Efficient Algorithms for Semirandom Planted CSPs at the Refutation Threshold

Jun-Ting (Tim) Hsieh, CMU

Joint work with

Venkat Guruswami
UC Berkeley

Pravesh K. Kothari
CMU

Peter Manohar
CMU

3-SAT

$$\psi = (x_1 \vee \overline{x}_2 \vee x_5) \wedge (x_2 \vee \overline{x}_4 \vee x_6) \wedge \cdots$$

3-SAT

$$\psi = (x_1 \vee \bar{x}_2 \vee x_5) \wedge (x_2 \vee \bar{x}_4 \vee x_6) \wedge \cdots$$

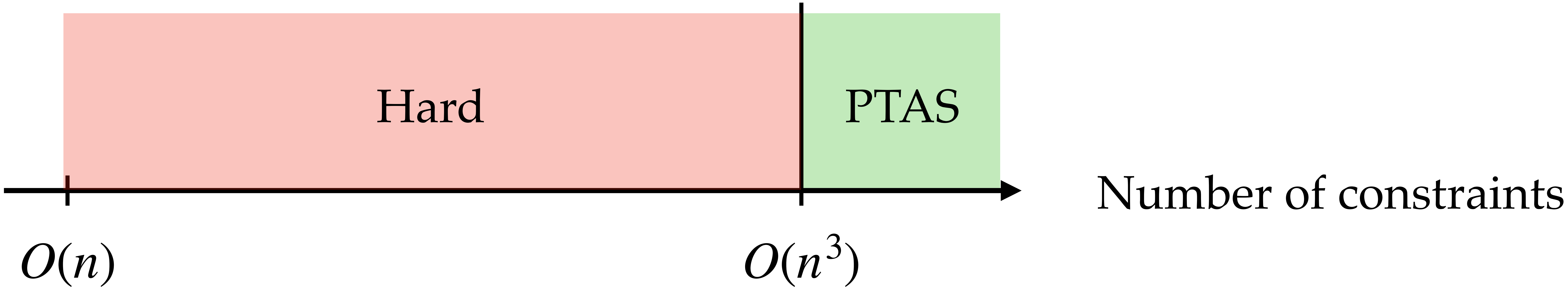
NP-hard to satisfy $\frac{7}{8} + \epsilon$ fraction of the clauses [Hastad'01].

3-SAT

$$\psi = (x_1 \vee \bar{x}_2 \vee x_5) \wedge (x_2 \vee \bar{x}_4 \vee x_6) \wedge \cdots$$

NP-hard to satisfy $\frac{7}{8} + \epsilon$ fraction of the clauses [Hastad'01].

Under ETH: **hard** even for dense instances ($o(n^3)$ clauses) [Fotakis-Lampis-Paschos'16].

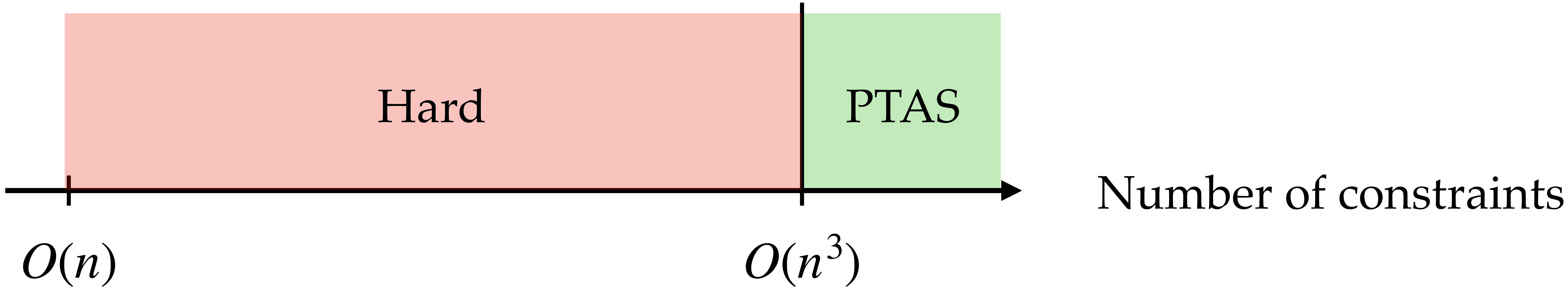


3-SAT

$$\psi = (x_1 \vee \bar{x}_2 \vee x_5) \wedge (x_2 \vee \bar{x}_4 \vee x_6) \wedge \cdots$$

NP-hard to satisfy $\frac{7}{8} + \epsilon$ fraction of the clauses [Hastad'01].

Under ETH: **hard** even for dense instances ($o(n^3)$ clauses) [Fotakis-Lampis-Paschos'16].



Worst-case setting.

Random 3-SAT

Random 3-SAT

Random 3-SAT instance: for each clause,

Random 3-SAT

Random 3-SAT instance: for each clause,

- Random 3-tuple, $(1, 2, 5)$ $(2, 4, 6)$

Random 3-SAT

Random 3-SAT instance: for each clause,

- Random 3-tuple, $(1, 2, 5)$ $(2, 4, 6)$
- Random literal patterns. $(x_1 \lor \bar{x}_2 \lor x_5)$ $(\bar{x}_2 \lor \bar{x}_4 \lor x_6)$

Random 3-SAT

Random 3-SAT instance: for each clause,

- Random 3-tuple, $(1, 2, 5)$ $(2, 4, 6)$

- Random literal patterns. $(x_1 \vee \bar{x}_2 \vee x_5)$ $(\bar{x}_2 \vee \bar{x}_4 \vee x_6)$

Can we find a good approximate solution?

Random 3-SAT

Random 3-SAT instance: for each clause,

- Random 3-tuple, $(1, 2, 5)$ $(2, 4, 6)$
- Random literal patterns. $(x_1 \vee \bar{x}_2 \vee x_5)$ $(\bar{x}_2 \vee \bar{x}_4 \vee x_6)$

Can we find a good approximate solution?

Fact: All assignments satisfy $\frac{7}{8} \pm o(1)$ fraction of the clauses.

Random 3-SAT refutation

Random 3-SAT refutation

Refutation: given a **random** instance, certify that it is unsatisfiable.

Random 3-SAT refutation

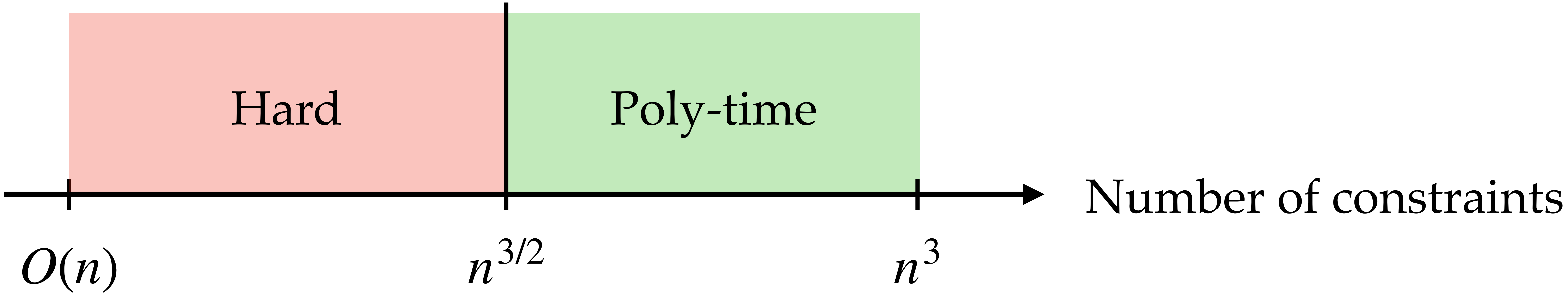
Refutation: given a **random** instance, certify that it is unsatisfiable.

There is poly-time algorithm when $m = O(n^{3/2})$ clauses [Coja-Oghlan-Goerdt-Lanka'07].

Random 3-SAT refutation

Refutation: given a **random** instance, certify that it is unsatisfiable.

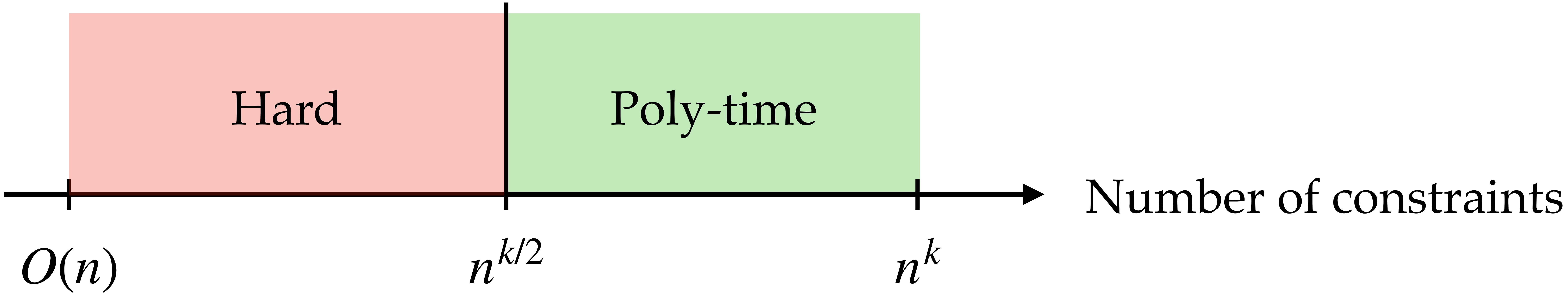
There is poly-time algorithm when $m = O(n^{3/2})$ clauses [Coja-Oghlan-Goerdt-Lanka'07].



Random k-CSP refutation

Refutation: given a **random** instance, certify that it is unsatisfiable.

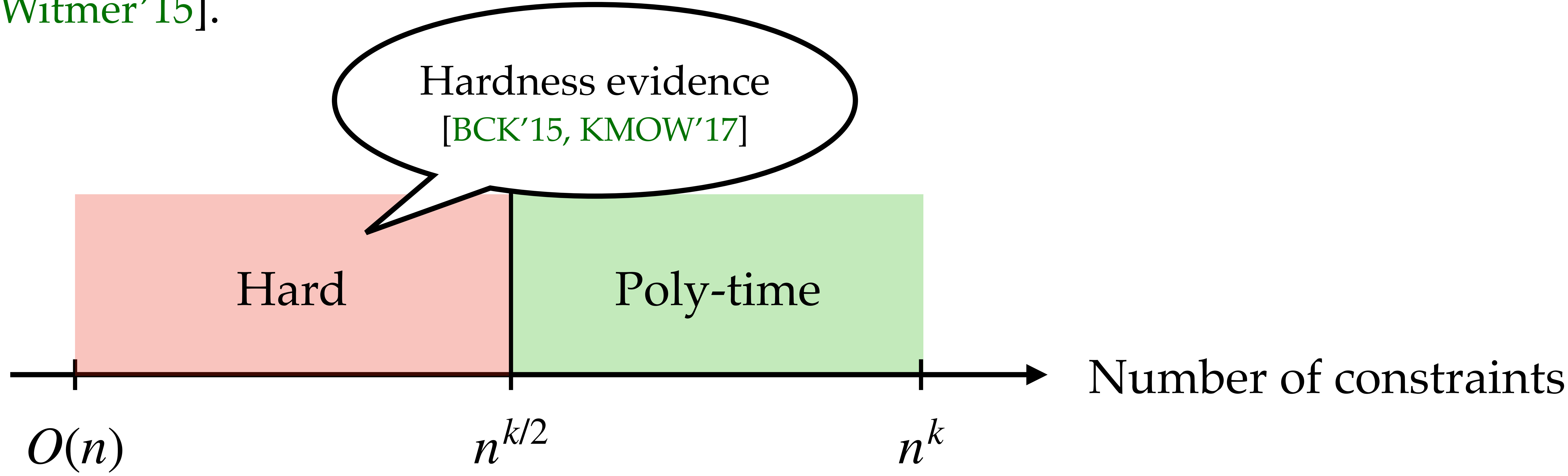
There is poly-time algorithm when $m = O(n^{k/2})$ clauses [Coja-Oghlan-Goerdt-Lanka'07, Allen-O'Donnel-Witmer'15].



Random k-CSP refutation

Refutation: given a **random** instance, certify that it is unsatisfiable.

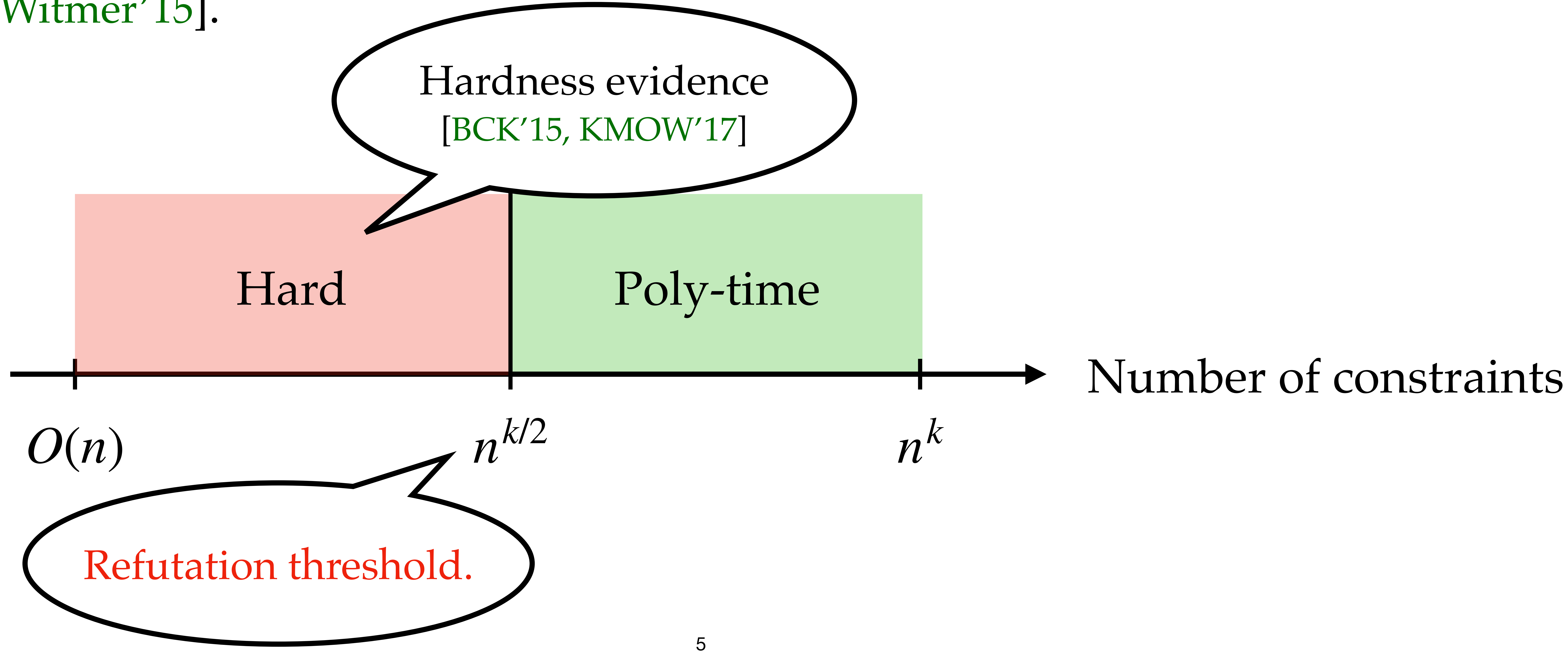
There is poly-time algorithm when $m = O(n^{k/2})$ clauses [Coja-Oghlan-Goerdt-Lanka'07, Allen-O'Donnel-Witmer'15].



Random k-CSP refutation

Refutation: given a **random** instance, certify that it is unsatisfiable.

There is poly-time algorithm when $m = O(n^{k/2})$ clauses [Coja-Oghlan-Goerdt-Lanka'07, Allen-O'Donnel-Witmer'15].



Random k-CSPs

Random k-CSPs

Search: the instance is randomly generated with a **satisfying** planted assignment x^*.

Random k-CSPs

Search: the instance is randomly generated with a **satisfying** planted assignment x^*.

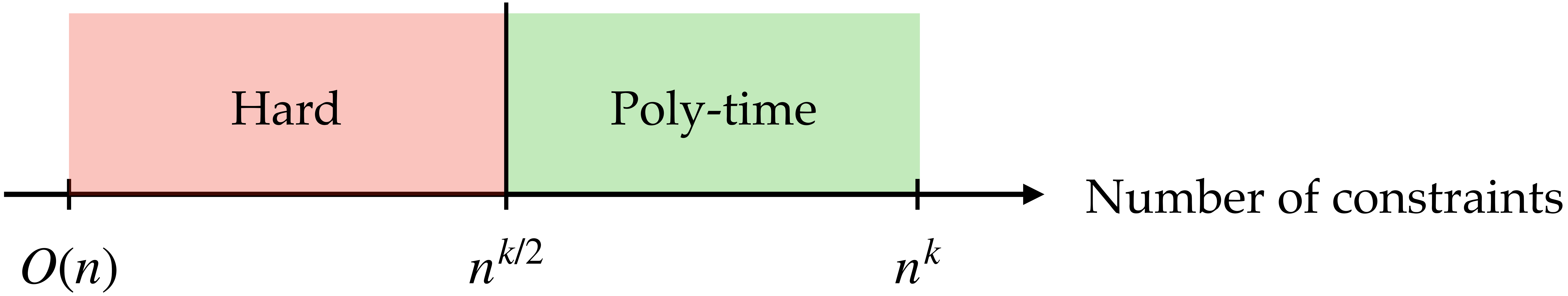
Goal: find x^*.

Random k-CSPs

Search: the instance is randomly generated with a **satisfying** planted assignment x^*.

Goal: find x^*.

There is a poly-time algorithm when $m = O(n^{k/2})$ clauses [Feldman-Perkins-Vempala'15].

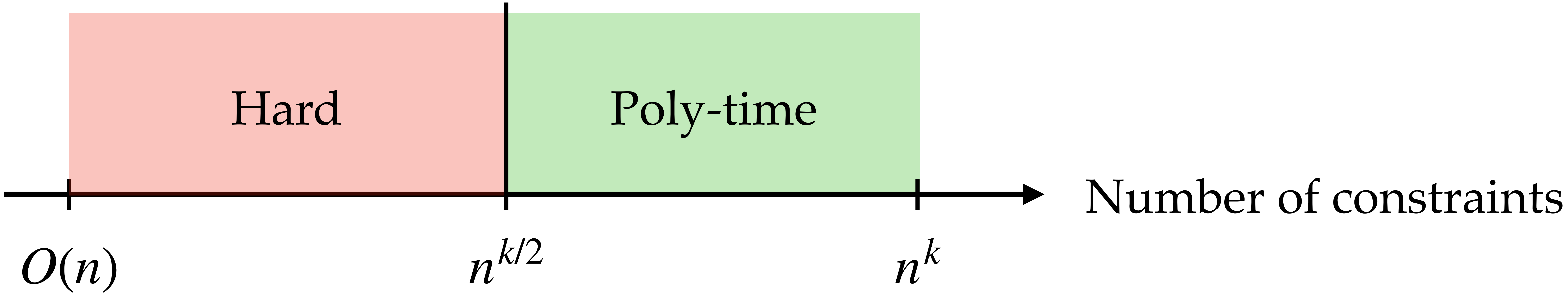


Random k-CSPs

Search: the instance is randomly generated with a **satisfying** planted assignment x^*.

Goal: find x^*.

There is a poly-time algorithm when $m = O(n^{k/2})$ clauses [Feldman-Perkins-Vempala'15].



Same as the refutation threshold!

Semirandom models

Question: Do these algorithms rely too heavily on the specific random models?

Semirandom models

Question: Do these algorithms rely too heavily on the specific random models?

Yes — most known algorithms break down under minor perturbations.

Semirandom models

Question: Do these algorithms rely too heavily on the specific random models?

Yes — most known algorithms break down under minor perturbations.

Semirandom models: instances constructed from both average-case and adversarial worst-case choices [Blum-Spencer'95, Feige-Kilian'01].

Semirandom models

Question: Do these algorithms rely too heavily on the specific random models?

Yes — most known algorithms break down under minor perturbations.

Semirandom models: instances constructed from both average-case and adversarial worst-case choices [Blum-Spencer'95, Feige-Kilian'01].

- Algorithms that succeed are more "robust".

Semirandom models

Question: Do these algorithms rely too heavily on the specific random models?

Yes — most known algorithms break down under minor perturbations.

Semirandom models: instances constructed from both average-case and adversarial worst-case choices [Blum-Spencer'95, Feige-Kilian'01].

- Algorithms that succeed are more "robust".
- Understand which "randomness" is unnecessary.

Semirandom CSPs

Semirandom CSPs

- Worst-case clause structure (hypergraph).

(1, 2, 5) (2, 4, 6)

Semirandom CSPs

- Worst-case clause structure (hypergraph).
- Random literal patterns.

$(1, 2, 5)$ $(2, 4, 6)$

$(x_1 \vee \bar{x}_2 \vee x_5)$ $(\bar{x}_2 \vee \bar{x}_4 \vee x_6)$

Semirandom CSPs

(Fully) random = semirandom + random hypergraph

- Worst-case clause structure (hypergraph).
- Random literal patterns.

(1, 2, 5) (2, 4, 6)

$(x_1 \vee \bar{x}_2 \vee x_5)$ $(\bar{x}_2 \vee \bar{x}_4 \vee x_6)$

Semirandom CSPs

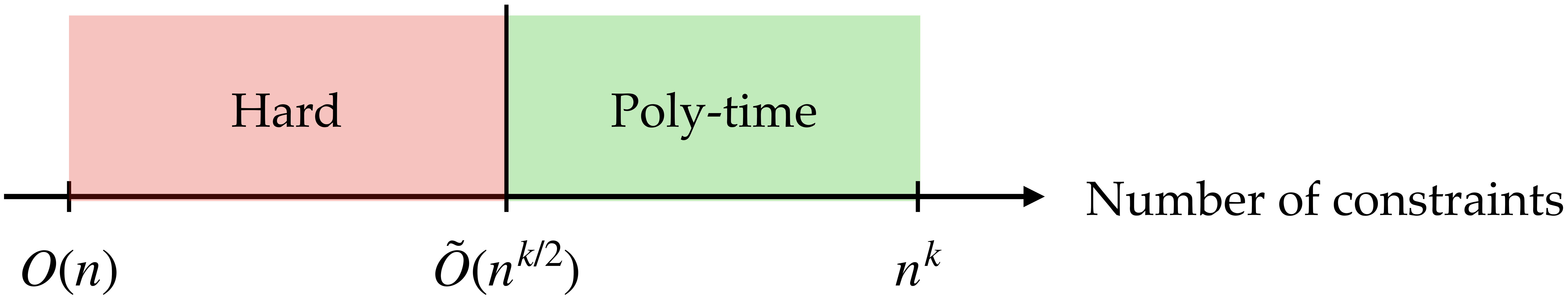
(Fully) random = semirandom + random hypergraph

- Worst-case clause structure (hypergraph).
- Random literal patterns.

(1, 2, 5)   (2, 4, 6)

$(x_1 \vee \bar{x}_2 \vee x_5) \quad (\bar{x}_2 \vee \bar{x}_4 \vee x_6)$

Refutation: same threshold as fully random [Abascal-Guruswami-Kothari'21, Guruswami-Kothari-Manohar'22, Hsieh-Kothari-Mohanty'22].



Semirandom CSPs

(Fully) random = semirandom + random hypergraph

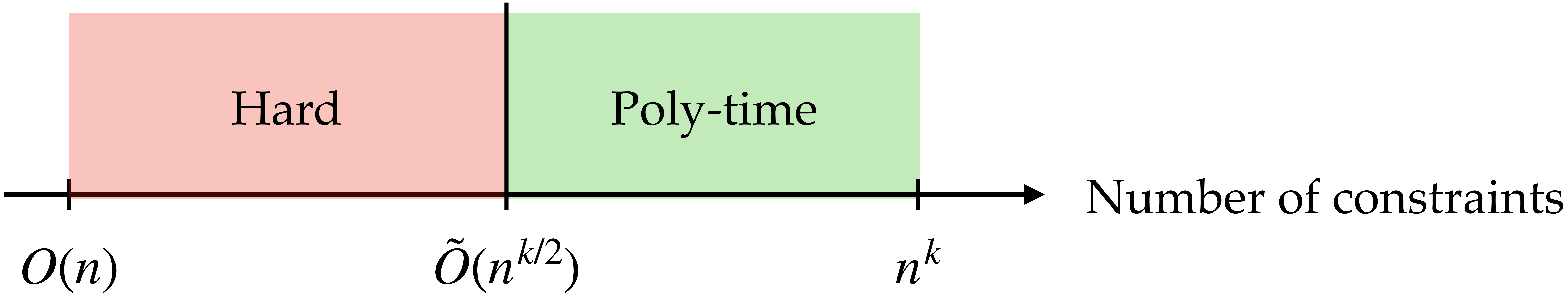
- Worst-case clause structure (hypergraph).
- Random literal patterns.

$(1, 2, 5)$ $\quad$ $(2, 4, 6)$

$(x_1 \vee \bar{x}_2 \vee x_5)$ $\quad$ $(\bar{x}_2 \vee \bar{x}_4 \vee x_6)$

Refutation: same threshold as fully random [Abascal-Guruswami-Kothari'21, Guruswami-Kothari-Manohar'22, Hsieh-Kothari-Mohanty'22].

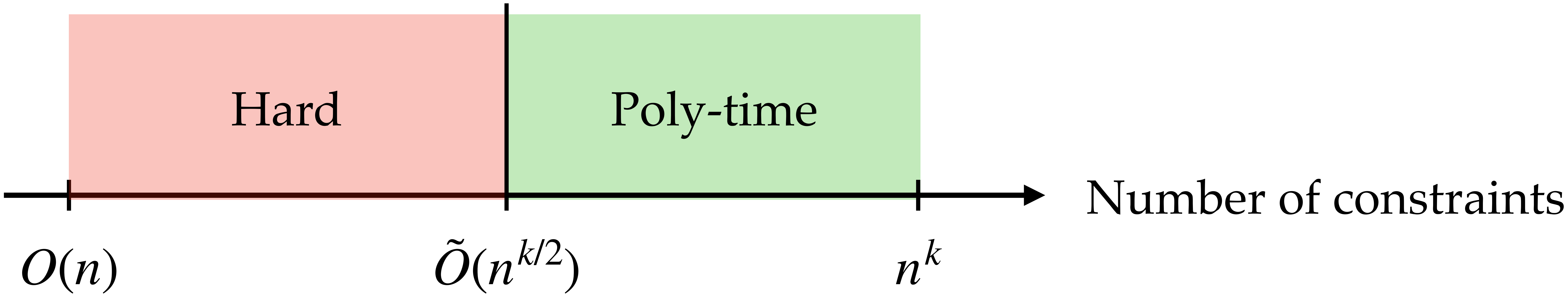
Search (planted): ?



Our results

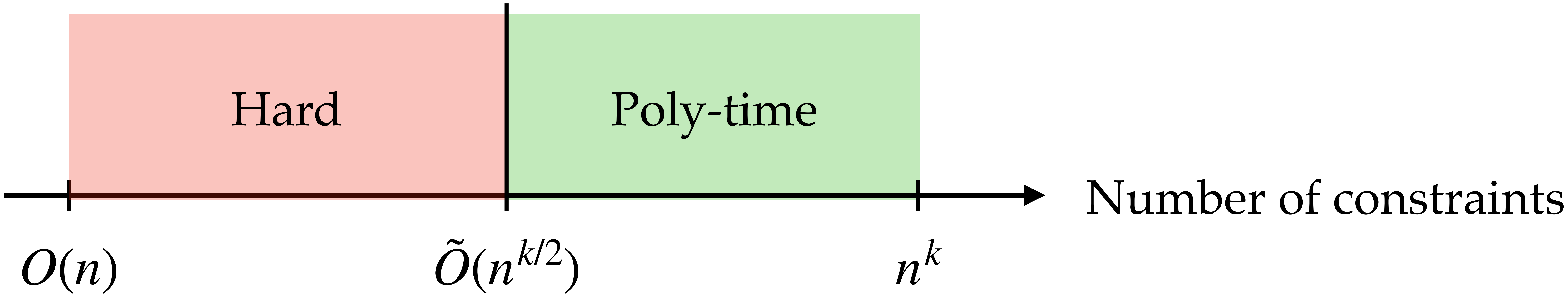
Our results

Theorem 1. Given a semirandom planted k-CSP with $m \geq \tilde{O}(n^{k/2})$ constraints, our algorithm outputs an x that satisfies $1 - o(1)$ fraction of the clauses.



Our results

Theorem 1. Given a semirandom planted k-CSP with $m \geq \tilde{O}(n^{k/2})$ constraints, our algorithm outputs an x that satisfies $1 - o(1)$ fraction of the clauses.



Same as the refutation threshold!

Our results

Theorem 1. Given a semirandom planted k-CSP with $m \geq \tilde{O}(n^{k/2})$ constraints, our algorithm outputs an x that satisfies $1 - o(1)$ fraction of the clauses.

Our algorithm goes through a reduction to **noisy planted *k*-XOR**.

Our results

Theorem 1. Given a semirandom planted k-CSP with $m \geq \tilde{O}(n^{k/2})$ constraints, our algorithm outputs an x that satisfies $1 - o(1)$ fraction of the clauses.

Our algorithm goes through a reduction to **noisy planted *k*-XOR**.

Fourier analysis.

Noisy planted k-XOR

Instance:

Noisy planted k-XOR

Instance:

- Arbitrary k-uniform hypergraph G.

Noisy planted k-XOR

Instance:

- Arbitrary k-uniform hypergraph G.
- Arbitrary $x^* \in \{\pm 1\}^n$.

Noisy planted k-XOR

Instance:

- Arbitrary k-uniform hypergraph G.
- Arbitrary $x^* \in \{\pm 1\}^n$.
- For each $C \in G$, add a constraint $\prod_{i \in C} x_i = b_C := \prod_{i \in C} x_i^*$.

Noisy planted k-XOR

Instance:

- Arbitrary k-uniform hypergraph G.
- Arbitrary $x^* \in \{\pm 1\}^n$.
- For each $C \in G$, add a constraint $\prod_{i \in C} x_i = b_C := \prod_{i \in C} x_i^*$.
- Flip the sign b_C of each $C \in G$ with probability $\eta < 1/2$.

Noisy planted k-XOR

The maximum info about x^* that we can retrieve is if we knew all the corrupted constraints.

Noisy planted k-XOR

The maximum info about x^* that we can retrieve is if we knew all the corrupted constraints.

Ideal Goal: identify all corrupted constraints.

Noisy planted k-XOR

The maximum info about x^* that we can retrieve is if we knew all the corrupted constraints.

Ideal Goal: identify all corrupted constraints.

Worst-case hypergraph: not possible.

Noisy planted k-XOR

The maximum info about x^* that we can retrieve is if we knew all the corrupted constraints.

Ideal Goal: identify all corrupted constraints.

Worst-case hypergraph: not possible.

Our result: we identify **almost all** corrupted constraints.

Our results

Theorem 2. Given a noisy k-XOR instance with $m \geq \tilde{O}(n^{k/2})$ constraints, our algorithm outputs:

Our results

Theorem 2. Given a noisy k-XOR instance with $m \geq \tilde{O}(n^{k/2})$ constraints, our algorithm outputs:

- **Discarded** edges: $A_1 \subseteq G$ where $|A_1| \leq o(m)$ and A_1 depends only on G,

Our results

Theorem 2. Given a noisy k-XOR instance with $m \geq \tilde{O}(n^{k/2})$ constraints, our algorithm outputs:

- **Discarded** edges: $A_1 \subseteq G$ where $|A_1| \leq o(m)$ and A_1 depends only on G,
- **Corrupted** edges: $A_2 \subseteq G$ where $A_2 = (\text{corrupted edges}) \cap (G \backslash A_1)$.

Our results

Theorem 2. Given a noisy k-XOR instance with $m \geq \tilde{O}(n^{k/2})$ constraints, our algorithm outputs:

- **Discarded** edges: $A_1 \subseteq G$ where $|A_1| \leq o(m)$ and A_1 depends only on G,
- **Corrupted** edges: $A_2 \subseteq G$ where $A_2 =$ (corrupted edges) $\cap (G \backslash A_1)$.

Except for A_1, we identify *exactly* all corrupted constraints!

Our results

Theorem 2. Given a noisy k-XOR instance with $m \geq \tilde{O}(n^{k/2})$ constraints, our algorithm outputs:

- **Discarded** edges: $A_1 \subseteq G$ where $|A_1| \leq o(m)$ and A_1 depends only on G,
- **Corrupted** edges: $A_2 \subseteq G$ where $A_2 =$ (corrupted edges) $\cap (G \backslash A_1)$.

Except for A_1, we identify *exactly* all corrupted constraints!

This stronger guarantee is necessary for
the reduction from semirandom k-CSPs to noisy k-XOR.

Key ideas

Key ideas

Determine the minimal condition on the constraint graph G that makes SDP uniquely identify x^*.

Key ideas

Determine the minimal condition on the constraint graph G that makes SDP uniquely identify x^*.

- A new connection to **spectral sparsification**.

Key ideas

Determine the minimal condition on the constraint graph G that makes SDP uniquely identify x^*.

- A new connection to **spectral sparsification**.

Decomposition that breaks up every instance into pieces satisfying the above condition.

Proof Idea:

Noisy 2-XOR

Noisy 2-XOR

Noisy 2-XOR

Arbitrary graph G and $x^* \in \{\pm 1\}^n$, constraints $x_i x_j = x_i^* x_j^*$ w.p. $1 - \eta$ and $x_i x_j = -x_i^* x_j^*$ w.p. η for each edge $(i, j) \in E$.

Noisy 2-XOR

Arbitrary graph G and $x^* \in \{\pm 1\}^n$, constraints $x_i x_j = x_i^* x_j^*$ w.p. $1 - \eta$ and $x_i x_j = - x_i^* x_j^*$ w.p. η for each edge $(i, j) \in E$.

Question: when can we recover x^* exactly (up to sign)?

Noisy 2-XOR

Arbitrary graph G and $x^* \in \{\pm 1\}^n$, constraints $x_i x_j = x_i^* x_j^*$ w.p. $1 - \eta$ and $x_i x_j = -x_i^* x_j^*$ w.p. η for each edge $(i, j) \in E$.

Question: when can we recover x^* exactly (up to sign)?

Observation. Let H be the corrupted edges. If for each $S \subseteq [n]$,

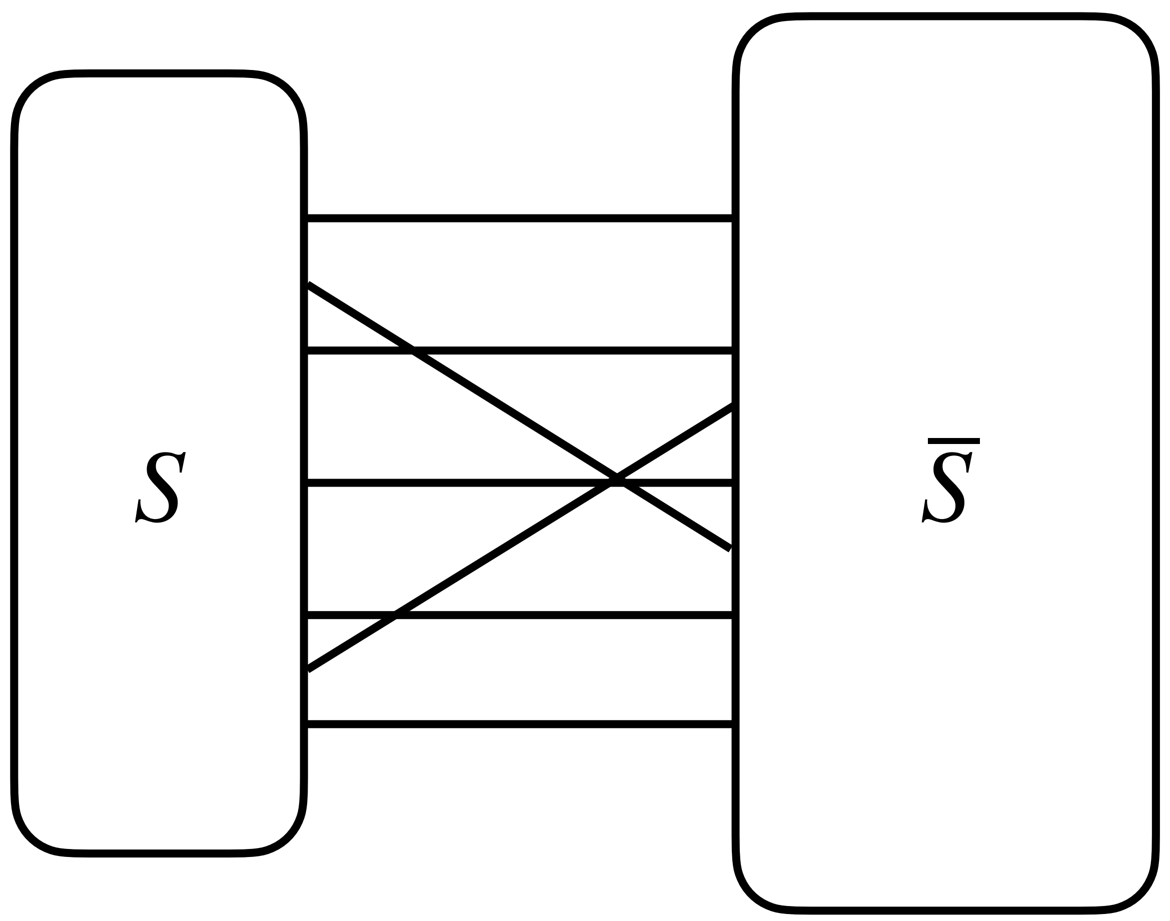
Noisy 2-XOR

Arbitrary graph G and $x^* \in \{\pm 1\}^n$, constraints $x_i x_j = x_i^* x_j^*$ w.p. $1 - \eta$ and $x_i x_j = -x_i^* x_j^*$ w.p. η for each edge $(i, j) \in E$.

Question: when can we recover x^* exactly (up to sign)?

Observation. Let H be the corrupted edges. If for each $S \subseteq [n]$,

$$|E_H(S, \bar{S})| < \frac{1}{2} \cdot |E_G(S, \bar{S})|$$



Noisy 2-XOR

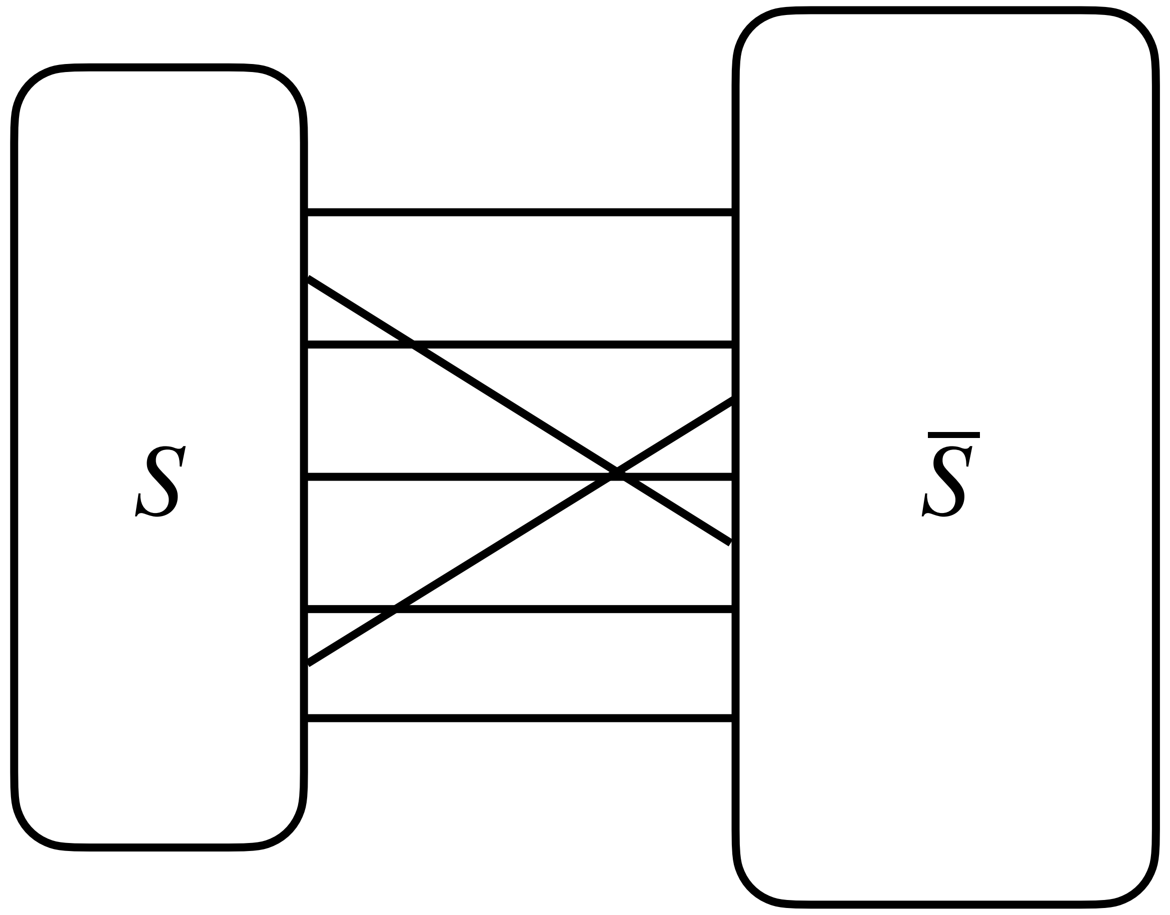
Arbitrary graph G and $x^* \in \{\pm 1\}^n$, constraints $x_i x_j = x_i^* x_j^*$ w.p. $1 - \eta$ and $x_i x_j = -x_i^* x_j^*$ w.p. η for each edge $(i,j) \in E$.

Question: when can we recover x^* exactly (up to sign)?

Observation. Let H be the corrupted edges. If for each $S \subseteq [n]$,

$$|E_H(S, \bar{S})| < \frac{1}{2} \cdot |E_G(S, \bar{S})|$$

then x^* is the unique optimal assignment (up to sign).



Noisy 2-XOR

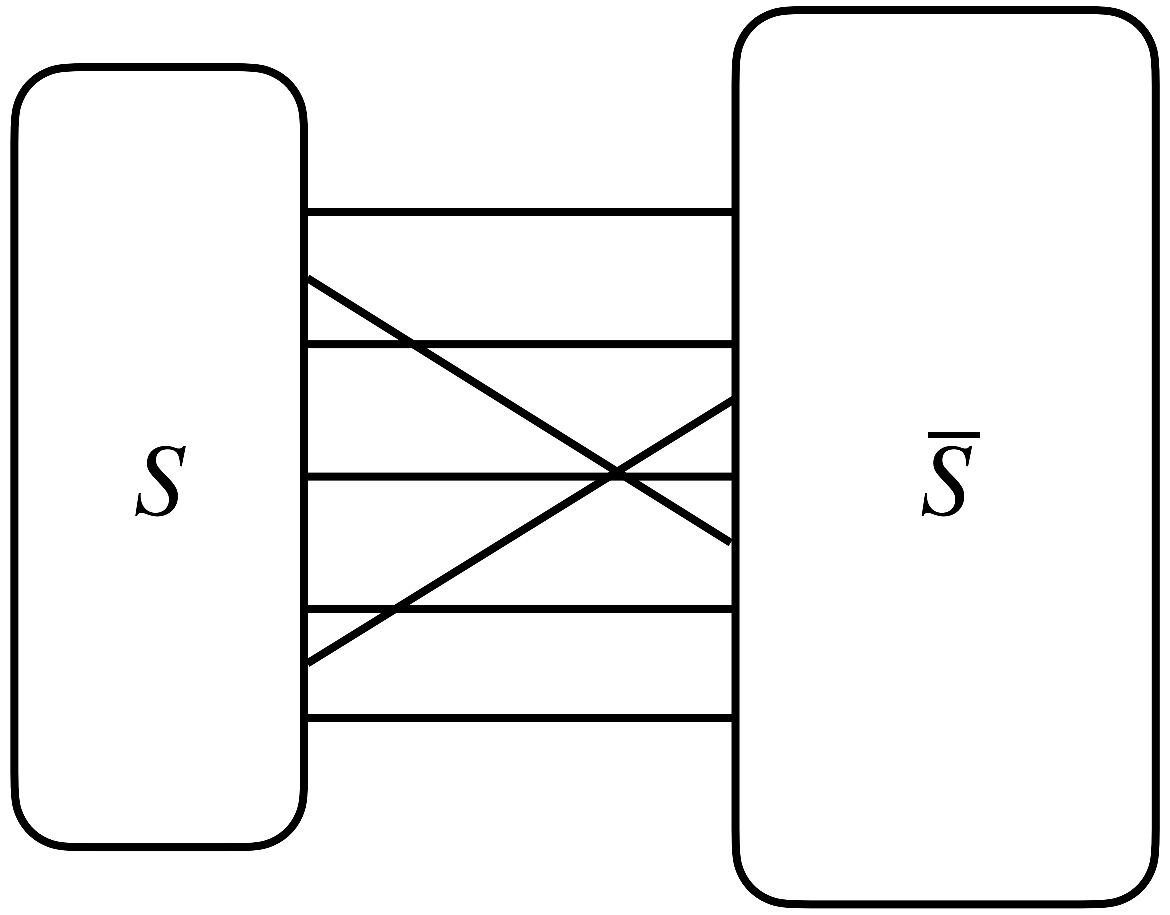
Arbitrary graph G and $x^* \in \{\pm 1\}^n$, constraints $x_i x_j = x_i^* x_j^*$ w.p. $1 - \eta$ and $x_i x_j = -x_i^* x_j^*$ w.p. η for each edge $(i, j) \in E$.

Question: when can we recover x^* exactly (up to sign)?

Observation. Let H be the corrupted edges. If for each $S \subseteq [n]$,

$$|E_H(S, \bar{S})| < \frac{1}{2} \cdot |E_G(S, \bar{S})|$$

then x^* is the unique optimal assignment (up to sign).



$S = \{i : x_i \neq x_i^*\}$

Noisy 2-XOR

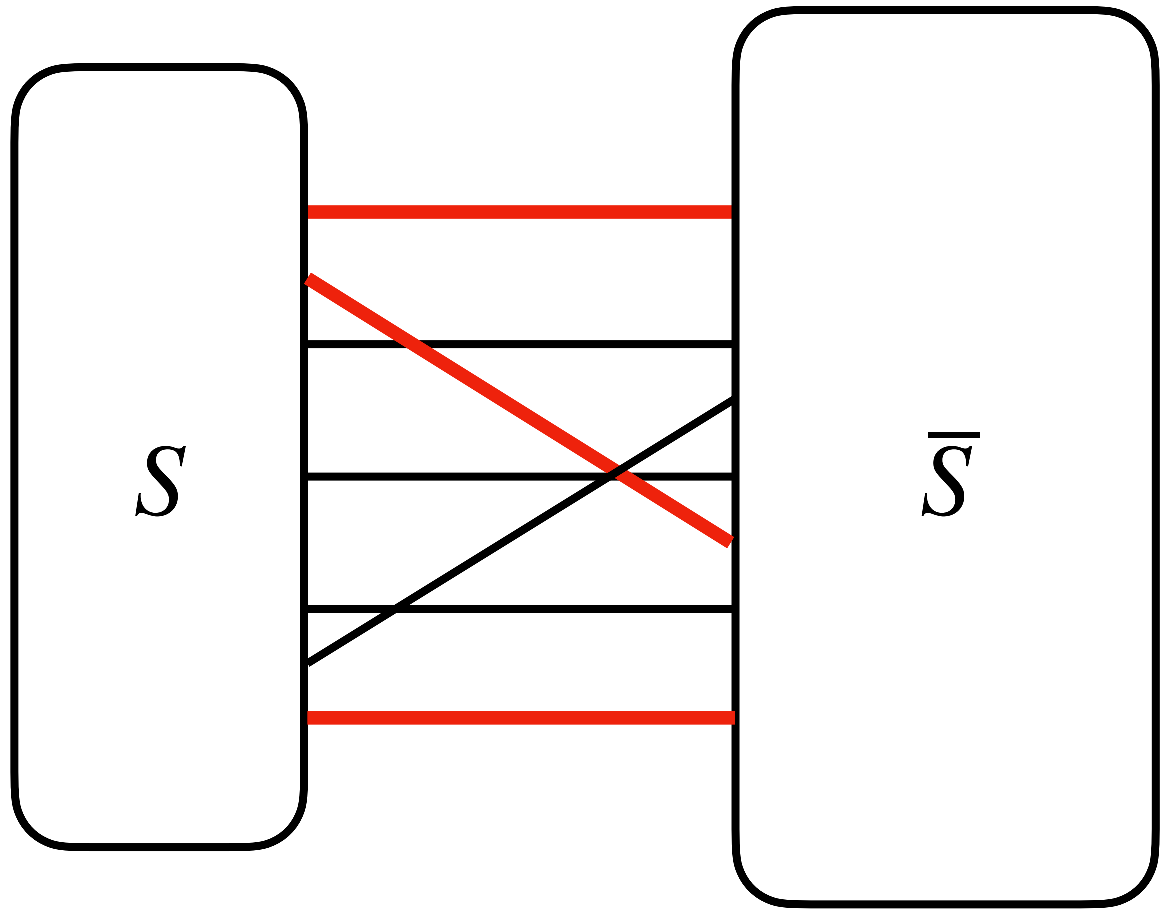
Arbitrary graph G and $x^* \in \{\pm 1\}^n$, constraints $x_i x_j = x_i^* x_j^*$ w.p. $1 - \eta$ and $x_i x_j = - x_i^* x_j^*$ w.p. η for each edge $(i, j) \in E$.

Question: when can we recover x^* exactly (up to sign)?

Observation. Let H be the corrupted edges. If for each $S \subseteq [n]$,

$$|E_H(S, \bar{S})| < \frac{1}{2} \cdot |E_G(S, \bar{S})|$$

then x^* is the unique optimal assignment (up to sign).



$S = \{i : x_i \neq x_i^*\}$

Noisy 2-XOR

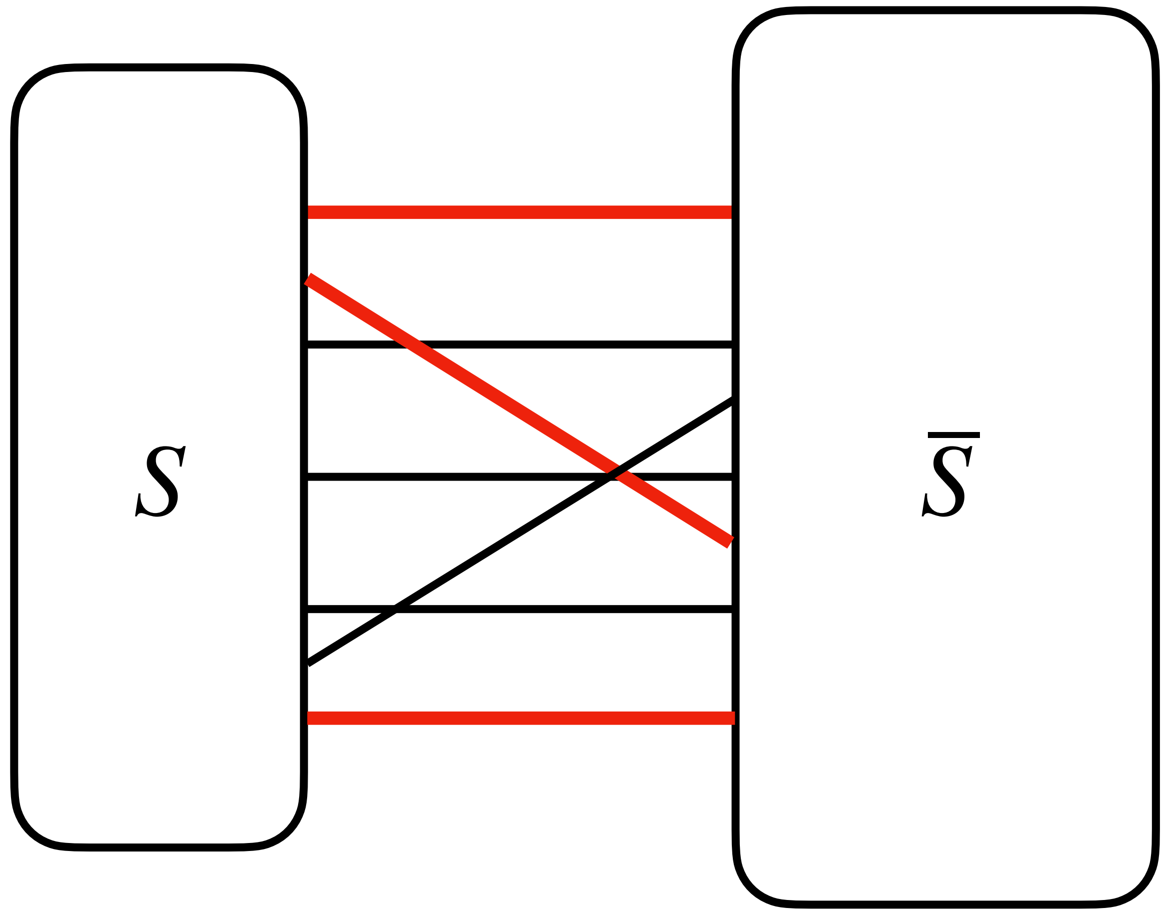
Arbitrary graph G and $x^* \in \{\pm 1\}^n$, constraints $x_i x_j = x_i^* x_j^*$ w.p. $1 - \eta$ and $x_i x_j = -x_i^* x_j^*$ w.p. η for each edge $(i, j) \in E$.

Question: when can we recover x^* exactly (up to sign)?

Observation. Let H be the corrupted edges. If for each $S \subseteq [n]$,

$$|E_H(S, \bar{S})| < \frac{1}{2} \cdot |E_G(S, \bar{S})|$$

then x^* is the unique optimal assignment (up to sign).



$S = \{i : x_i \neq x_i^*\}$

Cut sparsification!

Cut sparsification

Lemma [Karger'94]. Suppose G has min-cut $\geq \omega(\log n)$, then H is a cut sparsifier of G (w.h.p.), meaning

Cut sparsification

Lemma [Karger'94]. Suppose G has min-cut $\geq \omega(\log n)$, then H is a cut sparsifier of G (w.h.p.), meaning

$$x^\top L_H x = (1 \pm o(1)) \cdot \eta \cdot x^\top L_G x, \text{ for all } x \in \{\pm 1\}^n.$$

Cut sparsification

Lemma [Karger'94]. Suppose G has min-cut $\geq \omega(\log n)$, then H is a cut sparsifier of G (w.h.p.), meaning

$$x^\top L_H x = (1 \pm o(1)) \cdot \eta \cdot x^\top L_G x, \ \text{ for all } x \in \{\pm 1\}^n.$$

Here L_H and L_G are the Laplacian matrices.

Cut sparsification

Lemma [Karger'94]. Suppose G has min-cut $\geq \omega(\log n)$, then H is a cut sparsifier of G (w.h.p.), meaning

$$x^\top L_H x = (1 \pm o(1)) \cdot \eta \cdot x^\top L_G x, \text{ for all } x \in \{\pm 1\}^n.$$

Here L_H and L_G are the Laplacian matrices.

Large min-cut $\Longrightarrow x^*$ is the unique optimum.

Cut sparsification

Lemma [Karger'94]. Suppose G has min-cut $\geq \omega(\log n)$, then H is a cut sparsifier of G (w.h.p.), meaning

$$x^\top L_H x = (1 \pm o(1)) \cdot \eta \cdot x^\top L_G x, \text{ for all } x \in \{\pm 1\}^n.$$

Here L_H and L_G are the Laplacian matrices.

Large min-cut $\implies x^*$ is the unique optimum.

Can we recover it efficiently?

Efficient recovery

Semidefinite program (SDP) relaxation:

Efficient recovery

Semidefinite program (SDP) relaxation:

$$\max \sum_{(i,j)\in E(G)} b_{ij} X_{ij}$$

Efficient recovery

Semidefinite program (SDP) relaxation:

$$\max \sum_{(i,j)\in E(G)} b_{ij} X_{ij}$$

$$\text{s.t. } X \succeq 0, \ \text{diag}(X) = I.$$

Efficient recovery

Semidefinite program (SDP) relaxation:

$$\max \sum_{(i,j)\in E(G)} b_{ij} X_{ij}$$

$$\text{s.t. } X \succeq 0,\ \operatorname{diag}(X) = I.$$

Question: Does large min-cut $\Longrightarrow X = x^* x^{*\top}$ is the optimal SDP solution?

Efficient recovery

Semidefinite program (SDP) relaxation:

$$\max \sum_{(i,j)\in E(G)} b_{ij} X_{ij}$$

$$\text{s.t. } X \succeq 0,\ \operatorname{diag}(X) = I.$$

Question: Does large min-cut $\Longrightarrow X = x^* x^{*\top}$ is the optimal SDP solution?

Can solve it efficiently.

Efficient recovery

Semidefinite program (SDP) relaxation:

$$\max \sum_{(i,j)\in E(G)} b_{ij} X_{ij}$$

$$\text{s.t. } X \succeq 0,\ \operatorname{diag}(X) = I.$$

Question: Does large min-cut $\Longrightarrow X = x^* x^{*\top}$ is the optimal SDP solution?

No!

Efficient recovery

Semidefinite program (SDP) relaxation:

$$\max \sum_{(i,j)\in E(G)} b_{ij} X_{ij}$$

$$\text{s.t. } X \succeq 0,\ \operatorname{diag}(X) = I.$$

Question: Does large min-cut $\Longrightarrow X = x^* x^{*\top}$ is the optimal SDP solution?

No!

Need additional assumptions.

Spectral sparsification

Spectral sparsification

Lemma. $L_H \prec \frac{1}{2} L_G \implies X = x^* x^{*\top}$ is the unique optimal SDP solution.

Spectral sparsification

Lemma. $L_H \prec \frac{1}{2} L_G \implies X = x^* x^{*\top}$ is the unique optimal SDP solution.

Lemma. Suppose G has spectral gap λ and min-degree d such that $\lambda d \geq \omega(\log n)$, then H is a spectral sparsifier of G (w.h.p.), meaning

Spectral sparsification

Lemma. $L_H \prec \frac{1}{2} L_G \implies X = x^* x^{*\top}$ is the unique optimal SDP solution.

Lemma. Suppose G has spectral gap λ and min-degree d such that $\lambda d \geq \omega(\log n)$, then H is a spectral sparsifier of G (w.h.p.), meaning

$$L_H \preceq (1 + o(1)) \cdot \eta \cdot L_G.$$

Spectral sparsification

Lemma. $L_H \prec \frac{1}{2} L_G \implies X = x^* x^{*\top}$ is the unique optimal SDP solution.

Lemma. Suppose G has spectral gap λ and min-degree d such that $\lambda d \geq \omega(\log n)$, then H is a spectral sparsifier of G (w.h.p.), meaning

$$L_H \preceq (1 + o(1)) \cdot \eta \cdot L_G.$$

Large spectral gap + min degree $\implies x^* x^{*\top}$ is the unique SDP optimum.

Algorithm for noisy 2-XOR

Algorithm for noisy 2-XOR

Given a noisy 2-XOR instance on graph G,

Algorithm for noisy 2-XOR

Given a noisy 2-XOR instance on graph G,

- Expander decomposition: discard $o(1)$ fraction of edges.

Algorithm for noisy 2-XOR

Given a noisy 2-XOR instance on graph G,

- Expander decomposition: discard $o(1)$ fraction of edges.
- Run SDP on each **expanding** sub-instance,

Algorithm for noisy 2-XOR

Given a noisy 2-XOR instance on graph G,

- Expander decomposition: discard $o(1)$ fraction of edges.
- Run SDP on each **expanding** sub-instance,
  - recover x^* and identify corrupted edges in each sub-instance.

Algorithm for noisy k-XOR

Algorithm for noisy k-XOR

For **even** k, we can reduce to 2-XOR.

Algorithm for noisy k-XOR

For **even** k, we can reduce to 2-XOR.

For **odd** k, we can still reduce to 2-XOR but the corruption is no longer independent for each edge.

Algorithm for noisy k-XOR

For **even** k, we can reduce to 2-XOR.

For **odd** k, we can still reduce to 2-XOR but the corruption is no longer independent for each edge.

$\longrightarrow$ a generalized version of spectral sparsification.

Conclusion

Main open question:

Conclusion

Main open question:

Sub-exponential time algorithms when $m = O(n^{k/2-\epsilon})$ constraints.

Conclusion

Main open question:

Sub-exponential time algorithms when $m = O(n^{k/2-\epsilon})$ constraints.

- Can we do expander decomposition implicitly, i.e., round a single SDP relaxation?

Conclusion

Main open question:

Sub-exponential time algorithms when $m = O(n^{k/2-\epsilon})$ constraints.

- Can we do expander decomposition implicitly, i.e., round a single SDP relaxation?
- Can some log factors be removed (e.g., log factors from matrix Chernoff)?

Conclusion

Main open question:

Sub-exponential time algorithms when $m = O(n^{k/2-\epsilon})$ constraints.

- Can we do expander decomposition implicitly, i.e., round a single SDP relaxation?
- Can some log factors be removed (e.g., log factors from matrix Chernoff)?

Thank you!

https://arxiv.org/abs/2309.16897