,20

CMPUT 675: Approximation Algorithms **Fall 2011**

Lecture 19,20 (Nov 15&17, 2011): Hardness of Approximation, PCP theorem

Lecturer: Mohammad R. Salavatipour *Scribe: based on older notes*

19.1 Hardness of Approximation

So far we have been mostly talking about designing approximation algorithms and proving upper bounds. From no until the end of the course we will be talking about proving lower bounds (i.e. hardness of approximation).

We are familiar with the theory of NP-completeness. When we prove that a problem is NP-hard it implies that, assuming P≠NP there is no polynomail time algorithm that solves the problem (exactly).

For example, for SAT, deciding between Yes/No is hard (again assuming P≠NP). We would like to show that even deciding between those instances that are (almost) satisfiable and those that are far from being satisfiable is also hard. In other words, create a gap between Yes instances and No instances. These kinds of gaps imply hardness of approximation for optimization version of NP-hard problems.

In fact the PCP theorem (that we will see next lecture) is equivalent to the statement that MAX-SAT is NP-hard to approximate within a factor of $(1+\epsilon)$, for some fixed $\epsilon > 0$. Most of hardness of approximation results rely on PCP theorem.

For proving a hardness, for example for vertex cover, PCP shows that the existence of following reduction: Given a formula φ for SAT, we build a graph $G(V, E)$ in polytime such that:

- if φ is a yes-instance, then G has a vertex cover of size $\leq \frac{2}{3}|V|$;
- if φ is a no-instance, then every vertex cover of G has a size $> \alpha\frac{2}{3}|V|$ for some fixed $\alpha > 1$.

Corollary 1 *The vertex cover problem can not be approximated with a factor of α unless $P = NP$.*

In this reduction we have created a gap of size α between yes/no instances.

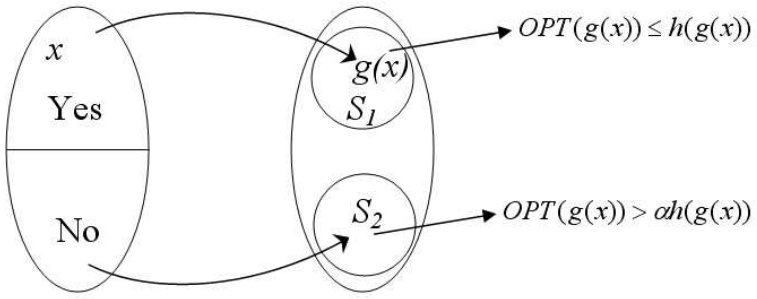

Figure 19.1: g is a gap-introducing reduction

Definition 1

Suppose that L is an NP-complete problem and π is a minimization problem. Suppose that g is a function computable in polytime that maps Yes-instances of L into a set S_1 of instances of π and No-instances of L into a set S_2 of instances of π. Assume that there is a polytime computable function h such that:

- *for every Yes-instance x of L: $OPT(g(x)) \leq h(g(x))$;*
- *for every No-instance x of L: $OPT(g(x)) > \alpha h(g(x))$.*

Then g is called a gap-introducing reduction from L to π and α is the size of the gap.

The chromatic number of a graph G, denoted by $\chi(G)$, is the smallest integer c such that G has a c-coloring. It is known that deciding whether a graph is 3-colorable or not is NP-complete. Using this we easily get the following hardness of approximation for the optimization problem of computing the chromatic number of a given graph G.

Theorem 1 *There is no better than $\frac{4}{3}$-approximation algorithm for computing the chromatic number, unless $P = NP$.*

Proof. Create a gap-introducing reduction from 3-colorability to the problem of computing chromatic numbers, as shown in the figure below. ■

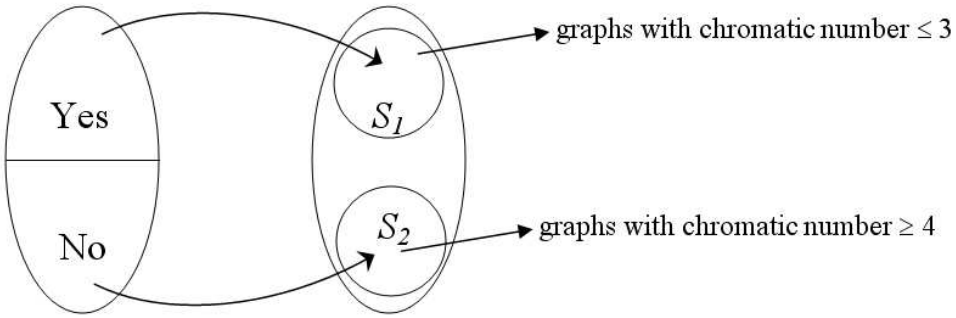

Figure 19.2: No better than $\frac{4}{3}$-approximation algorithm for computing the chromatic number, unless $P = NP$.

The reduction we gave for the (non-metric) traveling salesman problem in Lecture 5 was a gap-introducing reduction from Hamilton cycle. Once we have a gap-introducing reduction (e.g. from SAT) to π_1, then we can prove a hardness for another problem π_2 by giving a gap-preserving reduction from π_1 to π_2.

Definition 2 *A gap-preserving reduction g (g is polytime computable) from π_1 to π_2 has four parameters f_1, α, f_2, β. Given instance x of π_1, $g(x)$ is an instance of π_2 such that*

- *if $OPT(x) \leq f_1(x) \longrightarrow OPT(g(x)) \leq f_2(g(x))$;*
- *if $OPT(x) > \alpha f_1(x) \longrightarrow OPT(g(x)) > \beta f_2(g(x))$.*

19.1.1 MAX-SNP Problems and PCP Theorem

Many problems, such as Bin Packing and Knapsack, have PTAS's. A major open question was: does MAX-3SAT have a PTAS?

A significant result on this line was the paper by Papadimitriou and M. Yannakakis ('92) where they defined the class MAX-SNP. All problems in MAX-SNP have constant approximation algorithms.

They also defined the notion of completeness for this class and showed if a MAX-SNP-complete problem has a PTAS then every MAX-SNP problem has a PTAS. Several well-known problems including MAX-3SAT and TSP are MAX-SNP-complete. We have seen a $\frac{7}{8}$-approximation for MAX-3SAT. It turns out that even if there are not exactly 3 literals per clause this problem has a $\frac{7}{8}$-approximation algorithm. How about lower bounds?

There is a trivial gap-introducing reduction form 3-SAT to MAX-3SAT which shows that formula φ is a Yes-instance iff more than $\frac{m-1}{m}$ fraction of φ can be satisfied.

Suppose that for each $L \in NP$ there is a polytime computable g from L to instances of MAX-3SAT, such that

- for Yes-instance $y \in L$, all 3-clauses in $g(y)$ can be satisfied;
- for No-instance $y \in L$, at most 99% of the 3-clauses of $g(y)$ can be satisfied.

There is a standard proof system for $L \in NP$, that is, a deterministic verifier that takes as input an instance y of L together with a proof π, reads π and accepts iff π is a valid proof (solution).

We give an alternative proof system for NP using the gap-introducing reduction g assumed above. Define a proof that $y \in L$ to be a truth assignment satisfying $g(y)$ where g is the gap-instance reduction from L to MAX-3SAT. We define a randomized verifier V for L. V runs in polynomial time and

- takes y and the "new" proof π;
- accept iff π satisfies a 3-clause selected uniformly and randomly from $g(y)$.

If $y \in L$ then there is a proof (truth assignment) such that all the 3-clauses in $g(y)$ can be satisfied. For that proof Verifier $V(y, \pi)$ accepts with probability 1.

If $y \notin L$ then every truth assignment satisfies no greater of 99% of clauses in $g(y)$. So verifier $V(y, \pi)$ will reject with probability not less than $\frac{1}{100}$.

Note that under the assumption we made, we can achieve a constant separation between the two cases above by reading only 3 bits of the proof (regardless of the length of the proof). We can increase the probabability of success (i.e. decrease the probability of error) by simulating V a constant (but large) number of times. For example, by repeating 100 times, the probability of success will be at least $\frac{1}{2}$ in the second case and we are still checking a constant (300) bits of the proof.

19.2 PCP Theorem

To be consistent with the PCP definition later on, we give a slightly modified definition for class NP which is clearly equivalent to the original one:

Definition 3 *A language L is $\in NP$ if and only if there is a deterministic polynomial time verifier (i.e. algorithm) V that takes an input x and a proof y with $|y| = |x|^c$ for a constant $c > 0$ and it satisfies the following:*

- *Completeness: if* $x \in L \Rightarrow \exists\ y$ *such that* $V(x, y) = 1$.
- *Soundness: if* $x \notin L \Rightarrow \forall y,\ V(x, y) = 0$.

Now we define probabilistic verifiers that are restricted to look at (query about) only a few bits of the alleged proof y and make their decision based on those few bits instead of reading the whole proof.

Definition 4 *A* $(r(n), b(n))$*-restricted verifier is a randomized verifier that uses at most* $r(n)$ *random bits. It runs in probabilistic polynomial time and reads/queries at most* $b(n)$ *bits of the proof.*

Finally we provide the main definition of a language $\in PCP$. This definition is for $PCP_{1,\frac{1}{2}}(r(n), b(n))$.

Definition 5 *A language* L *is* $\in PCP(r(n), b(n))$ *if and only if there is a* $(r(n), b(n))$*-restricted verifier* V *such that given an input* x *with length* $|x| = n$ *and a proof* π*, it satisfies the following:*

- *Completeness: if* $x \in L \Rightarrow \exists$ *a proof* π *such that* $Pr[V(x, \pi) = 1] = 1$.
- *Soundness: if* $x \notin L \Rightarrow \forall \pi,\ Pr[V(x, \pi) = 1] \leq \frac{1}{2}$.

The probabilities in completeness and soundness given in definition above are 1 and $\frac{1}{2}$, respectively. A more general definition of PCP languages is by allowing these probabilities be some other constant values:

Definition 6 *For* $0 \leq s < c \leq 1$*, a language* L *is* $\in PCP_{c,s}(r(n), b(n))$ *if and only if there is a* $(r(n), b(n))$*-restricted verifier* V *such that given an input* x *with length* $|x| = n$ *and a proof* π*, it satisfies the following:*

- *Completeness: if* $x \in L \Rightarrow \exists$ *a proof* π *such that* $Pr[V(x, \pi) = 1] \geq C$.
- *Soundness: if* $x \notin L \Rightarrow \forall \pi,\ Pr[V(x, \pi) = 1] \leq S$.

In the more general definition of PCP language, we need to have the following restrictions on parameters:

- c and s are constants and $0 \leq S < C \leq 1$. This is to make sure the verifier can give a correct answer with higher probability than a wrong answer.
- $r(n)$ and $b(n)$ are at most polynomial, this is to make sure the verifier runs in polynomial.
- Proofs are at most $2^{r(n)}$ bits long. The reason is that the verifier V uses at most $r(n)$ random bits, so it can access at most $2^{r(n)}$ positions, and it must be able access to any position of the proof.

Now we give the first lemma about PCP:

Lemma 1 $PCP_{c,s}(O(\log n, n^{O(1)}) \subseteq NP$

Proof. Let L be a language in $PCP_{c,s}(O(\log n, n^{O(1)})$ with a verifier V. We construct a non-deterministic polytime Turing machine M for L. Starting with an input x, M guesses a proof π and simulates V on all $2^{O(\lg n)} = n^{O(1)}$ possible random bits. M accepts the proof π if at least a fraction c of all these runs accept, rejects otherwise. Thus:

- if $x \in L \Rightarrow V(x, \pi)$ accepts with $Prob \geq C \Rightarrow$ at least a fraction c of random bits cause the verifier V accept $\Rightarrow M$ accepts.

- if $x \notin L \Rightarrow$ the verifier accepts with $Prob \leq s < c \Rightarrow$ for only a fraction $< c$ of random bits the verifier V accepts $\Rightarrow M$ rejects.

Since there are $O(n^{O(1)})$ random bits and each simulation takes polytime, the running time of M is polytime. Therefore we get $PCP_{C,S}(O(\log n, n^{O(1)}) \subseteq NP$, and finished the proof. ■

A trivial observation about PCP is that if we do not read any random bits then it can be written as $PCP_{c,s}(0, n^{O(1)})$, this is just the same definition as NP. Therefore we get $NP \subseteq PCP_{c,s}(0, n^{O(1)})$. Combined with the lemma we just proved, we conclude that $PCP_{c,s}(O(\log n, n^{O(1)}) = NP$.

The remarkable PCP theorem, which is the least obvious (and probably the most difficult) result in computer science, proved by Arora and Safra [arora1] and Arora, Lund, Motwani, Sudan, and Szegedy [arora2] states:

Theorem 2 (PCP Theorem) $NP = PCP_{1,\frac{1}{2}}(O(\log n, O(1))$

Basically, this miraculas theorem says that for every problem in NP there is a verifier that queries only a constant number of bits of the proof (regardless of the length of the proof) and with sufficiently high probability gives a correct answer whether the proof is correct or not.

19.3 Hardness of MAX-3SAT

Starting from the PCP theorem, we show that approximating MAX-3SAT within some constant factor is NP-hard.

Theorem 3 *For some absolute constant $\epsilon > 0$, there is a gap-introducing reduction from SAT to MAX-3SAT such that it transforms a boolean formula ϕ for SAT to a boolean formula ψ with m clauses for MAX-3SAT such that:*

- *if ϕ is satisfiable, then $OPT(\psi) = m$.*
- *if ϕ is a NO-instance, then $OPT(\psi) \leq (1 - \epsilon)m$.*

Corollary 2 *Approximating MAX-3SAT with a factor better then $(1 - \epsilon)$ is NP-hard for some constant $\epsilon > 0$.*

Proof of Theorem 3: Since SAT is a NP problem, by PCP theorem, we know that it has a $PCP_{1,\frac{1}{2}}(O(\log n, n^{O(1)}$ verifier V. Let us assume that it is $PCP_{1,\frac{1}{2}}(d \log n, k)$ where d and k are some constants.

Let $r_1, \ldots, r_{n^d}$ be all the possible random bits (of length $d \log n$) that can be given as seed to verifier V. We will construct a formula f_i for every possible random bit r_i. Thus we will have formulas $f_1, \ldots, f_{n^d}$.

For any particular choice of random bits, the verifier can be considered to evaluate a boolean binary decision tree of height at most k. This tree contains at most 2^k variables and therefore, the total number of variables that we will have over all boolean formulas $f_1, \ldots, f_{n^d}$ will be $2^k n^d$. These variables correspond to the set of all possible positions that the verifier may read. We call this set of possible positions B.

This decision tree can be encoded as a boolean formula with at most 2^k variables and 2^k clauses each of length k. Think of every leaf is a variable and every path from root to leaf forms a clause.

Figure 1 shows an example. Here suppose $k = 2$ and we have a fixed random bit string. Based on the first random bit the position we read from the proof is x_j, if it returns 0 we get the second random bit and based on

that we read position x_k, else if x_j was 1 we read position x_l. So we can use four variables $\overline{x_j}, \overline{x_k}, x_j, x_l$ to form a formula encoding this tree: $(\overline{x_j} \wedge \overline{x_k}) \vee (x_j \wedge x_l)$. It is easy to see that the formula is satisfied if and only if the path that the verifier traverses on the tree ends at an "accept" leaf. Any truth assignment to the variables i.e. any proof, will give a unique path for each decision tree. If for a fixed random bit string and a proof (truth assignment) the path ends in an "accept" it means that the verifier accepts the proof, otherwise it rejects the proof.

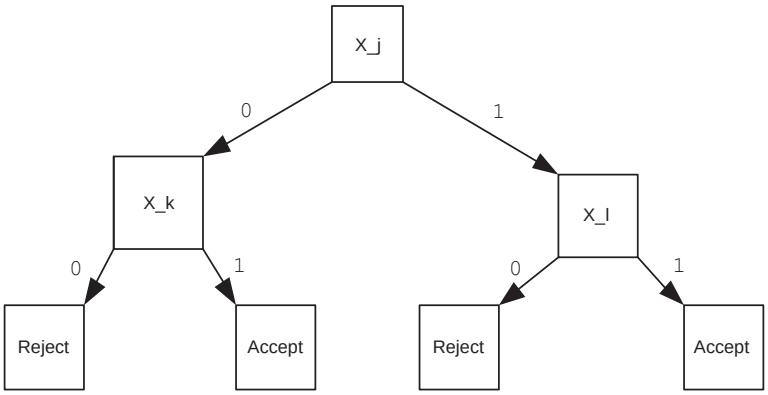

Figure 19.1: Example of decision tree

Alternately we can say that acceptance/rejection of V is a function of ϕ, r and the k bits V reads. Given ϕ, for any r_i we can consider the restriction of this function to the k bits of the proof. This is our function f_i and it can be computed in polynomial time.

If ϕ is a YES-instance, $\Rightarrow$ there is a truth assignment that works/accpets with probability of 1 (i.e., for any random bits it will accept) $\Rightarrow$ the corresponding truth assignment will give a path from root to an "accept" leaf in every decision tree (corresponding to a random bit string); so it satisfies all formulas $f_1, \ldots, f_{n^d}$.

If ϕ is a NO-instance $\Rightarrow$ for any proof (truth assignment) V accepts with probability $\leq \frac{1}{2}$ (this is from the PCP definition) $\Rightarrow$ for at least half of the decision trees, the truth assignment will give a root to leaf path that ends in "reject", i.e. the formula is not satisfied. Therefore, among all n^d formulas, at least $\frac{n^d}{2}$ of them are not satisfied.

Now on we can transform all formulas $f_1, \ldots, f_{n^d}$ into 3-CNF formulas $f'_1, \ldots, f'_{n^d}$ such that that f'_i is satisfiable if and only if f_i is. During the transformation (it is polynomial time) there will be some new disjoint variables created for each formula, however the number of these new variables is polynomial-bounded. In addition, the size of f'_i is at most k times of the size of f_i. This transformation is basically the Karp reduction used to prove NP-completeness of 3SAT from SAT and can be found in any standard text book (e.g. see the text book) It has the property that if f_i is satisfied then all the 3-clauses of f'_i are satisfied and if f_i is not satisfied then at least one 3-clause of f'_i is not satisfied.

Let $F = \bigcup_{i=1}^{n^d} f'_i$, the size of F (total number of clauses) is at most $k2^k \times n^d$. So the number of clauses that are not satisfiable in F (if ϕ is a No instance) is at least $\frac{n^d}{2}$; the ratio of the number of clauses that are not satisfied over the total number of clauses is $\frac{n^d/2}{k2^k \times n^d} = \frac{1}{k \times 2^{k+1}}$ then at most a $(1 - \frac{1}{k \times 2^{k+1}})$ fraction of F is satisfied. ■

This theorem shows how to derive a gap-introducing reduction from SAT to Max-3SAT. Interestingly, we can also derive the PCP theorem assuming the existence of such a reduction. The basic idea was explained in the previous lecture. There we showed how to derive a version of PCP theorem, assuming a gap-introducing reduction from SAT to Max-3SAT. These two results show that PCP theorem is equivalent to saying that

Max-3SAT is not approximable within some constant factor > 1 unless P=NP.

19.4 Improved PCP Theorems

As we saw in the last lecture, PCP theorem says: $NP = PCP_{1,\frac{1}{2}}(O(\log n), O(1))$. We are interested to reduce the number of query bits and also decrease the the probability of failure (i.e. better soundness). Our first theorem says that the number of query bits can be as small as 3 bits (and as you show in assignment 3 this is best possible).

Theorem 4 *For some $s < 1 : NP = PCP_{1,s}(O(\log n), 3)$*

Proof. Let $L \in NP$ be an arbitrary language and y be an instance of L. By PCP theorem we can construct a 3CNF formula F such that:

- if $y \in L$ there is a truth assignment for F such that all clauses of F are satisfied.
- if $y \notin L$ then for any truth assignment for F at most $(1 - \epsilon)$ fractions of clauses are satisfied.

We assume that the proof π given for y is the truth assignment to formula F above. The verifier V given y and proof π, first computes F in polytime. Then uses $O(\log n)$ bits to pick a random clause of F and query the truth assignment to its 3 variables (which of course requires only 3 bits of the proof). The verifier accepts if and only if the clause is satisfied. It is easy to see that:

- If $y \in L$ then there is a proof π s.t. V accepts with probability 1.
- If $y \notin L$ then for any proof π, V accepts with probability at most $1 - \epsilon$.

■

The downside of this theorem is that, although the number of query bits is best possible, the soundness probability is much worse than $\frac{1}{2}$. The following theorem shows that we can actually keep the soundness probability arbitrary close to $\frac{1}{2}$ while reading only 3 bits of the proof.

Theorem 5 (Guruswami/Sudan/Lewin/Trevisan/98) *For all $\epsilon > 0$*

$$NP = PCP_{1,\frac{1}{2}+\epsilon}(O(\log n), 3).$$

Note that the 3 bits of the proof are selected by the verifier adaptively. That is, the position of the second bit checked may depend on the truth value of the first bit read. This theorem is tight by the following result:

Theorem 6 $(Karloff/Zwick)$

$$P = PCP_{1,\frac{1}{2}}(O(\log n), 3).$$

Earlier, Håstad proved the following result from which several tight hardness results follow:

Theorem 7 (Håstad'97) $NP = PCP_{1-\epsilon,\frac{1}{2}+\epsilon}(O(\log n), 3)$ *where the verifier selects the 3 bits of the proof a priori. That is, the verifier uses $O(\log n)$ random bits to choose 3 positions, i_1, i_2, i_3 of the proof and a bit b and accepts if and only if $\pi(i_1) \oplus \pi(i_2) \oplus \pi(i_3) = b$.*

The following results follow from Håstad's theorem.

Corollary 3 *For any $\epsilon > 0$, it is NP-hard to approximate:*

- *Max-3SAT within a factor of $(\frac{7}{8} + \epsilon)$,*
- *Vertex cover within a factor of $(\frac{7}{6} + \epsilon)$,*
- *Maximum independent set is within a factor of $(\frac{1}{2} + \epsilon)$.*

Definition 7 *Given a system of linear equations module 2 with 3 distinct variable's per equation (e.g $X_1 \oplus X_2 \oplus X_3 = b$). The value of an assignment to the variable's is the number of equations satisfied by that assignment.*

Max-E3LIN2: Given a system of linear equations module 2 with 3 distinct variable's per equation, find an assignment with maximum number of satisfied equations.

Clearly, given an assignment we can find the value of the assignment in polynomial time. There is also a trivial $\frac{1}{2}$-approximation algorithm for this: consider the two assignments $x_i = 0$ for all i and $x_i = 1$ for all i. At least one of these two assignments satisfies at least half the equations. Surprisingly, this is the best possible approximation factor one can hope for.

Theorem 8 *Unless P=NP, there is no $(\frac{1}{2} + \epsilon)$ -approximation for Max-E3LIN2 for any $\epsilon > 0$.*

Proof. Given a language $L \in NP$ and an instance y for L consider the verifier V for L from Håstad PCP theorem. Assume that for every random string V is going to read 3 positions i_1, i_2, i_3 and accepts if and only if $\pi(i_1) \oplus \pi(i_2) \oplus \pi(i_3) = b$. So for every random bit string we have one linear equation.

We generate all the n^d random bits and from V and we get n^d equations, one for each string. For random string r_i, V accepts if and only if the truth assignment $\pi(i_1) \oplus \pi(i_2) \oplus \pi(i_3) = b$ satisfy that equation.

If y is yes instance then there is a proof for y such that V accepts with probability $\geq (1 - \epsilon)$, i.e. a fraction of $\geq (1 - \epsilon)$ of equations are satisfied.

If y is a no instance then for any any proof V accepts with probability $\leq \frac{1}{2} + \epsilon$, i.e. at most a fraction of $\leq \frac{1}{2} + \epsilon$ of equation are satisfied.

This implies a hardness gap of $\frac{\frac{1}{2}+\epsilon}{1-\epsilon} \leq \frac{1}{2} + 2\epsilon$. ■

Using this theorem we prove the hardness results mentioned in Corollary 3.

19.5 Hardness results for Max-3SAT, Vertex Cover, and Max Independent Set

Theorem 9 *Unless P=NP, there is no $(\frac{7}{8} + \epsilon)$-approximation algorithm for Max-3SAT, for any $\epsilon > 0$.*

Proof. We give a gap-preserving reduction form Max-E3LIN2 problem to Max-3SAT. The reduction maps every equation $x \oplus y \oplus z = b$ in the instance of Max-E3LIN2 into a 3CNF formula with 4 clauses depending on whether $b = 0$ or $b = 1$ as follows:

$x \oplus y \oplus z = 0, \Leftrightarrow (\overline{x} \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee y \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (x \vee y \vee \overline{z})$

$x \oplus y \oplus z = 1, \Leftrightarrow (x \vee y \vee z) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee y \vee \overline{z}) \wedge (\overline{x} \vee \overline{y} \vee \ z)$

Let I be the instance of Max-E3LIN2 and $\phi(I)$ be the 3CNF formula obtained. Clearly if I has m equations, then $\phi(I)$ has $4m$ 3-clauses.

If I is a yes instance, i.e. there is an assignment with value at least $(1-\epsilon)m$ then the same assignment satisfies at least $4(1-\epsilon)m + 3\epsilon m$ of clauses.

If I is a no instance, then every assignment has value at most $(\frac{1}{2}+\epsilon)m$. Therefore at most $4(\frac{1}{2}+\epsilon)m+3(\frac{1}{2}-\epsilon)m$ of clauses of $\phi(I)$ are satisfied by any truth assignment.

Thus the hardness gap is

$$\frac{4(\frac{1}{2}+\epsilon)+3(\frac{1}{2}-\epsilon)}{4(1-\epsilon)} = \frac{7}{8}+\epsilon',$$

where ϵ' is a constant depending on ϵ only.

■

Finally, we use Theorem 8 to prove hardness results for vertex cover and maximum independent set. Note that since a clique is the complement of an independent set, our hardness factor for maximum independent set is the hardness of Max-Clique too.

Theorem 10 *Unless P=NP, there is no $\frac{7}{6}+\epsilon$ -approximation for vertex cover problem and no $\frac{1}{2}+\epsilon$ -approximation for Clique/Independent set, for any $\epsilon > 0$.*

Proof. Note that the complement of a vertex cover is an independent set. We give a gap-preserving reduction from an instance of Max-E3LIN2. Starting from an instance I for Max-E3LIN2 we construct a graph G with $4m$ vertices where m is the number of equations of I.

For every equation there are exactly four assignments that satisfy the equation. We have one vertex for each of these 4 assignments. All four vertices of each equation are connected to each other.

We also add edges between pairs from different groups of 4 vertices that are "inconsistent". A pair of vertices from different equation are inconsistent if a variable is assigned different values in the assignment corresponding to those vertices. (see Figure 19.1).

If I is a yes instance then there is an assignment for I which satisfies at least $(1-\epsilon)m$ of equations. Consider the equations satisfied by this truth assignment and consider the assignment of these equations. Each of these assignments corresponds to a vertex of G. It is easy to see that these $(1-\epsilon)m$ vertices are all independent (i.e. no edge between them) because they are *not* inconsistent (come from the same truth assignment). Thus G has an independent set of size $\geq (1-\epsilon)m$, and so a vertex cover of size size $\leq (3+\epsilon)m$.

Now suppose that I is a no instance, i.e. every assignment has value at most $(\frac{1}{2}+\epsilon)m$. This implies that every set of size $(\frac{1}{2}+\epsilon)m+1$ in G must have an edge. The reason is that: clearly such a set has an edge if it has two vertices from the same set of 4 vertices. Also, if there is an independent set of size $(\frac{1}{2}+\epsilon)m+1$ then they correspond to a set of consistent assignments to equations that satisfy at least $(\frac{1}{2}+\epsilon)m+1$ of the equations, which cannot be true. Thus the every independent set of G has size at most $(\frac{1}{2}+\epsilon)m$, and every vertex cover has size at least $(3.5-\epsilon)m$.

The hardness gap for vertex cover is $\frac{\frac{(3.5-\epsilon)}{4}}{\frac{(3+\epsilon)}{4}} = \frac{7}{6}+\epsilon'$, and the hardness gap for independent set (and also clique) is $\frac{\frac{(\frac{1}{2}+\epsilon)}{4}}{\frac{1-\epsilon}{4}} = \frac{1}{2}+\epsilon'$.

■

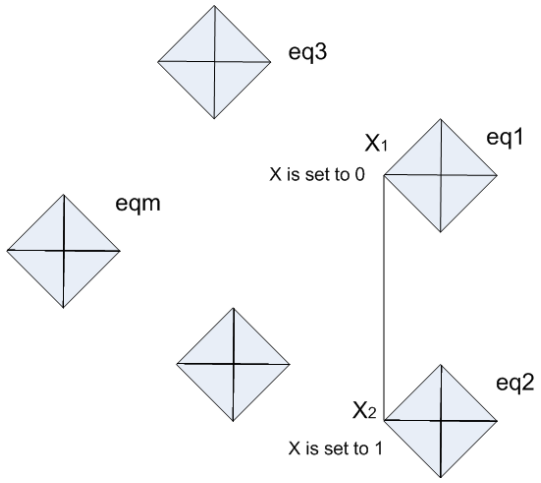

Figure 19.1: Reduction from Max-E3LIN2

19.6 Hardness of Clique

Definition 8 *(MAX-CLIQUE) Given a graph with n vertices, find a maximum size clique in it, i.e. a complete subgraph of maximum size.*

So far we have proved a constant hardness for Clique. The best known algorithm for approximating clique has a factor of $O(\frac{n}{\log^n})$ which is quite high. Note that this is slightly better than the trivial algorithm which picks a single vertex and returns (which has approximation factor $O(n)$). It turns out that we cannot do much better than this. Håstad has proved that for any $\epsilon > 0$ there is no polytime approximation algorithm for clique with factor $n^{\frac{1}{2}-\epsilon}$ (if P≠NP) or $n^{1-\epsilon}$ (if ZPP≠NP). Our goal is to prove a polynomial hardness for clique.

To prove this hardness result, we start with a constant hardness result for Clique. Then show that it is hard to approximate Clique within *any* constant factor. Finally, we show how to improve this to a polynomial.

Consider a $PCP_{1,\frac{1}{2}}(d \log n, q)$ verifier F for SAT, where d and q are constants. Let $r_1, r_2, ..., r_{n^d}$ be the set of all possible random strings to F. Given an instance ϕ of SAT, we construct a graph G from F and ϕ which will be an instance of Clique. G has one vertex $V_{r_i,\tau}$ for each pair (i, σ), where i is for the random string r_i and σ is a truth assignment to q variables. The number of of all the vertices (i.e. the size of the graph) is $|V| = n^d 2^q$.

Equivalently, we define the vertices as follows. An accepting transcript for F on ϕ with random string r_i is q pairs $(p_1, a_1), \ldots, (p_q, a_q)$ s.t. for every truth assignment that has values $a_1, \ldots, a_q$ for variables $p_1, \ldots, p_q$, verifier F given r_i checks positions $p_1, \ldots, p_q$ in that order and accepts. Note that once we have ϕ, r_i, $a_1, \ldots, a_q$ it is easy to compute $p_1, \ldots, p_q$. For each transcript we have a vertex in G.

Two vertices (i, σ) and (i', σ') are adjacent iff σ, σ' don't assign different values to same variable, i.e. they are consistent.

If ϕ is a yes instance: so there is a proof (truth assignment) π, such that F accepts (given π) on all random strings. For each r_i, there is a corresponding σ (which has the same answers as in π) and is an accepting transcript. We have n^d random strings and therefore there are n^d vertices of G (corresponding to those). They form a clique (because they come from the same truth assignment and therefore are consistent, i.e. adjacent); therefore G has a clique of size $\geq n^d$.

if ϕ is a no instance: we want to show that every clique in G has size at most $\frac{n^d}{2}$. By way of contradiction suppose we have a clique C of size $c > \frac{n^d}{2}$. Assume that $(i_1, \sigma_1)...(i_c, \sigma_c)$ are the vertices in the clique. Therefore the transcripts $\sigma_1...\sigma_c$ (partial truth assignment) are consistent. We can extend this truth assignment to a whole proof (truth assignment) such that on random strings $i_1, ..., i_c$, verifier F accepts. therefore verifier accepts for $> \frac{n^d}{2}$ strings, which contradicts the assumption that ϕ is a no instance.

The gap created here is exactly the soundness probability and the smaller S is, the larger gap we get.

By simulating a verifier $PCP_{1,\frac{1}{2}}(d \log n, q)$ for k times and accepting iff all of those simulations accept, we get a $PCP_{1,\frac{1}{2^k}}(k \cdot d \log n, k \cdot q)$ verifier. Note that in this case the size of the construction G is $n^{kd}2^{kq}$, which is polynomial as long as k is some constant. This will show a hardness of 2^k, which is a constant.

Corollary 4 *For any constant S, it is NP-hard to approximation clique within a factor of S. say ($S = 1/2^k$)*

To get an n^ε-gap, we need S to be polynomially small, and for that we need to repeat $k = \Omega(\log n)$ times. Therefore, $k \cdot q = O(\log n)$ which is Ok. But the length of random string becomes $k \log n = \Omega(\log^2 n)$, and the size of graph becomes super-polynomial. Therefore, to get a polynomial hardness result for Clique we need a PCP verifier with $O(\log n)$ random bits and polynomially small soundness probability, i.e. $PCP_{1,\frac{1}{n}}(O(\log n), O(\log n))$.

The trick here is to start with only $O(\log n)$ random bits and use expander graphs to generate $O(\log n)$ random strings, each of length about $\log n$.

Definition 9 *(Expander Graph) Every vertex has the same constant degree, say d, and for every non-empty set $S \subset V$, $|E(S, \overline{S})| \geq \min\{|S|, |\overline{S}|\}$.*

There are explicit construction of expander graphs. Let H be an expander graph with n^d nodes. For each node we assign a label which is a binary string of length $d \log n$. We can generate a random walk in H using only $O(\log n)$ random bits: we need $d \log n$ bits to choose the first vertex, and we only need constant number of random bits to choose the neighbor at every step. Therefore, to have a random walk of length $O(\log n)$ we need only $O(\log n)$ bits.

Theorem 11 *For any set S of vertices of H with $< \frac{n^d}{2}$ vertices, there is a constant k such that the probability that a random walk of length $k \log n$ lies entirely in S is $< \frac{1}{n}$.*

Here is the outline of the proof of this theorem. By definition of the expander graphs, if you have a set S of vertices, we expect a constant fraction of edges out of the vertices of S be going into $\overline{S}$. Therefore, if you start a random walk from a vertex in S, at every step, there is a constant probability that this walk jumps into $\overline{S}$. So the probability that a random walk of length $\Omega(\log n)$ stays entirely within S is polynomially small.

Next lecture, we will show how, given a PCP verifier F for a language in $PCP_{1,\frac{1}{2}}(d \log n, q)$, generate a PCP verifier F' which uses $O(\log n)$ random bits, queries only $O(\log n)$ bits, has completeness 1, and soundness $\frac{1}{n}$, i.e. $NP = PCP_{1,\frac{1}{n}}(O(\log n), O(\log n))$.

References

[1] S. Arora and S. Safra, *Probabilistic checking of proofs: a new characterization of NP*, J. ACM, 45(3):501-555, 1998. Earlier version in Proc. of IEEE FOCS 1992, pp 2-12.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, *Proof verification and intractability of approximation problems*, J. ACM, 45(1):70-122, 1998. Earlier version in Proc. IEEE FOCS 1992, pp 13-22.