Inapproximability in Combinatorial Optimization

Vahan Mkrtchyan

Lane Department of Computer Science and Electrical Engineering
West Virginia University

May 2, 2014

Hardness of Vertex Cover

How to show the hardness of approximation of Vertex cover

Suppose we would like to show that Vertex Cover cannot be approximated within some factor $\alpha > 1$.

Hardness of Vertex Cover

How to show the hardness of approximation of Vertex cover

Suppose we would like to show that Vertex Cover cannot be approximated within some factor $\alpha > 1$. In order to achieve this, we can find a polynomial reduction say from SAT, such that it maps an instance Φ of SAT to a graph $G = (V, E)$ that the following conditions are satisfied:

Hardness of Vertex Cover

How to show the hardness of approximation of Vertex cover

Suppose we would like to show that Vertex Cover cannot be approximated within some factor $\alpha > 1$. In order to achieve this, we can find a polynomial reduction say from SAT, such that it maps an instance Φ of SAT to a graph $G = (V, E)$ that the following conditions are satisfied:

- if Φ is satisfiable, then G has a vertex cover of size $\leq \frac{2}{3} \cdot |V|$,

Hardness of Vertex Cover

How to show the hardness of approximation of Vertex cover

Suppose we would like to show that Vertex Cover cannot be approximated within some factor $\alpha > 1$. In order to achieve this, we can find a polynomial reduction say from SAT, such that it maps an instance Φ of SAT to a graph $G = (V, E)$ that the following conditions are satisfied:

- if Φ is satisfiable, then G has a vertex cover of size $\leq \frac{2}{3} \cdot |V|$,
- if Φ is not satisfiable, then the size of the smallest vertex cover of G is $> \alpha \cdot \frac{2}{3} \cdot |V|$.

Hardness of Vertex Cover

How to show the hardness of approximation of Vertex cover

Suppose we would like to show that Vertex Cover cannot be approximated within some factor $\alpha > 1$. In order to achieve this, we can find a polynomial reduction say from SAT, such that it maps an instance Φ of SAT to a graph $G = (V, E)$ that the following conditions are satisfied:

- if Φ is satisfiable, then G has a vertex cover of size $\leq \frac{2}{3} \cdot |V|$,
- if Φ is not satisfiable, then the size of the smallest vertex cover of G is $> \alpha \cdot \frac{2}{3} \cdot |V|$.

Theorem

Vertex Cover does not admit an approximation algorithm within factor α, unless **P=NP**.

Gap Introducing Reduction

Definition

Let Π be a minimization problem.

Gap Introducing Reduction

Definition

Let Π be a minimization problem. A gap-introducing reduction from SAT to Π comes with two parameters, functions f and α.

Gap Introducing Reduction

Definition

Let Π be a minimization problem. A gap-introducing reduction from SAT to Π comes with two parameters, functions f and α. Given an instance Φ of SAT, it in polynomial time outputs an instance x of Π such that

Gap Introducing Reduction

Definition

Let Π be a minimization problem. A gap-introducing reduction from SAT to Π comes with two parameters, functions f and α. Given an instance Φ of SAT, it in polynomial time outputs an instance x of Π such that

- if Φ is satisfiable, then $OPT(x) \leq f(x)$,

Gap Introducing Reduction

Definition

Let Π be a minimization problem. A gap-introducing reduction from SAT to Π comes with two parameters, functions f and α. Given an instance Φ of SAT, it in polynomial time outputs an instance x of Π such that

- if Φ is satisfiable, then $OPT(x) \leq f(x)$,
- if Φ is not satisfiable, then $OPT(x) > \alpha(|x|) \cdot f(x)$.

Gap Introducing Reduction

Definition

Let Π be a minimization problem. A gap-introducing reduction from SAT to Π comes with two parameters, functions f and α. Given an instance Φ of SAT, it in polynomial time outputs an instance x of Π such that

- if Φ is satisfiable, then $OPT(x) \leq f(x)$,
- if Φ is not satisfiable, then $OPT(x) > \alpha(|x|) \cdot f(x)$.

Some remarks

Notice that f is a function of the instance, and α is a function of the size of the instance.

Gap Introducing Reduction

Definition

Let Π be a minimization problem. A gap-introducing reduction from SAT to Π comes with two parameters, functions f and α. Given an instance Φ of SAT, it in polynomial time outputs an instance x of Π such that

- if Φ is satisfiable, then $OPT(x) \leq f(x)$,
- if Φ is not satisfiable, then $OPT(x) > \alpha(|x|) \cdot f(x)$.

Some remarks

Notice that f is a function of the instance, and α is a function of the size of the instance. Since Π is a minimization problem, the function α satisfies $\alpha(|x|) \geq 1$.

Gap Introducing Reduction

Definition

Let Π be a maximization problem.

Gap Introducing Reduction

Definition

Let Π be a maximization problem. In this case the following conditions must satisfy:

Gap Introducing Reduction

Definition

Let Π be a maximization problem. In this case the following conditions must satisfy:

- if Φ is satisfiable, then $OPT(x) \geq f(x)$,

Gap Introducing Reduction

Definition

Let Π be a maximization problem. In this case the following conditions must satisfy:

- if Φ is satisfiable, then $OPT(x) \geq f(x)$,
- if Φ is not satisfiable, then $OPT(x) < \alpha(|x|) \cdot f(x)$.

Gap Introducing Reduction

Definition

Let Π be a maximization problem. In this case the following conditions must satisfy:

- if Φ is satisfiable, then $OPT(x) \geq f(x)$,
- if Φ is not satisfiable, then $OPT(x) < \alpha(|x|) \cdot f(x)$.

Some remarks

Notice that f is a function of the instance, and α is a function of the size of the instance.

Gap Introducing Reduction

Definition

Let Π be a maximization problem. In this case the following conditions must satisfy:

- if Φ is satisfiable, then $OPT(x) \geq f(x)$,
- if Φ is not satisfiable, then $OPT(x) < \alpha(|x|) \cdot f(x)$.

Some remarks

Notice that f is a function of the instance, and α is a function of the size of the instance. Since Π is a maximization problem, the function α satisfies $\alpha(|x|) \leq 1$.

Gap Preserving Reduction

Definition

Let Π_1 be a minimization problem, and let Π_2 be a maximization problem.

Gap Preserving Reduction

Definition

Let Π_1 be a minimization problem, and let Π_2 be a maximization problem. A gap-preserving reduction Γ from Π_1 to Π_2 comes with four parameters f_1, α, f_2 and β.

Gap Preserving Reduction

Definition

Let Π_1 be a minimization problem, and let Π_2 be a maximization problem. A gap-preserving reduction Γ from Π_1 to Π_2 comes with four parameters f_1, α, f_2 and β. Given an instance x of Π_1, it computes in polynomial time an instance y of Π_2, such that

Gap Preserving Reduction

Definition

Let Π_1 be a minimization problem, and let Π_2 be a maximization problem. A gap-preserving reduction Γ from Π_1 to Π_2 comes with four parameters f_1, α, f_2 and β. Given an instance x of Π_1, it computes in polynomial time an instance y of Π_2, such that

- if $OPT(x) \leq f_1(x)$, then $OPT(y) \geq f_2(y)$,

Gap Preserving Reduction

Definition

Let Π_1 be a minimization problem, and let Π_2 be a maximization problem. A gap-preserving reduction Γ from Π_1 to Π_2 comes with four parameters f_1, α, f_2 and β. Given an instance x of Π_1, it computes in polynomial time an instance y of Π_2, such that

- if $OPT(x) \leq f_1(x)$, then $OPT(y) \geq f_2(y)$,
- if $OPT(x) > \alpha(|x|) \cdot f_1(x)$, then $OPT(y) < \beta(|y|) \cdot f_2(y)$.

The class **NP** and Probabilistically Checkable Proofs (PCP)

PCP system

A PCP system comes with two parameters, the number of random bits required by the verifier, and the number of bits of the witness (or proof), that the verifier is allowed to examine.

The class **NP** and Probabilistically Checkable Proofs (PCP)

PCP system

A PCP system comes with two parameters, the number of random bits required by the verifier, and the number of bits of the witness (or proof), that the verifier is allowed to examine.

Verifier

- The verifier is a polynomial time algorithm, which besides the input and work tapes, has a special tape that provides it with a string of random bits and another special tape on which it is provided with the proof.

The class **NP** and Probabilistically Checkable Proofs (PCP)

PCP system

A PCP system comes with two parameters, the number of random bits required by the verifier, and the number of bits of the witness (or proof), that the verifier is allowed to examine.

Verifier

- The verifier is a polynomial time algorithm, which besides the input and work tapes, has a special tape that provides it with a string of random bits and another special tape on which it is provided with the proof.
- The machine can read any bit of the proof by simply specifying its location.

The class **NP** and Probabilistically Checkable Proofs (PCP)

PCP system

A PCP system comes with two parameters, the number of random bits required by the verifier, and the number of bits of the witness (or proof), that the verifier is allowed to examine.

Verifier

- The verifier is a polynomial time algorithm, which besides the input and work tapes, has a special tape that provides it with a string of random bits and another special tape on which it is provided with the proof.
- The machine can read any bit of the proof by simply specifying its location.
- The particular locations that it examines are a function of the input string and the random string.

The class **NP** and Probabilistically Checkable Proofs (PCP)

PCP system

A PCP system comes with two parameters, the number of random bits required by the verifier, and the number of bits of the witness (or proof), that the verifier is allowed to examine.

Verifier

- The verifier is a polynomial time algorithm, which besides the input and work tapes, has a special tape that provides it with a string of random bits and another special tape on which it is provided with the proof.
- The machine can read any bit of the proof by simply specifying its location.
- The particular locations that it examines are a function of the input string and the random string.
- At the end of computation, the machine either goes to accept state or reject state.

The class **NP** and Probabilistically Checkable Proofs (PCP)

Definition

For two functions $r(n)$ and $q(n)$, the class $\mathbf{PCP}(r(n), q(n))$ is comprised of all languages L,

The class **NP** and Probabilistically Checkable Proofs (PCP)

Definition

For two functions $r(n)$ and $q(n)$, the class $\mathbf{PCP}(r(n), q(n))$ is comprised of all languages L, for which there is a verifier V, such that on any input x the verifier obtains $O(r(n))$ random bits and queries $O(q(n))$ bits of the proof.

The class **NP** and Probabilistically Checkable Proofs (PCP)

Definition

For two functions $r(n)$ and $q(n)$, the class $\mathbf{PCP}(r(n), q(n))$ is comprised of all languages L, for which there is a verifier V, such that on any input x the verifier obtains $O(r(n))$ random bits and queries $O(q(n))$ bits of the proof. Furthermore,

- if $x \in L$, then there is a proof y that makes V accept with probability 1,

The class **NP** and Probabilistically Checkable Proofs (PCP)

Definition

For two functions $r(n)$ and $q(n)$, the class $\mathbf{PCP}(r(n), q(n))$ is comprised of all languages L, for which there is a verifier V, such that on any input x the verifier obtains $O(r(n))$ random bits and queries $O(q(n))$ bits of the proof. Furthermore,

- if $x \in L$, then there is a proof y that makes V accept with probability 1,
- if $x \notin L$, then for every proof y, V accepts with probability $< \frac{1}{2}$,

The class **NP** and Probabilistically Checkable Proofs (PCP)

Definition

For two functions $r(n)$ and $q(n)$, the class $\mathbf{PCP}(r(n), q(n))$ is comprised of all languages L, for which there is a verifier V, such that on any input x the verifier obtains $O(r(n))$ random bits and queries $O(q(n))$ bits of the proof. Furthermore,

- if $x \in L$, then there is a proof y that makes V accept with probability 1,
- if $x \notin L$, then for every proof y, V accepts with probability $< \frac{1}{2}$,

where the probability is taken over all random strings.

The class **NP** and Probabilistically Checkable Proofs (PCP)

Definition

For two functions $r(n)$ and $q(n)$, the class $\mathbf{PCP}(r(n), q(n))$ is comprised of all languages L, for which there is a verifier V, such that on any input x the verifier obtains $O(r(n))$ random bits and queries $O(q(n))$ bits of the proof. Furthermore,

- if $x \in L$, then there is a proof y that makes V accept with probability 1,
- if $x \notin L$, then for every proof y, V accepts with probability $< \frac{1}{2}$,

where the probability is taken over all random strings. The probability of accepting in case $x \notin L$ is called error probability.

The class **NP** and Probabilistically Checkable Proofs (PCP)

The definition of class **NP**

If $poly(n) = \bigcup_{k\geq 0}\{n^k\}$, then the definition of the class **NP** can be re-written as:
NP=PCP$(0, poly(n))$.

The class **NP** and Probabilistically Checkable Proofs (PCP)

The definition of class **NP**

If $poly(n) = \bigcup_{k \geq 0}\{n^k\}$, then the definition of the class **NP** can be re-written as:
NP=PCP$(0, poly(n))$.

The PCP theorem

The well-known PCP theorem offers a new definition of the class **NP**.

The class **NP** and Probabilistically Checkable Proofs (PCP)

The definition of class **NP**

If $poly(n) = \bigcup_{k \geq 0} \{n^k\}$, then the definition of the class **NP** can be re-written as:
NP=PCP$(0, poly(n))$.

The PCP theorem

The well-known PCP theorem offers a new definition of the class **NP**.

Theorem

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

First Implication of PCP theorem

The problem of maximizing the accept probability

Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT. On input Φ, a SAT formula, find a proof, that maximizes the probability of acceptance of V.

First Implication of PCP theorem

The problem of maximizing the accept probability

Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT. On input Φ, a SAT formula, find a proof, that maximizes the probability of acceptance of V.

Theorem

If $\mathbf{P} \neq \mathbf{NP}$*, then there is no* $\frac{1}{2}$*-factor approximation algorithm for the above problem.*

First Implication of PCP theorem

The problem of maximizing the accept probability

Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT. On input Φ, a SAT formula, find a proof, that maximizes the probability of acceptance of V.

Theorem

If $\mathbf{P} \neq \mathbf{NP}$*, then there is no* $\frac{1}{2}$*-factor approximation algorithm for the above problem.*

Proof.

- if Φ is satisfiable, then there is a proof that makes V accept with probability 1, and if Φ is not satisfiable, then on every proof V accepts with probability $< \frac{1}{2}$.

First Implication of PCP theorem

The problem of maximizing the accept probability

Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT. On input Φ, a SAT formula, find a proof, that maximizes the probability of acceptance of V.

Theorem

If $\mathbf{P} \neq \mathbf{NP}$*, then there is no* $\frac{1}{2}$*-factor approximation algorithm for the above problem.*

Proof.

- if Φ is satisfiable, then there is a proof that makes V accept with probability 1, and if Φ is not satisfiable, then on every proof V accepts with probability $< \frac{1}{2}$.
- Suppose that there exists a $\frac{1}{2}$-factor approximation algorithm for the problem.

First Implication of PCP theorem

The problem of maximizing the accept probability

Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT. On input Φ, a SAT formula, find a proof, that maximizes the probability of acceptance of V.

Theorem

If **P**$\neq$ **NP***, then there is no* $\frac{1}{2}$*-factor approximation algorithm for the above problem.*

Proof.

- if Φ is satisfiable, then there is a proof that makes V accept with probability 1, and if Φ is not satisfiable, then on every proof V accepts with probability $< \frac{1}{2}$.
- Suppose that there exists a $\frac{1}{2}$-factor approximation algorithm for the problem. If Φ is satisfiable, then this algorithm must provide a proof that makes V accept with probability $\geq \frac{1}{2}$.

First Implication of PCP theorem

The problem of maximizing the accept probability

Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT. On input Φ, a SAT formula, find a proof, that maximizes the probability of acceptance of V.

Theorem

If $\mathbf{P} \neq \mathbf{NP}$*, then there is no* $\frac{1}{2}$*-factor approximation algorithm for the above problem.*

Proof.

- if Φ is satisfiable, then there is a proof that makes V accept with probability 1, and if Φ is not satisfiable, then on every proof V accepts with probability $< \frac{1}{2}$.
- Suppose that there exists a $\frac{1}{2}$-factor approximation algorithm for the problem. If Φ is satisfiable, then this algorithm must provide a proof that makes V accept with probability $\geq \frac{1}{2}$.
- The acceptance probability can be computed in polynomial time, by simply simulating V for all random strings of length $O(\log(n))$.

First Implication of PCP theorem

The problem of maximizing the accept probability

Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT. On input Φ, a SAT formula, find a proof, that maximizes the probability of acceptance of V.

Theorem

If $\mathbf{P} \neq \mathbf{NP}$*, then there is no* $\frac{1}{2}$*-factor approximation algorithm for the above problem.*

Proof.

- if Φ is satisfiable, then there is a proof that makes V accept with probability 1, and if Φ is not satisfiable, then on every proof V accepts with probability $< \frac{1}{2}$.
- Suppose that there exists a $\frac{1}{2}$-factor approximation algorithm for the problem. If Φ is satisfiable, then this algorithm must provide a proof that makes V accept with probability $\geq \frac{1}{2}$.
- The acceptance probability can be computed in polynomial time, by simply simulating V for all random strings of length $O(\log(n))$.
- Observe that we created a gap with $\frac{1}{2}$-factor approximation algorithm for the problem.

First Implication of PCP theorem

The problem of maximizing the accept probability

Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT. On input Φ, a SAT formula, find a proof, that maximizes the probability of acceptance of V.

Theorem

If **P**$\neq$ **NP**, *then there is no* $\frac{1}{2}$*-factor approximation algorithm for the above problem.*

Proof.

- if Φ is satisfiable, then there is a proof that makes V accept with probability 1, and if Φ is not satisfiable, then on every proof V accepts with probability $< \frac{1}{2}$.
- Suppose that there exists a $\frac{1}{2}$-factor approximation algorithm for the problem. If Φ is satisfiable, then this algorithm must provide a proof that makes V accept with probability $\geq \frac{1}{2}$.
- The acceptance probability can be computed in polynomial time, by simply simulating V for all random strings of length $O(\log(n))$.
- Observe that we created a gap with $\frac{1}{2}$-factor approximation algorithm for the problem.
- Thus, this approximation algorithm can be used for deciding SAT in polynomial time.

$\square$

MAX-k-Function-SAT

The formulation of MAX-k-Function-SAT

Given n boolean variables $x_1, \ldots, x_n$ and m functions $f_1, \ldots, f_m$ each of which is a function of the k of variables.

MAX-k-Function-SAT

The formulation of MAX-k-Function-SAT

Given n boolean variables $x_1, \ldots, x_n$ and m functions $f_1, \ldots, f_m$ each of which is a function of the k of variables. Find a truth assignment to $x_1, \ldots, x_n$ that maximizes the number of satisfied functions.

MAX-k-Function-SAT

The formulation of MAX-k-Function-SAT

Given n boolean variables $x_1, \ldots, x_n$ and m functions $f_1, \ldots, f_m$ each of which is a function of the k of variables. Find a truth assignment to $x_1, \ldots, x_n$ that maximizes the number of satisfied functions. Here k is assumed to be a fixed constant (not part of the input).

MAX-k-Function-SAT

The formulation of MAX-k-Function-SAT

Given n boolean variables $x_1, \ldots, x_n$ and m functions $f_1, \ldots, f_m$ each of which is a function of the k of variables. Find a truth assignment to $x_1, \ldots, x_n$ that maximizes the number of satisfied functions. Here k is assumed to be a fixed constant (not part of the input).

Lemma

There is a constant k, for which there is a gap-introducing reduction from SAT to MAX-k-Function-SAT

MAX-k-Function-SAT

The formulation of MAX-k-Function-SAT

Given n boolean variables $x_1, \ldots, x_n$ and m functions $f_1, \ldots, f_m$ each of which is a function of the k of variables. Find a truth assignment to $x_1, \ldots, x_n$ that maximizes the number of satisfied functions. Here k is assumed to be a fixed constant (not part of the input).

Lemma

There is a constant k, for which there is a gap-introducing reduction from SAT to MAX-k-Function-SAT that transforms a boolean formula Φ to an instance I of MAX-k-Function-SAT, such that

MAX-k-Function-SAT

The formulation of MAX-k-Function-SAT

Given n boolean variables $x_1, \ldots, x_n$ and m functions $f_1, \ldots, f_m$ each of which is a function of the k of variables. Find a truth assignment to $x_1, \ldots, x_n$ that maximizes the number of satisfied functions. Here k is assumed to be a fixed constant (not part of the input).

Lemma

There is a constant k, for which there is a gap-introducing reduction from SAT to MAX-k-Function-SAT that transforms a boolean formula Φ to an instance I of MAX-k-Function-SAT, such that

- *if Φ is satisfiable, then $OPT(I) = m$,*

MAX-k-Function-SAT

The formulation of MAX-k-Function-SAT

Given n boolean variables $x_1, \ldots, x_n$ and m functions $f_1, \ldots, f_m$ each of which is a function of the k of variables. Find a truth assignment to $x_1, \ldots, x_n$ that maximizes the number of satisfied functions. Here k is assumed to be a fixed constant (not part of the input).

Lemma

There is a constant k, for which there is a gap-introducing reduction from SAT to MAX-k-Function-SAT that transforms a boolean formula Φ to an instance I of MAX-k-Function-SAT, such that

- *if Φ is satisfiable, then $OPT(I) = m$,*
- *if Φ is not satisfiable, then $OPT(I) < \frac{m}{2}$,*

MAX-k-Function-SAT

The formulation of MAX-k-Function-SAT

Given n boolean variables $x_1, \ldots, x_n$ and m functions $f_1, \ldots, f_m$ each of which is a function of the k of variables. Find a truth assignment to $x_1, \ldots, x_n$ that maximizes the number of satisfied functions. Here k is assumed to be a fixed constant (not part of the input).

Lemma

There is a constant k, for which there is a gap-introducing reduction from SAT to MAX-k-Function-SAT that transforms a boolean formula Φ to an instance I of MAX-k-Function-SAT, such that

- *if Φ is satisfiable, then $OPT(I) = m$,*
- *if Φ is not satisfiable, then $OPT(I) < \frac{m}{2}$,*

where m is the number of formulae in I.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT with associated parameters c and q.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT with associated parameters c and q. Let Φ be an instance of SAT of length n.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT with associated parameters c and q. Let Φ be an instance of SAT of length n.
- Corresponding to each string r of length $c \cdot \log(n)$, V reads q bits of the proof.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT with associated parameters c and q. Let Φ be an instance of SAT of length n.
- Corresponding to each string r of length $c \cdot \log(n)$, V reads q bits of the proof. Thus V reads a total of $\leq q \cdot n^c$ bits of the proof.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT with associated parameters c and q. Let Φ be an instance of SAT of length n.
- Corresponding to each string r of length $c \cdot \log(n)$, V reads q bits of the proof. Thus V reads a total of $\leq q \cdot n^c$ bits of the proof.
- Let B be a set of boolean variables corresponding to each of these bits.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT with associated parameters c and q. Let Φ be an instance of SAT of length n.
- Corresponding to each string r of length $c \cdot \log(n)$, V reads q bits of the proof. Thus V reads a total of $\leq q \cdot n^c$ bits of the proof.
- Let B be a set of boolean variables corresponding to each of these bits.The relevant part of each proof corresponds to an assignment to the variables of B.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT with associated parameters c and q. Let Φ be an instance of SAT of length n.
- Corresponding to each string r of length $c \cdot \log(n)$, V reads q bits of the proof. Thus V reads a total of $\leq q \cdot n^c$ bits of the proof.
- Let B be a set of boolean variables corresponding to each of these bits.The relevant part of each proof corresponds to an assignment to the variables of B.
- We will establish lemma for $k = q$.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT with associated parameters c and q. Let Φ be an instance of SAT of length n.
- Corresponding to each string r of length $c \cdot \log(n)$, V reads q bits of the proof. Thus V reads a total of $\leq q \cdot n^c$ bits of the proof.
- Let B be a set of boolean variables corresponding to each of these bits.The relevant part of each proof corresponds to an assignment to the variables of B.
- We will establish lemma for $k = q$. The acceptance or rejection of V is a function of Φ, r and the q-bits of the proof read by V.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT with associated parameters c and q. Let Φ be an instance of SAT of length n.
- Corresponding to each string r of length $c \cdot \log(n)$, V reads q bits of the proof. Thus V reads a total of $\leq q \cdot n^c$ bits of the proof.
- Let B be a set of boolean variables corresponding to each of these bits.The relevant part of each proof corresponds to an assignment to the variables of B.
- We will establish lemma for $k = q$. The acceptance or rejection of V is a function of Φ, r and the q-bits of the proof read by V. When Φ and r are fixed, consider the restriction of this function to the q bits of the proof. Denote the function by f_r.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a **PCP**$(\log(n), 1)$ verifier for SAT with associated parameters c and q. Let Φ be an instance of SAT of length n.
- Corresponding to each string r of length $c \cdot \log(n)$, V reads q bits of the proof. Thus V reads a total of $\leq q \cdot n^c$ bits of the proof.
- Let B be a set of boolean variables corresponding to each of these bits.The relevant part of each proof corresponds to an assignment to the variables of B.
- We will establish lemma for $k = q$. The acceptance or rejection of V is a function of Φ, r and the q-bits of the proof read by V. When Φ and r are fixed, consider the restriction of this function to the q bits of the proof. Denote the function by f_r.
- Clearly, there is a polynomial algorithm which for the input Φ, outputs all $m = n^c$ functions f_r.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT with associated parameters c and q. Let Φ be an instance of SAT of length n.
- Corresponding to each string r of length $c \cdot \log(n)$, V reads q bits of the proof. Thus V reads a total of $\leq q \cdot n^c$ bits of the proof.
- Let B be a set of boolean variables corresponding to each of these bits.The relevant part of each proof corresponds to an assignment to the variables of B.
- We will establish lemma for $k = q$. The acceptance or rejection of V is a function of Φ, r and the q-bits of the proof read by V. When Φ and r are fixed, consider the restriction of this function to the q bits of the proof. Denote the function by f_r.
- Clearly, there is a polynomial algorithm which for the input Φ, outputs all $m = n^c$ functions f_r.
- If Φ is satisfiable, then there is a proof that V accepts with probability 1.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT with associated parameters c and q. Let Φ be an instance of SAT of length n.
- Corresponding to each string r of length $c \cdot \log(n)$, V reads q bits of the proof. Thus V reads a total of $\leq q \cdot n^c$ bits of the proof.
- Let B be a set of boolean variables corresponding to each of these bits.The relevant part of each proof corresponds to an assignment to the variables of B.
- We will establish lemma for $k = q$. The acceptance or rejection of V is a function of Φ, r and the q-bits of the proof read by V. When Φ and r are fixed, consider the restriction of this function to the q bits of the proof. Denote the function by f_r.
- Clearly, there is a polynomial algorithm which for the input Φ, outputs all $m = n^c$ functions f_r.
- If Φ is satisfiable, then there is a proof that V accepts with probability 1. The corresponding assignment to B satisfies all n^c functions f_r.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT with associated parameters c and q. Let Φ be an instance of SAT of length n.
- Corresponding to each string r of length $c \cdot \log(n)$, V reads q bits of the proof. Thus V reads a total of $\leq q \cdot n^c$ bits of the proof.
- Let B be a set of boolean variables corresponding to each of these bits.The relevant part of each proof corresponds to an assignment to the variables of B.
- We will establish lemma for $k = q$. The acceptance or rejection of V is a function of Φ, r and the q-bits of the proof read by V. When Φ and r are fixed, consider the restriction of this function to the q bits of the proof. Denote the function by f_r.
- Clearly, there is a polynomial algorithm which for the input Φ, outputs all $m = n^c$ functions f_r.
- If Φ is satisfiable, then there is a proof that V accepts with probability 1. The corresponding assignment to B satisfies all n^c functions f_r. On the other hand, if Φ is not satisfiable, then on every proof V accepts with probability $< \frac{1}{2}$.

The proof of Lemma on MAX-k-Function-SAT

Proof.

- Let V be a **PCP**$(\log(n), 1)$ verifier for SAT with associated parameters c and q. Let Φ be an instance of SAT of length n.
- Corresponding to each string r of length $c \cdot \log(n)$, V reads q bits of the proof. Thus V reads a total of $\leq q \cdot n^c$ bits of the proof.
- Let B be a set of boolean variables corresponding to each of these bits.The relevant part of each proof corresponds to an assignment to the variables of B.
- We will establish lemma for $k = q$. The acceptance or rejection of V is a function of Φ, r and the q-bits of the proof read by V. When Φ and r are fixed, consider the restriction of this function to the q bits of the proof. Denote the function by f_r.
- Clearly, there is a polynomial algorithm which for the input Φ, outputs all $m = n^c$ functions f_r.
- If Φ is satisfiable, then there is a proof that V accepts with probability 1. The corresponding assignment to B satisfies all n^c functions f_r. On the other hand, if Φ is not satisfiable, then on every proof V accepts with probability $< \frac{1}{2}$. Thus in this case every assignment satisfies $< \frac{1}{2} \cdot n^c$ of these functions.

□

MAX-SAT and MAX-3-SAT

The formulation of MAX-SAT

Given a conjunctive normal form $f(x_1, \ldots, x_n) = D_1 \wedge \ldots \wedge D_r$. Find a truth assignment to boolean variables $x_1, \ldots, x_n$ that maximizes the number of satisfied clauses.

MAX-SAT and MAX-3-SAT

The formulation of MAX-SAT

Given a conjunctive normal form $f(x_1, \ldots, x_n) = D_1 \wedge \ldots \wedge D_r$. Find a truth assignment to boolean variables $x_1, \ldots, x_n$ that maximizes the number of satisfied clauses.

The formulation of MAX-3-SAT

Given a conjunctive normal form $f(x_1, \ldots, x_n) = D_1 \wedge \ldots \wedge D_r$, where each D_j contains at most three literals. Find a truth assignment to boolean variables $x_1, \ldots, x_n$ that maximizes the number of satisfied clauses.

The main result on MAX-3-SAT

Theorem

There is a constant $\varepsilon_M > 0$ *for which there is a gap-introducing reduction from SAT to MAX-3-SAT that transforms a boolean formula* Φ *to* Ψ*, such that*

- *if* Φ *is satisfiable, then* $OPT(\Psi) = m$*,*

The main result on MAX-3-SAT

Theorem

There is a constant $\varepsilon_M > 0$ *for which there is a gap-introducing reduction from SAT to MAX-3-SAT that transforms a boolean formula* Φ *to* Ψ*, such that*

- *if* Φ *is satisfiable, then* $OPT(\Psi) = m$*,*
- *if* Φ *is not satisfiable, then* $OPT(\Psi) < (1 - \varepsilon_M) \cdot m$*,*

The main result on MAX-3-SAT

Theorem

There is a constant $\varepsilon_M > 0$ for which there is a gap-introducing reduction from SAT to MAX-3-SAT that transforms a boolean formula Φ to Ψ, such that

- *if Φ is satisfiable, then $OPT(\Psi) = m$,*
- *if Φ is not satisfiable, then $OPT(\Psi) < (1 - \varepsilon_M) \cdot m$,*

where m is the number of clauses in Ψ.

The main result on MAX-3-SAT

Theorem

There is a constant $\varepsilon_M > 0$ for which there is a gap-introducing reduction from SAT to MAX-3-SAT that transforms a boolean formula Φ to Ψ, such that

- *if Φ is satisfiable, then $OPT(\Psi) = m$,*
- *if Φ is not satisfiable, then $OPT(\Psi) < (1 - \varepsilon_M) \cdot m$,*

where m is the number of clauses in Ψ.

Corollary

There is no approximation algorithm for MAX-3-SAT with approximation factor $(1 - \varepsilon_M)$ assuming **P**$\neq$ **NP**, *where ε_M is the constant defined in the above theorem.*

The Proof of a Result on MAX-3-SAT

Proof.

- Using the lemma on MAX-k-Function-SAT, we transform a SAT formula Φ to an instance of MAX-k-Function-SAT.

The Proof of a Result on MAX-3-SAT

Proof.

- Using the lemma on MAX-k-Function-SAT, we transform a SAT formula Φ to an instance of MAX-k-Function-SAT. Now we show how to obtain a 3-SAT formula from the n^c functions.

The Proof of a Result on MAX-3-SAT

Proof.

- Using the lemma on MAX-k-Function-SAT, we transform a SAT formula Φ to an instance of MAX-k-Function-SAT. Now we show how to obtain a 3-SAT formula from the n^c functions.
- Each function f_r constructed in the lemma can be written as a SAT formula ψ_r containing at most 2^q clauses.

The Proof of a Result on MAX-3-SAT

Proof.

- Using the lemma on MAX-k-Function-SAT, we transform a SAT formula Φ to an instance of MAX-k-Function-SAT. Now we show how to obtain a 3-SAT formula from the n^c functions.
- Each function f_r constructed in the lemma can be written as a SAT formula ψ_r containing at most 2^q clauses. Each clause of ψ_r contains at most q literals.

The Proof of a Result on MAX-3-SAT

Proof.

- Using the lemma on MAX-k-Function-SAT, we transform a SAT formula Φ to an instance of MAX-k-Function-SAT. Now we show how to obtain a 3-SAT formula from the n^c functions.
- Each function f_r constructed in the lemma can be written as a SAT formula ψ_r containing at most 2^q clauses. Each clause of ψ_r contains at most q literals. Let $\psi = \bigwedge_r \psi_r$.

The Proof of a Result on MAX-3-SAT

Proof.

- Using the lemma on MAX-k-Function-SAT, we transform a SAT formula Φ to an instance of MAX-k-Function-SAT. Now we show how to obtain a 3-SAT formula from the n^c functions.
- Each function f_r constructed in the lemma can be written as a SAT formula ψ_r containing at most 2^q clauses. Each clause of ψ_r contains at most q literals. Let $\psi = \bigwedge_r \psi_r$.
- If an assignment satisfies f_r, then it satisfies all the clauses of ψ_r.

The Proof of a Result on MAX-3-SAT

Proof.

- Using the lemma on MAX-k-Function-SAT, we transform a SAT formula Φ to an instance of MAX-k-Function-SAT. Now we show how to obtain a 3-SAT formula from the n^c functions.
- Each function f_r constructed in the lemma can be written as a SAT formula ψ_r containing at most 2^q clauses. Each clause of ψ_r contains at most q literals. Let $\psi = \bigwedge_r \psi_r$.
- If an assignment satisfies f_r, then it satisfies all the clauses of ψ_r. On the other hand, if it does not satisfy f_r, then it must leave at least one clause of ψ_r unsatisfied.

The Proof of a Result on MAX-3-SAT

Proof.

- Using the lemma on MAX-k-Function-SAT, we transform a SAT formula Φ to an instance of MAX-k-Function-SAT. Now we show how to obtain a 3-SAT formula from the n^c functions.
- Each function f_r constructed in the lemma can be written as a SAT formula ψ_r containing at most 2^q clauses. Each clause of ψ_r contains at most q literals. Let $\psi = \bigwedge_r \psi_r$.
- If an assignment satisfies f_r, then it satisfies all the clauses of ψ_r. On the other hand, if it does not satisfy f_r, then it must leave at least one clause of ψ_r unsatisfied. Hence if Φ is not satisfiable, then any assignment must leave $> \frac{1}{2} \cdot n^c$ clauses of ψ unsatisfied.

The Proof of a Result on MAX-3-SAT

Proof.

- Let us transform ψ into a 3-SAT formula.

The Proof of a Result on MAX-3-SAT

Proof.

- Let us transform ψ into a 3-SAT formula. Consider a clause $C = (x_1 \vee \ldots \vee x_k)$ with $k > 3$.

The Proof of a Result on MAX-3-SAT

Proof.

- Let us transform ψ into a 3-SAT formula. Consider a clause $C = (x_1 \vee \ldots \vee x_k)$ with $k > 3$. Introduce new variables $y_1, \ldots, y_{k-2}$, and consider the formula

$$C = (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\bar{y}_{k-2} \vee x_{k-1} \vee x_k).$$

The Proof of a Result on MAX-3-SAT

Proof.

- Let us transform ψ into a 3-SAT formula. Consider a clause $C = (x_1 \vee \ldots \vee x_k)$ with $k > 3$. Introduce new variables $y_1, \ldots, y_{k-2}$, and consider the formula

$$C = (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\bar{y}_{k-2} \vee x_{k-1} \vee x_k).$$

  Let τ be any assignment to $x_1, \ldots, x_k$.

The Proof of a Result on MAX-3-SAT

Proof.

- Let us transform ψ into a 3-SAT formula. Consider a clause $C = (x_1 \vee \ldots \vee x_k)$ with $k > 3$. Introduce new variables $y_1, \ldots, y_{k-2}$, and consider the formula

$$C = (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\bar{y}_{k-2} \vee x_{k-1} \vee x_k).$$

  Let τ be any assignment to $x_1, \ldots, x_k$. If τ satisfies C, then it can be extended to an assignment satisfying all clauses of f.

The Proof of a Result on MAX-3-SAT

Proof.

- Let us transform ψ into a 3-SAT formula. Consider a clause $C = (x_1 \vee \ldots \vee x_k)$ with $k > 3$. Introduce new variables $y_1, \ldots, y_{k-2}$, and consider the formula

$$C = (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\bar{y}_{k-2} \vee x_{k-1} \vee x_k).$$

  Let τ be any assignment to $x_1, \ldots, x_k$. If τ satisfies C, then it can be extended to an assignment satisfying all clauses of f. On the other hand, if τ does not satisfy C, then for any values of $y_1, \ldots, y_{k-2}$, at least one of the clauses of f remains unsatisfied.

The Proof of a Result on MAX-3-SAT

Proof.

- Let us transform ψ into a 3-SAT formula. Consider a clause $C = (x_1 \vee \ldots \vee x_k)$ with $k > 3$. Introduce new variables $y_1, \ldots, y_{k-2}$, and consider the formula

$$C = (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\bar{y}_{k-2} \vee x_{k-1} \vee x_k).$$

  Let τ be any assignment to $x_1, \ldots, x_k$. If τ satisfies C, then it can be extended to an assignment satisfying all clauses of f. On the other hand, if τ does not satisfy C, then for any values of $y_1, \ldots, y_{k-2}$, at least one of the clauses of f remains unsatisfied.
- Apply this trick to any clause of ψ with more than 3 literals.

The Proof of a Result on MAX-3-SAT

Proof.

- Let us transform ψ into a 3-SAT formula. Consider a clause $C = (x_1 \vee \dots \vee x_k)$ with $k > 3$. Introduce new variables $y_1, \dots, y_{k-2}$, and consider the formula

  $$C = (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge \dots \wedge (\bar{y}_{k-2} \vee x_{k-1} \vee x_k).$$

  Let τ be any assignment to $x_1, \dots, x_k$. If τ satisfies C, then it can be extended to an assignment satisfying all clauses of f. On the other hand, if τ does not satisfy C, then for any values of $y_1, \dots, y_{k-2}$, at least one of the clauses of f remains unsatisfied.
- Apply this trick to any clause of ψ with more than 3 literals. Let ψ' be the resulting 3-SAT formula.

The Proof of a Result on MAX-3-SAT

Proof.

- Let us transform ψ into a 3-SAT formula. Consider a clause $C = (x_1 \vee \ldots \vee x_k)$ with $k > 3$. Introduce new variables $y_1, \ldots, y_{k-2}$, and consider the formula

$$C = (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\bar{y}_{k-2} \vee x_{k-1} \vee x_k).$$

  Let τ be any assignment to $x_1, \ldots, x_k$. If τ satisfies C, then it can be extended to an assignment satisfying all clauses of f. On the other hand, if τ does not satisfy C, then for any values of $y_1, \ldots, y_{k-2}$, at least one of the clauses of f remains unsatisfied.

- Apply this trick to any clause of ψ with more than 3 literals. Let ψ' be the resulting 3-SAT formula. It contains at most $n^c \cdot 2^q \cdot (q-2)$ clauses.

The Proof of a Result on MAX-3-SAT

Proof.

- Let us transform ψ into a 3-SAT formula. Consider a clause $C = (x_1 \vee \ldots \vee x_k)$ with $k > 3$. Introduce new variables $y_1, \ldots, y_{k-2}$, and consider the formula

$$C = (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\bar{y}_{k-2} \vee x_{k-1} \vee x_k).$$

  Let τ be any assignment to $x_1, \ldots, x_k$. If τ satisfies C, then it can be extended to an assignment satisfying all clauses of f. On the other hand, if τ does not satisfy C, then for any values of $y_1, \ldots, y_{k-2}$, at least one of the clauses of f remains unsatisfied.
- Apply this trick to any clause of ψ with more than 3 literals. Let ψ' be the resulting 3-SAT formula. It contains at most $n^c \cdot 2^q \cdot (q-2)$ clauses. If Φ is satisfiable, then there is an assignment satisfying all clauses of ψ'.

The Proof of a Result on MAX-3-SAT

Proof.

- Let us transform ψ into a 3-SAT formula. Consider a clause $C = (x_1 \vee \ldots \vee x_k)$ with $k > 3$. Introduce new variables $y_1, \ldots, y_{k-2}$, and consider the formula

$$C = (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\bar{y}_{k-2} \vee x_{k-1} \vee x_k).$$

  Let τ be any assignment to $x_1, \ldots, x_k$. If τ satisfies C, then it can be extended to an assignment satisfying all clauses of f. On the other hand, if τ does not satisfy C, then for any values of $y_1, \ldots, y_{k-2}$, at least one of the clauses of f remains unsatisfied.
- Apply this trick to any clause of ψ with more than 3 literals. Let ψ' be the resulting 3-SAT formula. It contains at most $n^c \cdot 2^q \cdot (q-2)$ clauses. If Φ is satisfiable, then there is an assignment satisfying all clauses of ψ'. If Φ is not satisfiable, then $> \frac{1}{2} \cdot n^c$ of the clauses remain unsatisfied under any assignment.

The Proof of a Result on MAX-3-SAT

Proof.

- Let us transform ψ into a 3-SAT formula. Consider a clause $C = (x_1 \vee \ldots \vee x_k)$ with $k > 3$. Introduce new variables $y_1, \ldots, y_{k-2}$, and consider the formula

  $$C = (x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\bar{y}_{k-2} \vee x_{k-1} \vee x_k).$$

  Let τ be any assignment to $x_1, \ldots, x_k$. If τ satisfies C, then it can be extended to an assignment satisfying all clauses of f. On the other hand, if τ does not satisfy C, then for any values of $y_1, \ldots, y_{k-2}$, at least one of the clauses of f remains unsatisfied.
- Apply this trick to any clause of ψ with more than 3 literals. Let ψ' be the resulting 3-SAT formula. It contains at most $n^c \cdot 2^q \cdot (q-2)$ clauses. If Φ is satisfiable, then there is an assignment satisfying all clauses of ψ'. If Φ is not satisfiable, then $> \frac{1}{2} \cdot n^c$ of the clauses remain unsatisfied under any assignment.
- Taking $\varepsilon_M = \frac{1}{2^{q+1} \cdot (q-2)}$ gives the proof of the theorem.

$\square$

Expanders

Definition

A graph $G = (V,E)$ is an expander, if G is regular, that is every vertex has the same degree, and for every $\emptyset \subset S \subset V$ one has

$$|E(S,\bar{S})| > \min\{|S|,|\bar{S}|\},$$

where $E(S,\bar{S})$ denotes the set of edges of G that have one endpoint in S and the other in $\bar{S}$.

Expanders

Definition

A graph $G = (V, E)$ is an expander, if G is regular, that is every vertex has the same degree, and for every $\emptyset \subset S \subset V$ one has

$$|E(S, \bar{S})| > \min\{|S|, |\bar{S}|\},$$

where $E(S, \bar{S})$ denotes the set of edges of G that have one endpoint in S and the other in $\bar{S}$.

Theorem

There exists an algorithm A and a number N_0, such that for each $N \geq N_0$, A constructs a degree 14 expander on N vertices in time polynomial in N.

Expanders

Definition

A graph $G = (V, E)$ is an expander, if G is regular, that is every vertex has the same degree, and for every $\emptyset \subset S \subset V$ one has

$$|E(S, \bar{S})| > \min\{|S|, |\bar{S}|\},$$

where $E(S, \bar{S})$ denotes the set of edges of G that have one endpoint in S and the other in $\bar{S}$.

Theorem

There exists an algorithm A and a number N_0, such that for each $N \geq N_0$, A constructs a degree 14 expander on N vertices in time polynomial in N.

Remark

See remark 29.12 and section 29.9 of the Vazirani's book for details on this theorem.

The problem MAX-3-SAT(k)

Definition

For each fixed k, let MAX-3-SAT(k) denote the restriction of MAX-3-SAT to instances, in which each variable occurs at most k times.

The problem MAX-3-SAT(k)

Definition

For each fixed k, let MAX-3-SAT(k) denote the restriction of MAX-3-SAT to instances, in which each variable occurs at most k times.

Theorem

There is a a gap-preserving reduction from MAX-3-SAT to MAX-3-SAT(29) that transforms a boolean formula Φ to Ψ, such that

- *if $OPT(\Phi) = m$, then $OPT(\Psi) = m'$,*

The problem MAX-3-SAT(k)

Definition

For each fixed k, let MAX-3-SAT(k) denote the restriction of MAX-3-SAT to instances, in which each variable occurs at most k times.

Theorem

There is a a gap-preserving reduction from MAX-3-SAT to MAX-3-SAT(29) that transforms a boolean formula Φ to Ψ, such that

- *if $OPT(\Phi) = m$, then $OPT(\Psi) = m'$,*
- *if $OPT(\Phi) < (1 - \varepsilon_M) \cdot m$, then $OPT(\Psi) < (1 - \varepsilon_b) \cdot m'$,*

The problem MAX-3-SAT(k)

Definition

For each fixed k, let MAX-3-SAT(k) denote the restriction of MAX-3-SAT to instances, in which each variable occurs at most k times.

Theorem

There is a a gap-preserving reduction from MAX-3-SAT to MAX-3-SAT(29) that transforms a boolean formula Φ to Ψ, such that

- *if $OPT(\Phi) = m$, then $OPT(\Psi) = m'$,*
- *if $OPT(\Phi) < (1 - \varepsilon_M) \cdot m$, then $OPT(\Psi) < (1 - \varepsilon_b) \cdot m'$,*

where m and m' are the number of clauses in Φ and Ψ, and $\varepsilon_b = \frac{\varepsilon_M}{43}$.

The problem MAX-3-SAT(k)

Proof.

- Let $k \geq N_0$, and let G_x be a degree 14 expander on k vertices.

The problem MAX-3-SAT(k)

Proof.

- Let $k \geq N_0$, and let G_x be a degree 14 expander on k vertices. Label the vertices with distinct boolean variables $x_1, \ldots, x_k$.

The problem MAX-3-SAT(k)

Proof.

- Let $k \geq N_0$, and let G_x be a degree 14 expander on k vertices. Label the vertices with distinct boolean variables $x_1, \ldots, x_k$.
- Construct a CNF ψ_x as follows: corresponding to each edge (x_i, x_j) of G_x, we include the clauses $(\bar{x}_i \vee x_j)$ and $(\bar{x}_j \vee x_i)$ in ψ_x.

The problem MAX-3-SAT(k)

Proof.

- Let $k \geq N_0$, and let G_x be a degree 14 expander on k vertices. Label the vertices with distinct boolean variables $x_1, \ldots, x_k$.
- Construct a CNF ψ_x as follows: corresponding to each edge (x_i, x_j) of G_x, we include the clauses $(\bar{x}_i \vee x_j)$ and $(\bar{x}_j \vee x_i)$ in ψ_x.
- An assignment is said to be consistent if either all the variables are set to true or all are set to false.

The problem MAX-3-SAT(k)

Proof.

- Let $k \geq N_0$, and let G_x be a degree 14 expander on k vertices. Label the vertices with distinct boolean variables $x_1, \ldots, x_k$.
- Construct a CNF ψ_x as follows: corresponding to each edge (x_i, x_j) of G_x, we include the clauses $(\bar{x}_i \vee x_j)$ and $(\bar{x}_j \vee x_i)$ in ψ_x.
- An assignment is said to be consistent if either all the variables are set to true or all are set to false.
- An inconsistent assignment partitions the vertices of G_x into two sets S and $\bar{S}$.

The problem MAX-3-SAT(k)

Proof.

- Let $k \geq N_0$, and let G_x be a degree 14 expander on k vertices. Label the vertices with distinct boolean variables $x_1, \ldots, x_k$.
- Construct a CNF ψ_x as follows: corresponding to each edge (x_i, x_j) of G_x, we include the clauses $(\bar{x}_i \vee x_j)$ and $(\bar{x}_j \vee x_i)$ in ψ_x.
- An assignment is said to be consistent if either all the variables are set to true or all are set to false.
- An inconsistent assignment partitions the vertices of G_x into two sets S and $\bar{S}$. Assume that S is the smaller.

The problem MAX-3-SAT(k)

Proof.

- Let $k \geq N_0$, and let G_x be a degree 14 expander on k vertices. Label the vertices with distinct boolean variables $x_1, \ldots, x_k$.
- Construct a CNF ψ_x as follows: corresponding to each edge (x_i, x_j) of G_x, we include the clauses $(\bar{x}_i \vee x_j)$ and $(\bar{x}_j \vee x_i)$ in ψ_x.
- An assignment is said to be consistent if either all the variables are set to true or all are set to false.
- An inconsistent assignment partitions the vertices of G_x into two sets S and $\bar{S}$. Assume that S is the smaller.
- Corresponding to each edge in $E(S, \bar{S})$, ψ_x will have an unsatisfied clause.

The problem MAX-3-SAT(k)

Proof.

- Let $k \geq N_0$, and let G_x be a degree 14 expander on k vertices. Label the vertices with distinct boolean variables $x_1, \ldots, x_k$.
- Construct a CNF ψ_x as follows: corresponding to each edge (x_i, x_j) of G_x, we include the clauses $(\bar{x}_i \vee x_j)$ and $(\bar{x}_j \vee x_i)$ in ψ_x.
- An assignment is said to be consistent if either all the variables are set to true or all are set to false.
- An inconsistent assignment partitions the vertices of G_x into two sets S and $\bar{S}$. Assume that S is the smaller.
- Corresponding to each edge in $E(S, \bar{S})$, ψ_x will have an unsatisfied clause. Therefore, the number of unsatisfied clauses, that is $|E(S, \bar{S})|$, is at least $|S| + 1$.

The problem MAX-3-SAT(k)

Proof.

- Now we describe the reduction.

The problem MAX-3-SAT(k)

Proof.

- Now we describe the reduction. We may assume w.l.o.g. that every variable occurs in Φ at least N_0 times.

The problem MAX-3-SAT(k)

Proof.

- Now we describe the reduction. We may assume w.l.o.g. that every variable occurs in Φ at least N_0 times. If not, we can replicate each clause N_0 times without changing the approximability properties of the formula.

The problem MAX-3-SAT(k)

Proof.

- Now we describe the reduction. We may assume w.l.o.g. that every variable occurs in Φ at least N_0 times. If not, we can replicate each clause N_0 times without changing the approximability properties of the formula.
- Let B be the set of variables occurring in Φ.

The problem MAX-3-SAT(k)

Proof.

- Now we describe the reduction. We may assume w.l.o.g. that every variable occurs in Φ at least N_0 times. If not, we can replicate each clause N_0 times without changing the approximability properties of the formula.
- Let B be the set of variables occurring in Φ. For each variable $x \in B$ suppose x occurs $k \geq N_0$ times in Φ.

The problem MAX-3-SAT(k)

Proof.

- Now we describe the reduction. We may assume w.l.o.g. that every variable occurs in Φ at least N_0 times. If not, we can replicate each clause N_0 times without changing the approximability properties of the formula.
- Let B be the set of variables occurring in Φ. For each variable $x \in B$ suppose x occurs $k \geq N_0$ times in Φ.
- Let $V_x = \{x_1, \ldots, x_k\}$ be a set of new variables.

The problem MAX-3-SAT(k)

Proof.

- Now we describe the reduction. We may assume w.l.o.g. that every variable occurs in Φ at least N_0 times. If not, we can replicate each clause N_0 times without changing the approximability properties of the formula.
- Let B be the set of variables occurring in Φ. For each variable $x \in B$ suppose x occurs $k \geq N_0$ times in Φ.
- Let $V_x = \{x_1, \ldots, x_k\}$ be a set of new variables. Let G_x be a degree 14 expander on k vertices.

The problem MAX-3-SAT(k)

Proof.

- Now we describe the reduction. We may assume w.l.o.g. that every variable occurs in Φ at least N_0 times. If not, we can replicate each clause N_0 times without changing the approximability properties of the formula.
- Let B be the set of variables occurring in Φ. For each variable $x \in B$ suppose x occurs $k \geq N_0$ times in Φ.
- Let $V_x = \{x_1, \ldots, x_k\}$ be a set of new variables. Let G_x be a degree 14 expander on k vertices. Label its vertices with variables from V_x and obtain a formula ψ_x as described above.

The problem MAX-3-SAT(k)

Proof.

- Now we describe the reduction. We may assume w.l.o.g. that every variable occurs in Φ at least N_0 times. If not, we can replicate each clause N_0 times without changing the approximability properties of the formula.
- Let B be the set of variables occurring in Φ. For each variable $x \in B$ suppose x occurs $k \geq N_0$ times in Φ.
- Let $V_x = \{x_1, \ldots, x_k\}$ be a set of new variables. Let G_x be a degree 14 expander on k vertices. Label its vertices with variables from V_x and obtain a formula ψ_x as described above. Finally replace each occurrence of x in Φ by a distinct variable from V_x.

The problem MAX-3-SAT(k)

Proof.

- Now we describe the reduction. We may assume w.l.o.g. that every variable occurs in Φ at least N_0 times. If not, we can replicate each clause N_0 times without changing the approximability properties of the formula.
- Let B be the set of variables occurring in Φ. For each variable $x \in B$ suppose x occurs $k \geq N_0$ times in Φ.
- Let $V_x = \{x_1, \ldots, x_k\}$ be a set of new variables. Let G_x be a degree 14 expander on k vertices. Label its vertices with variables from V_x and obtain a formula ψ_x as described above. Finally replace each occurrence of x in Φ by a distinct variable from V_x.
- After this process, every occurrence of a variable from Φ is replaced by a distinct variable from

$$V = \bigcup_{x \in B} V_x.$$

The problem MAX-3-SAT(k)

Proof.

- Now we describe the reduction. We may assume w.l.o.g. that every variable occurs in Φ at least N_0 times. If not, we can replicate each clause N_0 times without changing the approximability properties of the formula.
- Let B be the set of variables occurring in Φ. For each variable $x \in B$ suppose x occurs $k \geq N_0$ times in Φ.
- Let $V_x = \{x_1, \ldots, x_k\}$ be a set of new variables. Let G_x be a degree 14 expander on k vertices. Label its vertices with variables from V_x and obtain a formula ψ_x as described above. Finally replace each occurrence of x in Φ by a distinct variable from V_x.
- After this process, every occurrence of a variable from Φ is replaced by a distinct variable from
$$V = \bigcup_{x \in B} V_x.$$
Let Φ' be the resulting formula.

The problem MAX-3-SAT(k)

Proof.

- Now we describe the reduction. We may assume w.l.o.g. that every variable occurs in Φ at least N_0 times. If not, we can replicate each clause N_0 times without changing the approximability properties of the formula.
- Let B be the set of variables occurring in Φ. For each variable $x \in B$ suppose x occurs $k \geq N_0$ times in Φ.
- Let $V_x = \{x_1, \dots, x_k\}$ be a set of new variables. Let G_x be a degree 14 expander on k vertices. Label its vertices with variables from V_x and obtain a formula ψ_x as described above. Finally replace each occurrence of x in Φ by a distinct variable from V_x.
- After this process, every occurrence of a variable from Φ is replaced by a distinct variable from
$$V = \bigcup_{x \in B} V_x.$$
Let Φ' be the resulting formula. In addition to each variable $x \in B$, a formula ψ_x has been constructed.

The problem MAX-3-SAT(k)

Proof.

- Now we describe the reduction. We may assume w.l.o.g. that every variable occurs in Φ at least N_0 times. If not, we can replicate each clause N_0 times without changing the approximability properties of the formula.
- Let B be the set of variables occurring in Φ. For each variable $x \in B$ suppose x occurs $k \geq N_0$ times in Φ.
- Let $V_x = \{x_1, \dots, x_k\}$ be a set of new variables. Let G_x be a degree 14 expander on k vertices. Label its vertices with variables from V_x and obtain a formula ψ_x as described above. Finally replace each occurrence of x in Φ by a distinct variable from V_x.
- After this process, every occurrence of a variable from Φ is replaced by a distinct variable from
$$V = \bigcup_{x \in B} V_x.$$
Let Φ' be the resulting formula. In addition to each variable $x \in B$, a formula ψ_x has been constructed. Let
$$\psi = \Phi' \wedge \bigwedge_{x \in B} \psi_x.$$

The problem MAX-3-SAT(k)

Proof.

- Observe that for each $x \in B$ each variable of V_x occurs exactly 29 times in ψ- once in Φ' and 28 times in ψ_x.

The problem MAX-3-SAT(k)

Proof.

- Observe that for each $x \in B$ each variable of V_x occurs exactly 29 times in ψ- once in Φ' and 28 times in ψ_x. Therefore, ψ is an instance of MAX-3-SAT(29).

The problem MAX-3-SAT(k)

Proof.

- Observe that for each $x \in B$ each variable of V_x occurs exactly 29 times in ψ- once in Φ' and 28 times in ψ_x. Therefore, ψ is an instance of MAX-3-SAT(29). We will say that the clauses of Φ' are Type I clauses, and the remaining clauses of ψ are Type II clauses.

The problem MAX-3-SAT(k)

Proof.

- Observe that for each $x \in B$ each variable of V_x occurs exactly 29 times in ψ- once in Φ' and 28 times in ψ_x. Therefore, ψ is an instance of MAX-3-SAT(29). We will say that the clauses of Φ' are Type I clauses, and the remaining clauses of ψ are Type II clauses.
- Let us show that an optimal assignment for ψ must satisfy all Type II clauses, and hence it must be consistent for each set $V_x, x \in B$.

The problem MAX-3-SAT(k)

Proof.

- Observe that for each $x \in B$ each variable of V_x occurs exactly 29 times in ψ- once in Φ' and 28 times in ψ_x. Therefore, ψ is an instance of MAX-3-SAT(29). We will say that the clauses of Φ' are Type I clauses, and the remaining clauses of ψ are Type II clauses.
- Let us show that an optimal assignment for ψ must satisfy all Type II clauses, and hence it must be consistent for each set $V_x, x \in B$. On the opposite assumption consider an optimal assignment τ, that is not consistent for V_x for some $x \in B$.

The problem MAX-3-SAT(k)

Proof.

- Observe that for each $x \in B$ each variable of V_x occurs exactly 29 times in ψ- once in Φ' and 28 times in ψ_x. Therefore, ψ is an instance of MAX-3-SAT(29). We will say that the clauses of Φ' are Type I clauses, and the remaining clauses of ψ are Type II clauses.
- Let us show that an optimal assignment for ψ must satisfy all Type II clauses, and hence it must be consistent for each set $V_x, x \in B$. On the opposite assumption consider an optimal assignment τ, that is not consistent for V_x for some $x \in B$.
- τ partitions the vertices of G_x into two sets S and $\bar{S}$ with S being the smaller set.

The problem MAX-3-SAT(k)

Proof.

- Observe that for each $x \in B$ each variable of V_x occurs exactly 29 times in ψ- once in Φ' and 28 times in ψ_x. Therefore, ψ is an instance of MAX-3-SAT(29). We will say that the clauses of Φ' are Type I clauses, and the remaining clauses of ψ are Type II clauses.
- Let us show that an optimal assignment for ψ must satisfy all Type II clauses, and hence it must be consistent for each set $V_x, x \in B$. On the opposite assumption consider an optimal assignment τ, that is not consistent for V_x for some $x \in B$.
- τ partitions the vertices of G_x into two sets S and $\bar{S}$ with S being the smaller set.Flip the assignment to variables in S keeping the rest of the assignment the same as τ.

The problem MAX-3-SAT(k)

Proof.

- Observe that for each $x \in B$ each variable of V_x occurs exactly 29 times in ψ- once in Φ' and 28 times in ψ_x. Therefore, ψ is an instance of MAX-3-SAT(29). We will say that the clauses of Φ' are Type I clauses, and the remaining clauses of ψ are Type II clauses.
- Let us show that an optimal assignment for ψ must satisfy all Type II clauses, and hence it must be consistent for each set $V_x, x \in B$. On the opposite assumption consider an optimal assignment τ, that is not consistent for V_x for some $x \in B$.
- τ partitions the vertices of G_x into two sets S and $\bar{S}$ with S being the smaller set.Flip the assignment to variables in S keeping the rest of the assignment the same as τ.
- As a result, some Type I clauses that were satisfied under τ may now not be satisfied.

The problem MAX-3-SAT(k)

Proof.

- Observe that for each $x \in B$ each variable of V_x occurs exactly 29 times in ψ- once in Φ' and 28 times in ψ_x. Therefore, ψ is an instance of MAX-3-SAT(29). We will say that the clauses of Φ' are Type I clauses, and the remaining clauses of ψ are Type II clauses.
- Let us show that an optimal assignment for ψ must satisfy all Type II clauses, and hence it must be consistent for each set $V_x, x \in B$. On the opposite assumption consider an optimal assignment τ, that is not consistent for V_x for some $x \in B$.
- τ partitions the vertices of G_x into two sets S and $\bar{S}$ with S being the smaller set.Flip the assignment to variables in S keeping the rest of the assignment the same as τ.
- As a result, some Type I clauses that were satisfied under τ may now not be satisfied. Each of these must contain a variable of S, hence their number is at most $|S|$.

The problem MAX-3-SAT(k)

Proof.

- Observe that for each $x \in B$ each variable of V_x occurs exactly 29 times in ψ- once in Φ' and 28 times in ψ_x. Therefore, ψ is an instance of MAX-3-SAT(29). We will say that the clauses of Φ' are Type I clauses, and the remaining clauses of ψ are Type II clauses.
- Let us show that an optimal assignment for ψ must satisfy all Type II clauses, and hence it must be consistent for each set $V_x, x \in B$. On the opposite assumption consider an optimal assignment τ, that is not consistent for V_x for some $x \in B$.
- τ partitions the vertices of G_x into two sets S and $\bar{S}$ with S being the smaller set.Flip the assignment to variables in S keeping the rest of the assignment the same as τ.
- As a result, some Type I clauses that were satisfied under τ may now not be satisfied. Each of these must contain a variable of S, hence their number is at most $|S|$. On the other hand, we get at least $|S| + 1$ new satisfied clauses corresponding to edges in $E(S, \bar{S})$.

The problem MAX-3-SAT(k)

Proof.

- Observe that for each $x \in B$ each variable of V_x occurs exactly 29 times in ψ- once in Φ' and 28 times in ψ_x. Therefore, ψ is an instance of MAX-3-SAT(29). We will say that the clauses of Φ' are Type I clauses, and the remaining clauses of ψ are Type II clauses.
- Let us show that an optimal assignment for ψ must satisfy all Type II clauses, and hence it must be consistent for each set $V_x, x \in B$. On the opposite assumption consider an optimal assignment τ, that is not consistent for V_x for some $x \in B$.
- τ partitions the vertices of G_x into two sets S and $\bar{S}$ with S being the smaller set.Flip the assignment to variables in S keeping the rest of the assignment the same as τ.
- As a result, some Type I clauses that were satisfied under τ may now not be satisfied. Each of these must contain a variable of S, hence their number is at most $|S|$. On the other hand, we get at least $|S| + 1$ new satisfied clauses corresponding to edges in $E(S, \bar{S})$.
- Thus, the flipped assignment satisfies more clauses than τ does contradicting the choice of τ.

The problem MAX-3-SAT(k)

Proof.

- Let m and m' be the number of clauses in Φ and ψ.

The problem MAX-3-SAT(k)

Proof.

- Let m and m' be the number of clauses in Φ and ψ. The total number of occurrences of all variables in Φ is at most $3 \cdot m$.

The problem MAX-3-SAT(k)

Proof.

- Let m and m' be the number of clauses in Φ and ψ. The total number of occurrences of all variables in Φ is at most $3 \cdot m$. Each occurrence participates in 28 Type II two-literal clauses, giving a total of at most $42 \cdot m$ Type II clauses.

The problem MAX-3-SAT(k)

Proof.

- Let m and m' be the number of clauses in Φ and ψ. The total number of occurrences of all variables in Φ is at most $3 \cdot m$. Each occurrence participates in 28 Type II two-literal clauses, giving a total of at most $42 \cdot m$ Type II clauses. In addition ψ has m Type I clauses.

The problem MAX-3-SAT(k)

Proof.

- Let m and m' be the number of clauses in Φ and ψ. The total number of occurrences of all variables in Φ is at most $3 \cdot m$. Each occurrence participates in 28 Type II two-literal clauses, giving a total of at most $42 \cdot m$ Type II clauses. In addition ψ has m Type I clauses. Therefore $m' \leq 43 \cdot m$.

The problem MAX-3-SAT(k)

Proof.

- Let m and m' be the number of clauses in Φ and ψ. The total number of occurrences of all variables in Φ is at most $3 \cdot m$. Each occurrence participates in 28 Type II two-literal clauses, giving a total of at most $42 \cdot m$ Type II clauses. In addition ψ has m Type I clauses. Therefore $m' \leq 43 \cdot m$.
- If Φ is satisfiable, then so is ψ.

The problem MAX-3-SAT(k)

Proof.

- Let m and m' be the number of clauses in Φ and ψ. The total number of occurrences of all variables in Φ is at most $3 \cdot m$. Each occurrence participates in 28 Type II two-literal clauses, giving a total of at most $42 \cdot m$ Type II clauses. In addition ψ has m Type I clauses. Therefore $m' \leq 43 \cdot m$.
- If Φ is satisfiable, then so is ψ. Next consider the case $OPT(\Phi) < (1 - \varepsilon_M) \cdot m$, that is, $> \varepsilon_M \cdot m$ clauses of Φ remain unsatisfied under any assignment.

The problem MAX-3-SAT(k)

Proof.

- Let m and m' be the number of clauses in Φ and ψ. The total number of occurrences of all variables in Φ is at most $3 \cdot m$. Each occurrence participates in 28 Type II two-literal clauses, giving a total of at most $42 \cdot m$ Type II clauses. In addition ψ has m Type I clauses. Therefore $m' \leq 43 \cdot m$.
- If Φ is satisfiable, then so is ψ. Next consider the case $OPT(\Phi) < (1 - \varepsilon_M) \cdot m$, that is, $> \varepsilon_M \cdot m$ clauses of Φ remain unsatisfied under any assignment. If so, by the above claim, $> \varepsilon_M \cdot m \geq \varepsilon_M \cdot \frac{m'}{43}$ of the clauses of ψ remain unsatisfied. The proof of the theorem is completed.

□

The problem Vertex Cover(d)

Definition

For each fixed $d \geq 1$, let Vertex Cover(d) denote the restriction of the Vertex Cover to instances in which each vertex of the graph is of degree at most d.

The problem Vertex Cover(d)

Definition

For each fixed $d \geq 1$, let Vertex Cover(d) denote the restriction of the Vertex Cover to instances in which each vertex of the graph is of degree at most d.

Theorem

There is a a gap-preserving reduction from MAX-3-SAT(29) to Vertex Cover(30) that transforms a boolean formula Φ *to a graph* $G = (V, E)$*, such that*

The problem Vertex Cover(d)

Definition

For each fixed $d \geq 1$, let Vertex Cover(d) denote the restriction of the Vertex Cover to instances in which each vertex of the graph is of degree at most d.

Theorem

There is a a gap-preserving reduction from MAX-3-SAT(29) to Vertex Cover(30) that transforms a boolean formula Φ to a graph $G = (V, E)$, such that

- *if $OPT(\Phi) = m$, then $OPT(G) \leq \frac{2}{3} \cdot |V|$,*

The problem Vertex Cover(d)

Definition

For each fixed $d \geq 1$, let Vertex Cover(d) denote the restriction of the Vertex Cover to instances in which each vertex of the graph is of degree at most d.

Theorem

There is a a gap-preserving reduction from MAX-3-SAT(29) to Vertex Cover(30) that transforms a boolean formula Φ to a graph $G = (V, E)$, such that

- *if* $OPT(\Phi) = m$, *then* $OPT(G) \leq \frac{2}{3} \cdot |V|$,
- *if* $OPT(\Phi) < (1 - \varepsilon_b) \cdot m$, *then* $OPT(G) > (1 + \varepsilon_v) \cdot \frac{2}{3} \cdot |V|$,

The problem Vertex Cover(d)

Definition

For each fixed $d \geq 1$, let Vertex Cover(d) denote the restriction of the Vertex Cover to instances in which each vertex of the graph is of degree at most d.

Theorem

There is a a gap-preserving reduction from MAX-3-SAT(29) to Vertex Cover(30) that transforms a boolean formula Φ to a graph $G = (V, E)$, such that

- *if $OPT(\Phi) = m$, then $OPT(G) \leq \frac{2}{3} \cdot |V|$,*
- *if $OPT(\Phi) < (1 - \varepsilon_b) \cdot m$, then $OPT(G) > (1 + \varepsilon_v) \cdot \frac{2}{3} \cdot |V|$,*

where m is the number of clauses in Φ, and $\varepsilon_v = \frac{\varepsilon_b}{2}$.

The problem Vertex Cover(30)

Proof.

- W.l.o.g. assume that each clause of Φ has exactly 3 literals (repeat literals within a clause if necessary).

The problem Vertex Cover(30)

Proof.

- W.l.o.g. assume that each clause of Φ has exactly 3 literals (repeat literals within a clause if necessary).
- Corresponding to each clause of Φ G has three vertices.

The problem Vertex Cover(30)

Proof.

- W.l.o.g. assume that each clause of Φ has exactly 3 literals (repeat literals within a clause if necessary).
- Corresponding to each clause of Φ G has three vertices. Each of these vertices is labeled with one literal of the clause.

The problem Vertex Cover(30)

Proof.

- W.l.o.g. assume that each clause of Φ has exactly 3 literals (repeat literals within a clause if necessary).
- Corresponding to each clause of Φ G has three vertices. Each of these vertices is labeled with one literal of the clause. Thus $|V| = 3 \cdot m$.

The problem Vertex Cover(30)

Proof.

- W.l.o.g. assume that each clause of Φ has exactly 3 literals (repeat literals within a clause if necessary).
- Corresponding to each clause of Φ G has three vertices. Each of these vertices is labeled with one literal of the clause. Thus $|V| = 3 \cdot m$.
- G has two types of edges: for each clause G has 3 edges connecting its 3 vertices,

The problem Vertex Cover(30)

Proof.

- W.l.o.g. assume that each clause of Φ has exactly 3 literals (repeat literals within a clause if necessary).
- Corresponding to each clause of Φ G has three vertices. Each of these vertices is labeled with one literal of the clause. Thus $|V| = 3 \cdot m$.
- G has two types of edges: for each clause G has 3 edges connecting its 3 vertices, and for each $u, v \in V$ if the literals labeling u and v are negations of each other, then (u, v) is an edge in G.

The problem Vertex Cover(30)

Proof.

- W.l.o.g. assume that each clause of Φ has exactly 3 literals (repeat literals within a clause if necessary).
- Corresponding to each clause of Φ G has three vertices. Each of these vertices is labeled with one literal of the clause. Thus $|V| = 3 \cdot m$.
- G has two types of edges: for each clause G has 3 edges connecting its 3 vertices, and for each $u, v \in V$ if the literals labeling u and v are negations of each other, then (u, v) is an edge in G.
- Each vertex of G has two edges of the first type, and at most 28 edges of the second type, hence G has degree at most 30.

The problem Vertex Cover(30)

Example

Let $\Phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$.

The problem Vertex Cover(30)

Example

Let $\Phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$. Then its corresponding graph will be:

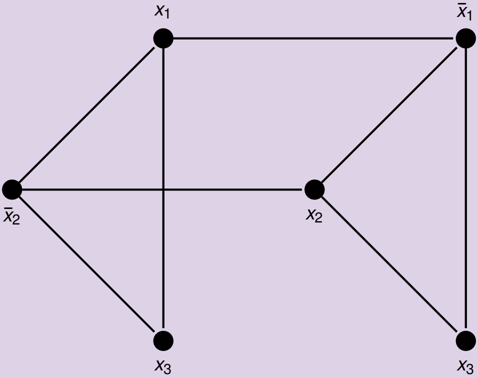

Figure: The graph G from reduction

The problem Vertex Cover(30)

Proof.

- Let us show that the size of maximum independent set in G is exactly $OPT(\Phi)$.

The problem Vertex Cover(30)

Proof.

- Let us show that the size of maximum independent set in G is exactly $OPT(\Phi)$.
- Consider an optimal assignment and pick one vertex corresponding to a satisfied literal from each clause.

The problem Vertex Cover(30)

Proof.

- Let us show that the size of maximum independent set in G is exactly $OPT(\Phi)$.
- Consider an optimal assignment and pick one vertex corresponding to a satisfied literal from each clause. Clearly, picked vertices form an independent set.

The problem Vertex Cover(30)

Proof.

- Let us show that the size of maximum independent set in G is exactly $OPT(\Phi)$.
- Consider an optimal assignment and pick one vertex corresponding to a satisfied literal from each clause. Clearly, picked vertices form an independent set.
- Conversely, consider an independent set I in G, and set the literals corresponding to its vertices to be true.

The problem Vertex Cover(30)

Proof.

- Let us show that the size of maximum independent set in G is exactly $OPT(\Phi)$.
- Consider an optimal assignment and pick one vertex corresponding to a satisfied literal from each clause. Clearly, picked vertices form an independent set.
- Conversely, consider an independent set I in G, and set the literals corresponding to its vertices to be true. Any extension of this assignment satisfies at least $|I|$ clauses.

The problem Vertex Cover(30)

Proof.

- Let us show that the size of maximum independent set in G is exactly $OPT(\Phi)$.
- Consider an optimal assignment and pick one vertex corresponding to a satisfied literal from each clause. Clearly, picked vertices form an independent set.
- Conversely, consider an independent set I in G, and set the literals corresponding to its vertices to be true. Any extension of this assignment satisfies at least $|I|$ clauses.
- The complement of a maximum independent set is a minimum vertex cover,

The problem Vertex Cover(30)

Proof.

- Let us show that the size of maximum independent set in G is exactly $OPT(\Phi)$.
- Consider an optimal assignment and pick one vertex corresponding to a satisfied literal from each clause. Clearly, picked vertices form an independent set.
- Conversely, consider an independent set I in G, and set the literals corresponding to its vertices to be true. Any extension of this assignment satisfies at least $|I|$ clauses.
- The complement of a maximum independent set is a minimum vertex cover, hence if $OPT(\Phi) = m$ then $OPT(G) = 2 \cdot m$.

The problem Vertex Cover(30)

Proof.

- Let us show that the size of maximum independent set in G is exactly $OPT(\Phi)$.
- Consider an optimal assignment and pick one vertex corresponding to a satisfied literal from each clause. Clearly, picked vertices form an independent set.
- Conversely, consider an independent set I in G, and set the literals corresponding to its vertices to be true. Any extension of this assignment satisfies at least $|I|$ clauses.
- The complement of a maximum independent set is a minimum vertex cover, hence if $OPT(\Phi) = m$ then $OPT(G) = 2 \cdot m$. On the other hand, if $OPT(\Phi) < (1 - \varepsilon_b) \cdot m$, then $OPT(G) > (1 + \varepsilon_b) \cdot m$. The proof is completed.

□

The Steiner Tree Problem

Theorem

There is a a gap-preserving reduction from Vertex Cover(30) to the Steiner Tree Problem.

The Steiner Tree Problem

Theorem

There is a a gap-preserving reduction from Vertex Cover(30) to the Steiner Tree Problem. It transforms an instance $G = (V, E)$ of Vertex Cover(30), to an instance $H = (R, S, cost)$ of Steiner tree problem, where R and S are the required and Steiner vertices of H, and cost is a metric defined on $R \cup S$.

The Steiner Tree Problem

Theorem

There is a a gap-preserving reduction from Vertex Cover(30) to the Steiner Tree Problem. It transforms an instance $G = (V, E)$ of Vertex Cover(30), to an instance $H = (R, S, cost)$ of Steiner tree problem, where R and S are the required and Steiner vertices of H, and cost is a metric defined on $R \cup S$. It satisfies

- *if $OPT(G) \leq \frac{2}{3} \cdot |V|$, then $OPT(H) \leq |R| + \frac{2}{3} \cdot |S| - 1$,*

The Steiner Tree Problem

Theorem

There is a a gap-preserving reduction from Vertex Cover(30) to the Steiner Tree Problem. It transforms an instance $G = (V, E)$ of Vertex Cover(30), to an instance $H = (R, S, cost)$ of Steiner tree problem, where R and S are the required and Steiner vertices of H, and cost is a metric defined on $R \cup S$. It satisfies

- *if $OPT(G) \leq \frac{2}{3} \cdot |V|$, then $OPT(H) \leq |R| + \frac{2}{3} \cdot |S| - 1$,*
- *if $OPT(G) > (1 + \varepsilon_v) \cdot \frac{2}{3} \cdot |V|$, then $OPT(H) > (1 + \varepsilon_s) \cdot (|R| + \frac{2}{3} \cdot |S| - 1)$,*

The Steiner Tree Problem

Theorem

There is a a gap-preserving reduction from Vertex Cover(30) to the Steiner Tree Problem. It transforms an instance $G = (V, E)$ of Vertex Cover(30), to an instance $H = (R, S, cost)$ of Steiner tree problem, where R and S are the required and Steiner vertices of H, and cost is a metric defined on $R \cup S$. It satisfies

- *if* $OPT(G) \leq \frac{2}{3} \cdot |V|$, *then* $OPT(H) \leq |R| + \frac{2}{3} \cdot |S| - 1$,
- *if* $OPT(G) > (1 + \varepsilon_v) \cdot \frac{2}{3} \cdot |V|$, *then* $OPT(H) > (1 + \varepsilon_s) \cdot (|R| + \frac{2}{3} \cdot |S| - 1)$,

where $\varepsilon_s = \frac{4\varepsilon_v}{97}$.

The Steiner Tree Problem

Proof.

- The instance $H = (R, S, cost)$ will have a required vertex r_e for each edge $e \in E$, and a Steiner vertex s_v for each vertex $v \in V$.

The Steiner Tree Problem

Proof.

- The instance $H = (R, S, cost)$ will have a required vertex r_e for each edge $e \in E$, and a Steiner vertex s_v for each vertex $v \in V$.
- Costs are defined as follows: an edge between a pair of Steiner vertices is of cost 1, and an edge between a pair of required vertices is of cost 2.

The Steiner Tree Problem

Proof.

- The instance $H = (R, S, cost)$ will have a required vertex r_e for each edge $e \in E$, and a Steiner vertex s_v for each vertex $v \in V$.
- Costs are defined as follows: an edge between a pair of Steiner vertices is of cost 1, and an edge between a pair of required vertices is of cost 2. An edge (r_e, s_v) is of cost 1 if edge e is incident at vertex v in G, and it is of cost 2 otherwise.

The Steiner Tree Problem

Proof.

- The instance $H = (R, S, cost)$ will have a required vertex r_e for each edge $e \in E$, and a Steiner vertex s_v for each vertex $v \in V$.
- Costs are defined as follows: an edge between a pair of Steiner vertices is of cost 1, and an edge between a pair of required vertices is of cost 2. An edge (r_e, s_v) is of cost 1 if edge e is incident at vertex v in G, and it is of cost 2 otherwise.
- Let us show that G has a vertex cover of size c if and only if H has a Steiner tree of cost $|R| + c - 1$.

The Steiner Tree Problem

Proof.

- The instance $H = (R, S, cost)$ will have a required vertex r_e for each edge $e \in E$, and a Steiner vertex s_v for each vertex $v \in V$.
- Costs are defined as follows: an edge between a pair of Steiner vertices is of cost 1, and an edge between a pair of required vertices is of cost 2. An edge (r_e, s_v) is of cost 1 if edge e is incident at vertex v in G, and it is of cost 2 otherwise.
- Let us show that G has a vertex cover of size c if and only if H has a Steiner tree of cost $|R| + c - 1$.
- Let S_c be the set of Steiner vertices in H corresponding to the c vertices in the cover.

The Steiner Tree Problem

Proof.

- The instance $H = (R, S, cost)$ will have a required vertex r_e for each edge $e \in E$, and a Steiner vertex s_v for each vertex $v \in V$.
- Costs are defined as follows: an edge between a pair of Steiner vertices is of cost 1, and an edge between a pair of required vertices is of cost 2. An edge (r_e, s_v) is of cost 1 if edge e is incident at vertex v in G, and it is of cost 2 otherwise.
- Let us show that G has a vertex cover of size c if and only if H has a Steiner tree of cost $|R| + c - 1$.
- Let S_c be the set of Steiner vertices in H corresponding to the c vertices in the cover. Observe that there is a tree in H covering $R \cup S_c$ using cost 1 edges only (since every edge $e \in E$ must be incident at a vertex in the cover).

The Steiner Tree Problem

Proof.

- The instance $H = (R, S, cost)$ will have a required vertex r_e for each edge $e \in E$, and a Steiner vertex s_v for each vertex $v \in V$.
- Costs are defined as follows: an edge between a pair of Steiner vertices is of cost 1, and an edge between a pair of required vertices is of cost 2. An edge (r_e, s_v) is of cost 1 if edge e is incident at vertex v in G, and it is of cost 2 otherwise.
- Let us show that G has a vertex cover of size c if and only if H has a Steiner tree of cost $|R| + c - 1$.
- Let S_c be the set of Steiner vertices in H corresponding to the c vertices in the cover. Observe that there is a tree in H covering $R \cup S_c$ using cost 1 edges only (since every edge $e \in E$ must be incident at a vertex in the cover). This Steiner tree has cost $|R| + c - 1$.

The Steiner Tree Problem

Proof.

- The instance $H = (R, S, cost)$ will have a required vertex r_e for each edge $e \in E$, and a Steiner vertex s_v for each vertex $v \in V$.
- Costs are defined as follows: an edge between a pair of Steiner vertices is of cost 1, and an edge between a pair of required vertices is of cost 2. An edge (r_e, s_v) is of cost 1 if edge e is incident at vertex v in G, and it is of cost 2 otherwise.
- Let us show that G has a vertex cover of size c if and only if H has a Steiner tree of cost $|R| + c - 1$.
- Let S_c be the set of Steiner vertices in H corresponding to the c vertices in the cover. Observe that there is a tree in H covering $R \cup S_c$ using cost 1 edges only (since every edge $e \in E$ must be incident at a vertex in the cover). This Steiner tree has cost $|R| + c - 1$.
- For the opposite direction, let T be a Steiner tree in H of cost $|R| + c - 1$.

The Steiner Tree Problem

Proof.

- The instance $H = (R, S, cost)$ will have a required vertex r_e for each edge $e \in E$, and a Steiner vertex s_v for each vertex $v \in V$.
- Costs are defined as follows: an edge between a pair of Steiner vertices is of cost 1, and an edge between a pair of required vertices is of cost 2. An edge (r_e, s_v) is of cost 1 if edge e is incident at vertex v in G, and it is of cost 2 otherwise.
- Let us show that G has a vertex cover of size c if and only if H has a Steiner tree of cost $|R| + c - 1$.
- Let S_c be the set of Steiner vertices in H corresponding to the c vertices in the cover. Observe that there is a tree in H covering $R \cup S_c$ using cost 1 edges only (since every edge $e \in E$ must be incident at a vertex in the cover). This Steiner tree has cost $|R| + c - 1$.
- For the opposite direction, let T be a Steiner tree in H of cost $|R| + c - 1$. We will show below that T can be transformed into a Steiner tree of the same cost that uses edges of cost 1 only.

The Steiner Tree Problem

Proof.

- The instance $H = (R, S, cost)$ will have a required vertex r_e for each edge $e \in E$, and a Steiner vertex s_v for each vertex $v \in V$.
- Costs are defined as follows: an edge between a pair of Steiner vertices is of cost 1, and an edge between a pair of required vertices is of cost 2. An edge (r_e, s_v) is of cost 1 if edge e is incident at vertex v in G, and it is of cost 2 otherwise.
- Let us show that G has a vertex cover of size c if and only if H has a Steiner tree of cost $|R| + c - 1$.
- Let S_c be the set of Steiner vertices in H corresponding to the c vertices in the cover. Observe that there is a tree in H covering $R \cup S_c$ using cost 1 edges only (since every edge $e \in E$ must be incident at a vertex in the cover). This Steiner tree has cost $|R| + c - 1$.
- For the opposite direction, let T be a Steiner tree in H of cost $|R| + c - 1$. We will show below that T can be transformed into a Steiner tree of the same cost that uses edges of cost 1 only.
- If so, the latter tree must contain exactly c Steiner vertices.

The Steiner Tree Problem

Proof.

- The instance $H = (R, S, cost)$ will have a required vertex r_e for each edge $e \in E$, and a Steiner vertex s_v for each vertex $v \in V$.
- Costs are defined as follows: an edge between a pair of Steiner vertices is of cost 1, and an edge between a pair of required vertices is of cost 2. An edge (r_e, s_v) is of cost 1 if edge e is incident at vertex v in G, and it is of cost 2 otherwise.
- Let us show that G has a vertex cover of size c if and only if H has a Steiner tree of cost $|R| + c - 1$.
- Let S_c be the set of Steiner vertices in H corresponding to the c vertices in the cover. Observe that there is a tree in H covering $R \cup S_c$ using cost 1 edges only (since every edge $e \in E$ must be incident at a vertex in the cover). This Steiner tree has cost $|R| + c - 1$.
- For the opposite direction, let T be a Steiner tree in H of cost $|R| + c - 1$. We will show below that T can be transformed into a Steiner tree of the same cost that uses edges of cost 1 only.
- If so, the latter tree must contain exactly c Steiner vertices. Moreover, every required vertex of H must have a unit cost edge to one of these Steiner vertices.

The Steiner Tree Problem

Proof.

- The instance $H = (R, S, cost)$ will have a required vertex r_e for each edge $e \in E$, and a Steiner vertex s_v for each vertex $v \in V$.
- Costs are defined as follows: an edge between a pair of Steiner vertices is of cost 1, and an edge between a pair of required vertices is of cost 2. An edge (r_e, s_v) is of cost 1 if edge e is incident at vertex v in G, and it is of cost 2 otherwise.
- Let us show that G has a vertex cover of size c if and only if H has a Steiner tree of cost $|R| + c - 1$.
- Let S_c be the set of Steiner vertices in H corresponding to the c vertices in the cover. Observe that there is a tree in H covering $R \cup S_c$ using cost 1 edges only (since every edge $e \in E$ must be incident at a vertex in the cover). This Steiner tree has cost $|R| + c - 1$.
- For the opposite direction, let T be a Steiner tree in H of cost $|R| + c - 1$. We will show below that T can be transformed into a Steiner tree of the same cost that uses edges of cost 1 only.
- If so, the latter tree must contain exactly c Steiner vertices. Moreover, every required vertex of H must have a unit cost edge to one of these Steiner vertices. Therefore, the corresponding c vertices of G form a cover.

The Steiner Tree Problem

Proof.

- Let (u, v) be an edge of cost 2 in T.

The Steiner Tree Problem

Proof.

- Let (u, v) be an edge of cost 2 in T. We may assume w.l.o.g. that u and v are both required.

The Steiner Tree Problem

Proof.

- Let (u, v) be an edge of cost 2 in T. We may assume w.l.o.g. that u and v are both required. If u is Steiner, remove (u, v) from T, getting two components. Throw in an edge from v to a required vertex to connect the two sides, and get a Steiner tree of the same cost as T.

The Steiner Tree Problem

Proof.

- Let (u, v) be an edge of cost 2 in T. We may assume w.l.o.g. that u and v are both required. If u is Steiner, remove (u, v) from T, getting two components. Throw in an edge from v to a required vertex to connect the two sides, and get a Steiner tree of the same cost as T.
- Let e_u and e_v be the edges, in G, corresponding to these vertices.

The Steiner Tree Problem

Proof.

- Let (u, v) be an edge of cost 2 in T. We may assume w.l.o.g. that u and v are both required. If u is Steiner, remove (u, v) from T, getting two components. Throw in an edge from v to a required vertex to connect the two sides, and get a Steiner tree of the same cost as T.
- Let e_u and e_v be the edges, in G, corresponding to these vertices. Since G is connected, there is a path, p, from one of the endpoints of e_u to one of the endpoints of e_v in G.

The Steiner Tree Problem

Proof.

- Let (u, v) be an edge of cost 2 in T. We may assume w.l.o.g. that u and v are both required. If u is Steiner, remove (u, v) from T, getting two components. Throw in an edge from v to a required vertex to connect the two sides, and get a Steiner tree of the same cost as T.
- Let e_u and e_v be the edges, in G, corresponding to these vertices. Since G is connected, there is a path, p, from one of the endpoints of e_u to one of the endpoints of e_v in G.
- Now, removing (u, v) from T gives two connected components. Let the set of required vertices in these two sets be R_1 and R_2.

The Steiner Tree Problem

Proof.

- Let (u, v) be an edge of cost 2 in T. We may assume w.l.o.g. that u and v are both required. If u is Steiner, remove (u, v) from T, getting two components. Throw in an edge from v to a required vertex to connect the two sides, and get a Steiner tree of the same cost as T.
- Let e_u and e_v be the edges, in G, corresponding to these vertices. Since G is connected, there is a path, p, from one of the endpoints of e_u to one of the endpoints of e_v in G.
- Now, removing (u, v) from T gives two connected components. Let the set of required vertices in these two sets be R_1 and R_2. Clearly, u and v lie in different sets, so path p must have two adjacent edges, say (a, b) and (b, c) such that their corresponding vertices, say w and w', lie in R_1 and R_2 , respectively.

The Steiner Tree Problem

Proof.

- Let (u, v) be an edge of cost 2 in T. We may assume w.l.o.g. that u and v are both required. If u is Steiner, remove (u, v) from T, getting two components. Throw in an edge from v to a required vertex to connect the two sides, and get a Steiner tree of the same cost as T.
- Let e_u and e_v be the edges, in G, corresponding to these vertices. Since G is connected, there is a path, p, from one of the endpoints of e_u to one of the endpoints of e_v in G.
- Now, removing (u, v) from T gives two connected components. Let the set of required vertices in these two sets be R_1 and R_2. Clearly, u and v lie in different sets, so path p must have two adjacent edges, say (a, b) and (b, c) such that their corresponding vertices, say w and w', lie in R_1 and R_2 , respectively.
- Let the Steiner vertex, in H, corresponding to b be s_b.

The Steiner Tree Problem

Proof.

- Let (u, v) be an edge of cost 2 in T. We may assume w.l.o.g. that u and v are both required. If u is Steiner, remove (u, v) from T, getting two components. Throw in an edge from v to a required vertex to connect the two sides, and get a Steiner tree of the same cost as T.
- Let e_u and e_v be the edges, in G, corresponding to these vertices. Since G is connected, there is a path, p, from one of the endpoints of e_u to one of the endpoints of e_v in G.
- Now, removing (u, v) from T gives two connected components. Let the set of required vertices in these two sets be R_1 and R_2. Clearly, u and v lie in different sets, so path p must have two adjacent edges, say (a, b) and (b, c) such that their corresponding vertices, say w and w', lie in R_1 and R_2 , respectively.
- Let the Steiner vertex, in H, corresponding to b be s_b. Now, throwing in the edges (s_b, w) and (s_b, w') must connect the two components. Observe that these two edges are of unit cost.

The Steiner Tree Problem

Proof.

- Let (u, v) be an edge of cost 2 in T. We may assume w.l.o.g. that u and v are both required. If u is Steiner, remove (u, v) from T, getting two components. Throw in an edge from v to a required vertex to connect the two sides, and get a Steiner tree of the same cost as T.
- Let e_u and e_v be the edges, in G, corresponding to these vertices. Since G is connected, there is a path, p, from one of the endpoints of e_u to one of the endpoints of e_v in G.
- Now, removing (u, v) from T gives two connected components. Let the set of required vertices in these two sets be R_1 and R_2. Clearly, u and v lie in different sets, so path p must have two adjacent edges, say (a, b) and (b, c) such that their corresponding vertices, say w and w', lie in R_1 and R_2 , respectively.
- Let the Steiner vertex, in H, corresponding to b be s_b. Now, throwing in the edges (s_b, w) and (s_b, w') must connect the two components. Observe that these two edges are of unit cost.
- Now, if $OPT(G) \leq \frac{2}{3} \cdot |V|$, then $OPT(H) \leq |R| + \frac{2}{3} \cdot |S| - 1$.

The Steiner Tree Problem

Proof.

- Let (u, v) be an edge of cost 2 in T. We may assume w.l.o.g. that u and v are both required. If u is Steiner, remove (u, v) from T, getting two components. Throw in an edge from v to a required vertex to connect the two sides, and get a Steiner tree of the same cost as T.
- Let e_u and e_v be the edges, in G, corresponding to these vertices. Since G is connected, there is a path, p, from one of the endpoints of e_u to one of the endpoints of e_v in G.
- Now, removing (u, v) from T gives two connected components. Let the set of required vertices in these two sets be R_1 and R_2. Clearly, u and v lie in different sets, so path p must have two adjacent edges, say (a, b) and (b, c) such that their corresponding vertices, say w and w', lie in R_1 and R_2 , respectively.
- Let the Steiner vertex, in H, corresponding to b be s_b. Now, throwing in the edges (s_b, w) and (s_b, w') must connect the two components. Observe that these two edges are of unit cost.
- Now, if $OPT(G) \leq \frac{2}{3} \cdot |V|$, then $OPT(H) \leq |R| + \frac{2}{3} \cdot |S| - 1$. On the other hand, if $OPT(G) > (1 + \varepsilon_v) \cdot \frac{2}{3} \cdot |V|$, then $OPT(H) > |R| + (1 + \varepsilon_v) \cdot \frac{2}{3} \cdot |S| - 1$. The proof of the theorem is completed.

$\square$

The maximum clique problem

Theorem

For fixed constants b and q, there is a gap-introducing reduction from SAT to clique problem that transforms a boolean formula Φ of size n to a graph $G=(V,E)$, where $|V|=2^q \cdot n^b$ such that

- *if Φ is satisfiable, then $OPT(G) \geq n^b$,*

The maximum clique problem

Theorem

For fixed constants b and q, there is a gap-introducing reduction from SAT to clique problem that transforms a boolean formula Φ of size n to a graph $G = (V, E)$, where $|V| = 2^q \cdot n^b$ such that

- *if Φ is satisfiable, then $OPT(G) \geq n^b$,*
- *if Φ is not satisfiable, then $OPT(G) < \frac{1}{2} \cdot n^b$.*

The maximum clique problem

Theorem

For fixed constants b and q, there is a gap-introducing reduction from SAT to clique problem that transforms a boolean formula Φ of size n to a graph $G = (V, E)$, where $|V| = 2^q \cdot n^b$ such that

- *if Φ is satisfiable, then $OPT(G) \geq n^b$,*
- *if Φ is not satisfiable, then $OPT(G) < \frac{1}{2} \cdot n^b$.*

Corollary

There is no $\frac{1}{2}$-approximation algorithm for maximum clique problem unless **P=NP***.*

The maximum clique problem

Proof.

- Let F be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT that requires $b \cdot \log(n)$ random bits and queries q bits of the proof.

The maximum clique problem

Proof.

- Let F be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT that requires $b \cdot \log(n)$ random bits and queries q bits of the proof.
- We will transform a SAT instance Φ of size n to a graph $G = (V, E)$ constructed as follows:

The maximum clique problem

Proof.

- Let F be a **PCP**$(\log(n), 1)$ verifier for SAT that requires $b \cdot \log(n)$ random bits and queries q bits of the proof.
- We will transform a SAT instance Φ of size n to a graph $G = (V, E)$ constructed as follows: for each choice of random string of length $b \cdot \log(n)$ and each assignment τ to q Boolean variables, there is a vertex $v_{r,\tau}$ in G.

The maximum clique problem

Proof.

- Let F be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT that requires $b \cdot \log(n)$ random bits and queries q bits of the proof.
- We will transform a SAT instance Φ of size n to a graph $G = (V, E)$ constructed as follows: for each choice of random string of length $b \cdot \log(n)$ and each assignment τ to q Boolean variables, there is a vertex $v_{r,\tau}$ in G. Observe that $|V| = 2^q \cdot n^b$.

The maximum clique problem

Proof.

- Let F be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT that requires $b \cdot \log(n)$ random bits and queries q bits of the proof.
- We will transform a SAT instance Φ of size n to a graph $G = (V, E)$ constructed as follows: for each choice of random string of length $b \cdot \log(n)$ and each assignment τ to q Boolean variables, there is a vertex $v_{r,\tau}$ in G. Observe that $|V| = 2^q \cdot n^b$.
- Let $Q(r)$ be the q positions in the proof that F queries when it is given string r as a random string.

The maximum clique problem

Proof.

- Let F be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT that requires $b \cdot \log(n)$ random bits and queries q bits of the proof.
- We will transform a SAT instance Φ of size n to a graph $G = (V, E)$ constructed as follows: for each choice of random string of length $b \cdot \log(n)$ and each assignment τ to q Boolean variables, there is a vertex $v_{r,\tau}$ in G. Observe that $|V| = 2^q \cdot n^b$.
- Let $Q(r)$ be the q positions in the proof that F queries when it is given string r as a random string.
- We will say that vertex $v_{r,\tau}$ is accepting if F accepts when it is given random string r and when it reads τ in the $Q(r)$ positions of the proof,

The maximum clique problem

Proof.

- Let F be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT that requires $b \cdot \log(n)$ random bits and queries q bits of the proof.
- We will transform a SAT instance Φ of size n to a graph $G = (V, E)$ constructed as follows: for each choice of random string of length $b \cdot \log(n)$ and each assignment τ to q Boolean variables, there is a vertex $v_{r,\tau}$ in G. Observe that $|V| = 2^q \cdot n^b$.
- Let $Q(r)$ be the q positions in the proof that F queries when it is given string r as a random string.
- We will say that vertex $v_{r,\tau}$ is accepting if F accepts when it is given random string r and when it reads τ in the $Q(r)$ positions of the proof, and it is rejecting otherwise.

The maximum clique problem

Proof.

- Let F be a **PCP**$(\log(n), 1)$ verifier for SAT that requires $b \cdot \log(n)$ random bits and queries q bits of the proof.
- We will transform a SAT instance Φ of size n to a graph $G = (V, E)$ constructed as follows: for each choice of random string of length $b \cdot \log(n)$ and each assignment τ to q Boolean variables, there is a vertex $v_{r,\tau}$ in G. Observe that $|V| = 2^q \cdot n^b$.
- Let $Q(r)$ be the q positions in the proof that F queries when it is given string r as a random string.
- We will say that vertex $v_{r,\tau}$ is accepting if F accepts when it is given random string r and when it reads τ in the $Q(r)$ positions of the proof, and it is rejecting otherwise.
- We will say that vertices v_{r_1,τ_1} and v_{r_2,τ_2} are consistent, if τ_1 and τ_2 agree at each position at which $Q(r_1)$ and $Q(r_2)$ overlap.

The maximum clique problem

Proof.

- Let F be a **PCP**$(\log(n), 1)$ verifier for SAT that requires $b \cdot \log(n)$ random bits and queries q bits of the proof.
- We will transform a SAT instance Φ of size n to a graph $G = (V, E)$ constructed as follows: for each choice of random string of length $b \cdot \log(n)$ and each assignment τ to q Boolean variables, there is a vertex $v_{r,\tau}$ in G. Observe that $|V| = 2^q \cdot n^b$.
- Let $Q(r)$ be the q positions in the proof that F queries when it is given string r as a random string.
- We will say that vertex $v_{r,\tau}$ is accepting if F accepts when it is given random string r and when it reads τ in the $Q(r)$ positions of the proof, and it is rejecting otherwise.
- We will say that vertices v_{r_1,τ_1} and v_{r_2,τ_2} are consistent, if τ_1 and τ_2 agree at each position at which $Q(r_1)$ and $Q(r_2)$ overlap. Clearly, a necessary condition for consistency is that $r_1 \neq r_2$.

The maximum clique problem

Proof.

- Let F be a **PCP**$(\log(n), 1)$ verifier for SAT that requires $b \cdot \log(n)$ random bits and queries q bits of the proof.
- We will transform a SAT instance Φ of size n to a graph $G = (V, E)$ constructed as follows: for each choice of random string of length $b \cdot \log(n)$ and each assignment τ to q Boolean variables, there is a vertex $v_{r,\tau}$ in G. Observe that $|V| = 2^q \cdot n^b$.
- Let $Q(r)$ be the q positions in the proof that F queries when it is given string r as a random string.
- We will say that vertex $v_{r,\tau}$ is accepting if F accepts when it is given random string r and when it reads τ in the $Q(r)$ positions of the proof, and it is rejecting otherwise.
- We will say that vertices v_{r_1,τ_1} and v_{r_2,τ_2} are consistent, if τ_1 and τ_2 agree at each position at which $Q(r_1)$ and $Q(r_2)$ overlap. Clearly, a necessary condition for consistency is that $r_1 \neq r_2$.
- Two distinct vertices v_{r_1,τ_1} and v_{r_2,τ_2} are connected by an edge in G if and only if they are consistent and they are both accepting.

The maximum clique problem

Proof.

- Let F be a $\mathbf{PCP}(\log(n), 1)$ verifier for SAT that requires $b \cdot \log(n)$ random bits and queries q bits of the proof.
- We will transform a SAT instance Φ of size n to a graph $G = (V, E)$ constructed as follows: for each choice of random string of length $b \cdot \log(n)$ and each assignment τ to q Boolean variables, there is a vertex $v_{r,\tau}$ in G. Observe that $|V| = 2^q \cdot n^b$.
- Let $Q(r)$ be the q positions in the proof that F queries when it is given string r as a random string.
- We will say that vertex $v_{r,\tau}$ is accepting if F accepts when it is given random string r and when it reads τ in the $Q(r)$ positions of the proof, and it is rejecting otherwise.
- We will say that vertices v_{r_1,τ_1} and v_{r_2,τ_2} are consistent, if τ_1 and τ_2 agree at each position at which $Q(r_1)$ and $Q(r_2)$ overlap. Clearly, a necessary condition for consistency is that $r_1 \neq r_2$.
- Two distinct vertices v_{r_1,τ_1} and v_{r_2,τ_2} are connected by an edge in G if and only if they are consistent and they are both accepting.
- Vertex $v_{r,\tau}$ is consistent with proof p if positions $Q(r)$ of p contains τ .

The maximum clique problem

Proof.

- If Φ is satisfiable, then there is a proof, p, on which F accepts for each choice, r, of the random string.

The maximum clique problem

Proof.

- If Φ is satisfiable, then there is a proof, p, on which F accepts for each choice, r, of the random string. For each r let $p(r)$ be the truth setting assigned by proof p to positions $Q(r)$.
- Now, the vertices $\{v_{r,p(r)} : |r| = b \cdot \log(n)\}$ form a clique of G of size n^b.

The maximum clique problem

Proof.

- If Φ is satisfiable, then there is a proof, p, on which F accepts for each choice, r, of the random string. For each r let $p(r)$ be the truth setting assigned by proof p to positions $Q(r)$.
- Now, the vertices $\{v_{r,p(r)} : |r| = b \cdot \log(n)\}$ form a clique of G of size n^b.
- Suppose that Φ is not satisfiable, and let C be a clique in G.

The maximum clique problem

Proof.

- If Φ is satisfiable, then there is a proof, p, on which F accepts for each choice, r, of the random string. For each r let $p(r)$ be the truth setting assigned by proof p to positions $Q(r)$.
- Now, the vertices $\{v_{r,p(r)} : |r| = b \cdot \log(n)\}$ form a clique of G of size n^b.
- Suppose that Φ is not satisfiable, and let C be a clique in G. Since the vertices of C are pairwise consistent, there is a proof, p, that is consistent with all vertices of C.

The maximum clique problem

Proof.

- If Φ is satisfiable, then there is a proof, p, on which F accepts for each choice, r, of the random string. For each r let $p(r)$ be the truth setting assigned by proof p to positions $Q(r)$.
- Now, the vertices $\{v_{r,p(r)} : |r| = b \cdot \log(n)\}$ form a clique of G of size n^b.
- Suppose that Φ is not satisfiable, and let C be a clique in G. Since the vertices of C are pairwise consistent, there is a proof, p, that is consistent with all vertices of C.Therefore, the probability of acceptance of F on proof p is at least $\frac{|C|}{n^b}$ (notice that the vertices of C must correspond to distinct random strings).

The maximum clique problem

Proof.

- If Φ is satisfiable, then there is a proof, p, on which F accepts for each choice, r, of the random string. For each r let $p(r)$ be the truth setting assigned by proof p to positions $Q(r)$.
- Now, the vertices $\{v_{r,p(r)} : |r| = b \cdot \log(n)\}$ form a clique of G of size n^b.
- Suppose that Φ is not satisfiable, and let C be a clique in G. Since the vertices of C are pairwise consistent, there is a proof, p, that is consistent with all vertices of C.Therefore, the probability of acceptance of F on proof p is at least $\frac{|C|}{n^b}$ (notice that the vertices of C must correspond to distinct random strings).
- Since the probability of acceptance of any proof is $< \frac{1}{2}$, the largest clique in G must be of size $< \frac{1}{2} \cdot n^b$.

$\square$

Extending the definition of the class **PCP**

Definition

The new definition of the class **PCP** comes with two additional parameters, c and s called completeness and soundness.

Extending the definition of the class **PCP**

Definition

The new definition of the class **PCP** comes with two additional parameters, c and s called completeness and soundness. For two functions $r(n)$ and $q(n)$, the class $\mathbf{PCP}_{c,s}(r(n), q(n))$ is comprised of all languages L,

Extending the definition of the class **PCP**

Definition

The new definition of the class **PCP** comes with two additional parameters, c and s called completeness and soundness. For two functions $r(n)$ and $q(n)$, the class $\mathbf{PCP}_{c,s}(r(n), q(n))$ is comprised of all languages L, for which there is a verifier V, such that on any input x the verifier obtains $O(r(n))$ random bits and quires $O(q(n))$ bits of the proof.

Extending the definition of the class **PCP**

Definition

The new definition of the class **PCP** comes with two additional parameters, c and s called completeness and soundness. For two functions $r(n)$ and $q(n)$, the class $\mathbf{PCP}_{c,s}(r(n), q(n))$ is comprised of all languages L, for which there is a verifier V, such that on any input x the verifier obtains $O(r(n))$ random bits and quires $O(q(n))$ bits of the proof. Furthermore,

- if $x \in L$, then there is a proof y that makes V accept with probability $\geq c$,

Extending the definition of the class **PCP**

Definition

The new definition of the class **PCP** comes with two additional parameters, c and s called completeness and soundness. For two functions $r(n)$ and $q(n)$, the class $\mathbf{PCP}_{c,s}(r(n), q(n))$ is comprised of all languages L, for which there is a verifier V, such that on any input x the verifier obtains $O(r(n))$ random bits and quires $O(q(n))$ bits of the proof. Furthermore,

- if $x \in L$, then there is a proof y that makes V accept with probability $\geq c$,
- if $x \notin L$, then for every proof y, V accepts with probability $< s$.

Extending the definition of the class **PCP**

Definition

The new definition of the class **PCP** comes with two additional parameters, c and s called completeness and soundness. For two functions $r(n)$ and $q(n)$, the class $\mathbf{PCP}_{c,s}(r(n), q(n))$ is comprised of all languages L, for which there is a verifier V, such that on any input x the verifier obtains $O(r(n))$ random bits and quires $O(q(n))$ bits of the proof. Furthermore,

- if $x \in L$, then there is a proof y that makes V accept with probability $\geq c$,
- if $x \notin L$, then for every proof y, V accepts with probability $< s$.

Corollary

For the previously introduced class $\mathbf{PCP}(r(n), q(n))$*, one has the following equality:* $\mathbf{PCP}(r(n), q(n)) = \mathbf{PCP}_{1,\frac{1}{2}}(r(n), q(n))$.

Extending the definition of the class **PCP**

Definition

The new definition of the class **PCP** comes with two additional parameters, c and s called completeness and soundness. For two functions $r(n)$ and $q(n)$, the class $\mathbf{PCP}_{c,s}(r(n), q(n))$ is comprised of all languages L, for which there is a verifier V, such that on any input x the verifier obtains $O(r(n))$ random bits and quires $O(q(n))$ bits of the proof. Furthermore,

- if $x \in L$, then there is a proof y that makes V accept with probability $\geq c$,
- if $x \notin L$, then for every proof y, V accepts with probability $< s$.

Corollary

For the previously introduced class $\mathbf{PCP}(r(n), q(n))$*, one has the following equality:* $\mathbf{PCP}(r(n), q(n)) = \mathbf{PCP}_{1,\frac{1}{2}}(r(n), q(n))$.

Remark

In general the parameters c and s may be functions of n.

Extending the definition of the class **PCP**

Definition

The new definition of the class **PCP** comes with two additional parameters, c and s called completeness and soundness. For two functions $r(n)$ and $q(n)$, the class $\mathbf{PCP}_{c,s}(r(n), q(n))$ is comprised of all languages L, for which there is a verifier V, such that on any input x the verifier obtains $O(r(n))$ random bits and quires $O(q(n))$ bits of the proof. Furthermore,

- if $x \in L$, then there is a proof y that makes V accept with probability $\geq c$,
- if $x \notin L$, then for every proof y, V accepts with probability $< s$.

Corollary

For the previously introduced class $\mathbf{PCP}(r(n), q(n))$*, one has the following equality:* $\mathbf{PCP}(r(n), q(n)) = \mathbf{PCP}_{1,\frac{1}{2}}(r(n), q(n))$.

Remark

In general the parameters c and s may be functions of n.

Theorem

The following equality holds: $\mathbf{NP} = \mathbf{PCP}_{1,\frac{1}{n}}(\log(n), \log(n))$.

Strong hardness result for maximum clique problem

Theorem

For fixed constants b and q, there is a gap-introducing reduction from SAT to clique problem that transforms a boolean formula Φ of size n to a graph $G = (V, E)$, where $|V| = n^{b+q}$ such that

- *if Φ is satisfiable, then $OPT(G) \geq n^b$,*

Strong hardness result for maximum clique problem

Theorem

For fixed constants b and q, there is a gap-introducing reduction from SAT to clique problem that transforms a boolean formula Φ of size n to a graph $G = (V, E)$, where $|V| = n^{b+q}$ such that

- *if Φ is satisfiable, then $OPT(G) \geq n^b$,*
- *if Φ is not satisfiable, then $OPT(G) < n^{b-1}$.*

Strong hardness result for maximum clique problem

Theorem

For fixed constants b and q, there is a gap-introducing reduction from SAT to clique problem that transforms a boolean formula Φ of size n to a graph $G = (V, E)$, where $|V| = n^{b+q}$ such that

- *if Φ is satisfiable, then $OPT(G) \geq n^b$,*
- *if Φ is not satisfiable, then $OPT(G) < n^{b-1}$.*

Corollary

There is no $\frac{1}{(n^{\varepsilon_q})}$-factor approximation algorithm for maximum clique problem, where $\varepsilon_q = \frac{1}{(b+q)}$ unless **P=NP***.*

Strong hardness result for maximum clique problem

Proof.

- Let F be a $\mathbf{PCP}_{1,\frac{1}{n}}(\log(n), \log(n))$ verifier for SAT, that requires $b \cdot \log(n)$ random bits and queries $q \cdot \log(n)$ bits of the proof.

Strong hardness result for maximum clique problem

Proof.

- Let F be a $\mathbf{PCP}_{1,\frac{1}{n}}(\log(n), \log(n))$ verifier for SAT, that requires $b \cdot \log(n)$ random bits and queries $q \cdot \log(n)$ bits of the proof. The transformation of SAT instance Φ to graph G is exactly as in $\frac{1}{2}$-inapproximability result. The only difference is that the increased number of bits queried results in a larger number of vertices.

Strong hardness result for maximum clique problem

Proof.

- Let F be a $\mathbf{PCP}_{1,\frac{1}{n}}(\log(n), \log(n))$ verifier for SAT, that requires $b \cdot \log(n)$ random bits and queries $q \cdot \log(n)$ bits of the proof. The transformation of SAT instance Φ to graph G is exactly as in $\frac{1}{2}$-inapproximability result. The only difference is that the increased number of bits queried results in a larger number of vertices.
- The correctness is also almost the same.

Strong hardness result for maximum clique problem

Proof.

- Let F be a $\mathbf{PCP}_{1,\frac{1}{n}}(\log(n), \log(n))$ verifier for SAT, that requires $b \cdot \log(n)$ random bits and queries $q \cdot \log(n)$ bits of the proof. The transformation of SAT instance Φ to graph G is exactly as in $\frac{1}{2}$-inapproximability result. The only difference is that the increased number of bits queried results in a larger number of vertices.
- The correctness is also almost the same.
- If Φ is satisfiable, then let p be a good proof, and pick the n^b vertices of G that are consistent with p, one for each choice of the random string. These vertices will form a clique in G.

Strong hardness result for maximum clique problem

Proof.

- Let F be a $\mathbf{PCP}_{1,\frac{1}{n}}(\log(n), \log(n))$ verifier for SAT, that requires $b \cdot \log(n)$ random bits and queries $q \cdot \log(n)$ bits of the proof. The transformation of SAT instance Φ to graph G is exactly as in $\frac{1}{2}$-inapproximability result. The only difference is that the increased number of bits queried results in a larger number of vertices.
- The correctness is also almost the same.
- If Φ is satisfiable, then let p be a good proof, and pick the n^b vertices of G that are consistent with p, one for each choice of the random string. These vertices will form a clique in G.
- Furthermore, any clique C in G gives rise to a proof that is accepted by F with probability $\geq \frac{|C|}{n^b}$.

Strong hardness result for maximum clique problem

Proof.

- Let F be a $\mathbf{PCP}_{1,\frac{1}{n}}(\log(n), \log(n))$ verifier for SAT, that requires $b \cdot \log(n)$ random bits and queries $q \cdot \log(n)$ bits of the proof. The transformation of SAT instance Φ to graph G is exactly as in $\frac{1}{2}$-inapproximability result. The only difference is that the increased number of bits queried results in a larger number of vertices.
- The correctness is also almost the same.
- If Φ is satisfiable, then let p be a good proof, and pick the n^b vertices of G that are consistent with p, one for each choice of the random string. These vertices will form a clique in G.
- Furthermore, any clique C in G gives rise to a proof that is accepted by F with probability $\geq \frac{|C|}{n^b}$. Since the soundness of F is $\frac{1}{n}$, if Φ is not satisfiable, then the largest clique is of size $< n^{b-1}$.

$\square$

Strong hardness result for set cover problem

Theorem

There is a constant $c > 0$ for which there is a randomized gap-introducing reduction Γ, requiring $n^{O(\log\log(n))}$ time, from SAT to set cover problem that transforms a boolean formula Φ to a set system $\mathscr{S}$ over a universal set of size $n^{O(\log\log(n))}$ such that

- *if Φ is satisfiable, then $OPT(\mathscr{S}) = 2 \cdot n^k$,*

Strong hardness result for set cover problem

Theorem

There is a constant $c > 0$ for which there is a randomized gap-introducing reduction Γ, requiring $n^{O(\log\log(n))}$ time, from SAT to set cover problem that transforms a boolean formula Φ to a set system $\mathscr{S}$ over a universal set of size $n^{O(\log\log(n))}$ such that

- *if Φ is satisfiable, then $OPT(\mathscr{S}) = 2 \cdot n^k$,*
- *if Φ is not satisfiable, then $Pr[OPT(\mathscr{S}) > c \cdot n^k k \log(n)] > \frac{1}{2}$,*

Strong hardness result for set cover problem

Theorem

There is a constant $c > 0$ for which there is a randomized gap-introducing reduction Γ, requiring $n^{O(\log\log(n))}$ time, from SAT to set cover problem that transforms a boolean formula Φ to a set system $\mathscr{S}$ over a universal set of size $n^{O(\log\log(n))}$ such that

- *if Φ is satisfiable, then $OPT(\mathscr{S}) = 2 \cdot n^k$,*
- *if Φ is not satisfiable, then $Pr[OPT(\mathscr{S}) > c \cdot n^k k \log(n)] > \frac{1}{2}$,*

where n is polynomial in the size of Φ. The parameter k is $\log\log(n)$.

The Book Used for Presentation

Vazirani's book

This talk is based on Chapter 29 of *V. V. Vazirani, "Approximation Algorithms", Springer, Corrected Second Printing 2003.*