E0249 Approximation Algorithms

11 April, 2015

Lecture 14: Hardness of Approximation

Lecturer: Arnab Bhattacharyya *Scribe: Anurita Mathur & Divya Ravi*

14.1 Introduction

In previous lectures, we have seen techniques for designing good approximation algorithms. In this lecture, we move to techniques for proving that problems are hard to approximate within certain factors. We will look at several ways in which these results are proven. First is reduction from NP-complete problems. Second, we can see reductions from other optimization problems. Third, we reduce from probabilistically checkable proofs or PCPs. These PCPs allow us to prove hardness of approximation results for a number of particular constraint satisfaction problems. We can then use approximation preserving reductions from these constraint satisfaction problems to derive hardness results for a number of other problems. Fourth, we look at a particular problem called the label cover problem; reductions from label cover problems are used to prove certain kinds of hardness results. Last, we can show reductions to problems from the unique games problem, which gives hardness results conditional on the truth of the unique games conjecture.

14.2 Reduction from NP-complete problems

Definition 14.1. ***Reduction from A to B ($A \preceq B$) :***

Problem A is reducible to problem B if an algorithm for solving problem B efficiently(if it existed) could also be used as a subroutine to solve problem A efficiently. When this is true, solving A cannot be harder than solving B. In these reductions, a YES instance of A maps to a YES instances of B and NO instances of A maps to a NO instances of B.

14.2.1 Gap Problems

Gap problems are kind of a promise problem, i.e a problem where not all possible inputs are possible , only inputs that satisfy a promise (say $\pi(x) \geq c$ or $\pi(x) \leq s$). We define a *[a,b]-Gap problem* for the case of maximization as follows : -

Definition 14.2. *Given a maximization problem P and two real numbers $0 \leq a \leq b$, the $[a, b]$-gap-P problem is the taks of distinguishing between two cases: for an instance I of problem P:*

- $Opt(I) \geq b$
- $Opt(I) < a$

The instances satisfying the former case are the YES instances and the latter are the NO instances.

Theorem 14.3. *If the $[a, b]$-Gap version of a problem P is NP-hard, then approximating OPT for that problem within a factor of $\frac{a}{b}$ is also NP-hard.*

Proof. For the maximization case, suppose there is an approximation algorithm C that, for every instance x, outputs $\frac{a}{b}$OPT$(x) \leq C(x) \leq$ OPT. Consider the algorithm that outputs YES if $C(x) \geq a$, and NO otherwise. We claim that it solves the $[a, b]$-Gap version of P.

- If OPT(x) $\geq$ b (the correct answer is 'YES') , then necessarily
$$C(x) \geq \frac{a}{b} * OPT(x) \geq \frac{a}{b} * b = a$$
Thus we answer 'YES' which is correct.
- If OPT(x) $<$ a (the correct answer is 'NO') , then necessarily
$$C(x) \leq OPT(x) < a$$
Thus we answer 'NO' which is correct.

□

We now see an example of a polynomial time reduction:

14.2.2 Minimum Makespan Scheduling:

We are given m machines for scheduling, indexed by the set M={ 1,2,...m}. There are furthermore n jobs given by the set J={ 1,2,...n} where job j takes p_{ij} units of time if job j is scheduled on machine i. Our objective is to minimize the maximum processing time for any machine. We have seen a 2-factor algorithm for this General Assignment problem (GAP) in one of the previous lectures.

Claim 14.4. *There is a polynomial time reduction from 3D-MATCHING to [3,4-ϵ] gap scheduling problem.*

Proof. Given an instance of 3D-MATCHING - sets A, B, C of n elements each and sets of m triples T_1,...$T_m \in$ A x B x C . We first look at reduction to [3,4-ϵ]-Gap scheduling $\forall\ \epsilon > 0$. The idea behind the reduction is: Given an instance of 3-D MATCHING we can construct a scheduling input as follows. There is a job x for each of the 3n elements in A $\cup$ B $\cup$ C, and there is a machine for each of the given m triples in F. For each job j that corresponds to an element of the triple T_i, we set its processing time P_{xj} to 1, but otherwise we set the processing time to ∞. Thus, given a 3-D MATCHING, we can schedule all 3n of these jobs on n of the machines. This leaves m-n machines without any assigned jobs and we complete the reduction by introducing m-n dummy jobs that require exactly 3 time units on each machine i, i=1,....m.If m $<$ n, we can construct some trivial 'no' instance of the scheduling problem.
There is a schedule with makespan 3 if and only if there is a 3D Matching. Suppose there is a matching, we schedule the jobs corresponding to the elements in T_i = (a, b , c) on machine i and schedule the dummy jobs on the m - n triples not present in the matching. This gives a schedule with makespan 3. Now suppose there is such a schedule - Each of the dummy jobs requires three time units on any machine and is thus scheduled by itself on some machine. Consider the set of n machines that are not processing dummy jobs. Since these are processing all of the 3n element jobs, each of these jobs is processed in one time unit. Each three jobs that are assigned to one machine must therefore correspond to elements that form the triple corresponding to that machine. Since each element job is scheduled exactly once, the n triples corresponding to the machines that are not processing dummy jobs form a matching.Thus for every $\rho < \frac{4}{3}$, there does not exist a polynomial ρ - approximation algorithm for the minimum makespan problem. As we know, approximating for a problem within a factor of $\frac{a}{b}$ is NP-hard if the [a, b] gap version of the problem is hard ; we have reduced to [3, 4 - ϵ] version. In other words, now we can claim that if the reduction outputs YES instance of [3,4-ϵ] gap, then pre image must be YES instance of 3D-MATCHING. □

Using the same technique as above , we can refine it to yeild a stronger result .

Claim 14.5. *There is a polynomial time reduction from 3D-MATCHING to [2,3-ϵ] gap scheduling problem.*

Proof. In this case , the reduction is as follows - There is a job x for each element of B and C. If we let each dummy job take 2 time units on each machine, we have the property that if there is a 3-D MATCHING, then there is a schedule of length 2. The reduction is as follows : Say we call the triples that contain a_j triples of type j. Let t_j be the number of triples of type j for j = 1 , . . . , n. As before, machine i corresponds to the triple T_i for i = 1 , . . . , m. There are now only 2n element jobs, corresponding to the 2n elements of B $\cup$ C. We refine the construction of the dummy jobs: there are t_i - 1 dummy jobs of type j forj = 1 , . . . , n. (Note that the total number of dummy jobs is m - n, as before) Machine i corresponding to a triple of type j, say = (a_j, b_k, c_l) , can process each of the element jobs corresponding to b_k and c_t in one time unit and each of the dummy jobs of type j in two time units; all other jobs require three time units on machine i. Suppose there is a matching. For each T_i= (a_j, b_k, c_l) in the matching, schedule the element jobs corresponding to b_k and c_l on machine i. For each j, this leaves t_j - 1 idle machines corresponding to triples of type j that are not in the matching; schedule the t_j-1 dummy jobs of type j on these machines. This completes a schedule with makespan 2. Conversely, suppose that there is such a schedule. Each dummy job of type j is scheduled on a machine corresponding to a triple of type j. Therefore, there is exactly one machine corresponding to a triple of type j that is not processing dummy jobs for j = 1 , . . . , n. Each such machine is processing two element jobs in one time unit each. If the machine corresponds to a triple of type and its two unit-time jobs correspond to b_k and c_l, then (a_j, b_k, c_l) must be the triple corresponding to that machine. Since each element job is scheduled exactly once, the n triples corresponding to the machines that are not processing dummy jobs form a matching. Thus it follows from the above that for every $\rho < \frac{3}{2}$, there does not exist a polynomial ρ - approximation algorithm for the minimum makespan problem. As we know, approximating for a problem within a factor of $\frac{a}{b}$ is NP-hard if the [a, b] gap version of the problem is hard ; we have reduced to [2, 3 - ϵ] version. In other words, now we can claim that if the reduction outputs YES instance of [2,3-ϵ] gap, then pre image must be YES instance of 3D-MATCHING. □

14.3 Reduction from Optimization Problems

In this section we look at reductions from some standard optimization problems. We also turn to the idea of an *approximation-preserving reduction*. In such reductions, we reduce a problem π to π' so that if there exists an α-approximation for π' , we can get a f(α)-approximation algorithm for π, where f is some function. Then if we know that π is hard to approximate to within some factor, the reduction implies that the problem π' is also hard to approximate within some factor. We look at one such reduction -

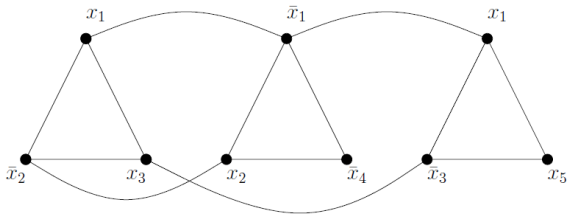

Figure 14.1: MAX 3-SAT to IND SET reduction

Theorem 14.6. *If MAXIMUM INDEPENDENT SET has an α-approximation then so does MAX 3-SAT.*

Proof. We reduce an instance of MAX 3-SAT to an instance of Max Independent set as follows : Suppose we have an instance F = $C_1 \wedge C_2 \wedge \ldots \wedge C_m$, where C_i is the disjunction of 3 variables, drawn from x_1, x_2, . . . , x_n and their

negations, $\neg(x_1), \neg(x_2), \ldots, \neg(x_n)$. We create the graph G as follows: For each variable in each clause, create a node, which we will label with the name of the variable. Therefore there may be multiple nodes with the label x_i or $\neg(x_i)$, if these variables appear in multiple clauses. For each clause, add an edge between the three nodes corresponding the variables from that clause. These three nodes and three edges are referred to as clause gadget. Finally, for all i, add an edge between every pair of nodes with one labeled x_i and the other labeled $\neg(x_i)$. The figure below shows the reduction for the three clauses $(x_1 \vee \neg(x_2) \vee x_3)$, $(\neg(x_1) \vee x_2 \vee \neg(x_4))$, $(x_1 \vee \neg(x_3) \vee x_5)$. Given any solution to an independent set instance, we can obtain a solution for 3-SAT by setting to true any x_i whose corresponding literal node is in the independent set. This leads to a consistent assignment to the variables since there is an edge between each variable and its complement. If for some variable, neither itself nor its complement is in the independent set, we can set it arbirarily. Atmost one literal node can be in the independent set for each clause; we satisy as many clauses as there are nodes in the independent set.Similarly given any solution to 3-SAT , for each satisfied clause we pick a literal that satisfies the clause and put the corresponding node in the independent set. Thus size of the optimal solution is the same .If there exists an independent set of size m , then there exists a satisfying assignment for the m clauses of the 3-SAT instance. Therefore, this is an approximation-preserving reduction - Any α approximation algorithm for MAX independent set yields an α approximation algorithm for MAX 3-SAT. □

Theorem 14.7. *If MAXIMUM INDEPENDENT SET has an α approximation for any $\alpha > 0$,then it has a $\sqrt{\alpha}$ approximation.*

Proof. Suppose we are given an α approximation algorithm for the maximum independent set problem, and we have an input graph G.We would like to find a solution of atleast $\sqrt{\alpha}$ k, where k is the size of the maximum independent set in G. To do this, we create a new graph G x G ,with a vertex set V'=VxV and an edge set E' in which there is an edge between $(u_1,u_2) \in$ V' and $(v_1,v_2) \in$ V' if either $(u_1,v_1) \in$ E or $(u_2,v_2) \in$ E.

Given an independent set S in G, we claim that S x S is an independent set in G x G ;this follows since for any $(u_1,u_2),(v_1,v_2) \in$ S x S,both $(u_1,v_1) \neq$ E and $(u_2,v_2) \neq$ E since S is independent. Furthermore, given an independent set S' $\subseteq$ V' in G x G , we claim that both S_1={u $\in$ V:$\exists$ (u,w) $\in$ S'} and S_2={u $\in$ V:$\exists$ (w,u) $\in$ S'} are independent sets in G. If both u,v $\in S_1$, then there exists $(u,w_1),(v,w_2) \in$ S'. Since S' is independent there can be no edge (u,v) $\in$ E. Thus given an independent set S in G, we can find an independent set of size atleast k^2 in G x G. Also, given an independent set S' in G x G, it is the case that S' $\subseteq S_1$ x S_2, so that $\|S'\| \leq \|S_1\|\|S_2|$; if we take the larger of the two sets S_1 and S_2, then we have an independent set in G of size atleast $\sqrt{k}$.

Given an α approximation algorithm for the maximum independent set problem and an instance G ,we construct the graph G x G , use the approximation algorithm to find an independent set S' in G x G of size atleast αk^2, then use this to find an independent set of size S in G of size atleast $\sqrt{\alpha}$k. Thus,this is an $\sqrt{\alpha}$-approximation algorithm for the maximum independent set problem □

14.4 Probabilistically Checkable Proofs:

Probabilistically Checkable proofs give us a direct way to show that certain constraint satisfaction problems cannot have particular performance guarantees unless P = NP. Let us recollect that NP is the set of problems that have a polynomial time verifier.

Definition 14.8. *Verifier is an algorithm V that takes as input instance x and proof π.*

- *If x is a YES instance,there is a proof π such that V(x,π)=YES.*
- *If x is a NO instance,then $\forall$ proofs π V(x,π)=NO*

Now let us look at the randomized concept of a verifier that only examines some number of bits of the proof and accepts/rejects based on computation on only some randomly chosen bits. Formalizing this below -

Local Verifier: Verifier has randomness $r(n)$ and query complexity $q(n)$ if it chooses $q(n)$ locations in π using $r(n)$ coin tosses, then queries π on those $q(n)$ locations and returns YES or NO.

Definition 14.9. ***Probabilistically Checkable Proofs (PCP)*** *:*
Problem A is in $PCP_{c,s}$ [r(n),q(n)] if $\exists$ polynomial time verifier of randomness r(n) and query complexity q(n) such that

- *If x is a YES instance, there is a proof π such that* $\Pr[V(x,\pi) = YES] \geq C$
- *If x is a NO instance, then $\forall$ proofs π,* $\Pr[V(x,\pi) = YES] < S$

The parameter C and S are called the *completeness* and *soundness* of the verifier. It is possible to capture the class NP with a verifier that looks at just some small constant number of bits of the proof while using only a logarithmic amount of randomness. We see a formal result of this below.

Theorem 14.10. ***PCP Theorem*** *- NP $\subseteq$ $PCP_{1,\frac{1}{2}}$ $[O(logn), O(1)]$*

The PCP theorem basically says that for every NP language L, there is an efficient randomized verifier that queries O(1) proof symbols and :

- x $\in$ L - There exists a proof that is always accepted.
- c $\notin$ L - For any proof , the probability to accept is $\leq \frac{1}{2}$

We see few results based on PCPs.

Claim 14.11. *Suppose [a,1]-gap 3-SAT is NP-hard then NP $\subseteq$ $PCP_{1,a}$[O(log n),3]*

Proof. Let A be a problem in NP and there is a polynomial reduction to [a,1]-gap 3-SAT. Let x be any input. First we reduce x to an instance ϕ of 3-SAT that is satisfiable iff x $\in$ L. As a membership proof for x, the verifier expects a satisfying assignment for ϕ. Given such a membership proof , it does a probabilistic check on it by randomly choosing a clause in ϕ and querying the the values of the three variables involved in that clause. It accepts if and only if the clause is satisified. Clearly if x$\in$ L , then a satisying assignment for ϕ will cause the verifier to accept with probability 1. On the other hand, if x $\notin$ L, then every assignment will satisfy less than fraction a of the clauses , so the verifier accepts with probability less than a. Verifier uses O(log n) random bits to make three queries. Therefore by the above notation as used in PCP theorem NP $\subseteq$ $PCP_{1,a}$[O(log n),3]. □

Lemma 14.12. *Suppose NP $\subseteq$ $PCP_{1,a}$[O(log n),O(1)] then [a',1] gap 3-SAT is NP-hard for some a' < 1.*

We will see a formal proof of the above and further details regarding PCP in the next lecture.

References

[2010] DAVID P. WILLIAMSON and DAVID B.SHMOYS, "The Design of Approximation Algorithms" *Cambridge University Press*