1 Set Cover Problem

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 $f^i_1, f^i_2, \ldots f^i_{k_i}$.
- 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 $\mathcal{F}$ of $m$ sets each a subset of $X$.
- **Defn:** a collection $C \subseteq \mathcal{F}$ 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 it is an NP optimization problem (exercise!).

It’s hard, but is it approximable?

2 An Approximation Algorithm

We will give an approximation algorithm for min set cover.

- 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 $O(|X||F|)$ in each loop iteration to compute $\delta$’s, so algo runs in time at most $O(|X|^2|F|)$.
  
• (We can do much better with some clever hashing.)

• OK, but what about an approximation ratio?

3 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 $F$.

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

**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

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

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

Final note: SET-COVER is known not to be approximable to within better than $c \log n$, for some $c > 0$, unless P=NP (Raz and Safra 1997).
4 How Good Can Approximation Ratios Get?

How “hard” are hard problems?

- All approx ratios we have seen so far are at least some constant.
- Wouldn’t it be nice if we could improve the ratio by working harder?
- For some (but not all) NP optimization problems, this is possible.
- Let $P$ be an NP optimization problem.
- Let $A(x, \epsilon)$ be an algorithm for $P$ that takes in an instance $x$ of $P$ and an extra parameter $\epsilon > 0$.
- Let $v^*$ be value of optimal solution to $x$, and let $v_\epsilon$ be value of solution produced by calling $A(x, \epsilon)$.

**Defn:** $A$ is called a *polynomial-time approximation scheme (PTAS)* for $P$ if

1. $A$ runs in time polynomial in $|x|$.
2. $\frac{v^*}{v_\epsilon} \leq 1 + \epsilon$ (for maximization), or
3. $\frac{v_\epsilon}{v^*} \leq 1 + \epsilon$ (for minimization).

- Intuitively, the smaller we make $\epsilon$, the better our approximation.
- No guarantee that $A$’s time doesn’t scale exponentially, or even worse, with decreasing $\epsilon$!!!

**Defn:** $A$ is called a *fully polynomial-time approximation scheme (FPTAS)* for $P$ if

1. $A$ is a PTAS for $P$.
2. $A$ runs in time polynomial in both $|x|$ and $\frac{1}{\epsilon}$.

- An FPTAS doesn’t blow up as $\epsilon$ gets smaller.

Next up, we give an FPTAS for the 0-1 knapsack problem!

5 Knapsack the Hard Way

We begin with following *pseudopolynomial* DP algorithm to solve 0-1 knapsack.

- Let $(S, W)$ be an instance of 0-1 knapsack.
- Item $i$ in $S$ has weight $w_i$ and value $v_i$.
- **Question:** what is weight of lightest subset of items with total value = $v$?
- **First choice:** opt soln either includes last item, or it does not.
- **Inductive Structure:**
  - If last item chosen, we seek lightest subset of items $1 \ldots n - 1$ with total value $v - v_n$. 
Else, we seek lightest subset of items 1...n − 1 with total value v.

**Optimal Substructure:** Let \( w' \) be weight of opt soln to subproblem.

- If last item is used, total weight is \( w' + w_n \).
- If last item is not used, total weight is \( w' \).

Let \( W_{i,v} \) be weight of lightest subset of items 1...i with value \( v \).

Following recurrence computes \( W_{i,v} \):

\[
W_{i,v} = \min (W_{i-1,v}, W_{i-1,v-v_i} + w_i)
\]

**Base cases:** \( W_{0,*} = \infty \) except for \( W_{0,0} = 0 \) \( W_{*,0} = 0 \)

**Subproblem ordering:** by increasing \( i \), and by increasing \( v \) within each \( i \).

So far, so good. But how does this help solve knapsack?

**Obs 1:** Let \( V = \max_{i=1}^n v_i \); then the maximum possible knapsack value is \( \leq nV \). Hence, we can compute \( W_{i,v} \) for \( 0 \leq i \leq n \) and \( 0 \leq v \leq nV \) in time \( O(n^2V) \).

**Obs 2:** For each possible \( v \), looking at \( W_{n,v} \) tells us whether any subset of items 1...n achieves value \( v \) with total weight \( < W \).

Hence, in additional time \( O(nV) \), we can find knapsack with total weight \( \leq W \) and maximum value!

Call this algorithm VBK, for “value-bound knapsack”.

VBK runs in time \( O(n^2V) \), which is pseudopolynomial in item values, but not in weights or capacity!

(Note that we can backtrack as usual, in time \( O(n) \), to reconstruct the lightest knapsack for a given \( v \).)

### 6 And Now, A Sneaky Trick

We will use VBK to build an FPTAS for 0-1 knapsack.

- Algorithm FP-KNAPSACK(\( S, W, \epsilon \)) runs as follows on input with \( n \) items.
  - Let \( V = \max_{i=1}^n v_i \) as above.
  - Set
    \[
    K = \frac{V}{n(\frac{1}{\epsilon} + 1)}
    \]
  - Form a new knapsack instance \((S', W)\) from \((S, W)\) by replacing value \( v_i \) of every item \( i \) with
    \[
    v_i' = \lfloor v_i/K \rfloor.
    \]
• (Note: as $\epsilon \to 0$, $K$ gets very small for any fixed $n$; if $K \leq 1$, we might as well use full VBK without any rounding. However, for any fixed $\epsilon$, you will eventually round given a large enough $n$.)

• Run VBK on $(S', W)$ and find optimal item set $\pi$.

• Return $\pi$.

Is this algorithm any good?

• Firstly, FP-KNAPSACK returns a feasible knapsack, since modified instance has same item weights and same capacity as original.

• Secondly, observe that largest item value in $(S', W)$ is now only $V/K$.

• Hence, running time of VBK, which dominates time for FP-KNAPSACK, is

$$O(n^2V/K) = O\left(n^3\left(\frac{1}{\epsilon} + 1\right)\right)$$

which is polynomial in both $n$ and $1/\epsilon$.

So far so good, but what about approximation ratio?

• **Claim**: FP-KNAPSACK yields a $1 + \epsilon$ approximation.

• **Pf**: Let $\pi^*$ be optimal soln to $(S, W)$, and let $\pi_\epsilon$ be solution found by algorithm with param $\epsilon$.

• Let $v^*$ and $v_\epsilon$ be values of $\pi^*$, $\pi_\epsilon$ with respect to $(S, W)$.

• Finally, let $v'$ be value of $\pi_\epsilon$ w/r to scaled problem $(S', W)$.

• We first show that, if $K$ is scaling factor computed by algorithm, then

$$v^* - Kv' \leq Kn.$$  

• Suppose not. If items in $\pi^*$ have values $\{v_1 \ldots v_t\}$ in $(S, W)$, then these items in $(S', W)$ have values $\{[v_1/K] \ldots [v_t/K]\}$.

• Total value of $\pi^*$ in $(S', W)$ is therefore

$$\sum_{i=1}^{t} \left[\frac{v_i}{K}\right] \geq \sum_{i=1}^{t} (\frac{v_i}{K} - 1)$$

$$= \frac{v^*}{K} - t$$

$$\geq \frac{v^*}{K} - n.$$  

• But by our assumption, $v^* - Kv' > Kn$, which implies

$$\frac{v^*}{K} - n > v',$$

meaning that the value of $\pi^*$ in $(S', W)$ is better than the optimum $\pi_\epsilon$ found by VBK.
• This is a contradiction! (End of subclaim)

• Now $v_e \geq Kn'$, since each elt in $\pi_\epsilon$ had its value divided by $K$ and then had the floor taken.

• Conclude that

$$v^* - v_\epsilon \leq Kn,$$

and so

$$\frac{v^*}{v_\epsilon} \leq \frac{v_\epsilon + Kn}{v_\epsilon},$$

$$= 1 + \frac{Kn}{v_\epsilon},$$

$$\leq 1 + \frac{Kn}{v^* - Kn}.$$

• Finally, observe that $v^* \geq V$, since we can always create a knapsack containing the single most valuable item.

• Hence,

$$\frac{v^*}{v_\epsilon} \leq 1 + \frac{Kn}{V - Kn}$$

$$\leq 1 + \epsilon.$$