**CIS 511: Theory of Computation** **Apr 20, 2017**

# Lecture 25: Practical Approaches to NP

*Professor Sampath Kannan* *Zach Schutzman*

**NB**: *These notes are from CIS511 at Penn. The course followed Michael Sipser's Introduction to the Theory of Computation (3ed) text.*

## Optimization Problems

Optimization problems usually minimize or maximize things. We would like ways to solve $NP$-Complete optimization problems, like minimum Vertex Cover, maximum clique, etc.

**Definition 25.1** *An algorithm $A$ achieves an* ***approximation ratio*** *of $C$ if for all inputs $x$, $\frac{A(x)}{OPT(x)} \leq C$, where $A(x)$ is $A$'s result and $OPT(x)$ is the optimal result.*

### Vertex Cover

*Let's think about Vertex Cover. Given a graph $G$, we want to find the smallest set of vertices such that every edge of $G$ is incident to a vertex in the cover.*

***Definition 25.2*** *A* ***maximal matching*** *is a set of edges in a graph which form a matching such that no other edge can be added and have it still remain a matching.*

*A maximal matching can be found efficiently by a greedy algorithm. Let $M$ be a maximal matching and $V(M)$ be the set of vertices in the matching. We have $|V(M)| = 2|M|$, so $OPT(G) \geq |M|$. We claim $V(M)$ is a Vertex Cover. To see this, suppose that it is not. Then some edge $(u, v)$ is not covered by $V(M)$. But then we could add $(u, v)$ to $M$, contradicting the assumption that $M$ is a maximal matching.*

*Therefore, a maximal matching is a 2-approximation for finding a Vertex Cover.*

*Note that reductions do not necessarily preserve approximation factors. Think of Vertex Cover and Independent Set. Simply taking the complement of a 2-approximate Vertex Cover could result in a size zero Independent Set if every vertex was added to the cover. In fact, Independent Set is strongly non-approximable (as is clique), and finding a good approximation algorithm would prove $P = NP$*

### Set Cover

*Now, let's think of Set Cover. Here, we are given a universe $S$ of $n$ elements and a collection of $m$ subsets $S_1, S_2, \ldots, S_m$. We want to find a minimum number of subsets such that their union is equal to $S$. Vertex cover is a special case of Set Cover, where the universe is the set of edges and the subsets are the sets of edges incident to each vertex.*

*A greedy algorithm to approximate Set Cover is to iteratively pick the $S_i$ with the largest number of uncovered elements until we cover everything. Suppose the greedy algorithm selects $k$ subsets $S_{i_1}, S_{i_2}$. We know that $S_{i_1}$*

*covers at least $\frac{n}{k}$ elements, as either it or a strictly smaller subset must be included in the optimal solution, as our greedy algorithm picks the largest subset at the first step. Consider $S_{i_2}$. For the same reason, we have that $S_{i_2}$ covers at least $\frac{n(1-\frac{1}{k})}{k}$ elements, and so on. We can upper-bound the number of uncovered elements after $t$ rounds by $n(1-\frac{1}{k})^t$. We want to find the $t$ such that this becomes smaller than 1. By taking some logarithms and doing algebra, we get $l \approx k$*
*ln(n) as the number of sets the greedy algorithm will choose. This is therefore a $\ln(n)$-approximation to Set Cover. We can also prove that this is tight. If Set Cover has a $(\ln(n) - \epsilon)$-approximation algorithm, then $P = NP$.*

## Subset-Sum

*The optimization version of the Subset-Sum problem, which asks to find a subset of a universe $a_1, a_2, \ldots, a_n$ whose sum is maximum subject to it being less than or equal to a target $T$. To solve this exactly, we can just check all $2^n$ subsets, and pick the one that sums to the largest value less than or equal to $T$. We can do this in exponential time with a dynamic programming approach by keeping an array $L$ such that $L_i$ is the list of sums we can get with a subset of the first $i$ elements, and we construct $L_{i+1} = L_i \oplus \{0, a_{i+1}\}$. This runs in time $nT$, but running time of $T$ is not polynomial.*

*We can find an arbitrarily good approximation to this problem. That is, for any $\epsilon > 0$, we have $\frac{OPT(I)}{A(I)} = 1+\epsilon$. Our approximation is similar to the dynamic programming. At each step, we eliminate from $L$ any value that is above $T$ or within a factor of $(1+\delta)$ of some other value. That is, for each element $z$, eliminate all remaining elements between $z$ and $1+\delta$. This trimming process means that we have elements that are no closer together than $z_1$, $z_1(1+\delta)$, $z_1(1+\delta)^2$, and so on, where $z_1$ is the first element of the list.*

*Since $z_1$ is at least 1, there are at most $\log_{1+\delta}(T)$ elements in any $L_i$. This is approximately $\frac{\log(T)}{\delta}$, so we get a relation that the length of the list increases as $\delta$ gets closer to zero. We also know that the length of $L_{i+1}$ is twice that of $L_i$, but since we are trimming at each step, the list never gets too big. $\log(T)$ is polynomial in $T$, so the total running time is $\frac{n\log(T)}{\delta}$.*

*How good of an approximation is this? By pruning values, we get no further than $(1+\delta)^n$, so if we set $\delta = \frac{\epsilon}{n}$, we get an approximation factor which looks like $(1+\frac{\epsilon}{n})^n$, which is close to $e^{\epsilon}$, and for small $\epsilon$, this is roughly $1+\epsilon$.*