CMPUT 675: Approximation Algorithms	**Winter 2005**

Lecture 20: Mar 30

Lecturer: Mohammad R. Salavatipour *Scribe: Jianjun Zhou*

20.1 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 20.1 *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.

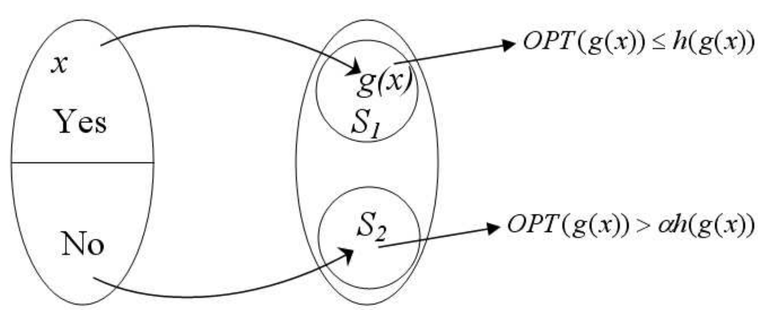

Figure 20.1: g is a gap-introducing reduction

Definition 20.2

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 20.3 *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. ∎

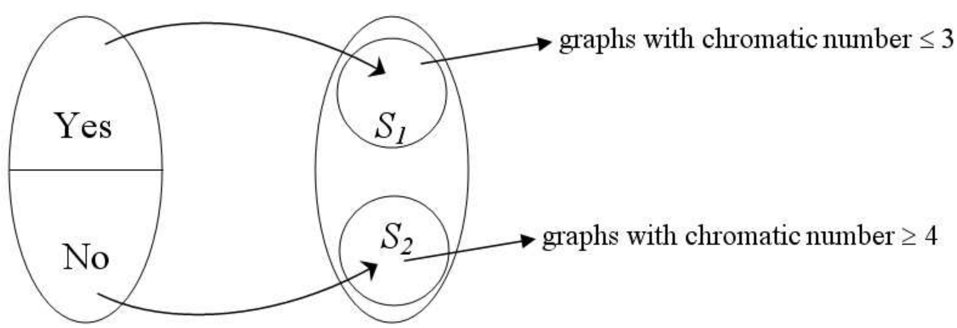

Figure 20.2: No better than $\frac{4}{3}$-approximation algorithm for computing the chromatic number, unless $P = NP$.

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 20.4 *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 $OPT(x) > \alpha f_1(x) \longrightarrow OPT(g(x)) > \beta f_2(g(x))$.*

20.1.1 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.