Speed-ups and time–memory trade-offs for tuple lattice sieving

Gottfried Herold, Elena Kirshanova, Thijs Laarhoven

ENS de Lyon, Eindhoven University of Technology

PKC 2018

Our results

- Improved time-memory trade-offs for k-tuple sieve
- Asymptotically faster k-tuple sieve with Near Neighbour search

Our results

- Improved time-memory trade-offs for k-tuple sieve
- Asymptotically faster k-tuple sieve with Near Neighbour search

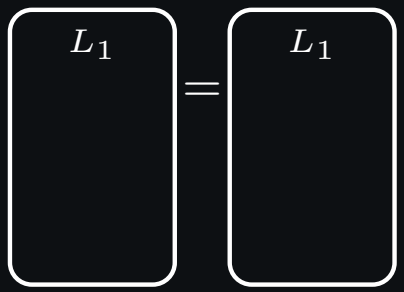
Shortest Vector Problem

Given a lattice $\mathcal{L} \in \mathbb{R}^n$, find $v \neq 0 \in \mathcal{L}$ s.t. $||v||_2$ is small.

Asymptotically best known (heuristic) algorithms for SVP are *sieving* algorithms.
They run in time $2^{c_1 n + o(n)}$ using $2^{c_2 n + o(n)}$ space.
Goal: improve the constants c_1, c_2, trade c_i for c_j.

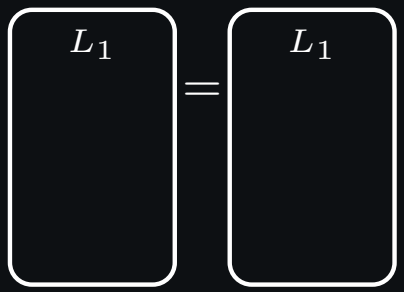
Sieving [AKS01,NV08,MV10]

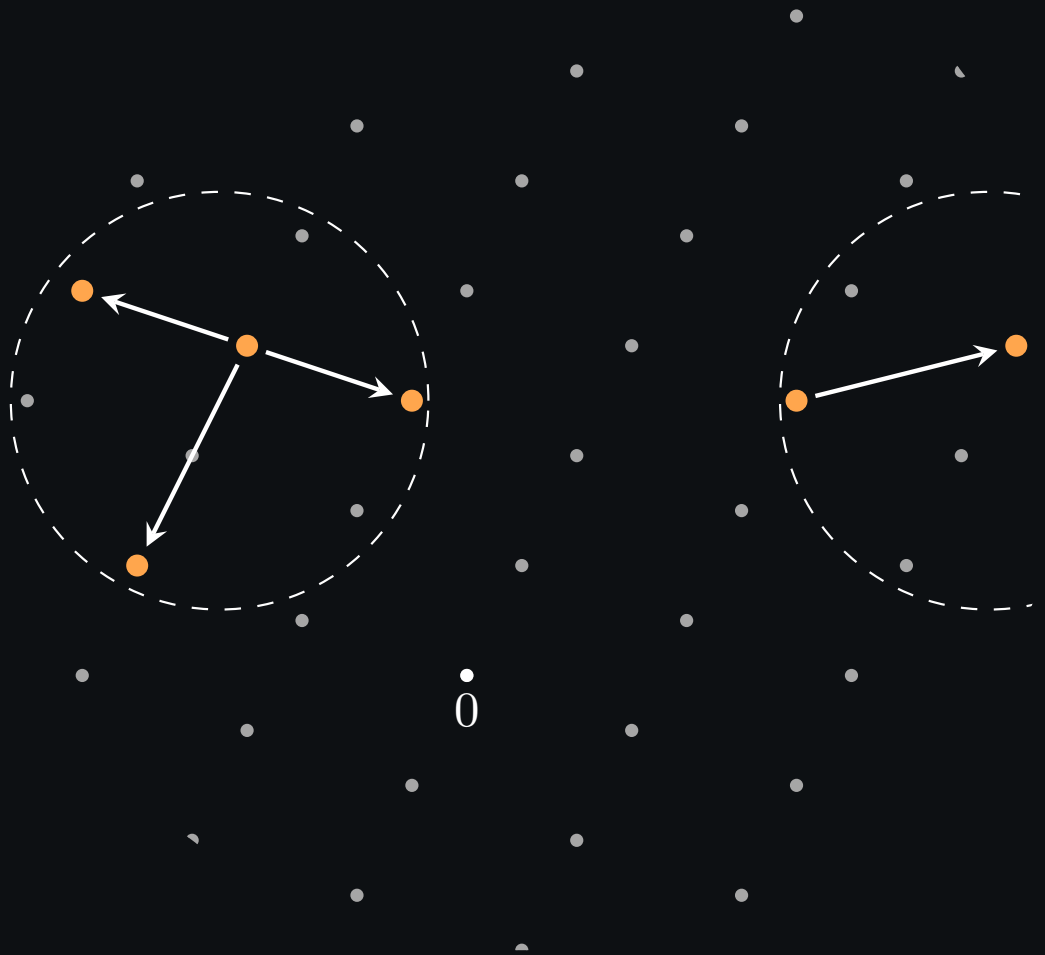
Basic idea: saturate space with lattice points until they give short pairs



Sieving [AKS01,NV08,MV10]

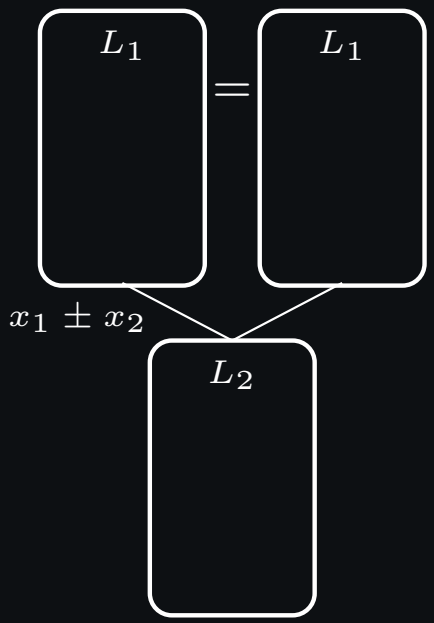
Basic idea: saturate space with lattice points until they give short pairs

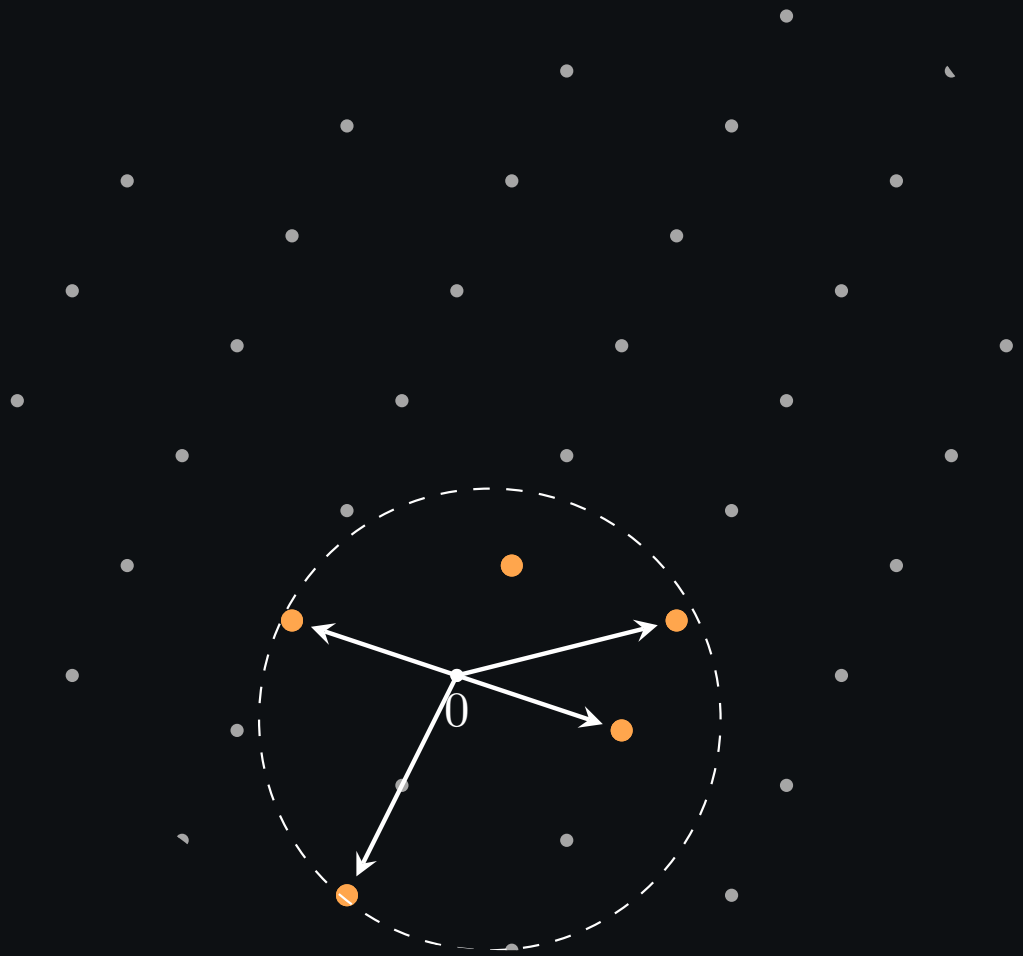




Sieving [AKS01,NV08,MV10]

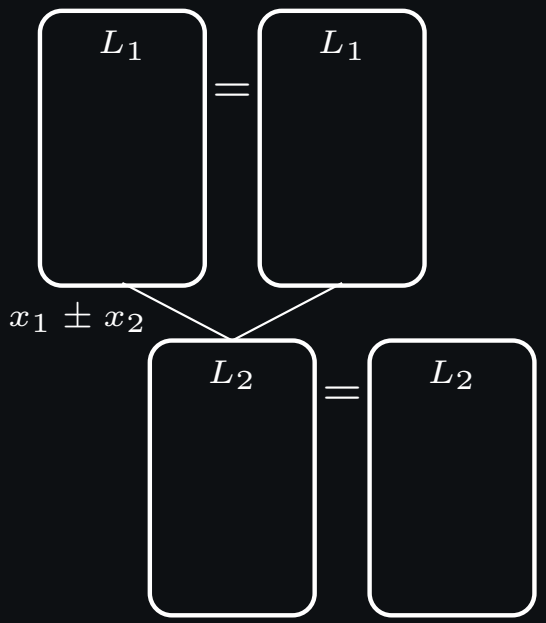
Basic idea: saturate space with lattice points until they give short pairs

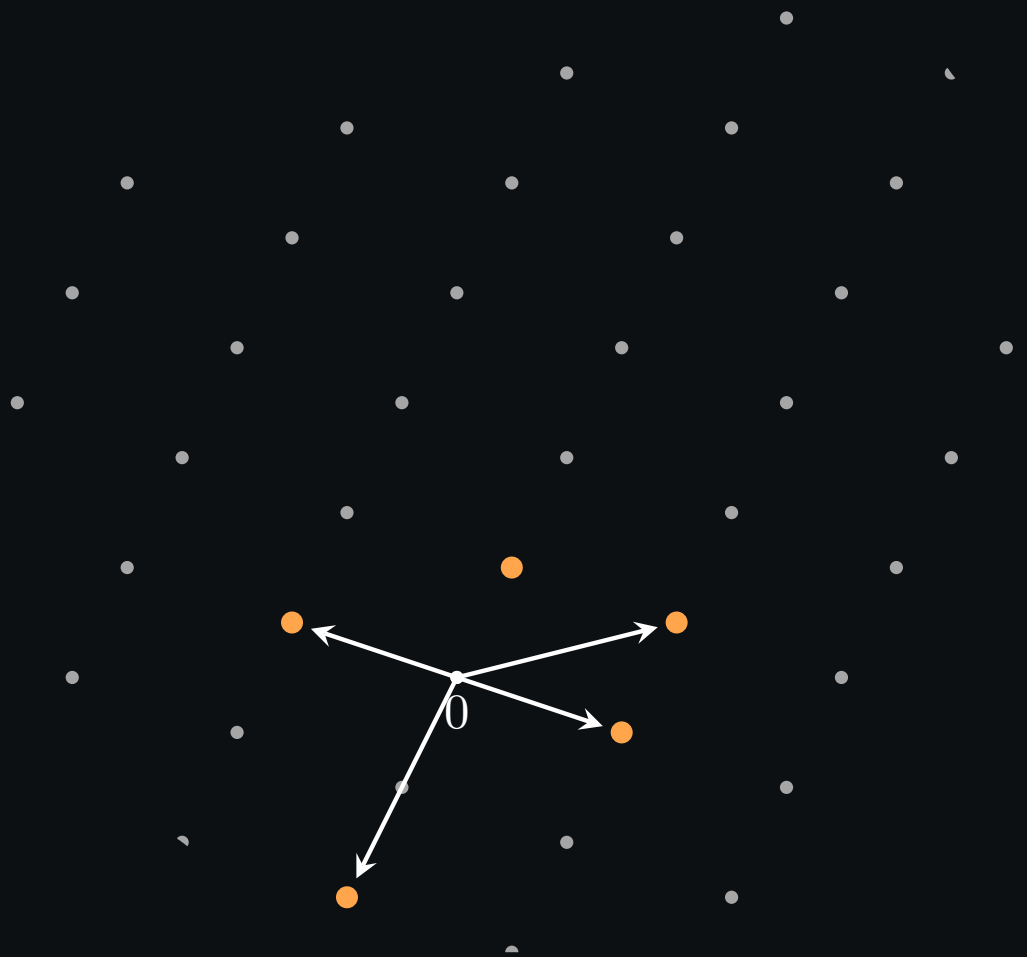




Sieving [AKS01,NV08,MV10]

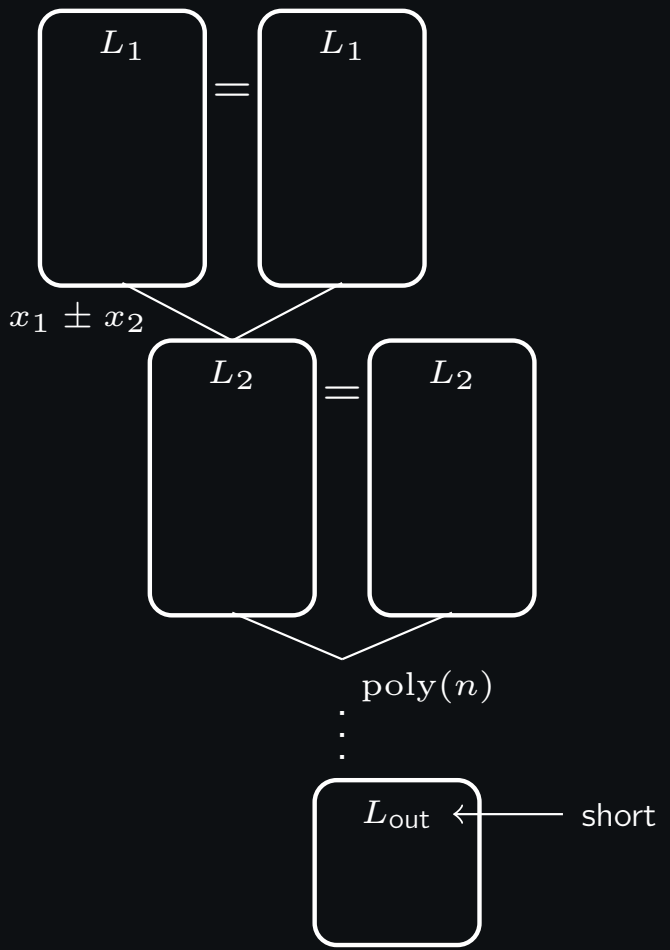
Basic idea: saturate space with lattice points until they give short pairs

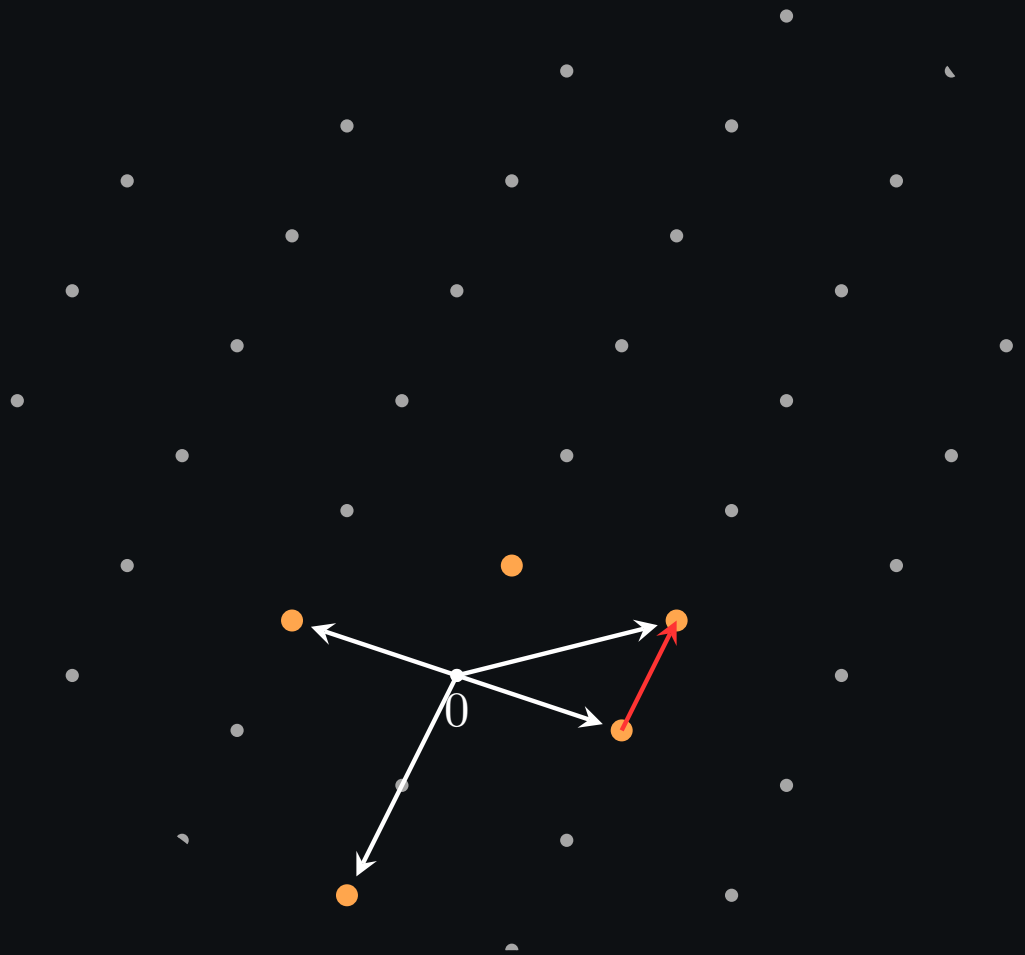




Sieving [AKS01,NV08,MV10]

Basic idea: saturate space with lattice points until they give short pairs





k-Sieve [BLS16, HK17]

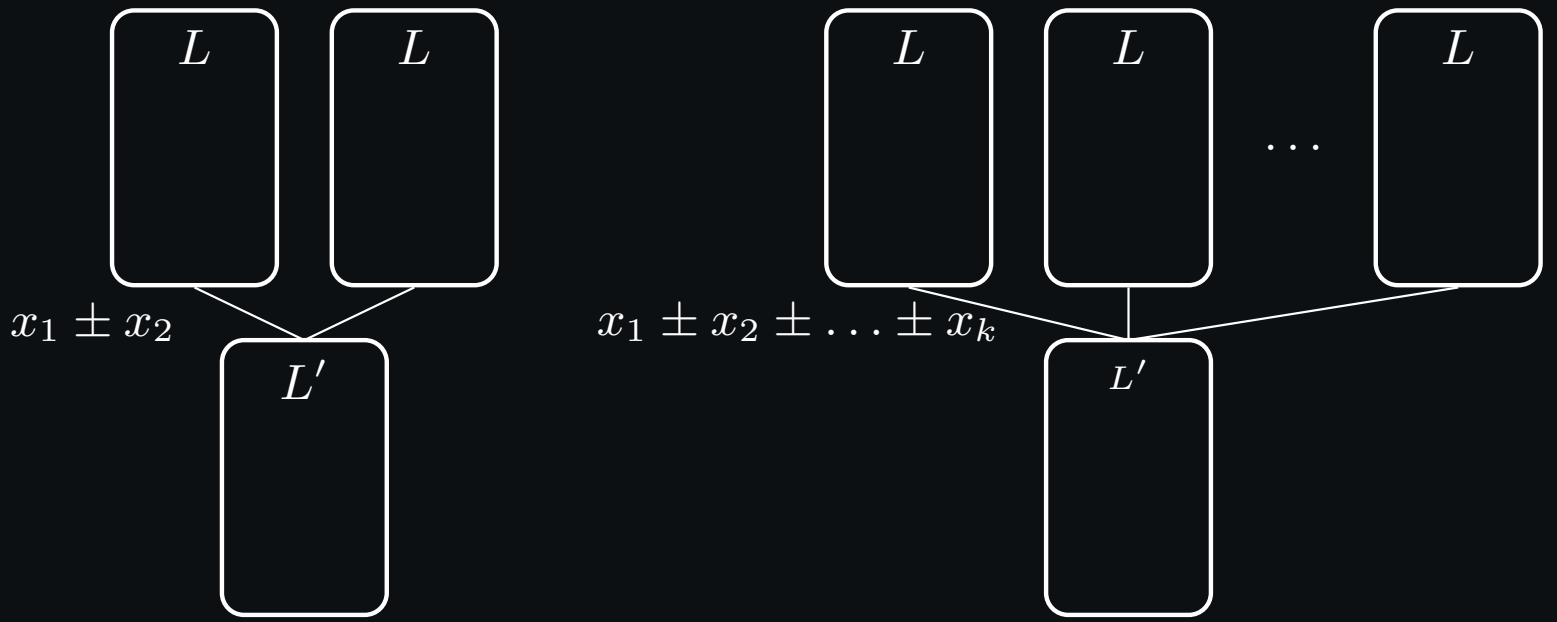
k-tuples

Basic idea: saturate space with lattice points until they give short ~~pairs~~

k-Sieve [BLS16, HK17]

k-tuples

Basic idea: saturate space with lattice points until they give short ~~pairs~~



- List-size determined by

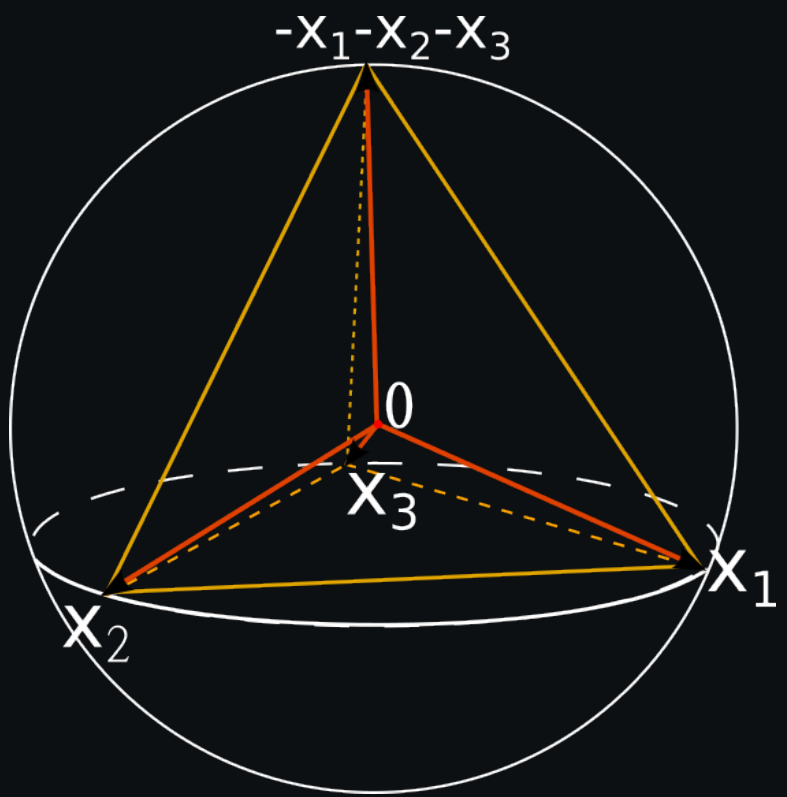
$$|L| = |L|^k \Pr[\ ||x_1 + \ldots + x_k|| \text{ short }]$$

- List-size decreases with k
- Runtime increases with k (except from $k = 2$ to $k = 3$)

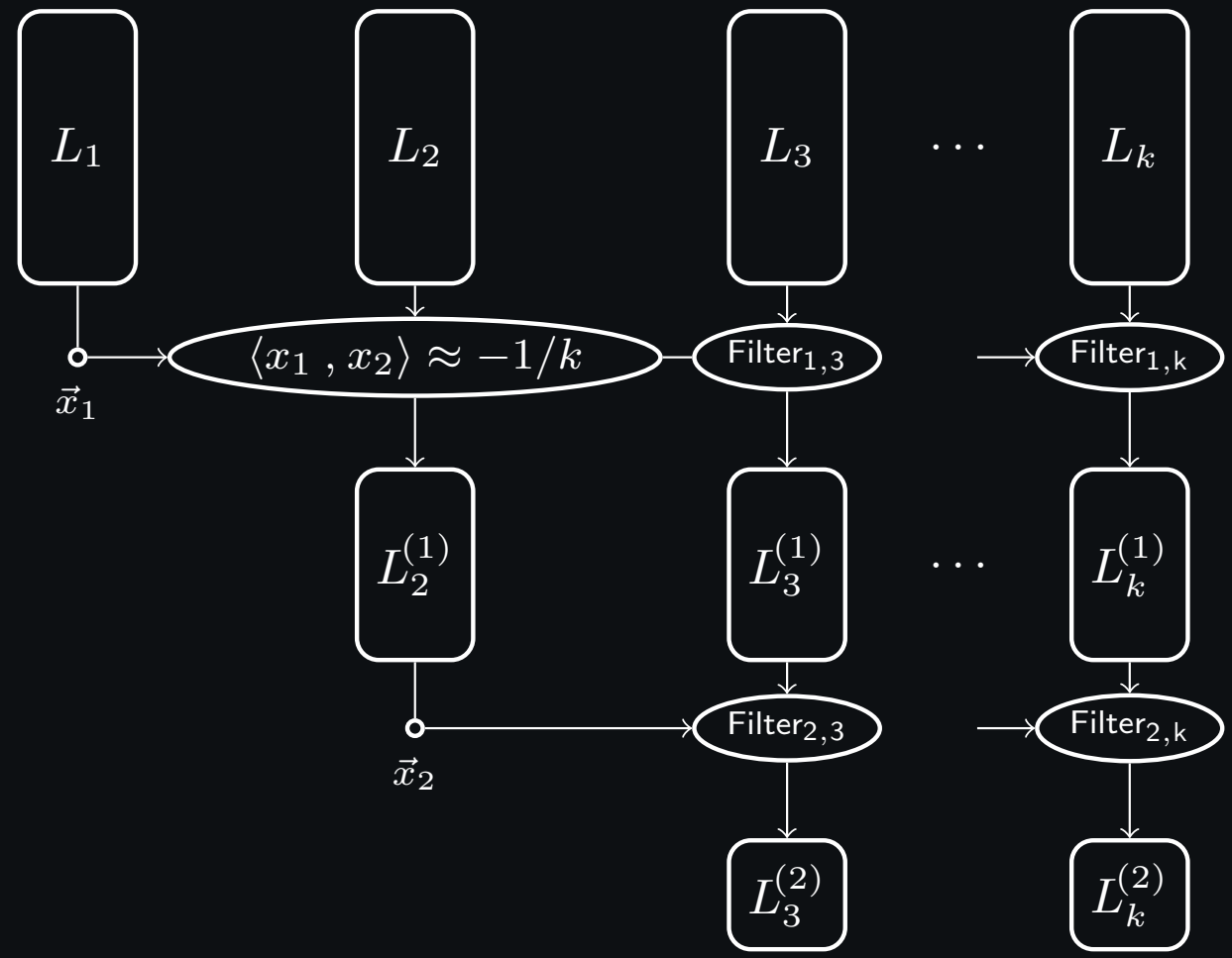
Analysis [HK17]

Most of the tuples $x_1, \ldots, x_k$ s.t. $||x_1 + \ldots + x_k|| \leq 1$ are concentrated around one specific configuration: their Gram matrix is

$$C = (\langle x_i \, , x_j \rangle)_{1 \leq i,j \leq k} = \begin{pmatrix} 1 & -\frac{1}{k} & \ldots & -\frac{1}{k} \\ -\frac{1}{k} & 1 & \ldots & -\frac{1}{k} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{1}{k} & -\frac{1}{k} & \ldots & 1 \end{pmatrix}$$



Algorithm v.1



We have explicit formulas for runtime T and memory M for fixed k.

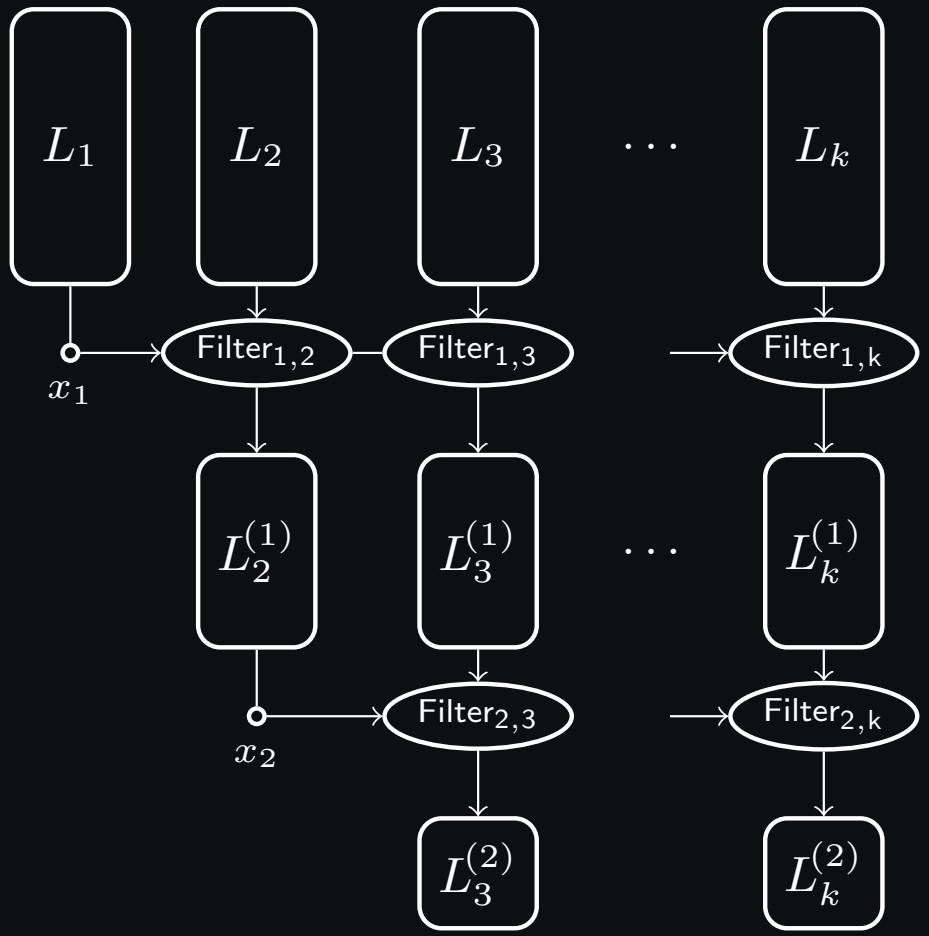
New idea

- The previous algorithm is optimized for memory
- If we increase the size of initial lists $|L_i|$, we only need to find an exponential fraction of solutions
- We know that a random k-tuple satisfies a given Gram-matrix C with probability $\mathcal{O}((\det C)^{n/2})$.

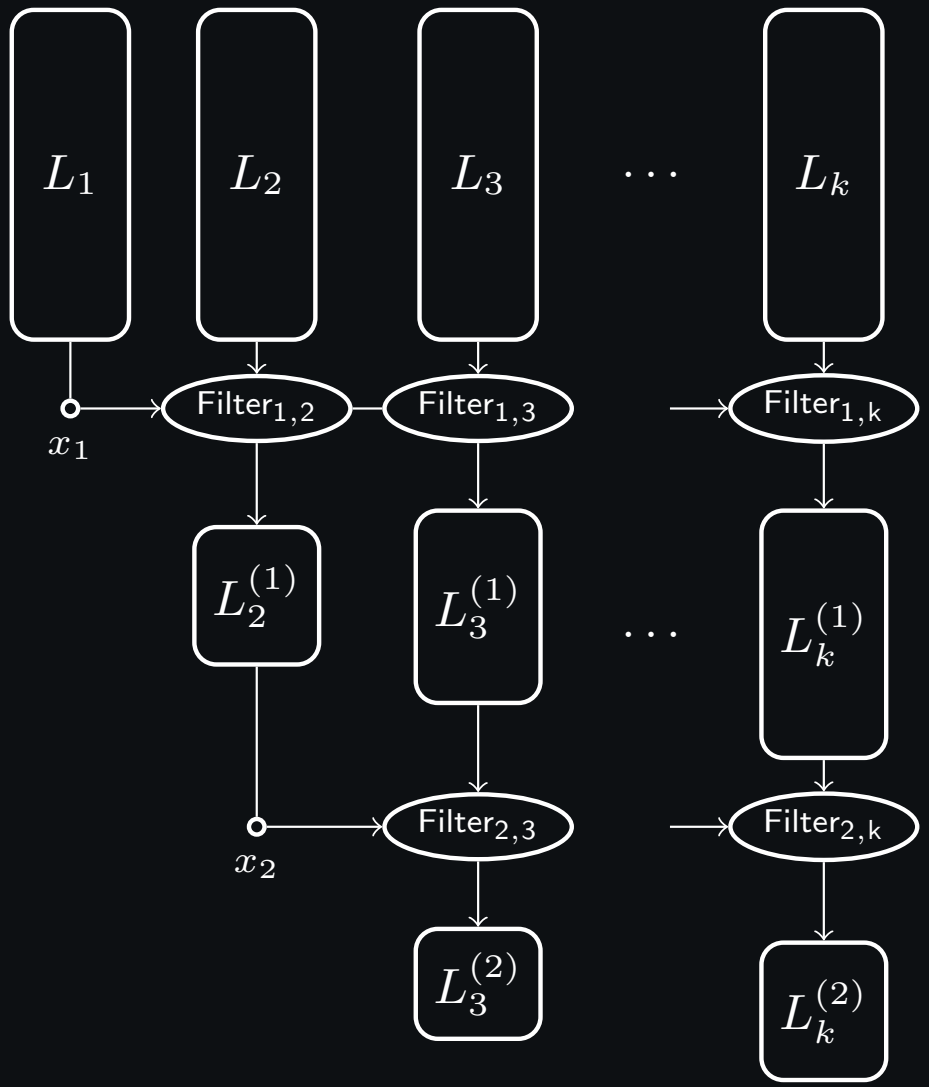
New idea

- The previous algorithm is optimized for memory
- If we increase the size of initial lists $|L_i|$, we only need to find an exponential fraction of solutions
- We know that a random k-tuple satisfies a given Gram-matrix C with probability $\mathcal{O}((\det C)^{n/2})$.
- Certain configurations turn out to be easier to find
- The problem of finding a configuration that satisfies a given bound on T and M is a an optimization problem.

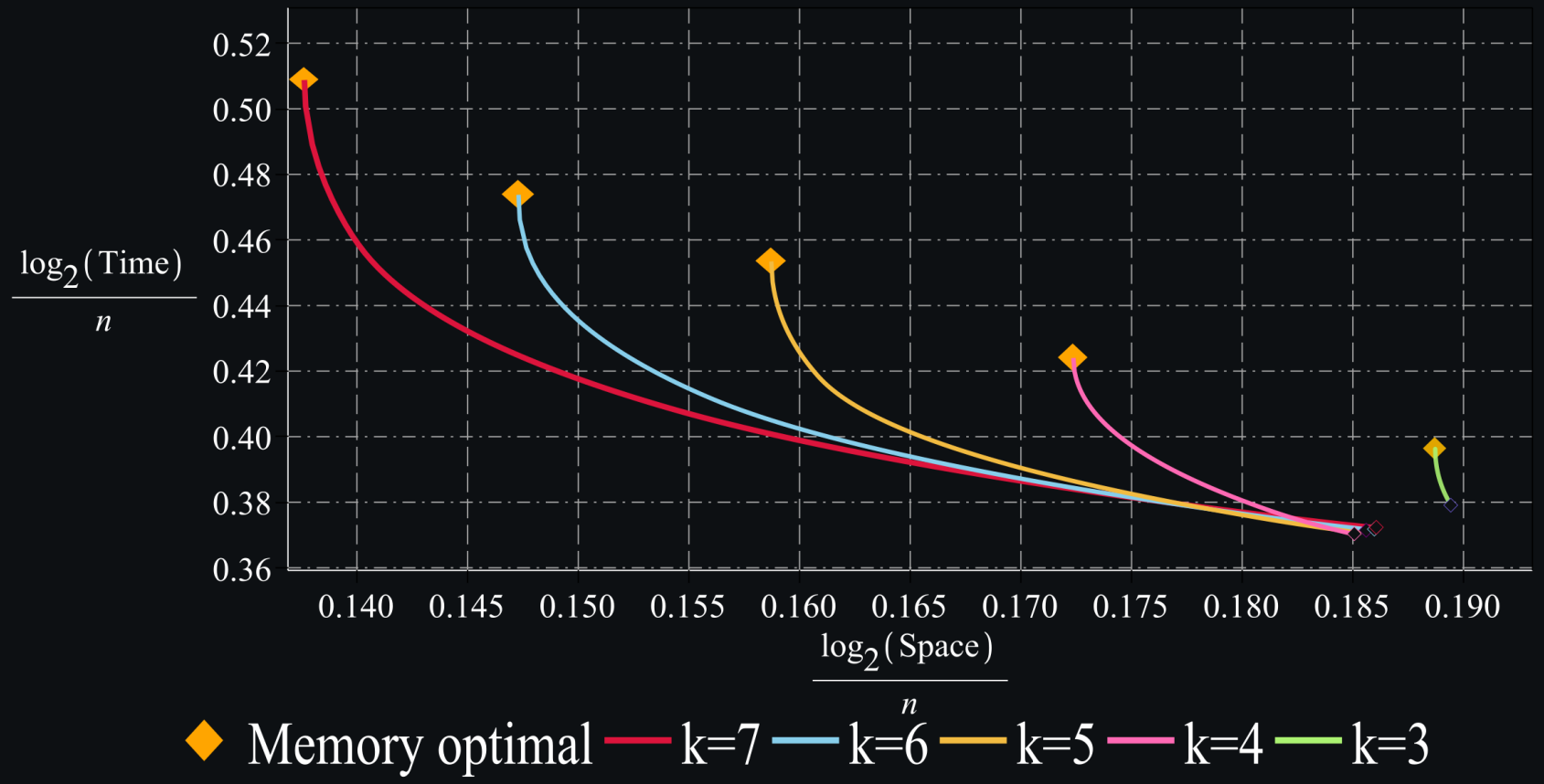
Algorithm v.1



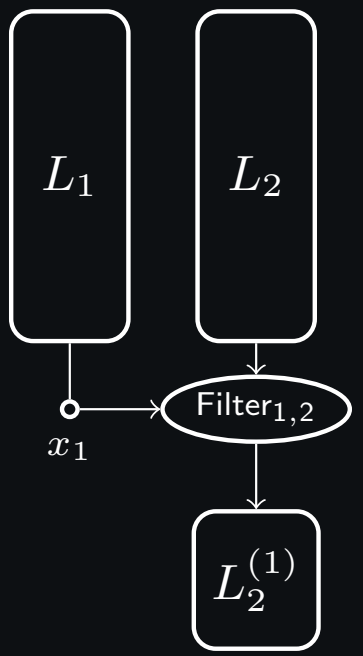
Algorithm v.2: the target configuration C is unbalanced



Time-memory trade-off



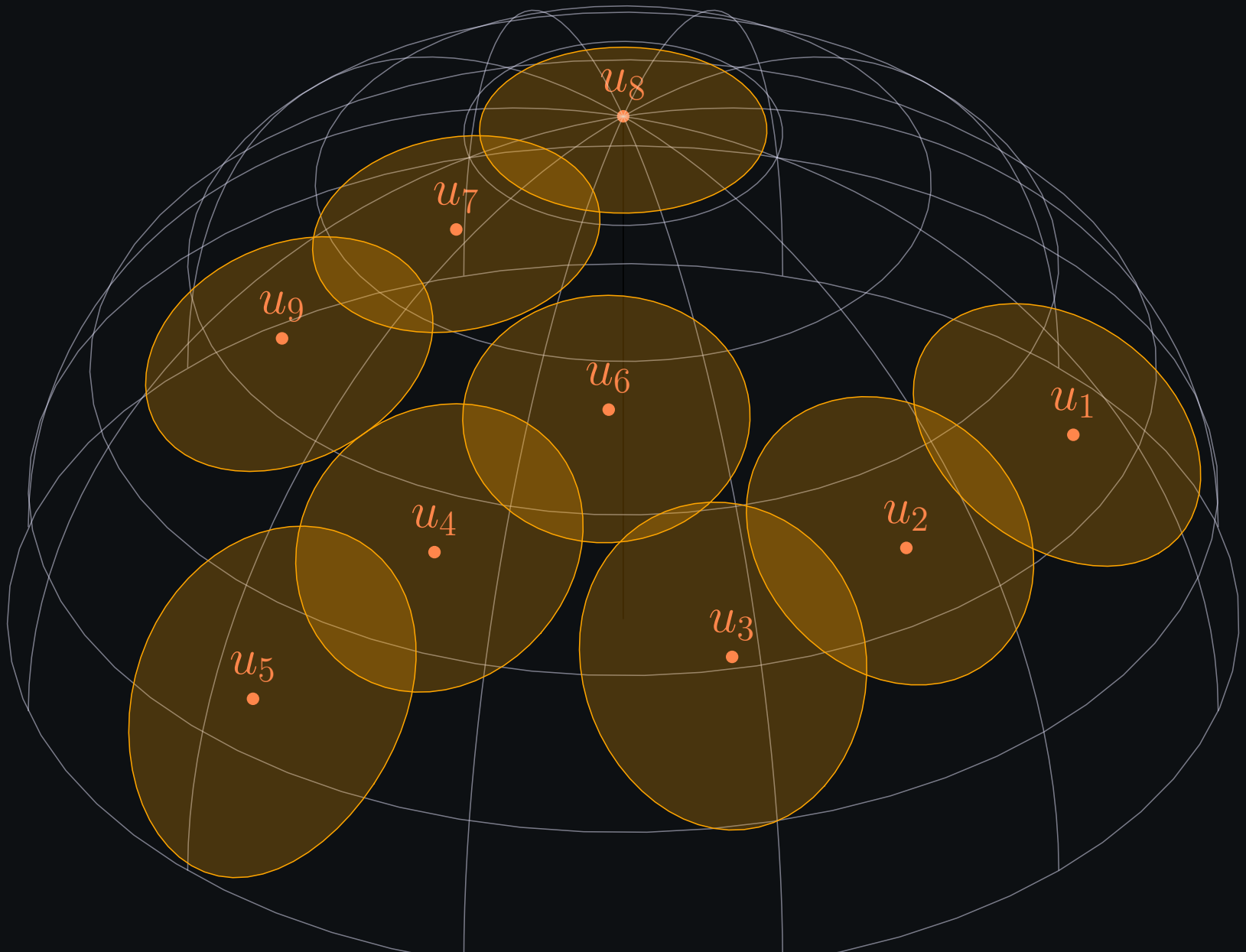
Near Neighbour search for sieving



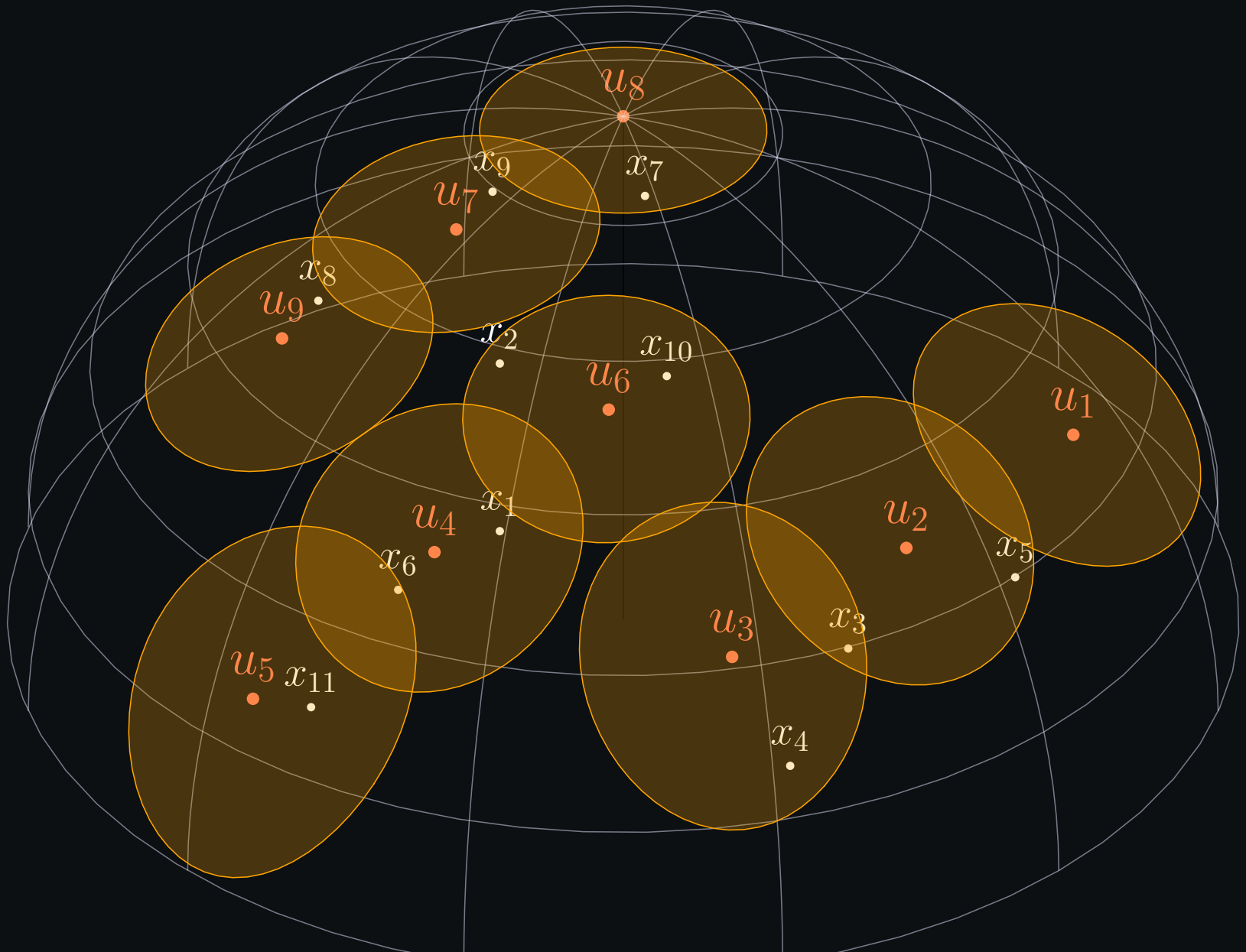
Near Neighbour problem on a sphere

Given a list L_2 of iid points on a sphere, preprocess L_2, s.t. given a query point x_1, one can quickly find $x_2 \in L_2$ with $\langle x_1 , x_2 \rangle \approx c$.

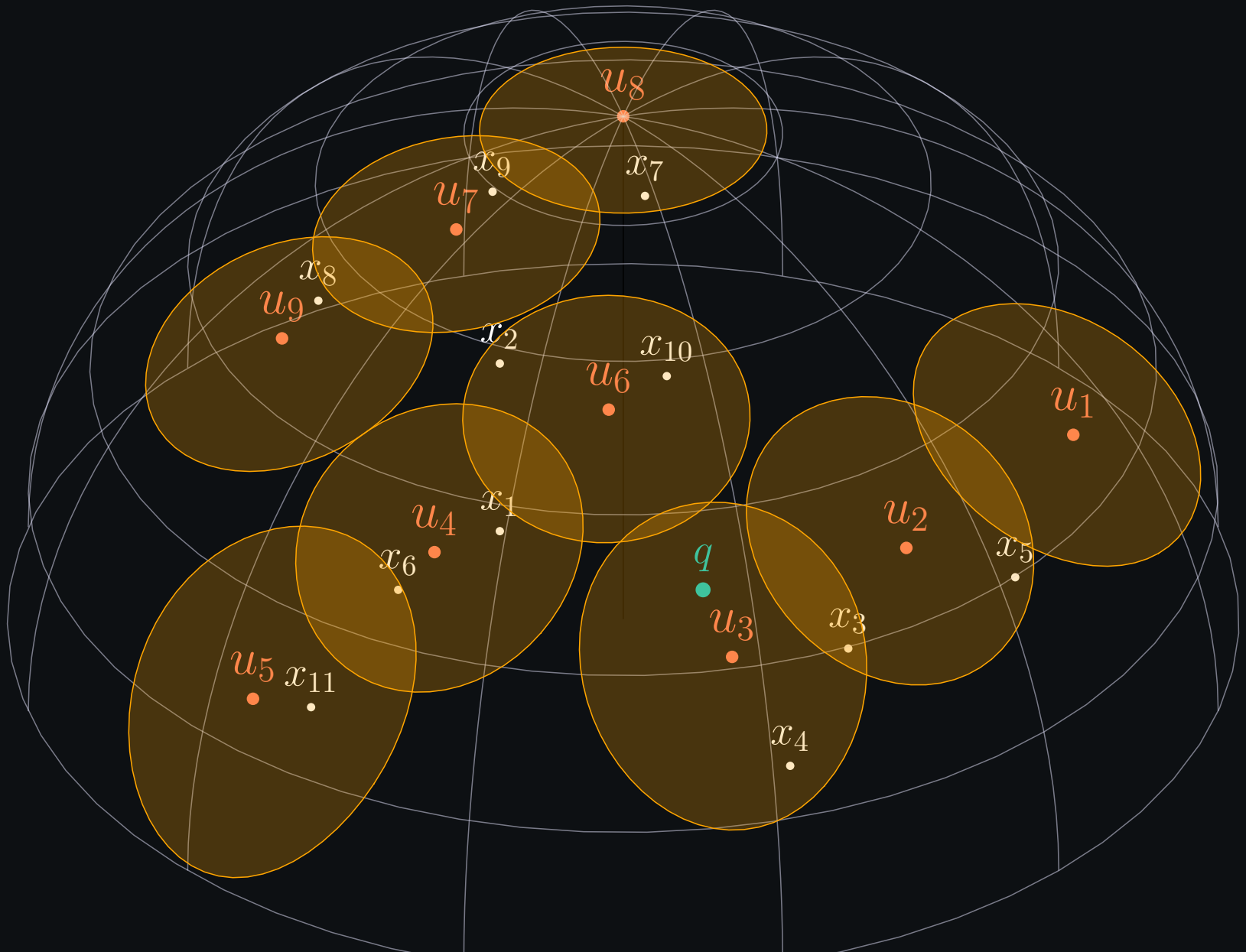
Locality-sensitive filtering [BDGL16]



Locality-sensitive filtering [BDGL16]



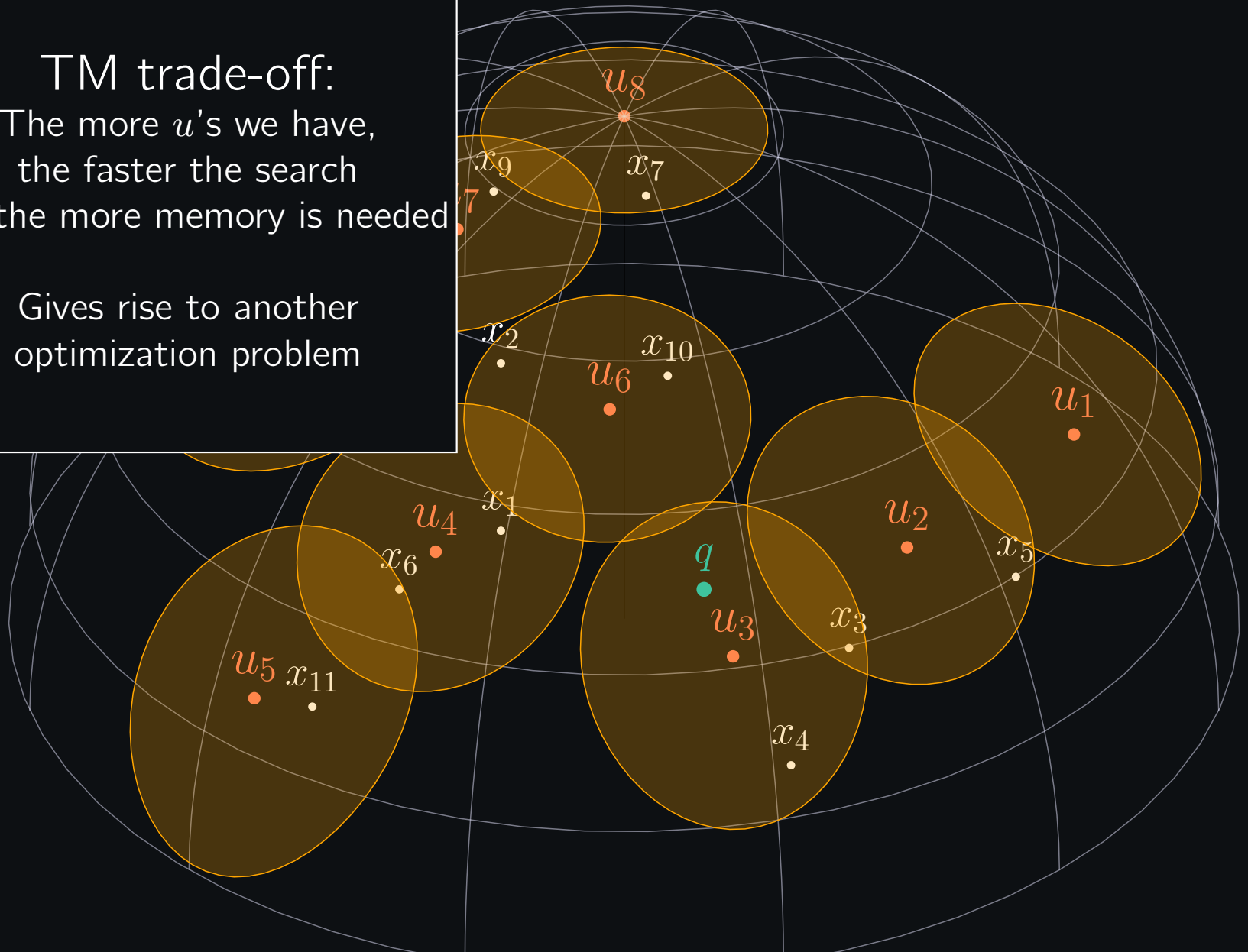
Locality-sensitive filtering [BDGL16]



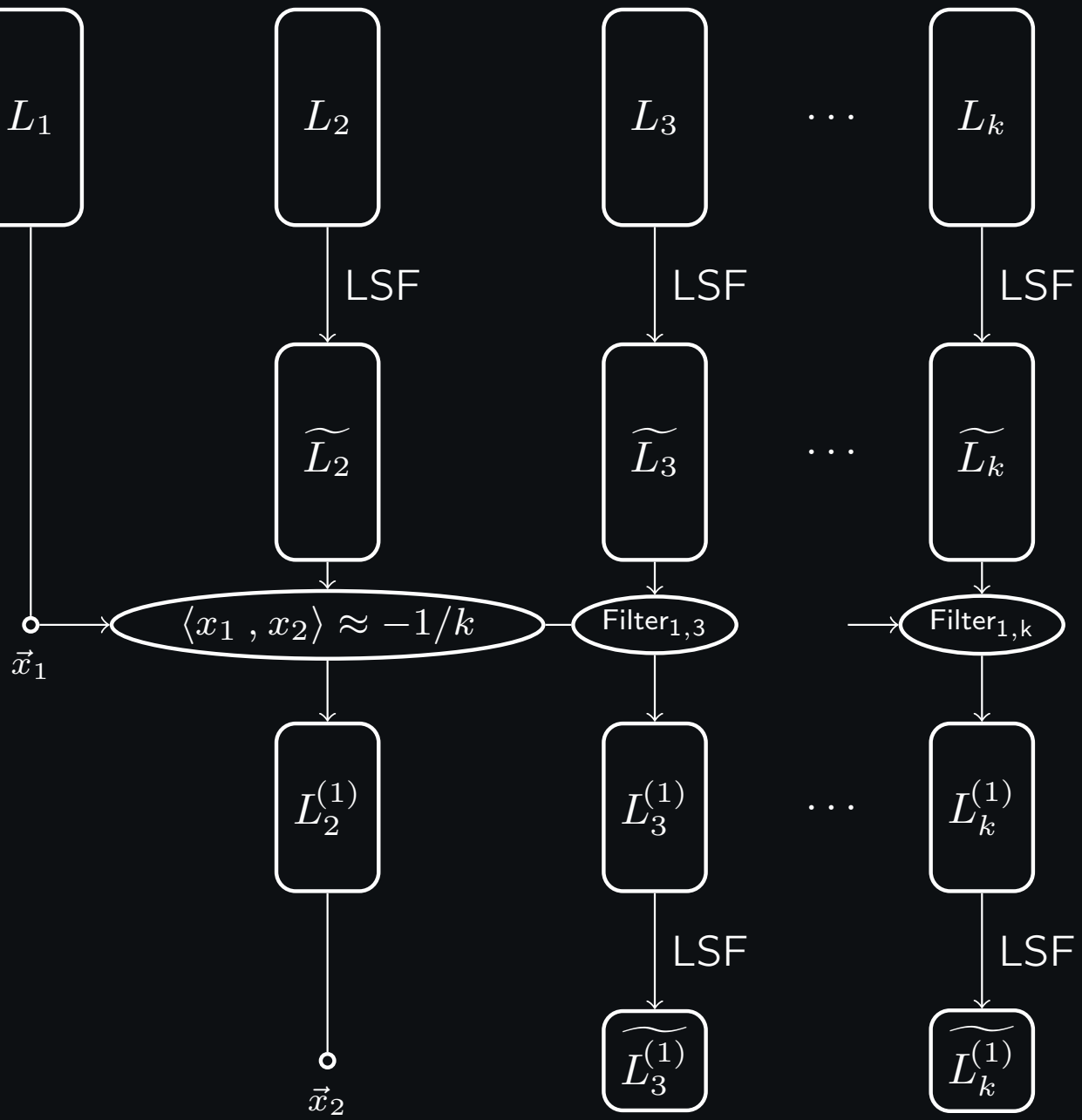
Locality-sensitive filtering [BDGL16]

TM trade-off:
The more u's we have,
the faster the search
but the more memory is needed

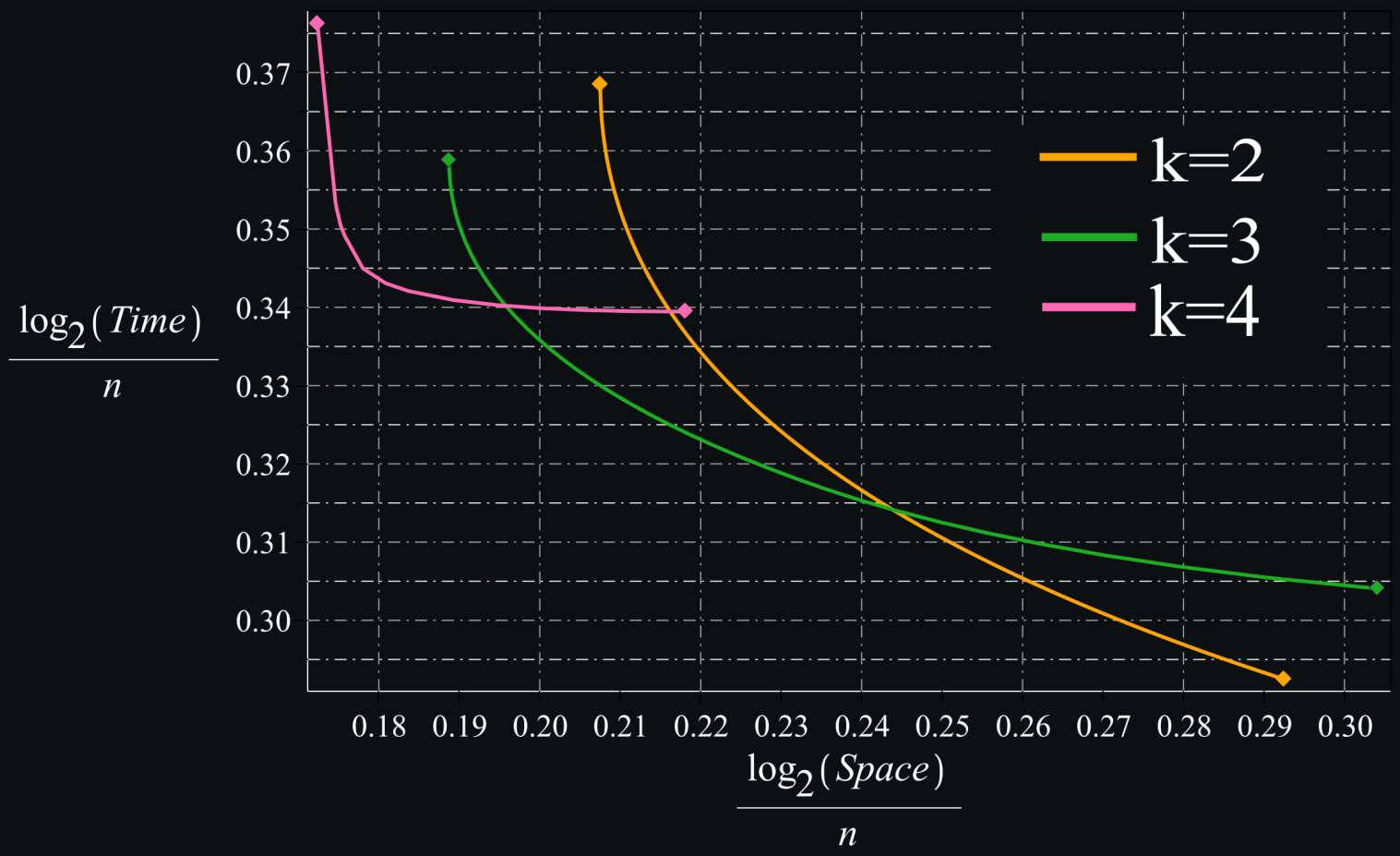
Gives rise to another
optimization problem



Algorithm v.3



More time-memory trade-offs



Tuple size (k)	2	3	4	5	6
Time	0.292	0.304	0.339	0.346	0.406
Space	0.292	0.304	0.218	0.255	0.243

Conclusions

- Estimating SVP hardness by lower-bounding memory for $k = 2$-sieve is unjustified
- Instead one should fix a memory bound, find the best k for *this* memory regime and use the complexity of the chosen k-sieve.

Conclusions

- Estimating SVP hardness by lower-bounding memory for $k = 2$-sieve is unjustified
- Instead one should fix a memory bound, find the best k for *this* memory regime and use the complexity of the chosen k-sieve.

Thank you for your attention!