Hardness of Approximation

We have seen several methods to find approximation algorithms for NP-hard problems

We have also seen a couple of examples where we could show lower bounds on the achievable approxmation ratio under the assumption that $P \neq NP$

We now discuss more formally the notion of *gap* reductions which will make it easy to build on existing results to prove hardness of approximation for new problems or improve existing results

Let us first review the definition of NPO problems and approximation ratio

Formal definition of NPO problems

P, NP: language/decision classes

NPO: NP Optimization problems, function class

Optimization problem

Π is an optimization problem

- Π is either a *min* or *max* type problem
- Instances I of Π are a subset of Σ^*
- $|I|$ size of instance
- for each I there are feasible solutions $\mathcal{S}(I)$
- for each I and solution $S \in \mathcal{S}(I)$ there is a real/rational number $val(S, I)$
- Goal: given I, find $OPT(I) = \min_{S \in \mathcal{S}(I)} val(S, I)$

NPO: NP Optimization problem

Π is an NPO problem if

- Given $x \in \Sigma^*$, can check if x is an instance of Π in poly($|x|$) time
- for each I, and $S \in \mathcal{S}(I)$, $|S|$ is poly($|I|$)
- there exists a poly-time decision procedure that for each I and $x \in \Sigma^*$, decides if $x \in \mathcal{S}(I)$
- val(I, S) is a poly-time computable function

NPO and NP

Π in NPO, minimization problem

For a rational number B
define $L(\Pi, B) = \{ I \mid OPT(I) \leq B \}$

Claim: $L(\Pi, B)$ is in NP

$L(\Pi, B)$: decision version of Π

Approximation Algorithm/Ratio

Minimization problem Π: $\mathcal{A}$ is an approximation algorithm with *(relative)* approximation ratio α iff

- $\mathcal{A}$ is polynomial time algorithm
- for all instance I of Π, $\mathcal{A}$ produces a feasible solution $\mathcal{A}(I)$ such that

  val $(\mathcal{A}(I)) \leq \alpha$ val (OPT(I))

(Note: $\alpha \geq 1$)

Remark: α can depend in size of I, hence technically it is $\alpha(|I|)$.

Example: $\alpha(|I|) = \log n$

Maximization problems

Maximization problem Π: $\mathcal{A}$ is an approximation algorithm with *(relative)* approximation ratio α iff

- $\mathcal{A}$ is polynomial time algorithm
- for all instance I of Π, $\mathcal{A}$ produces a feasible solution $\mathcal{A}(I)$ such that

  val $(\mathcal{A}(I)) \geq \alpha$ val (OPT(I))

(Note: $\alpha \leq 1$)

Very often people use $1/\alpha$ (≥ 1) as approximation ratio

Proving hardness of approximation

Proving hardness of approximation is essentially the following:

Let Π be a minimization problem

Suppose we want to prove that Π is $\alpha(|I|)$ hard to approximation where $|I|$ is the instance size

We need to show that if there is a polynomial time algorithm for Π with an approximation ratio $\alpha(|I|)$ then we can use it to solve an NP-Hard *decision problem* (any NP-Hard problem would do)

This implies that unless P=NP no $\alpha(|I|)$ approximation ratio for Π

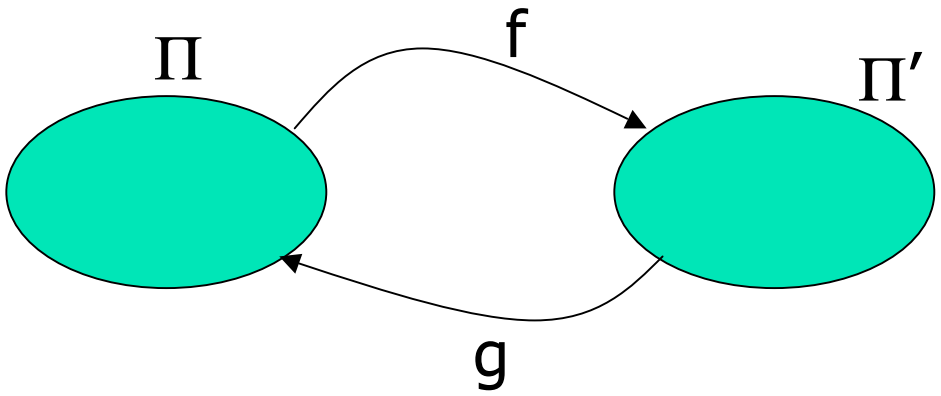
Reductions

Once we prove a particular problem Π is hard to approximate to within an α factor, we wish to use this to prove that another problem Π' is hard to approximate to within a β factor

To make it easy to compose reductions we need to define a proper notion of reduction. This is somewhat more involved than reductions to prove NP-Completeness for decision problems since we have function problems with solutions, quality of solutions etc.

We define two types of reductions

Approximation Preserving Reductions



Given an instance I of Π, $I' = f(I)$ is an instance of Π'

Given a solution s to I', $g(I, I', s)$ is a solution to I

Both f and g are poly-time computable functions

Approx preserving reductions

Other properties of f, g

We assume that both Π, Π' are minimization problems, the definitions change for min-max, max-max, etc

1. $OPT(I') \leq OPT(I)$
2. If $s \in \mathcal{S}(I')$ then $t = g(I,I',s)$ is a solution to I and $Val(t, I) \leq Val(s, I')$

(recall that $\mathcal{S}(I)$ is the set of solutions to instance I and that $Val(t, I)$ is the objective function value for soln t)

If f, g satisfy above properties then (f,g) is an approximation preserving reduction from Π to Π'

Approx preserving reductions

Using this notion of reduction allows us to claim a couple of simple but useful features

Lemma: If (f,g) is an approximation preserving reduction from Π to Π' and (f',g') is an approximation preserving reduction from Π' to Π'' then (f'',g'') is an approximation preserving reduction from Π to Π'' where $f'' = f' \circ f$ and $g'' = g \circ g'$

Approx preserving reductions

Lemma: If (f,g) is an approximation preserving reduction for Π to Π' then an α approximation to Π' where α is a constant implies an α approximation to Π

The converse of the above lemma is:

If (f,g) is an approximation reduction from Π to Π' and if Π does is NP-hard to approximate to a factor of β then Π' is NP-hard to approximate to within a factor of β

Both lemmas are straight forward exercises from the definitions

An example

We give a reduction from the Set Cover problem to the Node-Weighted Steiner tree problem

Set cover:

Given universe $\mathcal{U}$ of n elements and

sets $S_1, S_2, \ldots, S_m$ where each S_i is a subset of $\mathcal{U}$

Solution: $A \subseteq \{1,2, \ldots, m\}$ s.t $\cup_{i \in A} S_i = \mathcal{U}$

Objective function: $Val(A) = |A|$

Goal: minimization

An example

Node-weighted Steiner tree problem:

Given graph G=(V,E) and node weighs w: $V \rightarrow \mathcal{R}^+$

$T \subseteq V$, terminals

Solution: a (connected) subgraph $H=(V_H, E_H)$ of G s.t
$T \subseteq V_H$

Objective function: $w(V_H)$

Goal: minimization

Reduction

Given a set cover instance we obtain a node-weighted Steiner tree instance as follows:

n elements in $\mathcal{U}$, m sets $S_1,\ldots,S_m$

There are n+m+1 nodes in G one for each element, one for each set and an extra one for a root

The edges of G are as follows:

- root r is connected to each node corresponding to a set
- a node corresponding to a set is connected to each node corresponding to an element that is contained in the set

Reduction

Weights are defined as follows

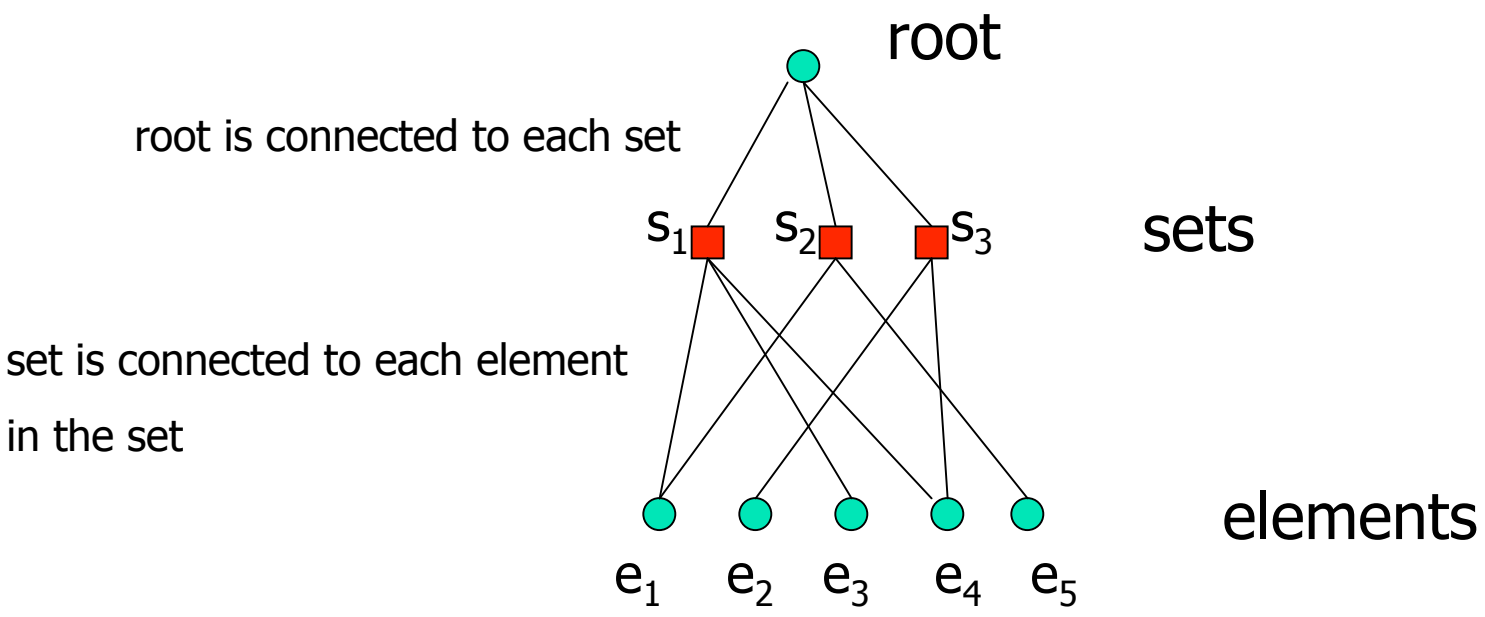
w(v) = 1 if v corresponds to set vertex, 0 otherwise

Terminals T is root and all the vertices corresponding to the elements of $\mathcal{U}$

Example

$\mathcal{U}$ = {1,2,3,4,5}, S_1={1,3,4}, S_2={1,5}, S_3={2,4}

The graph constructed is given below



Reduction

It is easy to see that given a set cover instance the reduction to the node-weighted Steiner tree problem is poly-time computable: this is the function f

Now we need to give the function g which maps solutions to the node-weighted Steiner tree instance back to solutions for the set cover instance

This is easy: given a subgraph $H=(V_H, E_H)$ of G in which all the terminals are connected,

$g(H) = \{i \mid s_i \in V_H\}$, that is all sets whose corresponding nodes are in V_H

Easy to see that g is polynomial time computable

Reduction

Let I be the Set Cover instance and $I' = f(I)$ be the node-weighted Steiner tree instance.

We need to check that $OPT(I') \leq OPT(I)$ and if H is a solution to I' then $|g(H)| \leq w(V_H)$

Both are easy to check - exercise to the reader

Reduction

It is known that Set Cover is hard to approximate within a factor of $c \log n$ unless P=NP for some constant c

Thus we would like to conclude that node-weighted Steiner tree is also $c \log n$ hard to approximate

Unfortunately we cannot do this in a straight forward way using the current machinery we set up

What we can conclude is the following:

Since Set Cover is hard to approximate to within any constant α, node-weighted Steiner tree problem is also hard to approximate to within any constant α

Reduction

Why aren't we able to conclude that node-weighted Steiner tree is hard to approximate to within c log n?

The main reason is because n related to the the size of the Set Cover instance

The reduction f ensures that f(|I|) is polynomial size but but we don't the precise polynomial

Thus to conclude more from the approximation preserving reductions we also need to address f and g

Further, pure approximation preserving reductions are not as frequent as relaxed versions and hence we prefer to use *gap reductions* that we define next

Gap Reductions

Gap reductions come in two flavors

The first is a gap reduction from an NP-Hard decision problem/language L to an NPO problem Π that establishes harness of approximation for Π

The second is a gap reduction from an NPO problem Π to another NPO problem Π' that establishes hardness for Π' from hardness for Π

Gap Reduction 1

Let L be an NP-hard/complete decision problem/language and Π an NPO problem. Assume Π is a min problem

Then a (f, c_1, c_2) gap reduction from L to Π is the following:

1. $f : \Sigma^* \rightarrow \Sigma^*$ maps strings to instances of Π
2. c_1, c_2 are functions from Z^+ to Q^+
3. f, c_1, c_2 are poly-time computable functions
4. If $x \in L$ then $OPT(f(x)) \leq c_1(|f(x)|)$
5. If $x \notin L$ then $OPT(f(x)) > c_2(|f(x)|)$

Gap Reduction 1

What is the use of a (f, c_1, c_2) reduction?

We can say the following: If there is such a reduction from L to Π then unless P=NP there is no $c_2(n)/c_1(n)$ approximation for Π where n is the size of an input for Π

Proof: exercise

Examples

We have seen such gap reductions before:

For k-center problem we gave a reduction from the domination set problem to the k-center problem where $c_1(n) = 1$ and $c_2(n) = 2-\varepsilon$ for any fixed $\varepsilon > 0$

For bin-packing problem we gave a reduction from the 2-partition problem where $c_1(n) = 2$ and $c_2(n) = 3-\varepsilon$ for any fixed $\varepsilon > 0$

We will later state the PCP theorem as giving a gap reduction from SAT to Max-3SAT

Gap Reduction 2

A gap reduction from an NPO problem Π to another NPO problem Π' is defined similarly. Assume both are min problems

$(f, c_1, c_2, c_1', c_2')$ is a gap reduction from Π to Π' if

1. $f:\Sigma^* \rightarrow \Sigma^*$ maps instances of Π to instances of Π'
2. c_1, c_2, c_1', c_2' functions from Z^+ to Q^+
3. f, c_1, c_2, c_1', c_2' are poly-time computable functions
4. If I is an instance of Π and if $OPT(I) \leq c_1(|I|)$ then $OPT(f(I)) \leq c_1'(|f(I)|)$
5. If $OPT(I) > c_2(|I|)$ then $OPT(f(I)) > c_2'(|f(I)|)$

Gap Reduction 2

We have the following easy composition:

If (f, c_1, c_2) is a gap reduction from a language L to a problem Π and $(g, c_1, c_2, c_1', c_2')$ is a gap reduction from Π to Π' then $(g \circ f, c_1', c_2')$ is a gap reduction from L to Π'

Therefore it follows that if L is NP-hard then Π' is hard to approximate to $c_2'(n)/c_1'(n)$

We will see an example of the use of compositions

More on reductions

Gap reductions are a convenient way to compose reductions to obtain hardness. However note that gap reductions, unlike approximation preserving reductions do not yield an algorithm for Π from an algorithm for Π' - there is no mapping from solutions to $f(I)$ to solutions to I

The definition of the reduction does not say anything about instances in the "middle". What if $c_1(|I|) < OPT(I) \leq c_2(|I|)$? The reduction does not care about such instances

Gap reductions are thus mainly tools for proving hardness of approximation and not algorithmic reductions

Assumption for hardness

Typically we wish to show that unless P=NP some NPO problem Π is hard to approximate to within a factor of $\alpha(n)$.

However sometimes we can only show hardness under a stronger assumption such as RP $\neq$ NP where RP is randomized polynomial time

This happens because the mapping function f that maps instances to instances might not be a polynomial time algorithm but a randomized polynomial time algorithm.

Similarly, sometimes the function f is not polynomial time but quasi-polynomial

Assumption for hardness

Similarly, sometimes the function f is not polynomial time but quasi-polynomial - often the main reason for this is that the new instance size is not polynomial in the original instance size

To be concrete, say (f, c_1, c_2) is a reduction from a NP-complete language L to Π but f maps instances of size n to instances of size $n^{c \log n}$

Then we can conclude the following: if Π has an approximation ratio of $c_2(|f(I)|)/c_1(|f(I)|)$ then there is a quasi-polynomial time algorithm to decide L

Techniques for proving hardness

Although proving hardness of approximation results requires considerable knowledge of the problem domain, there are a few broad methods used

1. Use existing hardness results and gap reductions to obtain hardness results for new problems. The gap reductions can be very sophisticated. Sometimes they are akin to complicated gadget reductions for NP-Completeness proofs but also need to pay attention to the gap properties
2. Amplification: some problems exhibit self-reducibility even in the approximation sense. Examples are clique, chromatic number, some lattice problems etc. This allows one to use a given hardness factor to something larger
3. PCPs (probabilistically checkable proofs) and other sophisticated technology like the parallel repetition theorem which establish gap reduction for some basic constrain satisfaction problems
4. To obtain gadgets and constructions, it is often useful to understand integrality gaps for natural linear programs.

Hardness of Max-3SAT

Recall that Max-3SAT problem:

Given m 3-CNF clauses on n boolean variables, find an assignment to maximize the number of satisfied clauses

Max-3SAT(k) is the restriction of Max-3SAT in which each variable occurs in at most k clauses. In this case it follows that $m \leq nk$ and if each clause is exactly length 3 (the uniform case) then $m \leq nk/3$

Recall that we discussed a 3/4 approximation for Max-SAT and hence also for Max-3SAT. Question is whether Max-SAT and in particular Max-3SAT is hard to approximate to within a constant factor or if there is a PTAS for it

Hardness of Max-3SAT

A fundamental and quite difficult result which is equivalent to the PCP theorem and in fact derived from it is the following:

Theorem: There is a gap reduction f from 3SAT to Max-3SAT with the following properties:

there exists an absolute constant ε_0 such that

if ϕ is satisfiable then $f(\phi)$ is also satisfiable

if ϕ is not satisfiable then less than $(1-\varepsilon_0)$ fraction of clauses in $f(\phi)$ are satisfiable by any assignment to $f(\phi)$

From above, one can use another gap reduction using expanders to prove that Max-3SAT(5) is also hard to approximate to within some $(1-\varepsilon_0')$ factor for some other fixed constant ε_0'

Reductions from Max-3SAT

We will now use a reduction from Max-3SAT to prove hardness for the maximum independent set and vertex cover problems

Given a graph $G=(V,E)$

- $\alpha(G)$: size of the maximum independent set in G
- $\omega(G)$: size of the maximum clique in G
- $\beta(G)$: size of the minimum vertex cover in G

Simple known facts:

$\alpha(G) + \beta(G) = |V|$

$\alpha(G) = w(G^c)$ where G^c is the complement of G

Reduction for $\alpha(G)$

Given a 3-SAT formula ϕ we assume wlog that it has exactly 3 literals in each clause - otherwise we can duplicate a literal

Let ϕ has clauses $C_1, C_2, \ldots, C_m$ on n variables $x_1, x_2, \ldots, x_n$

Given ϕ we obtain a graph G_ϕ as follows:

G_ϕ has 3 nodes per clause, for a total of $3m$ nodes

For clause C_i, let the nodes be v^i_1, v^i_1, v^i_3

We label the nodes of C_i with the literals in C_i

(see example next slide)

Reduction

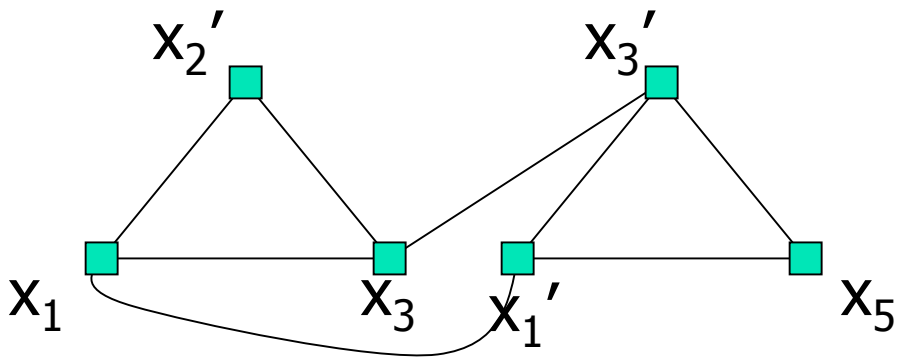
The edges in G_ϕ are as follows:

The three nodes corresponding to a clause C_i form a triangle (clique)

We also add an edge between v^i_k and v^j_l if they are labeled by two different literals of the same boolean variable in ϕ: example x_1 and x_1'

Example: $\phi = (x_1 \vee x_2' \vee x_3) \wedge (x_1' \vee x_3' \vee x_5)$

G_ϕ



Reduction

Let $OPT(\phi)$ be the maximum number of clauses satisfiable in ϕ. Then

Claim: $OPT(\phi) = \alpha(G_\phi)$

It is left as a relatively easy exercise

Note G_ϕ can be obtained from ϕ in polynomial time

The implication of the above is the following:

Since it is NP-hard to decide if $OPT(\phi) = m$ or $OPT(\phi) < (1-\varepsilon_0)\, m$ it is also NP-hard to decide if $\alpha(G_\phi) = m$ or $\alpha(G_\phi) < (1-\varepsilon_0)m$

Hence max independent set is hard to approximate to within $(1-\varepsilon_0)$

Reduction for $\beta(G)$

We can use the same reduction to also prove that $\beta(G)$ is hard to approximate to within a constant factor.

Note that $\beta(G_\phi) = 3m - \alpha(G_\phi)$

Therefore if

$OPT(\phi) = m$ then $\beta(G_\phi) = 3m - m = 2m$

If $OPT(\phi) < (1-\varepsilon_0)m$ then

$\beta(G_\phi) = 3m - \alpha(G_\phi)$

$= 3m - OPT(\phi) > 3m - (1-\varepsilon_0)m > (2+\varepsilon_0)m$

Thus it is NP-hard to approximate $\beta(G)$ to within

$(2+\varepsilon_0)/2 = (1+\varepsilon_0/2)$

Amplification for hardness of clique

Note that $\alpha(G) = \omega(G^c)$, hence approximating maximum independent set and maximum clique is equivalent (there is an approximation preserving reduction in both directions)

We have seen that $\alpha(G)$ is hard to approximate to within a $(1-\varepsilon_0)$ factor for some fixed $\varepsilon_0 > 0$. Note that this also implies the same hardness for $\omega(G)$

We now see how to *boost/amplify* this hardness using a simple graph product operation. For this purpose it is more convenient to work with $\omega(G)$ instead of $\alpha(G)$

Amplification for clique

Given two graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ we define $G = G_1 \times G_2$ as follows:

Let $E_1' = E_1 \cup \{ (u,u) \mid u \in V_1 \}$

$E_2' = E_2 \cup \{(a,a) \mid a \in V_2 \}$

$G = (V, E)$ where

$V = V_1 \times V_2$

$E = \{((u,a),(v,b)) \mid (u,v) \in E'_1 \text{ and } (a,b) \in E'_2 \}$

Note that $G_1 \times G_2$ is not necessarily same as $G_2 \times G_1$

Amplification for clique

Lemma: $\omega(G) = \omega(G_1)\ \omega(G_2)$

Relatively simple exercise to prove

For an integer k we define G^k as $G^{k-1} \times G$ where $G^1 = G$

It follows that $\omega(G^k) = (\omega(G))^k$

We can compose this with the hardness of $(1-\varepsilon_0)$ shown via reduction from Max-3SAT to get a hardness of c for any constant $c > 0$

Amplification for clique

We say that it was NP-hard to distinguish whether $\alpha(G_\phi) = m$ or $\alpha(G_\phi) < (1-\varepsilon_0)\, m$

Hence it is NP-hard to distinguish whether $\omega((G^c_\phi)^k) = m^k$ or $\omega((G^c_\phi)^k) < (1-\varepsilon_0)^k\, m^k$

Hence clique is hard to approximate to within $(1-\varepsilon_0)^k$

How large can be choose k?

The reduction from G to G^k needs to be polynomial time. Hence k has to be a constant (Why?).

Thus by choosing k large enough we obtain a hardness of c for clique for any $c > 0$