1 What To Do If Your Problem is NP-hard

NP-hardness does not mean you can give up!!!

• What do you do if you want to solve an optimization problem whose decision version is NP-complete?

• Let’s consider the case of vertex cover, which we briefly looked at when we introduced the idea of hard problems.

• **Idea 1**: use an exponential-time algorithm.

• If your inputs are all small, this is fine. (*Example*: the old GRE analytic section)

• (Also, if you never hit the worst case in practice, this is fine!)

• However, this approach does not scale!

• **Idea 2**: add constraints until the problem becomes polynomial-time solvable.

• **Example**: if $G$ is bipartite, the size of a minimum vertex cover equals the size of a maximum matching (Koenig’s Theorem), which can be found in polynomial time.

• But your problem may not be constrainable!

• **Idea 3**: find a polynomial-time optimization algorithm whose answer is “close” to best possible.

• Heuristic algorithms for optimization (gradient descent, simulated annealing, Gibbs sampling, etc etc etc) try various hacks to get close to either local or global optimum.

• Can we quantify how close a heuristic gets to the optimum?

• Maybe empirically, if we can afford to compute true optimum for enough inputs to run a benchmark.

• Sometimes, however, we can *prove* that a heuristic *always* gets close to the optimum!

We can adapt the idea of competitive analysis we saw last time to design algorithms that approximately solve really hard problems.

• Consider an optimization problem whose goal is to find a feasible solution of *minimum* cost.
For an arbitrary instance $x$ of the problem, let $C^*(x)$ be the cost of an optimal solution to $x$.

Let $A$ be an algorithm that computes a feasible solution for any instance, and let $C(x)$ be the cost of its solution for instance $x$.

**Defn:** we say that $A$ is an $f(n)$-approximation algorithm for our problem if, for any instance $x$ of size $n$, we have

\[
\frac{C(x)}{C^*(x)} \leq f(n).
\]

In other words, for an input of size $n$, $A$ gives a solution no worse than $f(n)$ times the optimum.

$f(n)$ is called the approximation ratio for $A$.

A similar definition holds for maximization problems.

Approximation algorithms are heuristics that come with a guarantee: they never return a solution more than $f(n)$ times worse than the optimum.

## 2 Approximating Vertex Cover

We will give polytime algorithm for vertex cover whose solution is never worse than twice the optimum.

- Given graph $G$, construct a cover $T$ as follows.
- Pick an arbitrary edge $e = (u, v)$ in $G$, and add both $u$ and $v$ to $T$.
- Delete $u$, $v$, and their incident edges from $G$.
- Repeat until $G$ contains no more edges.
- Call this algorithm FAST-COVER.

Does FAST-COVER efficiently produce a correct vertex cover?

- No edge is removed from $G$ unless at least one of its endpoints is covered by a vertex in $T$.
- When algo terminates, all edges have been removed.
- Hence, $T$ is a vertex cover for $G$. QED
- Moreover, selecting an edge from $G$ and removing an edge from it can both be done in time polynomial in $|G|$, so the whole algorithm is clearly polynomial-time.

Is FAST-COVER an approximation algorithm?

- **Lemma:** FAST-COVER is a 2-approximation algorithm for vertex cover.
- We will follow a similar proof strategy to what we did for online competitive analysis.
• Let \( x \) be an arbitrary instance of vertex cover.

• First, find a lower bound \( L(x) \) on the cost \( C^*(x) \) of a minimal cover for \( x \).

• Then, find an upper bound \( U(x) \) on the cost \( C(x) \) of FAST-COVER’s solution, in terms of \( L(x) \).

• Conclude that the approximation ratio for FAST-COVER is bounded as follows:

\[
\frac{C(x)}{C^*(x)} \leq \frac{C(x)}{L(x)} \leq \frac{U(x)}{L(x)}.
\]

• Finally, show that \( \frac{U(x)}{L(x)} \leq 2 \).

3 Proving the 2-Approximation

Let \( G = (V, E) \) be an input to vertex cover.

• First, we lower-bound the optimum.

• Let \( M \subseteq E \) be the list of edges of \( G \) whose endpoints are chosen by FAST-COVER.

• By construction, no two edges in \( M \) share an endpoint (that is, \( M \) is a matching for \( G \).

• Hence, any valid vertex cover for \( G \) must contain at least one endpoint of every edge in \( M \).

• Hence, every valid vertex cover for \( G \) has size at least \(|M|\).

One step down, two to go.

• Second, we upper-bound the algorithm’s solution.

• FAST-COVER adds both endpoints of every edge in \( M \) to the cover.

• Hence, its solution has size \( 2|M| \).

Finally, put upper and lower bounds together.

• We have shown that \( C(G) = 2|M| \), while \( C^*(G) \geq |M| \).

• Hence, FAST-COVER’s approximation ratio for \( G \) is

\[
\frac{C(G)}{C^*(G)} \leq \frac{2|M|}{|M|} = 2.
\]

• Conclude that FAST-COVER is a 2-approximation algorithm for vertex cover. QED
4 Set Cover – a Generalization of Vertex Cover

Here’s a very useful generalization of vertex cover.

• You are trying to collect a complete set of Pokemon cards!
• A complete set contains cards $f_1, f_2, \ldots, f_n$.
• There are $m$ dealers in town, but none have the complete set of cards.
• Calling around, you find that the $i$th dealer has cards $f_{i1}, f_{i2}, \ldots, f_{ik}$.
• How many dealers must you visit to collect the whole set?
• (This is a model for lots of resource allocation problems.)

Let’s abstract a little.

• Let $X$ be a universe of elements.
• You are given a collection $F$ of $m$ sets each a subset of $X$.
• **Defn:** a collection $C \subseteq F$ of sets is a set cover for $X$ if every element of $X$ is in at least one $S \in C$.
• **Problem:** Find a set cover $C$ for $X$ containing as few sets as possible.
• This is the *minimum set cover problem*.
• Can show that its decision version (Does $X$ have a set cover of size at most some $k$?) is NP-complete (exercise – try reducing from vertex cover).

5 An Approximation Algorithm for Set Cover

Consider the following greedy algorithm for set covering, which builds up a cover by sequentially picking sets $S_1, S_2, \ldots$.

• Repeat the following loop while $X$ is not covered.
• Let $U_{i-1}$ be the set of elements left uncovered after choosing sets $S_1 \ldots S_{i-1}$.
• For each unused set $S \in F$, compute
  $$\delta(S) = |U_{i-1} \cap S|$$
• Choose $S_i$ to be a set that maximizes $\delta(S_i)$, and add $S_i$ to the cover.
• $U_i \leftarrow U_{i-1} - S_i$. 
• Example:

Call this algorithm **GREEDY-COVER**.

• *Correctness?* We keep picking sets until every elt is covered.

• Hence, when we are done, $C$ is a set cover.

• *Cost?* Naively, we can spend $O(|X||\mathcal{F}|)$ in each loop iteration to compute $\delta$’s, so algo runs in time at most $O(|X|^2|\mathcal{F}|)$.

• (We can do much better with some clever hashing.)

• OK, but what about an approximation ratio?

# 6 Approximation Bound

**Thm:** Let $|X| = n$. Then GREEDY-COVER is an $O(\log n)$-approximation algorithm.

• In other words, cost depends on size of universe to be covered.

• Assumes nothing about the number or size of sets in $\mathcal{F}$.

• **Pf:** let $S_i$ be the $i$th set chosen by GREEDY-COVER, and let $\delta_i = |U_{i-1} \cap S_i|$ be the number of elements covered for the first time by $S_i$.

• First, observe that $\delta_i \leq \delta_{i-1}$ for every $i > 1$.

• (Otherwise, GREEDY-COVER would have chosen $S_i$ before $S_{i-1}$.)

• Now let $k^*$ be size of an optimal cover for $X$ from $\mathcal{F}$.

**Claim:** $\delta_i \geq \frac{|U_{i-1}|}{k^*}$.

• **PofC:** let $V = \{V_1 \ldots V_{k^*}\}$ be an optimal cover for $X$.

• $V$ covers all of $X$, and therefore all of $U_{i-1}$, using only $k^*$ sets.

• Hence, *average* number of elements of $U_{i-1}$ covered by a set in $V$ is at least $|U_{i-1}|/k^*$.

• But then *some* set $V_j \in V$ covers at least $|U_{i-1}|/k^*$ such elements!
• GREEDY-COVER selects set $S_i$ that covers the most elements of $U_{i-1}$; hence, $S_i$ covers at least as many such elts as does $V_j$.

• (If nothing else, $V_j$ is available to GREEDY-COVER because nothing in $U_{i-1}$ has been covered yet, so $V_j$ cannot have been chosen.)

• Conclude that $S_i$ covers at least $|U_{i-1}|/k^*$ elts of $U_{i-1}$.

Back to main proof!

• Suppose we run GREEDY-COVER for $k^* + 1$ iterations.

• From first fact, we have

$$\delta_1 \geq \delta_2 \geq \ldots \delta_{k^*} \geq \delta_{k^*+1}. $$

Hence,

$$\sum_{i=1}^{k^*} \delta_i \geq k^* \delta_{k^*+1}. $$

• Applying second fact, we have

$$\sum_{i=1}^{k^*} \delta_i \geq k^* \frac{|U_{k^*}|}{k^*} = |U_{k^*}|. $$

• Now LHS above is total number of elts covered by first $k^*$ sets used by GREEDY-COVER, while RHS is number of elts left uncovered. Sum of LHS and RHS is exactly $|X|$.

• LHS $\geq$ RHS, so GREEDY-COVER covers at least $|X|/2 = n/2$ elements using $k^*$ sets, leaving at most $n/2$ elts to cover.

• (Note that the remaining elts can also be covered with (at most) $k^*$ sets, so we can recursively do same analysis to show that next $k^*$ sets chosen cover at least $n/4$ elts, then $n/8$, and so on.)

• Let $G(n)$ be total number of sets used by GREEDY-COVER on $X$ of size $n$.

• Then $G(n)$ satisfies the recurrence

$$G(n) \leq k^* + G(n/2) $$

which implies that $G(n) = O(k^* \log n)$.

• Conclude that $\frac{G(n)}{k^*} = O(\log n)$. QED

Basic set cover prefers to use as few sets as possible to form the cover. What if we change the optimization criterion?

• Assign each set $S$ a weight $w(S)$.

• In weighted min set cover, the objective is to minimize the total weight of the sets used to form the cover.
• (The unweighted problem is equivalent to setting every set’s weight equal to 1.)

• We can tweak GREEDY-COVER to solve the weighted version as follows.

• After selecting the first \( i-1 \) sets, for each unused set \( S \in \mathcal{F} \), compute

\[
\delta'(S) = \frac{|U_{i-1} \cap S|}{w(S)}.
\]

• Select the set that maximizes \( \delta' \), i.e. that covers the most uncovered elements per unit weight.

• We can then easily modify the unweighted proof to show that this algorithm uses at most \( O(\log n) \) times the weight of an optimal (i.e. min-weight) cover – replace optimal size \( k^* \) with optimal weight \( w^* \) everywhere.

As another special case, if no set in the input contains more than \( d \) elements, GREEDY-COVER (weighted or unweighted) can be shown to be an \( O(\log d) \) approximation.