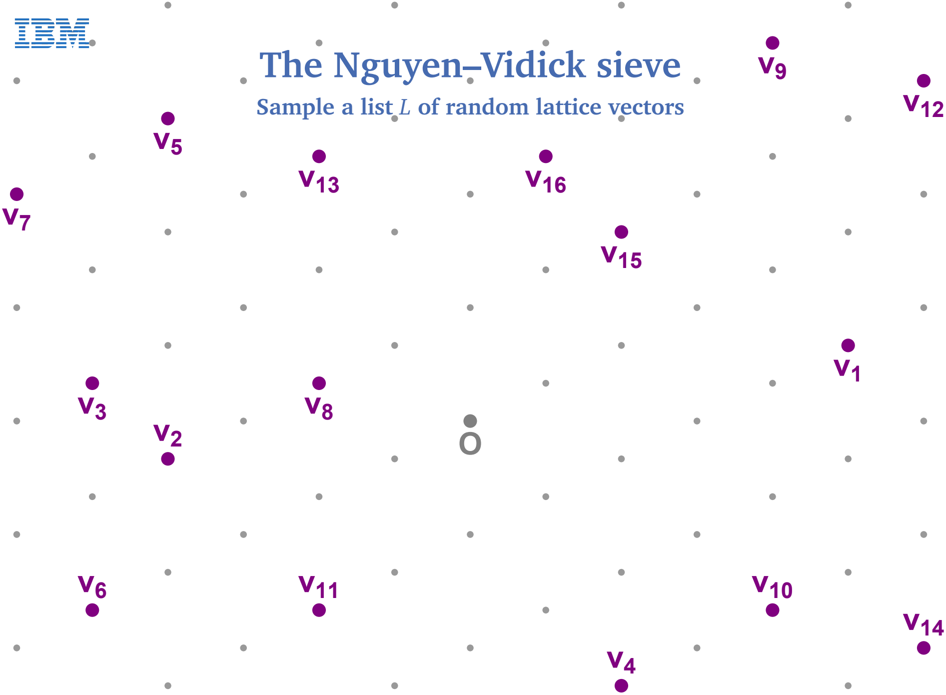
The Nguyen–Vidick sieve

Sample a list L of random lattice vectors



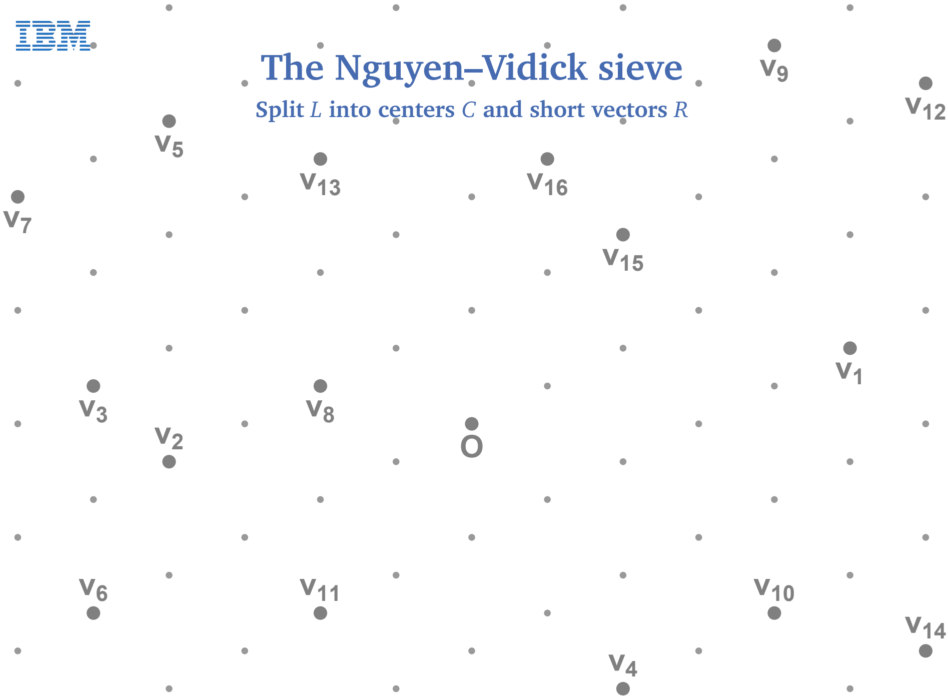
The Nguyen–Vidick sieve

Sample a list L of random lattice vectors



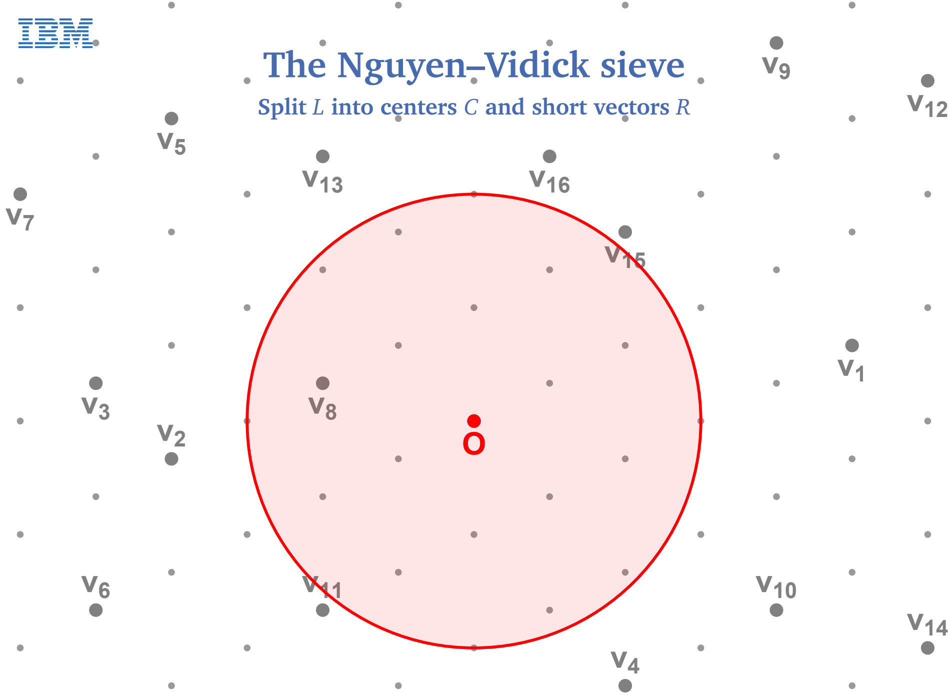
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



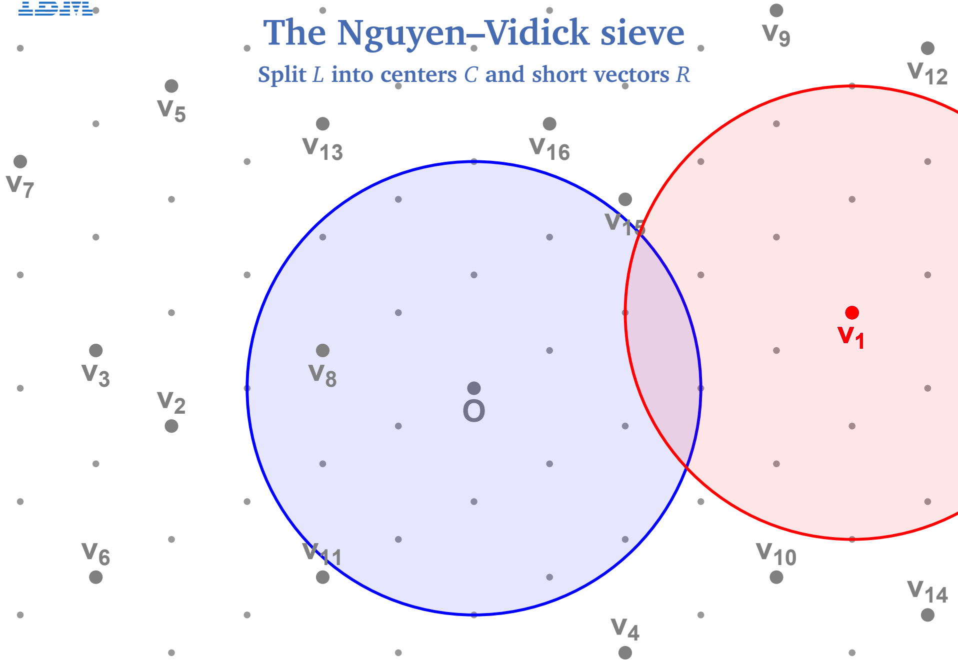
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



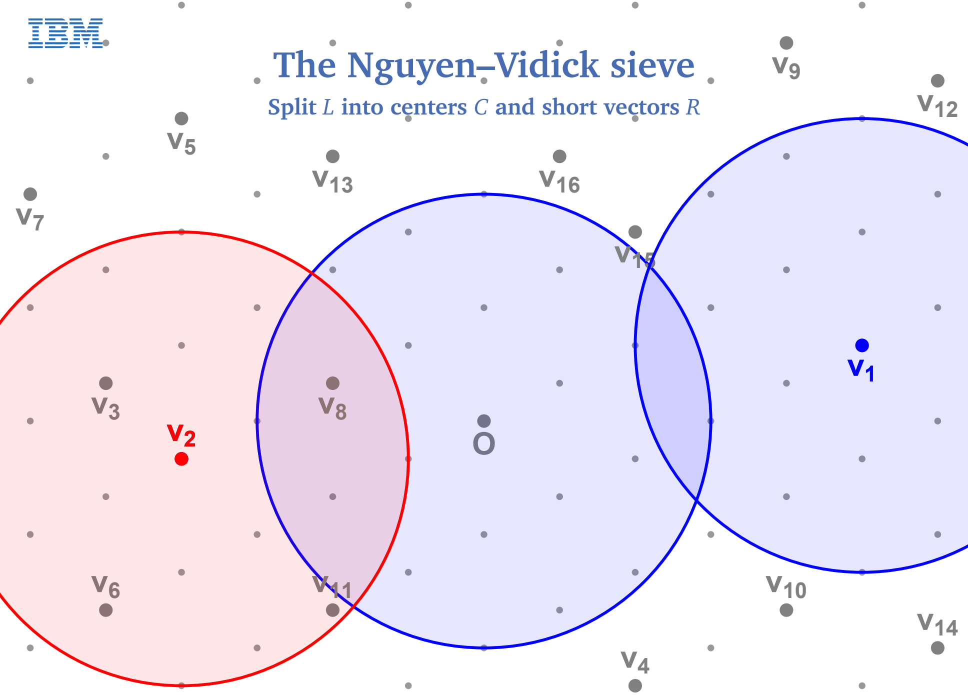
The Nguyen–Vidick sieve

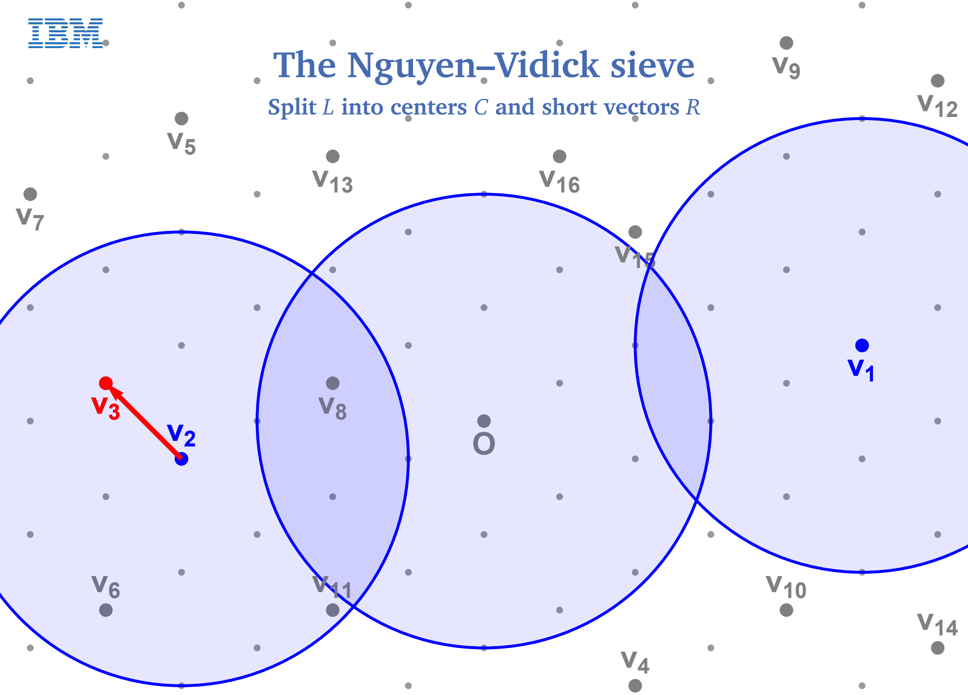
Split L into centers C and short vectors R



The Nguyen–Vidick sieve

Split L into centers C and short vectors R

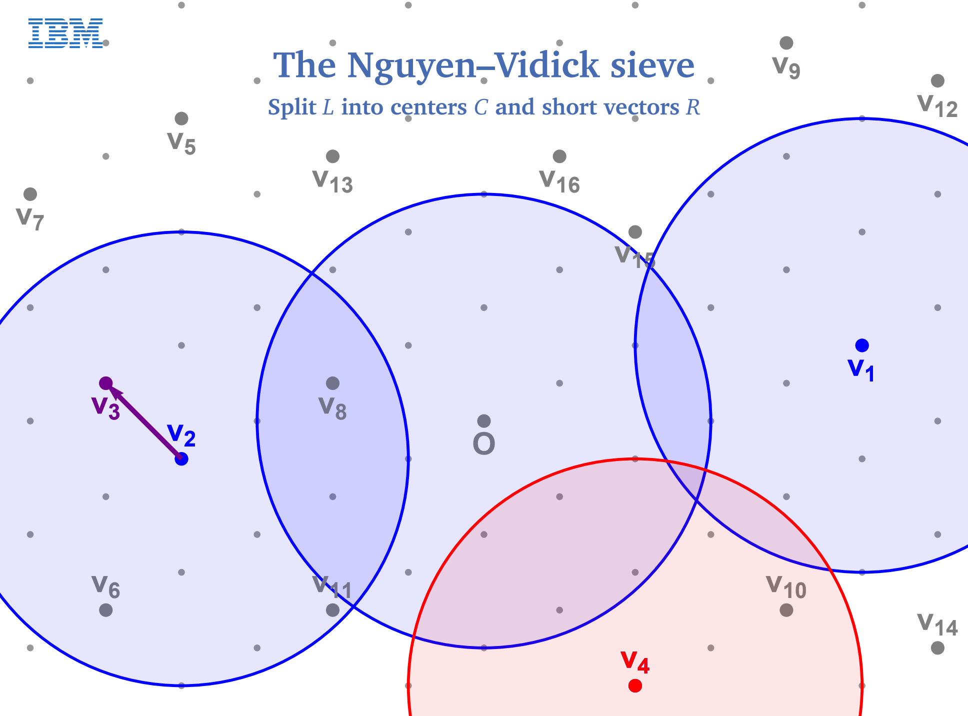


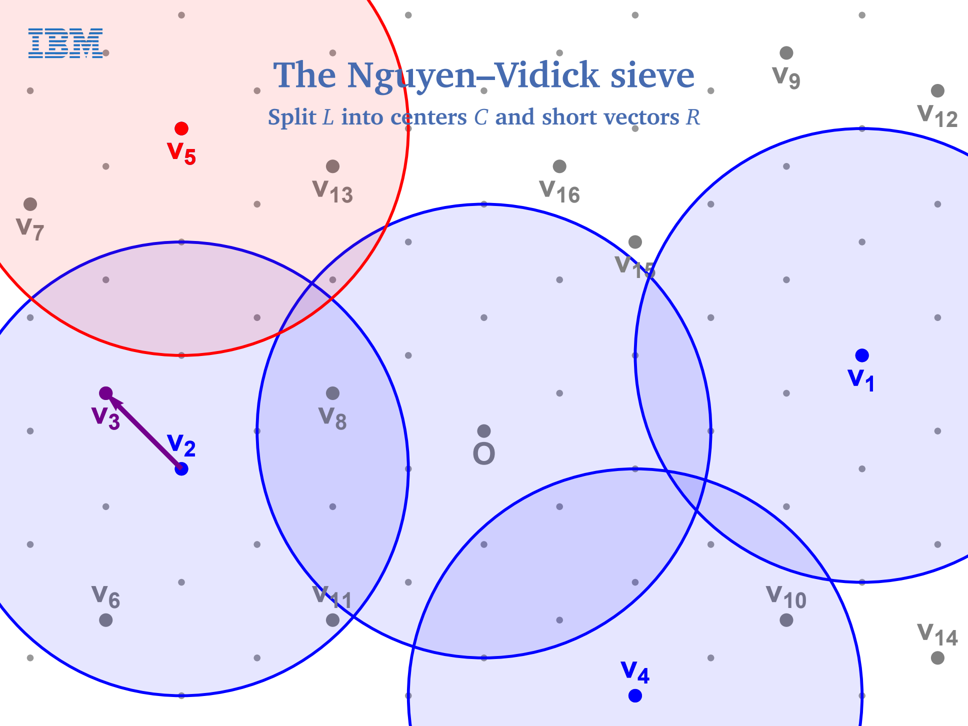


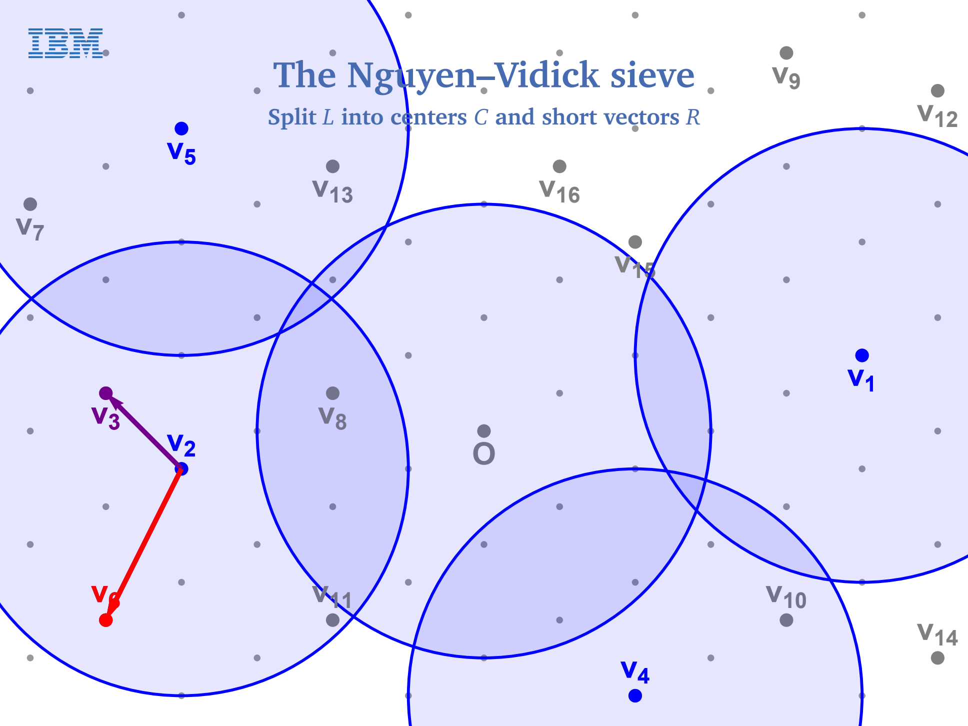
The Nguyen–Vidick sieve
Split L into centers C and short vectors R
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
v_{10}
v_{11}
v_{12}
v_{13}
v_{14}
v_{15}
v_{16}
O

The Nguyen–Vidick sieve

Split L into centers C and short vectors R

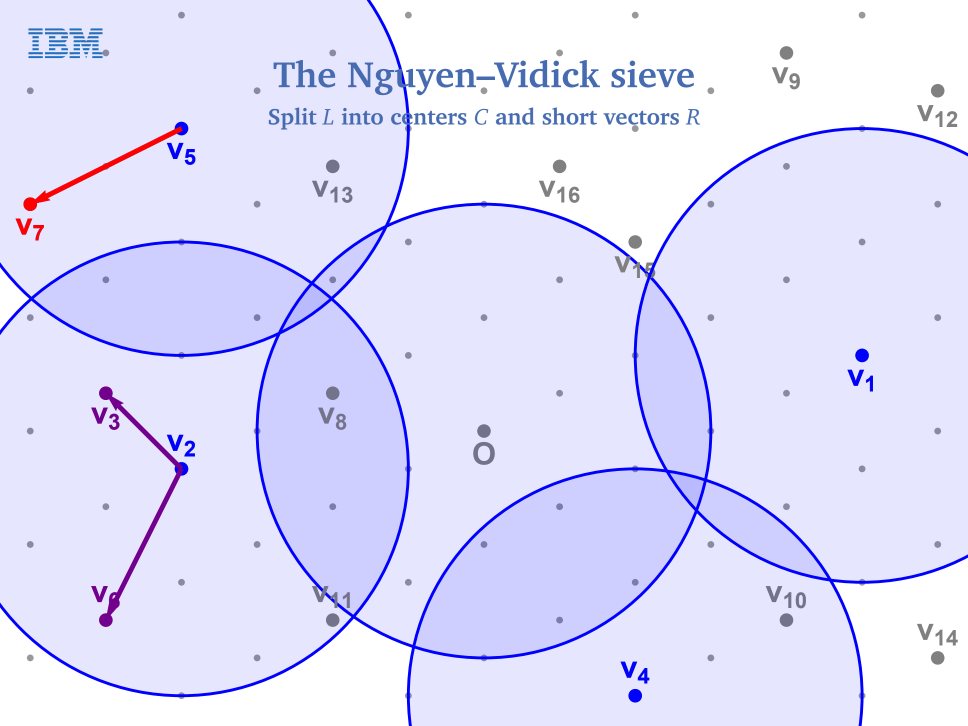



The Nguyen–Vidick sieve
Split L into centers C and short vectors R
v5
v13
v16
v9
v12
v7
v15
v1
v3
v8
v2
O
v6
v11
v10
v14
v4


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

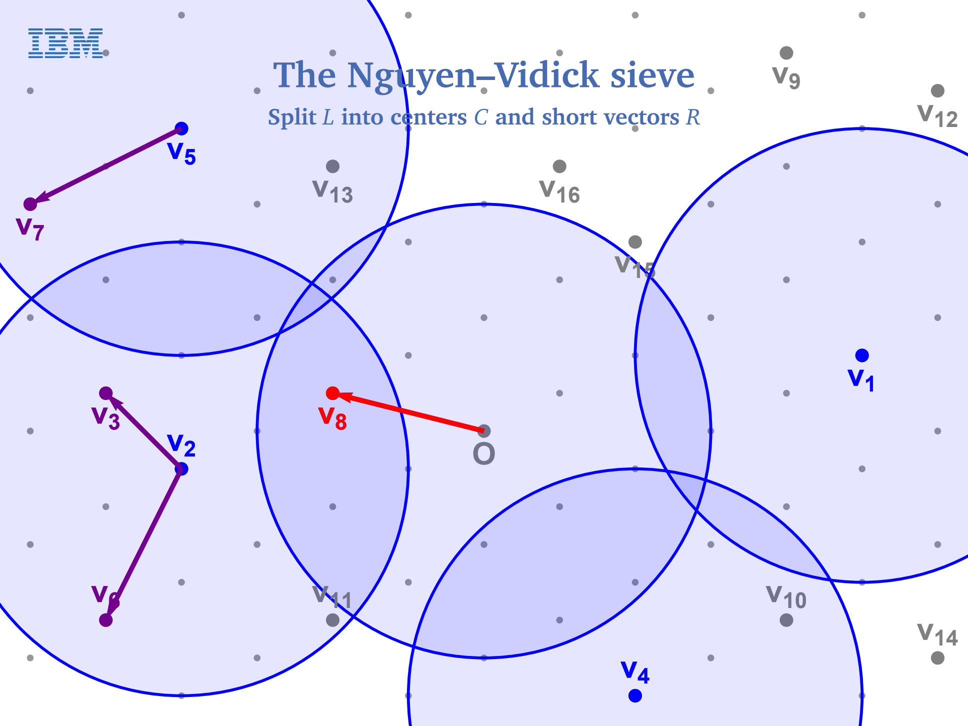
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



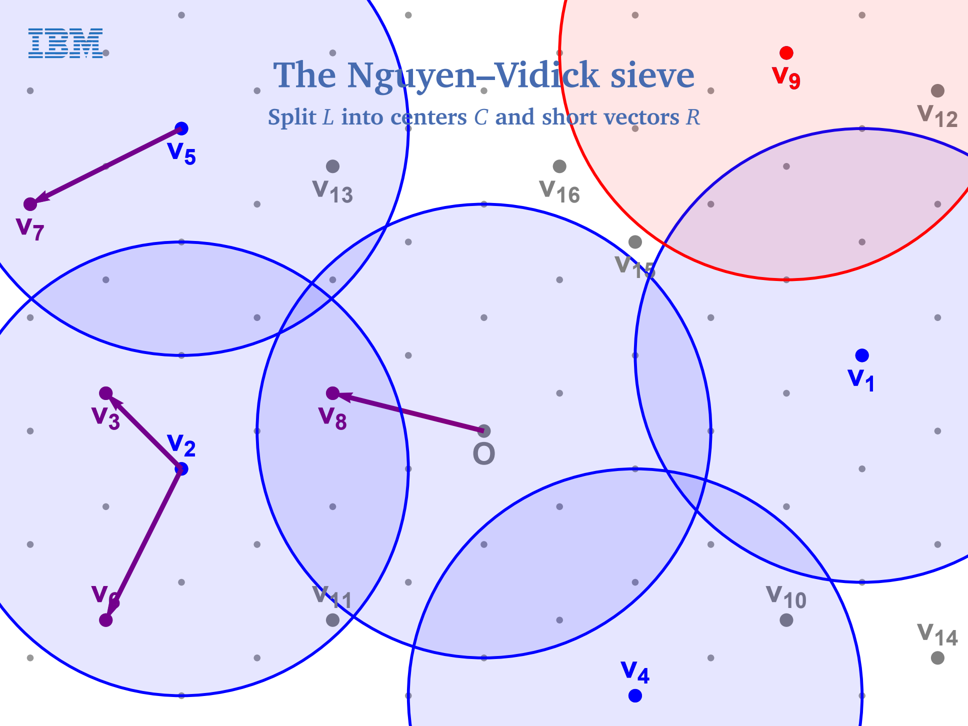
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



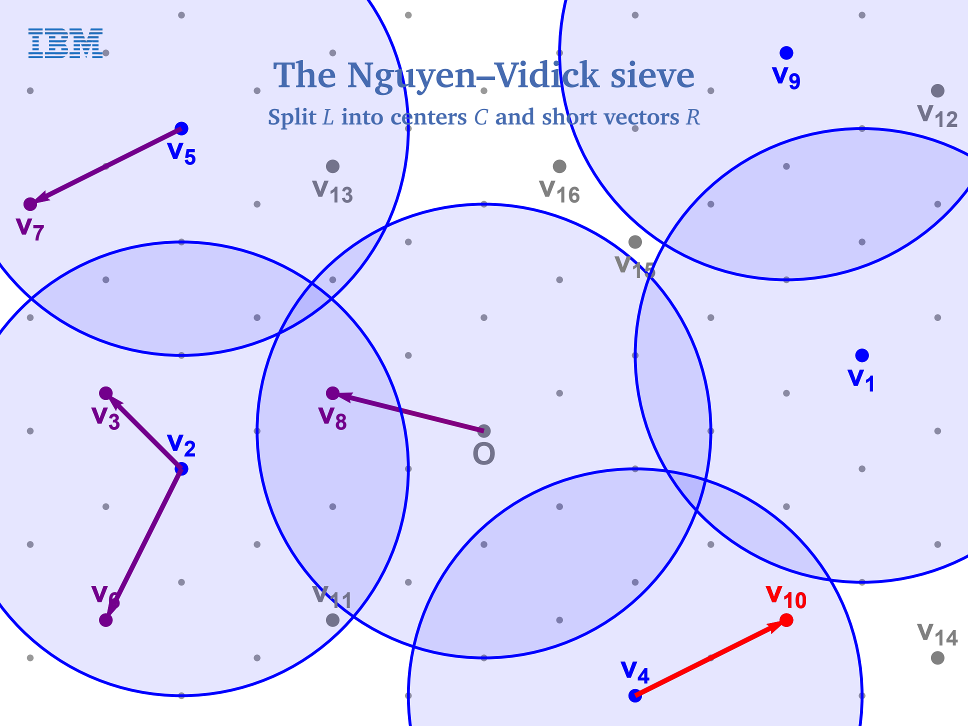
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



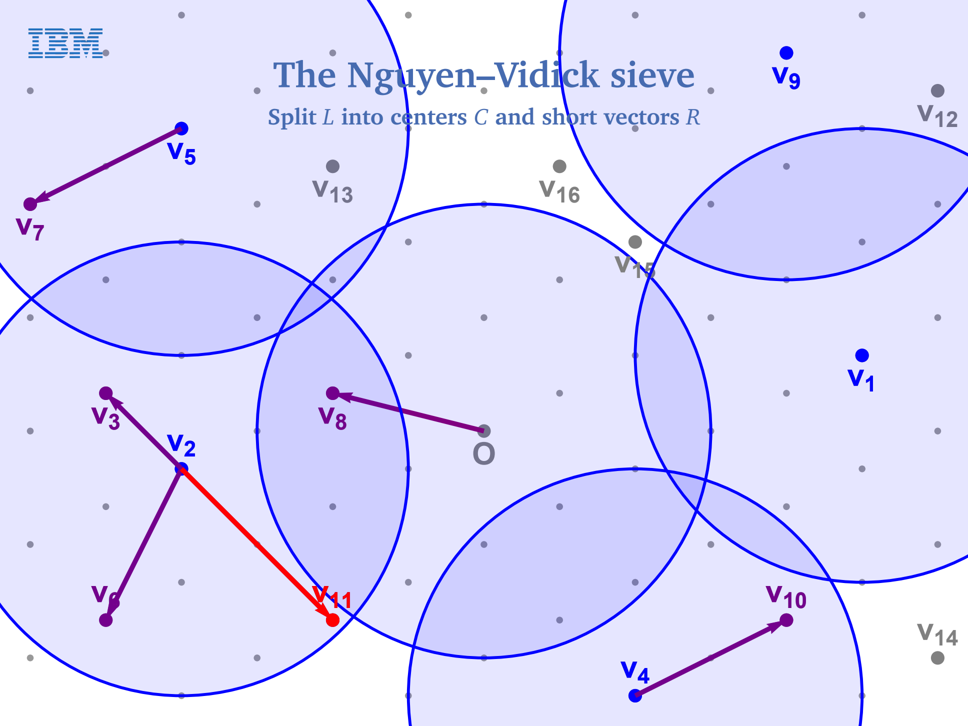
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



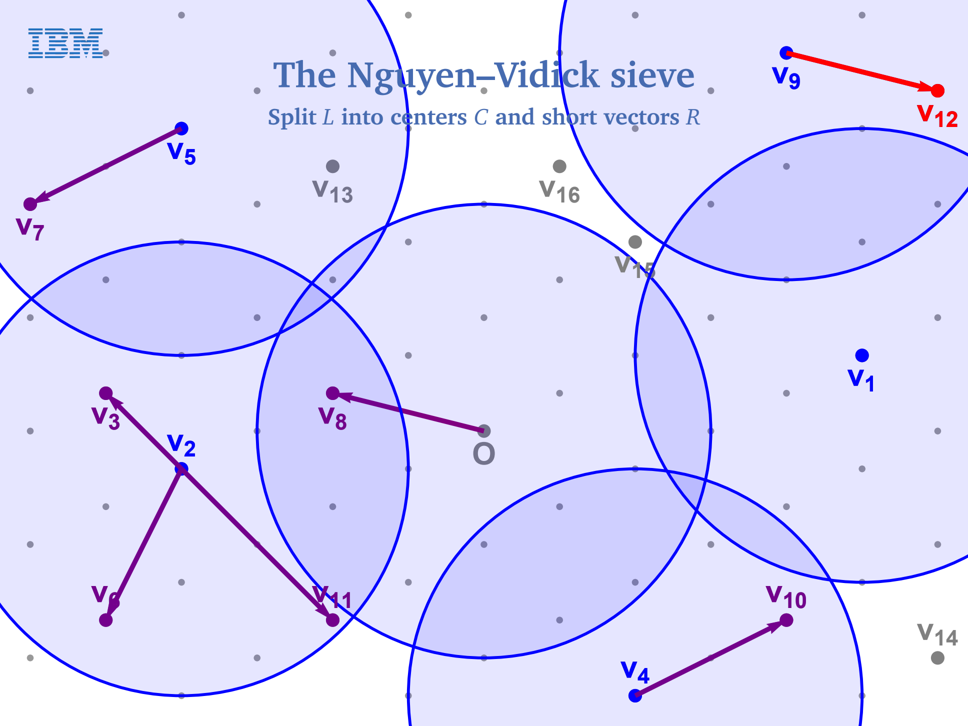
The Nguyen–Vidick sieve

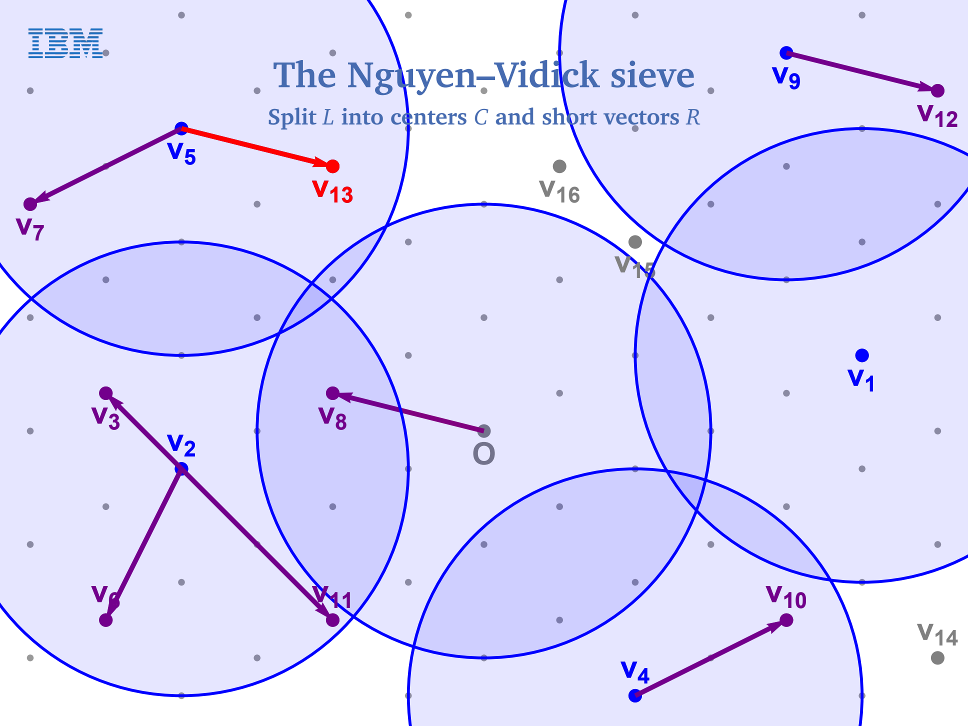
Split L into centers C and short vectors R



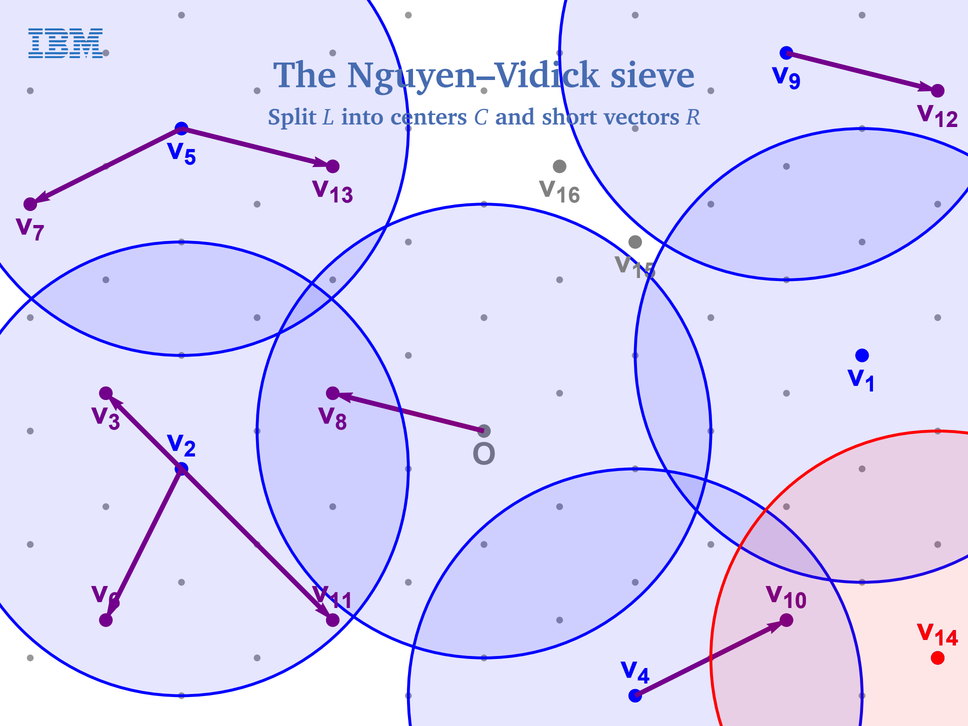
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



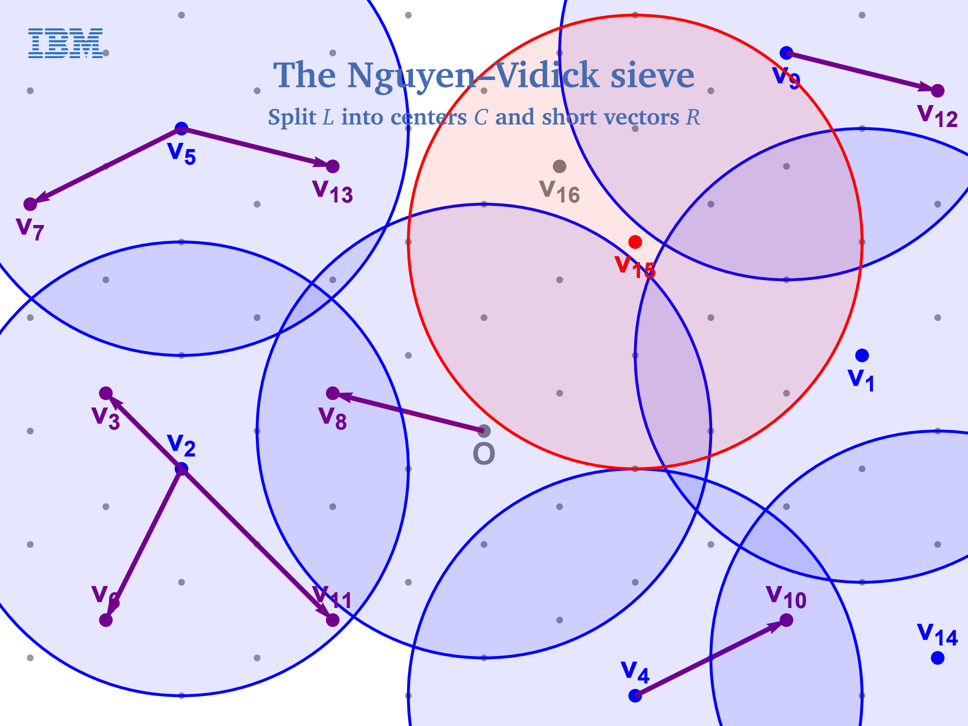


The Nguyen–Vidick sieve
Split L into centers C and short vectors R
v_1
v_2
v_3
v_4
v_5
v_7
v_8
v_9
v_{10}
v_{11}
v_{12}
v_{13}
v_{14}
v_{15}
v_{16}
O


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

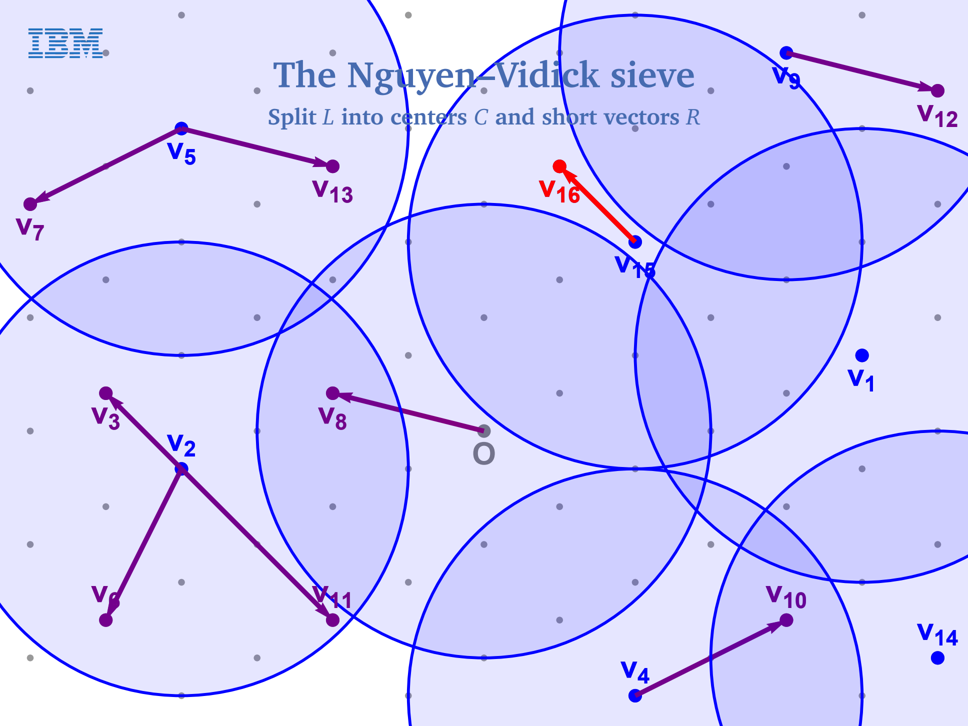
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



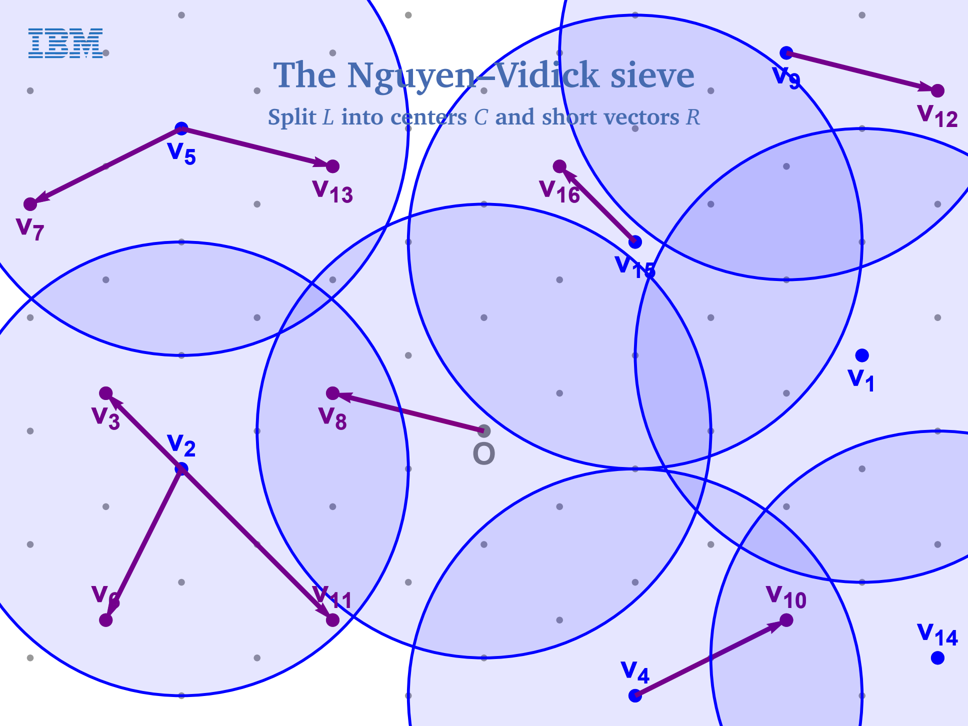
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



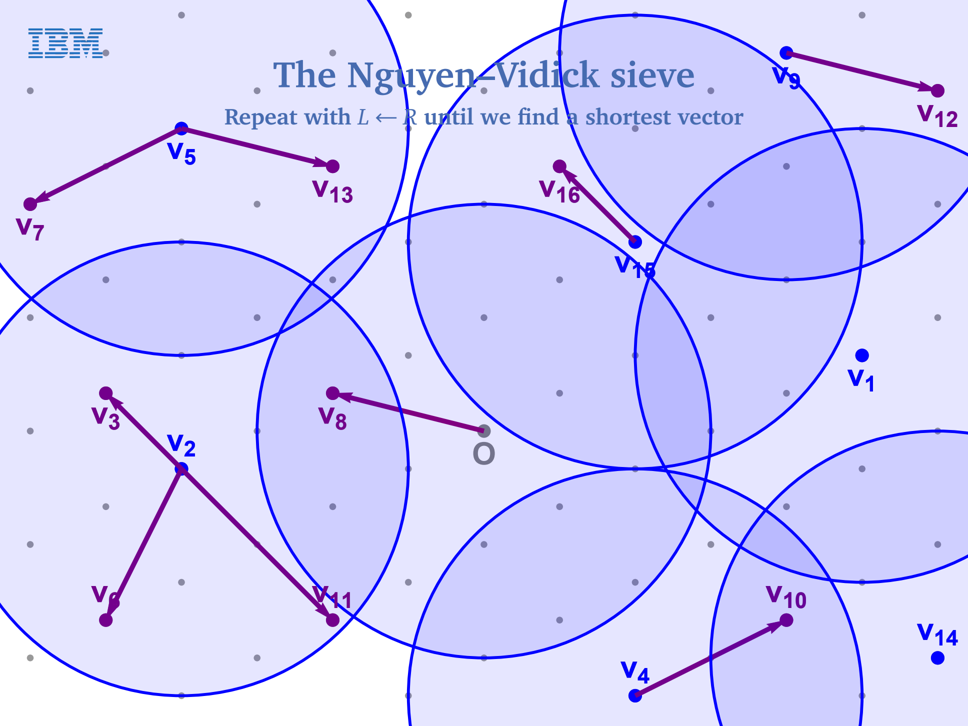
The Nguyen–Vidick sieve

Split L into centers C and short vectors R



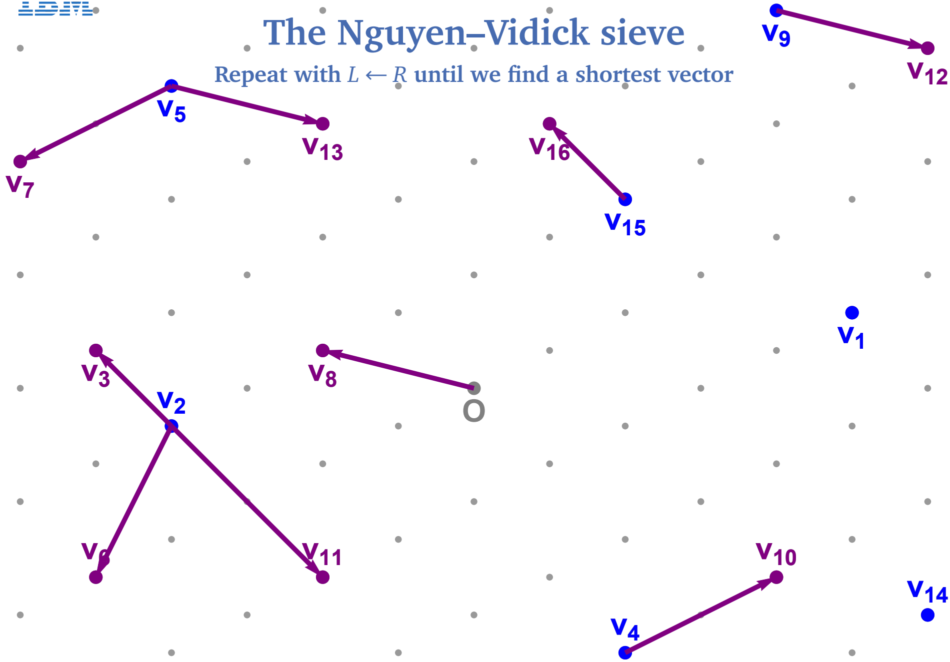
The Nguyen–Vidick sieve

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



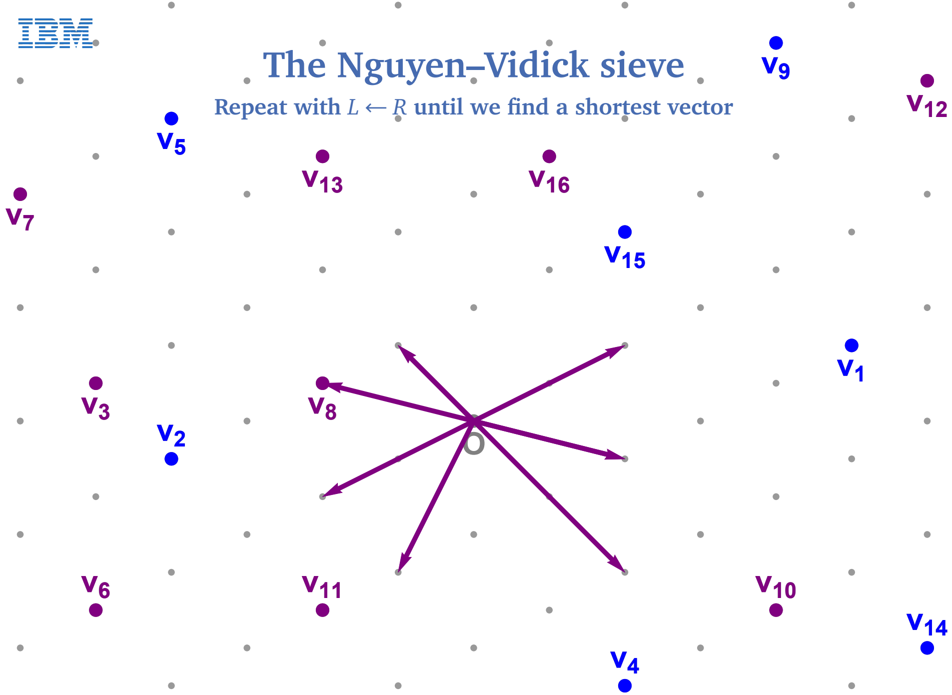
The Nguyen–Vidick sieve

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



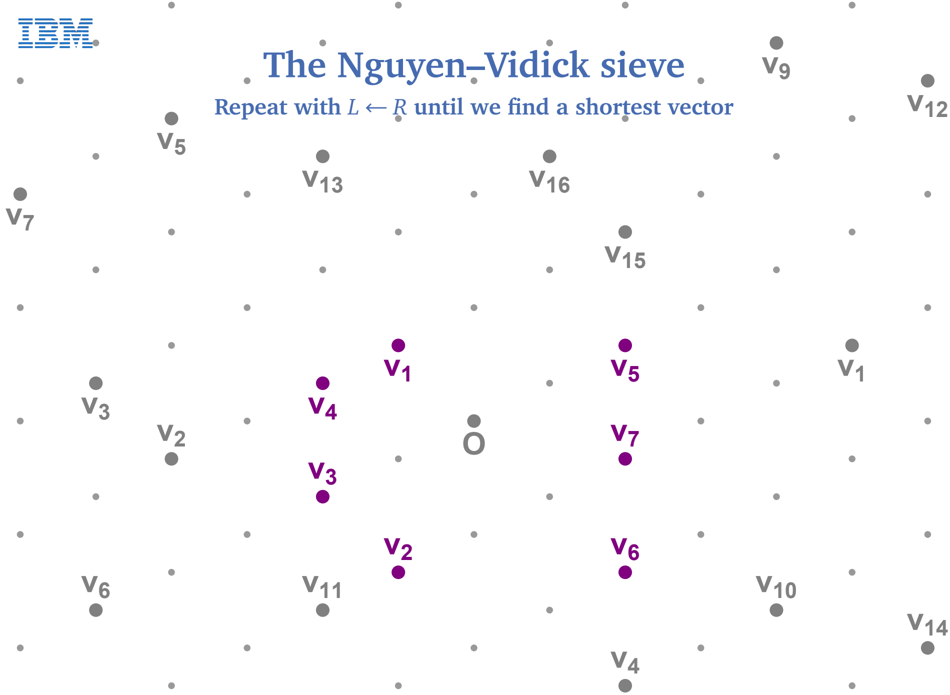
The Nguyen–Vidick sieve

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



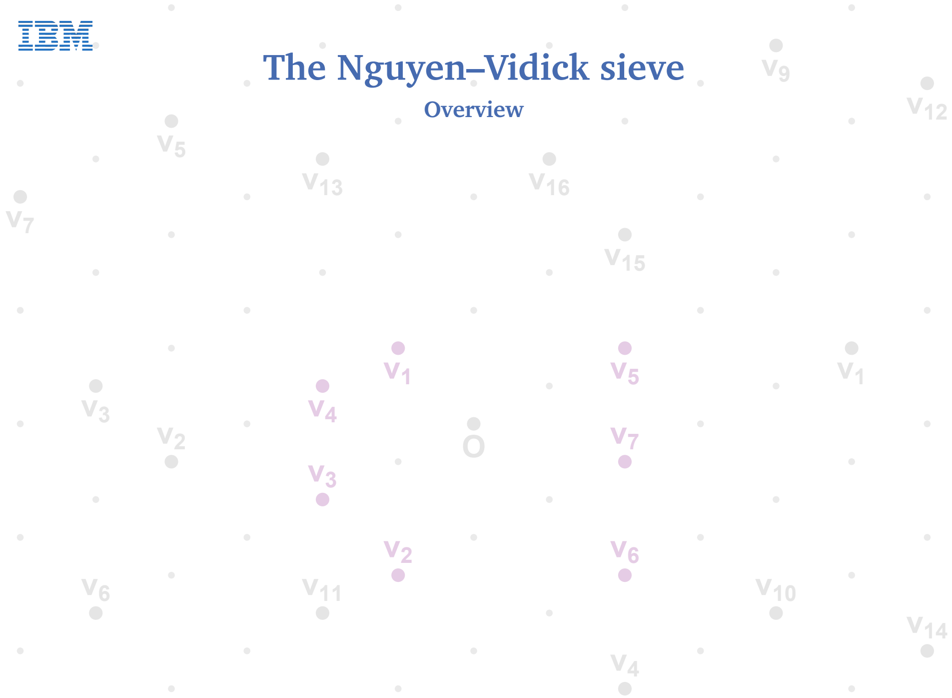
The Nguyen–Vidick sieve

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

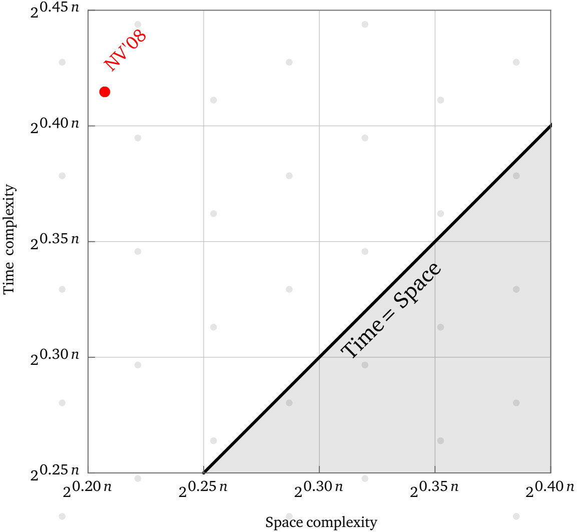
- Space complexity: $\sqrt{4/3}^n \approx 2^{0.21n+o(n)}$ vectors
  - Need $\sqrt{4/3}^n$ vectors to cover all corners of $\mathbb{R}^n$
- Time complexity: $(4/3)^n \approx 2^{0.42n+o(n)}$
  - Comparing a target vector to all centers: $2^{0.21n+o(n)}$
  - Repeating this for each list vector: $2^{0.21n+o(n)}$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

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

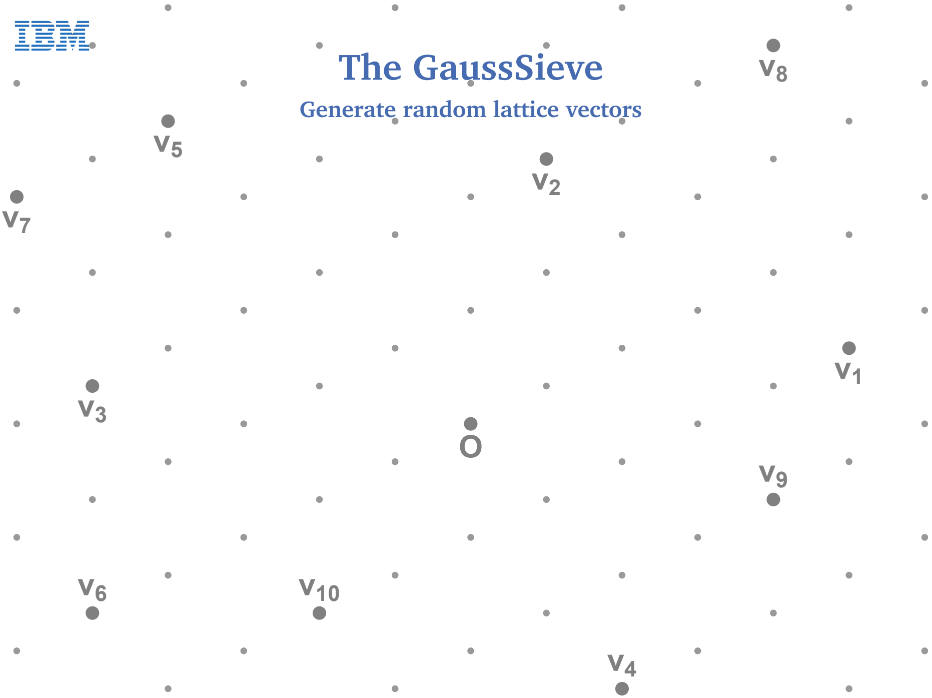


The GaussSieve

Generate random lattice vectors

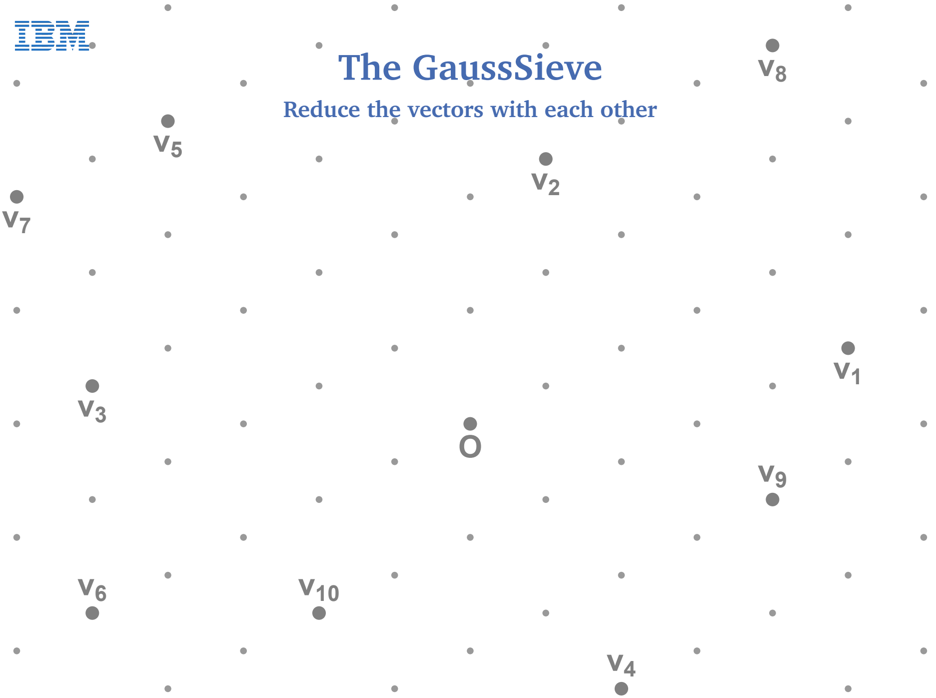
The GaussSieve

Generate random lattice vectors



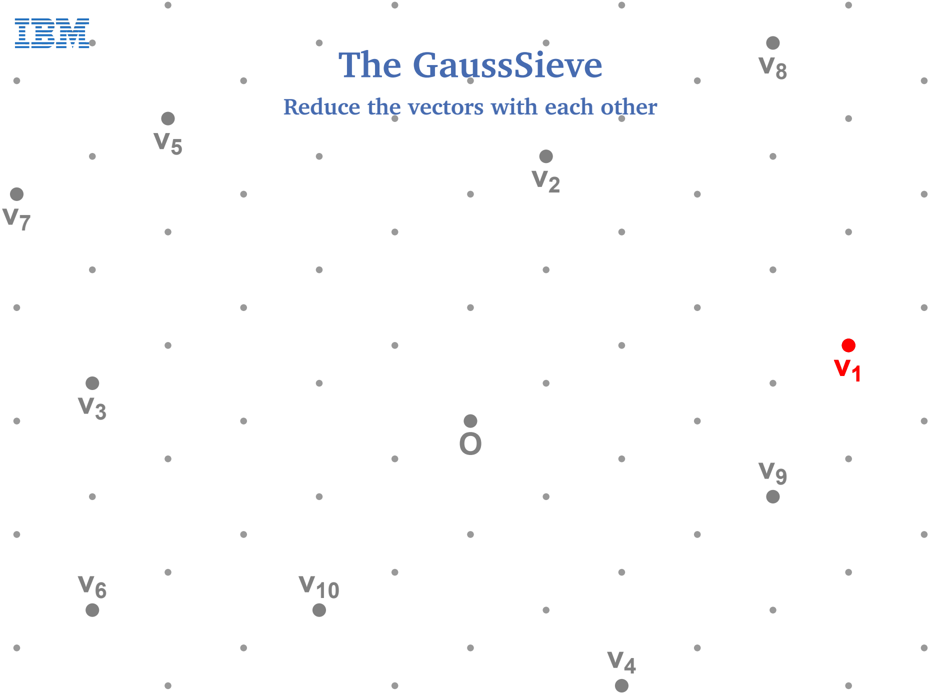
The GaussSieve

Reduce the vectors with each other



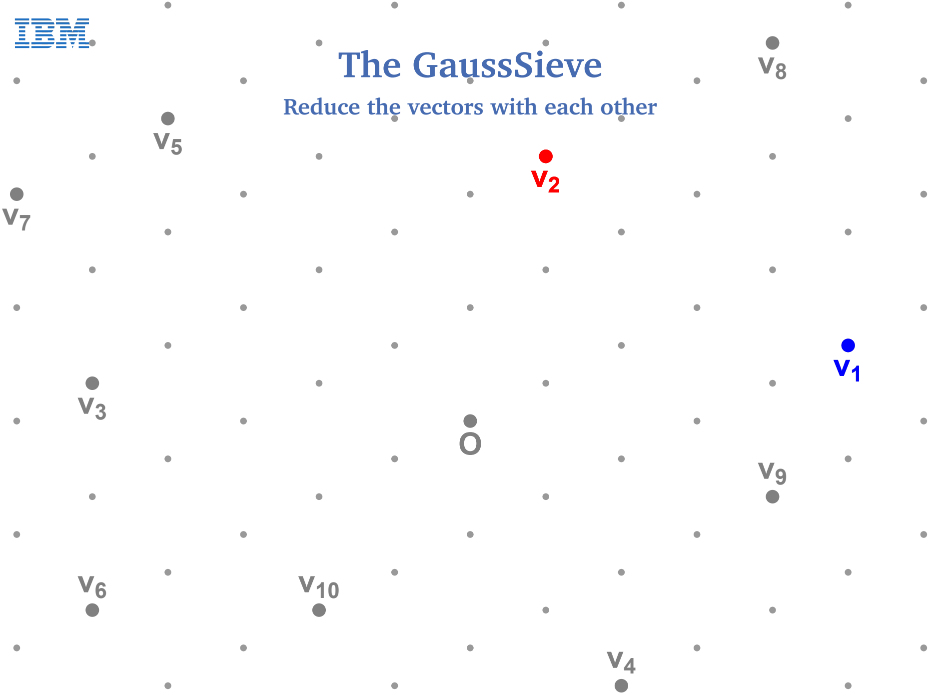
The GaussSieve

Reduce the vectors with each other



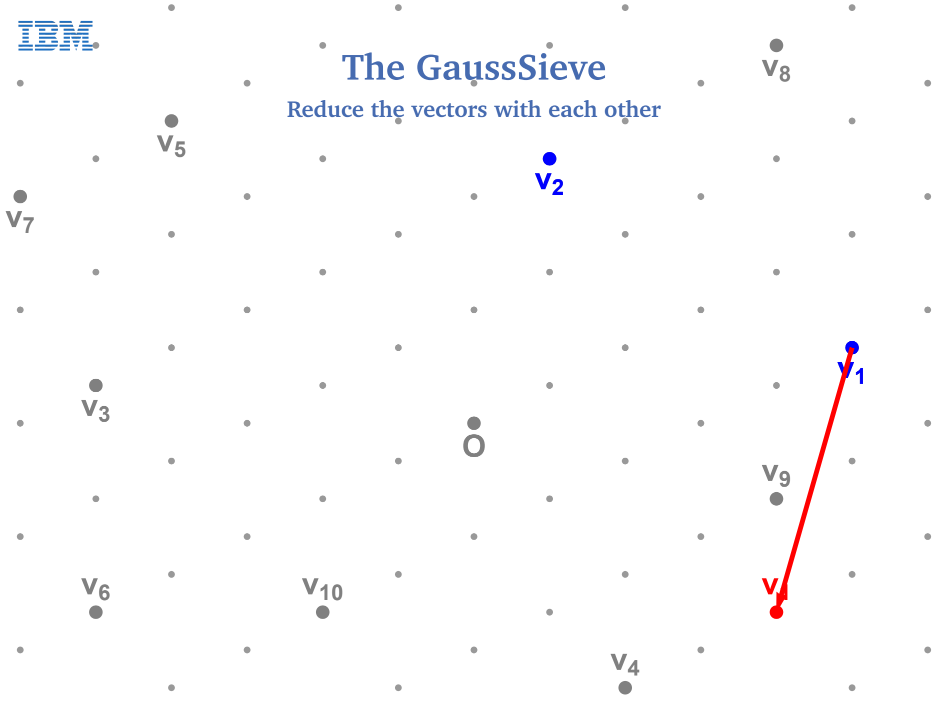
The GaussSieve

Reduce the vectors with each other



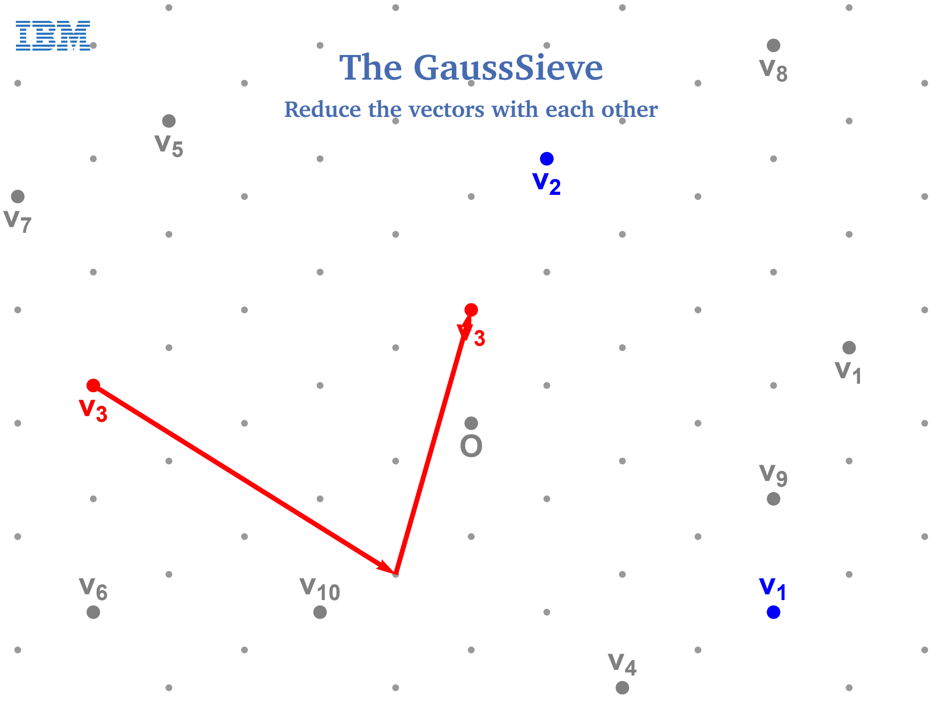
The GaussSieve

Reduce the vectors with each other



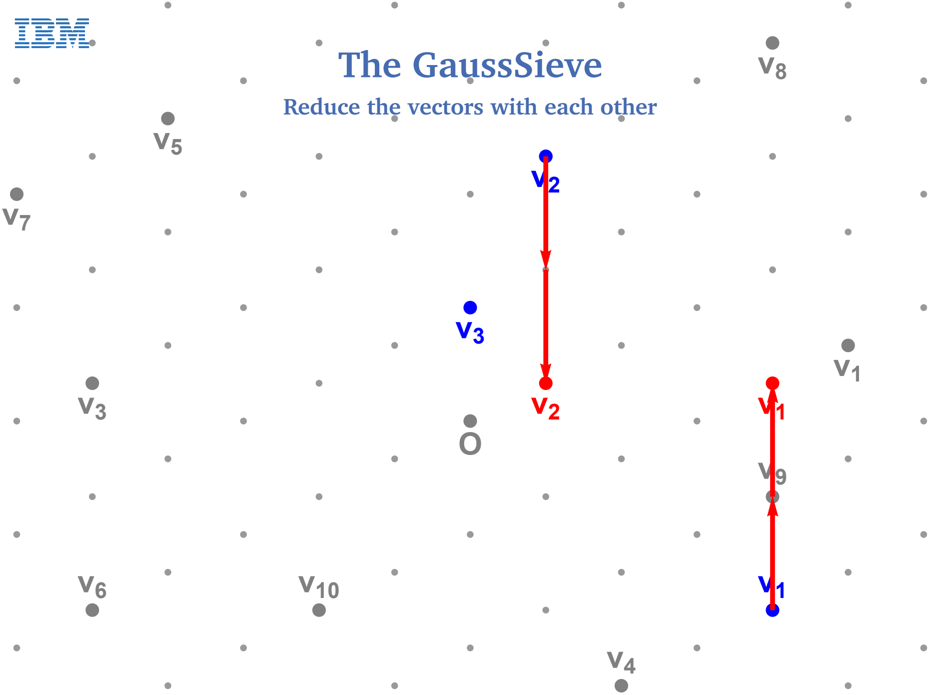
The GaussSieve

Reduce the vectors with each other



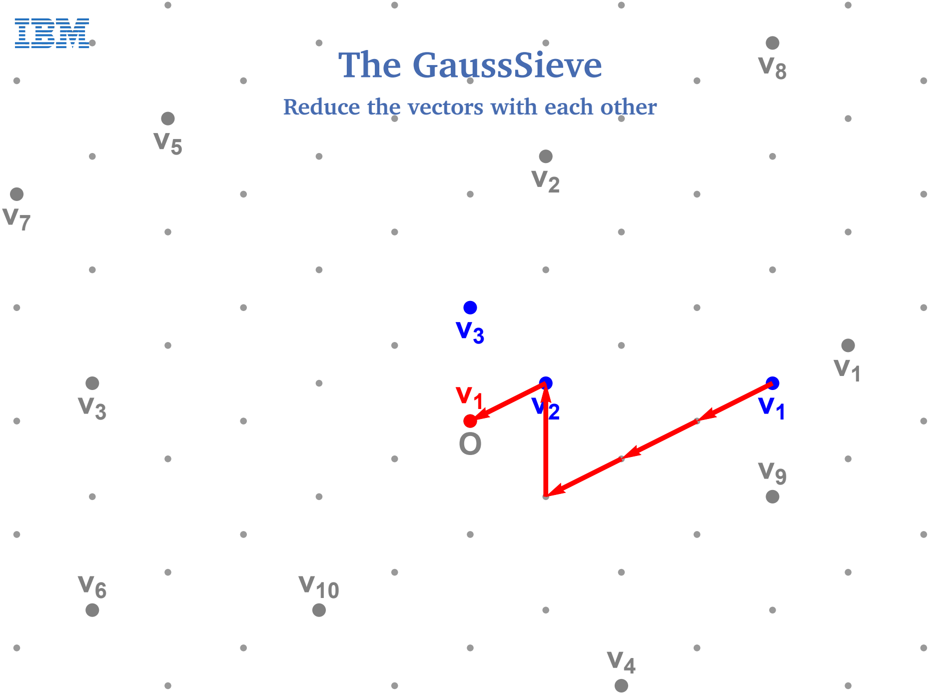
The GaussSieve

Reduce the vectors with each other



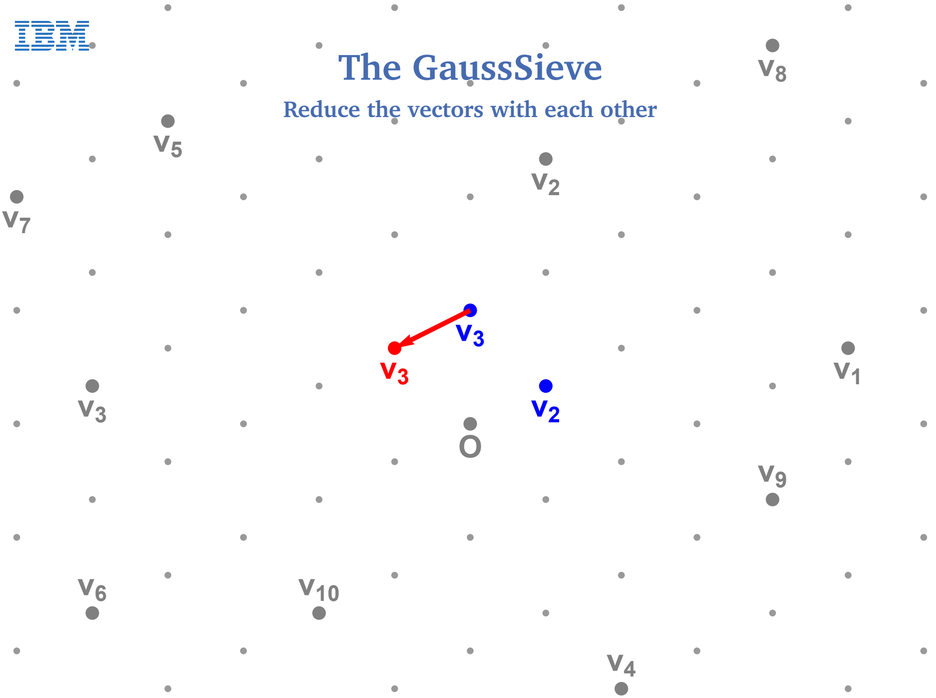
The GaussSieve

Reduce the vectors with each other



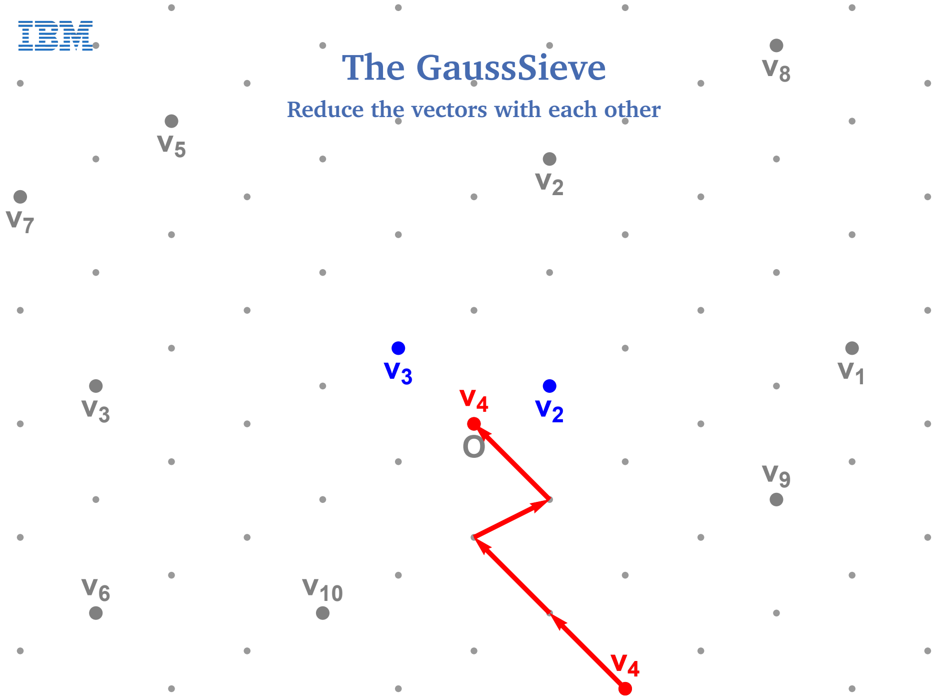
The GaussSieve

Reduce the vectors with each other



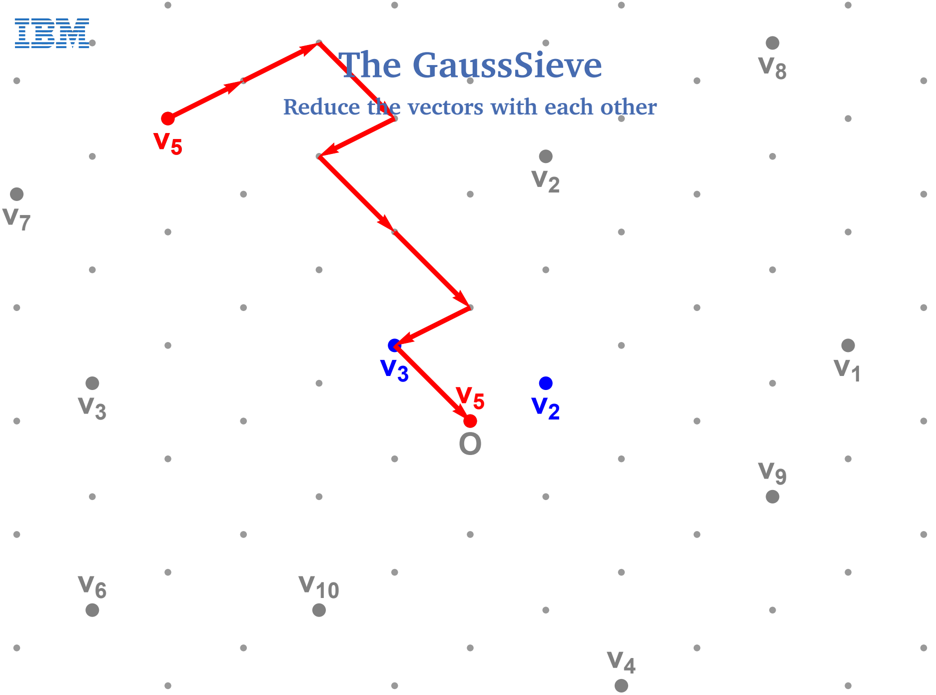
The GaussSieve

Reduce the vectors with each other



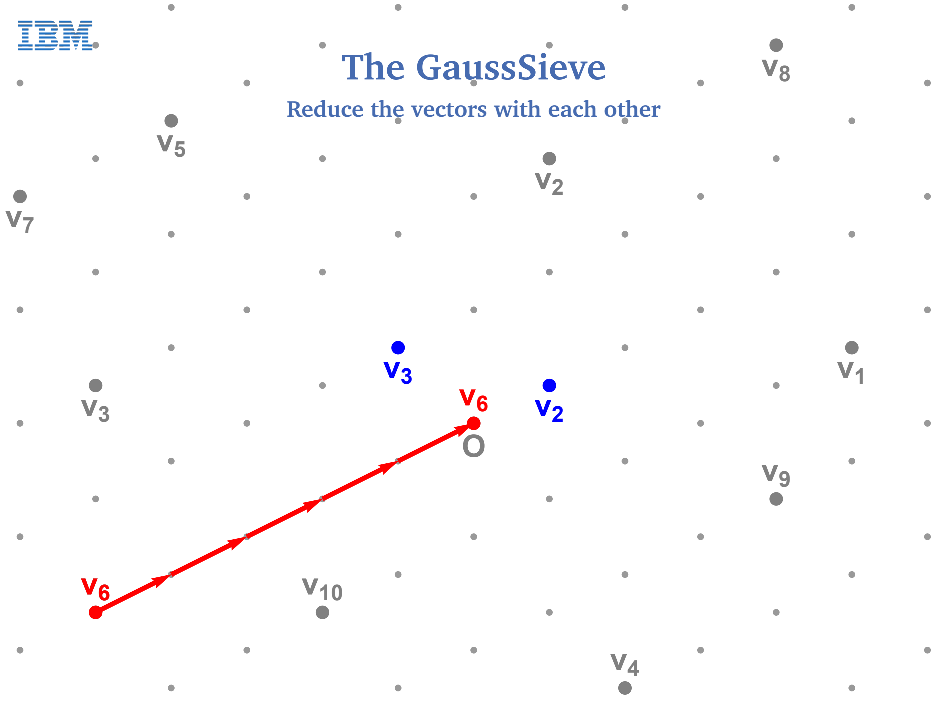
The GaussSieve

Reduce the vectors with each other



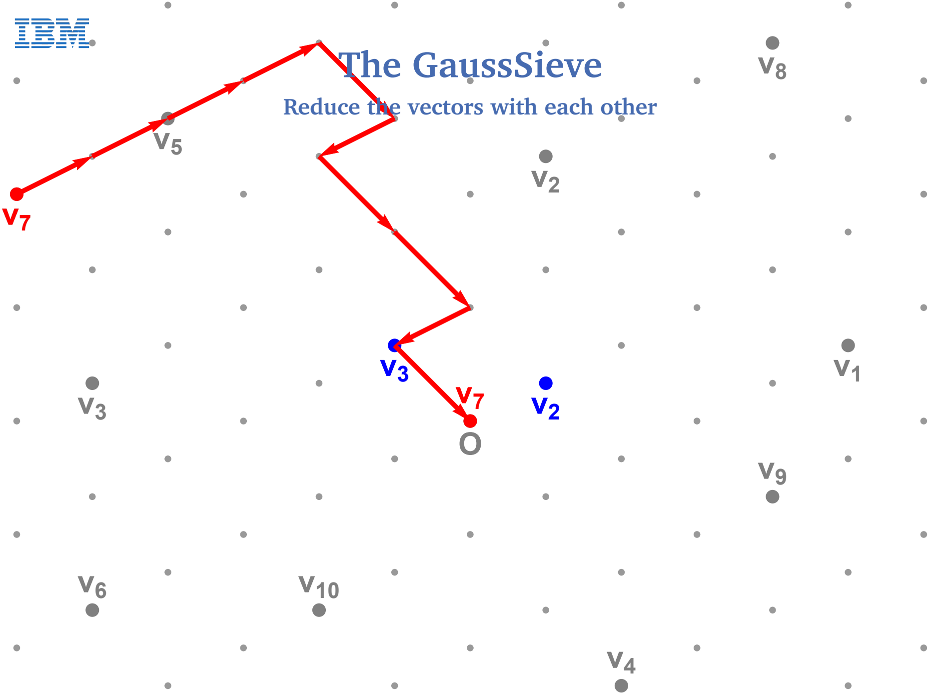
The GaussSieve

Reduce the vectors with each other



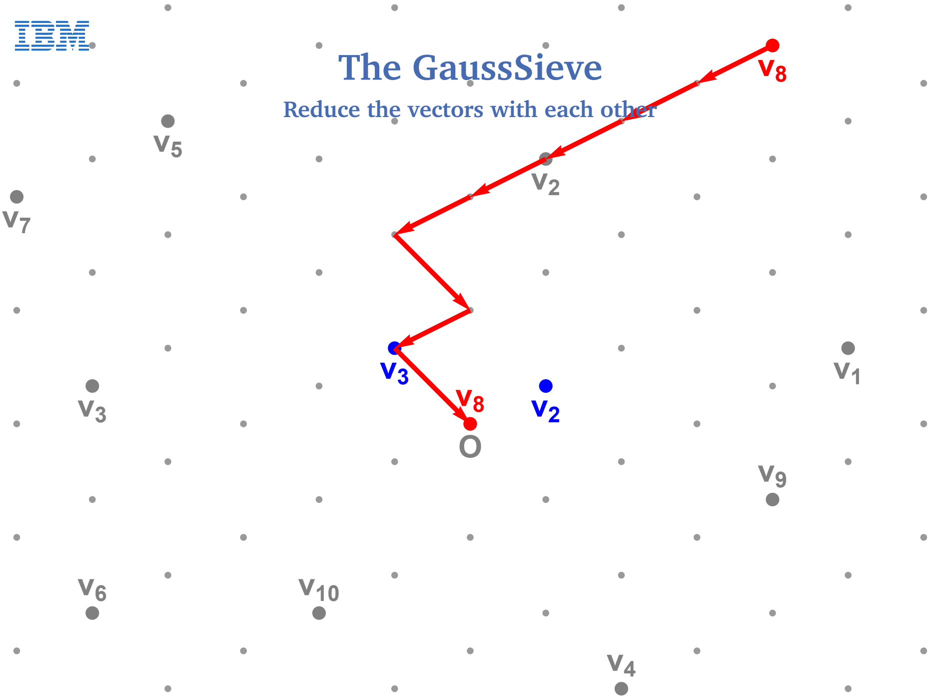
The GaussSieve

Reduce the vectors with each other



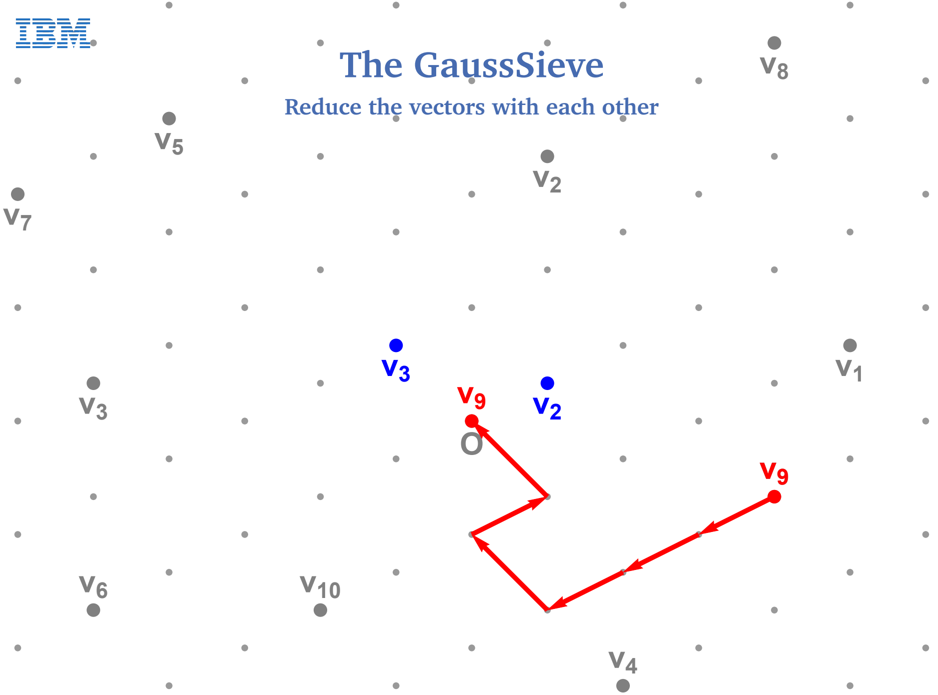
The GaussSieve

Reduce the vectors with each other



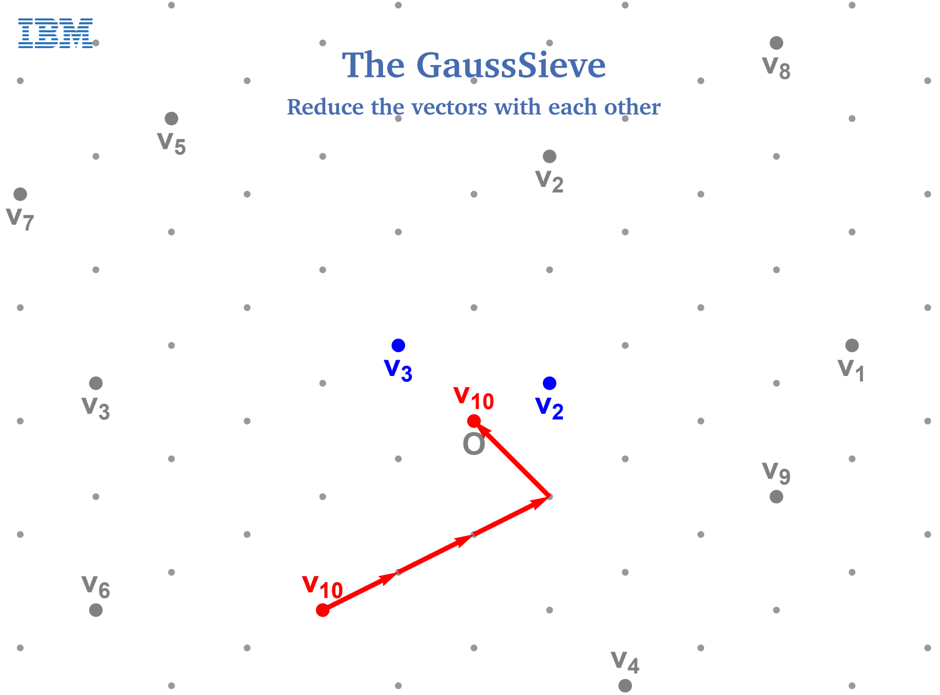
The GaussSieve

Reduce the vectors with each other



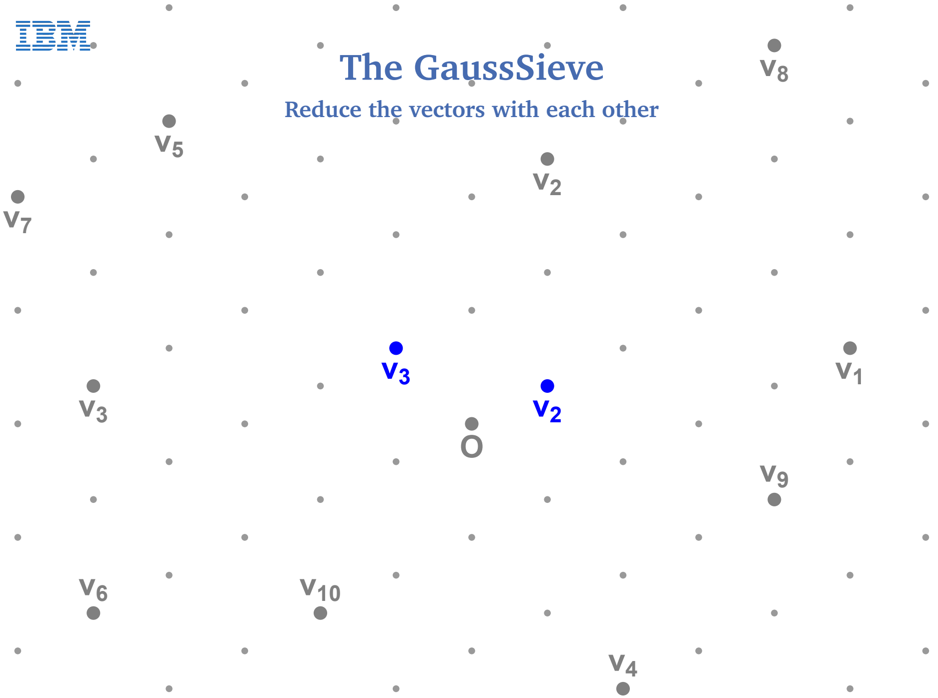
The GaussSieve

Reduce the vectors with each other



The GaussSieve

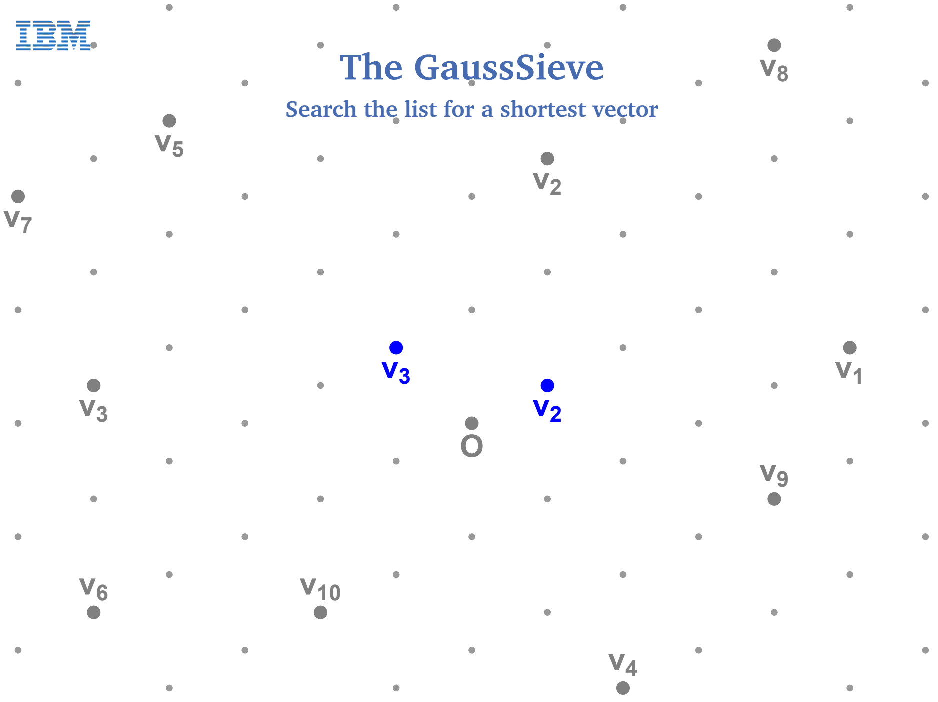
Reduce the vectors with each other



IBM

The GaussSieve

Search the list for a shortest vector

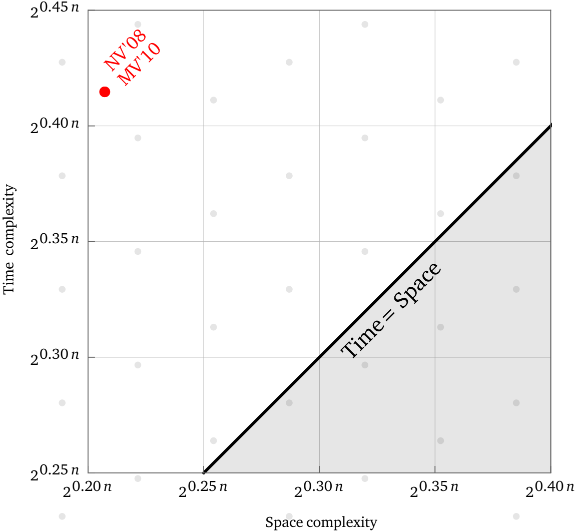


The GaussSieve

Search the list for a shortest vector

The GaussSieve

Space/time trade-off

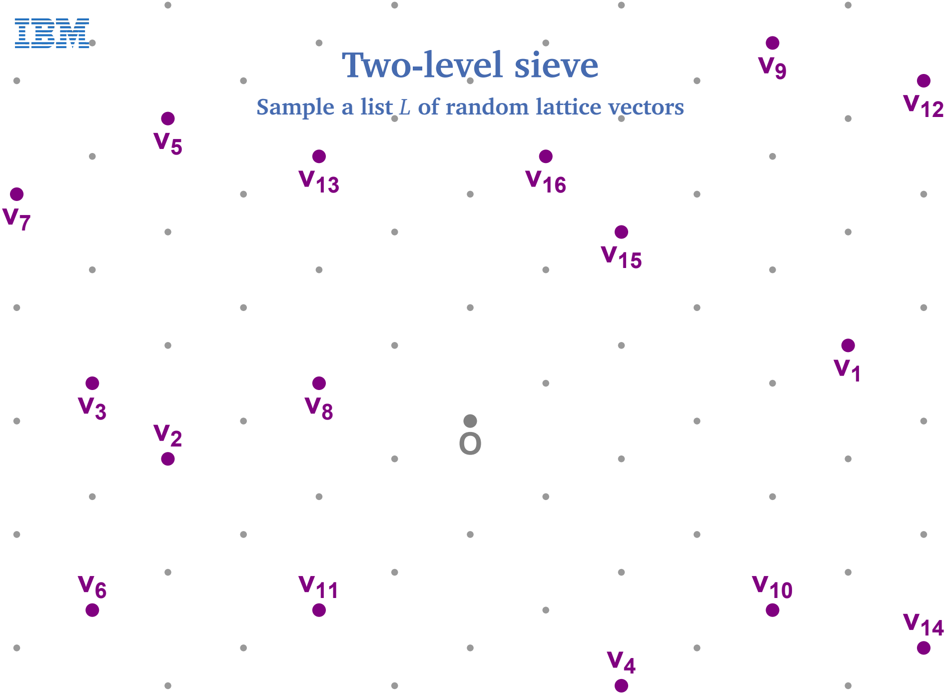


Two-level sieve

Sample a list L of random lattice vectors

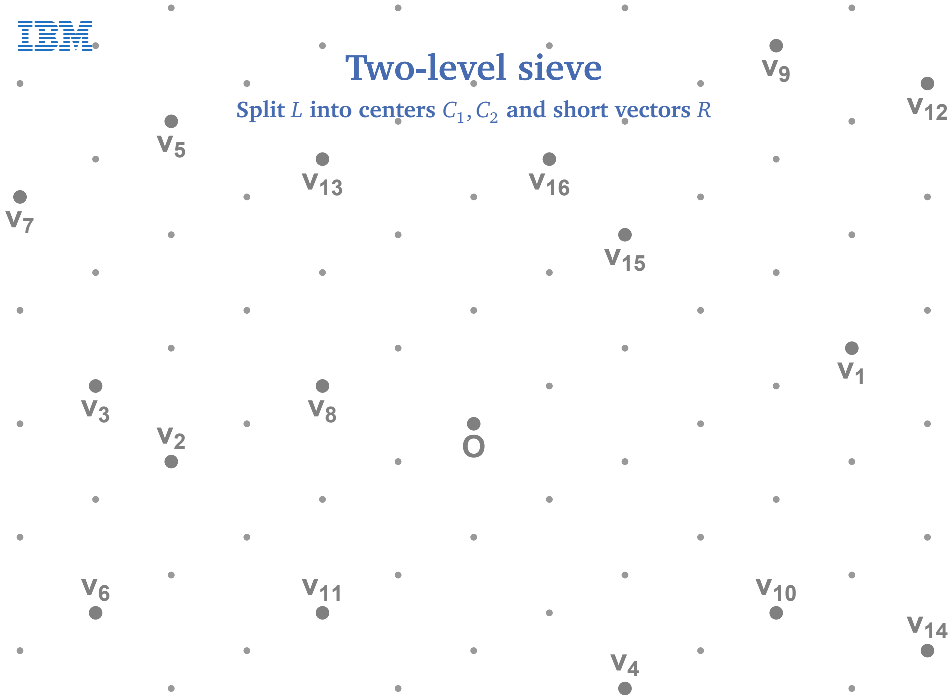
Two-level sieve

Sample a list L of random lattice vectors



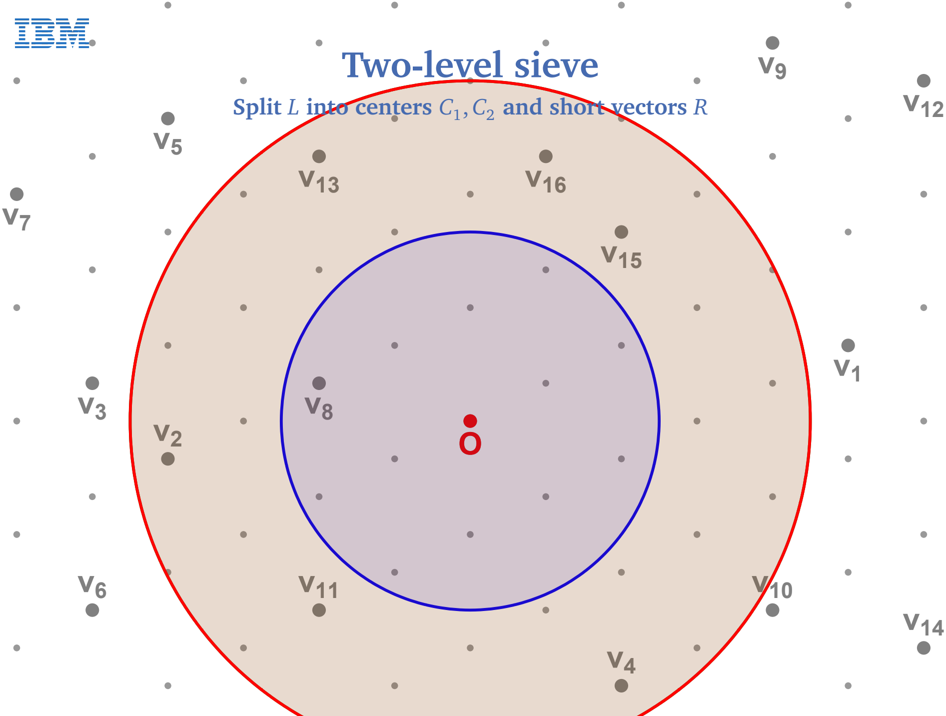
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



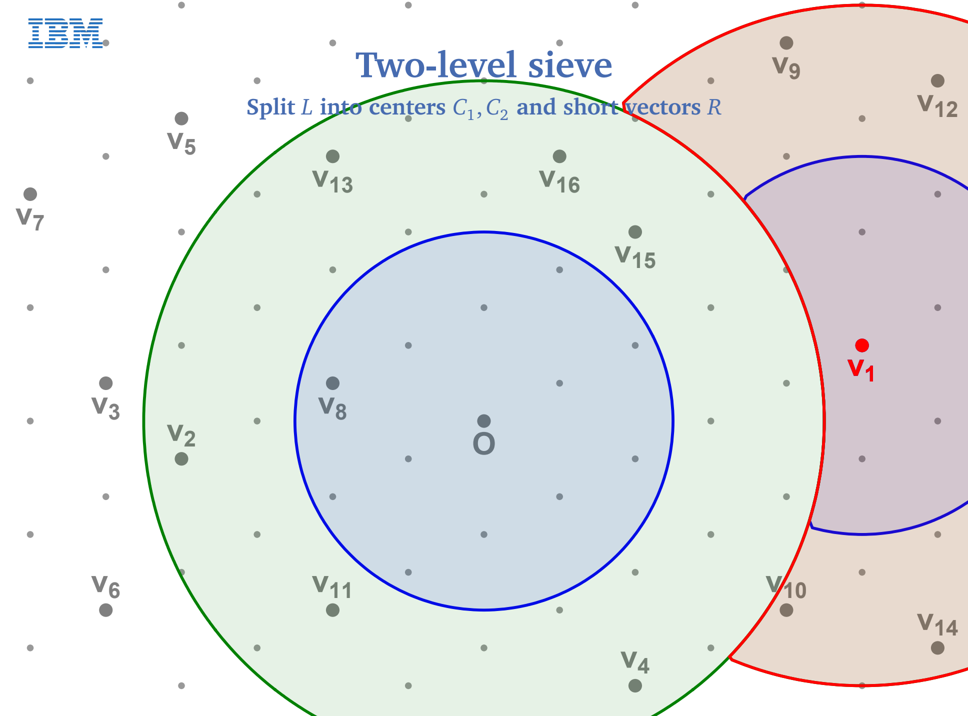
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



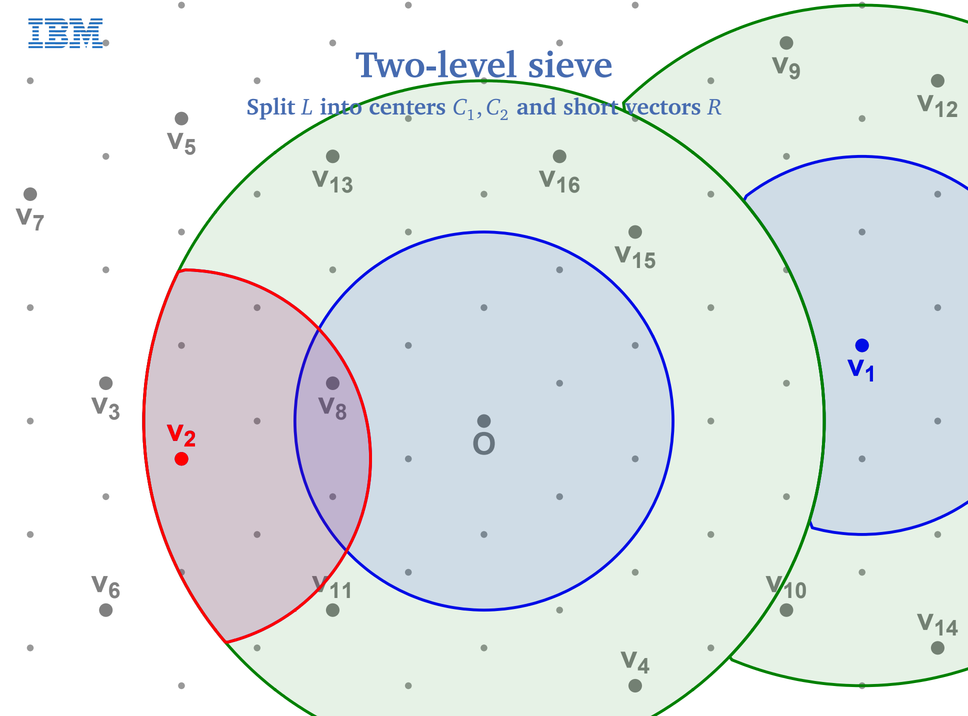
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



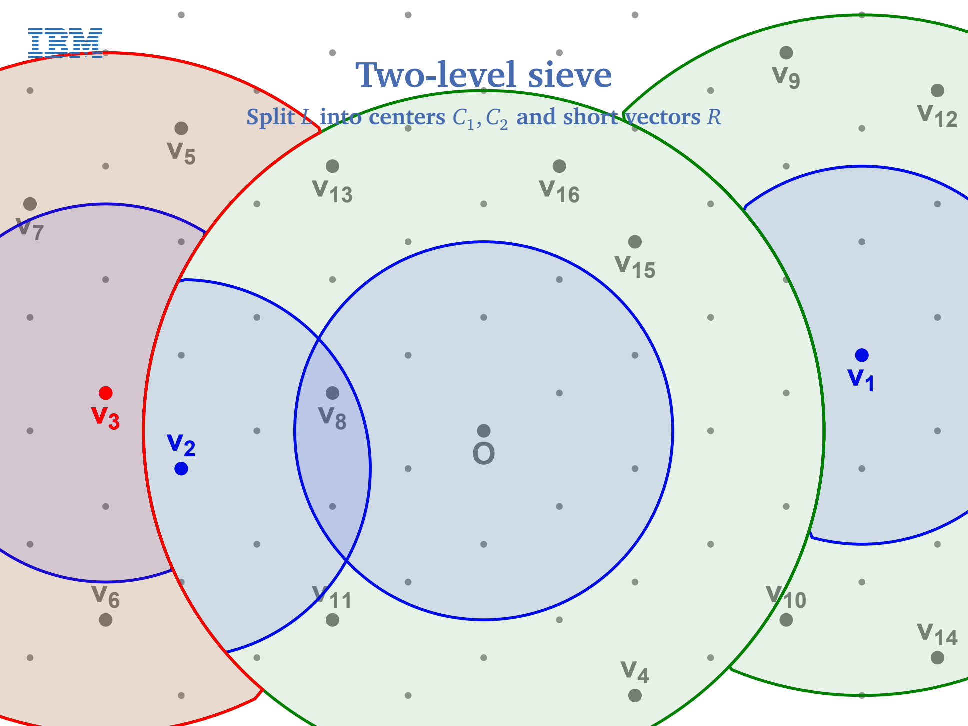
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



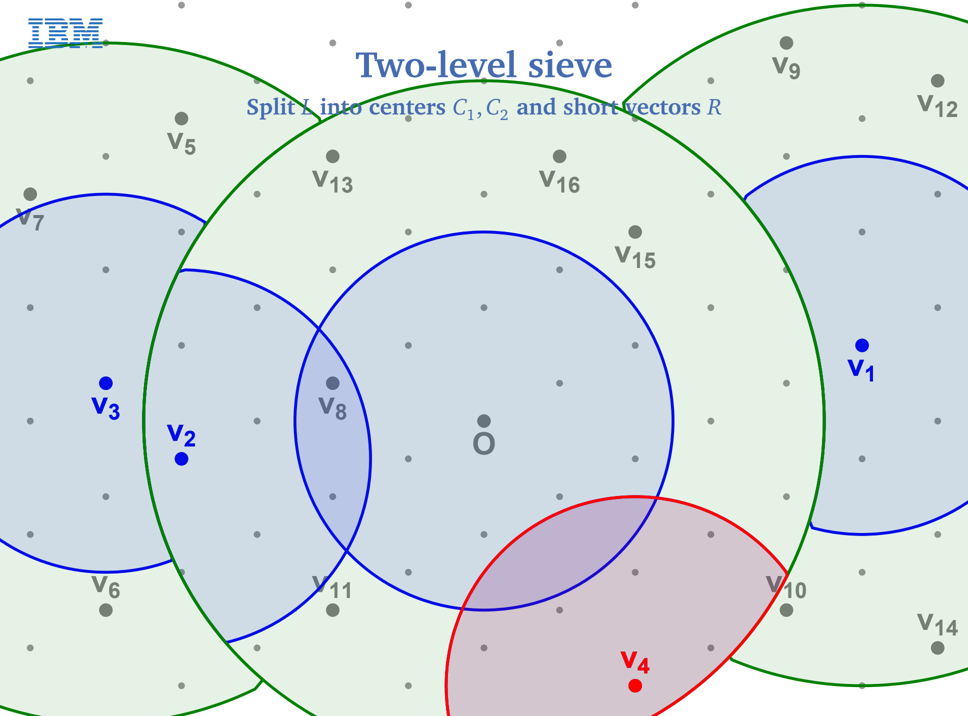
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



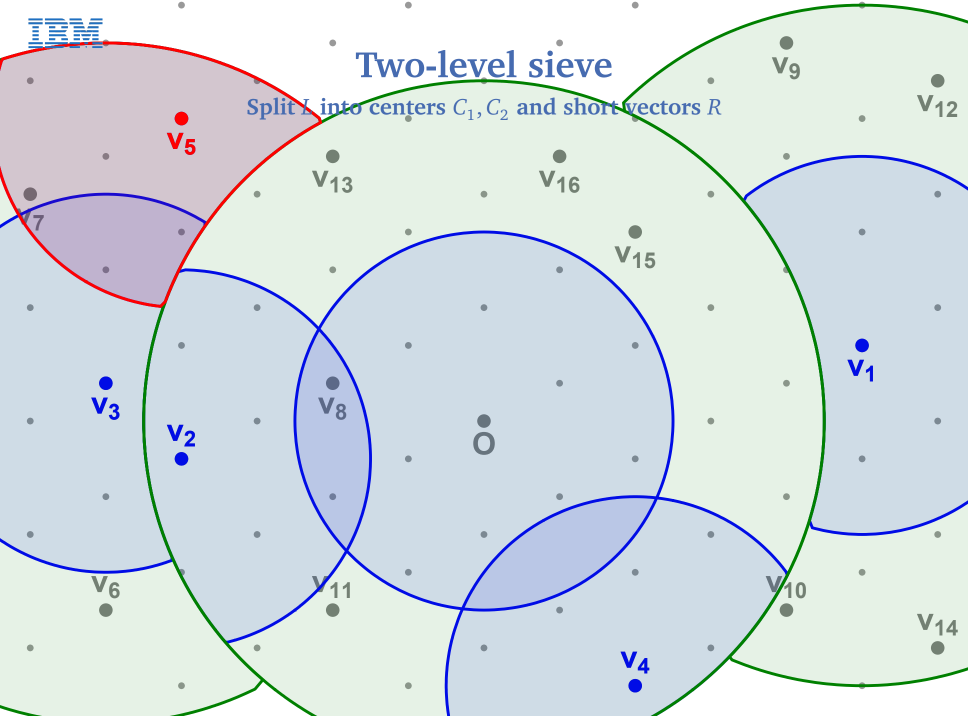
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



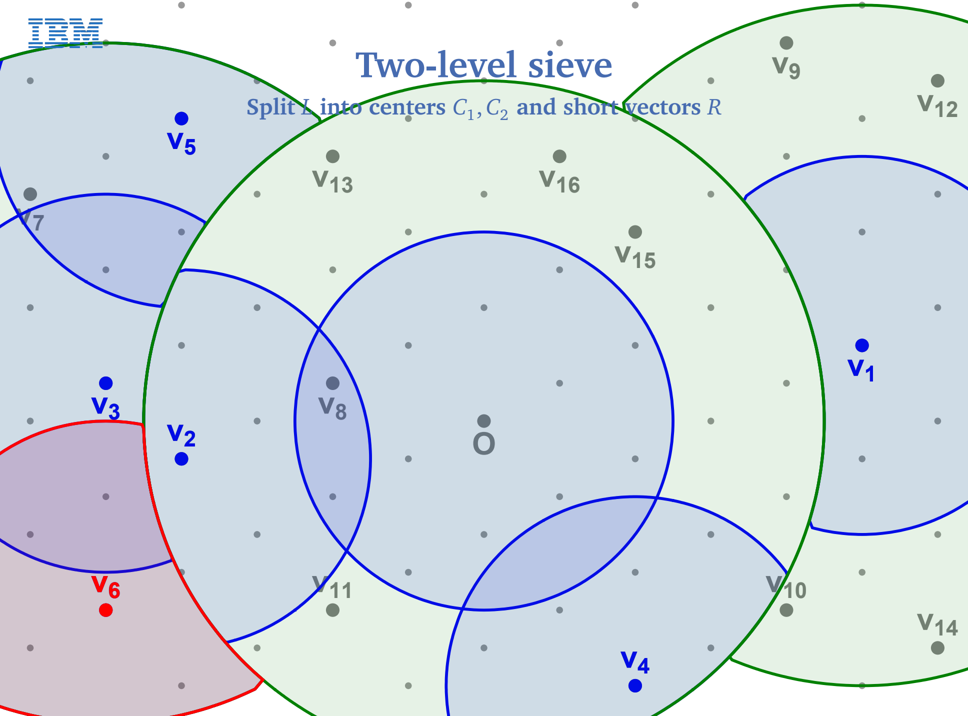
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



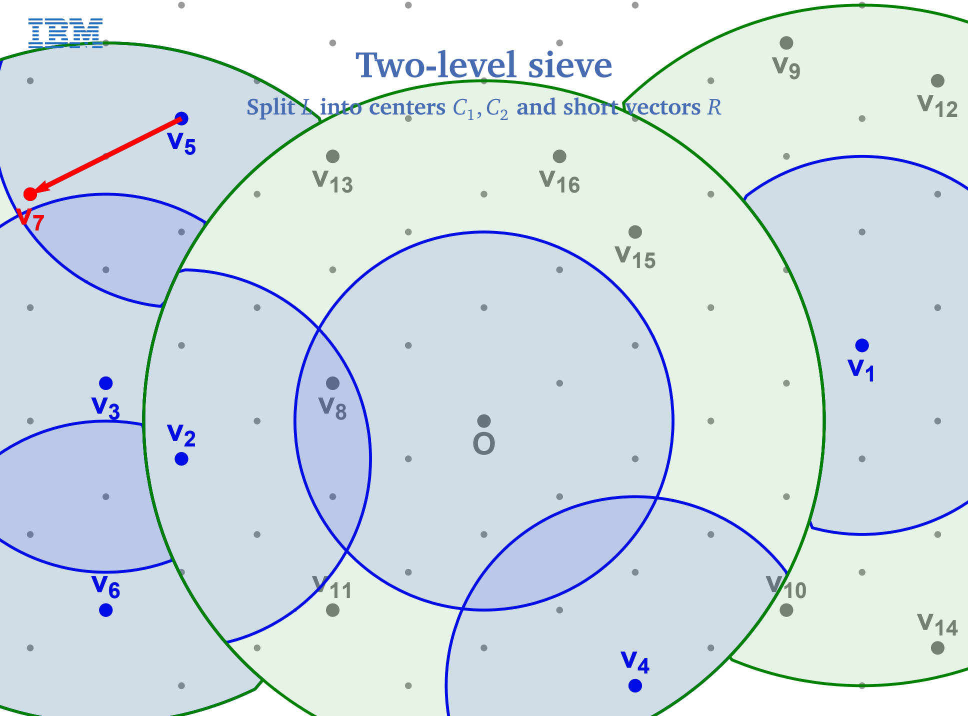
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



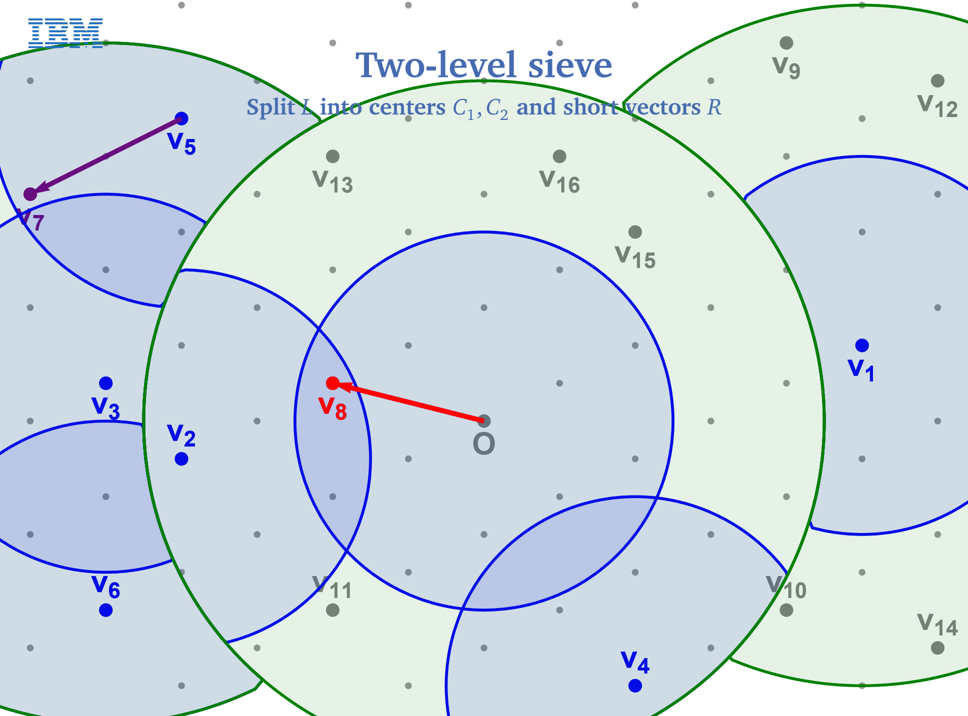
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



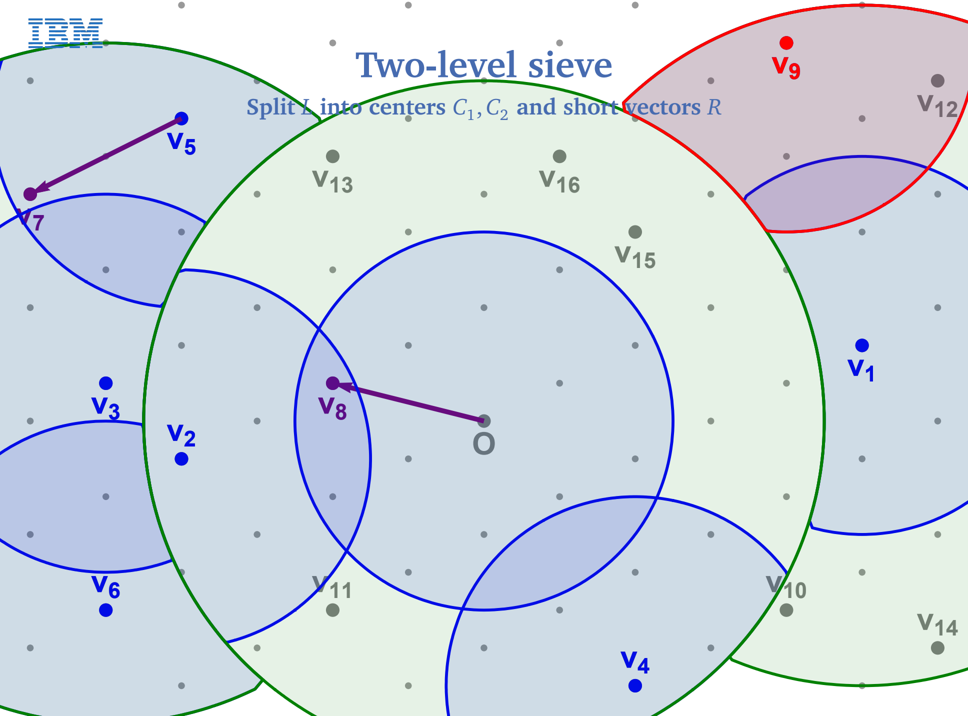
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



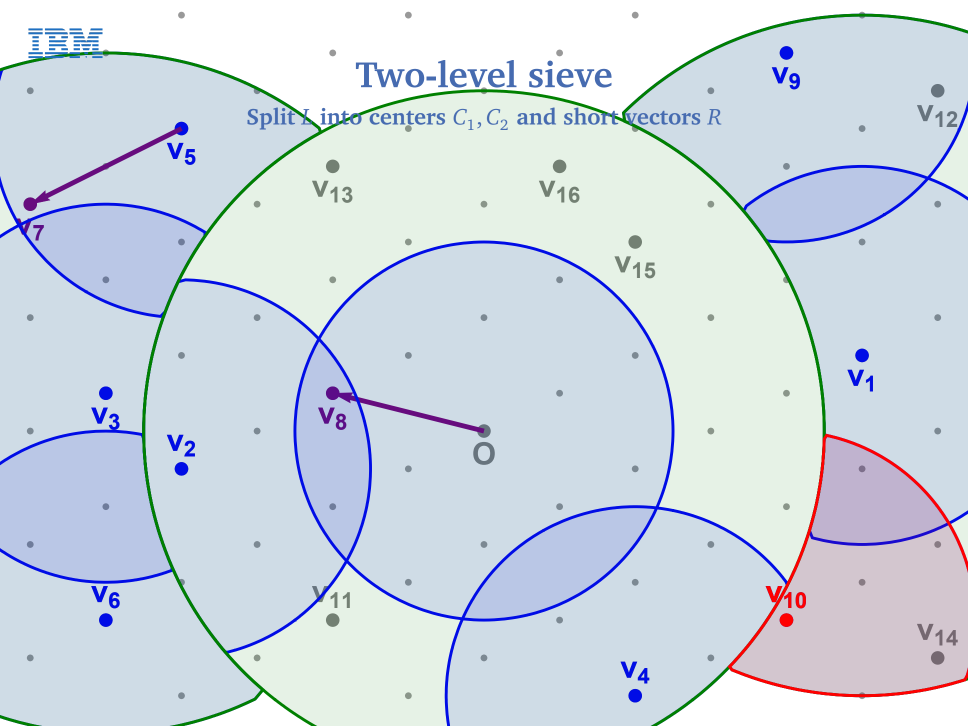
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



Two-level sieve

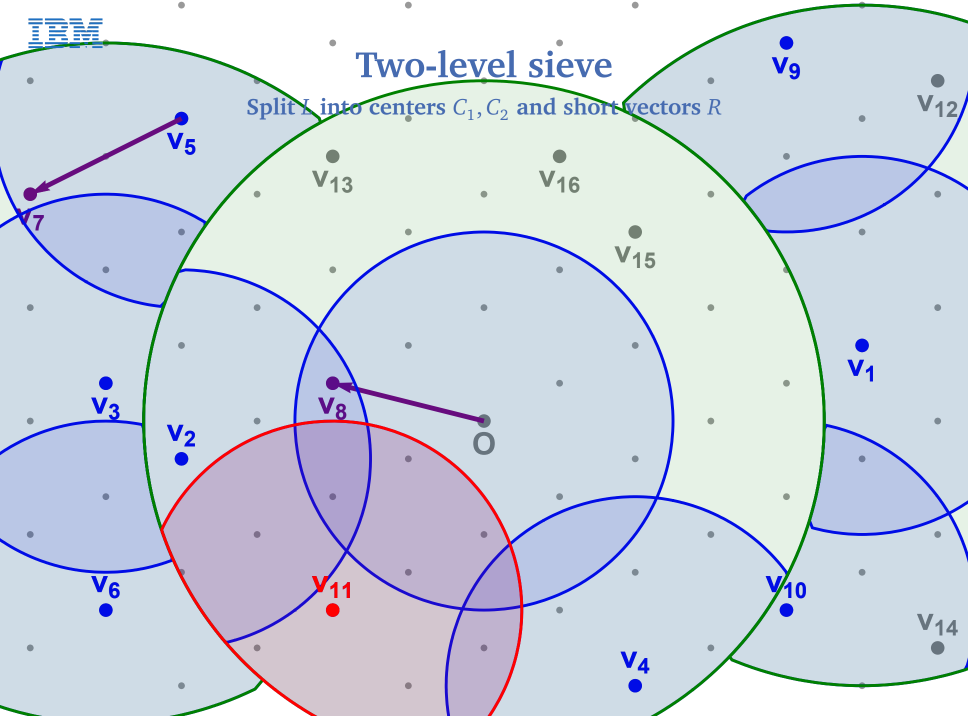
Split L into centers C_1, C_2 and short vectors R



IBM

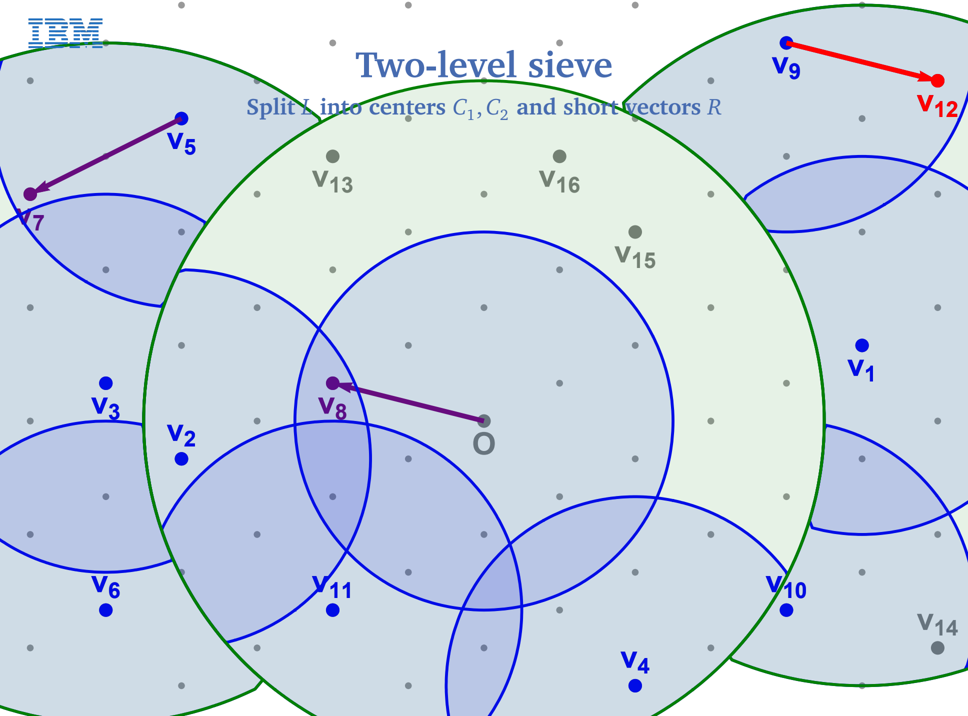
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



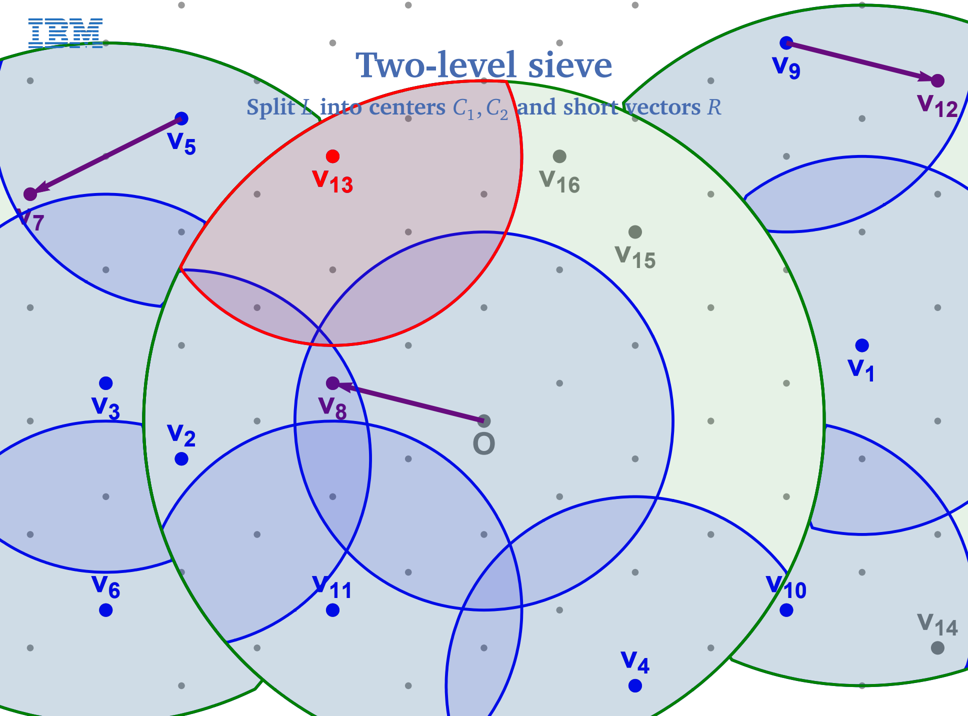
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



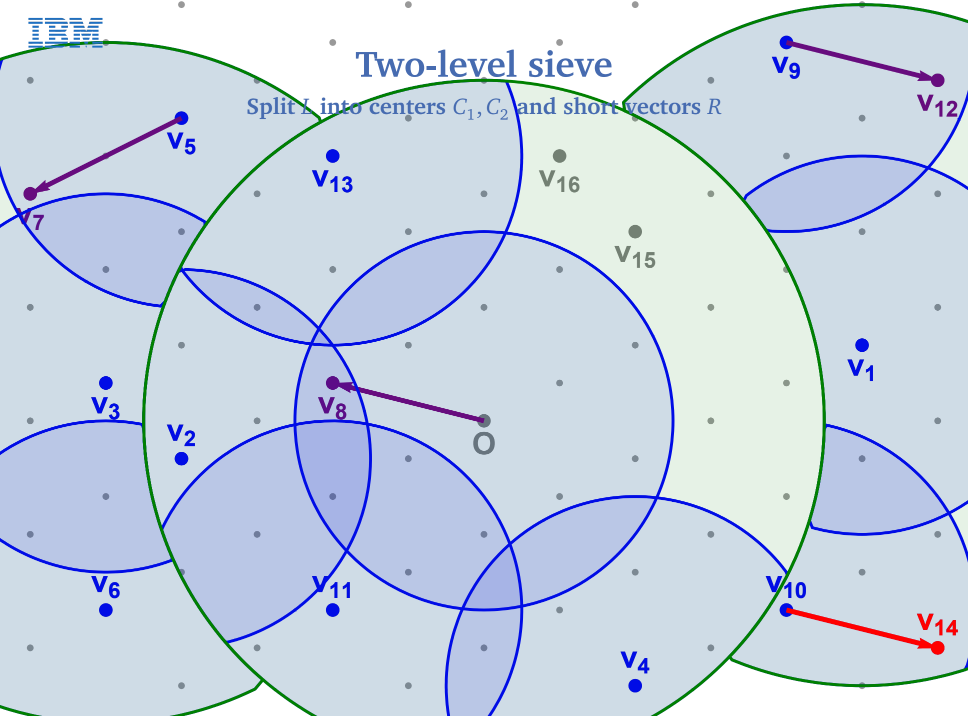
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



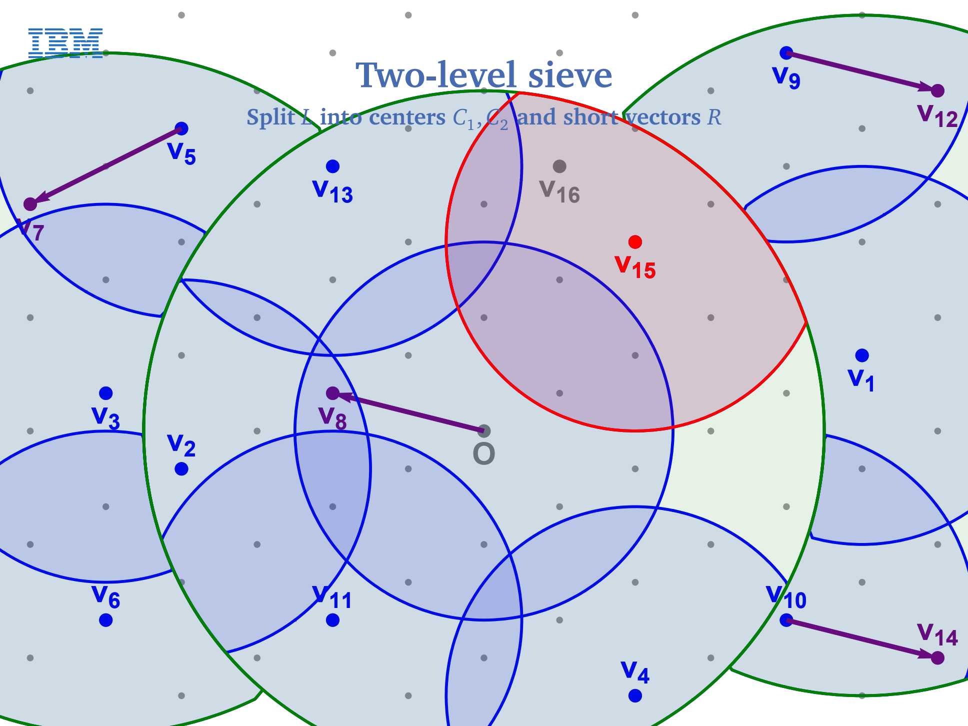
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



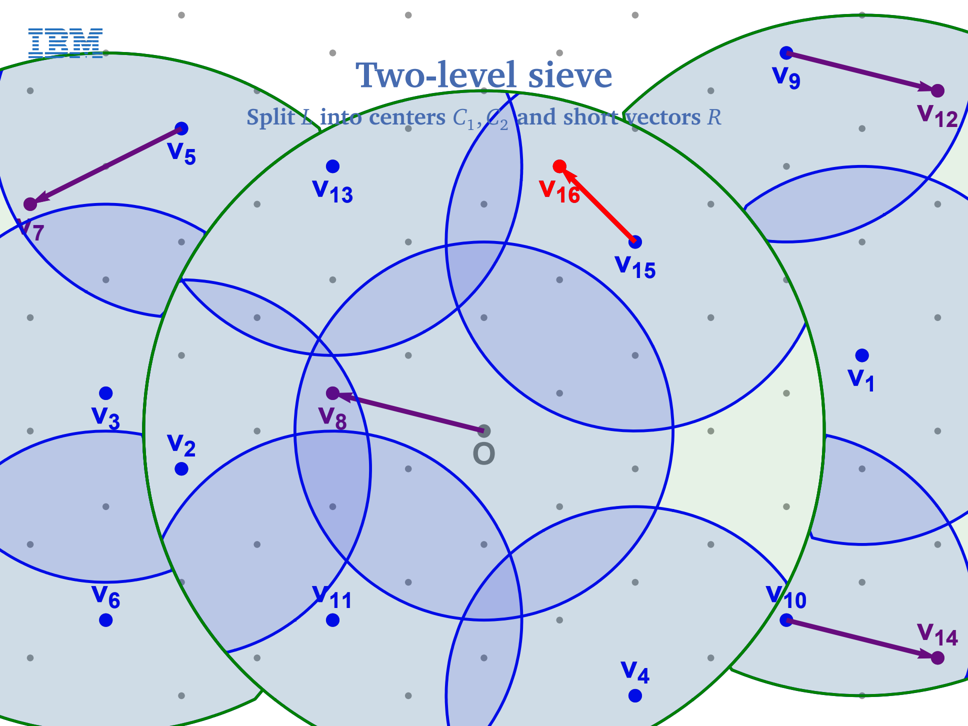
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



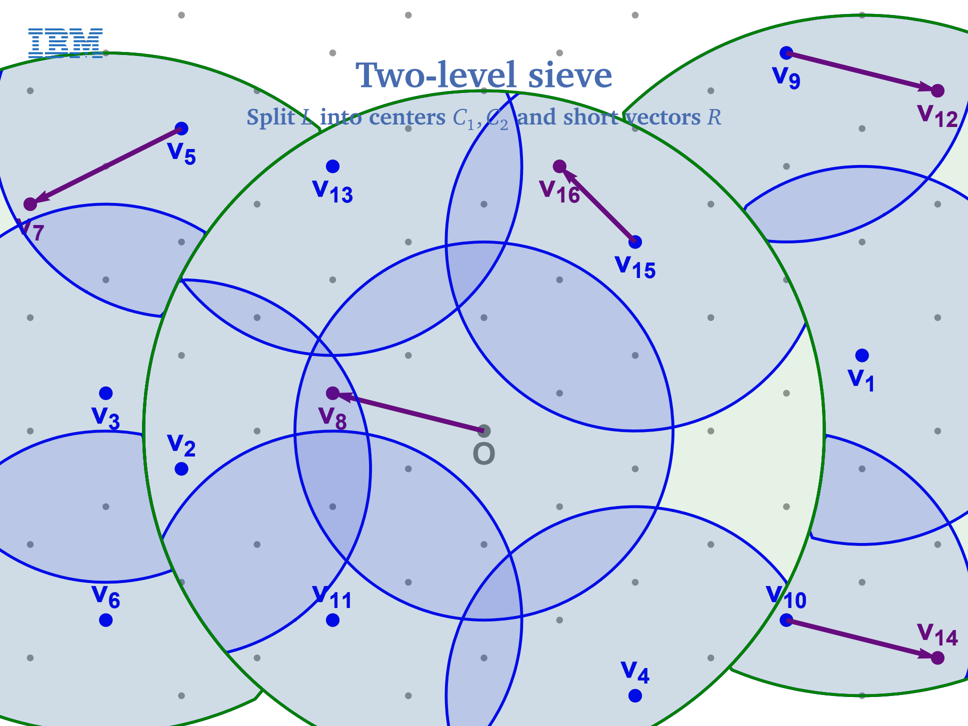
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



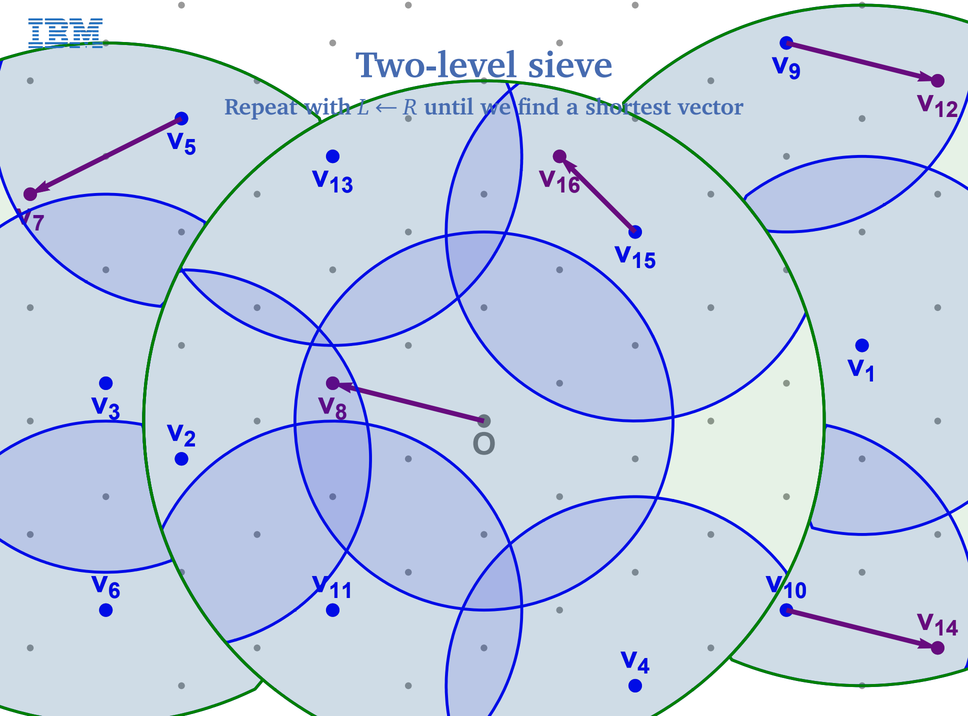
Two-level sieve

Split L into centers C_1, C_2 and short vectors R



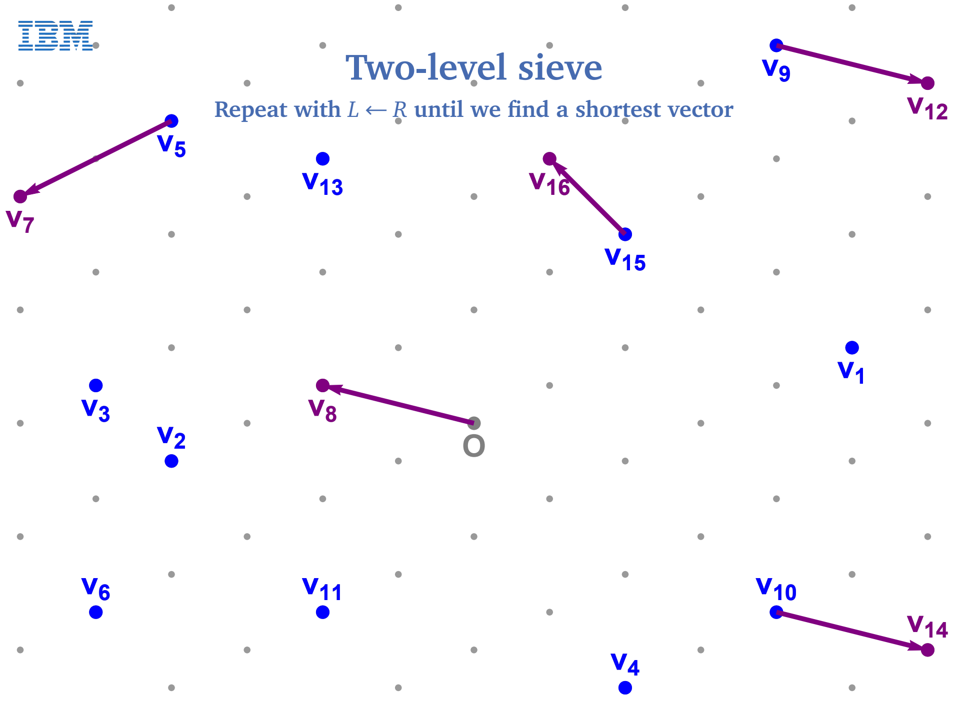
Two-level sieve

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



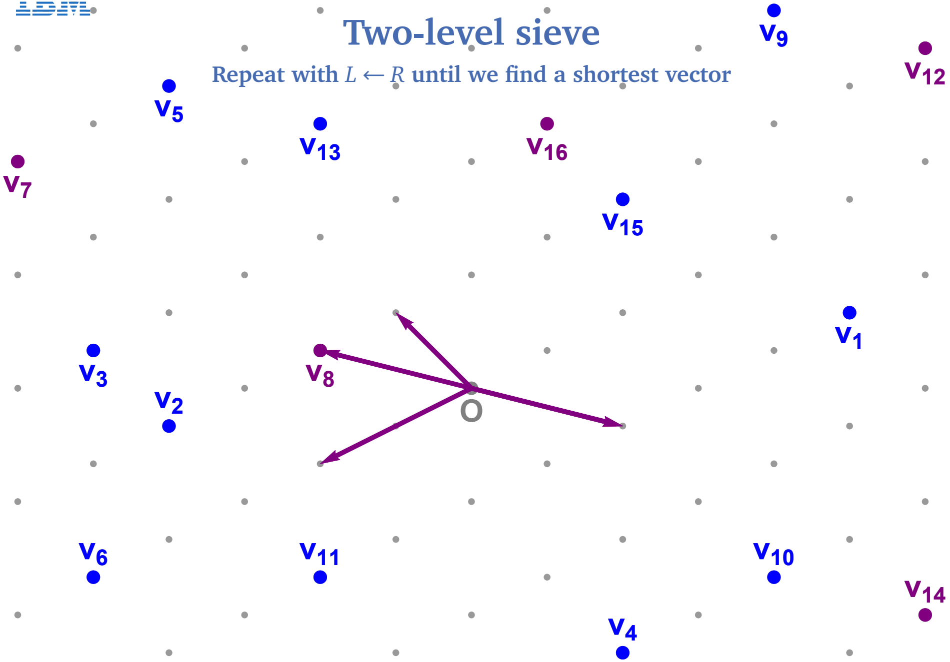
Two-level sieve

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



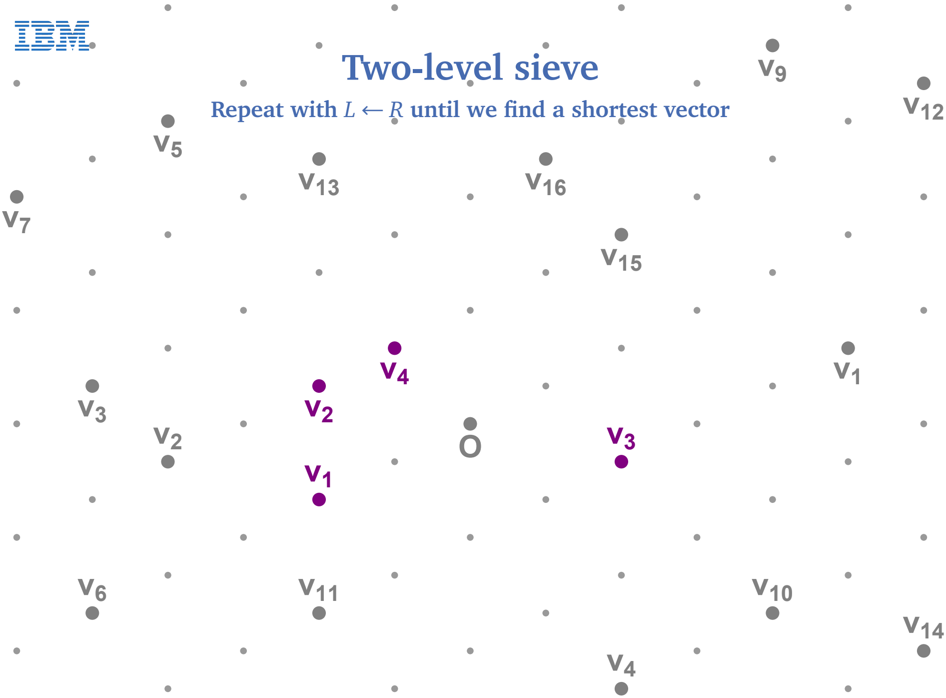
Two-level sieve

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



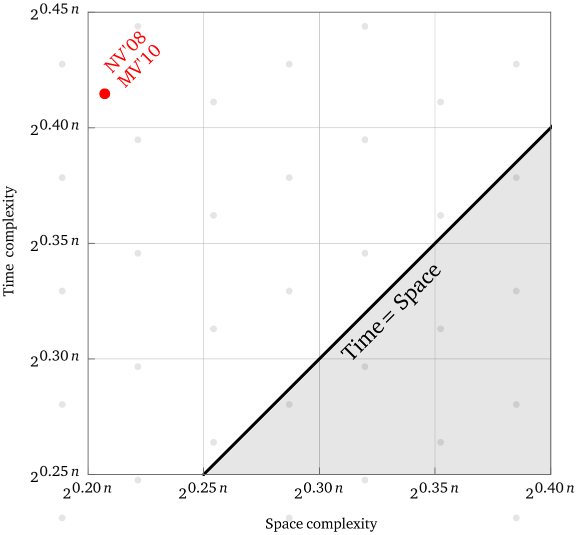
Two-level sieve

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



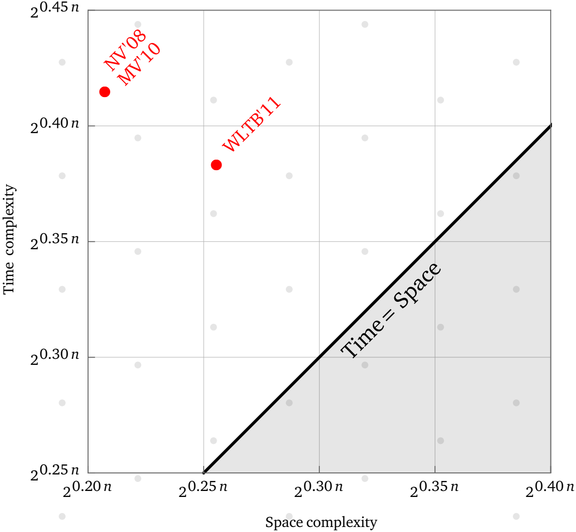
Two-level sieve

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* $2^{0.2075n}$.

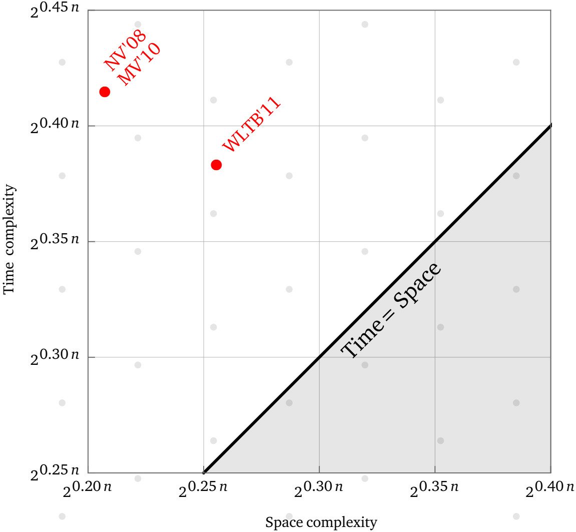
Heuristic result (Wang–Liu–Tian–Bi, ASIACCS’11)
The two-level sieve runs in time $2^{0.3836n}$ *and space* $2^{0.2557n}$.

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.*

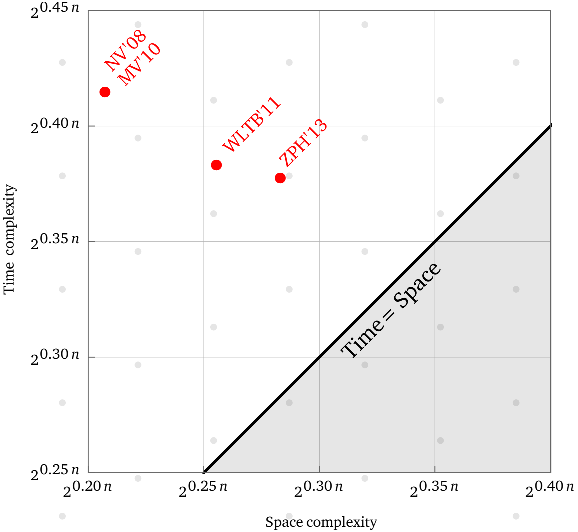
Three-level sieve

Space/time trade-off



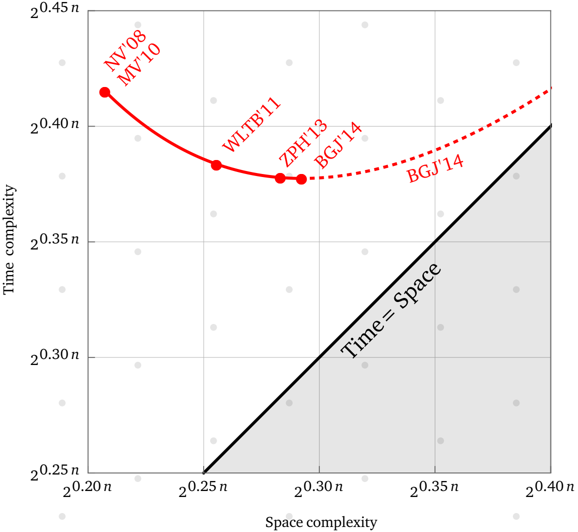
Three-level sieve

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

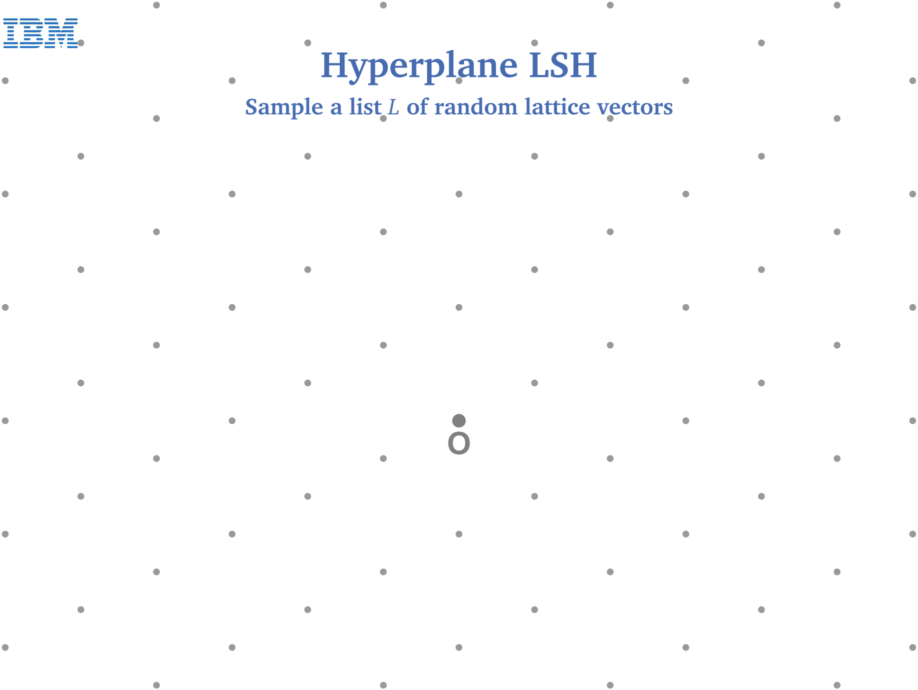
Introduction

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*

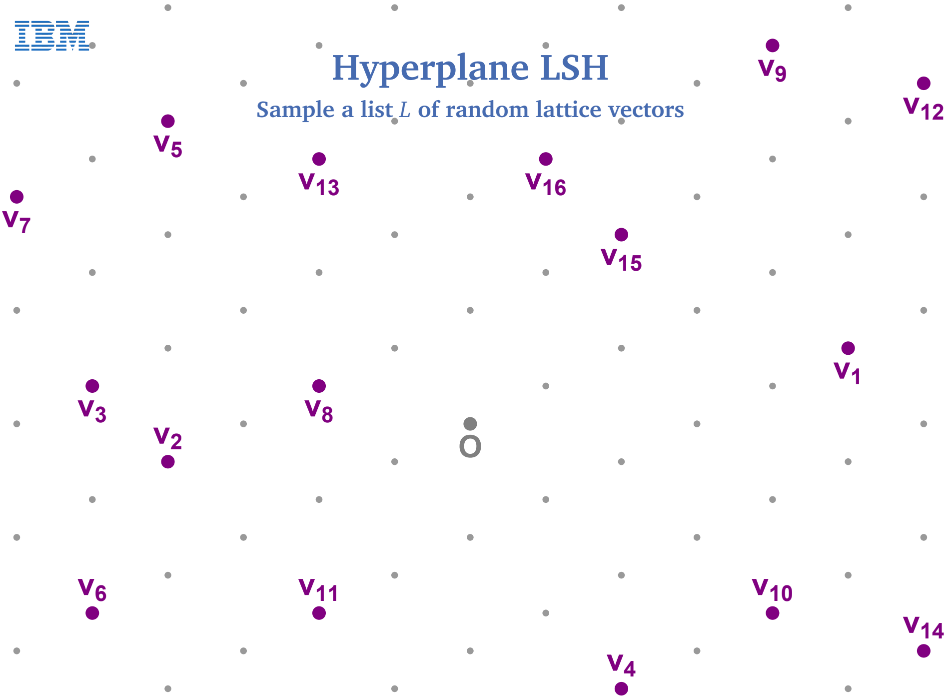
Hyperplane LSH

Sample a list L of random lattice vectors



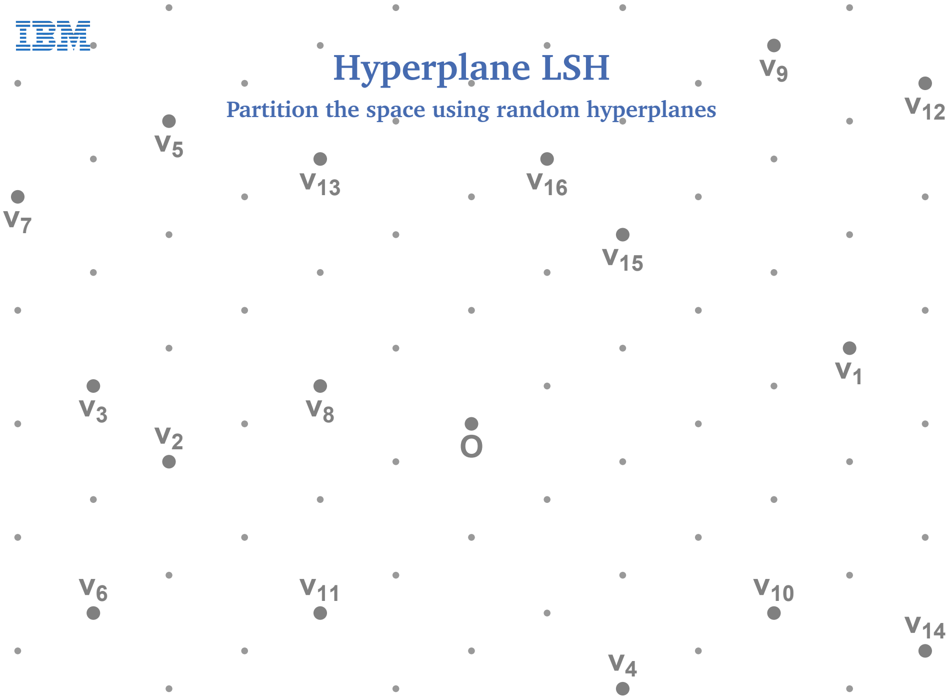
Hyperplane LSH

Sample a list L of random lattice vectors



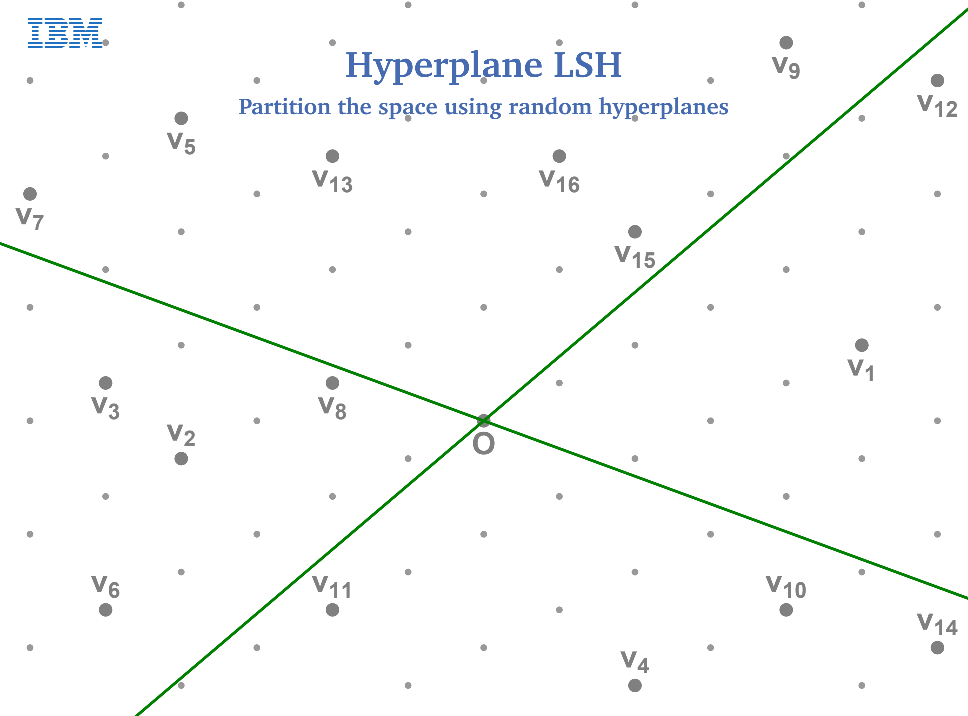
Hyperplane LSH

Partition the space using random hyperplanes



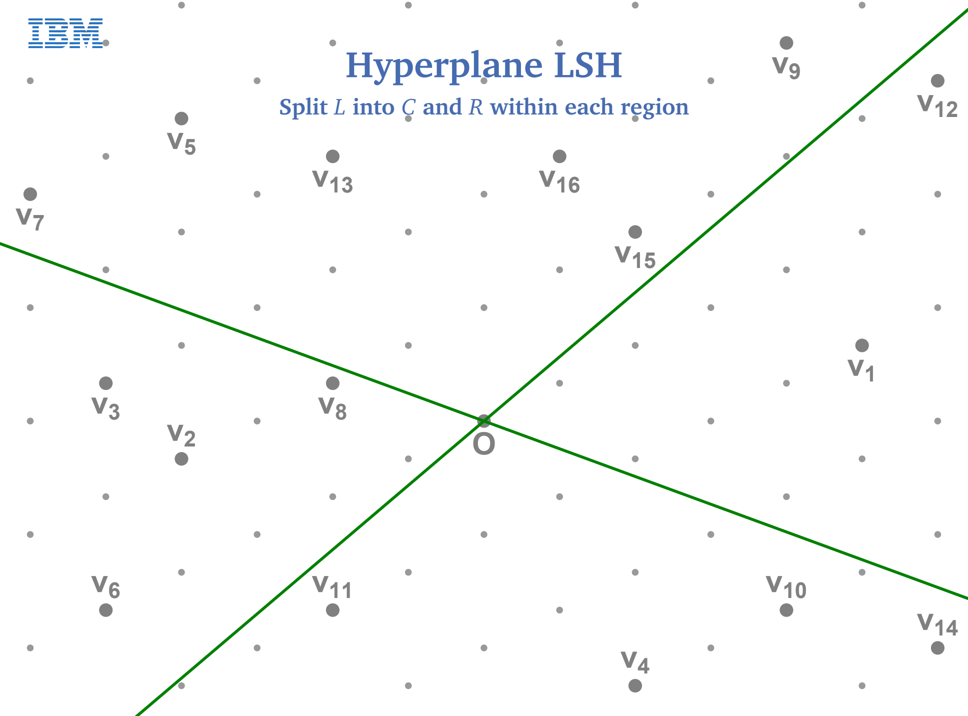
Hyperplane LSH

Partition the space using random hyperplanes



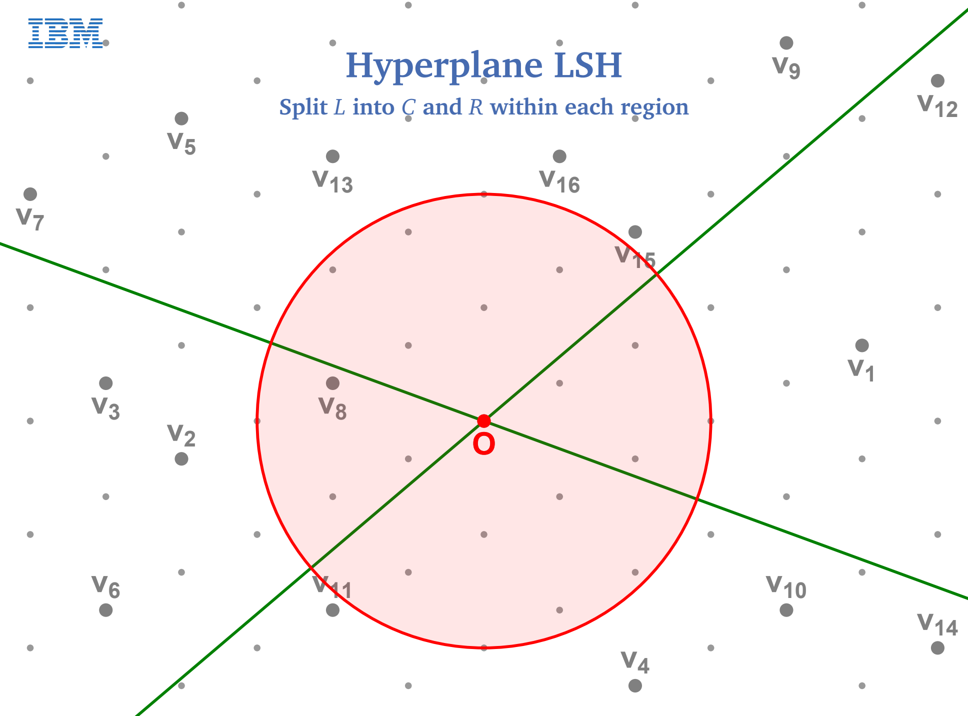
Hyperplane LSH

Split L into C and R within each region



Hyperplane LSH

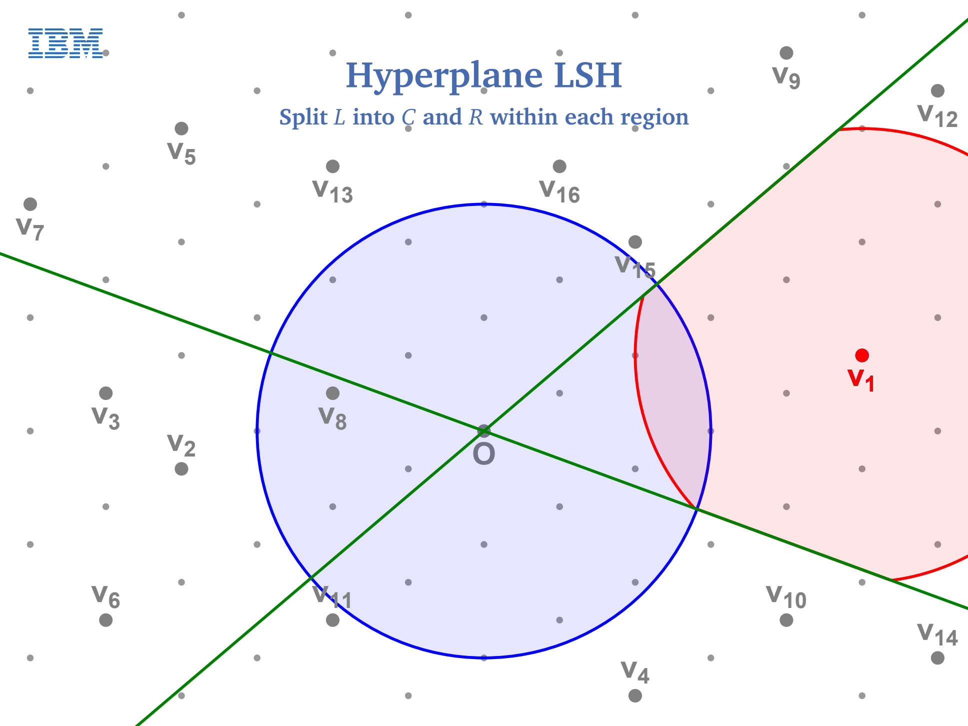
Split L into C and R within each region



IBM

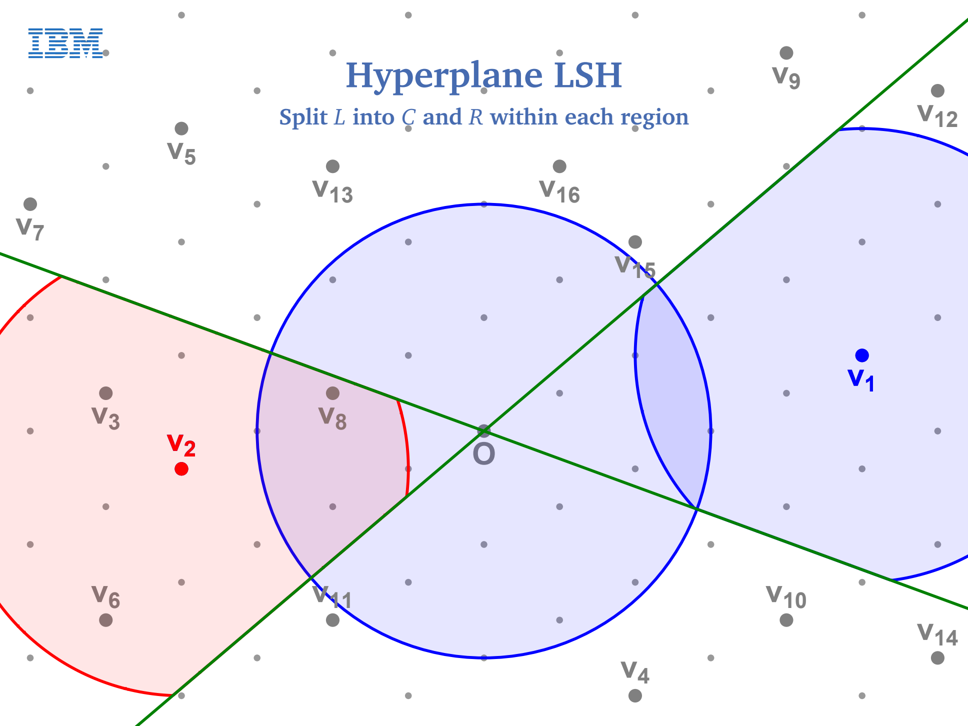
Hyperplane LSH

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



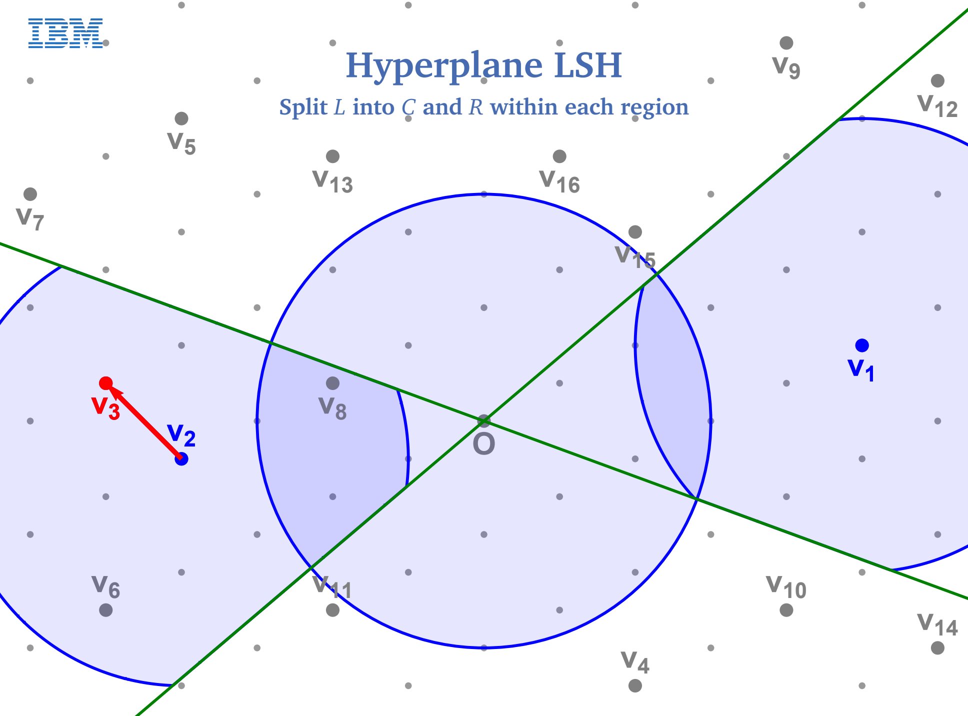
Hyperplane LSH

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



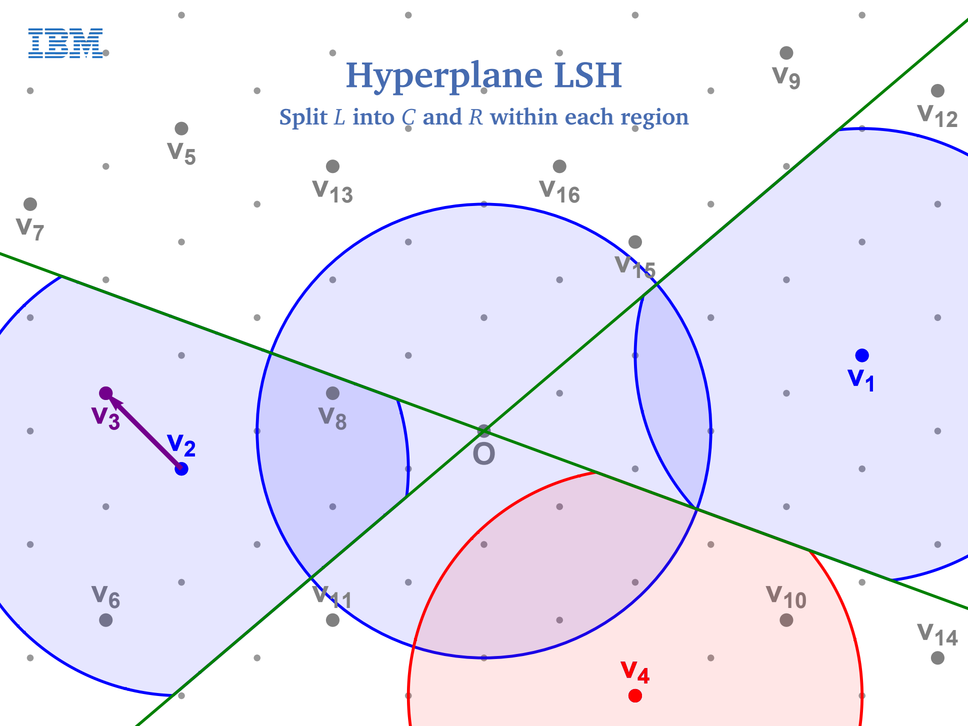
Hyperplane LSH

Split L into C and R within each region



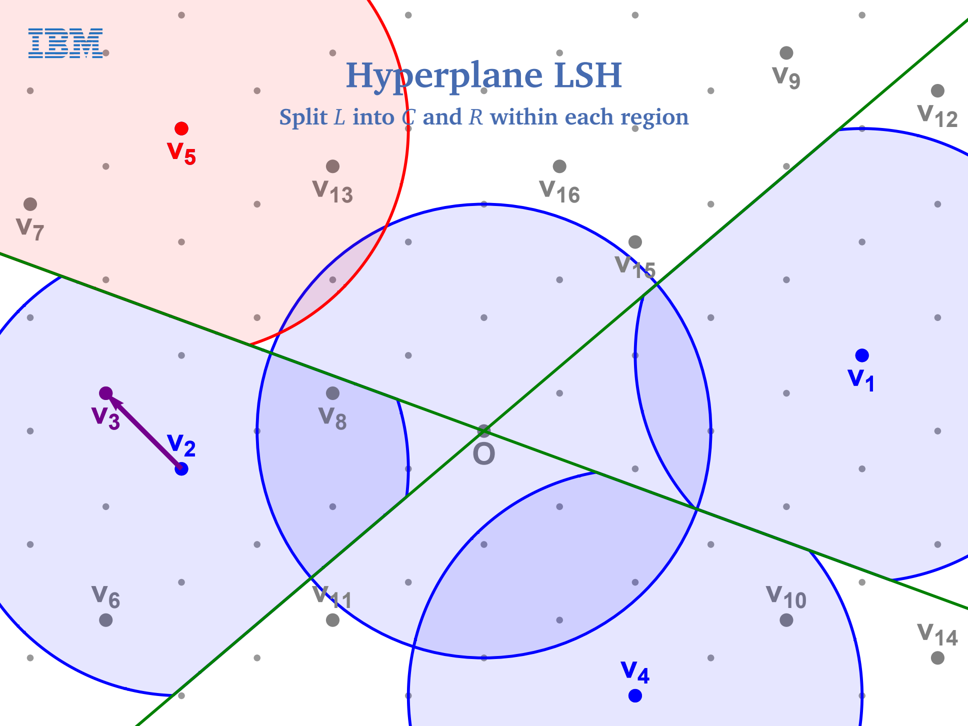
Hyperplane LSH

Split L into C and R within each region



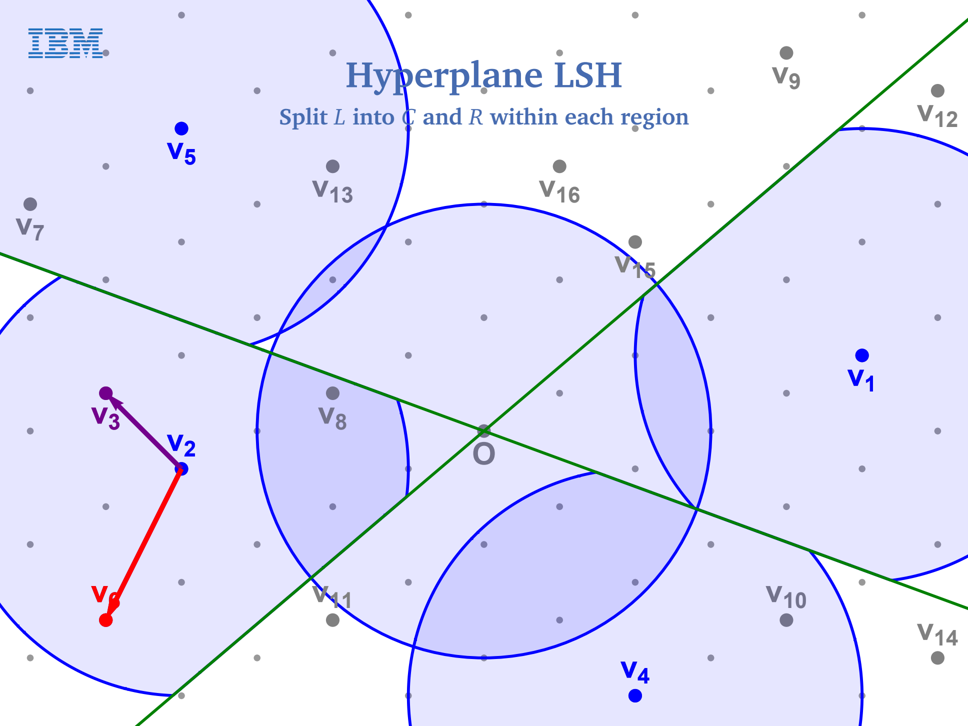
Hyperplane LSH

Split L into C and R within each region



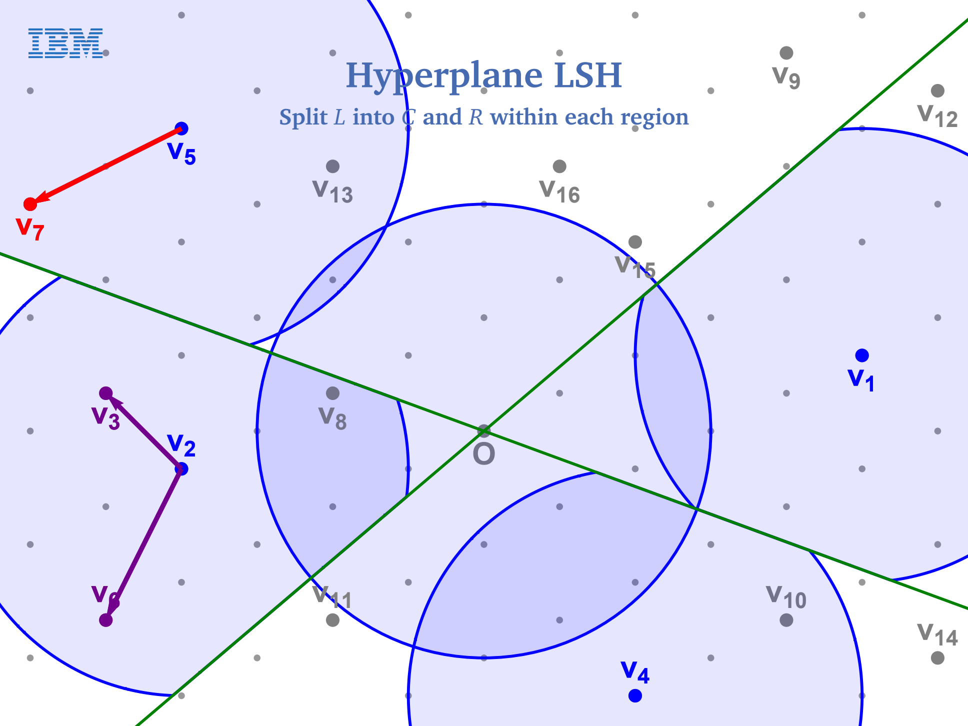
Hyperplane LSH

Split L into C and R within each region



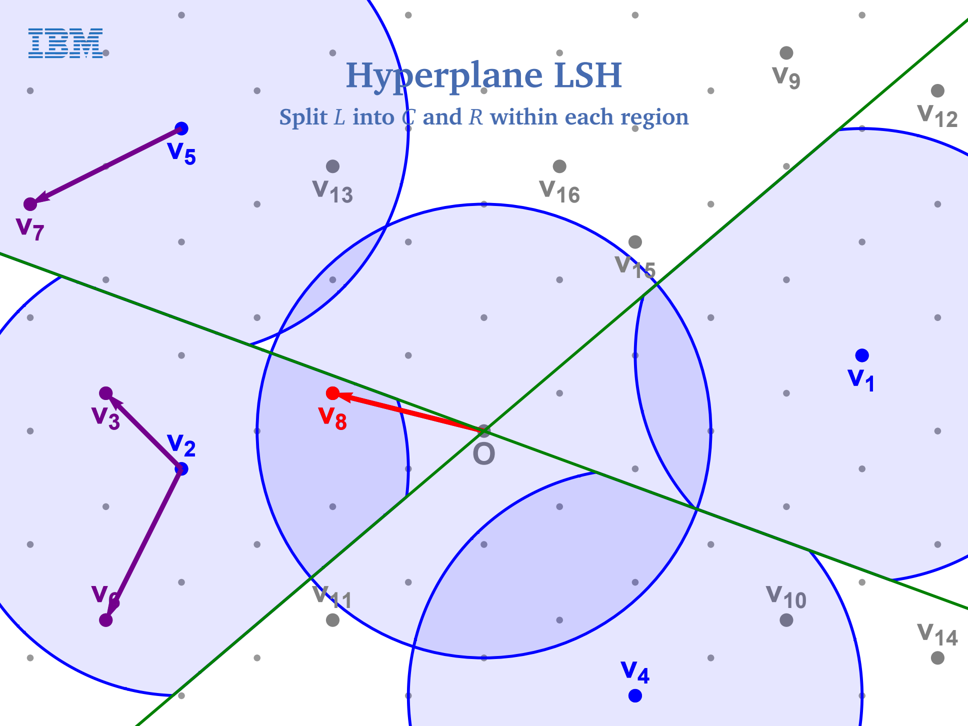
Hyperplane LSH

Split L into C and R within each region



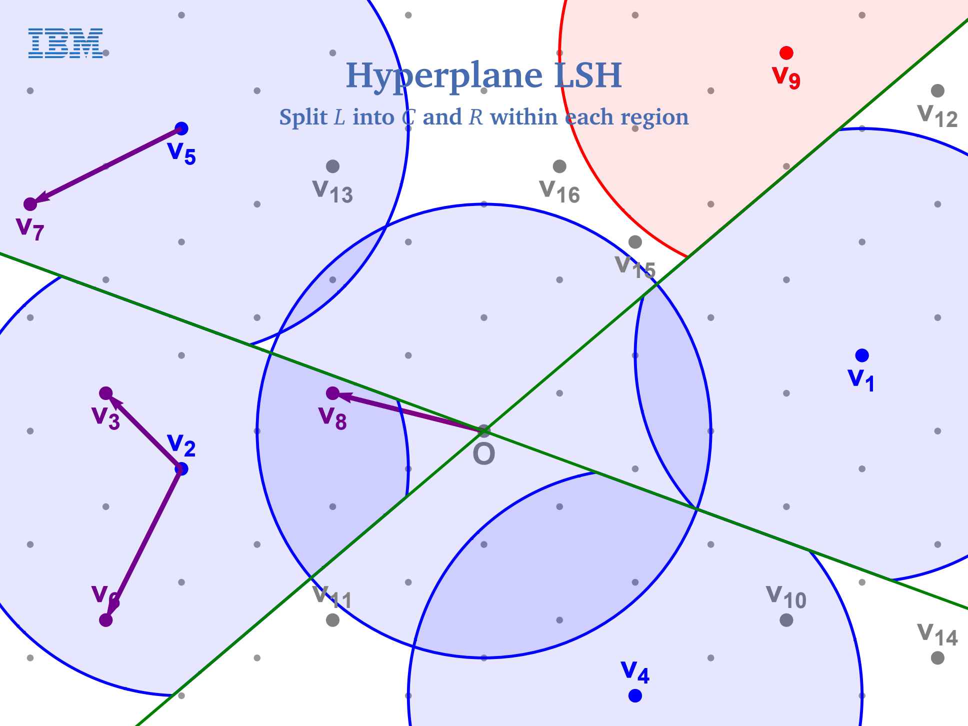
Hyperplane LSH

Split L into C and R within each region



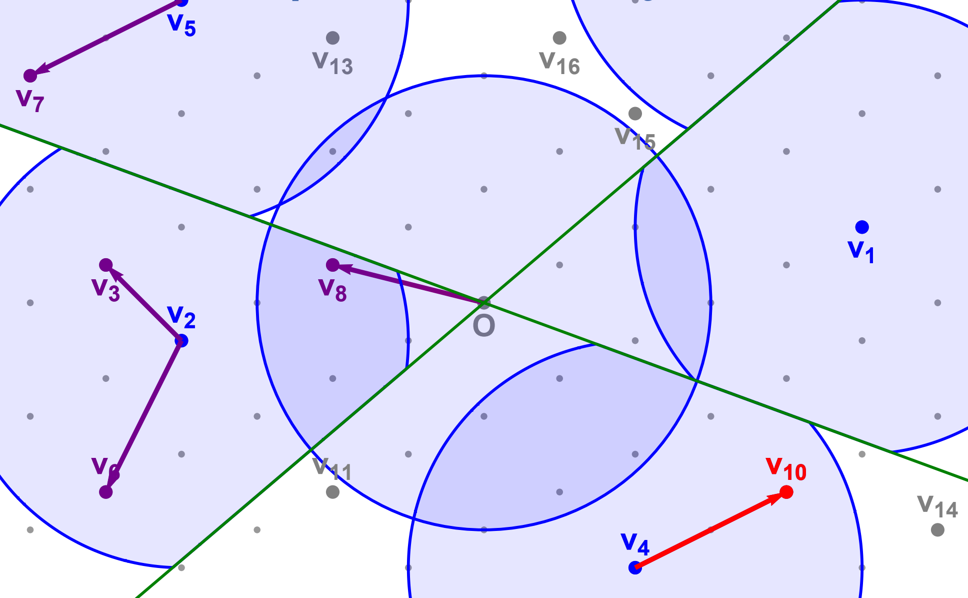
Hyperplane LSH

Split L into C and R within each region



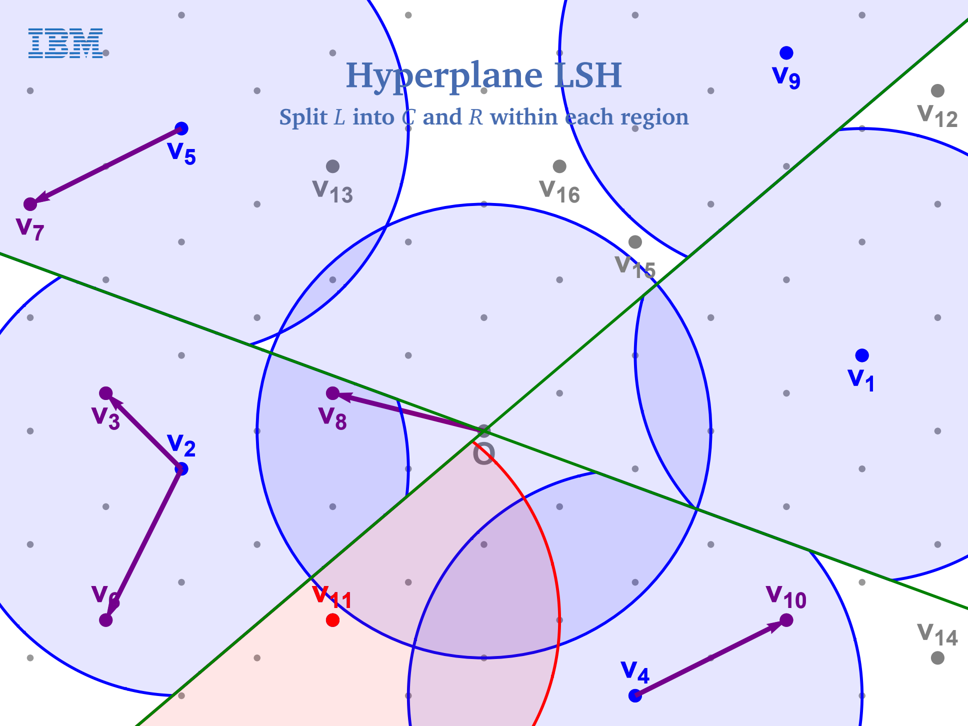
Hyperplane LSH

Split L into C and R within each region



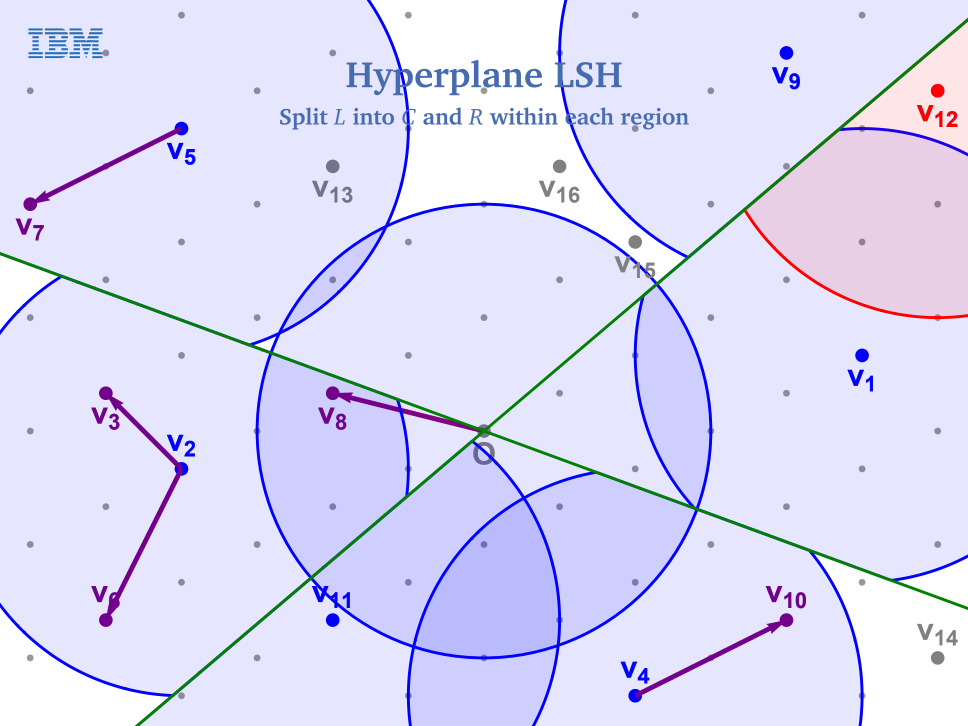
Hyperplane LSH

Split L into C and R within each region



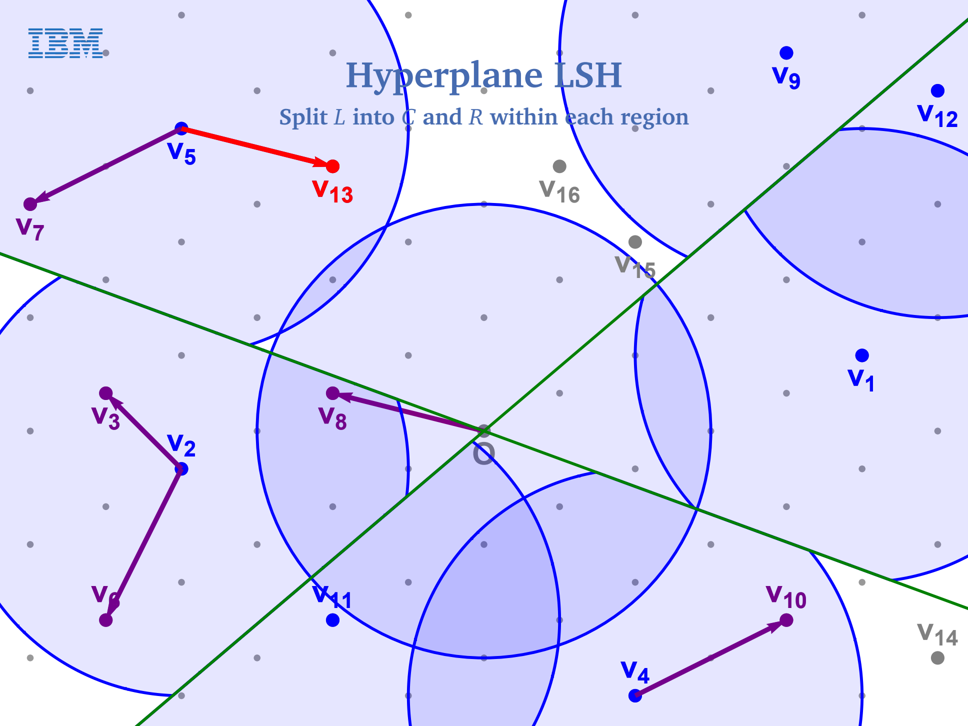
Hyperplane LSH

Split L into C and R within each region



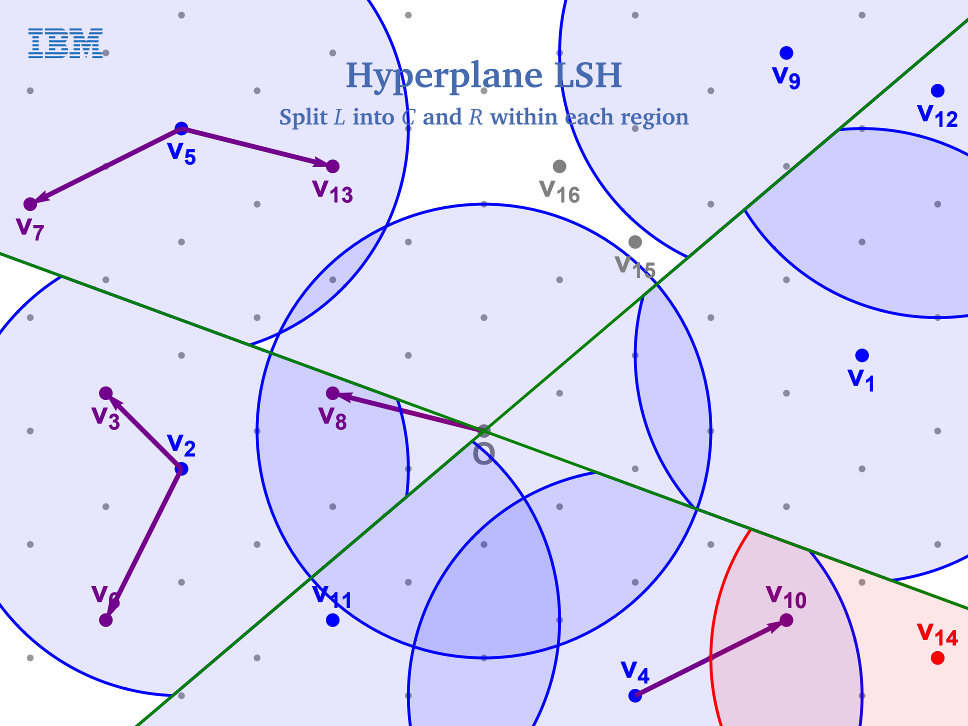
Hyperplane LSH

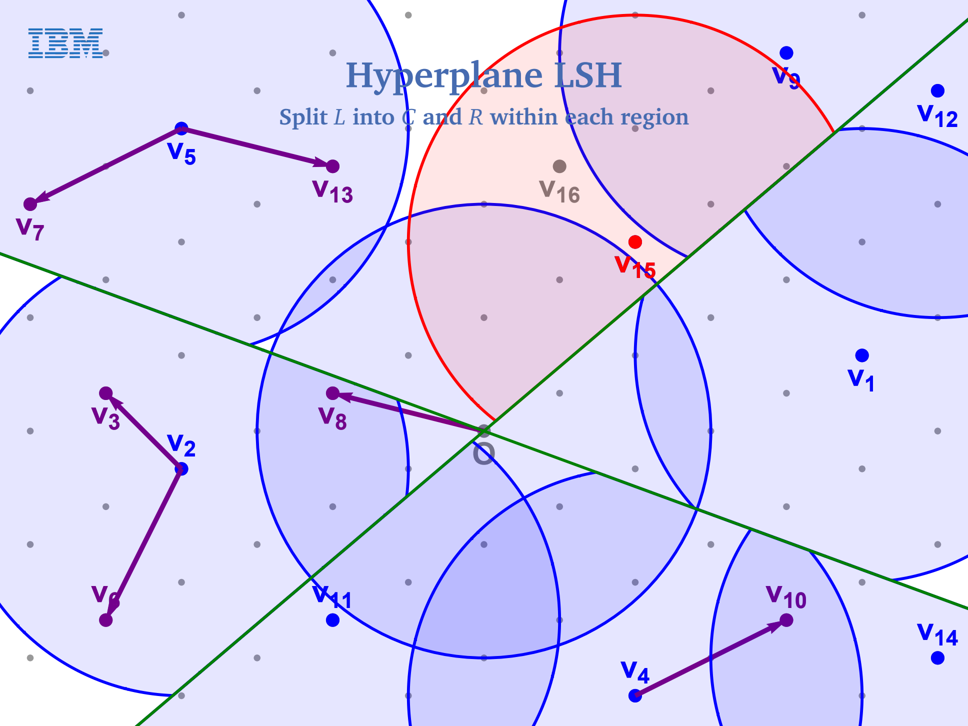
Split L into C and R within each region



Hyperplane LSH

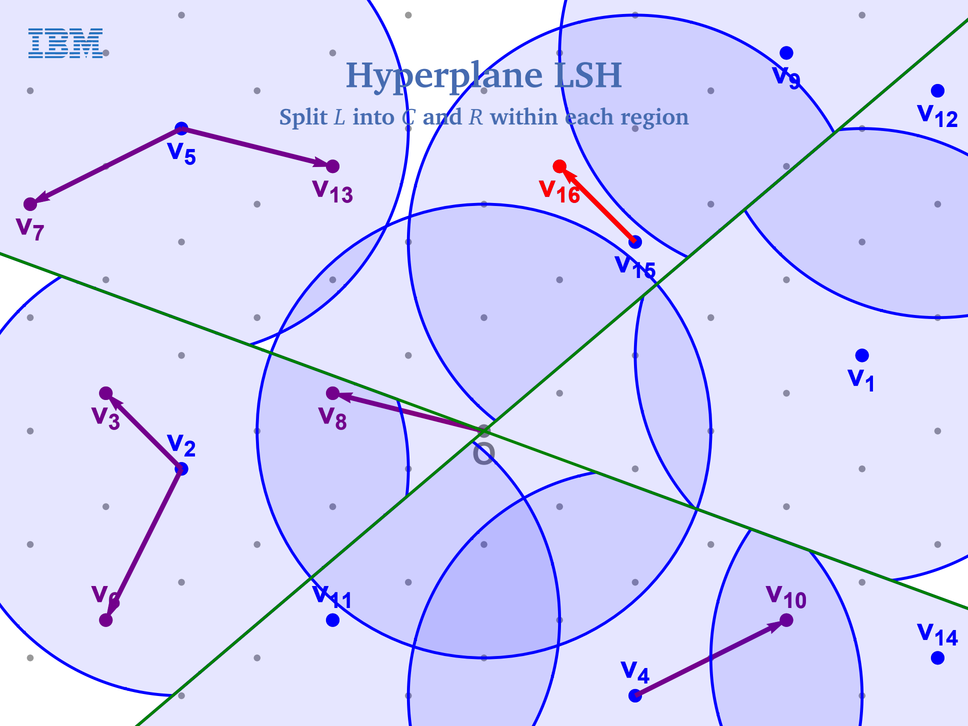
Split L into C and R within each region




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

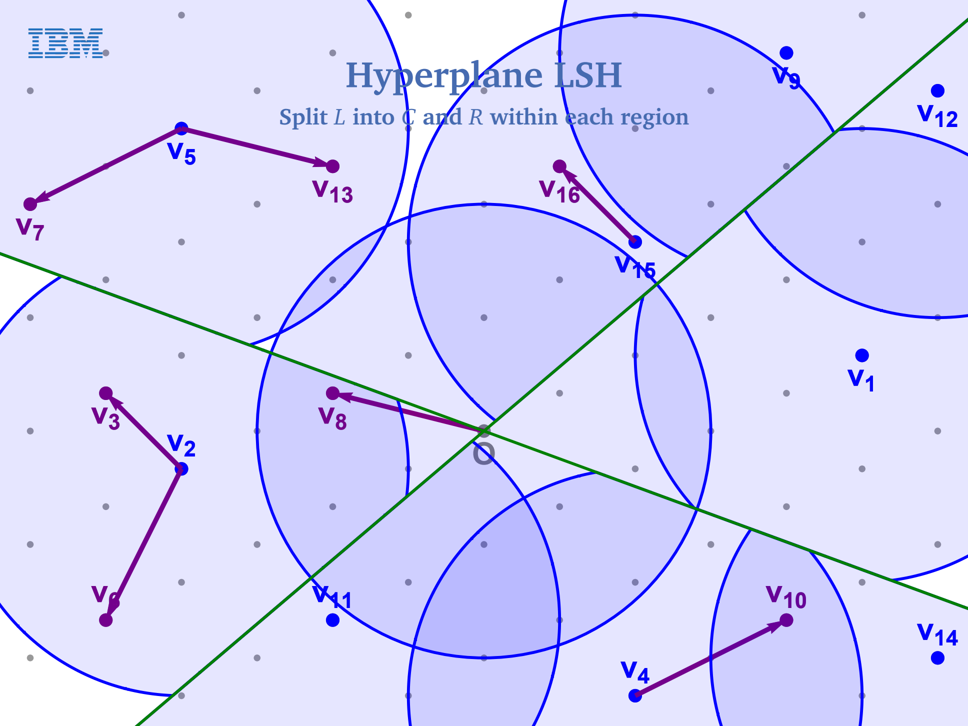
Hyperplane LSH

Split L into C and R within each region



Hyperplane LSH

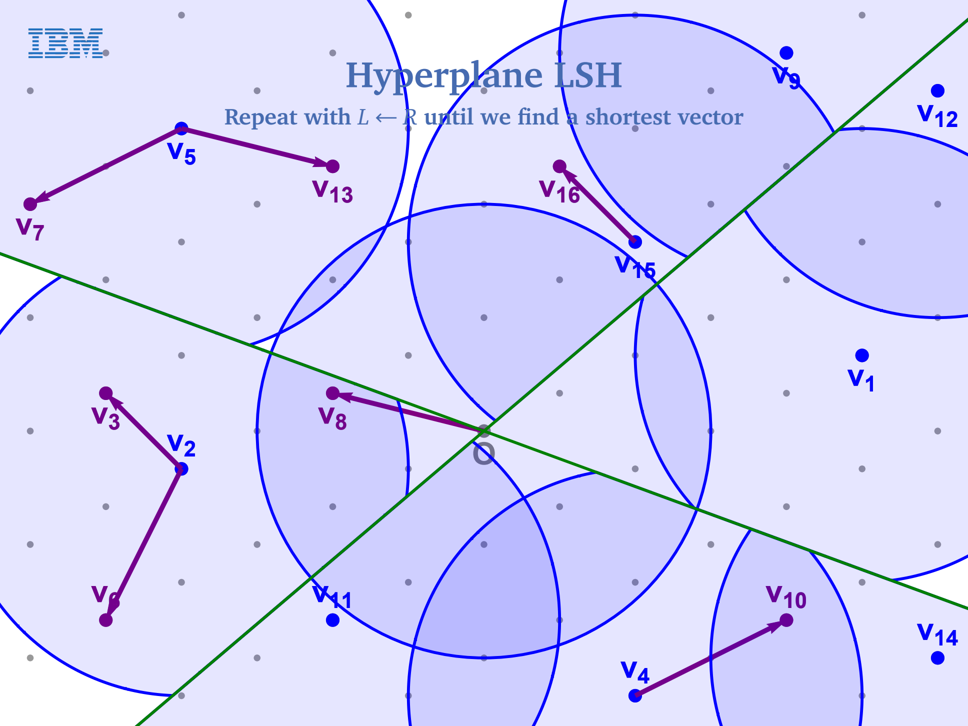
Split L into C and R within each region



IBM

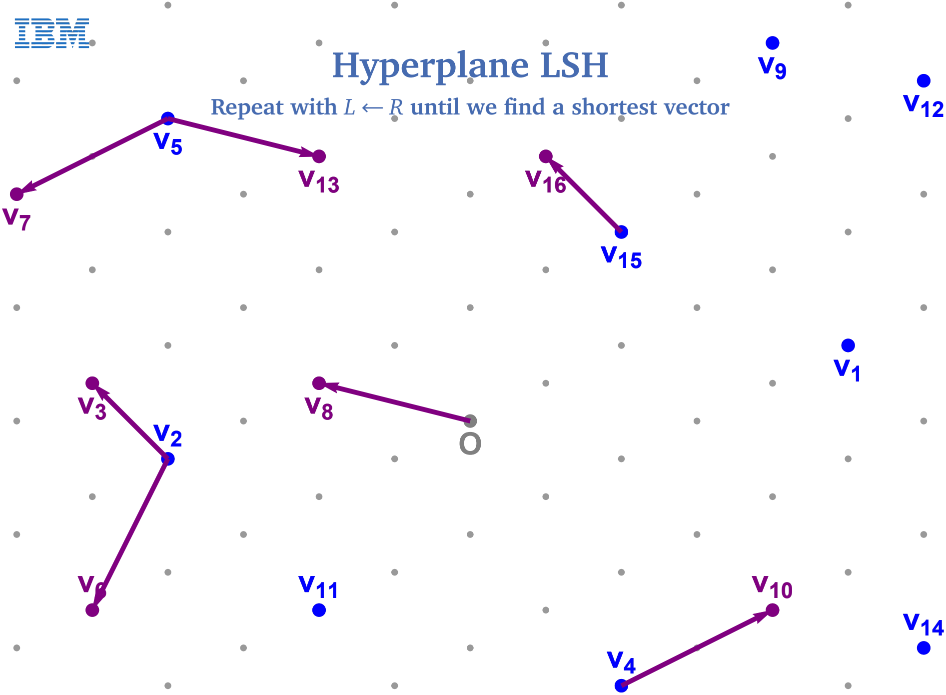
Hyperplane LSH

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



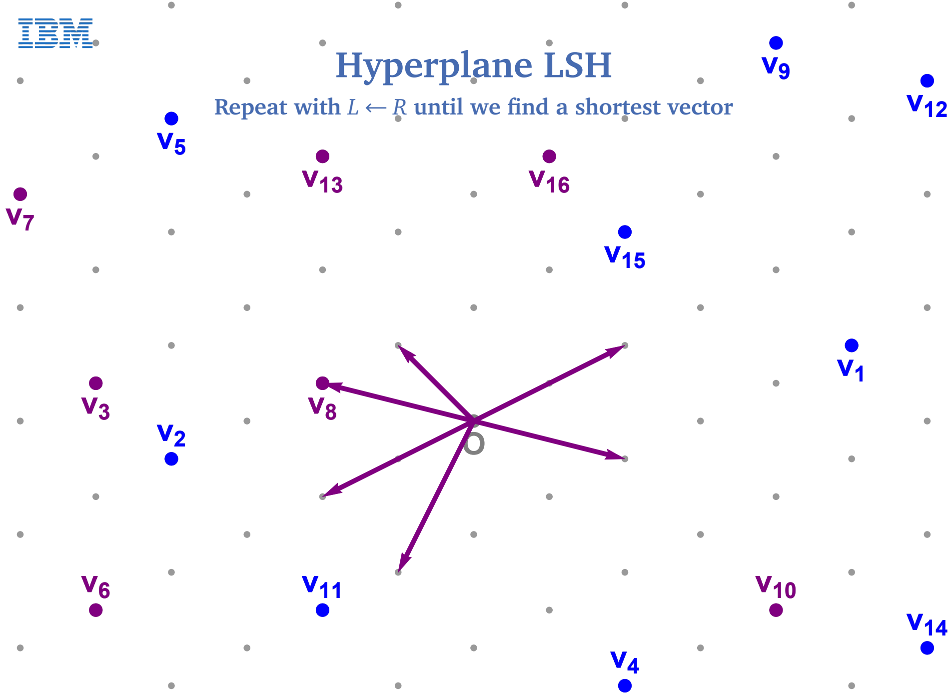
Hyperplane LSH

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



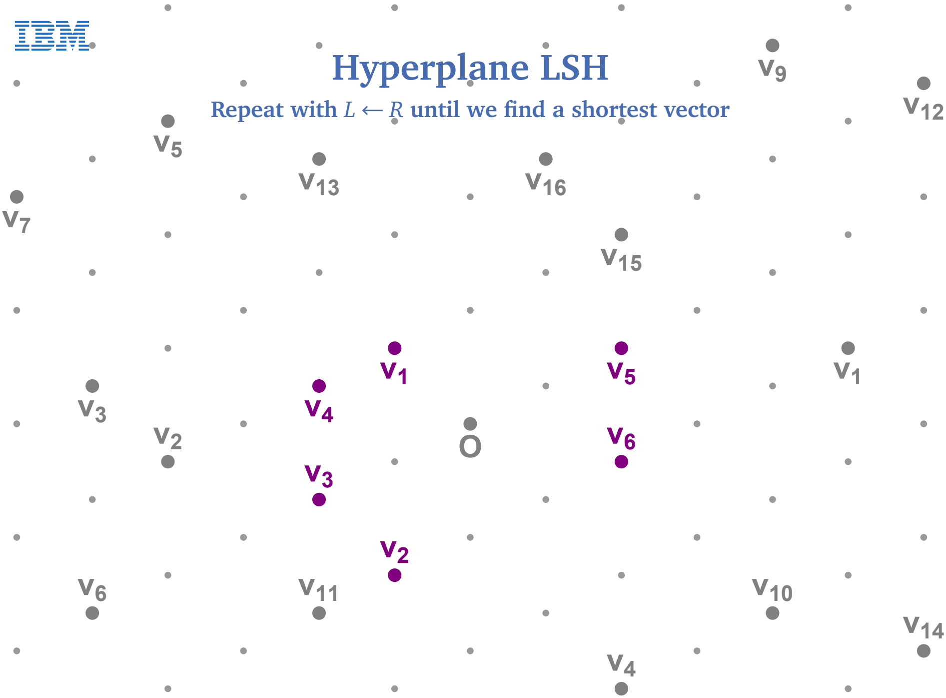
Hyperplane LSH

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

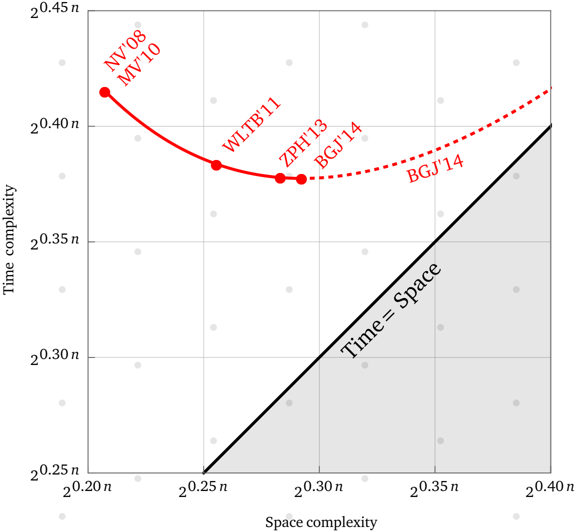
- Two parameters to tune
  - $k = O(n)$: Number of hyperplanes, leading to 2^k regions
  - $t = 2^{O(n)}$: Number of different, independent "hash tables"
- Space complexity: $2^{0.337n+o(n)}$
  - Number of vectors: $2^{0.208n+o(n)}$
  - Number of hash tables: $2^{0.129n+o(n)}$
  - Each hash table contains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

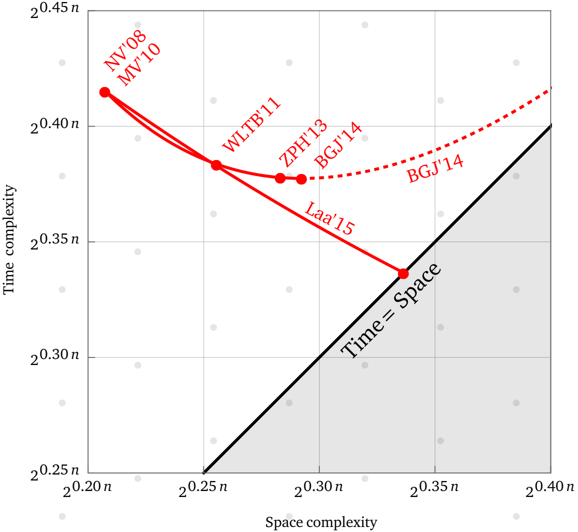
Hyperplane LSH

Space/time trade-off



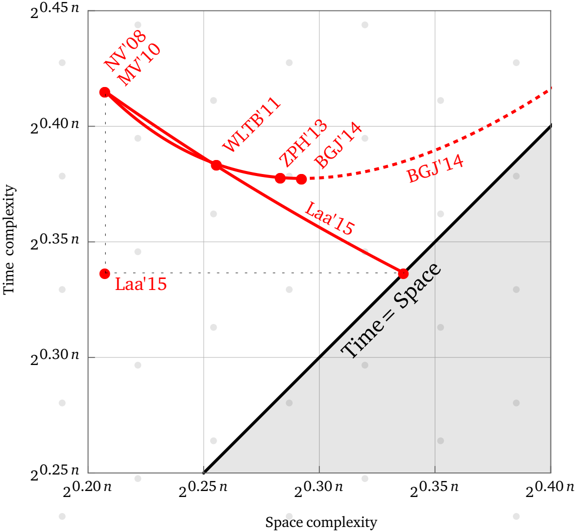
Hyperplane LSH

Space/time trade-off



Hyperplane LSH

Space/time trade-off

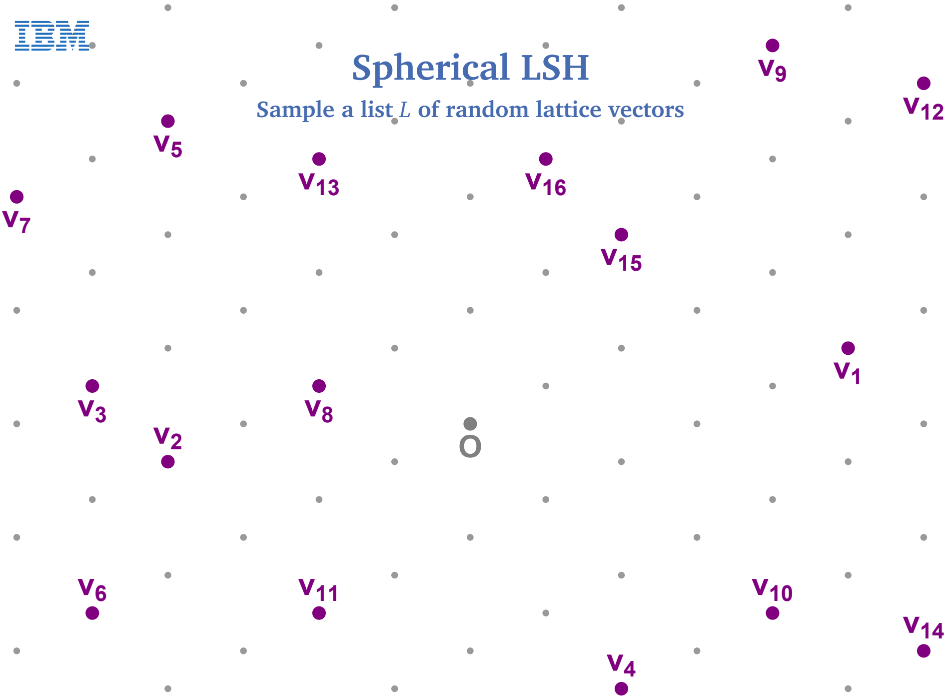


Spherical LSH

Sample a list L of random lattice vectors

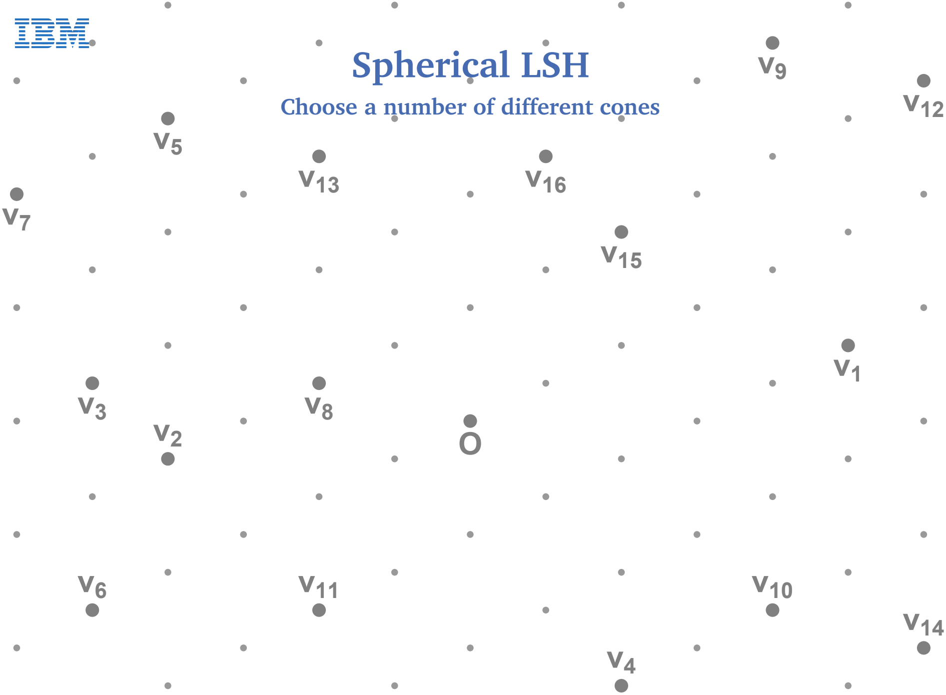
Spherical LSH

Sample a list L of random lattice vectors



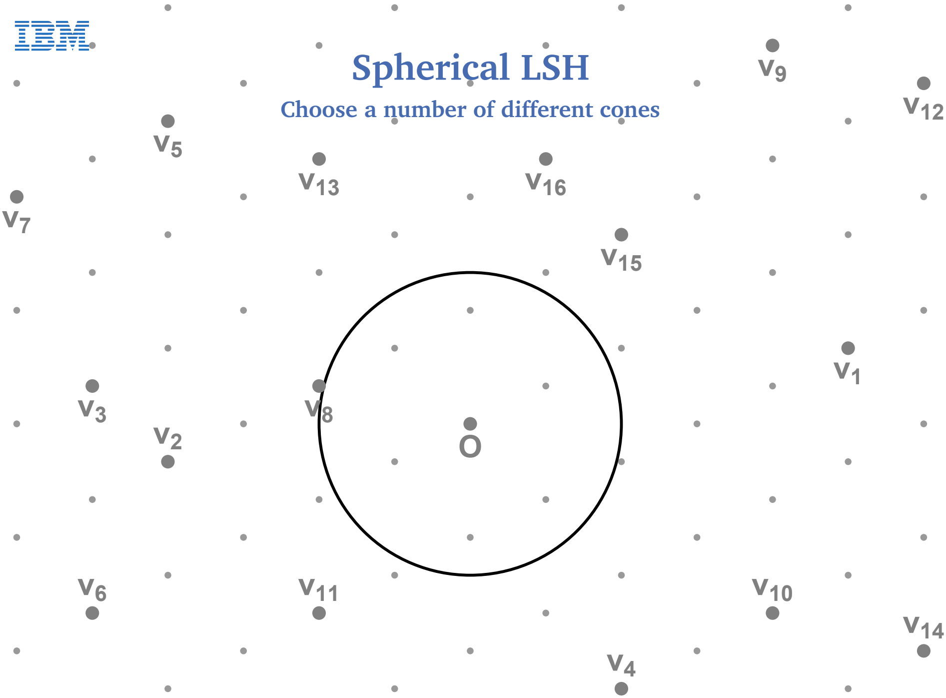
Spherical LSH

Choose a number of different cones



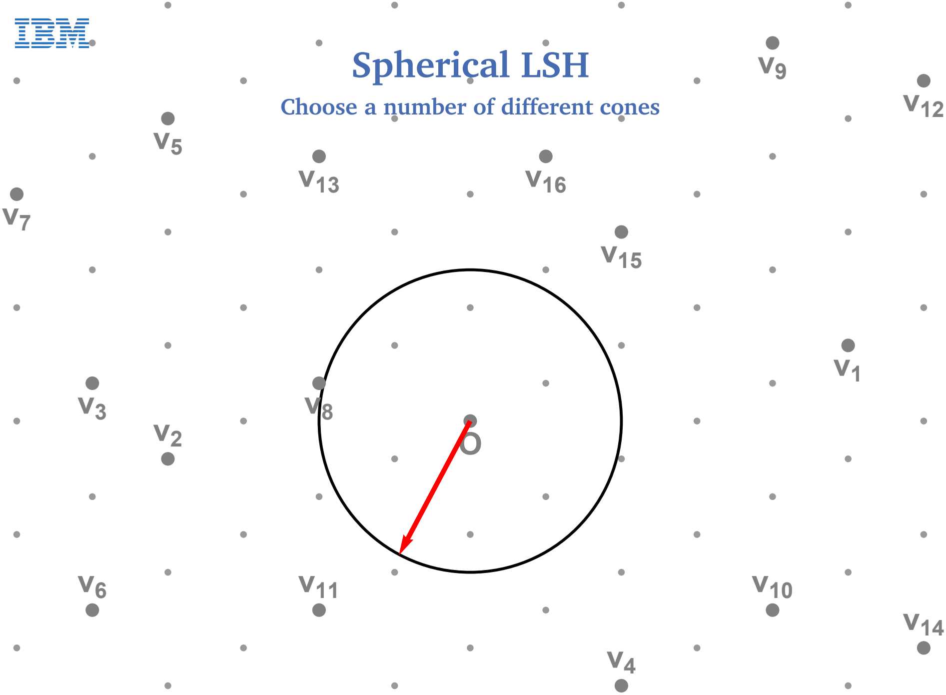
Spherical LSH

Choose a number of different cones



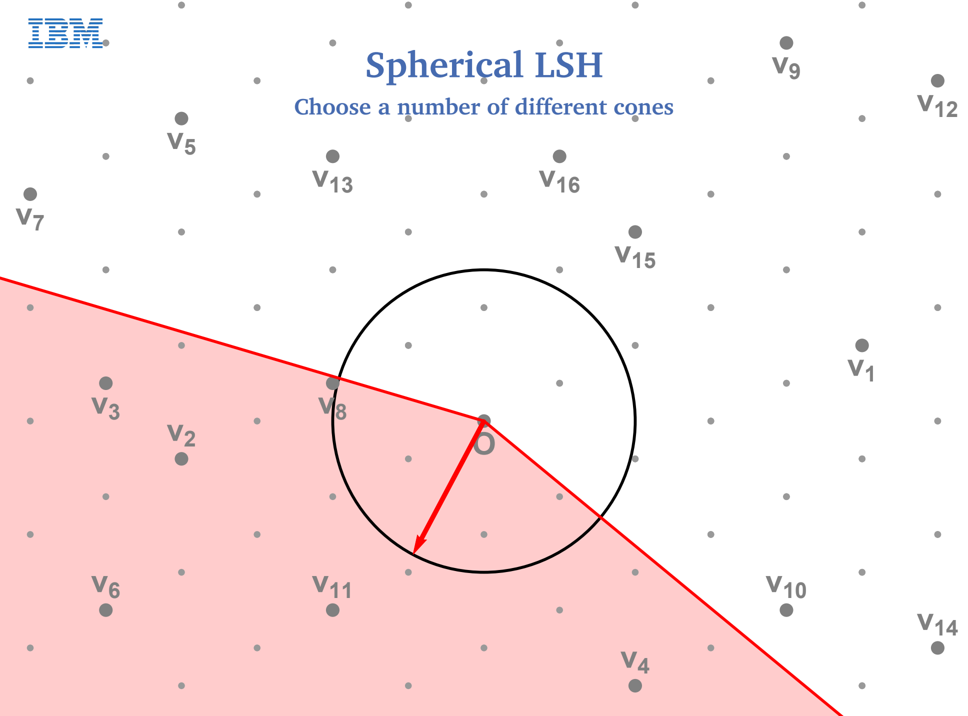
Spherical LSH

Choose a number of different cones



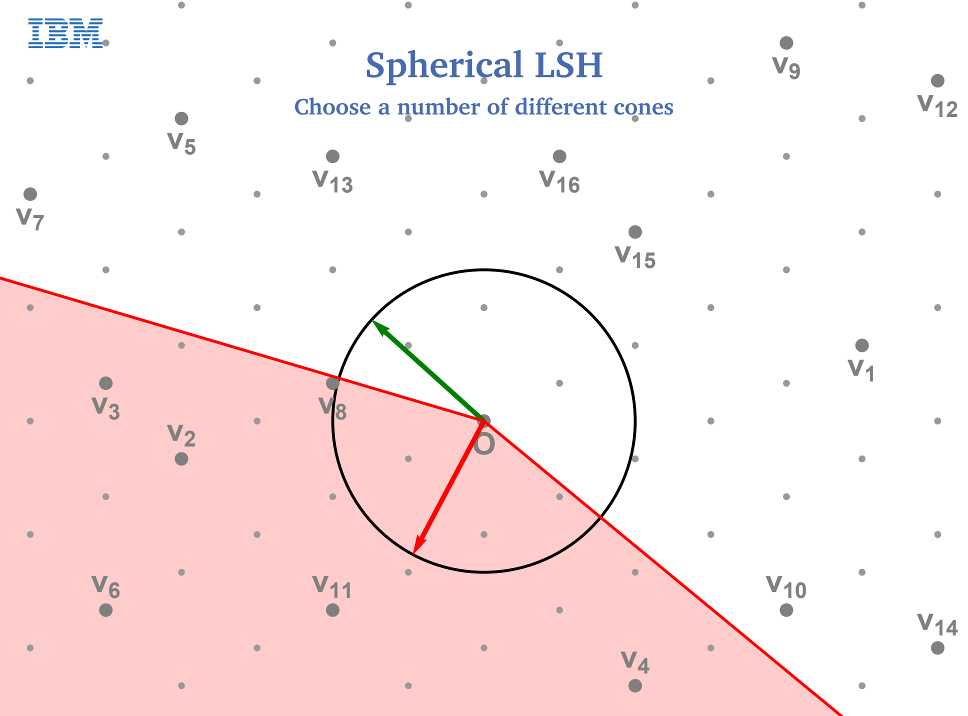
Spherical LSH

Choose a number of different cones



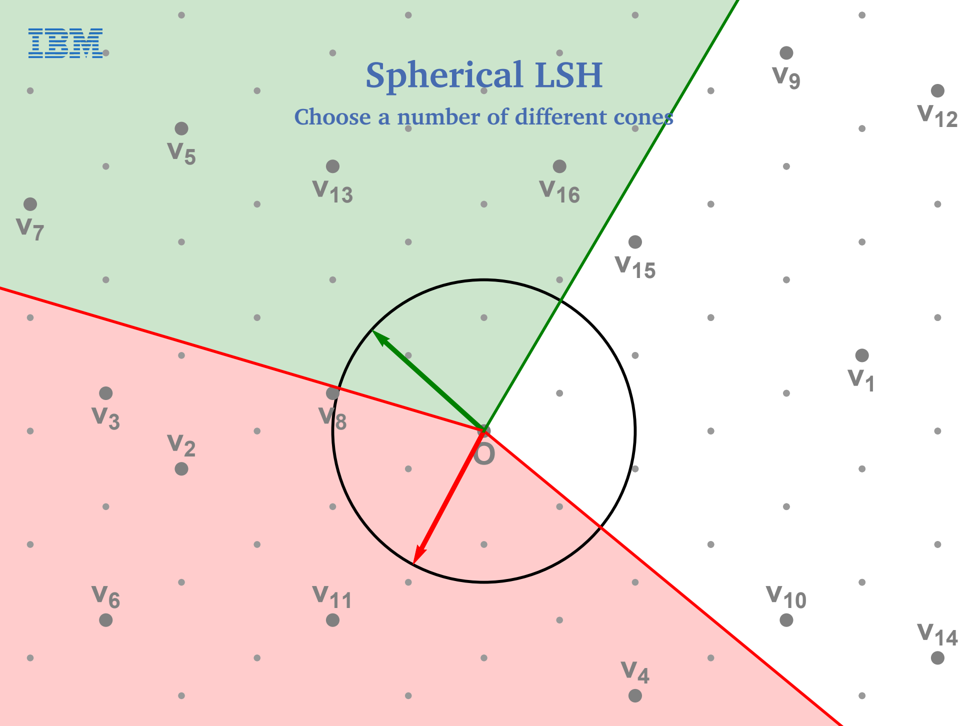
Spherical LSH

Choose a number of different cones



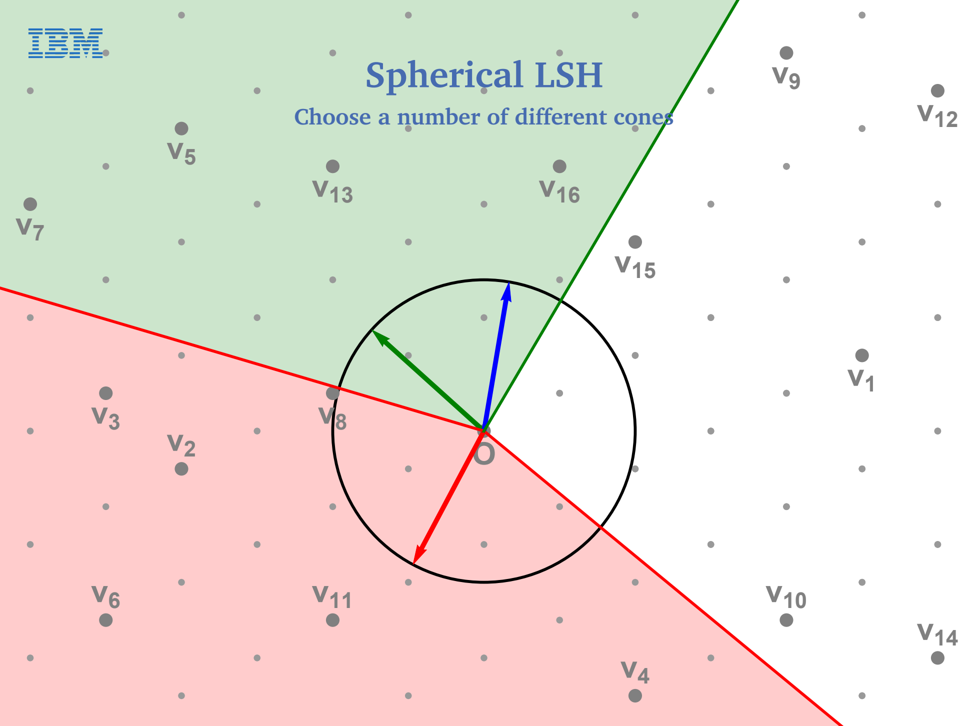
Spherical LSH

Choose a number of different cones



Spherical LSH

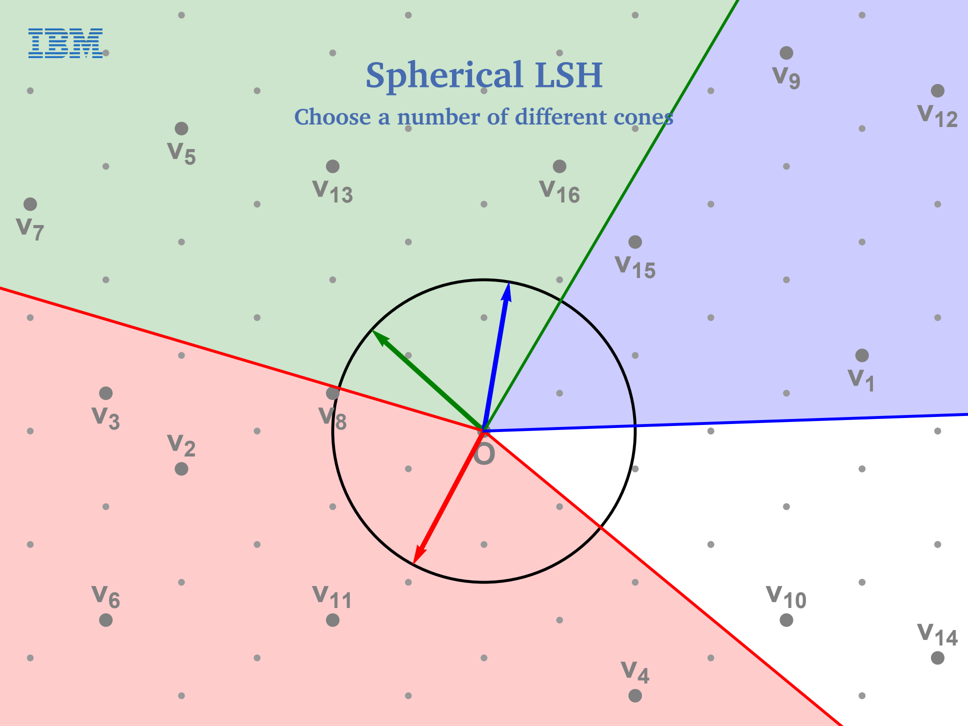
Choose a number of different cones

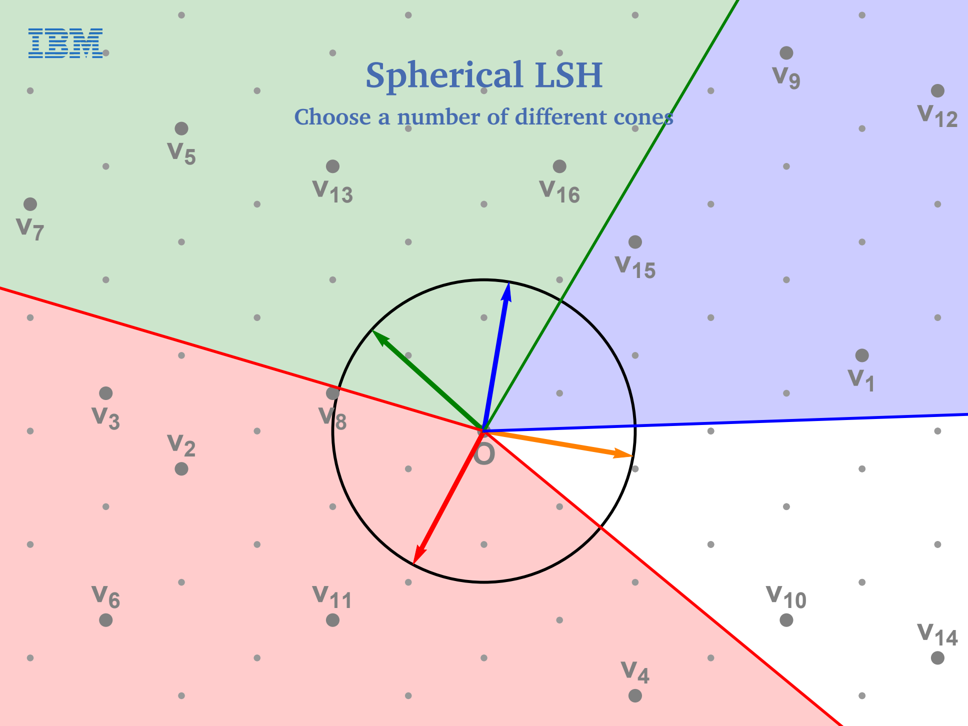


IBM

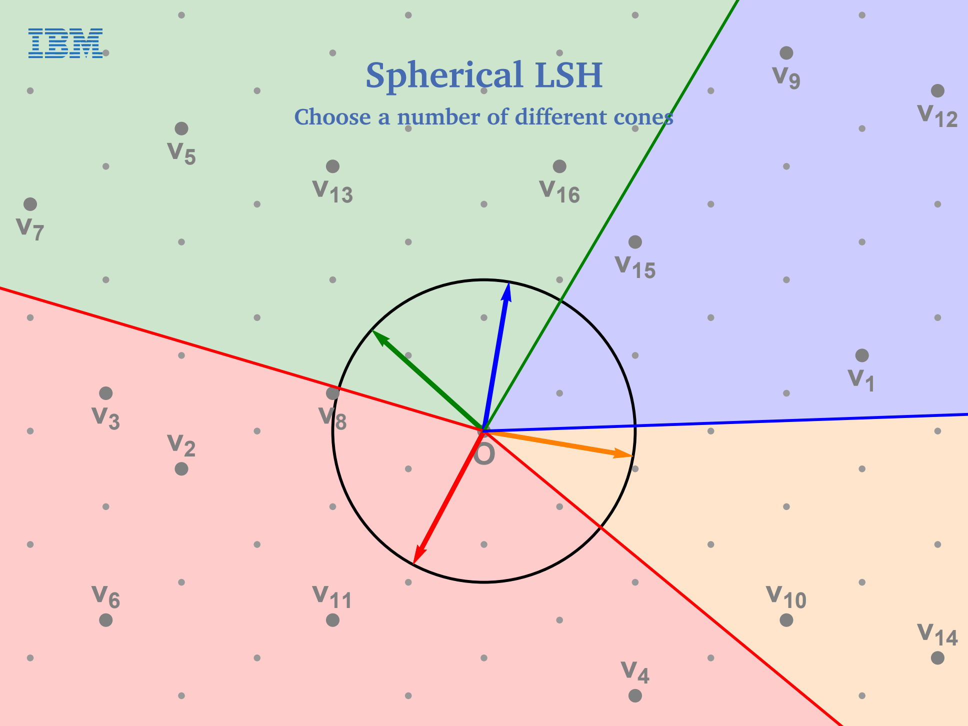
Spherical LSH

Choose a number of different cones



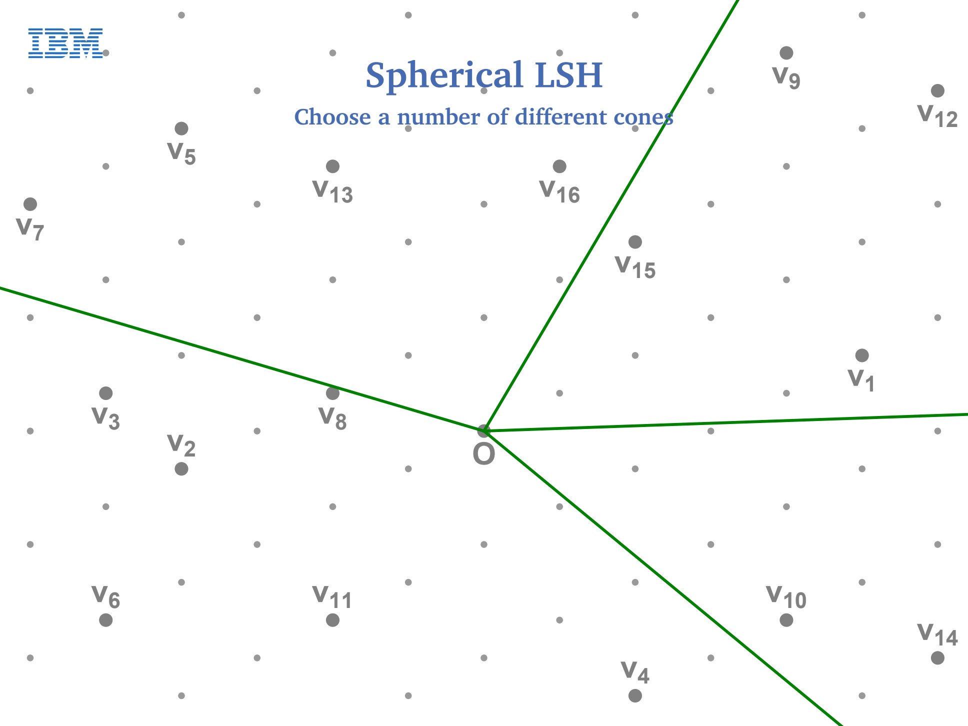


Spherical LSH
Choose a number of different cones
v5
v13
v16
v7
v15
v9
v12
v1
v3
v8
v2
O
v6
v11
v10
v14
v4


Spherical LSH
Choose a number of different cones
v9
v12
v5
v13
v16
v7
v15
v1
v3
v8
v2
O
v6
v11
v10
v14
v4

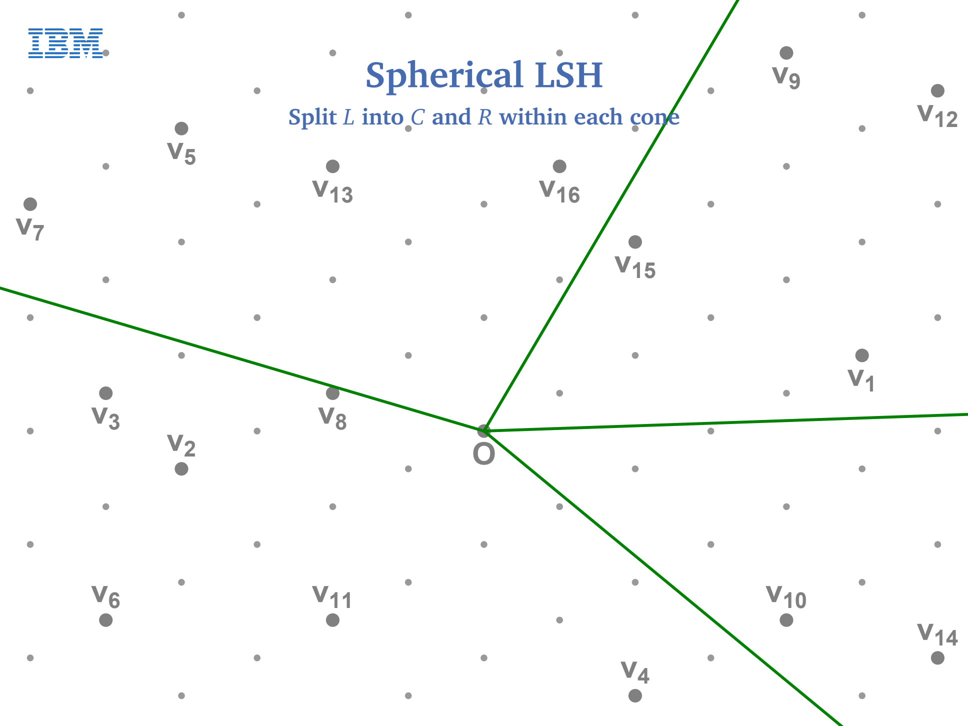
Spherical LSH

Choose a number of different cones



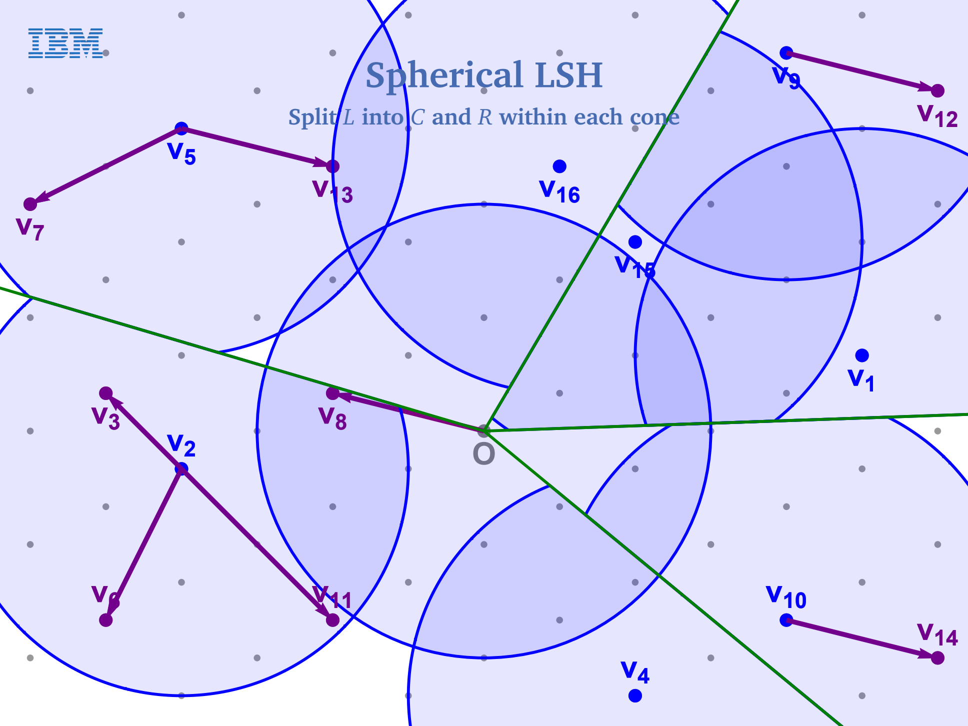
Spherical LSH

Split L into C and R within each cone



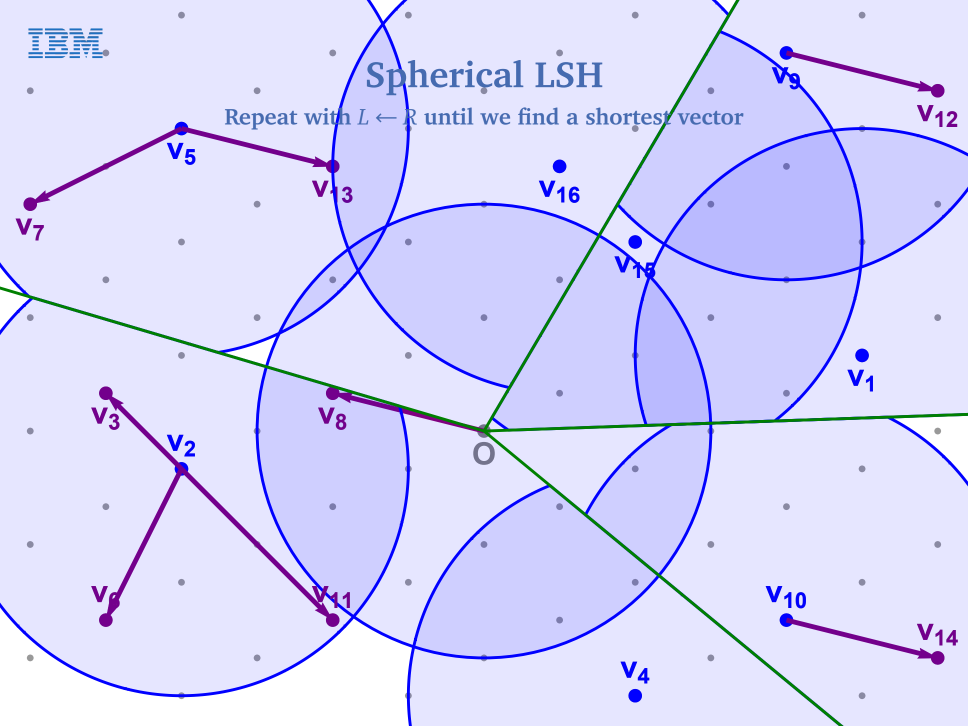
Spherical LSH

Split L into C and R within each cone



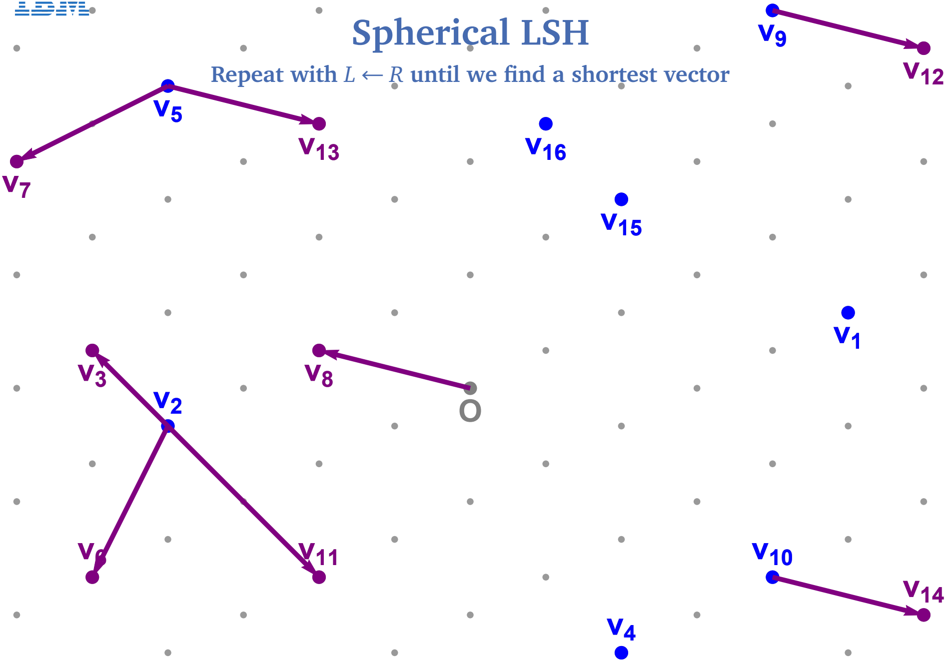
Spherical LSH

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



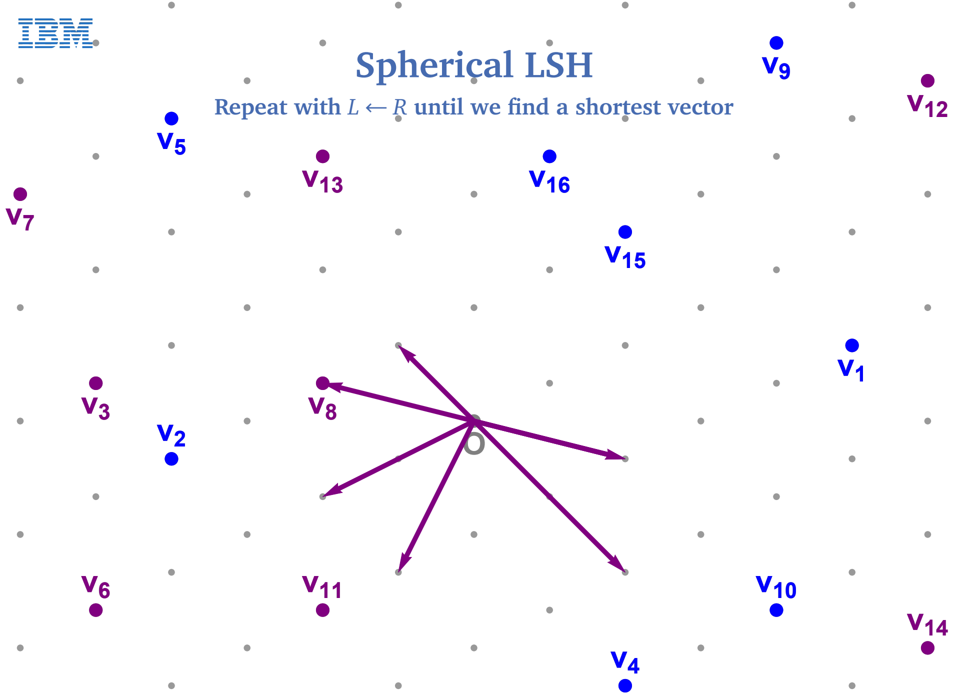
Spherical LSH

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



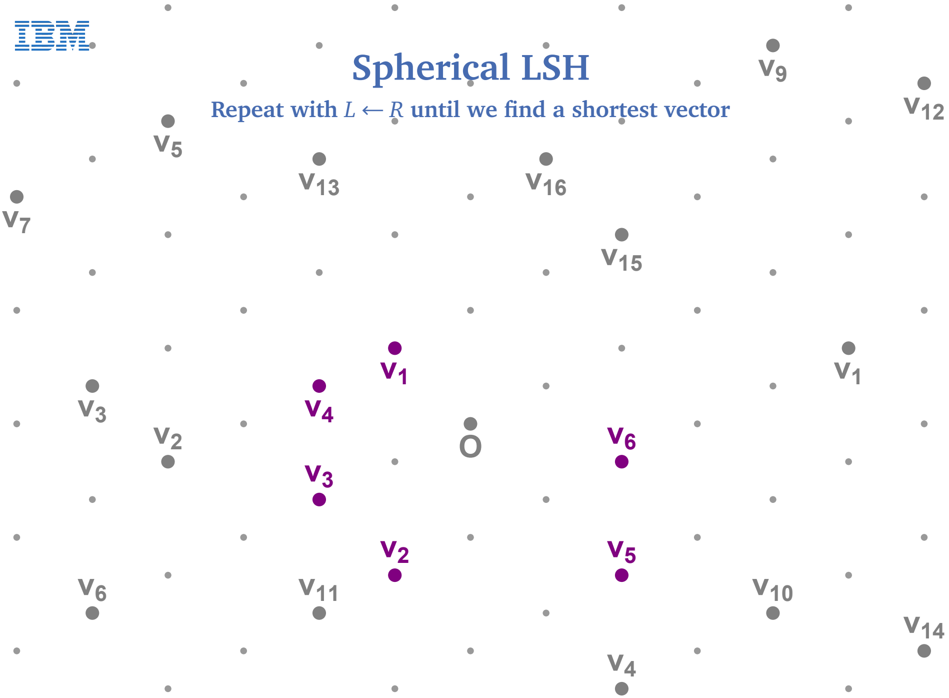
Spherical LSH

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



Spherical LSH

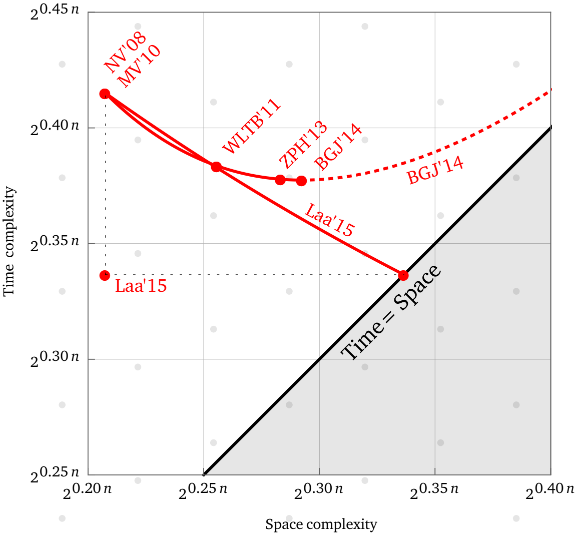
Repeat with $L \leftarrow R$ until we find a shortest vector



IBM

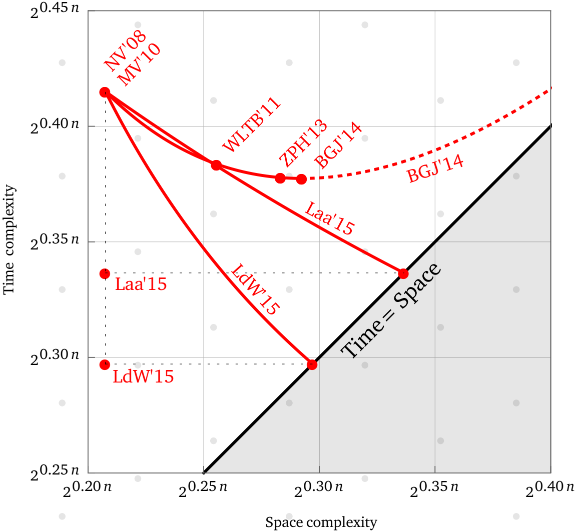
Spherical LSH

Space/time trade-off



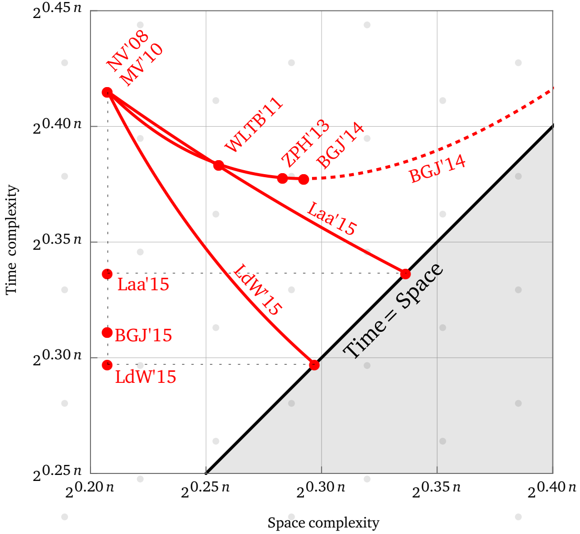
Spherical LSH

Space/time trade-off



May and Ozerov's NNS method

Space/time trade-off

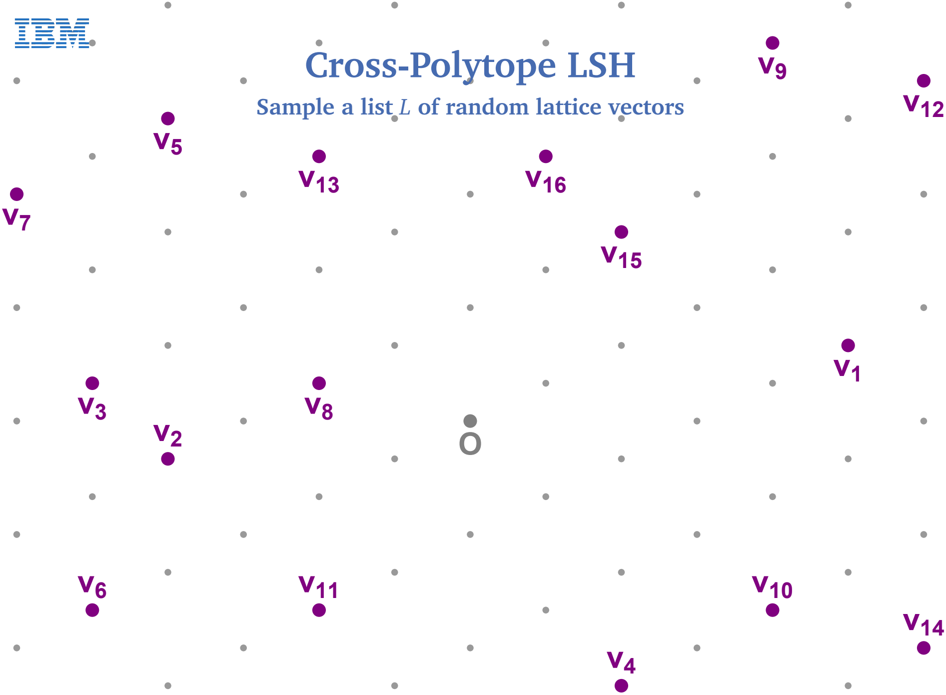


Cross-Polytope LSH

Sample a list L of random lattice vectors

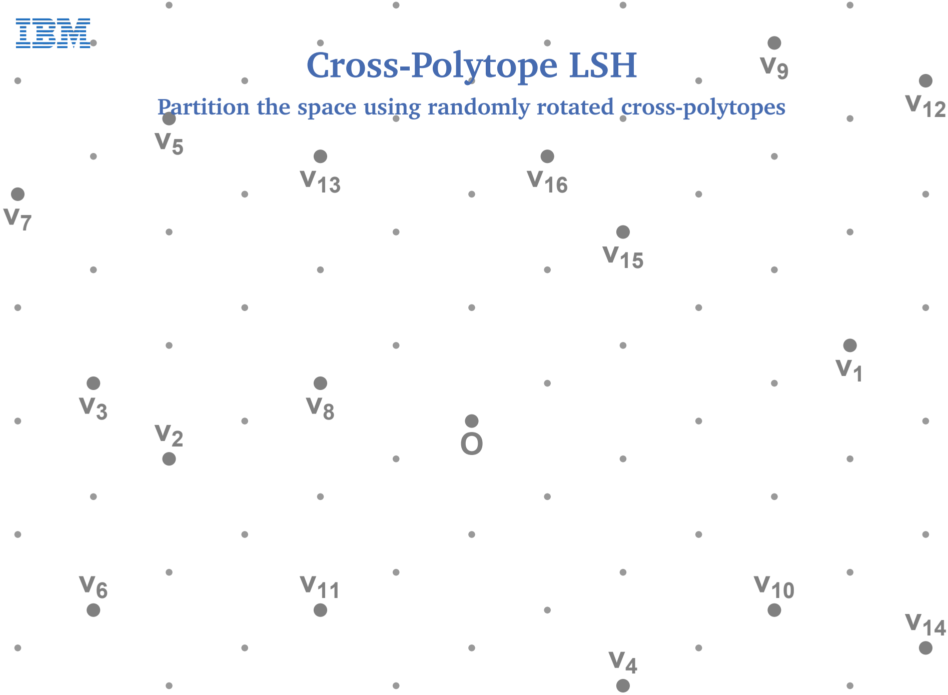
Cross-Polytope LSH

Sample a list L of random lattice vectors



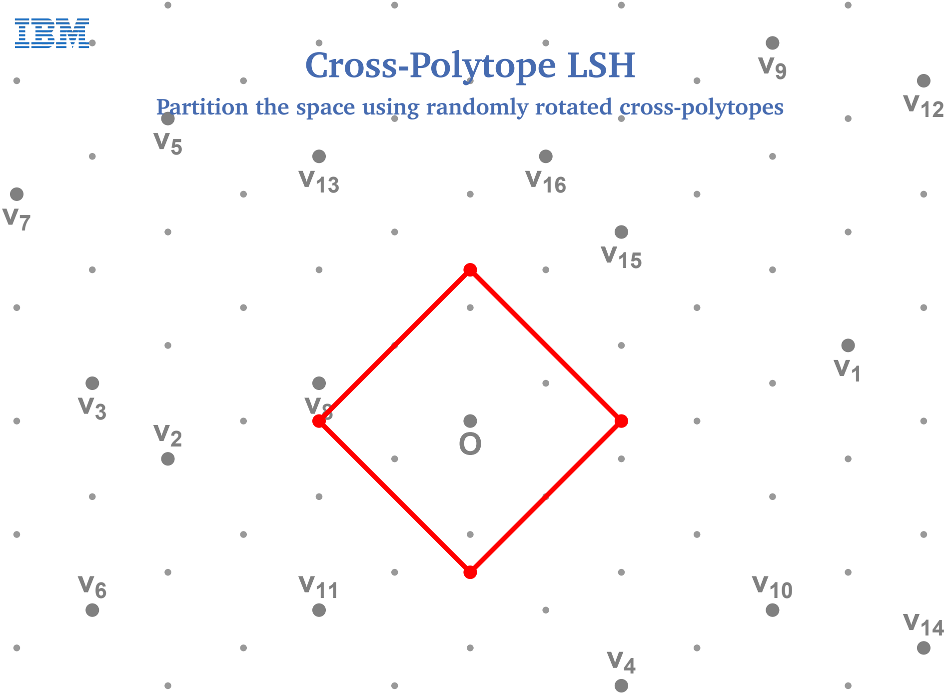
Cross-Polytope LSH

Partition the space using randomly rotated cross-polytopes



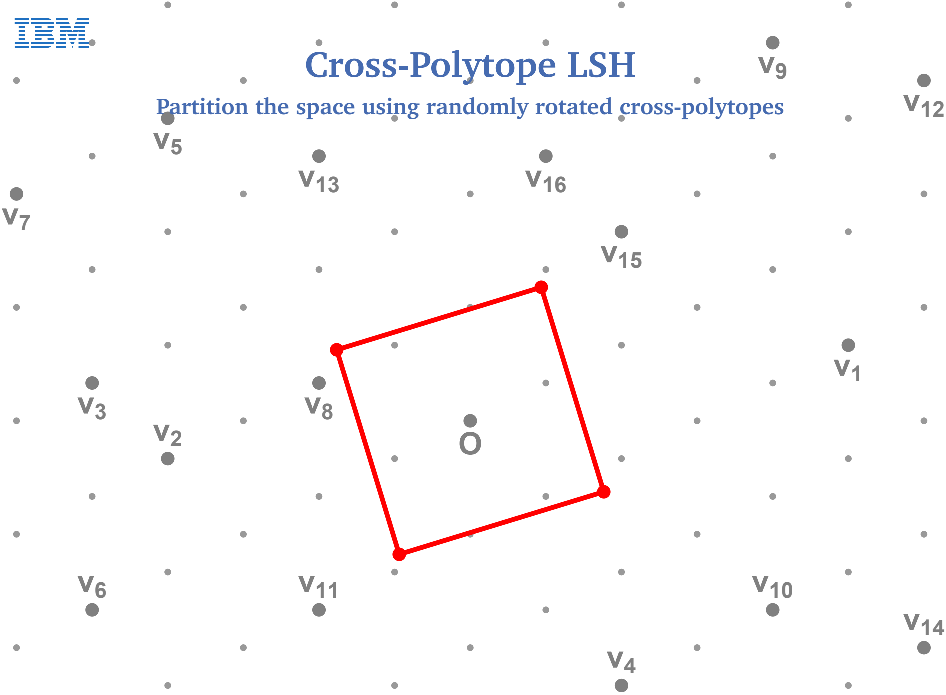
Cross-Polytope LSH

Partition the space using randomly rotated cross-polytopes



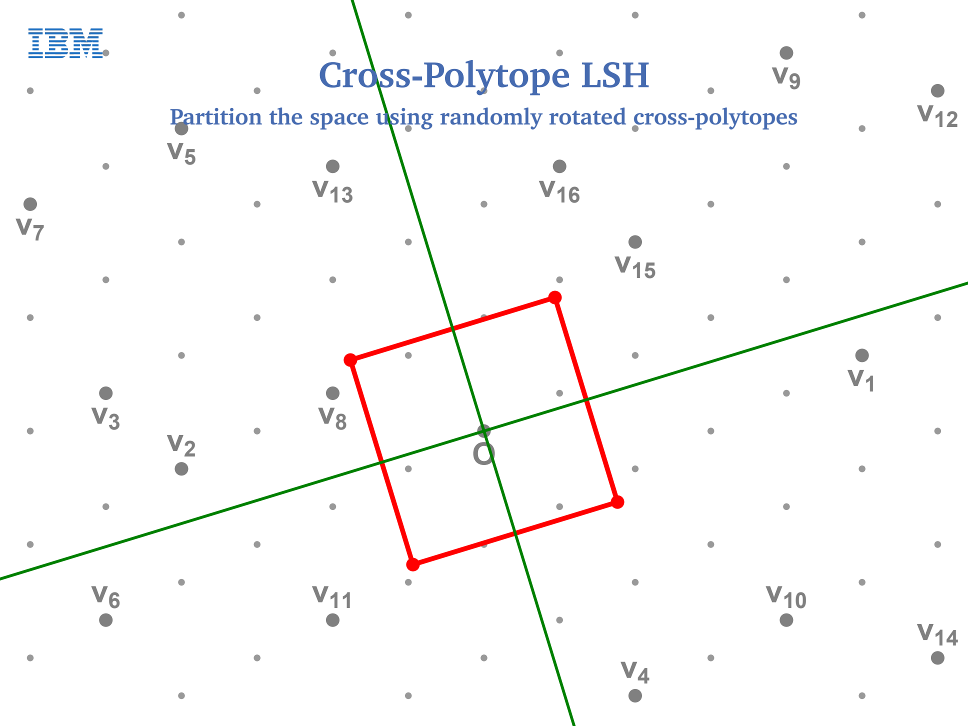
Cross-Polytope LSH

Partition the space using randomly rotated cross-polytopes



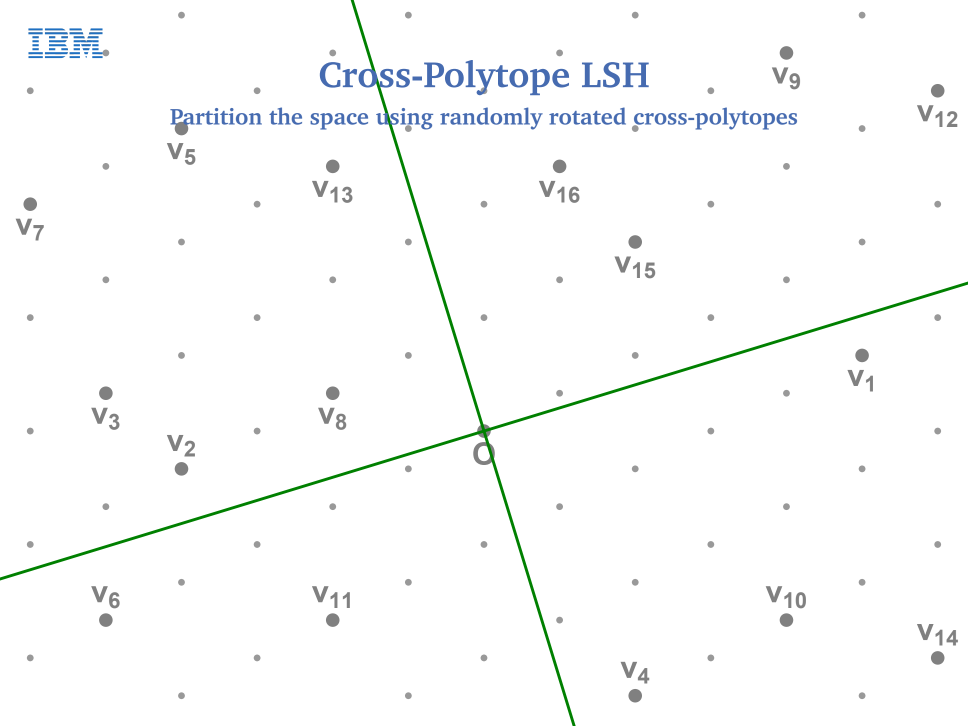
Cross-Polytope LSH

Partition the space using randomly rotated cross-polytopes



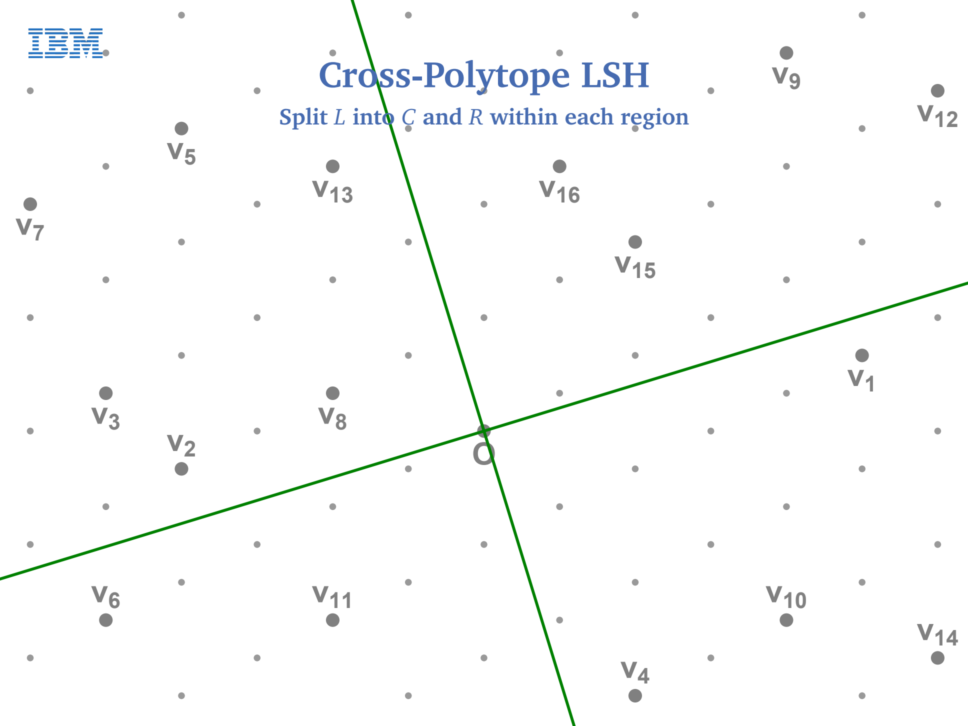
Cross-Polytope LSH

Partition the space using randomly rotated cross-polytopes



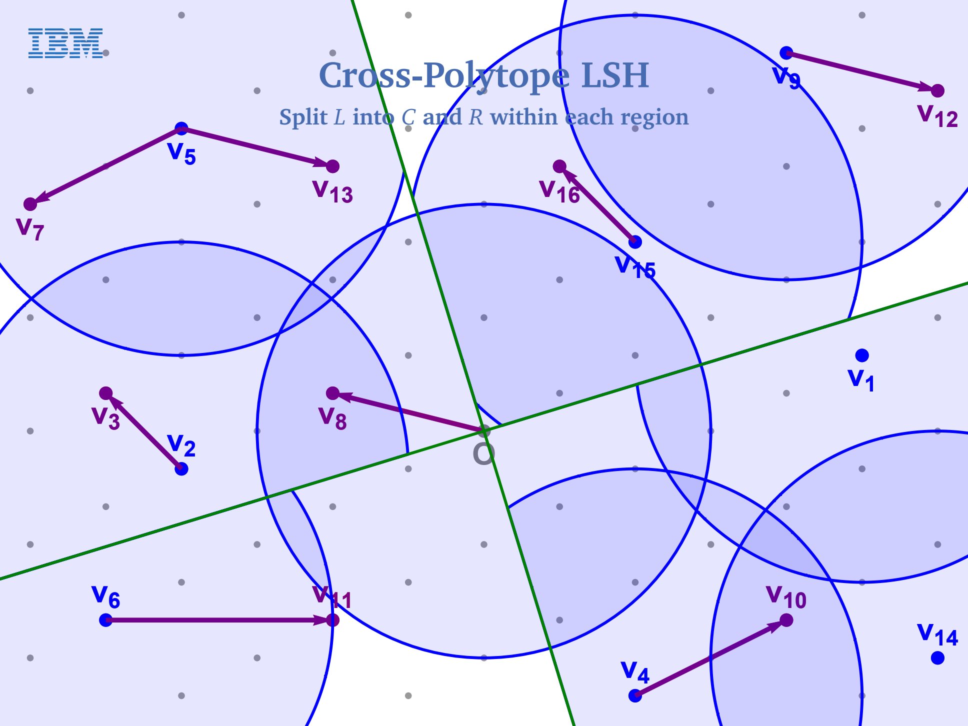
Cross-Polytope LSH

Split L into C and R within each region



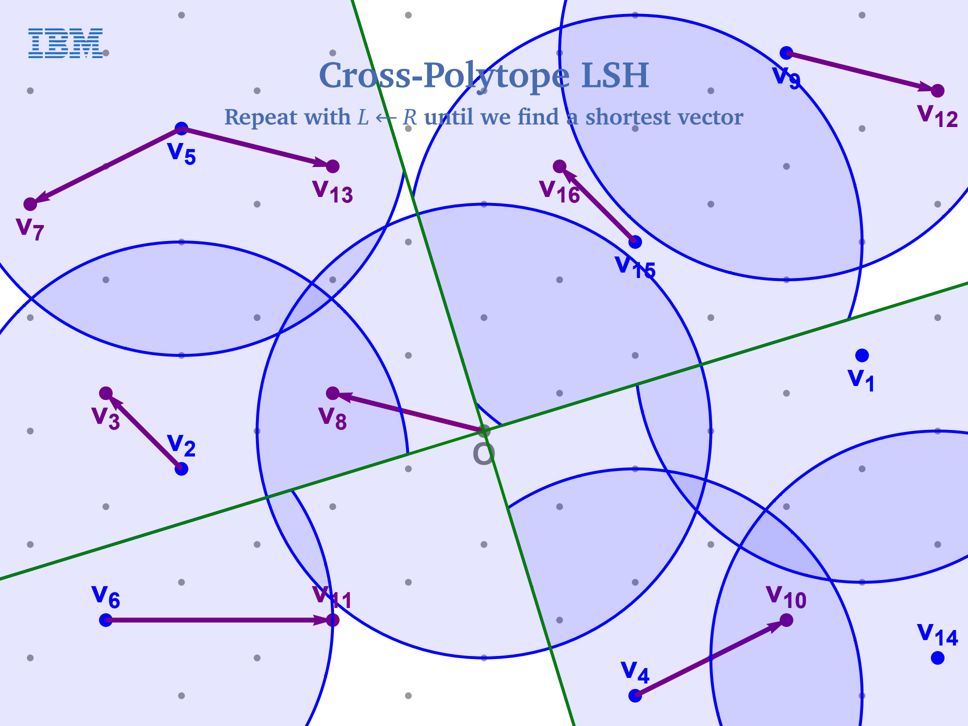
Cross-Polytope LSH

Split L into C and R within each region



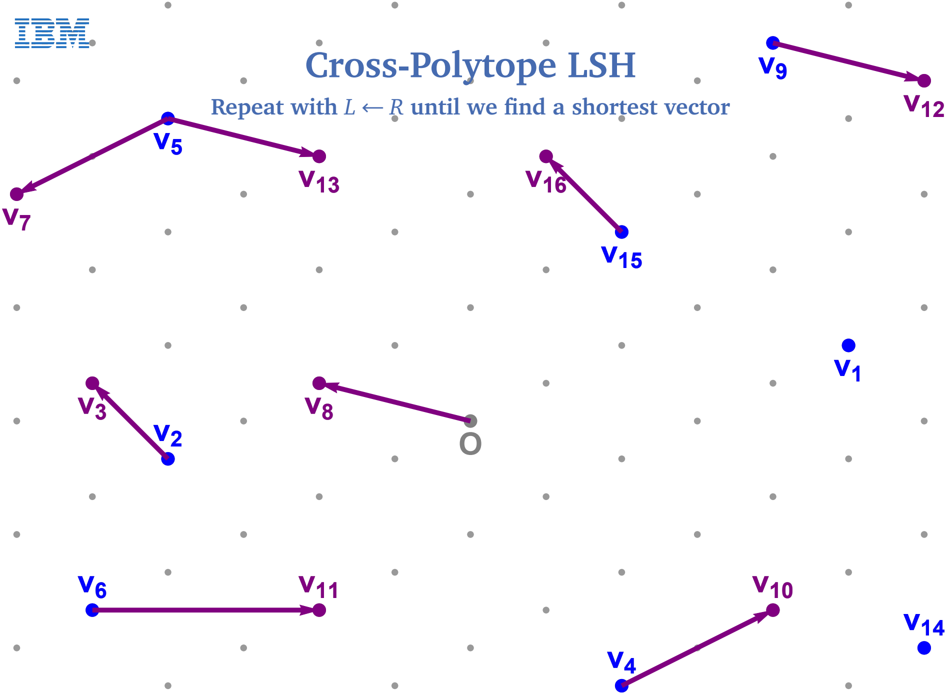
Cross-Polytope LSH

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



Cross-Polytope LSH

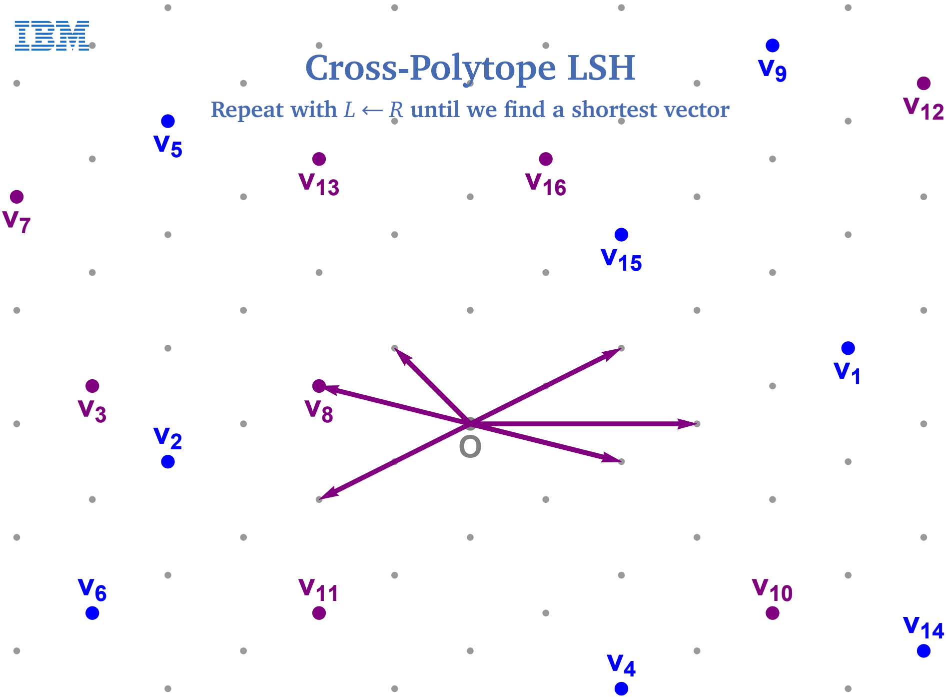
Repeat with $L \leftarrow R$ until we find a shortest vector



IBM

Cross-Polytope LSH

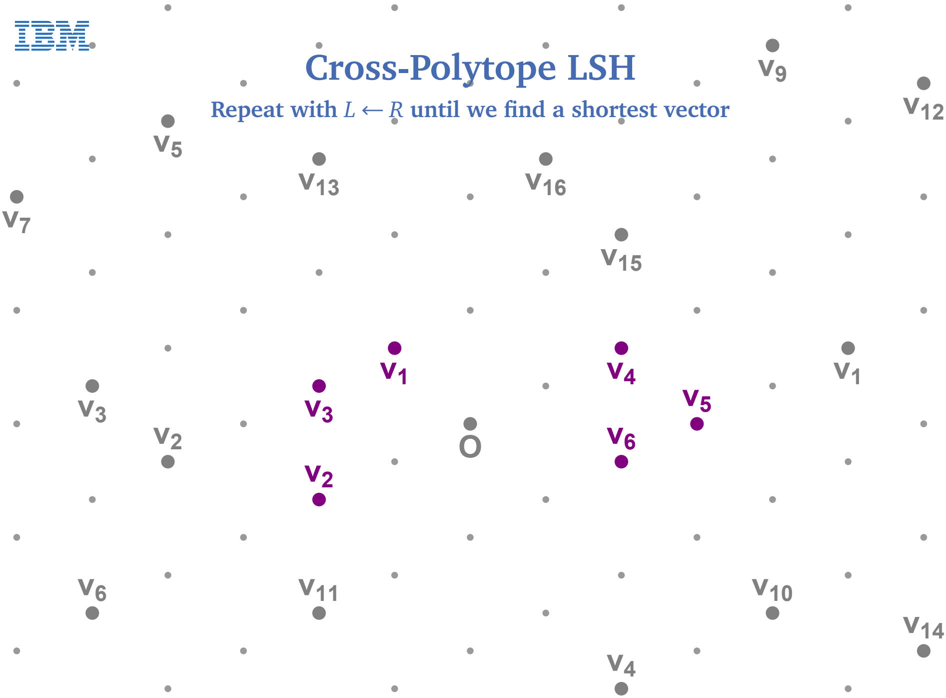
Repeat with $L \leftarrow R$ until we find a shortest vector



IBM

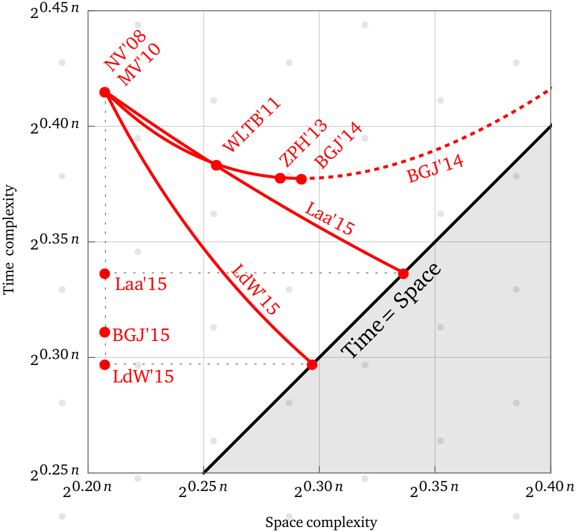
Cross-Polytope LSH

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



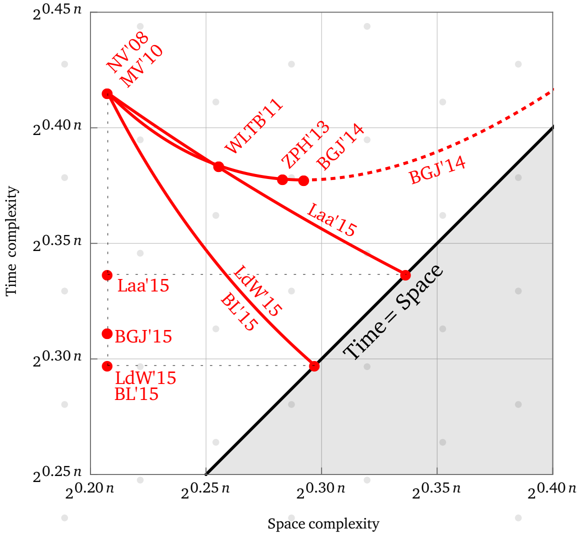
Cross-Polytope LSH

Space/time trade-off



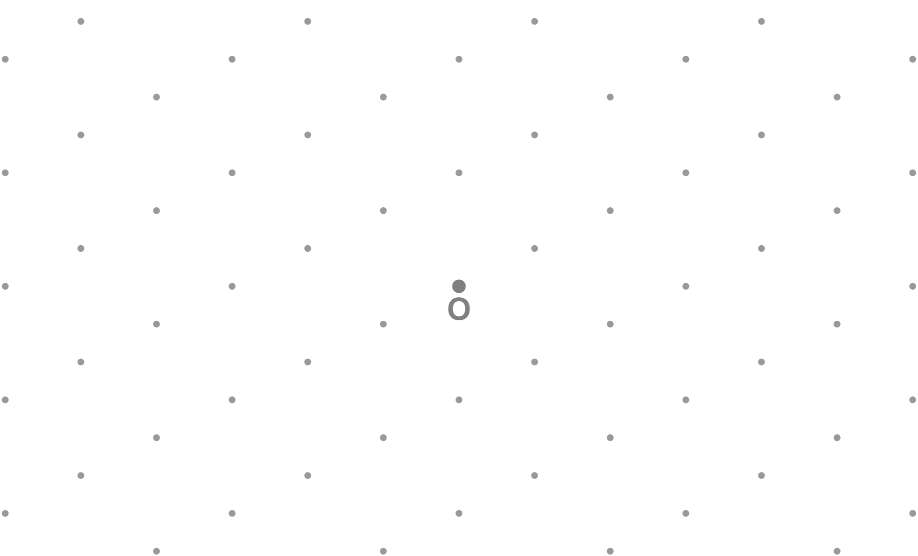
Cross-Polytope LSH

Space/time trade-off



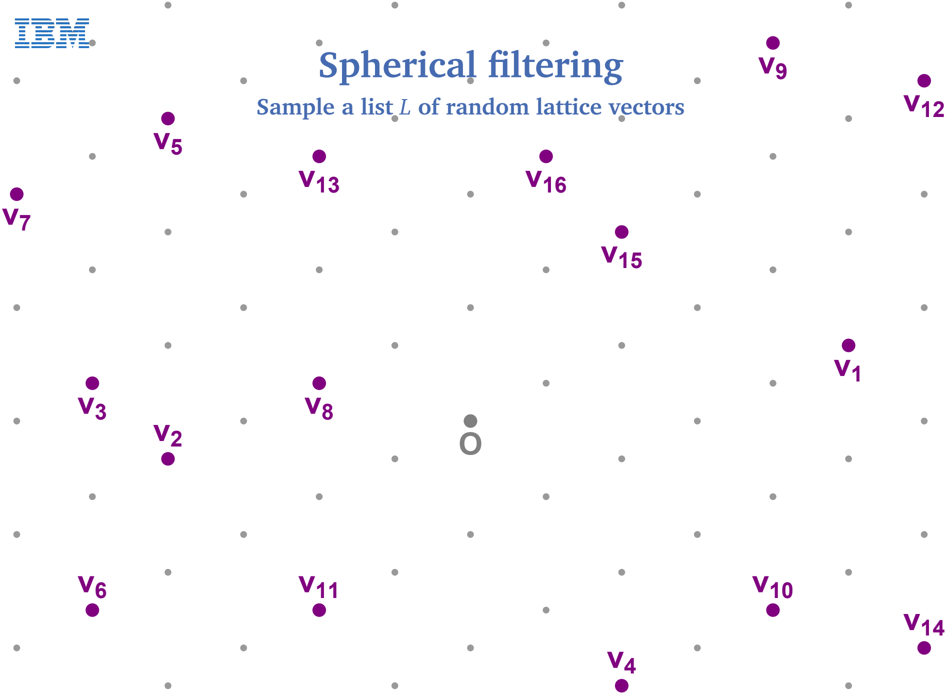
Spherical filtering

Sample a list L of random lattice vectors



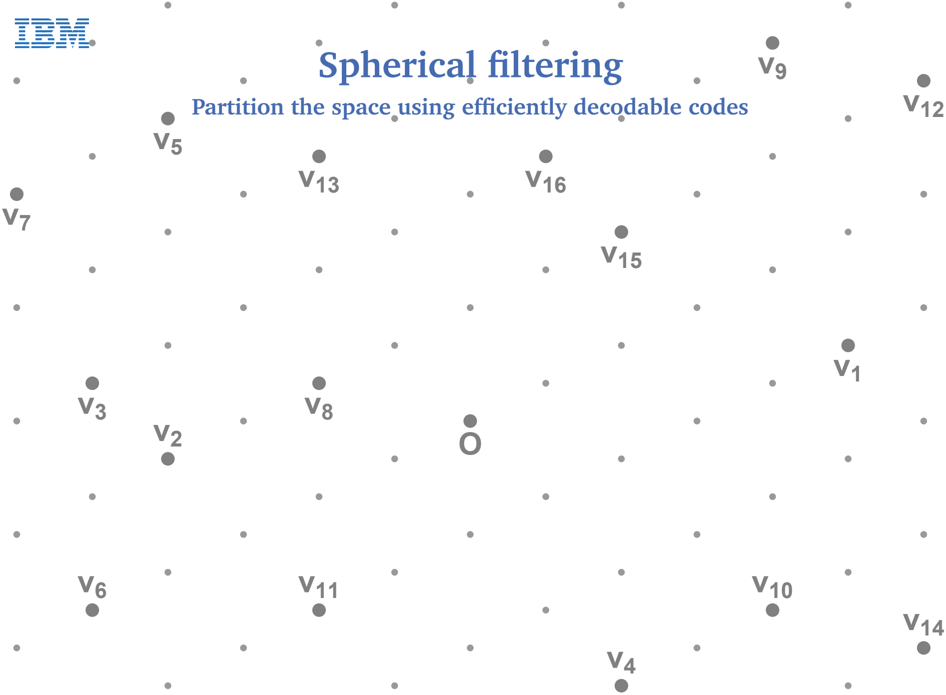
Spherical filtering

Sample a list L of random lattice vectors



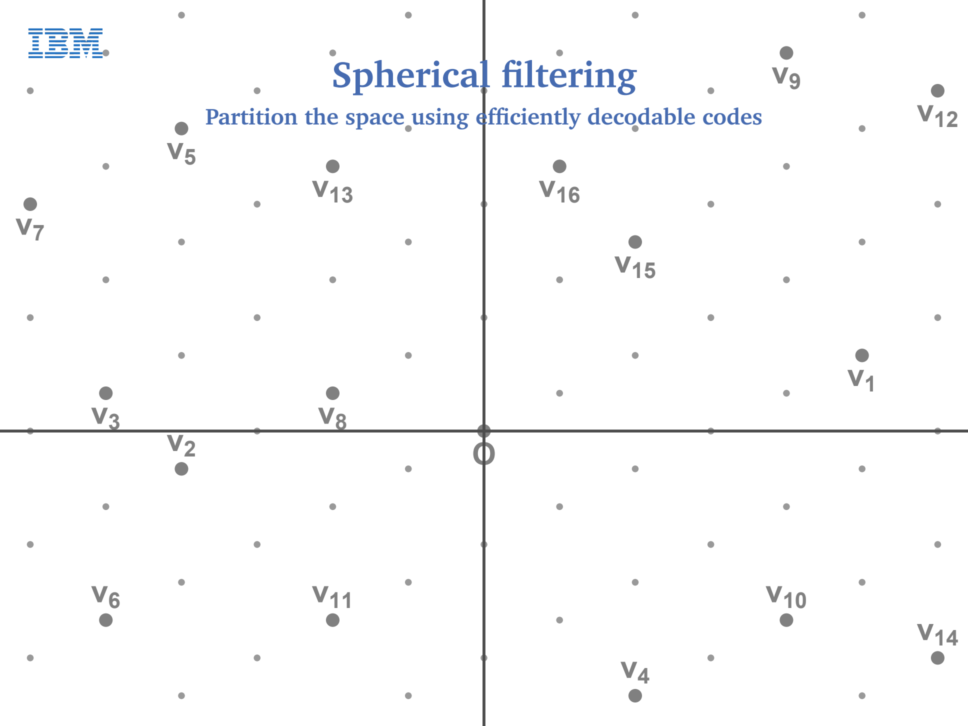
Spherical filtering

Partition the space using efficiently decodable codes



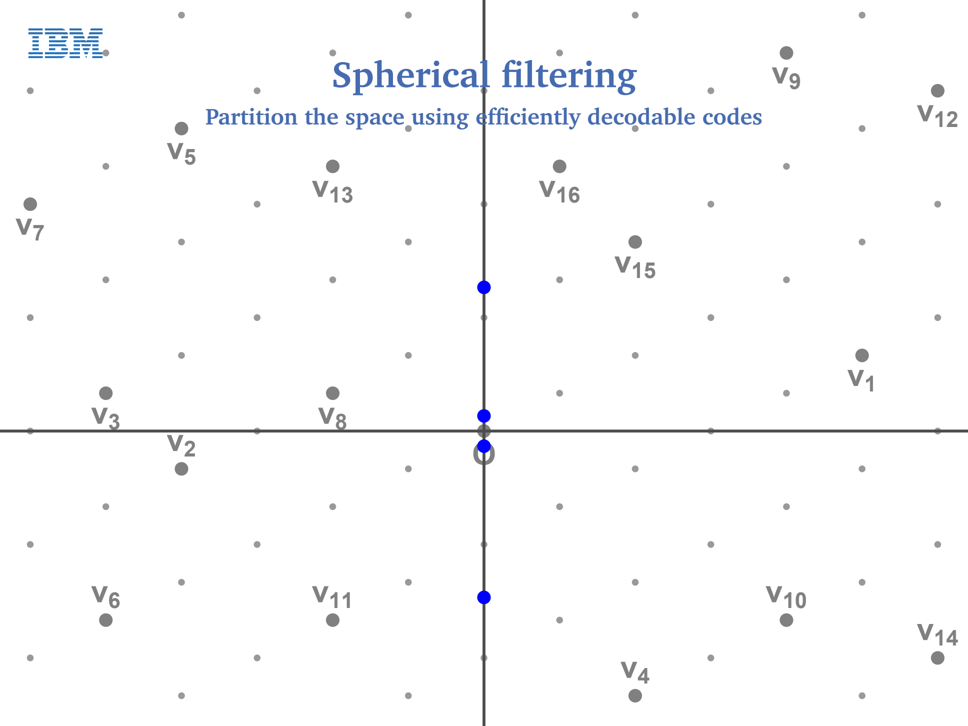
Spherical filtering

Partition the space using efficiently decodable codes



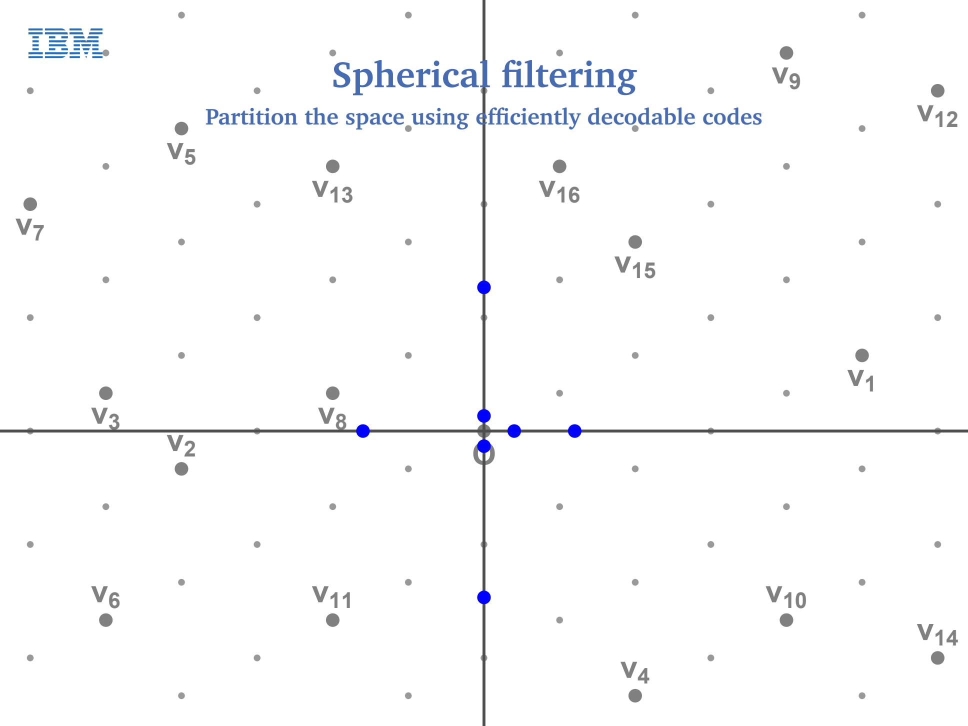
Spherical filtering

Partition the space using efficiently decodable codes



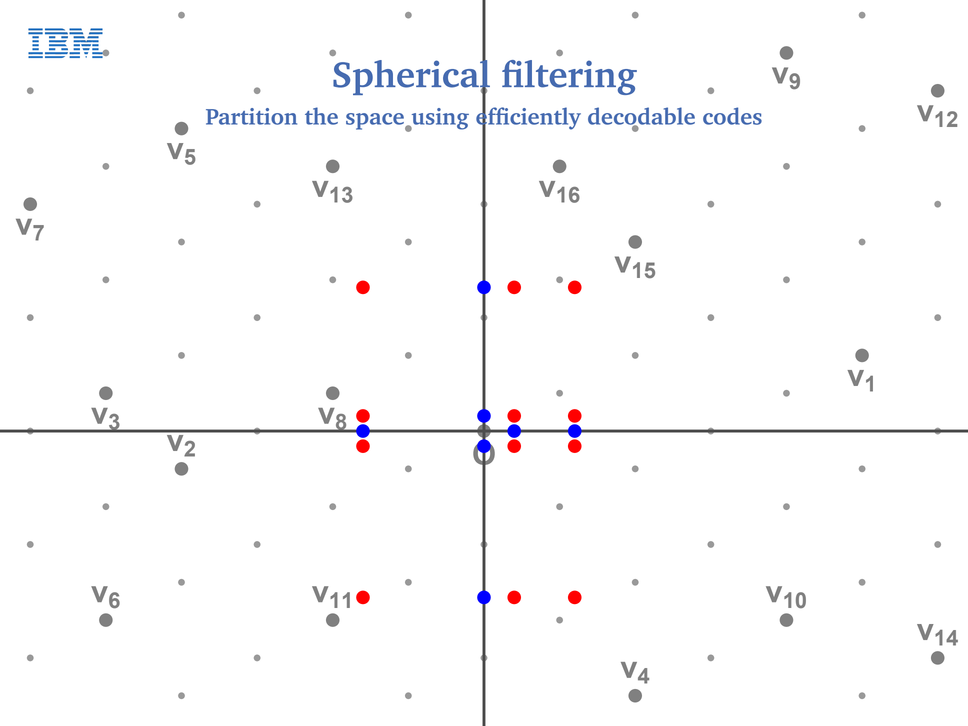
Spherical filtering

Partition the space using efficiently decodable codes



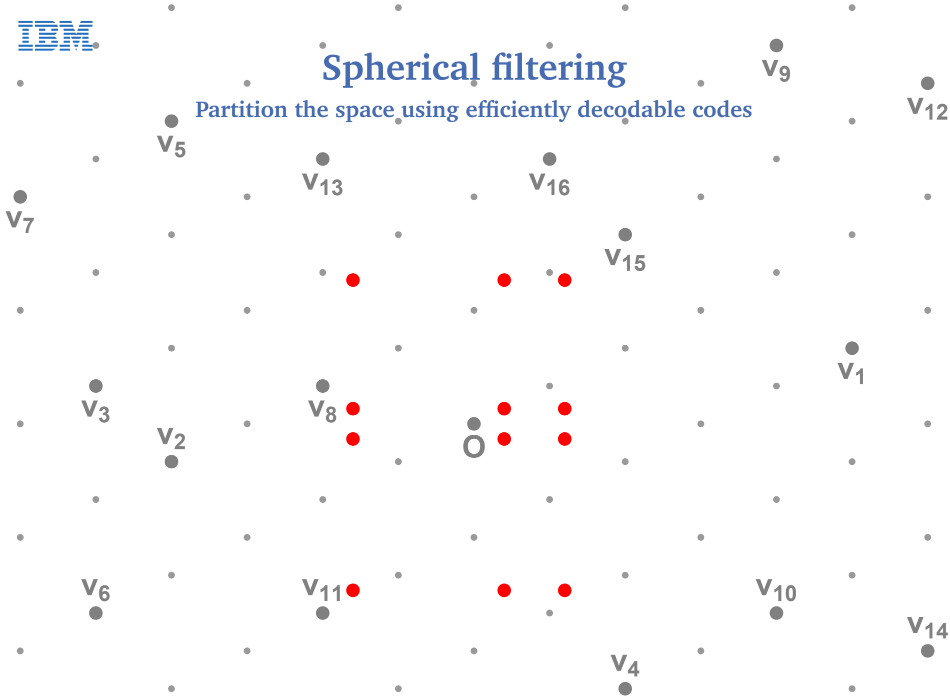
Spherical filtering

Partition the space using efficiently decodable codes



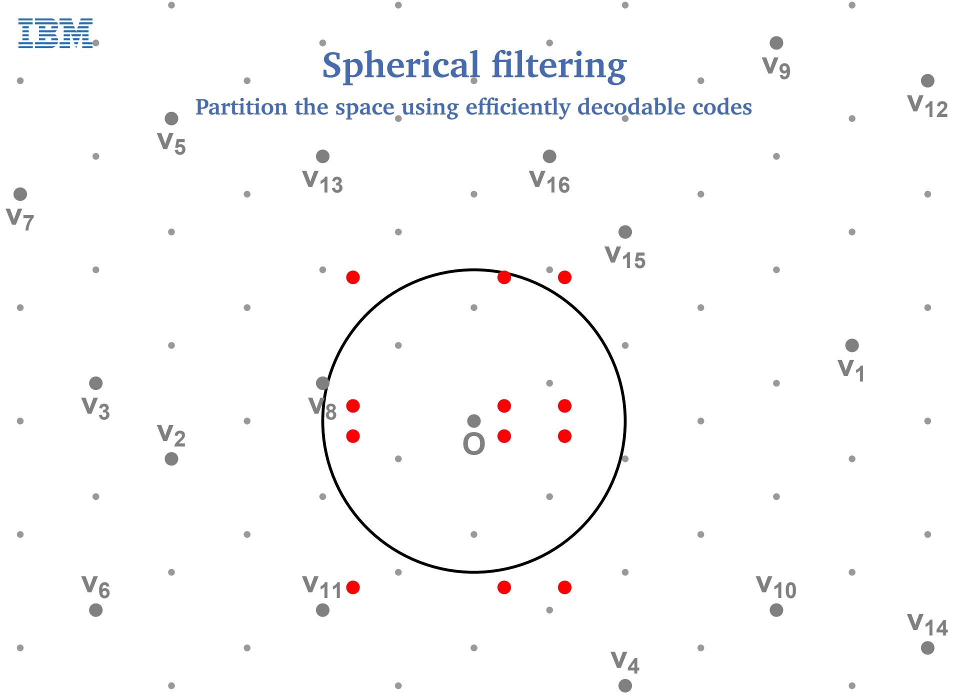
Spherical filtering

Partition the space using efficiently decodable codes



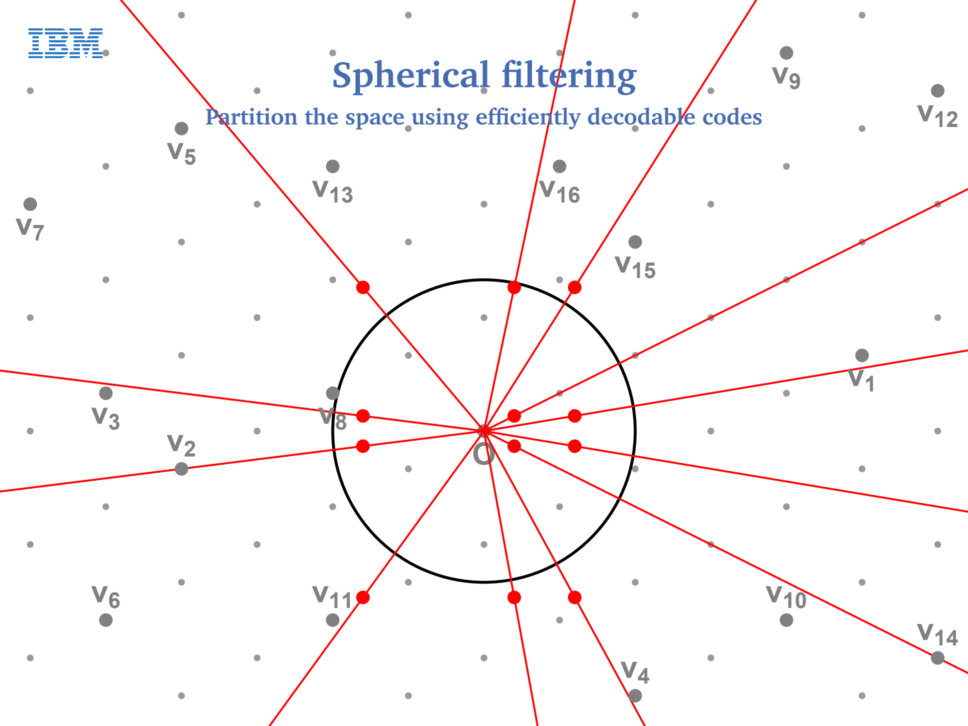
Spherical filtering

Partition the space using efficiently decodable codes



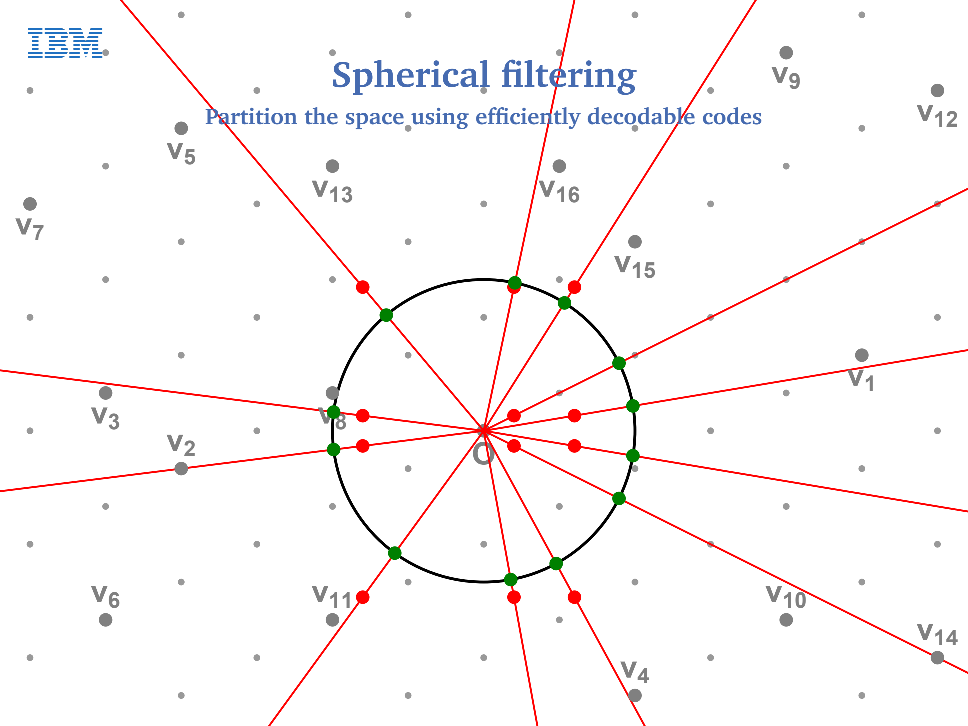
Spherical filtering

Partition the space using efficiently decodable codes



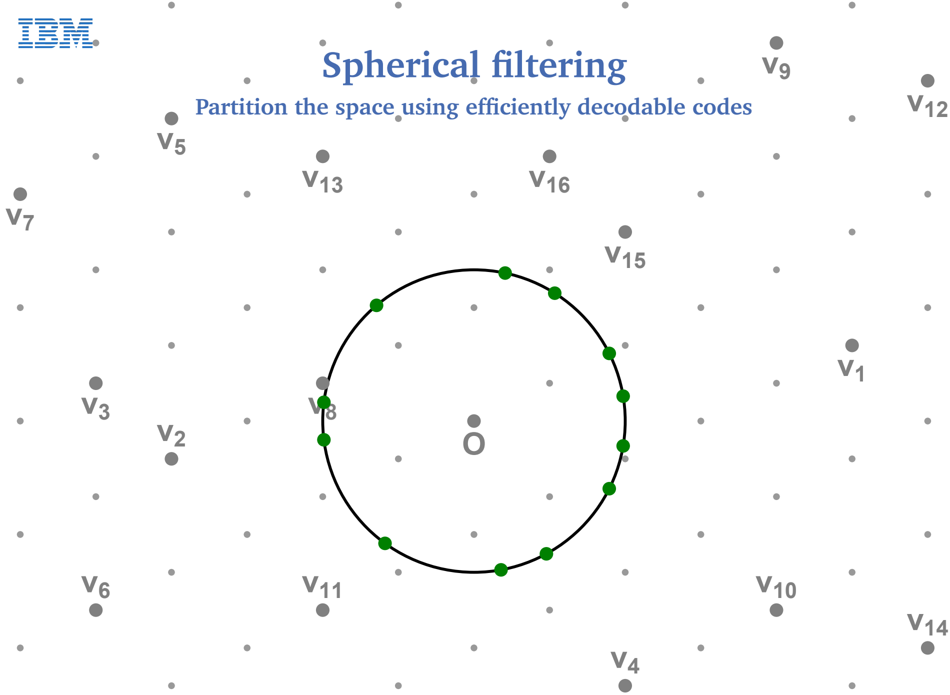
Spherical filtering

Partition the space using efficiently decodable codes



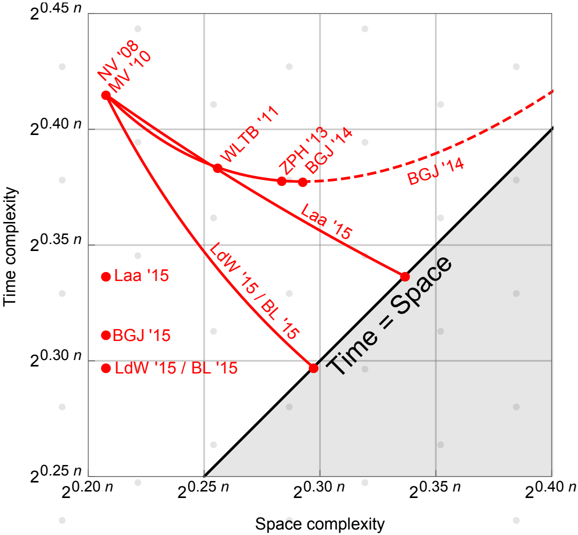
Spherical filtering

Partition the space using efficiently decodable codes



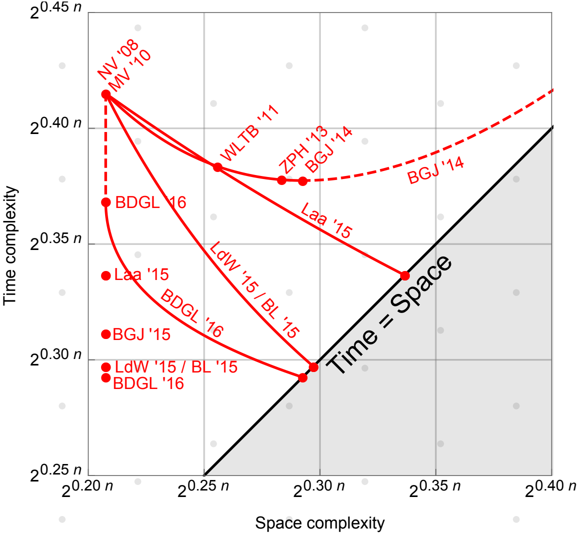
Spherical filtering

Space/time trade-off



Spherical filtering

Space/time trade-off



Outline

Lattices
- Basics
- Cryptography

Enumeration algorithms
- Fincke–Pohst enumeration
- Kannan enumeration
- Pruned enumeration

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

Practical comparison

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*

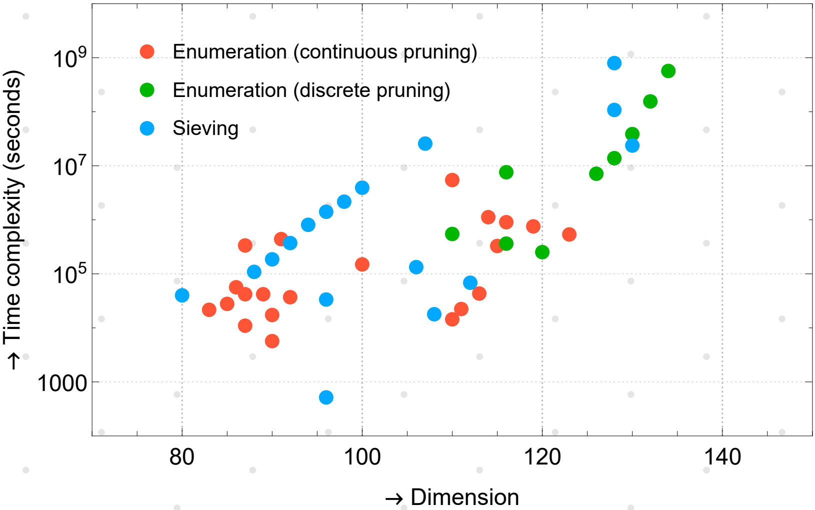
"As far as I know, everyone who has tried sieving as a BKZ subroutine in place of enumeration has concluded that sieving is much too slow to be useful—the cutoff is beyond cryptographically relevant sizes."

— Bernstein, *Google groups '16*

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

— Ducas, *Google groups '16*

SVP in practice



Take-home messages

- Lattice-based crypto relies on hardness of finding short bases
- State-of-the-art basis reduction: BKZ with fast SVP subroutine
- Enumeration for SVP:
  - Memory-efficient
  - Best in low dimensions
  - Fast pruning heuristics
- Sieving for SVP:
  - Large memory requirement
  - Fastest in high dimensions
  - Practical near neighbor speedups
- Enumeration still leading, but sieving is catching up!

Questions?