22C:253 Lecture 1

Scribe: Narendra B. Devta Prasanna

August 28, 2002

A *decision problem* is a problem Π such that every instance I of Π has a "yes/no" solution. An algorithm A that solves Π produces a correct "yes/no" answer for each instance I of Π.

Every instance I of an *optimization problem* Π has a non-empty feasible set of solutions, denoted $F_\Pi(I)$ such that associated with every feasible solution, $s \in F_\Pi(I)$, there is a non-negative rational cost, denoted $C_\Pi(I, s)$. Any feasible solution that optimizes $C_\Pi(I, s)$ is called an *optimal solution* for I, denoted $OPT_\Pi(I)$. An optimization problem Π can either be a maximization problem or a minimization problem and depending on this $OPT_\Pi(I)$ is a feasible solution that either maximizes cost or minimizes cost.

Typically, the following problems related to Π have polynomial time solutions:

- Determining if a given instance I is a legal instance of Π.
- Checking if a given solution s is feasible for a given instance I (that is, determining if $s \in F_\Pi(I)$).
- Given I and a feasible soltion s, determining the cost $C_\Pi(I, s)$.

So all of these problems are easy and the hardness of Π arises from the fact that $F_\Pi(I)$ is very large and there is no known efficient way of searching $F_\Pi(I)$ to find an optimal feasible solution. Specifically, the optimization problems we will consider will be all be NP-hard. What does it mean for an optimization problem to be NP-hard?

We can view an optimization problem Π as a decision problem by attaching to each problem instance I a rational B. So each instance of the decision version of Π is a pair (I, B). If Π is a maximization problem, then its decision version asks: *Does I have a feasible solution s with cost $C_\Pi(I, s) \geq B$?*. If Π is a minimization problem, then its decision version asks: *Does I have a feasible solution s with cost $C_\Pi(I, s) \leq B$?*. Given this, the following propositions are obvious.

Proposition 1 *If an optimization problem Π can be solved in polynomial time, then its decision version can also be solved in polynomial time.*

Proposition 2 *If the decision version of an optimization problem Π is NP-hard, then Π is also NP-hard.*

So whenever we talk about an optimization problem being NP-hard, we are actually talking about its decision version being NP-hard.

An algorithm A is a *factor-f approximation algorithm* for a minimization problem Π if

- A runs in poly-time, and
- For every instance I of Π, A finds a feasible solution s such that

$$C_\Pi(I, s) \le f \cdot OPT_\Pi(I). \tag{1}$$

Note that $f \ge 1$. If Π is a maximization problem, then A is a *factor-f approximation algorithm* if A runs in polynomial-time and for every instance I of Π, finds a feasible solution s such that

$$C_\Pi(I, s) \ge f \cdot OPT_\Pi(I). \tag{2}$$

Note here that $f \le 1$.

We will now discuss easy approximationm algorithms for some well-known problems. The table below shows the problems we will consider and the approximation factor f that the algorithms we present will achieve. Roughly speaking, these are the best known approximation factors for each of these problems.

Problem, Π	Factor f
Graph Coloring	$O(n^c)$ for $c < 1$
Set Cover	$O(\lg n)$
Cardinality Vertex Cover	2
Minimum Makespan	$(1 + \epsilon)$
Knapsack	$(1 + \epsilon)$

The approximation factor $(1 + \epsilon)$ for *Minimum Makespan* and *Knapsack* problems, means that for every $\epsilon > 0$, there is an algorithm A_ϵ such that A_ϵ produces a solution that is within $(1 + \epsilon)$ times the optimal. So technically speaking, here we have a family of algorithms rather than a single algorithm. This family of algorithms is called a *polynomial time approximation scheme (PTAS)*. The running time of a PTAS depends inversely on ϵ and we distinguish the case when the running time of a PTAS is a polynomial function of $1/\epsilon$. A PTAS for which this is the case is called a *fully polynomial time approximation scheme (FPTAS)*. A PTAS and an FPTAS will be defined more precisely later. We will present an FPTAS for Knapsack and a PTAS for Minimum Makespan.

Example of Approximation algorithm. A *vertex cover* for a graph $G = (V, E)$ is a subset $V' \subseteq V$ such that for every edge $\{u, v\} \in E$, either $u \in V'$ or $v \in V'$ (or both). If G is a vertex-weighted graph with weight function $w : V \to Q^+$ then the *weight* of a vertex cover is simply the sum of the weights of the vertices in it.

Vertex Cover (VC)
Input: A vertex-weighted graph $G = (V, E)$ with weight function $w : V \to Q^+$.
Output: A vertex cover of G with minimum weight.

In the "cardinality" version of the problem, called *Cardinality Vertex Cover (CVC)*, vertices have unit weights. This essentially means that we are looking for a vertex cover with fewest vertices in it.

We want to come up with an algorithm A such that for every instance I of CVC, A produces a vertex cover s such that

$$C_{CVC}(I, s) \le 2 \cdot OPT_{CVC}(I) \tag{3}$$

The problem with showing such an inequality is that we don't know anything about $OPT_{CVC}(I)$. This is the fundamental problem faced by people designing approximation algorithms Typically, to get around this problem, we first show a lower bound $LB_\Pi(I)$ on $OPT_\Pi(I)$. That is,

$$LB_\Pi(I) \leq OPT_\Pi(I) \qquad \text{for all } I \tag{4}$$

and then show that

$$C_\Pi(I,s) \leq 2 \cdot LB_\Pi(I) \leq 2 \cdot OPT_\Pi(I) \tag{5}$$

It turns out that it is extremely easy to obtain a lower bound on $OPT_{CVC}(I)$.

A *matching* M in a graph is a set of edges, no two of which share an endpoint. A *maximal matching* is a matching that is maximal with respect to inclusion, that is, adding any other edge to the maximal matching makes it not a matching.

Algorithm for CVC

1. Compute a maximal matching M of G.
2. Output the endpoints of the edges in M.

Lemma 3 *The above algorithm produces a vertex cover of G.*

Proof: Let V' be the set of endpoints of the edges in M. If V' is not a vertex cover, then there is an edge $\{u,v\} \in E$ such that $u \notin V'$ and $v \notin V'$. Hence, $\{u,v\}$ can be added to M and it would still be matching. This contradicts the fact that M is a maximal matching. Therefore V' is a vertex cover. □

Lemma 4 *For any matching M of G and any vertex cover V' of G, $|M| \leq |V'|$.*

Proof: For every edge in M, there is at least one of its end points in V'. Since M contains edges no two of which share an endpoint, $|M| \leq |V'|$. □

A corollary of the above lemma is that if OPT is the size of a minimum cardinality vertex cover of G and M is a maximal matching, $|M| \leq OPT$ If we let V' denote the output of the above algorithm, we have that $|V'| = 2 \cdot |M|$ therefore $|V'| \leq 2 \cdot OPT$. This shows that the above algorithm is a factor-2 approximation algorithm for CVC.

Remarks:

- Rather than use $OPT_\Pi(I)$ we will use OPT when Π and I are clear from the context. In fact, we will use OPT to denote not only the optimal cost, but also the optimal solution sometimes.
- A factor-2 approximation can also be achieved for the usual (weighted) vertex cover problem.