Approximation Algorithms and Hardness of Approximation

IPM, Jan 2006

Mohammad R. Salavatipour
Department of Computing Science
University of Alberta

Introduction

- For NP-hard optimization problems, we want to:
  1. find the optimal solution,
  2. find the solution fast, often in polynomial time
  3. find solutions for any instance
- Assuming P $\neq$ NP, we cannot have all the above simultaneously.
- If we
  1. relax (3), then we are into study of special cases.
  2. relax (2), we will be in the field of integer programming (branch-and-bound, etc).
  3. relax (1), we are into study of heuristics and approximation algorithms.
- We are going to focus on approximation algorithms:
  - finding solutions that are within a guaranteed factor of the optimal solutions, and
  - We want to find this solution fast (i.e. polynomial time).

- An NP optimization problem, Π, is a minimization (maximization) problem which consists of the following items:

  Valid instances: each valid instance I is recognizable in polytime; D_Π is set of all valid instances.

  Feasible solutions: Each $I \in D_\Pi$ has a set $S_\Pi(I)$ of feasible solutions and for each solution $s \in S_\Pi(I)$, $|s|$ is polynomial (in $|I|$).

  Objective function: A polytime computable function $f(s, I)$ that assigns a ≥ 0 rational value to each feasible solution.

  Typically, want a solution whose value is minimized (maximized): it is called the optimal solution.

- Example: Minimum Spanning Tree (MST):

  A connected graph $G(V, E)$, each $(u, v) \in E$ has a weight $w(u, v)$,
  Goal: find an acyclic connected subset $T \subseteq E$ whose total weight is minimized.

  valid instances : a graph with weighted edges.

  feasible solutions : all the spanning trees of the given weighted graph.

  objective functions : minimizing the total weight of a spanning tree.

- Approximation algorithms: An α-*approximation algorithm* is a polytime algorithm whose solution is always within factor α of optimal solution.

- For a minimization problem Π, algorithm A has factor $\alpha \geq 1$ if for its solution s: $f(s, I) \leq \alpha(|I|) \cdot OPT(I)$.
  α can be a constant or a function of the size of the instance.

- Example: **Vertex-Cover Problem**
  - Input: undirected graph $G = (V, E)$ and a cost function $c : V \rightarrow Q^+$
  - Goal: find a minimum cost vertex cover, i.e. a set $V' \subseteq V$ s.t. every edge has at least one endpoint in V'.

- The special case, in which all vertices are of unit cost: *cardinality vertex cover problem*.

- V.C. is NP-hard.

- Let's look for an approximation

- Perhaps the most natural greedy algorithm for this problem is:

  Algorithm VC1:
  $S \leftarrow \emptyset$
  `While` $E \neq \emptyset$ `do`
      let v be a vertex of maximum degree in G
      $S \leftarrow S \cup \{v\}$
      remove v and all its edges from G
  return S

- Easy to see that it finds a VC. What about the approximation ratio?

- It can be shown that the approximation ratio is $O(\log \Delta)$, where Δ is the maximum degree in G.

- The second approximation algorithm $VC2$ appears to be counter-intuitive at first glance:

  Algorithm VC2:
  $S \leftarrow \emptyset$
  `While` $E \neq \emptyset$ `do`
      let $(u, v) \in E$ be any edge
      $S \leftarrow S \cup \{u, v\}$
      delete u,v and all their edges from G
  return S

- **Lemma 1** *VC2 returns a vertex cover of G.*

  Proof: $VC2$ loops until every edge in G has been covered by some vertex in S. ∎

- **Lemma 2** *VC2 is a 2-approximation algorithm.*

  Proof:

  - Let A denote the set of edges selected by $VC2$.
  - Note that A forms a matching, i.e. no two edges selected share an end-point
  - Therefore, in order to cover the edges of A, any vertex cover must include at least one end-point of each edge in A.
  - So does any optimal V.C. S^*, i.e. $|S^*| \geq |A|$.
  - Since $|S| = 2|A|$, we have $|S| \leq 2|S^*|$.

  ∎

- **Lemma 3** *The analysis of $VC2$ is tight.*

  A complete bipartite graph $K_{n,n}$. $VC2$ picks all the $2n$ vertices, but optimal picks one part only.

- **Major Open question:** Is there a $2 - O(1)$-approx algorithm for V.C.?

- **Theorem 4 (Hastad 97)** *Unless* P = NP, *there is no approximation algorithm with ratio* $< \frac{7}{6}$ *for Vertex-Cover problem.*

Set Cover

- Set-cover is perhaps the single most important (and very well-studied) problem in the field of approximation algorithms.

  Set-Cover Problem

  - Input
    - U : a universe of n elements $e_1, \ldots, e_n$,
    - $S = \{S_1, S_2, \cdots, S_k\}$ a collection of subsets of U,
    - $c : S \to Q^+$ a cost function
  - Goal: find a min cost subcollection of S that covers all the elements of U, i.e. $I \subseteq \{1, 2, \cdots, k\}$ with min $\sum_{i \in I} c(S_i)$ such that $\bigcup_{i \in I} S_i = U$

- V.C. problem is a special case of Set-Cover: for a graph $G(V, E)$, let $U = E$, and
  $S_i = \{e \in E | e$ is incident within $v_i\}$

- We will give an $(\ln n)$-approx greedy algorithm for Set Cover.

- Rather than greedily picking the set which covers maximum number of elements, we have to take the cost into account at the same time.

- We pick the most cost-effective set and remove the covered elements until all elements are covered.

  Definition 5 *The cost-effectiveness of a set S_i is the average cost at which it covers new element, i.e., $\alpha = \frac{c(S_i)}{S_i - C}$, where C is the set of elements already covered.*
  We Define the price of an element to be the cost at which it is covered.

- Greedy Set-Cover algorithm

  $C \leftarrow \emptyset, T \leftarrow \emptyset$
  `While` $C \neq U$ `do`
  choose a S_i with the smallest α
  add S_i to T
  for each element $e \in S_i - C$, set $price(e) = \alpha$
  $C \leftarrow C \cup \{S_i\}$
  return T

- when a set S_i is picked, its cost is distributed equally among the new elements covered

- **Theorem 6** *The Greedy Set-Cover algorithm is an H_n factor approximation algorithm for the minimum set cover problem, where $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$. Note that $H_n \approx \ln n$.*

- **Proof:**
  - Let $e_1, e_2, \cdots, e_n$ be the order at which the elements are covered
  - It can be seen that:

$$\sum_{S_i \in T} c(S_i) = \sum_{k=1}^{n} price(e_k)$$

  - We try to estimate $price(e_k)$ (at this stage, we have covered $e_1, e_2, \cdots, e_{k-1}$, and have $n - k + 1$ uncovered elements).
  - Let T_{OPT} be an opt solution and C_{OPT} be its cost.
  - At any point of time, we can cover the elements in $U - C$ at a cost of at most C_{OPT}.
  - Thus among the sets not selected, $\exists$ a set with cost-effectiveness $\leq \frac{C_{OPT}}{|U-C|}$
  - In the iteration in which element e_k was covered, $|U - C| = n - k + 1$.

■

- Since e_k was covered by the most cost-effective set in this iteration:

$$price(e_k) \leq \frac{C_{OPT}}{n-k+1}$$

- As the cost of each set picked is distributed among the new elements covered, the total cost of the set cover picked is:

$$\begin{aligned} \sum_{S_i \in T} c(S_i) &= \sum_{k=1}^{n} price(e_k) \\ &\leq C_{OPT} \sum_{k=1}^{n} \frac{1}{n-k+1} \\ &= H_n \cdot C_{OPT} \end{aligned}$$

- The analysis of this greedy algorithm is tight:
  Example: each $e_1, e_2, \cdots, e_n$ by itself is a set, with cost $\frac{1}{n}, \frac{1}{n-1}, \cdots, 1$ respectively;
  one set contains all of U with cost $1+\epsilon$ for some $\epsilon > 0$.

  - the greedy solution picks n singleton sets; costs $\frac{1}{n} + \frac{1}{n-1} + \cdots + 1 = H_n$.
  - The optimal cover has cost $1+\epsilon$.

- **Theorem 7** *Based on the results of Lund and Yannakakis 92, Feige 86, Raz 98, Safra 97, Suduan 97:*
  - *There is a constant* $0 < c < 1$ *such that if there is a* $(c \ln n)$*-approximation algorithm for Set-Cover problem, then* P = NP*.*
  - *For any constant* $\epsilon > 0$*, if there is a* $(1 - \epsilon) \ln n$*-approximation algorithm for Set-Cover then* NP $\subseteq$ $DTIME(n^{O(\ln \ln n)})$*.*

- Next we give another algorithm for Set Cover using Linear Programming (LP).

- First, forumlate Set Cover as an IP.
  We have one indicator variable x_s for every set S:

$$\begin{array}{ll} \text{minimize} & \sum_{s \in S} c_s x_s \\ \text{subject to} & \forall e \in U : \quad \sum_{s: e \in S} x_s \geq 1 \\ & x_s \in \{0, 1\} \end{array}$$

- Now relax the integrality constraint to obtain the corresponding LP.
  Although IP is NP-hard, there are polytime algorithms for solving LP.

- Solution to an LP-relaxation is usually called the *fractional solution* and is denoted by OPT_f.

- We give another $O(\log n)$-approximation for Set Cover using a powerful technique called randomized rounding.

- The general idea is to start with the optimal fractional solution (solution to the LP) and then round the fractional values to 1 with some appropriate probabilities.

- Randomized Rounding Alg for Set cover:
  - Take the LP relaxation and solve it.
  - For each set S, pick S with probability $P_s = x_s^*$ (i.e. round x_s^* up to 1 with probability x_s^*), let's call the integer value $\widehat{x}_s$),

  Consider the collection $C = \{S_j \mid \widehat{x}_{S_j} = 1\}$:

-
$$\text{E}[cost(C)] = \sum_{S_j \in \mathcal{S}} \text{Pr}[S_j \text{ is picked}] \cdot c_{S_j} = \sum x^*_{S_j} \cdot c_{S_j} = OPT_f \tag{1}$$

- Let α be large enough s.t. $(\frac{1}{e})^{\alpha \log n} \leq \frac{1}{4n}$.

- Repeat the algorithm above $\alpha \log n$ times and let $C' = \bigcup_{i=1}^{\alpha \log n} C_i$ be the final solution, where C_i is the collection obtained after round i of the algorithm.

- Suppose e_j belongs to $S_1, ...S_q$.

- By the constraint for e_j, in any fractional feasible solution:

$$x_{S_1} + x_{S_2} + ... + x_{S_q} \geq 1$$

- It can be shown that the probability that e_j is covered is minimized when

$$\begin{aligned} & x_{S_1} = x_{S_2} = ... = x_{S_q} = \tfrac{1}{q} \\ \Rightarrow \quad & \Pr[e_j \text{ is not covered in } C_i] \leq (1 - \tfrac{1}{q})^q < \tfrac{1}{e} \\ \Rightarrow \quad & \Pr[e_j \notin C'] \leq (\tfrac{1}{e})^{\alpha \log n} \leq \tfrac{1}{4n} \end{aligned}$$

- Sum over all e_j:

$$\Pr[\exists e_j, e_j \notin C', \text{ (i.e. C}' \text{ is not a set cover)}] \leq \text{n} \cdot \frac{1}{4\text{n}} \leq \frac{1}{4}$$

- Let's call the event "C' is not a Set Cover", $E1$. By above:

$$\Pr[E_1] \leq \frac{1}{4}. \qquad (2)$$

- On the other hand, by (1) and by summing over all rounds:

$$\text{E}[cost(C')] \leq \alpha \log n \cdot OPT_f$$

- Markov's inequality says for any random variable X: $\Pr[X \geq t] \leq \frac{\mathsf{E}[X]}{t}$.

- Define the bad event E_2 to be the event that $cost(C') > 4\alpha \log n \cdot OPT$. Thus:

$$\begin{aligned} \Pr[E_2] &= \Pr[cost(C') > 4\alpha \log n \cdot OPT] \\ &\leq \frac{\alpha \log n \cdot OPT_f}{4\alpha \log n \cdot OPT_f} \\ &\leq \frac{1}{4} \end{aligned}$$

- The probability that either C' is not a set cover (i.e. E_1 happens) or that C' is a set cover with large cost (i.e. E_2 happens) is at most: $\Pr[E_1] + \Pr[E_2] \leq \frac{1}{2}$.

- Therefore, with probability $\geq \frac{1}{2}$, C' is a set cover with $cost(C') \leq 4\alpha \log n \cdot OPT_f \leq 4\alpha \log n \cdot OPT$.

- Repeating this algorithm t times, the probability of failure at all rounds is at most $\frac{1}{2^t}$.

- So, the probability of success for at least one run of the algorithm is $1 - \frac{1}{2^t}$.

Section 2:

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 $\neq$ 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 $\neq$ 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.

- As SAT is the canonical problem for NP-hardness, Max-SAT is the canonical problem for hardness of approximation:

  Max-SAT:

  - Input: A boolean formula Φ over variables $x_1, \ldots, x_n$ in CNF which has clauses $C_1, \ldots, C_M$.
  - Question: Find a truth assignment to maximizes the number of satisfied clauses.

- The PCP theorem implies that Max-SAT is NP-hard to approximate within a factor of $(1 - \epsilon_0)$, for some fixed $\epsilon_0 > 0$.

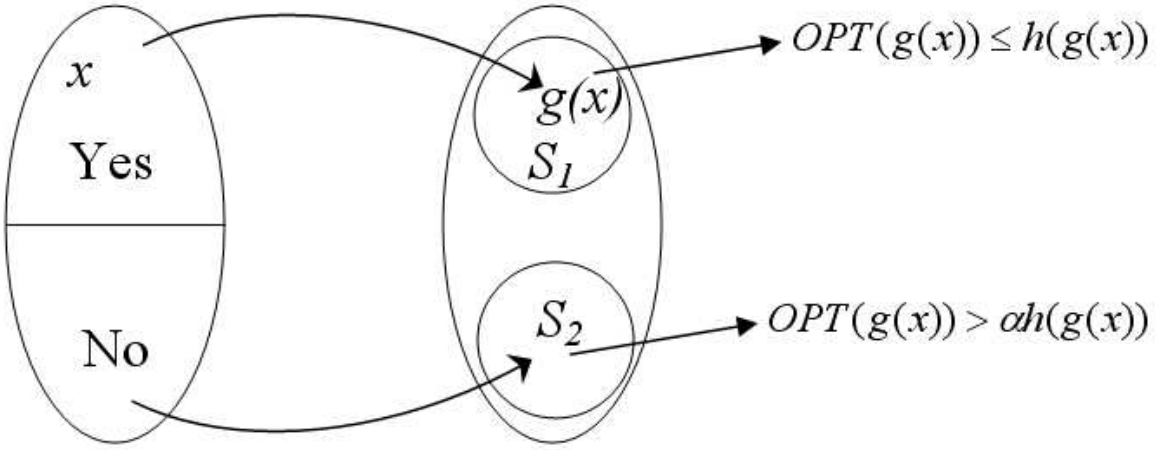
- For proving a hardness of approximation, for example for vertex cover, we prove a reduction like the following:

  Given a formula φ for Max-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 8** *The vertex cover problem cannot 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.



- Suppose L is NP-complete and π is a minimization problem.
- Let g be 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.
- 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.
- This implies π is hard to approximate within factor α.

- Many problems, such as Bin Packing and Knapsack, have PTAS's, i.e. a $(1+\epsilon)$-approximation for any constant $\epsilon > 0$.

- A major open question was: does Max-3SAT have a PTAS?

- A significant result on this line was by Papadimitriou and M. Yannakakis ('92); 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.

- The celeberated PCP theorem states that there is no PTAS for Max-SNP problems.

- It also give another characterization of NP.

- **Definition 9** *A language $L \in$ 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 $x \in L \Rightarrow \exists\ y$ s.t. $V(x, y) = 1$.*
  - *Soundness: if $x \notin L \Rightarrow \forall y$, $V(x, y) = 0$.*

- **Definition 10** *An $(r(n), b(n))$-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.*

- **Definition 11** *For $0 \leq s < c \leq 1$, a language $L \in$ 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$[V(x, \pi) = 1] \leq S$.*

- The probabilities in completeness and soundness given in definition above are typically $C = 1$ and $S = \frac{1}{2}$, respectively.

- From the definition, for any $0 \leq s < c \leq 1$:

  $\mathsf{NP} \subseteq \mathsf{PCP}_{c,s}(0, \mathsf{poly}(n)) \subseteq \mathsf{PCP}_{c,s}(O(\log n), \mathsf{poly}(n))$.

- **Lemma 12** $\mathsf{PCP}_{c,s}(O(\log n), \mathsf{poly}(n)) \subseteq \mathsf{NP}$

- Proof: Let L be a language in $\mathsf{PCP}_{c,s}(O(\log n), \mathsf{poly}(n))$ 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 $2^{O(\log n)} = \mathsf{poly}(n)$ possible random strings.

- M accepts if at least a fraction c of all these runs accept, rejects otherwise.

- Thus:
  - if $x \in L \Rightarrow V(x, \pi)$ accepts with probability at least c; thus at least a fraction c of random strings cause the verifier V and therefore M to accept.
  - if $x \notin L$ then the verifier accepts with probability at most s which is smaller than c; thus for only a fraction of $< c$ of random strings verifier V accepts; M rejects.

- Since there are $O(\text{poly}(n))$ random strings of length $O(\log n)$ and each simulation takes polytime, the running time of M is polytime.

- This lemma and the observation before it implies that

$$\text{PCP}_{c,s}(O(\log n), \text{poly}(n)) = \text{NP}.$$

- The remarkable PCP theorem, proved by Arora/Safra [92] and Arora/Lund/Motwani/Sudan/Szegedy[92] states:

  Theorem 13 (PCP Theorem)

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

- The original proof of PCP theorem was extremely difficult. There is a new a much simpler proof of PCP theorem using a different technique by Dinur [2005].

- Basically, the PCP 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.

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

- Before that note that there is a trivial $\frac{7}{8}$-approximation for Max-3SAT.

- Given a 3SAT formula Φ for Max-3SAT with
  - 3-clauses $C_1, \ldots, C_m$ and
  - variables $x_1, \ldots, x_n$,

  assign each x_i True/False u.r. with probability $\frac{1}{2}$.

- This is a $\frac{7}{8}$-approximation for Max-3SAT:

- For each clause $C_i = (x_5 \wedge \overline{x_1} \wedge x_3)$, the probability that C_i is *not* satisfied is:

$$\Pr[x_5 = F] \times \Pr[x_1 = T] \times \Pr[x_3 = F] = \frac{1}{8}$$

- Thus each clause C_i is satisfied with probability $\frac{7}{8}$; so the expected number of satisfied clauses is at least $\frac{7}{8}m$ (this can be easily de-randomized).

- **Theorem 14** *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 15** *Approximating Max-3SAT with a factor better then $(1-\epsilon)$ is NP-hard for some constant $\epsilon > 0$.*

Proof of Theorem 14:

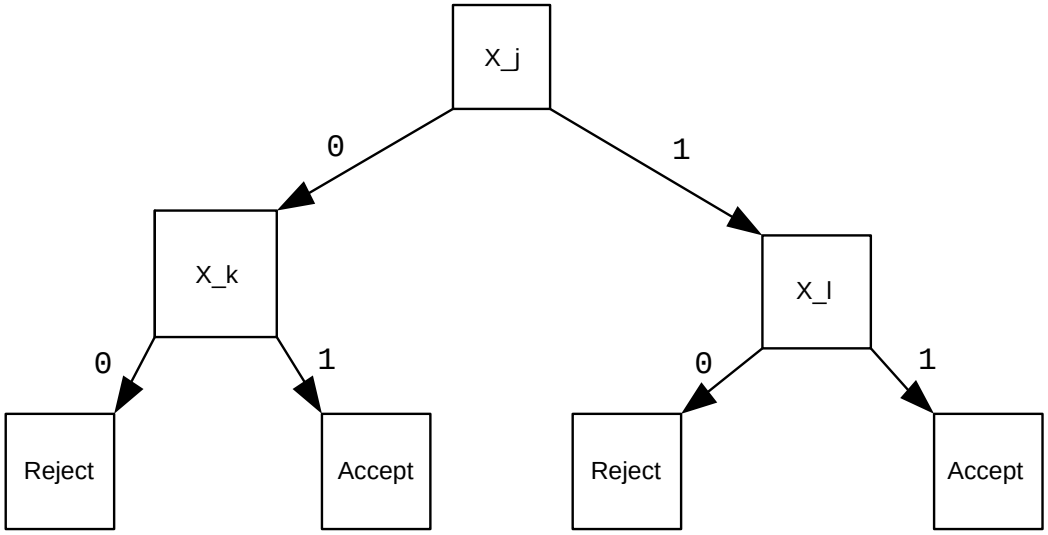
- By PCP theorem, SAT has a $PCP_{1,\frac{1}{2}}(O(\log n), O(1))$ verifier V. Let us assume that it is $PCP_{1,\frac{1}{2}}(d \log n, k)$ where d and k are some constants.

- 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 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 reads k positions of the proof; each of these positions is determined by the value read in previous position and the random bits.

- So it can be considered to evaluate a boolean binary decision tree of height at most k; where the decision to go left or right is based on the value read from the proof.



- Here suppose $k = 2$ and we have a fixed random bit string.
- Based on the first random bit the position we read 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

- In general, 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 node as a variable and every path from root to leaf forms a clause.

- 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.

- 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.

- This theorem showed that PCP theorem implies a gap-introducing reduction from SAT to Max-3SAT.

- The oposite is also true; i.e. assuming the existence of a gap-introducing reduction from SAT to Max-3SAT we can prove the PCP theorem, as follows.

- 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 $\frac{1}{2}$ of the 3-clauses of $g(y)$ can be satisfied.

- Define a proof that $y \in L$ to be a truth assignment satisfying $g(y)$.

- 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 more than $\frac{1}{2}$ of clauses in $g(y)$. So verifier $V(y, \pi)$ will reject with probability at least $\frac{1}{2}$.

- This, together with Theorem 14 implies that the PCP theorem is in fact equivalent to: "There is no PTAS for Max-3SAT."

Section 3:

Improved Hardness results

- Recall that PCP theorem says:

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

- Goal: reduce the number of query bits and also decrease the the probability of failure.

- **Theorem 16** *For some* $s < 1$:

$$NP = PCP_{1,s}(O(\log n), 3).$$

- Proof: Let $L \in NP$ be an arbitrary language; y be an instance of L.

- By PCP, we can construct a 3CNF formula F such that:
  - if $y \in L \Longrightarrow \exists$ a truth assignment for F s.t. all clauses of F are satisfied.
  - if $y \notin L \Longrightarrow$ 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.

- Verifier V given y and proof π, computes F in poly-time, then uses $O(\log n)$ bits to pick a random clause of F and query the truth assignment to its 3 variables.

- 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 $1 - \epsilon$.

- **Theorem 17 (Guruswami,Sudan,Lewin,Trevisan'93)** *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.

- **Theorem 18 (Karloff/Zwick, 97)**

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

- **Theorem 19 (Hástad'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* $\pi(i_1) \oplus \pi(i_2) \oplus \pi(i_3) = b$.

- **Corollary 20** *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* $(\frac{7}{6} + \epsilon)$,

- Recall that the simple algorithm we gave for Max-3SAT has approximation factor $\frac{7}{8}$.

- The best Hardness factor for VC is $10\sqrt{5} - 21 \approx 1.3606$.

- **Definition 21** *(MAX-CLIQUE) Given a graph with n vertices, find a maximum size clique in it, i.e. a complete subgraph of maximum size.*

- The best known algorithm has a factor of $O(\frac{n \cdot \log \log^2 n}{\log^3 n})$.

- Clique and Max-3SAT are both NP-hard, but why the approximation for clique is so bad?

- Hástad: for any $\epsilon > 0$ there is no polytime approximation algorithm for clique with factor $n^{\frac{1}{2}-\epsilon}$ (if P $\neq$ NP) or $n^{1-\epsilon}$ (if ZPP$\neq$ NP).

- Our goal is to prove a polynomial hardness for clique.

- 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 $\mathrm{PCP}_{1,\frac{1}{2}}(d \log n, q)$ verifier F for SAT, where d and q are constants (V exists by PCP).

- 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 $v_{r_i,\sigma}$ for each pair (i, σ), where r_i is one of the the random strings r_i and σ is a truth assignment to q variables. G has $n^d 2^q$ vertices.

- 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.

- 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.

- 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, and both are accepting.

- If ϕ is a yes instance then 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 so are consistent;

- Therefore G has a clique of size $\geq n^d$.

- For the case 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 this clique. Therefore the transcripts $\sigma_1...\sigma_c$ (partial truth assignments) are all consistent.

- We can extend this truth assignment to a whole proof (truth assignment) such that on random strings $i_1, ..., i_c$, verifier V accepts.

- Therefore the verifier accepts for more than $\frac{n^d}{2}$ strings, which contradicts the assumption that ϕ is a no instance.

- So it is NP-hard to decide whether:
  - G has a clique of size n^d
  - G every clique of G has size $< \frac{n^d}{2}$

- Note that the gap created here is exactly the soundness probability of the verifier; so the the smaller S is, the larger gap we get.

- By simulating a $\mathsf{PCP}_{1,\frac{1}{2}}(d \log n, q)$ verifier V for k times and accepting iff all of those simulations accept, we get a $\mathsf{PCP}_{1,\frac{1}{2^k}}(k \cdot d \log n, k \cdot q)$ verifier V'.

- Note that in this case the size of the construction G is $n^{kd}2^{kq}$, which is polynomial as long as k is constant.

- This will show a hardness of 2^k, which is a constant.

  Corollary 22 *For any constant S, it is NP-hard to approximation clique within a factor of S (say $S = 1/2^k$ in the above).*

- To get an n^{δ} 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 G becomes $2^{\Omega(\log^2 n)}$, which is super-polynomial.

- To get a polynomial hardness we need a

$$\mathsf{PCP}_{1,\frac{1}{n}}(O(\log n), O(\log n))$$

verifier.

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

Definition 23 *(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. To 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:
  - need $d \log n$ bits to choose the first vertex, and
  - need constant number of random bits to choose one neighbor at every step.
- Therefore, to have a random walk of length $O(\log n)$ in H we need only $O(\log n)$ bits.
- **Theorem 24** *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 $< \frac{1}{n}$.*
- Proof outline: By definition, 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.

- **Theorem 25**

$$\mathsf{PCP}_{1,\frac{1}{2}}(d \log n, q) \subseteq \mathsf{PCP}_{1,\frac{1}{n}}(O(\log n), O(\log n)).$$

- Proof: Let $L \in \mathsf{PCP}_{1,\frac{1}{2}}(d \log n, q)$ and $\mathcal{F}$ be a verifier for L.

- We give a $\mathsf{PCP}_{1,\frac{1}{n}}(O(\log n), O(\log n))$ verifier $\mathcal{F}'$ for L.

- $\mathcal{F}'$ builds the expander graph H of Theorem 24, then creates a random walk of length $k \log n$ using only $O(\log n)$ bits for some constant k

- This random walk yields $k \log n$ "random" string of length $d \log n$ each, which are the labels of the vertices of the walk.

- Then $\mathcal{F}'$ simulates $\mathcal{F}$ on each of these strings and accepts if and only if all these simulations accept.

- If $y \in L$ is a "yes" instance, then there is a proof π s.t. $\mathcal{F}$ accepts with probability 1 given π; so $\mathcal{F}'$ accepts.

- If $y \in L$ is a "no" instance, then $\mathcal{F}$ accepts on at most $\frac{n^d}{2}$ of the random strings.

- Let S be the set of vertices of H with those labels.

- Note that $|S| \leq \frac{n^d}{2}$. This means that $\mathcal{F}'$ accepts (wrongly) y only if the entire random walk is inside S.

- Now, based on Theorem 24 the probability that a random walk remains entirely in S is at most $\frac{1}{n}$

- Therefore, the probability that $\mathcal{F}'$ accepts y is at most $\frac{1}{n}$.

- This completes the proof of

$$\mathsf{PCP}_{1,\frac{1}{2}}(d \log n, q) \subseteq \mathsf{PCP}_{1,\frac{1}{n}}(O(\log n), O(\log n)).$$

- **Theorem 26** *For some $\delta > 0$, it is NP-hard to approximate clique within a factor of $\Omega(n^{\delta})$.*

- Proof: Given a SAT formula ϕ, let F be a $\mathsf{PCP}_{1,\frac{1}{n}}(d \log n, q \log n)$ verifier for it for some constants d and q.

- Construct the graph G from F in the same manner as we did earlier for the constant factor hardness.

- So the size of G is $n^d 2^{q \log n} = n^{d+q}$ and the gap created is equal to soundness probability, i.e. $\frac{1}{n}$.

- we have:

  - If ϕ is a yes instance then G has a clique of size n^d.

  - If ϕ is a no instance then every clique of G has size at most n^{d-1}.

  This creates a gap of n^{δ} with $\delta = \frac{1}{d+q}$.

Section 4:

Hardness of Set Cover

Hardness of Set Cover

- Our final lecture is the outline of the proof of a hardness of $O(\log n)$ for Set cover.
- We need to define another problem: Label Cover.
- This is a graph theoric representation of another proof system for NP (2 prover 1 round proof system).
- An instance of label cover consists of the followings:
  - $G(V \cup W, E)$ is a bipartite graph.
  - $[N] = \{1...N\}$, $[M] = \{1...M\}$ are 2 sets of labels, $[N]$ for the vertices in V and $[M]$ for the vertices in W.
  - $\{\Pi_{v,w}\}_{(v,w)\in E}$ denotes a (partial) function on every edge (v, w) such that $\Pi_{v,w} : [M] \rightarrow [N]$
- A labeling $l : V \rightarrow [N], W \rightarrow [M]$ is said to cover edge (v, w) if $\Pi_{v,w}(l(w)) = l(v)$.
- **Goal:** Given an instance of label cover, find a labeling that covers maximum fraction of the edge.

- It follows from PCP theorem that:

  Theorem 27 *There is an absolute $\epsilon > 0$ s.t. given an instance $\mathcal{L}(G, M = 7, N = 2, \{\Pi_{v,w}\})$ it is NP-hard to decide if*

  - $opt(\mathcal{L}) = 1$, *or*
  - $opt(\mathcal{L}) \leq 1 - \epsilon$

- From an instance $\mathcal{L}(\mathcal{G}(\mathcal{V}, \mathcal{W}, \mathcal{E}), [\mathcal{M}], [\mathcal{N}], \{\Pi_{vw}\})$, we obtain the k-power as following.

- we build $\mathcal{L}^k(G'(V', W', E'), [M'], [N'], \{\Pi'_{vw}\})$ where:
  - $V' = V^k$ (k-tuples of V)
  - $W' = W^k$
  - $[M]' = [M]^k$
  - $[N]' = [N]^k$
  - $(V', W') \in E' \Leftrightarrow (v_{i_j}, w_{i_j}) \in E,\ \forall i, j\ 1 \leq j \leq k$ $(V' = (v_{i_1}, \ldots, v_{i_k}), W' = (w_{i_1}, \ldots, w_{i_k}))$
  - $\Pi'_{vw}(b_1, \ldots, b_k) = \Pi_{v_{i_1}, w_{i_1}}(b_1), \Pi_{v_{i_2}, w_{i_2}}(b_2), \ldots, \Pi_{v_{i_k}, w_{i_k}}(b_k)$

- Theorem 27, together with a strong result of Raz'98 implies the following:

  Theorem 28 *There is a reduction from SAT to an instance* $\mathcal{L}(G(V,W,E),[7^k],[2^k],\{\Pi_{vw}\})$ *of label cover such that:*

  - *if* ϕ *is a yes instance* $\rightarrow$ $OPT(\mathcal{L}) = 1$
  - *if* ϕ *is a no instance* $\rightarrow$ $OPT(\mathcal{L}) = 2^{-ck}$ *for some constant* $c < 1$

  and $\mathcal{L} = n^{O(k)}$

- We need one more definition.

  A set-system with parameters m and l consists of:

  - U a universe (of elements)
  - $C_1, \ldots, C_m, \bar{C}_1, \ldots, \bar{C}_m$ are subsets of U
  - For any set of ℓ subsets from C_i's and $\bar{C}_i$'s that does not include a C_j's and $\bar{C}_j$ together, the union does not cover U.

- **Theorem 29** *Given* m, ℓ *there are explicit constructions for a set system with* $|U| = O(\ell \cdot \log m \cdot 2^{\ell})$.

- Consider a label cover instance $\mathcal{L}(G(V, W, E), M = [7^k], N = [2^k], \{\Pi_{v,w}\})$.
- Let's assume $|V| = |W|$. We build an instance of set cover $\mathcal{S}$ such that:
  - If $opt(\mathcal{L}) = 1$, then $opt(S) \leq |V| + |W|$.
  - If $opt(\mathcal{L}) < \frac{2}{l^2}$, then $opt(\mathcal{S}) > \frac{l}{16}(|V| + |W|)$.
- Consider a set system with $m = N = 2^k$ and l to be specified later.
- For every $e = (v, w) \in G$, we have a (disjoint) (m, l)-set system with universe U_e.
- Let $C_1^{vw}, \cdots, C_{N=m}^{vw}$ be the subsets of U_e.
- The union of all $U_e's$ (for all the edges e) is the universe of the set cover instance, denoted as

$$U = \bigcup_{(v,w)\in G} U_{vw}.$$

- Now we define the subsets. For every $v \in V$ ($w \in W$) and every label $i \in [2^k]$ ($j \in [7^k]$), we have a set

$$S_{v,i} = \bigcup_{w:(v,w)\in E} C_i^{vw} \qquad S_{w,j} = \bigcup_{v:(v,w)\in E} \overline{C_{\Pi_{vw}(j)}^{vw}}$$

- This completes the construction of $\mathcal{S}$ from $\mathcal{L}$.
- **Lemma 30** *If* $opt(\mathcal{L}) = 1$, *then* $opt(\mathcal{S}) \leq |V| + |W|$.
- Proof: Consider an optimal labeling $l : V \to [2^k], W \to [7^k]$ for $\mathcal{L}$.
- Because it is covering every edge $(v, w) \in E$, $\Pi_{vw}(l(w)) = l(v)$.
- This labeling defines a label for every vertex and every pair of vertex/label corresponds to a set in $\mathcal{S}$.
- From $C^{vw}_{l(v)} \subseteq S_{v,l(v)}$ and $\overline{C^{vw}_{l(v)}} = \overline{C^{vw}_{\Pi_{vw}(l(w))}} \subseteq S_{w,l(w)}$:

$$S_{v,l(v)} \cup S_{w,l(w)} \supseteq U_{vw}.$$

- Because all U_e's for $e \in E$ are covered, U is covered. So we have a set cover of size $|V| + |W|$.
- **Lemma 31** *if* $opt(\mathcal{S}) \leq \frac{l}{16}(|V| + |W|)$, *then* $opt(\mathcal{L}) \geq \frac{2}{l^2}$.
- Proof: From the set cover solution, we assign labels (maybe more than one label) to the vertices.
- If $S_{v,i}$ is in the solution, v gets label i.

- Since there are $\leq \frac{l}{16}(|V| + |W|)$ sets and $|V| + |W|$ vertices, the average number of labels per vertex is $\leq \frac{l}{16}$.

- We discard vertices with $> \frac{l}{2}$ labels.

- $\leq \frac{|V|}{4}$ vertices from each of V and W are discarded. Let V' and W' be the vertices remaining.

- so $|V'| > \frac{3}{4}|V|$ and $|W'| > \frac{3}{4}|W|$.

- Pick an edge $e = (v, w)$ from G randomly.

$$\Pr[v \in V' \, and \, w \in W'] \geq 1 - (\frac{1}{4} + \frac{1}{4}) = \frac{1}{2}.$$

- so $\geq \frac{1}{2}$ edges of G are between V' and W'.

- Define

$$T_v = \{S_{v,i} : \ i \text{ is a label of } v\}$$

$$T_w = \{S_{w,j} : \ j \text{ is a label of } w\}.$$

- We have $|T_v| \leq \frac{l}{2}$ and $|T_w| \leq \frac{l}{2}$.

- Note that sets in $T_v \cup T_w$ cover U_{vw}, i.e.

$$X_1 = \{C_i^{vw} : \ i \text{ is a label of } v\} \cup$$
$$X_2 = \{\overline{C^{vw}_{\Pi_{vw}(j)}} : \ j \text{ is a label of w}\}$$

  covers universe U_{vw}

- Since $|X_1| \leq \frac{l}{2}$ and $|X_2| \leq \frac{l}{2}$, $\exists$ a set $C_i^{vw} \in X_1$ and $\overline{C^{vw}_{\Pi_{vw}(j)}} \in X_2$ that are in $X_1 \cup X_2$.

- Because we pick labels of v and w randomly, with probability $\geq (\frac{2}{l})^2 = \frac{4}{l^2}$ we have set C_i^{vw} for v and $\overline{C_j^{vw}}$ for w i.e. the labels i for v and j for w cover edge $e \in E$.

- Thus the expected fraction of edges between V' and W' that are covered is $\geq \frac{4}{l^2}$.

- Therefore, at least a fraction of $\frac{2}{l^2}$ of edges of G are covered.

- This lemma is equivalent to saying that if $opt(\mathcal{L}) < \frac{2}{l^2}$ then $opt(\mathcal{S}) > \frac{l}{16}(|V| + |W|)$.

- Thus we have:

- If $opt(\mathcal{L}) = 1$, then $opt(S) \leq |V| + |W|$.
- If $opt(\mathcal{L}) < \frac{2}{l^2}$, then $opt(\mathcal{S}) > \frac{l}{16}(|V| + |W|)$.

- Let $l \in \Theta(2^{\frac{\delta k}{2}})$. Then, $l^2 \in \Theta(2^{\delta k})$.

- We get a hardness of $\Omega(l)$ for $\mathcal{S}$. The size of $\mathcal{S}$ is $n^{O(k)} \cdot O(l \cdot \log m \cdot 2^l)$.

- If $k = c \log \log n$ for sufficiently large c, $l = O(2^{O(\log \log n)}) \geq \log n \log \log n$.

- Thus $\log |\mathcal{S}| = O(\log \log n \cdot \log n + \log l + \log \log \log n + l) = \Theta(l)$.

  We have the following hardness result for set cover:

  Theorem 32 *Unless $NP \subseteq DTIME(n^{O(\log \log n)})$, set cover has no $o(\log n)$-approximation algorithm.*