Expander Graphs and Their Applications (XI)

Yijia Chen
Shanghai Jiaotong University

Review of the Previous Lecture

MAX-dSAT

Let $d \in \mathbb{N}$.

MAX-dSAT

Input:	An $\alpha \in d$CNF.
Solution:	An assignment $\mathcal{V}$ for α.
Cost:	the number of clauses in α that $\mathcal{V}$ satisfies.
Goal:	max.

Definition

For every dCNF-formula α and an assignment $\mathcal{V}$ for α,

$$\underline{\mathrm{sat}(\alpha, \mathcal{V})} := \text{the number of clauses in } \alpha \text{ that } \mathcal{V} \text{ satisfies}$$

and

$$\underline{\mathrm{optsat}(\alpha)} := \max_{\mathcal{V}} \mathrm{sat}(\alpha, \mathcal{V}).$$

Hardness of Approximating $\textsc{Max-}d\textsc{Sat}$

Definition

Let $\mathbb{A}$ be an algorithm that on every $d\textsc{Cnf}$-formula α always output an assignment for α which we denote by $\mathcal{V}_\alpha$. The performance ratio of $\mathbb{A}$ is

$$\max_{\alpha \in d\textsc{Cnf}} \frac{\mathsf{optsat}(\alpha)}{\mathsf{sat}(\alpha, \mathcal{V}_\alpha)}.$$

Theorem

For some constants $d \in \mathbb{N}$ and $r > 1$, there is no polynomial time approximation algorithm for $\textsc{Max-}d\textsc{Sat}$ with performance ratio r, unless $\mathsf{P} = \mathsf{NP}$*.*

Proof Idea.

We reduce an NP-complete Q with $(c \cdot \log n, d)$-restricted verifier to MAX-dSAT.

Given an $x \in \Sigma^*$, we will construct a dCNF formula α_x satisfying

Proof Idea.

We reduce an NP-complete Q with $(c \cdot \log n, d)$-restricted verifier to $\textsc{Max-}d\textsc{Sat}$.

Given an $x \in \Sigma^*$, we will construct a $d\textsc{Cnf}$ formula α_x satisfying

$$\begin{aligned} x \in Q \quad &\Rightarrow \quad \alpha_x \text{ is satisfiable,} \\ x \notin Q \quad &\Rightarrow \quad \text{for any assignment } \mathcal{V}, \\ &\qquad\quad \text{at least } 2^{-d-1}\text{-fraction of clauses are not satisfied.} \end{aligned}$$

unsat(α)

unsat(α)

Let $\alpha \in d\textsc{Cnf}$, and we define

$\underline{\mathsf{unsat}(\alpha)}$:= the smallest fraction of *unsatisfied* clauses of α under any assignment.

unsat(α)

Let $\alpha \in d\textsc{Cnf}$, and we define

$$\underline{\text{unsat}(\alpha)} := \text{the smallest fraction of } \textit{unsatisfied} \text{ clauses of } \alpha \text{ under any assignment.}$$

$$\begin{aligned} \alpha \text{ is satisfiable} \quad &\Longleftrightarrow \quad \text{unsat}(\alpha) = 0; \\ \alpha \text{ is not satisfiable} \quad &\Longleftrightarrow \quad \text{unsat}(\alpha) \geq \frac{1}{m} \end{aligned}$$

where m is the number of clauses in α.

Gap-dSat

GAP-dSAT

Let $\varepsilon > 0$.

GAP-dSAT

Let $\varepsilon > 0$.

GAP-dSAT$_\epsilon$

Input: A propositional formula $\alpha \in d$CNF, such that *either* unsat$(\alpha) = 0$ *or* unsat$(\alpha) \geq \epsilon$.

Problem: Decide whether α is satisfiable.

GAP-*d*SAT

Let $\varepsilon > 0$.

GAP-*d*SAT$_\epsilon$	
Input:	A propositional formula $\alpha \in d$CNF, such that *either* unsat$(\alpha) = 0$ *or* unsat$(\alpha) \geq \epsilon$.
Problem:	Decide whether α is satisfiable.

GAP-*d*SAT$_\epsilon$ is a *promise problem*.

Hardness of GAP-dSAT

Hardness of $\textsc{Gap-}d\textsc{Sat}$

Theorem

There exist some $d \in \mathbb{N}$ *and* $\epsilon > 0$ *such that* $\textsc{Gap-}d\textsc{Sat}_\epsilon$ *is* NP*-hard.*

Hardness of $\textsc{Gap-}d\textsc{Sat}$

Theorem

There exist some $d \in \mathbb{N}$ *and* $\epsilon > 0$ *such that* $\textsc{Gap-}d\textsc{Sat}_\epsilon$ *is* NP*-hard.*

Proof.

Recall in the previous proof, for some NP-hard problem Q, appropriate $d \in \mathbb{N}$ and $\epsilon > 0$, we implicitly define an algorithm $\mathbb{A}$, such that for each instance $x \in \Sigma^*$, the algorithm $\mathbb{A}$ produces a $d\textsc{Cnf}$-formula α_x satisfying:

Hardness of $\textsc{Gap-}d\textsc{Sat}$

Theorem

There exist some $d \in \mathbb{N}$ and $\epsilon > 0$ such that $\textsc{Gap-}d\textsc{Sat}_\epsilon$ is NP*-hard.*

Proof.

Recall in the previous proof, for some NP-hard problem Q, appropriate $d \in \mathbb{N}$ and $\epsilon > 0$, we implicitly define an algorithm $\mathbb{A}$, such that for each instance $x \in \Sigma^*$, the algorithm $\mathbb{A}$ produces a $d\textsc{Cnf}$-formula α_x satisfying:

$$\begin{aligned} x \in Q \quad &\Rightarrow \quad \alpha_x \text{ is satisfiable, i.e., } \mathrm{unsat}(\alpha_x) = 0, \\ x \notin Q \quad &\Rightarrow \quad \text{for any assignment } \mathcal{V}, \text{ at least } \epsilon\text{-fraction of} \\ &\qquad\qquad \text{clauses are not satisfied, i.e., } \mathrm{unsat}(\alpha_x) \geq \epsilon. \end{aligned}$$

Hardness of $\textsc{Gap-}d\textsc{Sat}$

Theorem

There exist some $d \in \mathbb{N}$ *and* $\epsilon > 0$ *such that* $\textsc{Gap-}d\textsc{Sat}_\epsilon$ *is* NP*-hard.*

Proof.

Recall in the previous proof, for some NP-hard problem Q, appropriate $d \in \mathbb{N}$ and $\epsilon > 0$, we implicitly define an algorithm $\mathbb{A}$, such that for each instance $x \in \Sigma^*$, the algorithm $\mathbb{A}$ produces a $d\textsc{Cnf}$-formula α_x satisfying:

$$\begin{aligned} x \in Q \quad &\Rightarrow \quad \alpha_x \text{ is satisfiable, i.e., } \mathrm{unsat}(\alpha_x) = 0, \\ x \notin Q \quad &\Rightarrow \quad \text{for any assignment } \mathcal{V}\text{, at least } \epsilon\text{-fraction of} \\ &\qquad\qquad \text{clauses are not satisfied, i.e., } \mathrm{unsat}(\alpha_x) \geq \epsilon. \end{aligned}$$

That is, $\mathbb{A}$ gives a reduction from Q to $\textsc{Gap-}d\textsc{Sat}_\epsilon$. ☐

Hardness of GAP-3SAT

Hardness of $\textsc{Gap-3Sat}$

Theorem

There exists an $\epsilon > 0$ *such that* $\textsc{Gap-3Sat}_{\epsilon}$ *is* NP*-hard.*

The Reduction from dSAT to 3SAT

The Reduction from dSAT to 3SAT

Lemma

Let $d > 3$ and $r \geq 1$.

The Reduction from dSat to 3Sat

Lemma

Let d > 3 and r ≥ 1. there is a polynomial time algorithm $\mathbb{R}$ such that, given an $\alpha \in d$Cnf, the algorithm $\mathbb{R}$ produces an $\alpha^* \in 3$Cnf such that

The Reduction from $d\textsc{Sat}$ to $3\textsc{Sat}$

Lemma

Let $d > 3$ and $r \geq 1$. there is a polynomial time algorithm $\mathbb{R}$ such that, given an $\alpha \in d\textsc{Cnf}$, the algorithm $\mathbb{R}$ produces an $\alpha^* \in 3\textsc{Cnf}$ such that

(i) *α is satisfiable $\Rightarrow \alpha^*$ is satisfiable.*

The Reduction from dSAT to 3SAT

Lemma

Let $d > 3$ and $r \geq 1$. there is a polynomial time algorithm $\mathbb{R}$ such that, given an $\alpha \in d\textsc{Cnf}$, the algorithm $\mathbb{R}$ produces an $\alpha^* \in 3\textsc{Cnf}$ such that

(i) *α is satisfiable $\Rightarrow \alpha^*$ is satisfiable.*

(ii) *For any assignment, at least ϵ-fraction of clauses in α is not satisfied $\Longrightarrow$ for any assignment, at least $\frac{\epsilon}{d-2}$-fraction of clauses in α^* is not satisfied.*

The Reduction from $d\textsc{Sat}$ to $3\textsc{Sat}$

Lemma

Let $d > 3$ and $r \geq 1$. there is a polynomial time algorithm $\mathbb{R}$ such that, given an $\alpha \in d\textsc{Cnf}$, the algorithm $\mathbb{R}$ produces an $\alpha^* \in 3\textsc{Cnf}$ such that

(i) *α is satisfiable $\Rightarrow$ α^* is satisfiable.*

(ii) *For any assignment, at least ϵ-fraction of clauses in α is not satisfied $\Longrightarrow$ for any assignment, at least $\frac{\epsilon}{d-2}$-fraction of clauses in α^* is not satisfied.*

Proof. Let $\alpha \in d\textsc{Cnf}$, i.e., for an index set I,

$$\alpha = \bigwedge_{i \in I} \beta_i$$

The Reduction from dSAT to 3SAT

Lemma

Let $d > 3$ and $r \geq 1$. there is a polynomial time algorithm $\mathbb{R}$ such that, given an $\alpha \in d\text{CNF}$, the algorithm $\mathbb{R}$ produces an $\alpha^* \in 3\text{CNF}$ such that

(i) *α is satisfiable $\Rightarrow$ α^* is satisfiable.*

(ii) *For any assignment, at least ϵ-fraction of clauses in α is not satisfied $\Longrightarrow$ for any assignment, at least $\frac{\epsilon}{d-2}$-fraction of clauses in α^* is not satisfied.*

Proof. Let $\alpha \in d\text{CNF}$, i.e., for an index set I,

$$\alpha = \bigwedge_{i \in I} \beta_i$$

where each

$$\beta_i = \lambda_{i,1} \vee \lambda_{i,2} \vee \ldots \vee \lambda_{i,t_i}$$

for some literals $\lambda_{i,1}, \lambda_{i,2}, \ldots, \lambda_{i,t_i}$ and $t_i \leq d$.

Proof. (cont'd)

Proof. (cont'd)

For every $i \in I$ with $t_i > 3$, let $Y_{i,1}, \ldots, Y_{i,t_i-3}$ be some new variables.

Proof. (cont'd)

For every $i \in I$ with $t_i > 3$, let $Y_{i,1}, \ldots, Y_{i,t_i-3}$ be some new variables. And let γ_i be the following formula

$$(\lambda_{i,1} \vee \lambda_{i,2} \vee Y_{i,1}) \wedge (\neg Y_{i,1} \vee \lambda_{i,3} \vee Y_{i,2}) \wedge \cdots \wedge (\neg Y_{i,t_i-3} \vee \lambda_{i,m-1} \vee \lambda_{i,t_i}).$$

Proof. (cont'd)

For every $i \in I$ with $t_i > 3$, let $Y_{i,1}, \ldots, Y_{i,t_i-3}$ be some new variables. And let γ_i be the following formula

$$(\lambda_{i,1} \lor \lambda_{i,2} \lor Y_{i,1}) \land (\neg Y_{i,1} \lor \lambda_{i,3} \lor Y_{i,2}) \land \cdots \land (\neg Y_{i,t_i-3} \lor \lambda_{i,m-1} \lor \lambda_{i,t_i}).$$

Otherwise let $\gamma_i := \beta_i$.

Proof. (cont'd)

For every $i \in I$ with $t_i > 3$, let $Y_{i,1}, \ldots, Y_{i,t_i-3}$ be some new variables. And let γ_i be the following formula

$$(\lambda_{i,1} \vee \lambda_{i,2} \vee Y_{i,1}) \wedge (\neg Y_{i,1} \vee \lambda_{i,3} \vee Y_{i,2}) \wedge \cdots \wedge (\neg Y_{i,t_i-3} \vee \lambda_{i,m-1} \vee \lambda_{i,t_i}).$$

Otherwise let $\gamma_i := \beta_i$. Then we take

$$\alpha^* := \bigwedge_{i \in I} \gamma_i$$

as the required $3\textsc{Cnf}$ formula.

Proof. (cont'd)

For every $i \in I$ with $t_i > 3$, let $Y_{i,1}, \ldots, Y_{i,t_i-3}$ be some new variables. And let γ_i be the following formula

$$(\lambda_{i,1} \vee \lambda_{i,2} \vee Y_{i,1}) \wedge (\neg Y_{i,1} \vee \lambda_{i,3} \vee Y_{i,2}) \wedge \cdots \wedge (\neg Y_{i,t_i-3} \vee \lambda_{i,m-1} \vee \lambda_{i,t_i}).$$

Otherwise let $\gamma_i := \beta_i$. Then we take

$$\alpha^* := \bigwedge_{i \in I} \gamma_i$$

as the required 3$\textsc{Cnf}$ formula.

(i) is easy.

Proof. (cont'd)

For every $i \in I$ with $t_i > 3$, let $Y_{i,1}, \ldots, Y_{i,t_i-3}$ be some new variables. And let γ_i be the following formula

$$(\lambda_{i,1} \vee \lambda_{i,2} \vee Y_{i,1}) \wedge (\neg Y_{i,1} \vee \lambda_{i,3} \vee Y_{i,2}) \wedge \cdots \wedge (\neg Y_{i,t_i-3} \vee \lambda_{i,m-1} \vee \lambda_{i,t_i}).$$

Otherwise let $\gamma_i := \beta_i$. Then we take

$$\alpha^* := \bigwedge_{i \in I} \gamma_i$$

as the required $3\textsc{Cnf}$ formula.

(i) is easy. We show (ii).

Proof. (cont'd)

For every $i \in I$ with $t_i > 3$, let $Y_{i,1}, \ldots, Y_{i,t_i-3}$ be some new variables. And let γ_i be the following formula

$$(\lambda_{i,1} \vee \lambda_{i,2} \vee Y_{i,1}) \wedge (\neg Y_{i,1} \vee \lambda_{i,3} \vee Y_{i,2}) \wedge \cdots \wedge (\neg Y_{i,t_i-3} \vee \lambda_{i,m-1} \vee \lambda_{i,t_i}).$$

Otherwise let $\gamma_i := \beta_i$. Then we take

$$\alpha^* := \bigwedge_{i \in I} \gamma_i$$

as the required 3CNF formula.

(i) is easy. We show (ii).
Let $\mathcal{V}^*$ be an assignment for α^*.

Proof. (cont'd)

For every $i \in I$ with $t_i > 3$, let $Y_{i,1}, \ldots, Y_{i,t_i-3}$ be some new variables. And let γ_i be the following formula

$$(\lambda_{i,1} \vee \lambda_{i,2} \vee Y_{i,1}) \wedge (\neg Y_{i,1} \vee \lambda_{i,3} \vee Y_{i,2}) \wedge \cdots \wedge (\neg Y_{i,t_i-3} \vee \lambda_{i,m-1} \vee \lambda_{i,t_i}).$$

Otherwise let $\gamma_i := \beta_i$. Then we take

$$\alpha^* := \bigwedge_{i \in I} \gamma_i$$

as the required 3CNF formula.

(i) is easy. We show (ii).
Let $\mathcal{V}^*$ be an assignment for α^*. And let $\mathcal{V}$ be *the restriction of $\mathcal{V}^*$ to the variables of α*.

Proof. (cont'd)

For every $i \in I$ with $t_i > 3$, let $Y_{i,1}, \ldots, Y_{i,t_i-3}$ be some new variables. And let γ_i be the following formula

$$(\lambda_{i,1} \vee \lambda_{i,2} \vee Y_{i,1}) \wedge (\neg Y_{i,1} \vee \lambda_{i,3} \vee Y_{i,2}) \wedge \cdots \wedge (\neg Y_{i,t_i-3} \vee \lambda_{i,m-1} \vee \lambda_{i,t_i}).$$

Otherwise let $\gamma_i := \beta_i$. Then we take

$$\alpha^* := \bigwedge_{i \in I} \gamma_i$$

as the required 3CNF formula.

(i) is easy. We show (ii).
Let $\mathcal{V}^*$ be an assignment for α^*. And let $\mathcal{V}$ be *the restriction of $\mathcal{V}^*$ to the variables of α*.
Assume α contains m clauses. By assumption, under $\mathcal{V}$, at least

$$\lceil \epsilon \cdot m \rceil$$

clauses are *not* satisfied.

Proof. (cont'd)

Proof. (cont'd)

Let $\beta_i = \lambda_{i,1} \vee \lambda_{i,2} \vee \ldots \vee \lambda_{i,t_i}$ be an unsatisfied clause.

Proof. (cont'd)

Let $\beta_i = \lambda_{i,1} \vee \lambda_{i,2} \vee \ldots \vee \lambda_{i,t_i}$ be an unsatisfied clause. Thus, in

$\gamma_i = (\lambda_{i,1} \vee \lambda_{i,2} \vee Y_{i,1}) \wedge (\neg Y_{i,1} \vee \lambda_{i,3} \vee Y_{i,2}) \wedge \cdots \wedge (\neg Y_{i,t_i-3} \vee \lambda_{i,m-1} \vee \lambda_{i,t_i})$.

at least one clause is not satisfied under $\mathcal{V}^*$.

Proof. (cont'd)

Let $\beta_i = \lambda_{i,1} \vee \lambda_{i,2} \vee \ldots \vee \lambda_{i,t_i}$ be an unsatisfied clause. Thus, in

$$\gamma_i = (\lambda_{i,1} \vee \lambda_{i,2} \vee Y_{i,1}) \wedge (\neg Y_{i,1} \vee \lambda_{i,3} \vee Y_{i,2}) \wedge \cdots \wedge (\neg Y_{i,t_i-3} \vee \lambda_{i,m-1} \vee \lambda_{i,t_i}).$$

at least one clause is not satisfied under $\mathcal{V}^*$.
Hence we have at least $\lceil \epsilon \cdot m \rceil$ unsatisfied clauses in α^*.

Proof. (cont'd)

Let $\beta_i = \lambda_{i,1} \vee \lambda_{i,2} \vee \ldots \vee \lambda_{i,t_i}$ be an unsatisfied clause. Thus, in

$$\gamma_i = (\lambda_{i,1} \vee \lambda_{i,2} \vee Y_{i,1}) \wedge (\neg Y_{i,1} \vee \lambda_{i,3} \vee Y_{i,2}) \wedge \cdots \wedge (\neg Y_{i,t_i-3} \vee \lambda_{i,m-1} \vee \lambda_{i,t_i}).$$

at least one clause is not satisfied under $\mathcal{V}^*$.
Hence we have at least $\lceil \epsilon \cdot m \rceil$ unsatisfied clauses in α^*.
α^* contains at most

$$(d-2) \cdot m$$

clauses.

Proof. (cont'd)

Let $\beta_i = \lambda_{i,1} \vee \lambda_{i,2} \vee \ldots \vee \lambda_{i,t_i}$ be an unsatisfied clause. Thus, in

$$\gamma_i = (\lambda_{i,1} \vee \lambda_{i,2} \vee Y_{i,1}) \wedge (\neg Y_{i,1} \vee \lambda_{i,3} \vee Y_{i,2}) \wedge \cdots \wedge (\neg Y_{i,t_i-3} \vee \lambda_{i,m-1} \vee \lambda_{i,t_i}).$$

at least one clause is not satisfied under $\mathcal{V}^*$.
Hence we have at least $\lceil \epsilon \cdot m \rceil$ unsatisfied clauses in α^*.
α^* contains at most

$$(d-2) \cdot m$$

clauses. Thus the fraction of clauses that are not satisfied under $\mathcal{V}^*$ is at least

$$\frac{\lceil \epsilon \cdot m \rceil}{(d-2) \cdot m} \geq \frac{\epsilon}{d-2}.$$

□

Hardness of MAX-3SAT

Hardness of MAX-3SAT

Theorem

There is a constant $r > 1$ such that there is no polynomial time approximation algorithm for MAX-3SAT *with performance ratio r, unless* $\mathsf{P} = \mathsf{NP}$.

Hardness of $\textsc{Max-3Sat}$

Theorem

There is a constant $r > 1$ such that there is no polynomial time approximation algorithm for $\textsc{Max-3Sat}$ *with performance ratio r, unless* P $=$ NP.

Proof. By the previous theorem, there is some $\epsilon > 0$ such that $\textsc{Gap-3Sat}_\epsilon$ is NP-hard.

Hardness of $\textsc{Max-3Sat}$

Theorem

There is a constant $r > 1$ such that there is no polynomial time approximation algorithm for $\textsc{Max-3Sat}$ with performance ratio r, unless $\mathsf{P} = \mathsf{NP}$.

Proof. By the previous theorem, there is some $\epsilon > 0$ such that $\textsc{Gap-3Sat}_\epsilon$ is NP-hard. Without loss of generality, we assume $\epsilon < 1$.

Hardness of MAX-3SAT

Theorem

There is a constant $r > 1$ such that there is no polynomial time approximation algorithm for MAX-3SAT *with performance ratio r, unless* P = NP.

Proof. By the previous theorem, there is some $\epsilon > 0$ such that GAP-3SAT$_\epsilon$ is NP-hard. Without loss of generality, we assume $\epsilon < 1$. Choose any

$$r < \frac{1}{1-\epsilon}.$$

Hardness of MAX-3SAT

Theorem

There is a constant $r > 1$ such that there is no polynomial time approximation algorithm for MAX-3SAT *with performance ratio r, unless $\mathrm{P} = \mathrm{NP}$.*

Proof. By the previous theorem, there is some $\epsilon > 0$ such that GAP-3SAT_ϵ is NP-hard. Without loss of generality, we assume $\epsilon < 1$. Choose any

$$r < \frac{1}{1-\epsilon}.$$

Assume an algorithm $\mathbb{A}$ approximates MAX-3SAT with performance ratio r.

Hardness of $\textsc{Max-3Sat}$

Theorem

There is a constant $r > 1$ such that there is no polynomial time approximation algorithm for $\textsc{Max-3Sat}$ *with performance ratio r, unless* $\mathrm{P} = \mathrm{NP}$.

Proof. By the previous theorem, there is some $\epsilon > 0$ such that $\textsc{Gap-3Sat}_\epsilon$ is NP-hard. Without loss of generality, we assume $\epsilon < 1$. Choose any

$$r < \frac{1}{1-\epsilon}.$$

Assume an algorithm $\mathbb{A}$ approximates $\textsc{Max-3Sat}$ with performance ratio r. Let $\alpha \in 3\textsc{Cnf}$ with m clauses.

Hardness of $\textsc{Max-3Sat}$

Theorem

There is a constant $r > 1$ such that there is no polynomial time approximation algorithm for $\textsc{Max-3Sat}$ with performance ratio r, unless $\mathrm{P} = \mathrm{NP}$.

Proof. By the previous theorem, there is some $\epsilon > 0$ such that $\textsc{Gap-3Sat}_\epsilon$ is NP-hard. Without loss of generality, we assume $\epsilon < 1$. Choose any

$$r < \frac{1}{1-\epsilon}.$$

Assume an algorithm $\mathbb{A}$ approximates $\textsc{Max-3Sat}$ with performance ratio r. Let $\alpha \in 3\textsc{Cnf}$ with m clauses.

(i) If $\mathrm{unsat}(\alpha) = 0$, then $\mathbb{A}$ outputs an assignment that satisfies at least

$$\left\lceil \frac{m}{r} \right\rceil > (1-\epsilon) \cdot m \quad \text{clauses.}$$

Hardness of MAX-3SAT

Theorem

There is a constant $r > 1$ such that there is no polynomial time approximation algorithm for MAX-3SAT *with performance ratio r, unless* $\mathrm{P} = \mathrm{NP}$.

Proof. By the previous theorem, there is some $\epsilon > 0$ such that $\textsc{Gap-3Sat}_\epsilon$ is NP-hard. Without loss of generality, we assume $\epsilon < 1$. Choose any

$$r < \frac{1}{1-\epsilon}.$$

Assume an algorithm $\mathbb{A}$ approximates MAX-3SAT with performance ratio r. Let $\alpha \in 3\textsc{Cnf}$ with m clauses.

(i) If $\mathrm{unsat}(\alpha) = 0$, then $\mathbb{A}$ outputs an assignment that satisfies at least

$$\left\lceil \frac{m}{r} \right\rceil > (1-\epsilon) \cdot m \quad \text{clauses.}$$

(ii) If $\mathrm{unsat}(\alpha) \geq \epsilon$, then any assignment (including the one $\mathbb{A}$ outputs) can satisfy less than

$$m - \lceil \epsilon \cdot m \rceil \leq m - \epsilon \cdot m \quad \text{clauses.}$$

$\square$

PCP by the Hardness of GAP-dSAT

PCP by the Hardness of $\textsc{Gap-}d\textsc{Sat}$

Theorem

If there exists some $1 \geq \epsilon > 0$*, such that* $\textsc{Gap-3Sat}_\epsilon$ *is* NP*-hard, then* $\mathsf{NP} = \mathsf{PCP}(\log n, 1)$.

Proof.

Proof.

Let $Q \subseteq \Sigma^*$ be a problem in NP.

Proof.

Let $Q \subseteq \Sigma^*$ be a problem in NP. Then there is a polynomial time algorithm $\mathbb{R}$ such that for every $x \in \Sigma^*$, the algorithm $\mathbb{A}$ computes an $\alpha_x \in 3\textsc{Cnf}$ such that

$$\begin{aligned} x \in Q \quad &\Longrightarrow \quad \mathrm{unsat}(\alpha_x) = 0; \\ x \notin Q \quad &\Longrightarrow \quad \mathrm{unsat}(\alpha_x) \geq \epsilon. \end{aligned}$$

Proof.

Let $Q \subseteq \Sigma^*$ be a problem in NP. Then there is a polynomial time algorithm $\mathbb{R}$ such that for every $x \in \Sigma^*$, the algorithm $\mathbb{A}$ computes an $\alpha_x \in 3\textsc{Cnf}$ such that

$$\begin{aligned} x \in Q \quad &\Longrightarrow \quad \mathsf{unsat}(\alpha_x) = 0; \\ x \notin Q \quad &\Longrightarrow \quad \mathsf{unsat}(\alpha_x) \geq \epsilon. \end{aligned}$$

Without loss of generality, we assume α_x *contains at most* $|x|^c$ *many clauses*, for some $c \in \mathbb{N}$.

Proof.

Let $Q \subseteq \Sigma^*$ be a problem in NP. Then there is a polynomial time algorithm $\mathbb{R}$ such that for every $x \in \Sigma^*$, the algorithm $\mathbb{A}$ computes an $\alpha_x \in 3\textsc{Cnf}$ such that

$$\begin{aligned} x \in Q \quad &\Longrightarrow \quad \mathrm{unsat}(\alpha_x) = 0; \\ x \notin Q \quad &\Longrightarrow \quad \mathrm{unsat}(\alpha_x) \geq \epsilon. \end{aligned}$$

Without loss of generality, we assume α_x *contains at most* $|x|^c$ *many clauses*, for some $c \in \mathbb{N}$.

Choose any $d \in \mathbb{N}$ with

$$\left(1 - \frac{\epsilon}{2}\right)^d \leq \frac{1}{2},$$

e.g., $d := \lceil 2/\epsilon \rceil$.

Proof. (cont'd)

Proof. (cont'd)

Consider the following verifier $V(x, \tau, \pi)$ with $|\tau| = c \cdot d \cdot \log|x|$:

Proof. (cont'd)

Consider the following verifier $V(x, \tau, \pi)$ with $|\tau| = c \cdot d \cdot \log |x|$:

1. We view τ as *a sequence of d many $c \cdot \log |x|$-bit binaries*, which correspond to numbers $t_1, \ldots, t_d$, each between 0 and $|x|^c - 1$.

Proof. (cont'd)

Consider the following verifier $V(x, \tau, \pi)$ with $|\tau| = c \cdot d \cdot \log |x|$:

1. We view τ as *a sequence of d many $c \cdot \log |x|$-bit binaries*, which correspond to numbers $t_1, \ldots, t_d$, each between 0 and $|x|^c - 1$.
2. Simulate $\mathbb{R}$ on x to compute α_x.

Proof. (cont'd)

Consider the following verifier $V(x, \tau, \pi)$ with $|\tau| = c \cdot d \cdot \log|x|$:

1. We view τ as *a sequence of d many $c \cdot \log|x|$-bit binaries*, which correspond to numbers $t_1, \ldots, t_d$, each between 0 and $|x|^c - 1$.
2. Simulate $\mathbb{R}$ on x to compute α_x.
3. $m \leftarrow$ the number of clauses in α_x.

Proof. (cont'd)

Consider the following verifier $V(x, \tau, \pi)$ with $|\tau| = c \cdot d \cdot \log |x|$:

1. We view τ as *a sequence of d many $c \cdot \log |x|$-bit binaries*, which correspond to numbers $t_1, \ldots, t_d$, each between 0 and $|x|^c - 1$.
2. Simulate $\mathbb{R}$ on x to compute α_x.
3. $m \leftarrow$ the number of clauses in α_x.
4. For all $1 \leq i \leq d$, $p_i \leftarrow (t_i \mod m) + 1$.

Proof. (cont'd)

Consider the following verifier $V(x, \tau, \pi)$ with $|\tau| = c \cdot d \cdot \log |x|$:

1. We view τ as *a sequence of d many $c \cdot \log |x|$-bit binaries*, which correspond to numbers $t_1, \ldots, t_d$, each between 0 and $|x|^c - 1$.
2. Simulate $\mathbb{R}$ on x to compute α_x.
3. $m \leftarrow$ the number of clauses in α_x.
4. For all $1 \le i \le d$, $p_i \leftarrow (t_i \mod m) + 1$.
5. Check whether π, as an assignment, satisfies the p_1-, ..., p_d-th clauses. If so, accept. Otherwise reject.

Proof. (cont'd)

Consider the following verifier $V(x, \tau, \pi)$ with $|\tau| = c \cdot d \cdot \log |x|$:

1. We view τ as *a sequence of d many $c \cdot \log |x|$-bit binaries*, which correspond to numbers $t_1, \ldots, t_d$, each between 0 and $|x|^c - 1$.
2. Simulate $\mathbb{R}$ on x to compute α_x.
3. $m \leftarrow$ the number of clauses in α_x.
4. For all $1 \leq i \leq d$, $p_i \leftarrow (t_i \mod m) + 1$.
5. Check whether π, as an assignment, satisfies the p_1-, ..., p_d-th clauses. If so, accept. Otherwise reject.

Clearly V is a $(\log n, 1)$-restricted verifier.

Proof. (cont'd)

Proof. (cont'd)

Now we have to check V correctly decides Q.

Proof. (cont'd)

Now we have to check V correctly decides Q.

If $x \in Q$, then $\mathrm{unsat}(\alpha_x) = 0$.

Proof. (cont'd)

Now we have to check V correctly decides Q.

If $x \in Q$, then $\mathsf{unsat}(\alpha_x) = 0$.
There is an assignment π satisfy α_x.

Proof. (cont'd)

Now we have to check V correctly decides Q.

If $x \in Q$, then $\text{unsat}(\alpha_x) = 0$.
There is an assignment π satisfy α_x. Thus no matter what τ we have, $V(x, \tau, \pi)$ always accepts.

Proof. (cont'd)

Now we have to check V correctly decides Q.

If $x \in Q$, then $\text{unsat}(\alpha_x) = 0$.
There is an assignment π satisfy α_x. Thus no matter what τ we have, $V(x, \tau, \pi)$ always accepts.

If $x \notin Q$, then $\text{unsat}(\alpha_x) \geq \epsilon$,

Proof. (cont'd)

Now we have to check V correctly decides Q.

If $x \in Q$, then unsat$(\alpha_x) = 0$.
There is an assignment π satisfy α_x. Thus no matter what τ we have, $V(x, \tau, \pi)$ always accepts.

If $x \notin Q$, then unsat$(\alpha_x) \geq \epsilon$, i.e., no matter what assignment π we have, at least ϵ-fraction of clauses are not satisfied.

Proof. (cont'd)

Now we have to check V correctly decides Q.

If $x \in Q$, then $\text{unsat}(\alpha_x) = 0$.
There is an assignment π satisfy α_x. Thus no matter what τ we have, $V(x, \tau, \pi)$ always accepts.

If $x \notin Q$, then $\text{unsat}(\alpha_x) \geq \epsilon$, i.e., no matter what assignment π we have, at least ϵ-fraction of clauses are not satisfied.
We pick a number t with $0 \leq t < |x|^c$ in random, and let

$$p := (t \mod m) + 1$$

where m is the number of clauses in α_x.

Proof. (cont'd)

Now we have to check V correctly decides Q.

If $x \in Q$, then $\mathrm{unsat}(\alpha_x) = 0$.
There is an assignment π satisfy α_x. Thus no matter what τ we have, $V(x, \tau, \pi)$ always accepts.

If $x \notin Q$, then $\mathrm{unsat}(\alpha_x) \geq \epsilon$, i.e., no matter what assignment π we have, at least ϵ-fraction of clauses are not satisfied.
We pick a number t with $0 \leq t < |x|^c$ in random, and let

$$p := (t \mod m) + 1$$

where m is the number of clauses in α_x. And let

$$s := \left\lfloor \frac{|x|^c}{m} \right\rfloor .$$

Proof. (cont'd)

Now we have to check V correctly decides Q.

If $x \in Q$, then $\text{unsat}(\alpha_x) = 0$.
There is an assignment π satisfy α_x. Thus no matter what τ we have, $V(x, \tau, \pi)$ always accepts.

If $x \notin Q$, then $\text{unsat}(\alpha_x) \geq \epsilon$, i.e., no matter what assignment π we have, at least ϵ-fraction of clauses are not satisfied.
We pick a number t with $0 \leq t < |x|^c$ in random, and let

$$p := (t \mod m) + 1$$

where m is the number of clauses in α_x. And let

$$s := \left\lfloor \frac{|x|^c}{m} \right\rfloor .$$

$s \geq 1$ by our assumption, and

$$s \cdot m \leq |x|^c \leq (s+1) \cdot m.$$

Proof. (cont'd)

Proof. (cont'd)

It follows that when t ranges from 0 to $|x|^c - 1$, for at least

$$\lceil \epsilon \cdot m \rceil \cdot s$$

many corresponding ps, the p-th clause is *not* satisfied.

Proof. (cont'd)

It follows that when t ranges from 0 to $|x|^c - 1$, for at least

$$\lceil \epsilon \cdot m \rceil \cdot s$$

many corresponding ps, the p-th clause is *not* satisfied.

$$\frac{\lceil \epsilon \cdot m \rceil \cdot s}{|x|^c} \geq \frac{s \cdot m}{(s+1) \cdot m} \cdot \epsilon \geq \frac{\epsilon}{2} \qquad \text{by } s \geq 1.$$

Proof. (cont'd)

It follows that when t ranges from 0 to $|x|^c - 1$, for at least

$$\lceil \epsilon \cdot m \rceil \cdot s$$

many corresponding ps, the p-th clause is *not* satisfied.

$$\frac{\lceil \epsilon \cdot m \rceil \cdot s}{|x|^c} \geq \frac{s \cdot m}{(s+1) \cdot m} \cdot \epsilon \geq \frac{\epsilon}{2} \qquad \text{by } s \geq 1.$$

Thus, the probability that the p-th clause is satisfied is less than

$$1 - \frac{\epsilon}{2}.$$

Proof. (cont'd)

It follows that when t ranges from 0 to $|x|^c - 1$, for at least

$$\lceil \epsilon \cdot m \rceil \cdot s$$

many corresponding ps, the p-th clause is *not* satisfied.

$$\frac{\lceil \epsilon \cdot m \rceil \cdot s}{|x|^c} \geq \frac{s \cdot m}{(s+1) \cdot m} \cdot \epsilon \geq \frac{\epsilon}{2} \qquad \text{by } s \geq 1.$$

Thus, the probability that the p-th clause is satisfied is less than

$$1 - \frac{\epsilon}{2}.$$

So the probability that V accepts is less than

$$\left(1 - \frac{\epsilon}{2}\right)^d,$$

hence less than $1/2$ by $(1 - \epsilon/2)^d \leq 1/2$.

□

Constraint Satisfaction Problems

Constraints

Constraints

Definition

Let $V = \{v_1, \ldots, v_n\}$ be a set of variables, and let Σ be a finite alphabet.

Constraints

Definition

Let $V = \{v_1, \ldots, v_n\}$ be a set of variables, and let Σ be a finite alphabet. A *q-ary constraint* $c = (C, i_1, \ldots, i_q)$ consists of

Constraints

Definition

Let $V = \{v_1, \ldots, v_n\}$ be a set of variables, and let Σ be a finite alphabet. A *q*-ary constraint $c = (C, i_1, \ldots, i_q)$ consists of

- a q-tuple of indices $i_1, \ldots, i_q \in [n]$,

Constraints

Definition

Let $V = \{v_1, \ldots, v_n\}$ be a set of variables, and let Σ be a finite alphabet. A q-ary constraint $c = (C, i_1, \ldots, i_q)$ consists of

- a q-tuple of indices $i_1, \ldots, i_q \in [n]$,
- and a subset $C \subseteq \Sigma^q$ of "acceptable" values.

Constraints

Definition

Let $V = \{v_1, \ldots, v_n\}$ be a set of variables, and let Σ be a finite alphabet. A q-ary constraint $c = (C, i_1, \ldots, i_q)$ consists of

- a q-tuple of indices $i_1, \ldots, i_q \in [n]$,
- and a subset $C \subseteq \Sigma^q$ of "acceptable" values.

An assignment $\mathcal{A} : V \to \Sigma$ *satisfies* c if

$$(\mathcal{A}(v_{i_1}), \ldots, \mathcal{A}(v_{i_q})) \in C.$$

Constraints

Definition

Let $V = \{v_1, \ldots, v_n\}$ be a set of variables, and let Σ be a finite alphabet. A <u>q-ary constraint</u> $c = (C, i_1, \ldots, i_q)$ consists of

- a q-tuple of indices $i_1, \ldots, i_q \in [n]$,
- and a subset $C \subseteq \Sigma^q$ of "acceptable" values.

An assignment $\mathcal{A} : V \to \Sigma$ *satisfies* c if

$$(\mathcal{A}(v_{i_1}), \ldots, \mathcal{A}(v_{i_q})) \in C.$$

The formula $X_3 \vee \neg X_7 \vee X_9$ can be viewed as a constraint $(C, 3, 7, 9)$ with

$$\{(1,0,0),(1,0,1),(1,1,0),(1,1,1),(0,0,0),(0,0,1),(0,1,1)\}.$$

Constraint Satisfaction Problems (CSP)

Constraint Satisfaction Problems (CSP)

CSP

Input: A system (i.e., set) $\mathscr{C}$ of constraints over a set of variables V and a finite alphabet Σ.

Problem: Is there an assignment $\mathcal{A} : V \to \Sigma$ that satisfies every constraint?

3SAT as CSP

3SAT as CSP

Lemma

3SAT *is reducible to* CSP*.*

3SAT as CSP

Lemma

3SAT *is reducible to* CSP*.*

Proof.

Let $\Sigma := \{0, 1\}$.

3SAT as CSP

Lemma

3SAT *is reducible to* CSP*.*

Proof.

Let $\Sigma := \{0, 1\}$. Given an instance α of 3SAT

$$\bigwedge_{i \in I} \lambda_{i,1} \vee \lambda_{i,2} \vee \lambda_{i,3},$$

let V be the set of propositional variables in α.

3SAT as CSP

Lemma

3SAT *is reducible to* CSP.

Proof.

Let $\Sigma := \{0, 1\}$. Given an instance α of 3SAT

$$\bigwedge_{i \in I} \lambda_{i,1} \vee \lambda_{i,2} \vee \lambda_{i,3},$$

let V be the set of propositional variables in α. For each $i \in I$, assume $\lambda_{i,j}$ is a literal over the variable X_{i_j} for $j = 1, 2, 3$.

3SAT as CSP

Lemma

3SAT *is reducible to* CSP*.*

Proof.

Let $\Sigma := \{0,1\}$. Given an instance α of 3SAT

$$\bigwedge_{i \in I} \lambda_{i,1} \vee \lambda_{i,2} \vee \lambda_{i,3},$$

let V be the set of propositional variables in α. For each $i \in I$, assume $\lambda_{i,j}$ is a literal over the variable X_{i_j} for $j = 1, 2, 3$. Then we introduce a constraint $c_i := (C_i, i_1, i_2, i_3)$

3SAT as CSP

Lemma

3SAT *is reducible to* CSP.

Proof.

Let $\Sigma := \{0, 1\}$. Given an instance α of 3SAT

$$\bigwedge_{i \in I} \lambda_{i,1} \vee \lambda_{i,2} \vee \lambda_{i,3},$$

let V be the set of propositional variables in α. For each $i \in I$, assume $\lambda_{i,j}$ is a literal over the variable X_{i_j} for $j = 1, 2, 3$. Then we introduce a constraint $c_i := (C_i, i_1, i_2, i_3)$ where

$$C_i := \big\{(b_1, b_2, b_3) \in \Sigma^3 \mid X_{i_1} \mapsto b_1,\ X_{i_2} \mapsto b_2,\ X_{i_3} \mapsto b_3 \\ \text{satisfies } \lambda_{i,1} \vee \lambda_{i,2} \vee \lambda_{i,3}\big\}.$$

3SAT as CSP

Lemma

3SAT *is reducible to* CSP.

Proof.

Let $\Sigma := \{0,1\}$. Given an instance α of 3SAT

$$\bigwedge_{i \in I} \lambda_{i,1} \vee \lambda_{i,2} \vee \lambda_{i,3},$$

let V be the set of propositional variables in α. For each $i \in I$, assume $\lambda_{i,j}$ is a literal over the variable X_{i_j} for $j = 1, 2, 3$. Then we introduce a constraint $c_i := (C_i, i_1, i_2, i_3)$ where

$$C_i := \big\{(b_1, b_2, b_3) \in \Sigma^3 \mid X_{i_1} \mapsto b_1,\ X_{i_2} \mapsto b_2,\ X_{i_3} \mapsto b_3 \text{ satisfies } \lambda_{i,1} \vee \lambda_{i,2} \vee \lambda_{i,3}\big\}.$$

Then

$$\alpha \text{ is satisfiable} \iff \text{there is an assignment satisfying } \{C_i \mid i \in I\}.$$

☐

3COLORABILITY as CSP

3COLORABILITY as CSP

Lemma

3COLORABILITY *is reducible to* CSP*.*

3Colorability as CSP

Lemma

3Colorability *is reducible to* CSP*.*

Proof.

Let $\Sigma := \{0, 1, 2\}$.

3Colorability as CSP

Lemma

3Colorability *is reducible to* CSP*.*

Proof.

Let $\Sigma := \{0, 1, 2\}$. Given a graph $G = (V, E)$, we identify V with the set of variables.

3COLORABILITY as CSP

Lemma

3COLORABILITY *is reducible to* CSP*.*

Proof.

Let $\Sigma := \{0, 1, 2\}$. Given a graph $G = (V, E)$, we identify V with the set of variables. For each edge $e \in E$ with end vertices v_i and v_j, we introduce a constraint

$$c_e := (C_e, i, j)$$

3COLORABILITY as CSP

Lemma

3COLORABILITY *is reducible to* CSP*.*

Proof.

Let $\Sigma := \{0, 1, 2\}$. Given a graph $G = (V, E)$, we identify V with the set of variables. For each edge $e \in E$ with end vertices v_i and v_j, we introduce a constraint

$$c_e := (C_e, i, j)$$

where

$$C_e := \{(a, b) \in \Sigma^2 \mid a \neq b\}.$$

3COLORABILITY as CSP

Lemma

3COLORABILITY *is reducible to* CSP*.*

Proof.

Let $\Sigma := \{0,1,2\}$. Given a graph $G = (V, E)$, we identify V with the set of variables. For each edge $e \in E$ with end vertices v_i and v_j, we introduce a constraint

$$c_e := (C_e, i, j)$$

where

$$C_e := \{(a,b) \in \Sigma^2 \mid a \neq b\}.$$

Then

$$G \text{ is 3-colorable} \iff \text{there is an assignment satisfying } \{C_e \mid e \in E\}.$$

☐