Course "Proofs and Computers", JASS'06

Probabilistically Checkable Proofs

Lukas Bulwahn

Faculty of Computer Science
TU Munich

March 30, 2006

- 1974: Foundational Paper from Johnson states approximation algorithms and inapproximability results for Max SAT, Set Cover, Independent Set, and Coloring.

- 1974: Foundational Paper from Johnson states approximation algorithms and inapproximability results for Max SAT, Set Cover, Independent Set, and Coloring.
- 1988: Ben-Or, Goldwasser et al. working on multi-prover interactive proofs.

- 1974: Foundational Paper from Johnson states approximation algorithms and inapproximability results for Max SAT, Set Cover, Independent Set, and Coloring.
- 1988: Ben-Or, Goldwasser et al. working on multi-prover interactive proofs.
- 1991: Feige, Goldwasser et al. created a new model for NP, namely PCPs.

- 1974: Foundational Paper from Johnson states approximation algorithms and inapproximability results for Max SAT, Set Cover, Independent Set, and Coloring.
- 1988: Ben-Or, Goldwasser et al. working on multi-prover interactive proofs.
- 1991: Feige, Goldwasser et al. created a new model for NP, namely PCPs.
- 1992: Arora proved the PCP Theorem.

- 1974: Foundational Paper from Johnson states approximation algorithms and inapproximability results for Max SAT, Set Cover, Independent Set, and Coloring.
- 1988: Ben-Or, Goldwasser et al. working on multi-prover interactive proofs.
- 1991: Feige, Goldwasser et al. created a new model for NP, namely PCPs.
- 1992: Arora proved the PCP Theorem.
- Many inapproximability results for problems were found in the mid 1990s.

- 1974: Foundational Paper from Johnson states approximation algorithms and inapproximability results for Max SAT, Set Cover, Independent Set, and Coloring.
- 1988: Ben-Or, Goldwasser et al. working on multi-prover interactive proofs.
- 1991: Feige, Goldwasser et al. created a new model for NP, namely PCPs.
- 1992: Arora proved the PCP Theorem.
- Many inapproximability results for problems were found in the mid 1990s.
- 2005: Dinur shows a new proof for the PCP Theorem.

Definition 1

A verifier V is a (r,q)-restricted verifier if for any input x, witness w, and random string τ of length $O(r)$, the decision $V^w(x,\tau) = $ "*yes*" is based on at most $O(q)$ bits from the witness w.

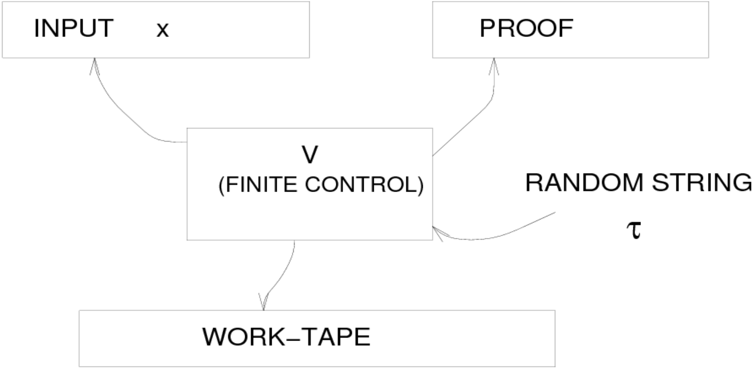
Definition 1

A verifier V is a (r,q)-restricted verifier if for any input x, witness w, and random string τ of length $O(r)$, the decision $V^w(x,\tau) = \text{"}yes\text{"}$ is based on at most $O(q)$ bits from the witness w.

Remarks:

A (r,q)-restricted verifier is called non-adaptive if the queries to the witness w only depend on the input x and the random string τ. If the next queries are also dependant from the previous queries from the witness w, the verifier is called adaptive.

A (r,q)-restricted verifier



Definition 2

A language L is probabilistically checkable using an (r,q)-restricted verifier V iff

Definition 2

A language L is probabilistically checkable using an (r,q)-restricted verifier V iff

- Completeness: If $x \in L$ then there exists a witness w such that $Pr_\tau[V^w(x,\tau) = "yes"] = 1$.

Definition 2

A language L is probabilistically checkable using an (r,q)-restricted verifier V iff

- Completeness: If $x \in L$ then there exists a witness w such that $Pr_\tau[V^w(x,\tau) = \text{"}yes\text{"}] = 1$.
- Soundness: If $x \notin L$ then for every witness w we have $Pr_\tau[V^w(x,\tau) = \text{"}yes\text{"}] < 1/2$.

Example 3

Some simple examples for PCP-Classes are:

Example 3

Some simple examples for PCP-Classes are:

- $\mathrm{P} = \mathrm{PCP}(0, 0)$

Example 3

Some simple examples for PCP-Classes are:

- P = PCP(0, 0)
- NP = PCP(0, poly)

Example 3

Some simple examples for PCP-Classes are:

- $P = PCP(0, 0)$
- $NP = PCP(0, poly)$
- $NP \subseteq PCP(\log, poly)$

Example 3

Some simple examples for PCP-Classes are:

- $\mathrm{P} = \mathrm{PCP}(0, 0)$
- $\mathrm{NP} = \mathrm{PCP}(0, \mathrm{poly})$
- $\mathrm{NP} \subseteq \mathrm{PCP}(\log, \mathrm{poly})$
- $\text{co-RP} = \mathrm{PCP}(\mathrm{poly}, 0)$

Example 3

Some simple examples for PCP-Classes are:

- $\mathrm{P} = \mathrm{PCP}(0, 0)$
- $\mathrm{NP} = \mathrm{PCP}(0, \mathrm{poly})$
- $\mathrm{NP} \subseteq \mathrm{PCP}(\log, \mathrm{poly})$
- $\text{co-RP} = \mathrm{PCP}(\mathrm{poly}, 0)$

Example 3

Some simple examples for PCP-Classes are:

- $\mathrm{P} = \mathrm{PCP}(0, 0)$
- $\mathrm{NP} = \mathrm{PCP}(0, \mathrm{poly})$
- $\mathrm{NP} \subseteq \mathrm{PCP}(\log, \mathrm{poly})$
- $\text{co-RP} = \mathrm{PCP}(\mathrm{poly}, 0)$

Proofs are easily shown by constructing verifiers which reduces the one class on the other and vice versa.

Definition 4 (PCP Theorem)

$NP = PCP(log(n), 1)$

First, we will prove the easier side of the PCP theorem:
PCP(log(n), 1) $\subseteq$ NP.

First, we will prove the easier side of the PCP theorem: PCP(log(n), 1) $\subseteq$ NP.

Proof.

Let L $\in$ PCP(log(n), 1) $\Rightarrow$ there is a (log(n), 1)-verifier V.
For τ there are $2^{O(log(n))} \leq n^c$ many random strings, namely $\tau^1, \cdots, \tau^{n^c}$.

First, we will prove the easier side of the PCP theorem:
PCP(log(n), 1) $\subseteq$ NP.

Proof.

Let L $\in$ PCP(log(n), 1) $\Rightarrow$ there is a (log(n), 1)-verifier V.
For τ there are $2^{O(log(n))} \leq n^c$ many random strings, namely $\tau^1, \cdots, \tau^{n^c}$.
The verifier V will work as follows:

1. Reads a random string $\tau^i, 1 \leq \ i \leq n^c$.
2. Uses x and τ^i to calculate q positions $i_1, \cdots, i_q$ to read from the witness string.
3. Run a calculation with x and $w_{i_1}, \cdots, w_{i_q}$, and answer "yes" or "no".

Proof.

Now, we will simulate the verifier V on a non-deterministic Turing machine V'.

The witness string for V' is w which has polynomial length since V can only access polynomial positions.

V' now calculates step 2 and 3 from V for every possible τ^i and answers "yes" if all simulated calculations of V answered "yes".

Proof.

Now, we will simulate the verifier V on a non-deterministic Turing machine V'.

The witness string for V' is w which has polynomial length since V can only access polynomial positions.

V' now calculates step 2 and 3 from V for every possible τ^i and answers "yes" if all simulated calculations of V answered "yes".

It is left to show that V' behaves like V.

Proof.

$\square$

Proof.

- $x \in L$ and $L \in PCP(log(n), 1) \Rightarrow$ For a given w, V returns yes with probability 1.
  With this witness w V' will also return yes.

□

Proof.

- $x \in L$ and $L \in PCP(log(n), 1) \Rightarrow$ For a given w, V returns yes with probability 1.
  With this witness w V' will also return yes.
- $x \notin L \Rightarrow$ There is no witness string w for V' because at least on half of the calculations will not answer "yes".

$\square$

Definition 5

An optimization problem O is defined by a cost function $C : \Sigma^* \times \Sigma^* \rightarrow R_+ \cup \{\perp\}$, that given an instance string x and a solution string s outputs $C(x, s)$ which is either the cost of the solution or $\perp$ if the solution is illegal.

Definition 5

An optimization problem O is defined by a cost function $C : \Sigma^* \times \Sigma^* \to R_+ \cup \{\perp\}$, that given an instance string x and a solution string s outputs $C(x, s)$ which is either the cost of the solution or $\perp$ if the solution is illegal.
Let $OPT(x)$ denote the optimal value a solution can get, then: $OPT(x) = max_{s:C(x,s)\neq\perp} C(x, s)$.

Definition 5

An optimization problem O is defined by a cost function $C : \Sigma^* \times \Sigma^* \rightarrow R_+ \cup \{\perp\}$, that given an instance string x and a solution string s outputs $C(x, s)$ which is either the cost of the solution or $\perp$ if the solution is illegal.
Let $OPT(x)$ denote the optimal value a solution can get, then: $OPT(x) = max_{s:C(x,s)\neq\perp} C(x, s)$.
An optimization problem is to find a legal solution s^* that attains the optimal value of cost, $C(x, s^*) = OPT(x)$.

Example 6

MAX-3SAT is the problem of finding an assignment A which maximizes the percent of satisfied clauses of a 3CNF formula ψ. Of course, if ψ is satisfiable, then the optimal value of MAX-3SAT is 1.

Definition 7

A is an r-approximation algorithm for a maximation problem iff for any input x, A finds a solution s that $C(x, s) \geq rOPT(x)$.

Definition 8

Let O be a maximization problem. Let x be an instance of the problem. A gap(α, β)-O is the problem of deciding between the following alternatives:

Definition 8

Let O be a maximization problem. Let x be an instance of the problem. A gap(α, β)-O is the problem of deciding between the following alternatives:

Definition 8

Let O be a maximization problem. Let x be an instance of the problem. A gap(α, β)-O is the problem of deciding between the following alternatives:

- "Yes": $OPT(x) \geq \beta$

Definition 8

Let O be a maximization problem. Let x be an instance of the problem. A gap(α, β)-O is the problem of deciding between the following alternatives:

- "Yes": $OPT(x) \geq \beta$

Definition 8

Let O be a maximization problem. Let x be an instance of the problem. A gap(α, β)-O is the problem of deciding between the following alternatives:

- "Yes": $OPT(x) \geq \beta$
- "No": $OPT(x) \leq \alpha$

Definition 8

Let O be a maximization problem. Let x be an instance of the problem. A gap(α, β)-O is the problem of deciding between the following alternatives:

- "Yes": $OPT(x) \geq \beta$
- "No": $OPT(x) \leq \alpha$

Definition 8

Let O be a maximization problem. Let x be an instance of the problem. A gap(α, β)-O is the problem of deciding between the following alternatives:

- "Yes": $OPT(x) \geq \beta$
- "No": $OPT(x) \leq \alpha$

If $OPT \in [\alpha, \beta)$ then both alternatives are acceptable.

Definition 8

Let O be a maximization problem. Let x be an instance of the problem. A gap(α, β)-O is the problem of deciding between the following alternatives:

- "Yes": $OPT(x) \geq \beta$
- "No": $OPT(x) \leq \alpha$

If $OPT \in [\alpha, \beta)$ then both alternatives are acceptable.

If the gap problem is NP-hard, then it is the $\frac{\alpha}{\beta}$-approximation algorithm is also NP-hard.

Lemma 9

The following statements are equivalent:

1. *(PCP Theorem)* $NP = PCP(log(n), 1)$
2. *There exists* $\alpha \in (0, 1)$, *such that gap(*α*, 1)-MAX-3SAT is NP hard.*

Proof.

$(2 \Rightarrow 1)$

Let language $L \in NP$.

Assumption: gap(α, 1)-MAX- 3SAT is NP hard.

Proof.

$(2 \Rightarrow 1)$

Let language $L \in NP$.

Assumption: gap(α, 1)-MAX- 3SAT is NP hard.

Proof.

$(2 \Rightarrow 1)$

Let language $L \in NP$.

Assumption: gap(α, 1)-MAX- 3SAT is NP hard.

$\Longrightarrow$ there exists a 3CNF formula, $\psi_{x,L} = c_1 \wedge \ldots \wedge c_m$, such that

1. $x \in L \Leftrightarrow \psi_{x,L}$ is satisfiable.

Proof.

$(2 \Rightarrow 1)$

Let language $L \in NP$.

Assumption: gap(α, 1)-MAX- 3SAT is NP hard.

$\Longrightarrow$ there exists a 3CNF formula, $\psi_{x,L} = c_1 \wedge \ldots \wedge c_m$, such that

1. $x \in L \Leftrightarrow \psi_{x,L}$ is satisfiable.
2. $x \notin L \Leftrightarrow$ for every assignment A, the number of clauses in $\psi_{x,L}$ that are satisfied is less than αm.

The verifier V can use the following algorithm to check if a string x is in the language L:

Algorithm

The verifier V can use the following algorithm to check if a string x is in the language L:

Algorithm

1. step: Construct the 3CNF formula $\psi_{x,L}$.

The verifier V can use the following algorithm to check if a string x is in the language L:

Algorithm

1. step: Construct the 3CNF formula $\psi_{x,L}$.
2. step: Get an assignment A and create witness/proof $w = \psi_{x,L} \circ A$.

The verifier V can use the following algorithm to check if a string x is in the language L:

Algorithm

1. step: Construct the 3CNF formula $\psi_{x,L}$.
2. step: Get an assignment A and create witness/proof $w = \psi_{x,L} \circ A$.
3. step: Choose $k = O(1)$ clauses from the witness.

The verifier V can use the following algorithm to check if a string x is in the language L:

Algorithm

1. step: Construct the 3CNF formula $\psi_{x,L}$.
2. step: Get an assignment A and create witness/proof $w = \psi_{x,L} \circ A$.
3. step: Choose $k = O(1)$ clauses from the witness.
4. step: If all k clauses are satisfied, return "yes".

Proof.

Proof.

- Completeness:
  If the assignment A satisfies the formula, V will answer "yes" no matter which k clauses were chosen. $\surd$

Proof.

- Completeness:
  If the assignment A satisfies the formula, V will answer "yes" no matter which k clauses were chosen.$\surd$
- Soundness:
  If A does not satisfy $\psi_{x,L}$, then it satisfies at most αm clauses $\Longrightarrow$the probability to answer "yes" is at most α^k.

Proof.

- Completeness:
  If the assignment A satisfies the formula, V will answer "yes" no matter which k clauses were chosen. $\surd$
- Soundness:
  If A does not satisfy $\psi_{x,L}$, then it satisfies at most αm clauses $\Longrightarrow$ the probability to answer "yes" is at most α^k.

Proof.

- Completeness:
  If the assignment A satisfies the formula, V will answer "yes" no matter which k clauses were chosen.$\surd$
- Soundness:
  If A does not satisfy $\psi_{x,L}$, then it satisfies at most αm clauses $\Longrightarrow$the probability to answer "yes" is at most α^k.
  With $k > log(1/2)/log(\alpha) : Pr_\tau[V^w x, \tau = "yes"] \leq \alpha^k \Longrightarrow$ $Pr_\tau[V^w(x,\tau) = "yes"] < 1/2.\surd$

$\square$

Proof.

$(1 \Rightarrow 2)$

Proof by reduction from gap-MAX-3SAT to 3SAT.

$3SAT \in NP \Longrightarrow 3SAT \in PCP[log, 1]$

$\Longrightarrow$ there exists a verifier V such that a given 3CNF formula ϕ:

Proof.

$(1 \Rightarrow 2)$

Proof by reduction from gap-MAX-3SAT to 3SAT.

$3SAT \in NP \Longrightarrow 3SAT \in PCP[log, 1]$

$\Longrightarrow$ there exists a verifier V such that a given 3CNF formula ϕ:

- ϕ is satisfiable $\Rightarrow \exists w : Pr_\tau[V^w(\phi, \tau) = "yes"] = 1$.

Proof.

($1 \Rightarrow 2$)

Proof by reduction from gap-MAX-3SAT to 3SAT.

$3SAT \in NP \implies 3SAT \in PCP[log, 1]$

$\implies$ there exists a verifier V such that a given 3CNF formula ϕ:

- ϕ is satisfiable $\Rightarrow \exists w : Pr_\tau[V^w(\phi, \tau) = "yes"] = 1$.
- ϕ is not satisfiable $\Rightarrow \forall w : Pr_\tau[V^w(\phi, \tau) = "yes"] < 1/2$.

Proof.

The verifier only considers q bits of the witness *w* for its decision. $\Rightarrow$ acceptance is determined with local constraint ψ_τ^ϕ and variable assignment according to the positions in the witness *w*. It is still true that:

- ϕ is satisfiable $\Rightarrow$ all local constraints ψ_τ^ϕ are satisfied.
- ϕ is not satisfiable $\Rightarrow \forall$ assignments A at most half of the local constraints are satisfied.

Proof.

Construct a new formula $\phi' = \psi^{\phi}_{\tau_1} \wedge \cdots \wedge \psi^{\phi}_{\tau^c_n}$ with $\tau_1, \cdots \tau_{n^c}$ are all random string of the length $O(log(n))$. For ϕ', we have:

- ϕ is satisfiable $\Rightarrow$ ϕ' is satisfied.
- ϕ is not satisfiable $\Rightarrow$ any assignment for ϕ' satisfies at most half of the clauses of ϕ'.

Proof.

Construct a 3CNF formula from each local constraint.

$$\phi'' = \underbrace{(\psi_{1,1} \wedge \ldots \wedge \psi_{1,k})}_{\psi^{\phi}_{\tau_1}} \wedge \ldots \wedge \underbrace{(\psi_{n^c,1} \wedge \ldots \wedge \psi_{n^c,k})}_{\psi^{\phi}_{\tau_{n^c}}}$$

- ϕ is satisfiable $\Rightarrow$ ϕ'' is satisfied.
- ϕ is not satisfiable $\Rightarrow$ any assignment for ϕ' satisfies at most $\frac{2k-1}{2k}$ of the clauses of ϕ''.

This conclues the reduction from gap(α,1)-MAX-3SAT for $\alpha = \frac{2k-1}{2k}$ to 3SAT assuming the PCP Theorem. $\square$

Theorem 10 (John Hastad, 1997)

For any $\alpha \in \left(\frac{7}{8}, 1\right)$, the problem gap($\alpha$,1)-MAX-3SAT is NP-hard.

Fact

But notice this interesting fact:
Howard Karloff and Uri Zwick have stated a $\frac{7}{8}$-Approximation Algorithm for MAX-3-SAT
and provided strong evidence that the algorithm performs equally well on arbitrary MAX-3-SAT instances.

Definition 11

$G = \langle (V, E), \Sigma, C \rangle$ is called a constraint graph, if

Definition 11

$G = \langle (V, E), \Sigma, C \rangle$ is called a constraint graph, if

1. (V,E) is an undirected graph, called the underlying graph of G.

Definition 11

$G = \langle (V, E), \Sigma, C \rangle$ is called a constraint graph, if

1. (V,E) is an undirected graph, called the underlying graph of G.
2. The set V is also viewed as a set of variables assuming values over alphabet Σ.

Definition 11

$G = \langle (V, E), \Sigma, C \rangle$ is called a constraint graph, if

1. (V,E) is an undirected graph, called the underlying graph of G.
2. The set V is also viewed as a set of variables assuming values over alphabet Σ.
3. Each edge $e \in E$ carries a constraint $c^e : \Sigma^2 \rightarrow \{T, F\}$ and $C = \{c^e\}_{e \in E}$.

Definition 12

An assignment is a mapping $\sigma : V \rightarrow \Sigma$ that gives each vertex in V a value from Σ.

Definition 12

An assignment is a mapping $\sigma : V \to \Sigma$ that gives each vertex in V a value from Σ.
For any assignment σ, define $SAT_\sigma(G) = Pr(c^e(\sigma(u), \sigma(v)) = T]$ and $SAT(G) = max_\sigma SAT_\sigma(G)$.

Example 13

Constructing a constraint graph from a 3-SAT-formula:

$$\phi = \underbrace{(A \vee B \vee C)}_{v_1} \wedge \underbrace{(A \vee D \vee E)}_{v_2} \wedge \underbrace{(D \vee F \vee G)}_{v_3}$$

Example 13

Constructing a constraint graph from a 3-SAT-formula:

$$\phi = \underbrace{(A \vee B \vee C)}_{v_1} \wedge \underbrace{(A \vee D \vee E)}_{v_2} \wedge \underbrace{(D \vee F \vee G)}_{v_3}$$

1. Encode each clause as a vertex.

Example 13

Constructing a constraint graph from a 3-SAT-formula:

$$\phi = \underbrace{(A \vee B \vee C)}_{v_1} \wedge \underbrace{(A \vee D \vee E)}_{v_2} \wedge \underbrace{(D \vee F \vee G)}_{v_3}$$

1. Encode each clause as a vertex.
2. Encode the satisfying assigments to a clause as the alphabet Σ.

(T, T, T)	(T, T, F)	(T, F, T)	(T, F, F)	(F, T, T)	(F, T, F)	(F, F, T)
1	2	3	4	5	6	7

Example 13

Constructing a constraint graph from a 3-SAT-formula:

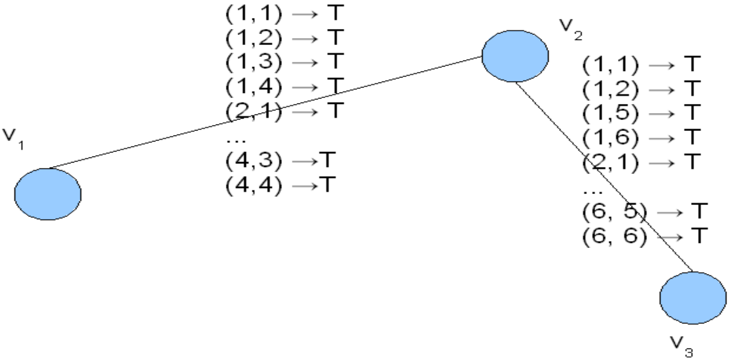
$$\phi = \underbrace{(A \vee B \vee C)}_{v_1} \wedge \underbrace{(A \vee D \vee E)}_{v_2} \wedge \underbrace{(D \vee F \vee G)}_{v_3}$$

1. Encode each clause as a vertex.
2. Encode the satisfying assigments to a clause as the alphabet Σ.

(T, T, T)	(T, T, F)	(T, F, T)	(T, F, F)	(F, T, T)	(F, T, F)	(F, F, T)
1	2	3	4	5	6	7

3. Put a consistency constraint for every pair of clauses that a share a variable.

For ϕ the constraint graph G will look like this:



Theorem 14

Given a constraint graph $G = \langle (V, E), \Sigma, C \rangle$ with $|\Sigma| \leq 7$, it is NP-hard to decide if $SAT(G) = 1$.

Proof by using last Example to reduce to 3SAT.

Definition 15

Let $G = (V, E)$ be a d-regular graph. Let $E(S, \overline{S}) = \left|(S \times \overline{S}) \cap E\right|$ equal the number of edges from a subset $S \subseteq V$ to its complement. The edge expansion is defined as $h(G) = min_{S,|S|<|V|/2} \frac{E(S,\overline{S})}{|S|}$.

Example 16

Example 16

- A disconnected graph has an expansion of 0.

Example 16

- A disconnected graph has an expansion of 0.
- A random d-regular graph has an expansion of about $d/2$, independent of the number of vertices.

Lemma 17

There exist $d_0 \in N$ and $h_0 > 0$, such that there is a polynomial-time constructible family $\{X_n\}_{n \in N}$ of d_0-regular graphs X_n on n vertices with $h(X_n) \geq h_0$.

Example 18

All graphs of size p (for all primes). Here $V_p = Z_p$ and $d = 3$. Ever vertex is connected to its neighbors $(x + 1, x - 1)$ and its inverse (x^{-1}).

Lemma 19

Let G be a d-regular graph, $h(G)$ denotes the edge expansion of G and let $\lambda(G)$ be the second largest eigenvalue of the adjacency matrix of G. Then $\lambda(G) \leq d - \frac{h(G)^2}{d}$.

Lemma 20 (Expander Mixing Lemma)

for all $S, T \subseteq V$*:*

$$\left| E(S,T) - \frac{d|S||T|}{n} \right| \leq \lambda\sqrt{|S|\,|T|}$$

Lemma 20 (Expander Mixing Lemma)

for all $S, T \subseteq V$:

$$\left| E(S,T) - \frac{d|S||T|}{n} \right| \leq \lambda \sqrt{|S|\,|T|}$$

A small λ means a graph with allot of "randomness".

Theorem 21

Let $G = (V, E)$ be a d-regular graph with a second largest eigenvalue λ. Let $F \subseteq E$ be a set of edges. The probability p that a random walk that starts at a random edge in F takes the $i + 1$st step in F as well, is bounded by $\frac{|F|}{|E|} + \left(\frac{\lambda}{d}\right)^i$.

Example 22 (Amplifying the success probability of random algorithms)

$L \in RP$. A decides whether $x \in L$ with m coin tosses and one-sided-error probability β.
Simple way: Pr(A fails)$\leq \beta^t$ and uses $m \cdot t$ coin tosses.
With random walk on expander graphs:
Pr(A fails)$\leq (\beta + \frac{\lambda}{d})^t$ and uses $m + t \cdot log(d)$ coin tosses.

Lemma 23

For any non-negative variable X, $Pr(X > 0) \geq \frac{E^2(X)}{E(X^2)}$.

Proof.

X is non-negative $\Longrightarrow E(X^2) = E(X^2 : X > 0) \cdot Pr(X > 0)$ and $E(X) = E(X : X > 0) \cdot Pr(X > 0)$.

$\Longrightarrow \frac{E^2(X)}{E(X^2)} = \frac{(E(X:X>0) \cdot Pr(X>0))^2}{E(X^2:X>0) \cdot Pr(X>0)} \leq Pr[(X > 0)$.

Lemma 23

For any non-negative variable X, $Pr(X > 0) \geq \frac{E^2(X)}{E(X^2)}$.

Proof.

X is non-negative $\Longrightarrow E(X^2) = E(X^2 : X > 0) \cdot Pr(X > 0)$ and $E(X) = E(X : X > 0) \cdot Pr(X > 0)$.

$\Longrightarrow \frac{E^2(X)}{E(X^2)} = \frac{(E(X:X>0) \cdot Pr(X>0))^2}{E(X^2:X>0) \cdot Pr(X>0)} \leq Pr[(X > 0)$.

because $E(X^2 : X > 0) \geq E^2(X : X > 0)$.

$\square$

Thank you for your attention!

Mihir Bellare.
Proof checking and approximation: Towards tight results.
Sigact News, Vol.27, No. 1, 1996.

Irit Dinur.
Advanced topics in complexity: Pcp theory, 2004.

Irit Dinur.
The pcp theorem by gap amplification.
Electronic Colloquium on Computational Complexity, 2005.

Uri Zwick Howard Karloff.
A 7/8-approximation algorithm for max 3sat?, 1997.

Avi Wigderson Nati Linial.
Expander graphs and their application, 2003.

Christos Papadimitriou.
Computational complexity.
Addision-Wesley, 1994.

Ingo Wegener.
Complexity theory : exploring the limits of efficient algorithms.
Springer, 2005.