Expander Graphs and Their Applications (X)

Yijia Chen
Shanghai Jiaotong University

Review of the Previous Lecture

Probabilistic Verifier

Definition

A verifier V is a *probabilistic polynomial time algorithm* with access to an input $x \in \Sigma^*$ and a string $\tau \in \Sigma^*$ of (internal) *random binary bits*. Furthermore, V has access to a proof $\pi \in \Sigma^*$.

V will either accept or reject the input x, depending on (x, τ, π).

We require that V is non-adaptive, i.e., it first reads the input x and the random bits τ, and then decides which positions in the proof τ it wants to query. That is, the positions V queries do not depend on the answers that V got from *previous queries*.

The result of V's computation on x, τ and π is denoted by $V(x, \tau, \pi)$. As usual 1 means *accepting*, 0 for *rejecting*.

Probabilistic Verifier (cont'd)

1. V is polynomial time in $|x|$.
2. The query positions depend on and only on x and τ (non-adaptive), so we might view the query positions as a function

$$p_V : \Sigma^* \times \Sigma^* \to \mathbb{N}^*,$$

   such that $p_V(x, \tau)$ is computable in time polynomial in $|x|$. We may assume p_V is given *explicitly* with the verifier V.
3. The run of $V(x, \tau, \pi)$ that does not involve π might also depend on the random string τ.
4. We can divide the computation of $V(x, \tau, \pi)$ into 3 parts:
   (i) Compute from x and τ a sequence of positions $\bar{p} = p_1 p_2 \dots p_\ell = p_V(x, \tau)$.
   (ii) Read the letters on the positions $\bar{p}$ of π to form a string

$$\pi \upharpoonright \bar{p} := \pi(p_1)\pi(p_2)\dots\pi(p_\ell),$$

   where $\pi(p)$ denotes the letter in the p-th position of π.
   (iii) Compute an answer from x, τ, and $\pi \upharpoonright \bar{p}$.

$(r(n), q(n))$-Restricted Verifier

Definition

Let $r, q : \mathbb{N} \to \mathbb{N}$ be two monotone functions.

An $(r(n), q(n))$-restricted verifier is a verifier that for inputs of length n uses at most $O(r(n))$ random bits and queries at most $O(q(n))$ bits from the proof.

1. If V is $(r(n), q(n))$-restricted, then for any input $x \in \Sigma^*$, Then $|\tau| = O(r(n))$ and $|p_V(x, \tau)| = O(q(n))$.
2. There is no restriction on the length of the proof π.

PCP Classes

Definition

Let $r, q : \mathbb{N} \to \mathbb{N}$ be two monotone functions.
The class $\underline{\mathrm{PCP}(r(n), q(n))}$ consists of all languages Q where there exists an $(r(n), q(n))$-restricted verifier V such that for all x

$$x \in Q \iff \exists \pi \ \Pr_{\tau}[V(x, \tau, \pi) = 1] = 1,$$

$$x \notin Q \iff \forall \pi \ \Pr_{\tau}[V(x, \tau, \pi) = 1] < \frac{1}{2}.$$

Here in $\Pr_{\tau}[\ldots]$ the probability is taken over all random strings τ of length $O(r(n))$.

Remark. $1/2$ *in the above definition can be replaced by any* $0 < \varepsilon < 1$.

Definition

Let $R, Q \subseteq \mathbb{N} \to \mathbb{N}$ be two classes of monotone functions.

$$\underline{\mathrm{PCP}(R, Q)} := \bigcup_{r \in R, q \in Q} \mathrm{PCP}(r(n), q(n)).$$

Let $\mathtt{poly}$ denote the class of *all polynomials over* $\mathbb{N}$, i.e. $\mathbb{N}[x]$.

Theorem

$$\mathsf{PCP}(\mathtt{poly}, 0) = \mathsf{coRP},$$
$$\mathsf{PCP}(0, \mathtt{poly}) = \mathsf{NP}.$$

Max-dSat

Max-dSat

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

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

Max-dSat

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

Max-dSat	
Input:	An $\alpha \in d\text{Cnf}$.
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,

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

Max-dSat

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

Max-dSat	
Input:	An $\alpha \in d\text{Cnf}$.
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,

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

and

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

Hardness of Approximating MAX-dSAT

Definition

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

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

Hardness of Approximating Max-dSat

Definition

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

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

Theorem

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

Proof.

Proof.

Let $Q \subseteq \Sigma^*$ be an NP-complete language.

Proof.

Let $Q \subseteq \Sigma^*$ be an NP-complete language. By the PCP Theorem, there is a $(\log n, 1)$-restricted verifier V accepting Q:

Proof.

Let $Q \subseteq \Sigma^*$ be an NP-complete language. By the PCP Theorem, there is a $(\log n, 1)$-restricted verifier V accepting Q: for $x \in \Sigma^*$, V uses $c \cdot \log |x|$ random bits and makes d oracle queries for two constants $c, d \in \mathbb{N}$.

Proof.

Let $Q \subseteq \Sigma^*$ be an NP-complete language. By the PCP Theorem, there is a $(\log n, 1)$-restricted verifier V accepting Q: for $x \in \Sigma^*$, V uses $c \cdot \log |x|$ random bits and makes d oracle queries for two constants $c, d \in \mathbb{N}$.

$$\text{(R1)} \quad x \in Q \iff \exists \pi \Pr_{\tau \in \{0,1\}^{c \cdot \log |x|}} [V(x, \tau, \pi) = 1] = 1,$$

$$\text{(R2)} \quad x \notin Q \iff \forall \pi \Pr_{\tau \in \{0,1\}^{c \cdot \log |x|}} [V(x, \tau, \pi) = 1] < \frac{1}{2}.$$

Proof. (cont'd)

Proof. (cont'd)

We define the algorithm $\mathbb{A}(x, \tau, a_1, a_2, \ldots, a_d)$:

Proof. (cont'd)

We define the algorithm $\mathbb{A}(x, \tau, a_1, a_2, \ldots, a_d)$:

1. $\bar{p} = p_1 \ldots p_d \leftarrow p_V(x, \tau)$.

Proof. (cont'd)

We define the algorithm $\mathbb{A}(x, \tau, a_1, a_2, \ldots, a_d)$:

1. $\bar{p} = p_1 \ldots p_d \leftarrow p_V(x, \tau)$.
2. Simulate $V(x, \pi)$ for an "imaginary" π by replacing each π_{p_i} by a_i for $1 \leq i \leq d$.

Proof. (cont'd)

We define the algorithm $\mathbb{A}(x, \tau, a_1, a_2, \ldots, a_d)$:

1. $\bar{p} = p_1 \ldots p_d \leftarrow p_V(x, \tau)$.
2. Simulate $V(x, \pi)$ for an "imaginary" π by replacing each π_{p_i} by a_i for $1 \leq i \leq d$.

where $\tau \in \{0,1\}^{c \cdot \log |x|}$,

Proof. (cont'd)

We define the algorithm $\mathbb{A}(x, \tau, a_1, a_2, \ldots, a_d)$:

1. $\bar{p} = p_1 \ldots p_d \leftarrow p_V(x, \tau)$.
2. Simulate $V(x, \pi)$ for an "imaginary" π by replacing each π_{p_i} by a_i for $1 \leq i \leq d$.

where $\tau \in \{0,1\}^{c \cdot \log |x|}$, and $a_1, a_2, \ldots, a_d \in \{0,1\}$.

Proof. (cont'd)

We define the algorithm $\mathbb{A}(x, \tau, a_1, a_2, \ldots, a_d)$:

1. $\bar{p} = p_1 \ldots p_d \leftarrow p_V(x, \tau)$.
2. Simulate $V(x, \pi)$ for an "imaginary" π by replacing each π_{p_i} by a_i for $1 \leq i \leq d$.

where $\tau \in \{0,1\}^{c \cdot \log |x|}$, and $a_1, a_2, \ldots, a_d \in \{0,1\}$.

Clearly $\mathbb{A}$ is polynomial time in $|x|$.

Proof. (cont'd)

Proof. (cont'd)

We reduce Q to Max-dSat.

Proof. (cont'd)

We reduce Q to MAX-dSAT.

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

Proof. (cont'd)

We reduce Q to MAX-dSAT.

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

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

Proof. (cont'd)

We reduce Q to MAX-dSAT.

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

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

First let

$$P_x := \left\{ p \mid \text{for some } \tau \in \{0,1\}^{c \cdot \log |x|} \; p \text{ appears in the sequence } p_V(x, \tau) \right\},$$

Proof. (cont'd)

We reduce Q to MAX-dSAT.

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

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

First let

$$P_x := \left\{ p \mid \text{for some } \tau \in \{0,1\}^{c \cdot \log |x|} \; p \text{ appears in the sequence } p_V(x, \tau) \right\},$$

i.e. P_x is the set of positions that V makes queries.

Proof. (cont'd)

We reduce Q to MAX-dSAT.

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

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

First let

$$P_x := \left\{ p \mid \text{for some } \tau \in \{0,1\}^{c \cdot \log |x|} \ p \text{ appears in the sequence } p_V(x, \tau) \right\},$$

i.e. P_x is the set of positions that V makes queries.
Obviously P_x is computable in time polynomial in $|x|$.

Proof. (cont'd)

We reduce Q to MAX-dSAT.

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

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

First let

$$P_x := \left\{ p \mid \text{for some } \tau \in \{0,1\}^{c \cdot \log |x|} \; p \text{ appears in the sequence } p_V(x,\tau) \right\},$$

i.e. P_x is the set of positions that V makes queries.
Obviously P_x is computable in time polynomial in $|x|$. In addition

$$|P_x| \leq d \cdot |x|^c.$$

Proof. (cont'd)

We reduce Q to MAX-dSAT.

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

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

First let

$$P_x := \left\{ p \mid \text{for some } \tau \in \{0,1\}^{c \cdot \log |x|} \; p \text{ appears in the sequence } p_V(x,\tau) \right\},$$

i.e. P_x is the set of positions that V makes queries.
Obviously P_x is computable in time polynomial in $|x|$. In addition

$$|P_x| \leq d \cdot |x|^c.$$

For each $p \in P_x$, we introduce a propositional variable X_p.

Proof. (cont'd)

We reduce Q to MAX-dSAT.

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

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

First let

$$P_x := \left\{ p \mid \text{for some } \tau \in \{0,1\}^{c \cdot \log |x|} \ p \text{ appears in the sequence } p_V(x,\tau) \right\},$$

i.e. P_x is the set of positions that V makes queries.
Obviously P_x is computable in time polynomial in $|x|$. In addition

$$|P_x| \leq d \cdot |x|^c.$$

For each $p \in P_x$, we introduce a propositional variable X_p. Then, every proof π can be viewed as an assignment $\mathcal{V}_\pi$ with

$$\mathcal{V}_\pi(X_p) := \pi(p).$$

Proof. (cont'd)

Proof. (cont'd)

For every $\tau \in \{0,1\}^{c \cdot \log |x|}$, we define a $d\textsc{Cnf}$ formula $\alpha_{x,\tau}$:

Proof. (cont'd)

For every $\tau \in \{0,1\}^{c \cdot \log |x|}$, we define a $d\mathrm{CNF}$ formula $\alpha_{x,\tau}$: For every $a_1, \ldots,$ $a_d \in \{0,1\}$, and $\bar{p} = p_1 \ldots p_d = p_V(x,\tau)$,

Proof. (cont'd)

For every $\tau \in \{0,1\}^{c \cdot \log |x|}$, we define a $d\mathrm{CNF}$ formula $\alpha_{x,\tau}$: For every $a_1, \ldots, a_d \in \{0,1\}$, and $\bar{p} = p_1 \ldots p_d = p_V(x,\tau)$, if $\mathbb{A}(x,\tau,a_1,\ldots,a_d)$ *rejects*, then $\alpha_{x,\tau}$ contains the clause

$$T_1 \vee T_2 \vee \cdots \vee T_d$$

where

$$T_i := \begin{cases} X_{p_i} & \text{if } a_i = 0, \\ \neg X_{p_i} & \text{if } a_i = 1. \end{cases}$$

Proof. (cont'd)

For every $\tau \in \{0,1\}^{c \cdot \log |x|}$, we define a $d\mathrm{CNF}$ formula $\alpha_{x,\tau}$: For every $a_1, \ldots, a_d \in \{0,1\}$, and $\bar{p} = p_1 \ldots p_d = p_V(x,\tau)$, if $\mathbb{A}(x,\tau,a_1,\ldots,a_d)$ *rejects*, then $\alpha_{x,\tau}$ contains the clause

$$T_1 \vee T_2 \vee \cdots \vee T_d$$

where

$$T_i := \begin{cases} X_{p_i} & \text{if } a_i = 0, \\ \neg X_{p_i} & \text{if } a_i = 1. \end{cases}$$

$\alpha_{x,\tau}$ has *at most* 2^d *clauses*.

Proof. (cont'd)

For every $\tau \in \{0,1\}^{c \cdot \log |x|}$, we define a $d\mathrm{CNF}$ formula $\alpha_{x,\tau}$: For every $a_1, \ldots, a_d \in \{0,1\}$, and $\bar{p} = p_1 \ldots p_d = p_V(x,\tau)$, if $\mathbb{A}(x,\tau,a_1,\ldots,a_d)$ *rejects*, then $\alpha_{x,\tau}$ contains the clause

$$T_1 \vee T_2 \vee \cdots \vee T_d$$

where

$$T_i := \begin{cases} X_{p_i} & \text{if } a_i = 0, \\ \neg X_{p_i} & \text{if } a_i = 1. \end{cases}$$

$\alpha_{x,\tau}$ has *at most* 2^d *clauses*. More importantly: Let π be a proof with corresponding assignment $\mathcal{V}_\pi$

(F1) if $V(x,\tau,\pi)$ accepts, then $\mathcal{V}_\pi$ satisfies $\alpha_{x,\tau}$,

(F2) if $V(x,\tau,\pi)$ rejects,
then $\mathcal{V}_\pi$ makes at least *one* of the clauses in $\alpha_{x,\tau}$ *false*.

Proof. (cont'd)

Proof. (cont'd)

Let

$$\alpha_x := \bigwedge_{\tau \in \{0,1\}^{c \cdot \log |x|}} \alpha_{x,\tau}$$

Proof. (cont'd)

Let

$$\alpha_x := \bigwedge_{\tau \in \{0,1\}^{c \cdot \log |x|}} \alpha_{x,\tau}$$

α_x has *at most* $|x|^c \cdot 2^d$ *clauses*.

Proof. (cont'd)

Let

$$\alpha_x := \bigwedge_{\tau \in \{0,1\}^{c \cdot \log |x|}} \alpha_{x,\tau}$$

α_x has *at most* $|x|^c \cdot 2^d$ *clauses*. We need to show:

Proof. (cont'd)

Let

$$\alpha_x := \bigwedge_{\tau \in \{0,1\}^{c \cdot \log |x|}} \alpha_{x,\tau}$$

α_x has *at most* $|x|^c \cdot 2^d$ *clauses*. We need to show:

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

Proof. (cont'd)

Let $x \in Q$.

Proof. (cont'd)

Let $x \in Q$. By

$$\text{(R1)} \quad x \in Q \iff \exists \pi \Pr_{\tau \in \{0,1\}^{c \cdot \log |x|}} [V(x, \tau, \pi) = 1] = 1,$$

we have a proof π such that $V(x, \tau, \pi)$ always accepts for any $\tau \in \{0,1\}^{c \cdot \log |x|}$.

Proof. (cont'd)

Let $x \in Q$. By

$$\text{(R1)} \quad x \in Q \iff \exists \pi \Pr_{\tau \in \{0,1\}^{c \cdot \log |x|}} [V(x, \tau, \pi) = 1] = 1,$$

we have a proof π such that $V(x, \tau, \pi)$ always accepts for any $\tau \in \{0,1\}^{c \cdot \log |x|}$.

Thus $\mathcal{V}_\pi$ satisfies all $\alpha_{x,\tau}$ by

$$\text{(F1)} \quad \text{if } V(x, \tau, \pi) \text{ accepts, then } \mathcal{V}_\pi \text{ satisfies } \alpha_{x,\tau},$$

Proof. (cont'd)

Let $x \in Q$. By

$$\text{(R1)} \quad x \in Q \iff \exists \pi \Pr_{\tau \in \{0,1\}^{c \cdot \log |x|}} [V(x, \tau, \pi) = 1] = 1,$$

we have a proof π such that $V(x, \tau, \pi)$ always accepts for any $\tau \in \{0,1\}^{c \cdot \log |x|}$.

Thus $\mathcal{V}_\pi$ satisfies all $\alpha_{x,\tau}$ by

$$\text{(F1)} \quad \text{if } V(x, \tau, \pi) \text{ accepts, then } \mathcal{V}_\pi \text{ satisfies } \alpha_{x,\tau},$$

and hence α_x, i.e.,

$$x \in Q \Rightarrow \alpha_x \text{ is satisfiable.}$$

Proof. (cont'd)

Proof. (cont'd)

Let $x \notin Q$.

Proof. (cont'd)

Let $x \notin Q$. For any assignment $\mathcal{V}$ of α_x, it is easy to see we can view it as some $\mathcal{V}_\pi$ for an appropriate proof π.

Proof. (cont'd)

Let $x \notin Q$. For any assignment $\mathcal{V}$ of α_x, it is easy to see we can view it as some $\mathcal{V}_\pi$ for an appropriate proof π.

The by

$$\text{(R2)} \quad x \notin Q \iff \forall \pi \Pr_{\tau \in \{0,1\}^{c \cdot \log |x|}} [V(x, \tau, \pi) = 1] < \frac{1}{2}.$$

we have for *more than half* of all possible $\tau \in \{0,1\}^{c \cdot \log |x|}$, $V(x, \tau, \pi)$ rejects.

Proof. (cont'd)

Let $x \notin Q$. For any assignment $\mathcal{V}$ of α_x, it is easy to see we can view it as some $\mathcal{V}_\pi$ for an appropriate proof π.

The by

$$\text{(R2)} \quad x \notin Q \iff \forall \pi \Pr_{\tau \in \{0,1\}^{c \cdot \log |x|}} [V(x, \tau, \pi) = 1] < \frac{1}{2}.$$

we have for *more than half* of all possible $\tau \in \{0,1\}^{c \cdot \log |x|}$, $V(x, \tau, \pi)$ rejects. For any τ with $V(x, \tau, \pi)$ rejecting, then

(F2) if $V(x, \tau, \pi)$ rejects,
then $\mathcal{V}_\pi$ makes at least *one* of the clauses in $\alpha_{x,\tau}$ *false*.

implies that *at least one clause* in $\alpha_{x,\tau}$ is false under $\mathcal{V}_\pi = \mathcal{V}$.

Proof. (cont'd)

Let $x \notin Q$. For any assignment $\mathcal{V}$ of α_x, it is easy to see we can view it as some $\mathcal{V}_\pi$ for an appropriate proof π.

The by

$$\text{(R2)} \quad x \notin Q \iff \forall \pi \Pr_{\tau \in \{0,1\}^{c \cdot \log |x|}} [V(x, \tau, \pi) = 1] < \frac{1}{2}.$$

we have for *more than half* of all possible $\tau \in \{0,1\}^{c \cdot \log |x|}$, $V(x, \tau, \pi)$ rejects. For any τ with $V(x, \tau, \pi)$ rejecting, then

(F2) if $V(x, \tau, \pi)$ rejects,
then $\mathcal{V}_\pi$ makes at least *one* of the clauses in $\alpha_{x,\tau}$ *false*.

implies that *at least one clause* in $\alpha_{x,\tau}$ is false under $\mathcal{V}_\pi = \mathcal{V}$.
Recall $\alpha_{x,\tau}$ contains at most 2^d many clauses. Thus in total we have *more than*

$$2^{-1} \cdot 2^{-d} = 2^{-d-1}$$

fraction of clauses in α_x are not satisfied under $\mathcal{V}$.

Proof. (cont'd)

Proof. (cont'd)

Let

$$r := \frac{1}{1 - 2^{-d-1}}.$$

Proof. (cont'd)

Let

$$r := \frac{1}{1 - 2^{-d-1}}.$$

We claim MAX-dSAT *cannot* have a polynomial time approximation algorithm with performance ratio r.

Proof. (cont'd)

Let

$$r := \frac{1}{1 - 2^{-d-1}}.$$

We claim MAX-dSAT *cannot* have a polynomial time approximation algorithm with performance ratio r.

Assume $\mathbb{B}$ is such an algorithm. We would use it to decide Q:

Proof. (cont'd)

Let

$$r := \frac{1}{1 - 2^{-d-1}}.$$

We claim MAX-dSAT *cannot* have a polynomial time approximation algorithm with performance ratio r.

Assume $\mathbb{B}$ is such an algorithm. We would use it to decide Q:

Let $x \in \Sigma^*$.

Proof. (cont'd)

Let

$$r := \frac{1}{1 - 2^{-d-1}}.$$

We claim MAX-dSAT *cannot* have a polynomial time approximation algorithm with performance ratio r.

Assume $\mathbb{B}$ is such an algorithm. We would use it to decide Q:

Let $x \in \Sigma^*$. We compute the dCNF-formula α_x as described above.

Proof. (cont'd)

Let

$$r := \frac{1}{1 - 2^{-d-1}}.$$

We claim MAX-dSAT *cannot* have a polynomial time approximation algorithm with performance ratio r.

Assume $\mathbb{B}$ is such an algorithm. We would use it to decide Q:

Let $x \in \Sigma^*$. We compute the dCNF-formula α_x as described above. Assume α_x has m clauses.

Proof. (cont'd)

Let

$$r := \frac{1}{1 - 2^{-d-1}}.$$

We claim $\textsc{Max}$-$d\textsc{Sat}$ *cannot* have a polynomial time approximation algorithm with performance ratio r.

Assume $\mathbb{B}$ is such an algorithm. We would use it to decide Q:

Let $x \in \Sigma^*$. We compute the $d\textsc{Cnf}$-formula α_x as described above. Assume α_x has m clauses.

If $x \in Q$, then α_x is satisfiable.

Proof. (cont'd)

Let

$$r := \frac{1}{1 - 2^{-d-1}}.$$

We claim MAX-dSAT *cannot* have a polynomial time approximation algorithm with performance ratio r.

Assume $\mathbb{B}$ is such an algorithm. We would use it to decide Q:

Let $x \in \Sigma^*$. We compute the dCNF-formula α_x as described above. Assume α_x has m clauses.

If $x \in Q$, then α_x is satisfiable. Thus $\mathbb{B}$ would find an assignment that satisfies at least

$$\frac{m}{r} = m \cdot (1 - 2^{-d-1})$$

clauses of α_x.

Proof. (cont'd)

Proof. (cont'd)

If $x \notin Q$, then less than

$$m \cdot (1 - 2^{-d-1})$$

clauses of α_x can be satisfied by any assignment.

Proof. (cont'd)

If $x \notin Q$, then less than

$$m \cdot (1 - 2^{-d-1})$$

clauses of α_x can be satisfied by any assignment. In particular the assignment that $\mathbb{B}$ outputs on α_x cannot satisfy $m \cdot (1 - 2^{-d-1})$ clauses.

Proof. (cont'd)

If $x \notin Q$, then less than

$$m \cdot (1 - 2^{-d-1})$$

clauses of α_x can be satisfied by any assignment. In particular the assignment that $\mathbb{B}$ outputs on α_x cannot satisfy $m \cdot (1 - 2^{-d-1})$ clauses.

In summary:

$$\begin{aligned} x \in Q \quad &\Rightarrow \quad \mathbb{B}(\alpha_x) \text{ satisfies } \geq m \cdot (1 - 2^{-d-1}) \text{ clauses;} \\ x \notin Q \quad &\Rightarrow \quad \mathbb{B}(\alpha_x) \text{ satisfies } < m \cdot (1 - 2^{-d-1}) \text{ clauses.} \end{aligned}$$

□