Dealing with NP-Completeness

Note: We will resume talking about optimization problems, rather than yes/no questions.

What to do?

- Give up
- Solve small instances
- Look for special structure that makes your problem easy (e.g. planar graphs, each variable in at most 2 clauses, ...)
- Run an exponential time algorithm that might do well on some instances (e.g. branch-and-bound, integer programming, constraint programming)
- Heuristics – algorithms that run for a limited amount of time and return a solution that is hopefully close to optimal, but with no guarantees
- Approximation Algorithms – algorithms that run in polynomial time and give a guarantee on the quality of the solution returned
Heuristics

• Simple algorithms like “add max degree vertex to the vertex cover”
• Local search
• Metaheuristics are popular
  – simulated annealing
  – tabu search
  – genetic algorithms
  – GRASP
  – Greedy
Approximation Algorithms

Set up: We have a minimization problem $X$, inputs $I$, algorithm $A$.

- $OPT(I)$ is the value of the optimal solution on input $I$.
- $A(I)$ is the value returned when running algorithm $A$ on input $I$.

Def: Algorithm $A$ is an $\rho$-approximation algorithm for Problem $X$ if, for all inputs $I$

- $A$ runs in polynomial time
- $A(I) \leq \rho OPT(I)$.

Note: $\rho \geq 1$, small $\rho$ is good.
Methodology

Lower bound: Given an instance $I$, a lower bound, $LB(I)$ is an “easily-computed” value such that $LB(I) \leq OPT(I)$.

Methodology

- Compute a lower bound $LB(I)$.
- Give an algorithm $A$, that computes a solution to the optimization problem on input $I$ with a guarantee that $A(I) \leq \rho LB(I)$ for some $\rho \geq 1$.
- Conclude that $A(I) \leq \rho OPT(I)$. 
Matching

- A matching $M$ of a graph $G$ is a subset of the edges $M \subseteq E$, such that each vertex $v \in V$ is incident to at most one edge in $M$.
- A maximum matching can be computed in polynomial time.
- A maximum matching in a bipartite graph can be computed via maximum flow.
A 2-approximation for Vertex Cover

First find a good lower bound: A matching!

Given a graph $G$, let

- $MM(I)$ be the size of the maximum matching on $I$.
- $OPT(I)$ be the size of the minimum-sized vertex cover on $I$.
- $VC(I)$ be the size of the vertex cover returned by the algorithm below.

Claim: $MM(I) \leq OPT(I)$

Proof: Look at each edge in the maximum matching $M$. Each vertex in a vertex cover covers at most one edge in $M$.

Algorithm

1. Compute a maximum matching $M$.
2. For each edge $(v, w) \in M$, add both $v$ and $w$ to $C$. 
**Analysis**

$C$ is a vertex cover: .

**Proof:** Assume not. Then some edge $(v, w)$ has neither $v$ nor $w$ in $C$. But then neither $v$ nor $w$ is incident to an edge in $M$, which means that you could add $(v, w)$ to $M$, contradicting the fact that $M$ is a maximum matching.

**Solution value:**

$$VC(I) = 2MM(I) \leq 2OPT(I)$$

Therefore we have a 2-approximation algorithm.
Euler Tour

• Give an even-degree graph $G$, an Euler Tour is a (non-simple) cycle that visits each edge exactly once.

• Every even-edge graph has an Euler tour.

• You can find one in linear time.
Variant: We will consider the symmetric TSP with triangle-inequality.

- $w(a, b) = w(b, a)$
- $w(a, b) \leq w(a, c) + w(c, b)$

Notes:

- Without triangle inequality, you cannot approximate TSP (unless P=NP)
- Asymmetric version is harder to approximate.
Approximating TSP

- A minimum spanning tree is a lower bound on the TSP. $MST(I) \leq OPT(I)$

- A minimum spanning tree doubled is an even degree graph $GG$, and therefore has an Euler tour of total length $GG(I)$, with $GG(I) = 2MST(I)$.

- Because of triangle inequality, we can “shortcut” the Euler tour $GG$ to find a tour with $TSP(I) \leq GG(I)$

Combining, we have

$$TSP(I) \leq GG(I) = 2MST(I) \leq 2OPT(I)$$

- 2-approximation for TSP
- 3/2-approximation is possible.

- If points are in the plane, there exists a polynomial time approximation scheme, an algorithm that, for any fixed $\epsilon > 0$ returns a tour of length at most $(1 + \epsilon)OPT(I)$ in polynomial time. (The dependence on $\epsilon$ can be large).
MAX-3-SAT

**Definition**  Given a boolean CNF formula with 3 literals per clause. We want to satisfy the maximum possible number of clauses.

**Note:**  We have to invert definition of approximation, want to find $A(I) \geq \rho OPT(I)$.

**Algorithm**

- Randomly set each variable to true with probability $1/2$. 
Analysis

- Let $Y$ be the number of clauses satisfied.
- Let $m$ be the number of clauses. ($m \geq OPT(I)$).
- Let $Y_i$ be the i.r.v representing the $i$ th clause being satisfied.
- $Y = \sum_{i=1}^{m} Y_i$.
- $E[Y] = \sum_{i=1}^{m} E[Y_i]$.
- What is $E[Y_i]$, the probability that the $i$ th clause is true?
  - The only way for a clause to be false is for all three literals to be false
  - The probability a clause is false is therefore $(1/2)^3 = 1/8$
  - Probability a clause is true is therefore $1 - 1/8 = 7/8$.
- $E[Y] = (7/8)m$
- $E[Y] = (7/8)m \geq (7/8)OPT(I)$

Conclusion 7/8 -approximation algorithm.
Set Cover

An instance $(X, \mathcal{F})$ of the set-covering problem consists of a finite set $X$ and a family $\mathcal{F}$ of subsets of $X$, such that every element of $X$ belongs to at least one subset in $\mathcal{F}$:

$$X = \bigcup_{S \in \mathcal{F}} S.$$

We say that a subset $S \in \mathcal{F}$ covers its elements. The problem is to find a minimum-size subset $\mathcal{C} \subseteq \mathcal{F}$ whose members cover all of $X$:

$$X = \bigcup_{S \in \mathcal{C}} S.$$
Greedy Algorithm

**Greedy-Set-Cover**$(X, \mathcal{F})$

1. $U \leftarrow X$
2. $C \leftarrow \emptyset$
3. while $U \neq \emptyset$
4. do select an $S \in \mathcal{F}$ that maximizes $|S \cap U|$  
5. $U \leftarrow U - S$
6. $C \leftarrow C \cup \{S\}$
7. return $C$

**Claim:** If the optimal set cover has $k$ elements, then $C$ has at most $k \log n$ elements.
Proof

Claim: If the optimal set cover has $k$ sets, then $C$ has at most $k \log n$ sets.

Proof:

• Optimal set cover has $k$ sets.

• One of the sets must therefore cover at least $n/k$ of the elements.

• First greedy step must therefore choose a set that covers at least $n/k$ of the elements.

• After first greedy step, the number of uncovered elements is at most $n - n/k = n(1 - 1/k)$.
Proof continued

Iterate argument

• On remaining uncovered elements, one set in optimal must cover at least a $\frac{1}{k}$ fraction of the remaining elements.

• So after two steps, the number of uncovered elements is at most

$$n \left(1 - \frac{1}{k}\right)^2$$

So after $j$ iterations, the number of uncovered elements is at most

$$n \left(1 - \frac{1}{k}\right)^j \leq n e^{-j/k}$$

When $j = k \ln n$, the number of uncovered elements is at most

$$n e^{-j/k} = n e^{-k \ln n/k} = n e^{-\ln n} = n/n = 1$$

Therefore, the algorithm stops after choosing at most $k \ln n$ sets (without knowing $k$).