

Алгоритм Просеивания для решения задачи SVP

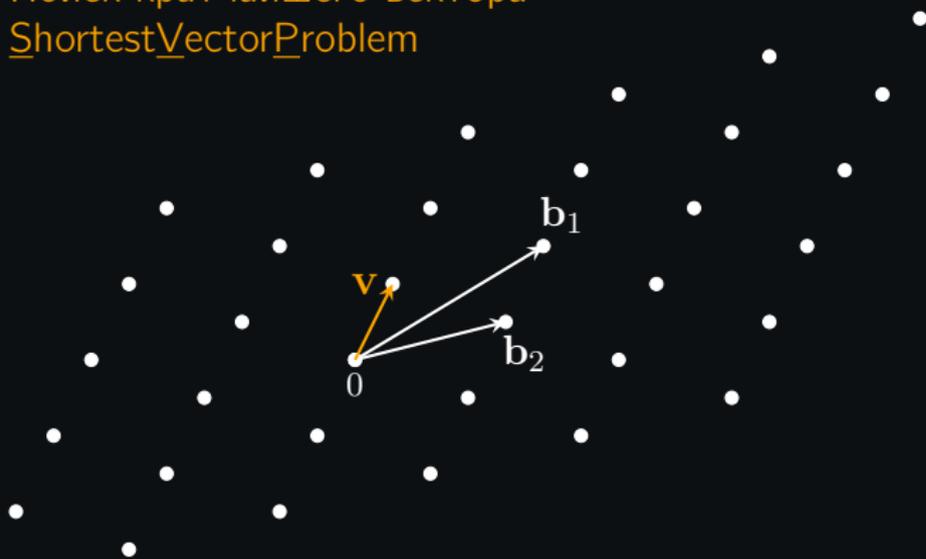
Елена Киршанова

Курс “Криптография на решетках”
16 февраля 2021 г.



Балтийский
федеральный университет
имени Иммануила Канта

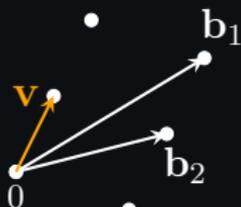
Поиск кратчайшего вектора ShortestVectorProblem



В задаче поиска кратчайшего вектора (SVP) требуется найти $\mathbf{v}_{\text{shortest}} \in \mathcal{L}$:

$$\|\mathbf{v}_{\text{shortest}}\| = \lambda_1(\mathcal{L})$$

Поиск кратчайшего вектора ShortestVectorProblem



В задаче поиска кратчайшего вектора (SVP) требуется найти $\mathbf{v}_{\text{shortest}} \in \mathcal{L}$:

$$\|\mathbf{v}_{\text{shortest}}\| = \lambda_1(\mathcal{L})$$

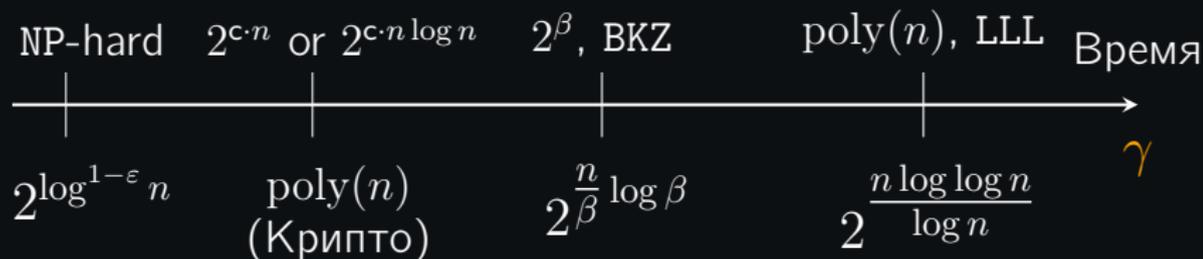
Упрощение: поиск аппроксимации (γ -SVP) к $\mathbf{v}_{\text{shortest}}$:

$$\|\mathbf{v}_{\text{short}}\| \leq \gamma \cdot \lambda_1(\mathcal{L})$$

Асимптотическая сложность SVP ($o()$ опущены)

$$\|\mathbf{v}_{\text{shortest}}\| \leq \sqrt{n} \cdot \det(\mathcal{L})^{1/n}$$

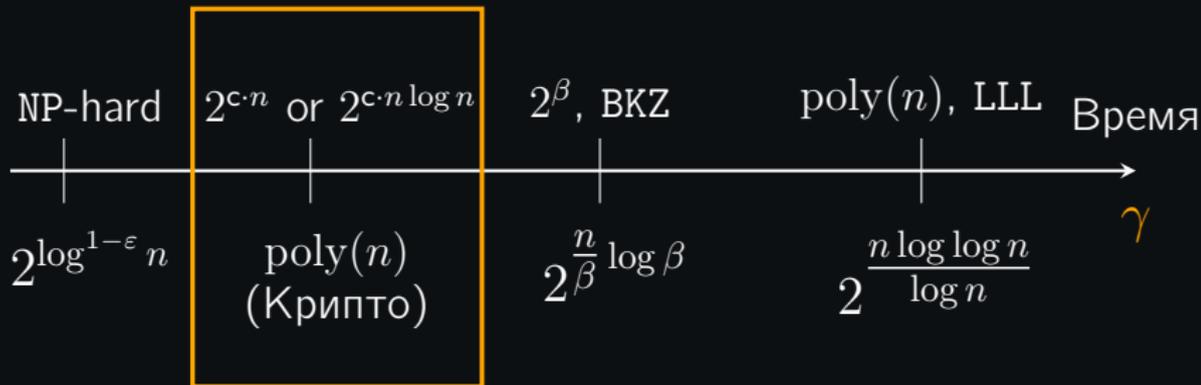
$$\|\mathbf{v}_{\text{short}}\| \leq \gamma \cdot \|\mathbf{v}_{\text{shortest}}\|$$



Асимптотическая сложность SVP ($o()$ опущены)

$$\|\mathbf{v}_{\text{shortest}}\| \leq \sqrt{n} \cdot \det(\mathcal{L})^{1/n}$$

$$\|\mathbf{v}_{\text{short}}\| \leq \gamma \cdot \|\mathbf{v}_{\text{shortest}}\|$$



- **Просеивание** (эвристический):

$$\text{Время}(\text{exactSVP}) = 2^{0.292n}$$

$$\text{Память} = 2^{0.2075n}$$

- **Перечисление:**

$$\text{Время}(\text{exactSVP}) = 2^{(1/2\epsilon)n \log n}$$

$$\text{Память} = \text{poly}(n)$$

Алгоритмы для SVP

Алгоритм	Время	Память
Детерминированные алгоритмы:		
Перечисление	$n^{(1/2e)n+o(n)}$	$\text{poly}(n)$
Диаграмма Вороного	$2^{2n+o(n)}$	$2^{n+o(n)}$
Вероятностные алгоритмы:		
Гауссова выборка	$2^{n+o(n)}$	$2^{n+o(n)}$
Просеивание		
– С док-вом	$2^{2.465n+o(n)}$	$2^{1.325n+o(n)}$
– Эвристические:		
– 2-просеивание	$2^{0.415n+o(n)}$	$2^{0.208n+o(n)}$
– 2-просеивание+ LSH	$2^{0.292n+o(n)}$	$2^{0.208n+o(n)}$
– 3-просеивание	$2^{0.396n+o(n)}$	$2^{0.1887n+o(n)}$

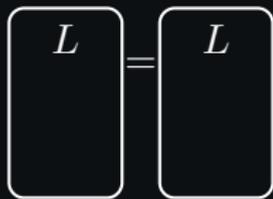
Часть II

Алгоритмы просеивания



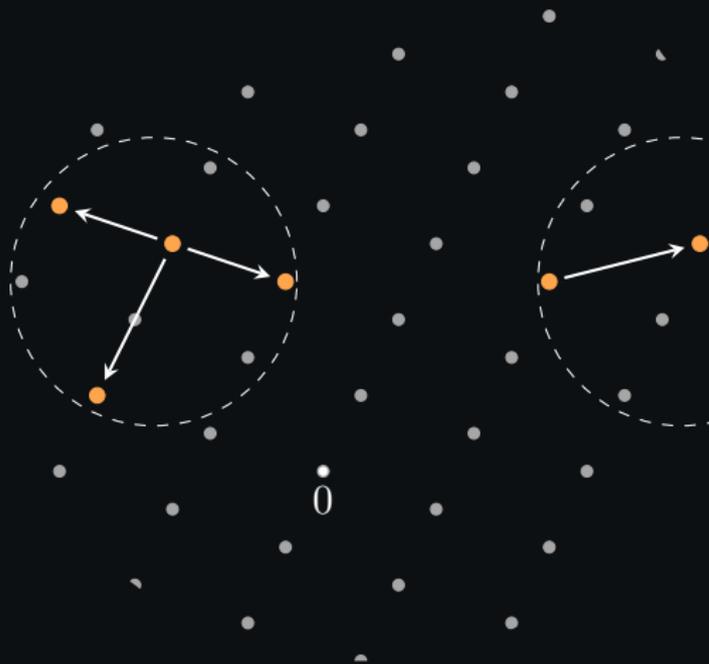
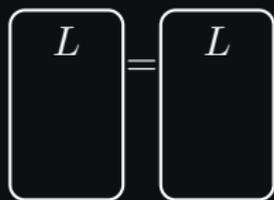
2-Просеивание (Nguyen-Vidick)

Идея: насыщать пространство векторами решётки так, чтобы суммы их пар давали короткие вектора



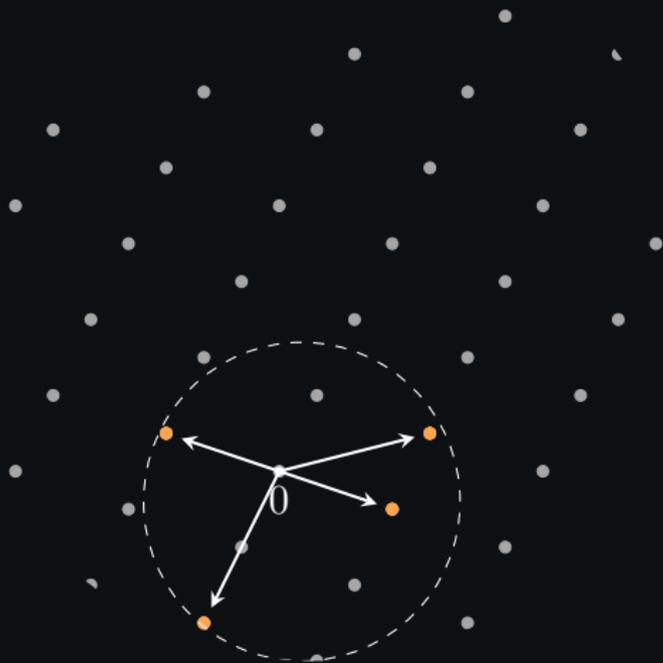
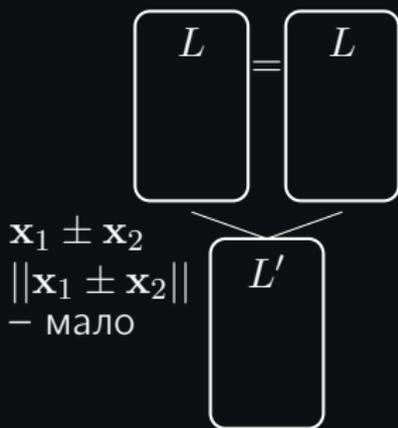
2-Просеивание (Nguyen-Vidick)

Идея: насыщать пространство векторами решётки так, чтобы суммы их пар давали короткие вектора



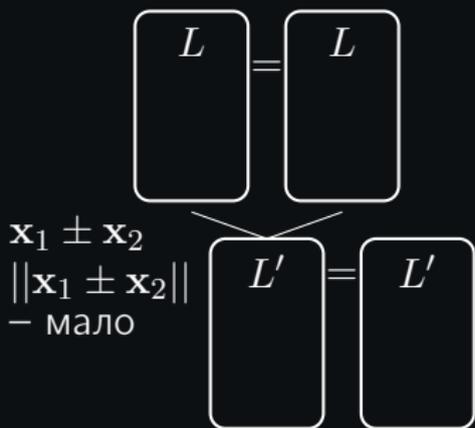
2-Просеивание (Nguyen-Vidick)

Идея: насыщать пространство векторами решётки так, чтобы суммы их пар давали короткие вектора



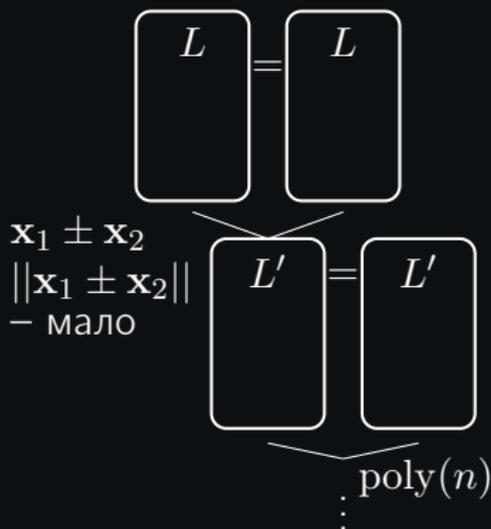
2-Проеивание (Nguyen-Vidick)

Идея: насыщать пространство векторами решётки так, чтобы суммы их пар давали короткие вектора



2-Просеивание (Nguyen-Vidick)

Идея: насыщать пространство векторами решётки так, чтобы суммы их пар давали короткие вектора



Насколько большим должно быть $|L|$?

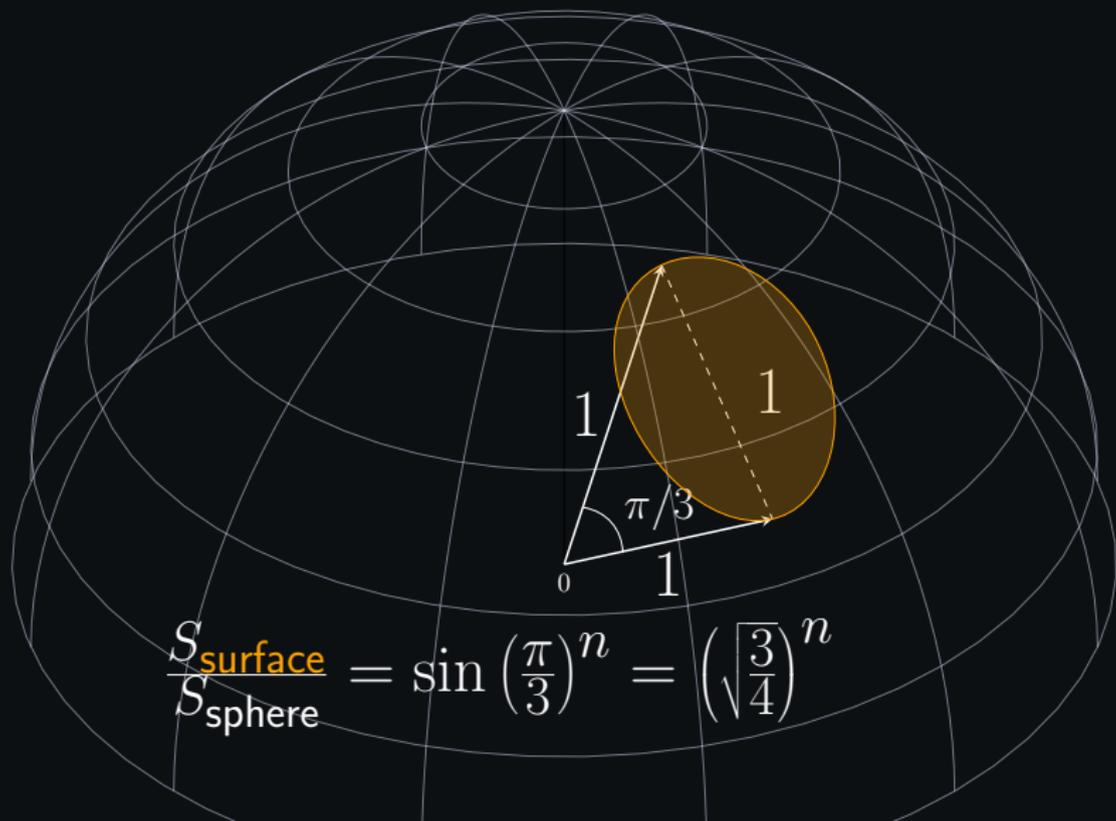
Насколько большим должно быть $|L|$?

Предположение: вектора (нормализованные) в $|L|$ равномерно независимо распределены в S^{n-1} .



Насколько большим должно быть $|L|$?

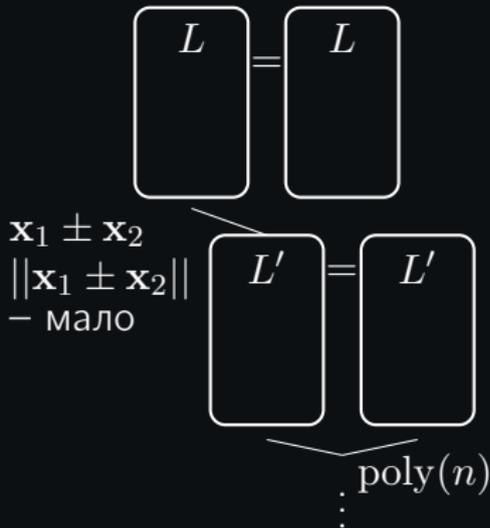
Предположение: вектора (нормализованные) в $|L|$ равномерно независимо распределены в S^{n-1} .



$$\frac{S_{\text{surface}}}{S_{\text{sphere}}} = \sin\left(\frac{\pi}{3}\right)^n = \left(\sqrt{\frac{3}{4}}\right)^n$$

2-Просеивание (Nguyen-Vidick)

Идея: насыщать пространство векторами решётки так, чтобы суммы их пар давали короткие вектора



$$|L| = \left(\sqrt{\frac{3}{4}} \right)^{-n} = 2^{0.2075n}$$

$$T(\text{2-Просеивания}) = |L|^2 = 2^{0.415n}$$

Улучшенное просеивание с хэшированием

Как достичь $T = 2^{0.292n+o(n)}$?

Использовать метод "Ближнего соседа" (Near Neighbour)!
(aka Locality-Sensitive techniques)

Улучшенное просеивание с хэшированием

Как достичь $T = 2^{0.292n+o(n)}$?

Использовать метод "Ближнего соседа" (Near Neighbour)!
(aka Locality-Sensitive techniques)

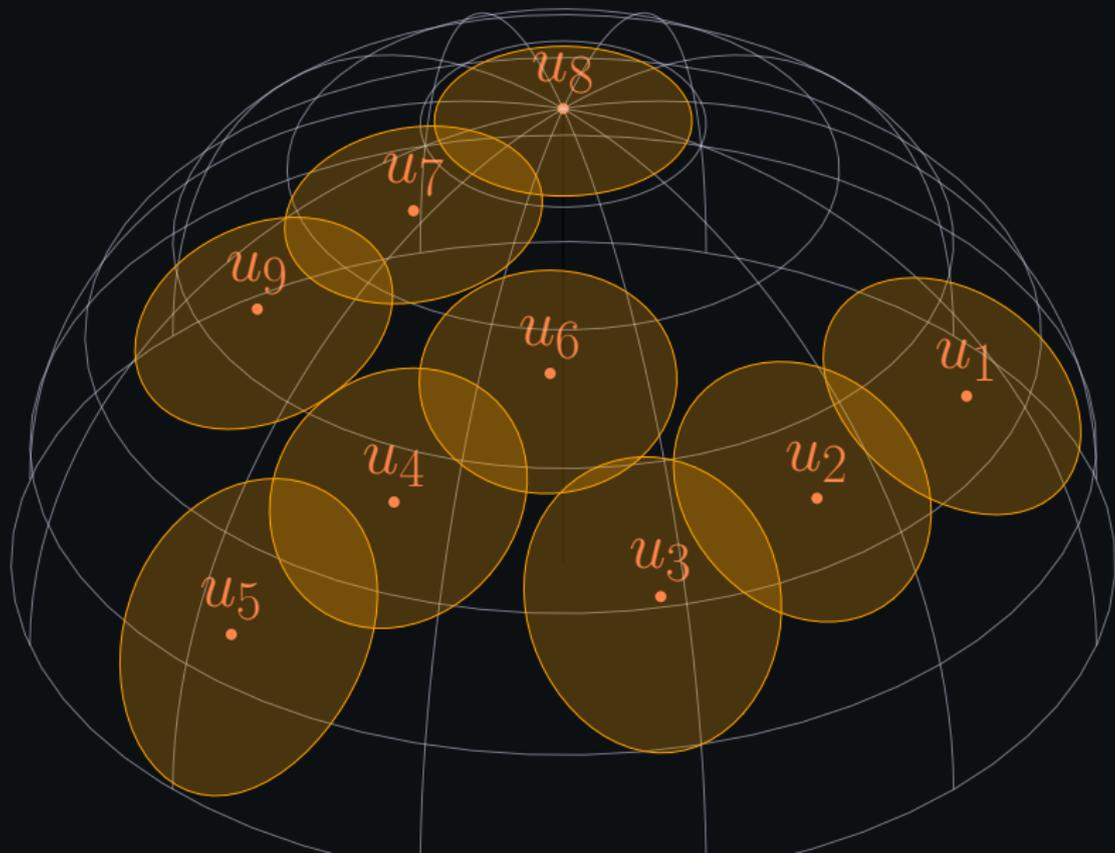
$$\|\mathbf{x}_1 - \mathbf{x}_2\| < \|\mathbf{x}_1\| \iff$$

$$\|\mathbf{x}_1 - \mathbf{x}_2\|^2 < \|\mathbf{x}_1\|^2 \iff$$

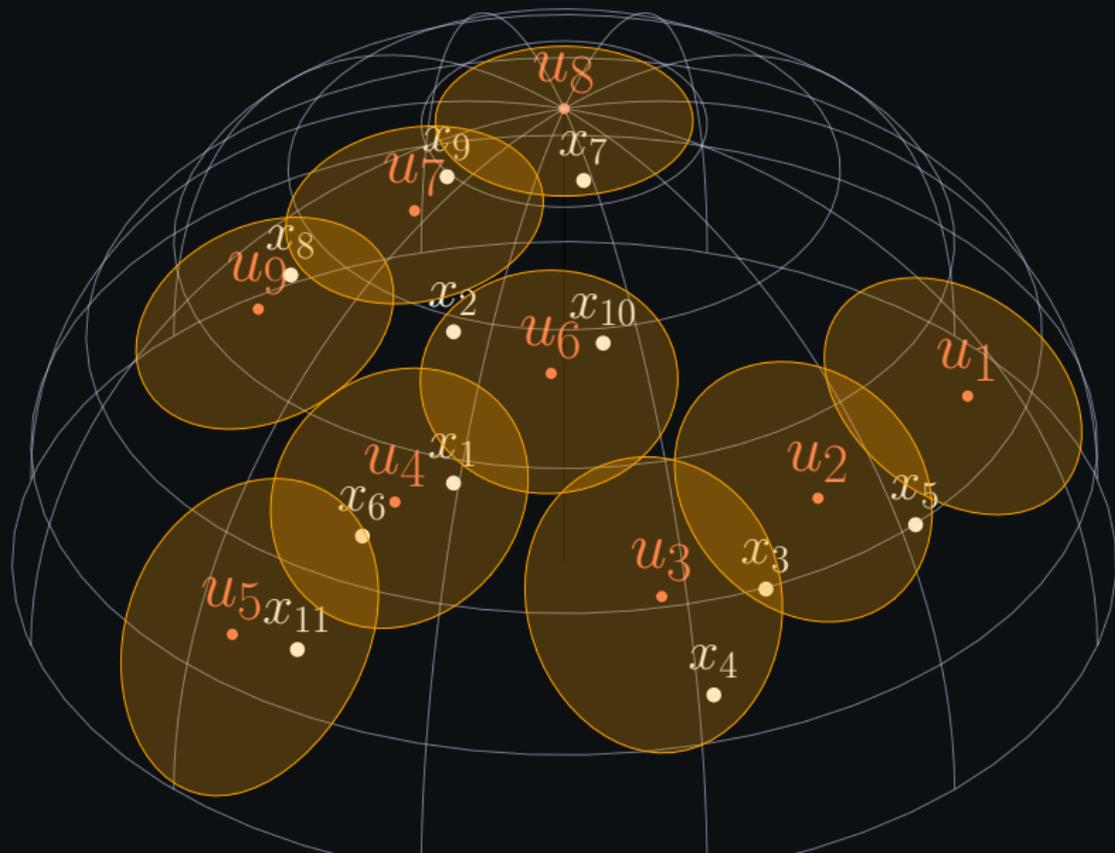
$$\|\mathbf{x}_1\|^2 - 2\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + \|\mathbf{x}_2\|^2 < \|\mathbf{x}_1\|^2 \iff$$

$$\langle \mathbf{x}_1, \mathbf{x}_2 \rangle > \frac{1}{2}\|\mathbf{x}_2\|^2$$

Locality-sensitive filtering (Фильтрация по расстоянию) [BGJ15, BDGL16]



Locality-sensitive filtering (Фильтрация по расстоянию)[BGJ15, BDGL16]



Locality-sensitive filtering (Фильтрация по расстоянию) [BGJ15, BDGL16]

u_i – центры "корзин"

$x_i \in L$



Locality-sensitive filtering (Фильтрация по расстоянию) [BGJ15, BDGL16]

For all u_i :

For all $x \in L$

If $|\langle x, u_i \rangle|$ достаточно велико
добавить x в $\text{Bucket}(u_i)$



Locality-sensitive filtering (Фильтрация по расстоянию) [BGJ15, BDGL16]

For all \mathbf{u}_i :

For all $\mathbf{x} \in L$

If $|\langle \mathbf{x}, \mathbf{u}_i \rangle|$ достаточно велико
добавить \mathbf{x} в $\text{Bucket}(\mathbf{u}_i)$

For all \mathbf{u}_i :

For all $(\mathbf{x}, \mathbf{x}') \in \text{Bucket}(\mathbf{u}_i)$

Проверить если $\|\mathbf{x} \pm \mathbf{x}'\|$ – мало

Для $2^{(0.142+o(1))n}$ \mathbf{u}_i :

$$T = 2^{(0.349+o(1))n}$$

Если \mathbf{u}_i имеют особую форму

$$T = 2^{(0.292+o(1))n}$$

Анализ I

1. Как эффективно найти релевантные центры u_i ?

[MO15, BDGL16]: Взять в качестве u_i кодовые слова из некоторого кода

$$u \in C_1 \times C_2 \times \dots \times C_t =: U,$$

C_i -сферический код длины $o(n)$.

Для получения всех центров, содержащих

$x = [x_1|x_2|\dots|x_t]$, декодируем x_i относительно C_i .

Анализ I

1. Как эффективно найти релевантные центры u_i ?
[MO15, BDGL16]: Взять в качестве u_i кодовые слова из некоторого кода

$$u \in C_1 \times C_2 \times \dots \times C_t =: U,$$

C_i -сферический код длины $o(n)$.

Для получения всех центров, содержащих $x = [x_1|x_2|\dots|x_t]$, декодируем x_i относительно C_i .

2. Для каждого $x \in L$: нахождение всех релевантных $u_i \in U$ (построение хэш-таблиц)

$$T_{\text{Update}} = |U| \cdot \Pr_{u_i \in \mathcal{S}^{n-1}} [\langle u_i, x \rangle < \alpha] = |U| \cdot (1 - \alpha^2)^{n/2}.$$

Анализ I

1. Как эффективно найти релевантные центры u_i ?
[MO15, BDGL16]: Взять в качестве u_i кодовые слова из некоторого кода

$$u \in C_1 \times C_2 \times \dots \times C_t =: U,$$

C_i -сферический код длины $o(n)$.

Для получения всех центров, содержащих $x = [x_1|x_2|\dots|x_t]$, декодируем x_i относительно C_i .

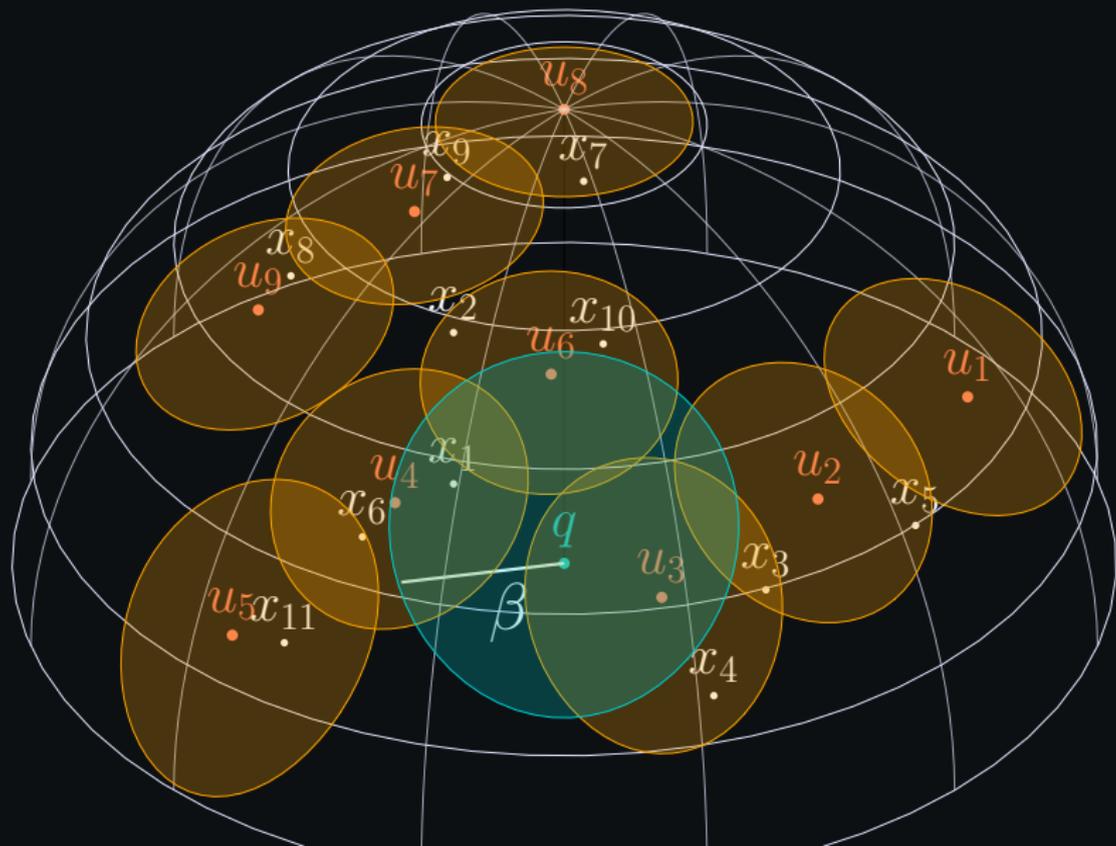
2. Для каждого $x \in L$: нахождение всех релевантных $u_i \in U$ (построение хэш-таблиц)

$$T_{\text{Update}} = |U| \cdot \Pr_{u_i \in \mathcal{S}^{n-1}} [\langle u_i, x \rangle < \alpha] = |U| \cdot (1 - \alpha^2)^{n/2}.$$

3. Запрос для каждого $x \in L$:

$$T_{\text{Query}} = |U| \cdot \Pr_{u_i \in \mathcal{S}^{n-1}} [\langle u_i, x \rangle < \beta] = |U| \cdot (1 - \beta^2)^{n/2}.$$

$|U|?$



$|U|?$

$|U|$ определено $P = 1/|U|$ при условии
 $\langle x_3, x_4 \rangle = 1/2$

- $\langle u, x_3 \rangle = \alpha$

- $\langle u, x_4 \rangle = \beta$

$$P = \Pr_{u \in \mathcal{S}^{n-1}} [\langle u, x_3 \rangle = \alpha, \langle u, x_4 \rangle = \beta \mid \langle x_3, x_4 \rangle = 1/2]$$

$$= \frac{\left(\det \begin{pmatrix} 1 & \alpha & \beta \\ \alpha & 1 & c \\ \beta & c & 1 \end{pmatrix} \right)^{n/2}}{\left(\det \begin{pmatrix} 1 & c \\ c & 1 \end{pmatrix} \right)^{n/2}}.$$

Для $\alpha = \beta = c = 1/2$, $P^{-1} = \left(\frac{3}{2}\right)^{n/2} = 2^{0.292n}$

$$|U|?$$

$|U|$ определено $P = 1/|U|$ при условии
 $\langle x_3, x_4 \rangle = 1/2$

$$1. \langle u, x_3 \rangle = \alpha$$

$$2. \langle u, x_4 \rangle = \beta$$

$$P = \Pr_{u \in \mathcal{S}^{n-1}} [\langle u, x_3 \rangle = \alpha, \langle u, x_4 \rangle = \beta \mid \langle x_3, x_4 \rangle = 1/2]$$

$$= \frac{\left(\det \begin{pmatrix} 1 & \alpha & \beta \\ \alpha & 1 & c \\ \beta & c & 1 \end{pmatrix} \right)^{n/2}}{\left(\det \begin{pmatrix} 1 & c \\ c & 1 \end{pmatrix} \right)^{n/2}}.$$

$|U|?$

$|U|$ определено $P = 1/|U|$ при условии
 $\langle x_3, x_4 \rangle = 1/2$

- $\langle u, x_3 \rangle = \alpha$

- $\langle u, x_4 \rangle = \beta$

$$P = \Pr_{u \in \mathcal{S}^{n-1}} [\langle u, x_3 \rangle = \alpha, \langle u, x_4 \rangle = \beta \mid \langle x_3, x_4 \rangle = 1/2]$$

$$= \frac{\left(\det \begin{pmatrix} 1 & \alpha & \beta \\ \alpha & 1 & c \\ \beta & c & 1 \end{pmatrix} \right)^{n/2}}{\left(\det \begin{pmatrix} 1 & c \\ c & 1 \end{pmatrix} \right)^{n/2}}.$$

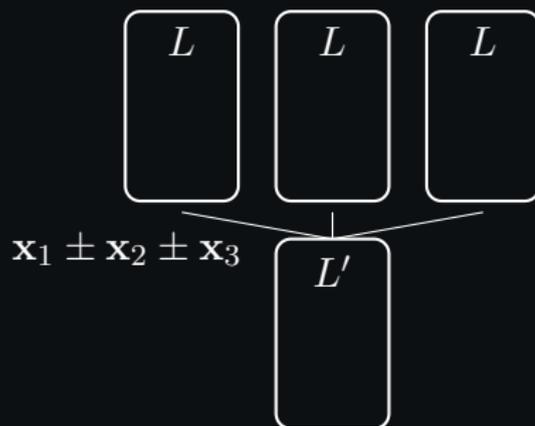
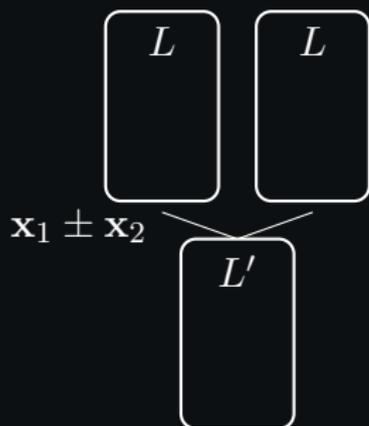
Для $\alpha = \beta = c = 1/2$, $P^{-1} = \left(\frac{3}{2}\right)^{n/2} = 2^{0.292n}$

3-Просеивание [BLS16, HK17, HKL18]: уменьшение затрат памяти

Идея: насыщать пространство, пока **тройки** векторов не станут давать короткие суммы

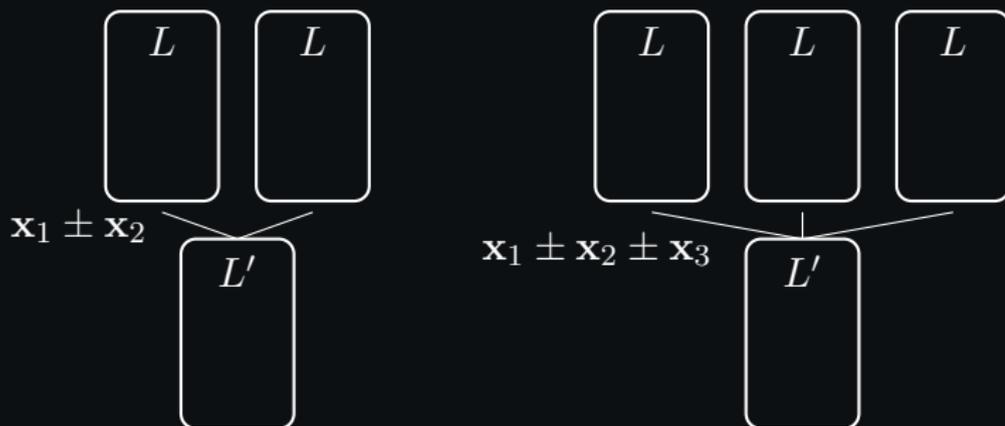
3-Просеивание [BLS16, НК17, НКЛ18]: уменьшение затрат памяти

Идея: насыщать пространство, пока **тройки** векторов не станут давать короткие суммы



3-Просеивание [BLS16, НК17, НКЛ18]: уменьшение затрат памяти

Идея: насыщать пространство, пока **тройки** векторов не станут давать короткие суммы



- Размер списка

$$|L| = |L|^3 \cdot \Pr[\|\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3\| \text{ small}]$$

- Для 3-Просеивания имеем

$$|L| = 2^{0.179n}, \quad \text{cf. } 2^{0.208n} \text{ for 2-Sieve}$$

$$T = 2^{0.359n}, \quad \text{cf. } 2^{0.292n} \text{ for 2-Sieve}$$

Улучшенное 3-просеивание с хэшированием

For all \mathbf{u}_i :

For all $\mathbf{x} \in L$

If $|\langle \mathbf{x}, \mathbf{u}_i \rangle|$ – велико

Добавить \mathbf{x} в $\text{Bucket}(\mathbf{u}_i)$

For all \mathbf{u}_i :

For all $(\mathbf{x}, \mathbf{x}') \in \text{Bucket}(\mathbf{u}_i)$

Проверить $\|\mathbf{x} \pm \mathbf{x}'\|$ – мало

Проверить $\|\mathbf{x} \pm \mathbf{x}' \pm \mathbf{u}\|$ – мало

для "хорошей" комбинации ± 1



Часть III

Решение челленджей SVP алгоритмами просеивания¹



¹или Как получить автомат

Реальные SVP задачи

- TU Darmstadt предлагает решить задачу 1.05-SVP ²
- Для “случайных” решёток вида:

$$B = \begin{pmatrix} p & x_1 & \dots & x_{n-1} \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

- Рекорд (до 2019) принадлежал Teruya et al.: решение 1.05-SVP в решётке ранга 150-dim алгоритмом **перечисления** за 2^{22} core-часов

²<https://www.latticechallenge.org/svp-challenge/halloffame.php>

Решения 1.05-SVP с помощью алгоритма G6K³

SVP dim	γ	Sieve Wall time	Total CPU time	Memory usage
155	1.00803	14d 16h	1056d	246 GiB
153	1.02102	11d 15h	911d	139 GiB
151	1.04411	11d 19h	457.5d	160 GiB
149	0.98506	60h 7m	4.66kh	59 GiB
147	1.03863	123h 29m	4.79kh	67.0 GiB
145	1.04267	39h 3m	1496h	37.7 GiB

³<https://github.com/fplll/g6k>