New Arenas in
Hardness Amplification

Karthik C. S.
(Weizmann Institute of Science)

Joint work with

Elazar Goldenberg
(The Academic College of Tel Aviv-Yaffo)
Necessity is the Mother of Invention
Average Case Complexity
Average Case Complexity
  - Hardness of Problems in Practice
Necessity is the Mother of Invention

- Average Case Complexity
  - Hardness of Problems in Practice

- Modern Cryptography
Necessity is the Mother of Invention

- Average Case Complexity
  - Hardness of Problems in Practice

- Modern Cryptography
  - Hard on average function in NP
Necessity is the Mother of Invention

- Average Case Complexity
  - Hardness of Problems in Practice

- Modern Cryptography
  - Hard on average function in NP

Modest Goal:
Necessity is the Mother of Invention

- Average Case Complexity
  - Hardness of Problems in Practice

- Modern Cryptography
  - Hard on average function in NP

Modest Goal: Hardness Amplification
Necessity is the Mother of Invention

- Average Case Complexity
  - Hardness of Problems in Practice

- Modern Cryptography
  - Hard on average function in NP

Modest Goal: Hardness Amplification
Mild average case ⇒ Sharp average case
Family of functions \( \{ f_n \}_{n \in \mathbb{N}} \)
Family of functions $\{f_n\}_{n \in \mathbb{N}}$

Every algorithm $\mathcal{A}$ running in time $t(n)$, fails on $p(n)$ fraction of inputs
Family of functions \( \{ f_n \}_{n \in \mathbb{N}} \)

Every algorithm \( \mathcal{A} \) running in time \( t(n) \),
fails on \( p(n) \) fraction of inputs
The Utopic Theorem of Hardness Amplification

- Family of functions \( \{ f_n \}_{n \in \mathbb{N}} \)

- Every algorithm \( \mathcal{A} \) running in time \( t(n) \), fails on \( p(n) \) fraction of inputs

\( \Downarrow \)

- Family of functions \( \{ g_n \}_{n \in \mathbb{N}} \)
The Utopic Theorem of Hardness Amplification

- Family of functions \( \{f_n\}_{n \in \mathbb{N}} \)

- Every algorithm \( \mathcal{A} \) running in time \( t(n) \), fails on \( p(n) \) fraction of inputs

\[ \Downarrow \]

- Family of functions \( \{g_n\}_{n \in \mathbb{N}} \)

- Every algorithm \( \mathcal{A}' \) running in time \( t'(n) \), fails on \( p'(n) \) fraction of inputs
Family of functions \( \{ f_n \}_{n \in \mathbb{N}} \)

Every algorithm \( \mathcal{A} \) running in time \( t(n) \), fails on \( p(n) \) fraction of inputs

\[ \Rightarrow \]

Family of functions \( \{ g_n \}_{n \in \mathbb{N}} \)

Every algorithm \( \mathcal{A}' \) running in time \( t'(n) \), fails on \( p'(n) \) fraction of inputs

\[ p'(n) \gg p(n) \]
The Utopic Theorem of Hardness Amplification

- Family of functions \{f_n\}_{n \in \mathbb{N}}

- Every algorithm \mathcal{A} running in time \(t(n)\), fails on \(p(n)\) fraction of inputs

- Family of functions \{g_n\}_{n \in \mathbb{N}}

- Every algorithm \(\mathcal{A}'\) running in time \(t'(n)\), fails on \(p'(n)\) fraction of inputs

- \(p'(n) \gg p(n)\)
- \(f_n = g_n\)
The Utopic Theorem of Hardness Amplification

- Family of functions $\{f_n\}_{n \in \mathbb{N}}$
- Every algorithm $A$ running in time $t(n)$, fails on $p(n)$ fraction of inputs

$\Downarrow$

- Family of functions $\{g_n\}_{n \in \mathbb{N}}$
- Every algorithm $A'$ running in time $t'(n)$, fails on $p'(n)$ fraction of inputs

- $p'(n) \gg p(n)$
- $f_n = g_n$
- $f$ is “interesting”
Can we do hardness amplification
Can we do hardness amplification for problems
The Big Question

Can we do hardness amplification for problems we care about and
The Big Question

Can we do hardness amplification for problems we care about and we believe are hard on average?
The Story so far

- **#P (Lipton’89):**
#P (Lipton’89): If Permanent can be computed:

- deterministically in polynomial time
- on $1/2$ the matrices
The Story so far

- **#P** (Lipton’89): If Permanent can be computed:
  - deterministically in *polynomial* time
  - on $1/2$ the matrices

then $\text{Permanent}$ is in BPP.
The Story so far

- **#P (Lipton’89):** If Permanent can be computed:
  - deterministically in polynomial time
  - on 1/2 the matrices

  then \textit{Permanent} is in BPP.

- **EXP (Trevisan-Vadhan’07):**
The Story so far

- **#P (Lipton’89):** If Permanent can be computed:
  - deterministically in polynomial time
  - on $1/2$ the matrices
Then Permanent is in BPP.

- **EXP (Trevisan-Vadhan’07):** If $\exists \Pi \in \text{EXP}$:
  - cannot be efficiently solved in the worst case by
  - uniform probabilistic algorithms
The Story so far

- **#P (Lipton’89):** If Permanent can be computed:
  - deterministically in **polynomial** time
  - on $1/2$ the matrices

  then $\text{Permanent}$ is in $\text{BPP}$.

- **EXP (Trevisan-Vadhan’07):** If $\exists \Pi \in \text{EXP}$:
  - cannot be efficiently solved in the **worst case** by
  - uniform probabilistic algorithms

  then $\exists \Lambda \in \text{EXP}$:
  - cannot be efficiently solved on **random** instances
  - noticeably better than guessing the answer at **random**.
What about NP?

Non-uniform case (Healy-Vadhan-Viola’04):

- If $\exists f \in \text{NP} \circ \text{circuits of size } s(n)$ fails to compute $f \circ$ on $\frac{1}{\text{poly}(n)}$ fraction of inputs, then $\exists f' \in \text{NP} \circ \text{circuits of size } s'(n)$ fails to compute $f' \circ$ on $\frac{1}{2} - \frac{1}{s'(n)}$ fraction of inputs.

Uniform case (Trevisan’05):

- If every problem in $\text{NP} \circ$ admits an efficient uniform algorithm $\circ$ succeeds with probability at least $\frac{1}{2} + \frac{1}{(\log n) O(1)}$ then for every problem in $\text{NP} \circ$ there is an efficient uniform algorithm $\circ$ succeeds with probability at least $1 - \frac{1}{\text{poly}(n)}$. 

What about NP?

- **Non-uniform** case (Healy-Vadhan-Viola’04): If $\exists f$ in NP
  - circuits of size $s(n)$ fails to compute $f$
  - on $1/poly(n)$ fraction of inputs,
What about NP?

○ **Non-uniform** case (Healy-Vadhan-Viola’04): If ∃f in NP
  ○ circuits of size $s(n)$ fails to compute $f$
  ○ on $1/poly(n)$ fraction of inputs,

then ∃$f'$ in NP

○ circuits of size $s'(n) = s(\sqrt{n})^{\Omega(1)}$ fails to compute $f'$
○ on $1/2 - 1/s'(n)$ fraction of inputs.
What about NP?

- **Non-uniform** case (Healy-Vadhan-Viola’04): If $\exists f$ in NP
  - circuits of size $s(n)$ fails to compute $f$
  - on $1/\text{poly}(n)$ fraction of inputs,

  then $\exists f'$ in NP
  - circuits of size $s'(n) = s(\sqrt{n})^{\Omega(1)}$ fails to compute $f'$
  - on $1/2 - 1/s'(n)$ fraction of inputs.

- **Uniform** case (Trevisan’05):
What about NP?

- **Non-uniform** case (Healy-Vadhan-Viola’04): If \( \exists f \) in NP
  - circuits of size \( s(n) \) fails to compute \( f \)
  - on \( \frac{1}{\text{poly}(n)} \) fraction of inputs,
  then \( \exists f' \) in NP
    - circuits of size \( s'(n) = s(\sqrt{n})^{\Omega(1)} \) fails to compute \( f' \)
    - on \( \frac{1}{2} - \frac{1}{s'(n)} \) fraction of inputs.

- **Uniform** case (Trevisan’05): If every problem in NP
  - admits an efficient **uniform** algorithm
  - succeeds with probability at least \( \frac{1}{2} + \frac{1}{(\log n)^{O(1)}} \)
What about NP?

- **Non-uniform** case (Healy-Vadhan-Viola’04): If ∃f in NP
  - circuits of size \( s(n) \) fails to compute \( f \)
  - on \( 1 / \text{poly}(n) \) fraction of inputs,
  then ∃\( f' \) in NP
    - circuits of size \( s'(n) = s(\sqrt{n})^{O(1)} \) fails to compute \( f' \)
    - on \( 1/2 - 1/s'(n) \) fraction of inputs.

- **Uniform** case (Trevisan’05): If every problem in NP
  - admits an efficient uniform algorithm
    - succeeds with probability at least \( 1/2 + 1/(\log n)^{O(1)} \)
  then for every problem in NP
    - there is an efficient uniform algorithm
    - succeeds with probability at least \( 1 - 1/\text{poly}(n) \)
The Verona, Pompeii, Flavian, and Fiesole arenas may not be as well known as the Colosseum, but are just as impressive.

— Roman history trivia
Arenas in Hardness Amplification

NP
Arenas in Hardness Amplification
Arenas in Hardness Amplification

\[ \text{NP} \]
\[ \text{EXP} \]
\[ \#P \]
\[ \text{P} \]
Arenas in Hardness Amplification

- #P
- EXP
- NP
- P

Optimization Problems
Focus of this Talk

Optimization Problems
Focus of this Talk

Optimization Problems

- NP-hard problems
Focus of this Talk

Optimization Problems

- **NP-hard** problems
- **Subquadratic-hard** problems
Focus of this Talk

Optimization Problems

- **NP-hard** problems
- **Subquadratic-hard** problems
- **Total** Problems
Maximum Clique

Input: A graph $G$
Output: Clique of maximum size in $G$
Input: A graph $G$
Maximum Clique

Input: A graph $G$

Output: Clique of maximum size in $G$
Maximum Clique

Input: A graph $G$

Output: Clique of maximum size in $G$
Maximum Clique

Input: A graph $G$

Output: Clique of maximum size in $G$
Let $\mathcal{D}$ be $\text{poly}(n)$ time samplable distribution over graphs on $n$ vertices.
Our Result for Maximum Clique

Theorem (Goldenberg-K’19)

Let \( \mathcal{D} \) be \( \text{poly}(n) \) time samplable distribution over graphs on \( n \) vertices such that for every randomized algorithm \( A \) running in time \( \text{poly}(n) \),

\[
\Pr_{G \sim \mathcal{D}}[A \text{ finds max-clique in } G] \geq \frac{2}{3} 
\]

\[
\leq 1 - \frac{1}{n}.
\]
Our Result for Maximum Clique

Theorem (Goldenberg-K’19)

Let $\mathcal{D}$ be $\text{poly}(n)$ time samplable distribution over graphs on $n$ vertices such that for every randomized algorithm $\mathcal{A}$ running in time $\text{poly}(n)$, we have:

$$\Pr_{G \sim \mathcal{D}} \left[ \mathcal{A} \text{ finds max-clique in } G \text{ w.p. } \geq \frac{2}{3} \right] \leq 1 - \frac{1}{n}.$$
Our Result for Maximum Clique

**Theorem (Goldenberg-K’19)**

Let $\mathcal{D}$ be $\text{poly}(n)$ time samplable distribution over graphs on $n$ vertices such that for every randomized algorithm $A$ running in time $\text{poly}(n)$, we have:

$$\Pr_{G \sim \mathcal{D}} \left[ A \text{ finds max-clique in } G \text{ w.p. } \geq \frac{2}{3} \right] \leq 1 - \frac{1}{n}.$$ 

Then there is $\mathcal{D}'$ a $\text{poly}(n)$ time samplable distribution over graphs on $\text{poly}(n)$ vertices
Our Result for Maximum Clique

Theorem (Goldenberg-K’19)

Let $\mathcal{D}$ be $\text{poly}(n)$ time samplable distribution over graphs on $n$ vertices such that for every randomized algorithm $\mathcal{A}$ running in time $\text{poly}(n)$, we have:

$$\Pr_{G \sim \mathcal{D}} [\mathcal{A} \text{ finds max-clique in } G \text{ w.p. } \geq 2/3] \leq 1 - 1/n.$$ 

Then there is $\mathcal{D}'$ a $\text{poly}(n)$ time samplable distribution over graphs on $\text{poly}(n)$ vertices such that for every randomized algorithm $\mathcal{A}'$ running in time $\text{poly}(n)$,
Our Result for Maximum Clique

Theorem (Goldenberg-K’19)

Let $\mathcal{D}$ be $\text{poly}(n)$ time samplable distribution over graphs on $n$ vertices such that for every randomized algorithm $\mathcal{A}$ running in time $\text{poly}(n)$, we have:

$$\Pr_{G \sim \mathcal{D}} [\mathcal{A} \text{ finds max-clique in } G \text{ w.p. } \geq \frac{2}{3}] \leq 1 - \frac{1}{n}.$$ 

Then there is $\mathcal{D}'$ a $\text{poly}(n)$ time samplable distribution over graphs on $\text{poly}(n)$ vertices such that for every randomized algorithm $\mathcal{A}'$ running in time $\text{poly}(n)$, we have:

$$\Pr_{G' \sim \mathcal{D}'} [\mathcal{A}' \text{ finds max-clique in } G' \text{ w.p. } \geq \frac{2}{3}] \leq 0.01.$$
Proof Overview

1. Define new distribution $D'$
2. Given $A'$ for $D'$ design $A$
3. Argue that if $A'$ is correct on a fraction of inputs then $A$ is correct on $1 - \frac{1}{n}$ fraction of inputs
1. Define **new** distribution $\mathcal{D}'$
Proof Overview

1. Define new distribution $\mathcal{D}'$

2. Given $\mathcal{A}'$ for $\mathcal{D}'$ design $\mathcal{A}$ for $\mathcal{D}$
1. Define new distribution $\mathcal{D}'$

2. Given $\mathcal{A}'$ for $\mathcal{D}'$ design $\mathcal{A}$ for $\mathcal{D}$

3. Argue that if $\mathcal{A}'$ is correct on $0.01$ fraction of inputs then $\mathcal{A}$ is correct on $1 - 1/n$ fraction of inputs
$\mathcal{D}'$ samples a graph $H$ as follows:
\[ \mathcal{D} \] samples a graph \( H \) as follows:

1. **Independently** sample \( G_1, \ldots, G_k \) from \( \mathcal{D} \) (\( k := \text{poly}(n) \))
\( \mathcal{D}' \) samples a graph \( H \) as follows:

1. Independently sample \( G_1, \ldots, G_k \) from \( \mathcal{D} \) (\( k := \text{poly}(n) \))

2. Define \( H := G_1 \cup \cdots \cup G_k \)
\( \mathcal{D}' \) samples a graph \( H \) as follows:

1. **Independently** sample \( G_1, \ldots, G_k \) from \( \mathcal{D} \) (\( k := \text{poly}(n) \))

2. Define \( H := G_1 \cup \cdots \cup G_k \)

3. For every \( i \neq j \) insert every edge between \( G_i \) and \( G_j \)
$\mathcal{D}'$ samples a graph $H$ as follows:

1. Independently sample $G_1, \ldots, G_k$ from $\mathcal{D}$ ($k := \text{poly}(n)$)
2. Define $H := G_1 \cup \cdots \cup G_k$
3. For every $i \neq j$ insert every edge between $G_i$ and $G_j$
4. Output $H$
\[\mathcal{D}' \text{ samples} \] a graph \(H\) as follows:

1. \underline{Independently} sample \(G_1, \ldots, G_k\) from \(\mathcal{D}\) \((k := \text{poly}(n))\)

2. Define \(H := G_1 \cup \cdots \cup G_k\)

3. For every \(i \neq j\) insert every edge between \(G_i\) and \(G_j\)

4. \underline{Output} \(H\)

\underline{Sampling time: \text{poly}(n)}
Algorithm for Original Distribution

Input: A graph \( G \) sampled from \( D \)

Output: A maximum clique in \( G \)

1. Set \( S \) to be empty.

2. Repeat following \( O(k) \) times.

3. Pick randomly \( i \in [k] \)

4. Independently sample \( G_1, \ldots, G_{i-1}, G_{i+1}, \ldots, G_k \) from \( D \)

5. Construct \( H \) setting \( G_i \) to be \( G \)

6. Find clique in \( H \) using \( A' \)

7. Restrict clique in \( H \) to \( G \) and add to \( S \)

3. Output the largest clique in \( S \)

1/3
Algorithm for Original Distribution

Algorithm \( \mathcal{A} \)

Input: A graph \( G \) sampled from \( \mathcal{D} \)

Output: A maximum clique in \( G \)
Algorithm for Original Distribution

Algorithm $A$

Input: A graph $G$ sampled from $\mathcal{D}$
Output: A maximum clique in $G$

1. Set Solution to be empty.
Algorithm for Original Distribution

Algorithm $\mathcal{A}$

Input: A graph $G$ sampled from $\mathcal{D}$
Output: A maximum clique in $G$

1. Set Solution to be empty.
2. Repeat following $O(1)$ times.
Algorithm for Original Distribution

Algorithm $\mathcal{A}$

Input: A graph $G$ sampled from $\mathcal{D}$
Output: A maximum clique in $G$

1. Set Solution to be empty.
2. Repeat following $O(1)$ times.
   2.1 Pick randomly $i \in [k]$
Algorithm for Original Distribution

Algorithm $\mathcal{A}$

Input: A graph $G$ sampled from $\mathcal{D}$
Output: A maximum clique in $G$

1. Set Solution to be empty.
2. Repeat following $O(1)$ times.
   2.1 Pick randomly $i \in [k]$
   2.2 Independently sample $G_1, \ldots, G_{i-1}, G_{i+1}, \ldots G_k$ from $\mathcal{D}$
Algorithm $A$

Input: A graph $G$ sampled from $\mathcal{D}$

Output: A maximum clique in $G$

1. Set Solution to be empty.

2. Repeat following $O(1)$ times.
   2.1 Pick randomly $i \in [k]$  
   2.2 Independently sample $G_1, \ldots, G_{i-1}, G_{i+1}, \ldots G_k$ from $\mathcal{D}$  
   2.3 Construct $H$ setting $G_i$ to be $G$
Algorithm for Original Distribution

Algorithm $\mathcal{A}$

Input: A graph $G$ sampled from $D$
Output: A maximum clique in $G$

1. Set Solution to be empty.
2. Repeat following $O(1)$ times.
   2.1 Pick randomly $i \in [k]$
   2.2 Independently sample $G_1, \ldots, G_{i-1}, G_{i+1}, \ldots, G_k$ from $D$
   2.3 Construct $H$ setting $G_i$ to be $G$
   2.4 Find clique in $H$ using $\mathcal{A}'$
Algorithm for Original Distribution

Algorithm \( \mathcal{A} \)

Input: A graph \( G \) sampled from \( \mathcal{D} \)
Output: A maximum clique in \( G \)

1. Set Solution to be empty.
2. Repeat following \( O(1) \) times.
   2.1 Pick randomly \( i \in [k] \)
   2.2 Independently sample \( G_1, \ldots, G_{i-1}, G_{i+1}, \ldots G_k \) from \( \mathcal{D} \)
   2.3 Construct \( H \) setting \( G_i \) to be \( G \)
   2.4 Find clique in \( H \) using \( \mathcal{A}' \)
   2.5 Restrict clique in \( H \) to \( G \) and add to Solution
Algorithm for Original Distribution

Algorithm \( \mathcal{A} \)

Input: A graph \( G \) sampled from \( \mathcal{D} \)
Output: A maximum clique in \( G \)

1. Set Solution to be empty.
2. Repeat following \( O(1) \) times.
   2.1 Pick randomly \( i \in [k] \)
   2.2 \textbf{Independently} sample \( G_1, \ldots, G_{i-1}, G_{i+1}, \ldots G_k \) from \( \mathcal{D} \)
   2.3 Construct \( H \) setting \( G_i \) to be \( G \)
   2.4 Find clique in \( H \) using \( \mathcal{A}' \)
   2.5 \textbf{Restrict} clique in \( H \) to \( G \) and add to Solution
3. Output the \textbf{largest} clique in Solution
Claim

If $S$ is a maximum clique of $H$ then for any $i \in [k]$ its restriction to vertices of $G_i$ gives a maximum clique of $G_i$. 
A_0$ be one iteration of Step 2 of $\mathcal{A}$
Correctness of Algorithm

- $A_0$ be one iteration of Step 2 of $A$

- If $A_0$ outputs maximum clique w.p. $\varepsilon$ on $1 - 1/n$ fraction of samples from $D$ then,
Correctness of Algorithm

- $A_0$ be one iteration of Step 2 of $A$

- If $A_0$ outputs maximum clique w.p. $\epsilon$ on $1 - 1/n$ fraction of samples from $D$ then,
  $A$ outputs maximum clique w.p. $2/3$ on $1 - 1/n$ fraction of samples from $D$. 
Correctness of Algorithm

- $A_0$ be one iteration of Step 2 of $A$

- If $A_0$ outputs maximum clique w.p. $\epsilon$ on $1 - 1/n$ fraction of samples from $D$ then,
  $A$ outputs maximum clique w.p. $2/3$ on $1 - 1/n$ fraction of samples from $D$.

- Suffices to show: $A'$ outputs maximum clique in Step 2.5 w.p. $\epsilon$ on $1 - 1/n$ fraction of samples from $D$. 
A Direct Product Lemma

Lemma (Feige-Kilian'94)

Let $T$ be a distribution over $X$. Let $f : X^k \rightarrow \{0, 1\}$. Then, 
$$\Pr_{x \sim T} i \in [k] \mid \mu_i, x - \mu \geq k - 1/3 \leq k - 1/3,$$ 
where $\mu_i = \mathbb{E}_{x \sim T} f(x)$.

$f(x^k) \equiv A' \text{outputs maximum clique w.p.} \frac{2}{3}$. 

one.taboldstyle/six.taboldstyle
A Direct Product Lemma

Lemma (Feige-Kilian’94)

Let $\mathcal{T}$ be a distribution over $X$. Let $f : X^k \rightarrow \{0, 1\}$. 

---

$f(x^k) =$ TRUE $\iff$ $A'$ outputs maximum clique w.p. $2/3$. 

---
Lemma (Feige-Kilian’94)

Let $\mathcal{T}$ be a distribution over $X$. Let $f : X^k \rightarrow \{0, 1\}$. Then,

$$\Pr_{x \sim \mathcal{T}} \left[ |\mu_{i,x} - \mu| \geq k^{-1/3} \right] \leq k^{-1/3},$$
Lemma (Feige-Kilian’94)

Let $\mathcal{T}$ be a distribution over $X$. Let $f : X^k \to \{0, 1\}$. Then,

$$\Pr_{x \sim \mathcal{T}, i \in [k]} \left[ |\mu_{i,x} - \mu| \geq k^{-1/3} \right] \leq k^{-1/3},$$

where

$$\mu = \mathbb{E}_{x^k \sim \mathcal{T}^k} \left[ f \left( x^k \right) \right],$$
A Direct Product Lemma

Lemma (Feige-Kilian’94)

Let \( \mathcal{T} \) be a distribution over \( X \). Let \( f : X^k \rightarrow \{0, 1\} \). Then,

\[
\Pr_{x \sim \mathcal{T}, i \in [k]} \left[ |\mu_{i,x} - \mu| \geq k^{-1/3} \right] \leq k^{-1/3},
\]

where

\[
\mu = \mathbb{E}_{x^k \sim \mathcal{T}^k} \left[ f(x^k) \right],
\]

\[
\mu_{i,x} = \mathbb{E}_{x_1, \ldots, x_i, x_{i+1}, \ldots, x_k \sim \mathcal{T}} \left[ f(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_k) \right].
\]
A Direct Product Lemma

Lemma (Feige-Kilian’94)

Let $\mathcal{T}$ be a distribution over $X$. Let $f : X^k \rightarrow \{0, 1\}$. Then,

$$\Pr_{x \sim \mathcal{T}, i \in [k]} \left[ |\mu_{i,x} - \mu| \geq k^{-1/3} \right] \leq k^{-1/3},$$

where

$$\mu = \mathbb{E}_{x^k \sim \mathcal{T}^k} \left[ f(x^k) \right],$$

$$\mu_{i,x} = \mathbb{E}_{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k \sim \mathcal{T}} \left[ f(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_k) \right].$$

$$f(x^k) = 1 \iff \mathcal{A}' \text{ outputs maximum clique w.p. } 2/3$$
Proof Summary

- New Distribution: Direct Product of Old Distribution with solution preserving property
- Invoke Feige-Kilian lemma to show amplification of hardness
New Distribution: Direct Product of Old Distribution with solution preserving property
Proof Summary

- New Distribution: Direct Product of Old Distribution with solution preserving property
- Invoke Feige-Kilian lemma to show amplification of hardness
An optimization problem $\Pi$ is the quadruple $(I_{\Pi}, \text{Sol}_{\Pi}, \Delta_{\Pi}, \text{goal}_{\Pi})$:

- $I_{\Pi}$: set of instances of $\Pi$;
- $\text{Sol}_{\Pi}$: function from $I_{\Pi}$ to set of feasible solutions;
- $\Delta_{\Pi}$: assigns $(x \in I_{\Pi}, y \in \text{Sol}_{\Pi}(x))$ a non-negative integer;
- $\text{goal}_{\Pi} \in \{\text{min}, \text{max}\}$.
An optimization problem $\Pi$ is the quadruple $(I_\Pi, \text{Sol}_\Pi, \Delta_\Pi, \text{goal}_\Pi)$:
An optimization problem $\Pi$ is the quadruple $(I_\Pi, Sol_\Pi, \Delta_\Pi, \text{goal}_\Pi)$:

- $I_\Pi$: set of instances of $\Pi$;

A quadruple is a set of four elements, each of which is associated with a specific role or function in the context of optimization problems. The elements in this quadruple are:

1. $I_\Pi$: The set of instances, which represents the various problem settings or scenarios that the optimization problem $\Pi$ addresses.
2. $Sol_\Pi$: The solution function, which maps instances from $I_\Pi$ to feasible solutions within the problem space.
3. $\Delta_\Pi$: The perturbation function, which assigns a non-negative integer to pairs of an instance $x \in I_\Pi$ and a solution $y \in Sol_\Pi(x)$, indicating the degree of perturbation or deviation from optimality.
4. $\text{goal}_\Pi$: The optimization goal, which specifies whether the goal is to minimize or maximize some objective function. The options are $\min$ or $\max$.
An optimization problem $\Pi$ is the quadruple $(I_\Pi, S\text{ol}_\Pi, \Delta_\Pi, \text{goal}_\Pi)$:

- $I_\Pi$: set of instances of $\Pi$;
- $S\text{ol}_\Pi$: function from $I_\Pi$ to set of feasible solutions;
An optimization problem $\Pi$ is the quadruple $(I_\Pi, Sol_\Pi, \Delta_\Pi, \text{goal}_\Pi)$:

- $I_\Pi$: set of instances of $\Pi$;
- $Sol_\Pi$: function from $I_\Pi$ to set of feasible solutions;
- $\Delta_\Pi$: assigns $(x \in I_\Pi, y \in Sol_\Pi(x))$ a non-negative integer;
An optimization problem $\Pi$ is the quadruple $(I_{\Pi}, Sol_{\Pi}, \Delta_{\Pi}, goal_{\Pi})$:

- $I_{\Pi}$: set of instances of $\Pi$;
- $Sol_{\Pi}$: function from $I_{\Pi}$ to set of feasible solutions;
- $\Delta_{\Pi}$: assigns ($x \in I_{\Pi}, y \in Sol_{\Pi}(x)$) a non-negative integer;
- $goal_{\Pi} \in \{\text{min}, \text{max}\}$.
Let $S, T : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$. 
Direct Product Feasibility

Let \( S, T : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \).

We say \( \Pi(I_\Pi, \text{Sol}_\Pi, \Delta_\Pi, \text{goal}_\Pi) \) is \( (S, T) \)-direct product feasible.
Direct Product Feasibility

Let $S, T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

We say $\Pi(l_\Pi, Sol_\Pi, \Delta_\Pi, goal_\Pi)$ is $(S, T)$-direct product feasible
if there exists deterministic $(Gen, Dec)$:

\begin{align*}
\text{Gen} : & \text{Input: } x_1, \ldots, x_k \in I_\Pi(n) \\
& \text{Output: } x'_1 \in I_\Pi(S(n, k)) \\
\text{Dec} : & \text{Input: } i \in [k], x_1, \ldots, x_k \in I_\Pi(n), \text{ and optimal } y' \in Sol_\Pi(x'_1) \\
& \text{Output: optimal } y \in Sol_\Pi(x_i) \\
\end{align*}

$Gen$ and $Dec$ run in $T(n, k)$ time.
Let $S, T : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$.

We say $\Pi(l_\Pi, Sol_\Pi, \Delta_\Pi, \text{goal}_\Pi)$ is $(S, T)$-direct product feasible if there exists deterministic $(Gen, Dec)$:

- Gen:
  - Input: $x_1, \ldots, x_k \in l_\Pi(n)$
  - Output: $x' \in l_\Pi(S(n, k))$
Direct Product Feasibility

Let $S, T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

We say $\Pi(l_\Pi, \text{Sol}_\Pi, \Delta_\Pi, \text{goal}_\Pi)$ is $(S, T)$-direct product feasible if there exists deterministic $(\text{Gen}, \text{Dec})$:

- **Gen**: 
  - **Input**: $x_1, \ldots, x_k \in l_\Pi(n)$
  - **Output**: $x' \in l_\Pi(S(n, k))$

- **Dec**: 
  - **Input**: $i \in [k], x_1, \ldots, x_k \in l_\Pi(n)$, and optimal $y' \in \text{Sol}_\Pi(x')$
  - **Output**: optimal $y \in \text{Sol}_\Pi(x_i)$
Direct Product Feasibility

Let $S, T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

We say $\Pi(l_\Pi, \text{Sol}_\Pi, \Delta_\Pi, \text{goal}_\Pi)$ is $(S, T)$-direct product feasible if there exists deterministic $(\text{Gen}, \text{Dec})$:

- **Gen:**
  - **Input:** $x_1, \ldots, x_k \in l_\Pi(n)$
  - **Output:** $x' \in l_\Pi(S(n, k))$

- **Dec:**
  - **Input:** $i \in [k], x_1, \ldots, x_k \in l_\Pi(n)$, and optimal $y' \in \text{Sol}_\Pi(x')$
  - **Output:** optimal $y \in \text{Sol}_\Pi(x_i)$

- Gen and Dec run in $T(n, k)$ time.
Our General Result

Theorem (Goldenberg-K’19)

Let $\Pi$ be $(S, T)$-direct product feasible. Let $D$ be $s(n)$ time samplable distribution over $l_\Pi(n)$ such that for every randomized algorithm $A$ running in time $t(n)$, we have:

$$\Pr_{x \sim D} [A \text{ finds optimal solution of } x \text{ w.p. } \geq \frac{2}{3}] \leq 1 - \frac{1}{p(n)}.$$ 

*Conditions apply.*
Our General Result

Theorem (Goldenberg-K’19)

Let $\Pi$ be $(S, T)$-direct product feasible. Let $D$ be $s(n)$ time samplable distribution over $l_{\Pi}(n)$ such that for every randomized algorithm $A$ running in time $t(n)$, we have:

$$\Pr_{x \sim D} \left[ A \text{ finds optimal solution of } x \text{ w.p. } \geq 2/3 \right] \leq 1 - \frac{1}{p(n)}.$$

Then for $k = \text{poly}(p(n))$ there is $D'$ a $\tilde{O}(k \cdot s(n) + T(n, k))$ time samplable distribution over $l_{\Pi}(S(n, k))$ such that for every randomized algorithm $A'$ running in time* $\tilde{O}(t(n))$, we have:

$$\Pr_{x' \sim D'} \left[ A' \text{ finds optimal solution of } x' \text{ w.p. } \geq 2/3 \right] \leq 0.01.$$

*Conditions apply.
Theorem (Goldenberg-K’19)

Let $D$ be $\tilde{O}(n)$ time samplable distribution over LCS/Edit Distance such that for every randomized algorithm $\mathcal{A}$ running in time $n^{2-\epsilon}$, we have:

$$\Pr_{x \sim D} [\mathcal{A} \text{ finds optimal alignment of } x \text{ w.p. } \geq \frac{2}{3}] \leq 1 - \frac{1}{n^{o(1)}}.$$
Theorem (Goldenberg-K’19)

Let $D$ be $\tilde{O}(n)$ time samplable distribution over LCS/Edit Distance such that for every randomized algorithm $A$ running in time $n^{2-\epsilon}$, we have:

$$\Pr_{x \sim D} [A \text{ finds optimal alignment of } x \text{ w.p. } \geq \frac{2}{3}] \leq 1 - \frac{1}{n^{o(1)}}.$$ 

Then there is $D'$ a $\tilde{O}(n)$ time samplable distribution over LCS/Edit Distance such that for every randomized algorithm $A'$ running in time $n^{2-2\epsilon}$, we have:

$$\Pr_{x' \sim D'} [A' \text{ finds optimal alignment of } x' \text{ w.p. } \geq \frac{2}{3}] \leq 0.01.$$
Theorem (Goldenberg-K’19)

Let $D$ be $\tilde{O}(n)$ time samplable distribution over LCS/Edit Distance such that for every randomized algorithm $A$ running in time $n^{2-\varepsilon}$, we have:

$$\Pr_{x \sim D} [A \text{ finds optimal alignment of } x \text{ w.p. } \geq 2/3] \leq 1 - \frac{1}{n^{o(1)}}.$$

Then there is $D'$ a $\tilde{O}(n)$ time samplable distribution over LCS/Edit Distance such that for every randomized algorithm $A'$ running in time $n^{2-2\varepsilon}$, we have:

$$\Pr_{x' \sim D'} [A' \text{ finds optimal alignment of } x' \text{ w.p. } \geq 2/3] \leq 0.01.$$

What about Fréchet Distance?
Theorem (Goldenberg-K’19)

Let $D$ be $\tilde{O}(n)$ time samplable distribution over LCS/Edit Distance such that for every randomized algorithm $\mathcal{A}$ running in time $n^{2-\epsilon}$, we have:

$$\Pr_{x \sim D} \left[ \mathcal{A} \text{ finds optimal alignment of } x \text{ w.p. } \geq \frac{2}{3} \right] \leq 1 - \frac{1}{n^{o(1)}}.$$

Then there exists a $\tilde{O}(n)$ time samplable distribution over LCS/Edit Distance such that for every randomized algorithm $\mathcal{A}'$ running in time $n^{2-2\epsilon}$, we have:

$$\Pr_{x' \sim D'} \left[ \mathcal{A}' \text{ finds optimal alignment of } x' \text{ w.p. } \geq \frac{2}{3} \right] \leq 0.01.$$

What about Fréchet Distance?
Theorem (Goldenberg-K’19)

Let $D$ be $\text{poly}(n)$ time samplable distribution over Max-SAT such that for every randomized algorithm $\mathcal{A}$ running in time $2^{o(n)}$, we have:

$$\Pr_{x \sim D} \left[ \mathcal{A} \text{ finds optimal assignment of } x \text{ w.p. } \geq 2/3 \right] \leq 1 - \frac{1}{2^{n^{1-o(1)}}}.$$
Connection to Max-SAT

Theorem (Goldenberg-K’19)

Let $D$ be $\text{poly}(n)$ time samplable distribution over Max-SAT such that for every randomized algorithm $\mathcal{A}$ running in time $2^{o(n)}$, we have:

\[
\Pr_{x \sim D} [\mathcal{A} \text{ finds optimal assignment of } x \text{ w.p. } \geq \frac{2}{3}] \leq 1 - \frac{1}{2^{n^{1-o(1)}}}.
\]

Then there is $D'$ a $\text{poly}(n)$ time samplable distribution over Max-SAT such that for every randomized algorithm $\mathcal{A}'$ running in time $n^{\omega(1)}$, we have:

\[
\Pr_{x' \sim D'} [\mathcal{A}' \text{ finds optimal assignment of } x' \text{ w.p. } \geq \frac{2}{3}] \leq 0.01.
\]

Can be extended to Vertex Cover, Dominating Set, etc. Can even be extended to Knapsack, and other maximization problems!
Theorem (Goldenberg-K’19)

Let $D$ be a poly($n$) time samplable distribution over Max-SAT such that for every randomized algorithm $A$ running in time $2^{o(n)}$, we have:

$$\Pr_{x \sim D} [A \text{ finds optimal assignment of } x \text{ w.p. } \geq 2/3] \leq 1 - \frac{1}{2^{n^{1-o(1)}}}.$$

Then there is $D'$ a poly($n$) time samplable distribution over Max-SAT such that for every randomized algorithm $A'$ running in time $n^{\omega(1)}$, we have:

$$\Pr_{x' \sim D'} [A' \text{ finds optimal assignment of } x' \text{ w.p. } \geq 2/3] \leq 0.01.$$
Theorem (Goldenberg-K’19)
Let $D$ be a poly($n$) time samplable distribution over Max-SAT such that for every randomized algorithm $A$ running in time $2^{o(n)}$, we have:

$$
\Pr_{x \sim D}[A \text{ finds optimal assignment of } x \text{ w.p. } \geq \frac{2}{3}] \leq 1 - \frac{1}{2^{n^{1-o(1)}}}.
$$

Then there is $D'$ a poly($n$) time samplable distribution over Max-SAT such that for every randomized algorithm $A'$ running in time $n^{\omega(1)}$, we have:

$$
\Pr_{x' \sim D'}[A' \text{ finds optimal assignment of } x' \text{ w.p. } \geq \frac{2}{3}] \leq 0.01.
$$

Can be extended to Vertex Cover, Dominating Set, etc.

Can even be extended to Knapsack, and other maximization problems!
Factoring
Factoring

- Given $N \in [2^n]$ find all its prime factors
Connection to TFNP

Factoring

- Given $N \in [2^n]$ find all its prime factors
- Gen multiplies input integers
Connection to TFNP

Factoring

- Given $N \in [2^n]$ find all its prime factors
- Gen multiplies input integers
- Dec checks if candidate prime divides input integer
Connection to TFNP

Factoring

- Given $N \in [2^n]$ find all its prime factors
- Gen multiplies input integers
- Dec checks if candidate prime divides input integer

End of Line Problem

- Given $P, S : \{0, 1\}^n \rightarrow \{0, 1\}^n$ such that $P(0^n) = 0^n \neq S(0^n)$
Connection to TFNP

Factoring

⊙ Given $N \in [2^n]$ find all its prime factors
⊙ Gen multiplies input integers
⊙ Dec checks if candidate prime divides input integer

End of Line Problem

⊙ Given $P, S : \{0, 1\}^n \to \{0, 1\}^n$ such that $P(0^n) = 0^n \neq S(0^n)$
⊙ Find $x$ such that $P(S(x)) \neq x$ or $S(P(x)) = x \neq 0^n$
Connection to TFNP

Factoring

- Given $N \in [2^n]$ find all its prime factors
- Gen *multiplies* input integers
- Dec checks if candidate prime *divides* input integer

End of Line Problem

- Given $P, S : \{0, 1\}^n \rightarrow \{0, 1\}^n$ such that $P(0^n) = 0^n \neq S(0^n)$
- Find $x$ such that $P(S(x)) \neq x$ or $S(P(x)) = x \neq 0^n$
- Gen *concatenates* input and output gates
Connection to TFNP

Factoring

- Given $N \in [2^n]$ find all its prime factors
- Gen multiplies input integers
- Dec checks if candidate prime divides input integer

End of Line Problem

- Given $P, S : \{0, 1\}^n \rightarrow \{0, 1\}^n$ such that $P(0^n) = 0^n \neq S(0^n)$
- Find $x$ such that $P(S(x)) \neq x$ or $S(P(x)) = x \neq 0^n$
- Gen concatenates input and output gates
- Dec restricts on the corresponding block
Open Problem 1

Average case hard problems in P
Open Problem 1

Average case hard problems in P

- Can we show some *natural* problem in P is hard for the *uniform* distribution?
Open Problem 1

Average case hard problems in P

- Can we show some *natural* problem in P is hard for the *uniform* distribution?

- Can we construct a *fine-grained* one way function from *worst case* assumptions?
Open Problem 2

Gap Amplification vs. Hardness Amplification
Open Problem 2

Gap Amplification vs. Hardness Amplification

Can we obtain a trade-off between gap and hardness?
Open Problem 2

Gap Amplification vs. Hardness Amplification

- Can we obtain a trade-off between gap and hardness?
- Can we say something stronger about Max-SAT assuming Gap-ETH?
Open Problem 3

Direct Product Feasibility

Can we characterize direct product feasible pairs?

Can we show Orthogonal Vectors is self direct product feasible?

Can we show LCS is self direct product feasible?
Open Problem 3

Direct Product Feasibility

- Can we characterize direct product feasible pairs?
Open Problem 3

Direct Product Feasibility

- Can we characterize direct product feasible pairs?
- Can we show Orthogonal Vectors is self direct product feasible?
Open Problem 3

Direct Product Feasibility

- Can we characterize direct product feasible pairs?
- Can we show Orthogonal Vectors is self direct product feasible?
- Can we show LCS is self direct product feasible?
Key Takeaways

- Hardness Amplification Technique

For Optimization problems
- via Direct Products
- against Randomized algorithms

- Hardness Amplification meets Fine-Grained Complexity
  - Amplify hardness from \( \frac{1}{n} \) to \( 1 - o(1) \) for LCS, Edit Distance, etc.

- If ETH is true on mild worst case then Max-SAT is hard on average
Key Takeaways

- Hardness Amplification Technique
  - for Optimization problems
  - via Direct Products
  - against Randomized algorithms
Key Takeaways

- **Hardness Amplification Technique**
  - for **Optimization** problems
  - via **Direct Products**
  - against **Randomized** algorithms

- **Hardness Amplification meets Fine-Grained Complexity**
Key Takeaways

- Hardness Amplification Technique
  - for Optimization problems
  - via Direct Products
  - against Randomized algorithms

- Hardness Amplification meets Fine-Grained Complexity
  - Amplify hardness from $1/n^{o(1)}$ to $1 - o(1)$
    for LCS, Edit Distance, etc.
Key Takeaways

- **Hardness Amplification Technique**
  - for Optimization problems
  - via Direct Products
  - against Randomized algorithms

- **Hardness Amplification meets Fine-Grained Complexity**
  - Amplify hardness from $1/n^{o(1)}$ to $1 - o(1)$
    for LCS, Edit Distance, etc.
  - If ETH is true on mild worst case then Max-SAT is hard on average
THANK YOU!