How to plant Orthogonal Vectors?

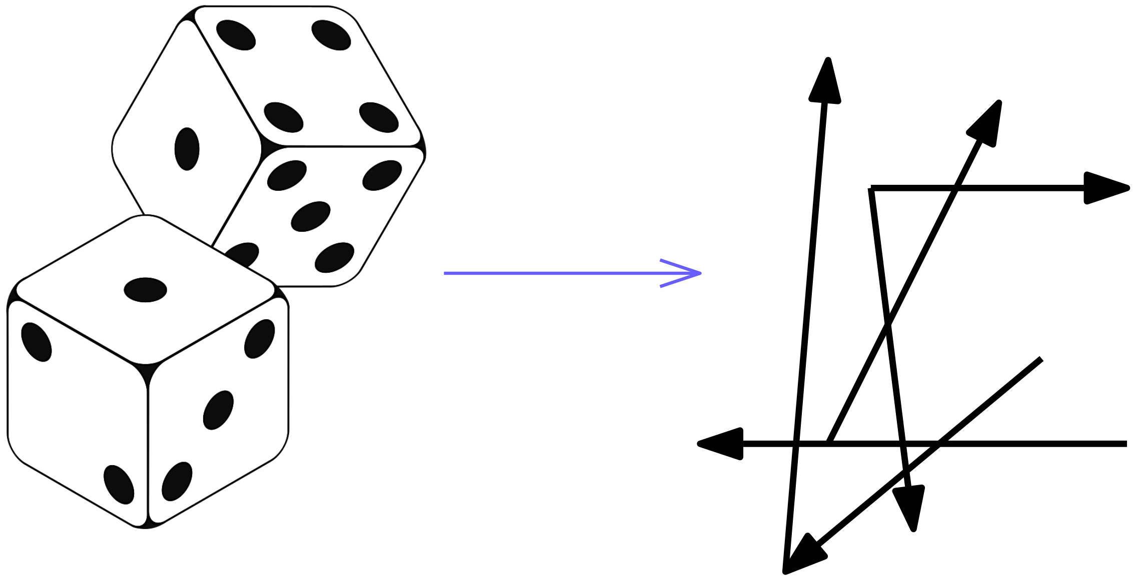
David Kühnemann

Adam Polak

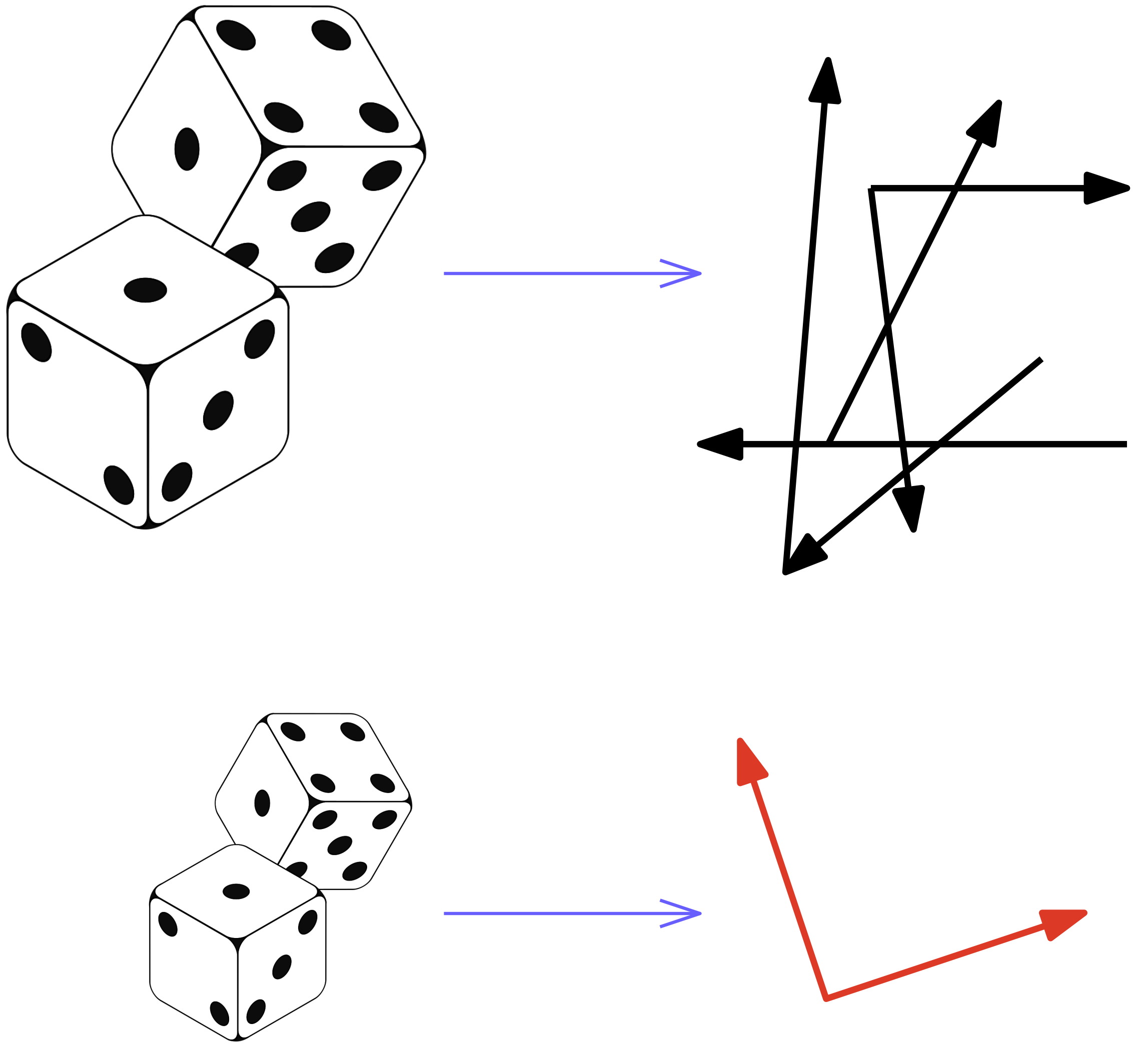
Alon Rosen

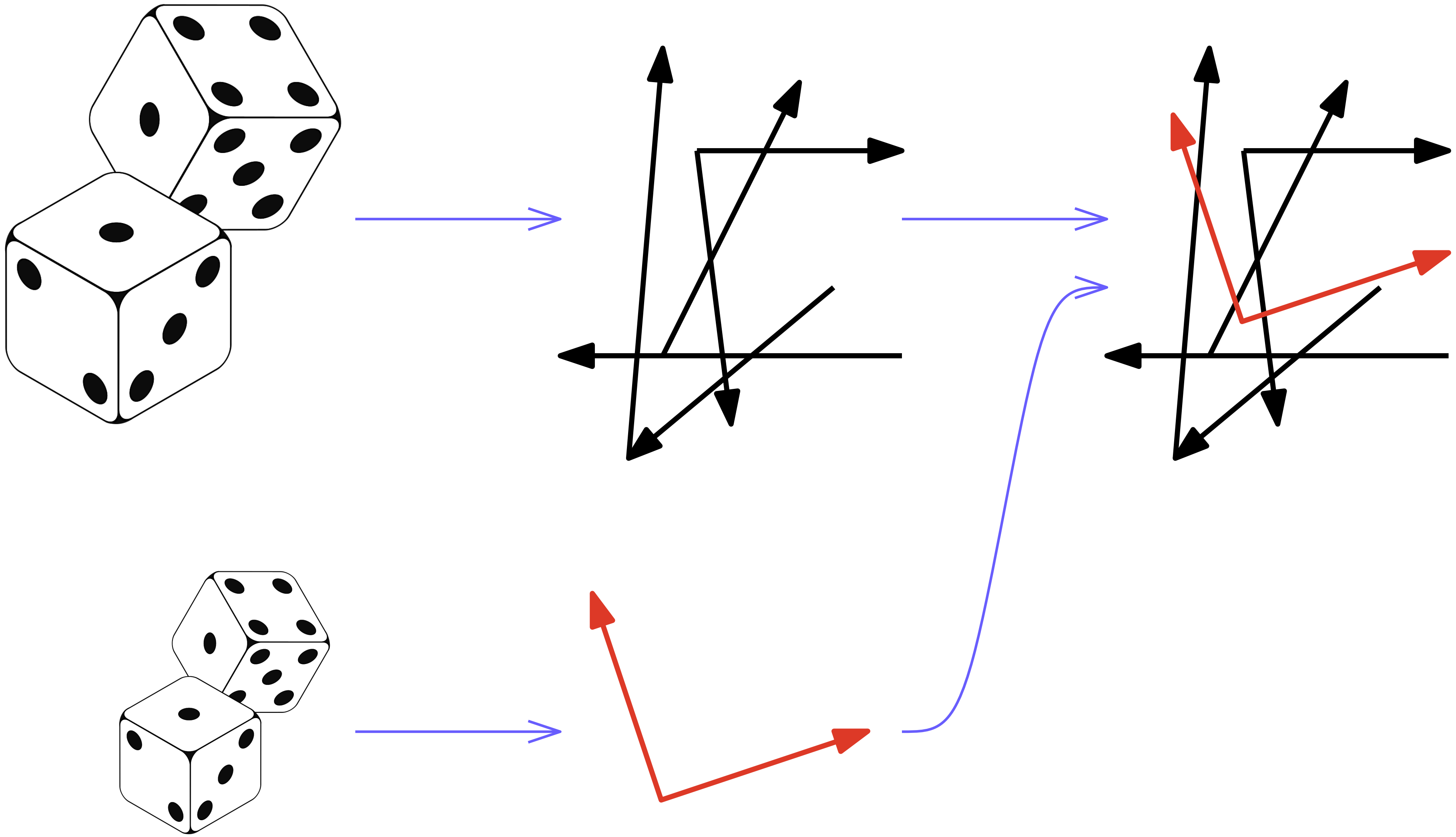
University of Amsterdam

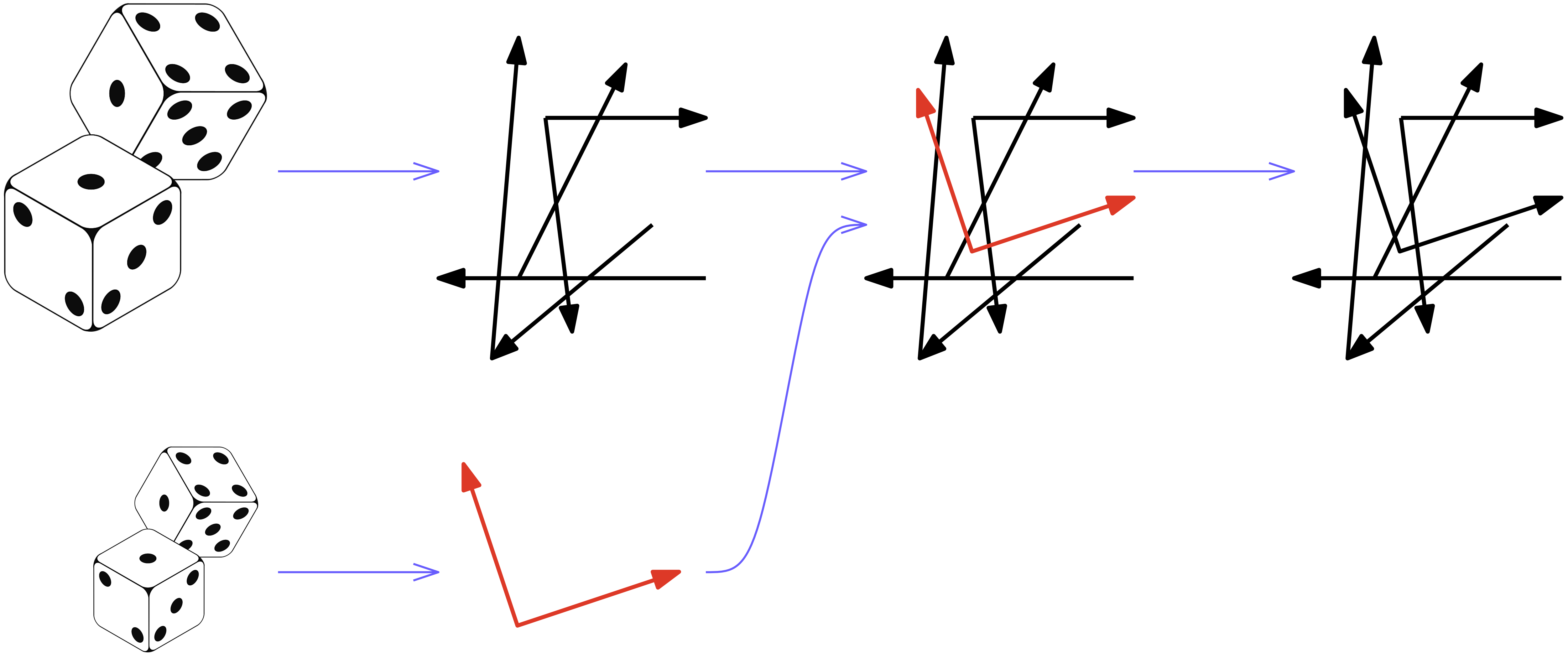
Bocconi

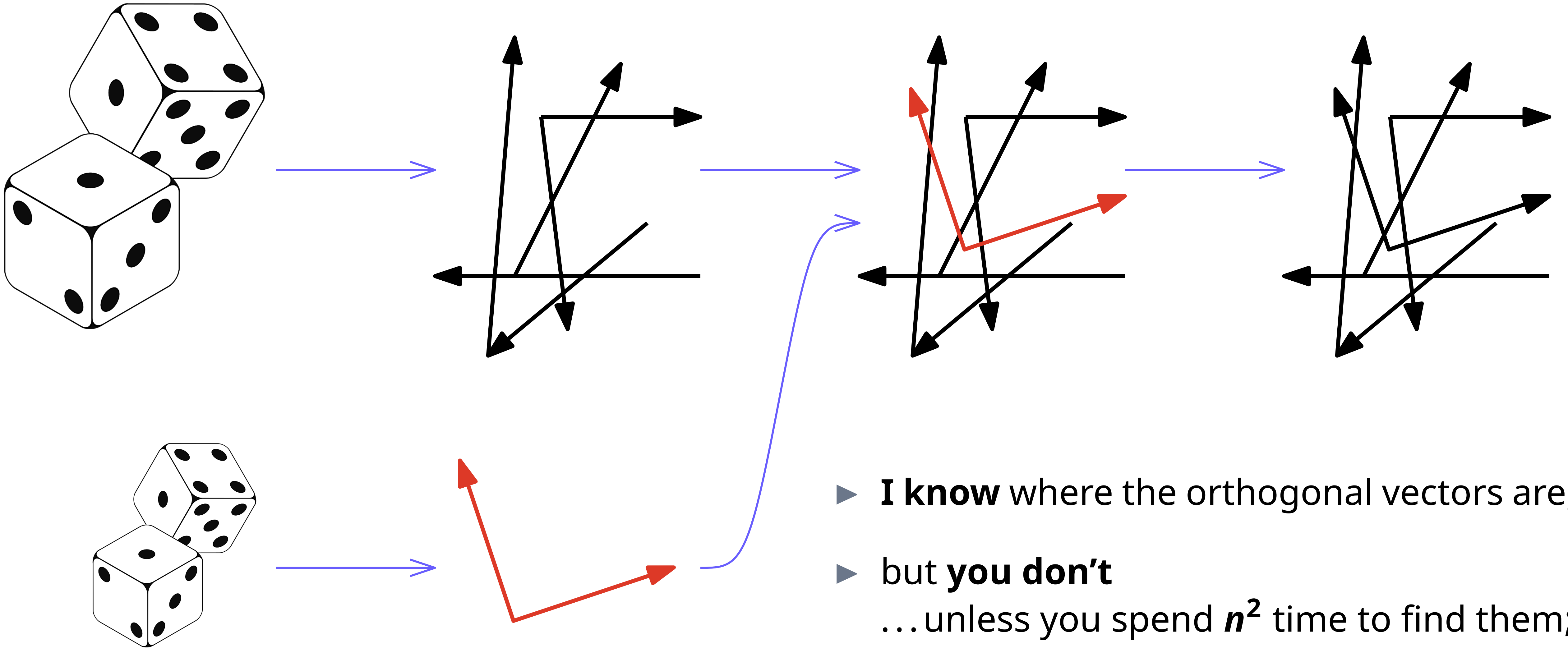


no orthogonal vectors (w.h.p.)









- **I know** where the orthogonal vectors are;
- but **you don't**
  …unless you spend $\boldsymbol{n^2}$ time to find them;
- construction has some **nice properties**.

Why study planted problems?

- Interesting theory
- Fine-grained cryptography

Why study planted problems?

- Interesting theory
- Fine-grained cryptography

Cryptographers: “Give us a problem for which we can **efficiently** generate **hard** instances with **known solutions**, like integer factorization, discrete logairthm, k-SUM, Zero-k-Clique.”

Why study fine-grained cryptography?

Why study fine-grained cryptography?

- **Dream:** "If you break my cipher in poly(n) time, then P=NP"

Why study fine-grained cryptography?

- **Dream:** “If you break my cipher in poly(n) time, then P=NP”

- **Reality:** “If you break my cipher in poly(n) time,
  then **discrete logarithm** can be solved in poly(n) time”

Why study fine-grained cryptography?

- **Dream:** "If you break my cipher in poly(n) time, then P=NP"

- **Reality:** "If you break my cipher in poly(n) time,
then **discrete logarithm** can be solved in poly(n) time"

- **Peter Shor:** "Give me a decently-sized **quantum computer**,
and I'll solve discrete logarithm in poly(n) time"

Why study fine-grained cryptography?

- **Dream:** “If you break my cipher in poly(n) time, then P=NP”
- **Reality:** “If you break my cipher in poly(n) time,
  then **discrete logarithm** can be solved in poly(n) time”
- **Peter Shor:** “Give me a decently-sized **quantum computer**,
  and I’ll solve discrete logarithm in poly(n) time”
- **Fine-grained dream:** “If you break my cipher in $\boldsymbol{n^{99}}$ time,
  then Zero-100-Clique Conjecture fails”

Why study fine-grained cryptography?

- **Dream:** “If you break my cipher in poly(n) time, then P=NP”
- **Reality:** “If you break my cipher in poly(n) time,
  then **discrete logarithm** can be solved in poly(n) time”
- **Peter Shor:** “Give me a decently-sized **quantum computer**,
  and I’ll solve discrete logarithm in poly(n) time”
- **Fine-grained dream:** “If you break my cipher in $\boldsymbol{n^{99}}$ time,
  then Zero-100-Clique Conjecture fails”
- **LaVigne–Lincoln–Vassilevska Williams:** “If you break my cipher in $\boldsymbol{n^{1.99}}$ time,
  then **Average-Case** Zero-100-Clique Conjecture fails”

Orthogonal Vectors

Input: Two sets $\boldsymbol{U}, \boldsymbol{V} \subseteq \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{d}}$ of $\boldsymbol{n}$ binary vectors of dimension $\boldsymbol{d}$

Output: Do there exist $u \in U, v \in V$ such that $\boldsymbol{u} \perp \boldsymbol{v}$?

Orthogonal Vectors

Input: Two sets $\boldsymbol{U}, \boldsymbol{V} \subseteq \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{d}}$ of $\boldsymbol{n}$ binary vectors of dimension $\boldsymbol{d}$

Output: Do there exist $u \in U, v \in V$ such that $\boldsymbol{u} \perp \boldsymbol{v}$?

$$\boxed{\forall_{i \in [d]}\ u[i] = 0 \lor v[i] = 0}$$

Orthogonal Vectors

Input: Two sets $\boldsymbol{U}, \boldsymbol{V} \subseteq \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{d}}$ of $\boldsymbol{n}$ binary vectors of dimension $\boldsymbol{d}$

Output: Do there exist $u \in U, v \in V$ such that $\boldsymbol{u} \perp \boldsymbol{v}$?

$\forall_{i \in [d]}\ u[i] = 0 \vee v[i] = 0$

Worst-case complexity:

- Naive: $\boldsymbol{O}(\boldsymbol{n}^{\mathbf{2}}\boldsymbol{d})$
- Polynomial method: $\boldsymbol{O}(\boldsymbol{n}^{\mathbf{2}-\mathbf{1}/\mathbf{log}\,\boldsymbol{c}})$ for $\boldsymbol{d} = \boldsymbol{c}\,\mathbf{log}\,\boldsymbol{n}$ [Abboud–Williams–Yu 2014] [Chan–Williams 2016]
- Under SETH, requires time $\boldsymbol{n}^{\mathbf{2}-\boldsymbol{o}(\mathbf{1})}$ for $d = \omega(\log n)$ [Williams 2005]

Orthogonal Vectors

Input: Two sets $\boldsymbol{U}, \boldsymbol{V} \subseteq \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{d}}$ of $\boldsymbol{n}$ binary vectors of dimension $\boldsymbol{d}$

Output: Do there exist $u \in U, v \in V$ such that $\boldsymbol{u} \perp \boldsymbol{v}$?

$\forall_{i \in [d]}\ u[i] = 0 \vee v[i] = 0$

Worst-case complexity:

- Naive: $\boldsymbol{O(n^2 d)}$
- Polynomial method: $\boldsymbol{O(n^{2-1/\log c})}$ for $\boldsymbol{d = c \log n}$ [Abboud–Williams–Yu 2014] [Chan–Williams 2016]
- Under SETH, requires time $\boldsymbol{n^{2-o(1)}}$ for $d = \omega(\log n)$ [Williams 2005]

Average-case complexity (i.i.d. p-biased vector entries, $d = c \log n$, $p = \Theta(1/\sqrt{c})$)

- Just look for sparse vectors: $\boldsymbol{O(n^{2-1/\log c})}$ [Kane–Williams 2019]
- Polynomial method: $\boldsymbol{O(n^{2-\log\log c/\log c})}$ [Alman–Andoni–Zhang 2025]

2-Orthogonal Vectors

Input: Two sets $\boldsymbol{U}, \boldsymbol{V} \subseteq \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{d}}$ of $\boldsymbol{n}$ binary vectors of dimension $\boldsymbol{d}$

Output: Do there exist $u \in U, v \in V$ such that $\boldsymbol{u} \perp \boldsymbol{v}$?

$\forall_{i \in [d]}\ u[i] = 0 \vee v[i] = 0$

Worst-case complexity:

- Naive: $\boldsymbol{O}(\boldsymbol{n}^2 \boldsymbol{d})$
- Polynomial method: $\boldsymbol{O}(\boldsymbol{n}^{2-1/\log c})$ for $\boldsymbol{d} = \boldsymbol{c} \log \boldsymbol{n}$ [Abboud–Williams–Yu 2014] [Chan–Williams 2016]
- Under SETH, requires time $\boldsymbol{n}^{2-o(1)}$ for $d = \omega(\log n)$ [Williams 2005]

Average-case complexity (i.i.d. p-biased vector entries, $d = c \log n$, $p = \Theta(1/\sqrt{c})$)

- Just look for sparse vectors: $\boldsymbol{O}(\boldsymbol{n}^{2-1/\log c})$ [Kane–Williams 2019]
- Polynomial method: $\boldsymbol{O}(\boldsymbol{n}^{2-\log\log c/\log c})$ [Alman–Andoni–Zhang 2025]

k-Orthogonal Vectors

Input: k sets $U_1, U_2, \ldots, U_k \subseteq \{0,1\}^d$ of n binary vectors of dimension d

Output: Do there exist $u_1 \in U_1, u_2 \in U_2, \ldots, u_k \in U_k$ such that

$$\forall_{i \in [d]}\ u_1[i] = 0 \lor u_2[i] = 0 \lor \cdots \lor u_k[i] = 0?$$

Worst-case complexity:

- Naive: $O(n^k d)$
- Polynomial method: $O(n^{k-1/\log c})$ for $d = c \log n$ [Abboud–Williams–Yu 2014] [Chan–Williams 2016]
- Under SETH, requires time $n^{k-o(1)}$ for $d = \omega(\log n)$ [Williams 2005]

How to plant orthogonal vectors so that they are hard to find?

The Planted k-Orthogonal Vectors Problem

Model distribution: i.i.d. p-biased entries

The Planted k-Orthogonal Vectors Problem

Model distribution: i.i.d. $\boldsymbol{p}$-biased entries

- $d = \omega(\log n)$ (otherwise polynomial method works in subquadratic time)
- $\boldsymbol{p} = \Omega\left(\sqrt[k]{\frac{\log n}{d}}\right)$ (guarantees NO-instance w.h.p.)
- $\boldsymbol{p} \leqslant \frac{1}{2}$ (technical detail required later, w.l.o.g. for large enough n)

The Planted k-Orthogonal Vectors Problem

Model distribution: i.i.d. $\boldsymbol{p}$-biased entries

- $d = \omega(\log n)$ (otherwise polynomial method works in subquadratic time)
- $\boldsymbol{p} = \Omega\left(\sqrt[k]{\frac{\log n}{d}}\right)$ (guarantees NO-instance w.h.p.)
- $\boldsymbol{p} \leqslant \frac{1}{2}$ (technical detail required later, w.l.o.g. for large enough n)

Planted distribution: any $k' \leqslant k-1$ vectors have **model marginal distribution**

"(k-1)-wise independence"; trivial, e.g., in k-SUM

How to generate k orthogonal vectors to plant?

For each coordinate independently, pick **#ones** according to

$$\Pr[\#\text{ones} = \boldsymbol{m}] = \binom{k}{\boldsymbol{m}} \cdot \left(p^{\boldsymbol{m}} \cdot (1-p)^{k-\boldsymbol{m}} - (-1)^{k-\boldsymbol{m}} \cdot p^k\right)$$

and pick locations of $\boldsymbol{m}$ ones uniformly at random from $\binom{k}{\boldsymbol{m}}$ options.

How to generate k orthogonal vectors to plant?

For each coordinate independently, pick **#ones** according to

Needs $p \leqslant 1/2$

$$\Pr[\#\text{ones} = m] = \binom{k}{m} \cdot \left(p^m \cdot (1-p)^{k-m} - (-1)^{k-m} \cdot p^k\right)$$

and pick locations of m ones uniformly at random from $\binom{k}{m}$ options.

How to generate k orthogonal vectors to plant?

For each coordinate independently, pick **#ones** according to

Needs $p \leqslant 1/2$

$$\Pr[\#\text{ones} = \boldsymbol{m}] = \binom{k}{\boldsymbol{m}} \cdot \left(p^{\boldsymbol{m}} \cdot (1-p)^{k-\boldsymbol{m}} - (-1)^{k-\boldsymbol{m}} \cdot p^k\right)$$

and pick locations of $\boldsymbol{m}$ ones uniformly at random from $\binom{k}{\boldsymbol{m}}$ options.

Example ($k = 2$):

$$\Pr[\mathbf{00}] = 1 - 2p \qquad \Pr[\mathbf{01}] = p \qquad \Pr[\mathbf{10}] = p \qquad \Pr[\mathbf{11}] = 0$$

How to generate k orthogonal vectors to plant?

For each coordinate independently, pick **#ones** according to

Needs $p \leqslant 1/2$

$$\Pr[\#\text{ones} = m] = \binom{k}{m} \cdot \left(p^{m} \cdot (1-p)^{k-m} - (-1)^{k-m} \cdot p^{k}\right)$$

and pick locations of m ones uniformly at random from $\binom{k}{m}$ options.

Claim: For any $\boldsymbol{k-1}$ out of those k vectors, for any coordinate, we have

$$\Pr[\#\text{ones} = \ell] = \binom{k-1}{\ell} \cdot p^{\ell} \cdot (1-p)^{k-1-\ell}$$

How to generate k orthogonal vectors to plant?

For each coordinate independently, pick **#ones** according to

Needs $p \leqslant 1/2$

$$\Pr[\#\text{ones} = \boldsymbol{m}] = \binom{k}{\boldsymbol{m}} \cdot \left(p^{\boldsymbol{m}} \cdot (1-p)^{k-\boldsymbol{m}} - (-1)^{k-\boldsymbol{m}} \cdot p^k\right)$$

and pick locations of $\boldsymbol{m}$ ones uniformly at random from $\binom{k}{\boldsymbol{m}}$ options.

Claim: For any $\boldsymbol{k-1}$ out of those k vectors, for any coordinate, we have

$$\Pr[\#\text{ones} = \ell] = \binom{k-1}{\ell} \cdot p^{\ell} \cdot (1-p)^{k-1-\ell}$$

Same as $k-1$ i.i.d. p-biased entries

How to generate k orthogonal vectors to plant?

For each coordinate independently, pick **#ones** according to

Needs $p \leqslant 1/2$

$$\Pr[\#\text{ones} = m] = \binom{k}{m} \cdot \left(p^{m} \cdot (1-p)^{k-m} - (-1)^{k-m} \cdot p^{k}\right)$$

and pick locations of m ones uniformly at random from $\binom{k}{m}$ options.

Claim: For any $\boldsymbol{k-1}$ out of those k vectors, for any coordinate, we have

$$\Pr[\#\text{ones} = \ell] = \binom{k-1}{\ell} \cdot p^{\ell} \cdot (1-p)^{k-1-\ell}$$

Proof: Just do the math.

Same as $k-1$ i.i.d. p-biased entries

How to generate k orthogonal vectors to plant?

conveniently

- Generate $\boldsymbol{k-1}$ i.i.d. p-biased vectors $u_1, u_2, \ldots, u_{k-1}$
- For each coordinate $i \in [d]$
  - $\boldsymbol{\ell} := u_1[i] + u_2[i] + \cdots + u_{k-1}[i]$
  - Set $u_k[i] := \begin{cases} 1 & \text{with probability} \quad p \cdot \left(1 - \left(\frac{-p}{1-p}\right)^{k-1-\boldsymbol{\ell}}\right) \\ 0 & \text{otherwise} \end{cases}$

#ones among first $k-1$ vectors

How to generate k orthogonal vectors to plant?

conveniently

- Generate $\boldsymbol{k-1}$ i.i.d. p-biased vectors $u_1, u_2, \ldots, u_{k-1}$
- For each coordinate $i \in [d]$

#ones among first $k-1$ vectors

  - $\boldsymbol{\ell} := u_1[i] + u_2[i] + \cdots + u_{k-1}[i]$
  - Set $u_k[i] := \begin{cases} 1 & \text{with probability} \quad p \cdot \left(1 - \left(\frac{-p}{1-p}\right)^{k-1-\boldsymbol{\ell}}\right) \\ 0 & \text{otherwise} \end{cases}$

Claim: Same distribtution as the previous slide.

How to generate k orthogonal vectors to plant?

conveniently

- Generate $\boldsymbol{k-1}$ i.i.d. p-biased vectors $u_1, u_2, \ldots, u_{k-1}$
- For each coordinate $i \in [d]$

  #ones among first $k-1$ vectors

  - $\boldsymbol{\ell} := u_1[i] + u_2[i] + \cdots + u_{k-1}[i]$
  - Set $u_k[i] := \begin{cases} 1 & \text{with probability} \quad p \cdot \left(1 - \left(\frac{-p}{1-p}\right)^{k-1-\boldsymbol{\ell}}\right) \\ 0 & \text{otherwise} \end{cases}$

Claim: Same distribtution as the previous slide.

Proof: Just do the math.

Search-to-decision reduction

A **decision algorithm** gets an instance drawn either from the **model distribution** or the **planted distribution** and outputs which one it was

A **search algorithm** gets an instance drawn from the **planted distribution** and outputs indices of the planted vectors

Theorem: If there is a $T(n)$-time **decision algorithm** for k-OV with success probability $1 - p(n)$, then there is an $O(T(n) \log n)$-time **search algorithm** for k-OV with success probability $1 - k \log(n) p(n)$

Search-to-decision reduction

Key idea:

- **Replace some vectors** in the input instance with i.i.d. p-biased vectors
- If **none of the planted vectors** got replaced, the instance is distributed according to the **planted dsitribution**
- If **at least one planted vector** got replaced, the instance is distributed according to the **model dsitribution**

Search-to-decision reduction

Key idea:

- **Replace some vectors** in the input instance with i.i.d. p-biased vectors
- If **none of the planted vectors** got replaced, the instance is distributed according to the **planted dsitribution**
- If **at least one planted vector** got replaced, the instance is distributed according to the **model dsitribution**

Thanks to (k-1)-wise independence

Search-to-decision reduction

Key idea:

- **Replace some vectors** in the input instance with i.i.d. p-biased vectors
- If **none of the planted vectors** got replaced, the instance is distributed according to the **planted dsitribution**
- If **at least one planted vector** got replaced, the instance is distributed according to the **model dsitribution**

Thanks to (k-1)-wise independence

Reduction: Use the key idea to implement **binary search**

A different search-to-decision reduction

Theorem [Agrawal et al., CRYPTO'24]

For every $\delta > 0$ there exists $\epsilon > 0$ such that if there is a $T(n)$-time **decision algorithm** for k-SUM with success probability $1 - \epsilon$, then there is an $O(T(n) \log n)$-time **search algorithm** for k-SUM with success probability $1 - \delta$

A different search-to-decision reduction

Theorem [Agrawal et al., CRYPTO'24]

For every $\delta > 0$ there exists $\epsilon > 0$ such that
if there is a $T(n)$-time **decision algorithm** for k-SUM with success probability $1 - \epsilon$,
then there is an $O(T(n) \log n)$-time **search algorithm** for k-SUM with success probability $1 - \delta$

Theorem [this work]

For every $\delta > 0$ there exists $\epsilon > 0$ such that
if there is a $T(n)$-time **decision algorithm** for **k-OV** with success probability $1 - \epsilon$,
then there is an $O(T(n) \log n)$-time **search algorithm** for **k-OV** with success probability $1 - \delta$

A different search-to-decision reduction

Theorem: For every $\delta > 0$ there exists $\epsilon > 0$ such that [this work]
if there is a $T(n)$-time **decision algorithm** for **k-OV** with success probability $1 - \epsilon$, then there is an $O(T(n) \log n)$-time **search algorithm** for **k-OV** with success probability $1 - \delta$

A different search-to-decision reduction

Theorem: For every $\delta > 0$ there exists $\epsilon > 0$ such that [this work]
if there is a $T(n)$-time **decision algorithm** for **k-OV** with success probability $1 - \epsilon$, then there is an $O(T(n) \log n)$-time **search algorithm** for **k-OV** with success probability $1 - \delta$

Search algorithm:

- Create a **counter** for each vector, initially set to 0
- **Repeat** $O(\log n)$ times:
  - **Replace** a random $1 - \sqrt[k]{1/2}$ fraction of vectors with freshly sampled ones
  - If the **decision algorithm** says *planted*, increment counters of non-replaced
- Return the k vectors with **highest counters**

A different search-to-decision reduction

Theorem: For every $\delta > 0$ there exists $\epsilon > 0$ such that [this work]
if there is a $T(n)$-time **decision algorithm** for **k-OV** with success probability $1 - \epsilon$, then there is an $O(T(n) \log n)$-time **search algorithm** for **k-OV** with success probability $1 - \delta$

Search algorithm:

- Create a **counter** for each vector, initially set to 0
- **Repeat** $O(\log n)$ times:
  - **Replace** a random $1 - \sqrt[k]{1/2}$ fraction of vectors with freshly sampled ones
  - If the **decision algorithm** says *planted*, increment counters of non-replaced
- Return the k vectors with **highest counters**

Proof of correctness: “Just” do the math (~3 pages of math)

Summary

- We show how to sample k orthogonal vectors so that any $k-1$ of them look like i.i.d. p-biased vectors
- We conjecture it takes time $n^{k-o(1)}$ to find them among i.i.d. p-biased vectors

Summary

- We show how to sample k orthogonal vectors so that any $k-1$ of them look like i.i.d. p-biased vectors
- We conjecture it takes time $n^{k-o(1)}$ to find them among i.i.d. p-biased vectors

Open problems

- Build **public-key cryptography** based on hardness of k-OV
- Show hardness amplification, or a worst-case to average-case reduction
- **Refute** our conjecture

Summary

- We show how to sample k orthogonal vectors so that any $k-1$ of them look like i.i.d. p-biased vectors
- We conjecture it takes time $n^{k-o(1)}$ to find them among i.i.d. p-biased vectors

Open problems

- Build **public-key cryptography** based on hardness of k-OV
- Show hardness amplification, or a worst-case to average-case reduction
- **Refute** our conjecture

THANK YOU!