1 Overview

In the last lecture we talked about set cover:

- Sets $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$.
- $S$ has cost $c_S$.
- Goal: Cover all of $[n]$ with minimal cost.

One algorithm is the greedy algorithm: while there exists an uncovered element, pick $S$ minimizing $\# \text{ new elements covered by } S$.

In this lecture we

- Finish study approximation algorithms via dual fitting.
- Learn about LP Integrality Gaps
- Define poly-time approximation schemes (PTAS, FPTAS, FPRAS)

2 Approximation algorithms via dual fitting

2.1 Greedy algorithm for set cover

**Theorem 1.** The greedy algorithm is an $O(\log n)$ approximation.

The unweighted case is proven in [1] and [2]. The weighted case is proven in [3].

**Proof.** We consider the following LP relaxation for set cover.

- **Primal:** minimize $\sum_S a_S x_S$ with the constraints $\forall i \in [n], \sum_{i \in S} X_S \geq 1$, and $\forall S, x_S \geq 0$.
- **Dual:** maximize $\sum_{e=1}^n$, with the constraints $\forall S, \sum_{e \in S} y_e \leq c_S$ and $\forall e, y_e \geq 0$.

We will build the primal solution, and show how we can maintain the dual solution. When we take set $S$ into our cover:

1. Set $x_S \leftarrow 1$ in primal.
2. For each newly covered element $e$, set $y_e = \frac{c_S}{\# \text{ elements newly covered by } S}$.

We see that $\text{cost}_P(x) = \text{cost}(y)$. It is clear that this procedure makes $x$ feasible.

Claim 2. $\frac{y}{\Pi_c}$ is feasible.

Proof. We need to show that $\forall S$, $\sum_{e \in S} y_e \leq c_S H_n$. Let us order the elements of $S$ by when they are first covered, $e_1, e_2, \ldots, e_k$ (where $|S| = k$). Note that $e_i$ is not necessarily covered by $S$. Right before $e_i$ was covered, we could have chosen $S$ at price $\frac{c_S}{k-i+1}$, so that $y_{e_i} \leq \frac{c_S}{k-i+1}$. Hence,

$$\sum_{e \in S} y_e \leq c_S \sum_{i=1}^{k} \frac{1}{i} = c_S H_k \leq c_S H_n.$$ 

Hence, the greedy algorithm gives a log $n$ approximation.

2.2 Vertex cover

We now consider the vertex cover problem. The input is an undirected graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. We want to pick the minimal $S \subseteq V$ such that each edge $e \in E$ is incident upon at least one vertex of $S$. Note that vertex cover is a special case of set cover, where each vertex corresponds to a set, and each edge to an element in the universe.

There is a greedy algorithm for vertex cover. Namely, while there is an uncovered edge $e = (u, v)$, we add both $u$ and $v$ to the vertex cover.

Claim 3. The greedy algorithm gives a 2-approximation.

Proof. There is a simple proof, as detailed in the CS 124 notes. We will present another proof using primal-dual approach. We consider the following LP relaxation:

- **Primal**: Minimize $\sum_{v=1}^{n} x_v$ where $\forall e = (u, v)$, $x_u + x_v \geq 1$ and $x \geq 0$.
- **Dual**: Maximize $\sum_{e \in E} y_e$ where $\forall v \in V$, $\sum_{v \in e} y_e \leq 1$ and $y \geq 0$.

We again use dual fitting. When greedy covers an edge $e = (u, v) \in E$, set $x_u \leftarrow 1$ and $x_v \leftarrow 1$, and set $y_e \leftarrow 1$. These give feasible solutions. Moreover, primal cost is at most twice the dual cost, so this is a 2-approximation.

3 LP Integrality Gaps

The question we want to answer is, what is the limit of using a given LP relaxation?

In our proofs, we achieve $\text{cost} \leq \alpha \text{OPT}(LP)$. But if there is some gap $\beta \text{OPT}_{LP} \leq \text{OPT}(IP)$ for some particular instance, then we cannot hope to achieve $\alpha < \beta$ via LP relaxation. We call $\beta$ the integrality gap.
Example 4. Consider vertex cover. We had the constraint that $\forall u, x_u \geq 0$. This should have been $x_u = \{0, 1\}$, but we relaxed it to $x_u \geq 0$. This can cause some problems. For instance, consider the complete graph on $n$-vertices, $K_n$. Then, we can assign $x_u \leftarrow \frac{1}{2}$, and in fact, this is an optimal solution. Hence, $\text{OPT}(LP) = \frac{n}{2}$. However, with integer programming, $\text{OPT}(IP) = n - 1$, since we must take all but 1 vertex. This gives an integrality gap of 2.

Example 5. Consider set cover. We define $n = 2^q - 1$ for some $q > 0$. The elements of the universe are in correspondence with elements of $\mathbb{F}_2^q \setminus \{0\}$. Then, sets are also in correspondence with elements of $\mathbb{F}_2^q$: if $\alpha \in \mathbb{F}_2^q$, $S_\alpha = \{e : \langle \alpha, e \rangle = 1\}$.

First consider the fractional solution. Each element is contained in exactly half the sets. Hence, we can take $x_u = \frac{2}{m}$ for all $m$ (i.e., $\frac{1}{\# \text{ sets containing } e}$), so $\text{OPT}(LP) \leq 2$.

For the integer solution, suppose we have a solution with $q - 1$ sets, $S_{\alpha_1}, \ldots, S_{\alpha_{q - 1}}$. Then, $\bigcap_{i=1}^{q-1} S_{\alpha_{q-1}} = \{0\}$, a dimension zero vector space. But
\[
\bigcap_{i=1}^{q-1} S_{\alpha_{q-1}} = \{e : \forall i \in 1, \ldots, q - 1, \langle \alpha_i, e \rangle = 0\}
\]
has codimension at most $q - 1$, so it has dimension at least 1. We conclude that the gap is $q/2 = O(\log n)$.

Remark 6. The integrality problem shows that the relaxation we considered cannot achieve a certain approximation gap. A stronger statement would be to show that unless $\mathbf{P} = \mathbf{NP}$, we cannot achieve a better approximation gap. These theorems typically use the PCP theorem.

4 Polynomial time approximation schemes

We consider minimization problems.

Definition 7. A problem $m$ admits a PTAS (polynomial time approximation scheme) if $\forall \epsilon \in (0, 1)$ fixed, and for all problem size $n$, we can achieve a $(1 + \epsilon)$-approximation in time $O(n^{f(1/\epsilon)})$.

Definition 8. A problem $m$ admits a FPTAS (fully polynomial time approximation scheme) if $\forall \epsilon \in (0, 1)$ fixed, and for all problem size $n$, we can achieve a $(1 + \epsilon)$-approximation in time $O\left(poly\left(\frac{n^\epsilon}{\epsilon}\right)\right)$.

Definition 9. A problem $m$ admits a FPRAS (fully polynomial time randomized approximation scheme) if $\forall \epsilon \in (0, 1)$ fixed, and for all problem size $n$, we can achieve a $(1 + \epsilon)$-approximation in time $O\left(poly\left(\frac{n^\epsilon}{\epsilon}\right)\right)$ with probability at least $\frac{2}{3}$.

Remark 10. By running a FPRAS multiple times and taking medians, we can reduce the error probability to be arbitrarily small.

We will provide a PTAS/FPTAS for knapsack and a FPRAS for counting solutions to DNF [4].

4.1 Knapsack Problem

We have a knapsack with capacity $W$ and given items with weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$. We want to pack our knapsack to maximize the total value $\sum_{i=1}^{n} v_i x_i$ with constraints $\sum_{i=1}^{n} w_i x_i \leq W$. 
W with \( x_i \in \{0, 1\} \). This problem is \( \text{NP} \)-hard. There are dynamic programming solutions for the knapsack problem. For example, there is a \( O(nW) \) algorithm. We set
\[
f(i, b) = \begin{cases} 
0 & \text{if } i = 0 \\
\max(f(i - 1, b), v_i + f(i - 1, b - w_i)) & \text{if } b - w_i \geq 0 
\end{cases}
\]
This does not violate the \( \text{NP} \)-hardness result, since \( W \) takes only \( \log W \) bits to specify, so the algorithm is exponential in the size of the problem.

Another dynamic programming approach gives time \( O(nV) \), where \( V = \sum_{i=1}^{n} v_i \). Namely, we define
\[
f(i, p) = \text{min weight to get value exactly } v \text{ using only items from } \{1, \ldots, i\}.
\]
Suppose we relax the last constraint to \( 0 \leq x_i \leq 1 \). Then, we would sort the items in decreasing order by \( \frac{v_i}{w_i} \) and fill knapsack in this order (put as much of item \( i \) as you can before starting to pack item \( i + 1 \)). This algorithm can be bad, when, for example, \( v_1 = 1 + \epsilon, w_1 = 1, v_w = W, w_2 = W \). This is only a \( W \)-approximation.

But let us now consider a modified greedy algorithm that avoid this bad case. Namely, we take either the output of greedy algorithm or we take the most valuable item, depending on which has greater value.

**Claim 11.** The above algorithm is a 2-approximation.

**Proof.**

**Lemma 12.** Suppose the greedy algorithm takes items
\[
\frac{v_1}{w_1} \geq \frac{v_w}{w_2} \geq \cdots \geq \frac{v_{k-1}}{w_{k-1}}
\]
and we have no room for the \( k \)-th item. Then, \( \sum_{i=1}^{k} v_i \geq \text{OPT} \).

**Proof.** The sum \( \sum_{i=1}^{k} v_i \) is even larger than the fractional knapsack’s optimal solution, which is at least the optimum of the integral knapsack problem.

Now, \( \sum_{i=1}^{k} v_i = (v_1 + \ldots + v_{k-1}) + v_k \geq \text{OPT} \) by the lemma. Hence, one of \( v_1 + \ldots + v_{k-1} \) or \( v_k \) is at least \( \frac{\text{OPT}}{2} \) as desired.

**Observation 13.** If no item has value \( > \epsilon \text{OPT} \), then the greedy algorithm achieves \( \geq (1 - \epsilon)\text{OPT} \).

**Observation 14.** At most \( \left\lfloor \frac{1}{\epsilon} \right\rfloor \) items in optimal solution have value \( > \epsilon \text{OPT} \).

Now, we present a PTAS algorithm for the knapsack problem. We first guess the set \( S \) consisting of the elements in the optimal solution of value at least \( \epsilon \text{OPT} \). Then, \( |S| \leq \left\lfloor \frac{1}{\epsilon} \right\rfloor \). We then remove all items remaining which have value bigger than anything in \( S \). Finally, we run the greedy algorithm on what is left, and our knapsack will contain \( S \) and whatever the greedy algorithm chooses. We can do this for all \( O(n^{1/\epsilon}) \) possible subsets of size at most \( \frac{1}{\epsilon} \), so the running time is \( O(n^{1/\epsilon} \text{poly}(n)) \).

**Claim 15.** The profit achieved by this scheme is \( \geq (1 - \epsilon)\text{OPT} \).
Proof. Suppose we guess $S$ correctly. Then, $\text{OPT} = v(S) + \text{OPT}'$, where $v(S)$ is the value of packing $S$ and $\text{OPT}'$ is the best packing of items of value at most $\epsilon \text{OPT}$. Our scheme then achieve $v(S) + \text{greedy on rest}$.

Now let us analyze how the greedy algorithm performs. We know that $v_k \leq \epsilon \text{OPT}$ (recall that $v_k$ is the value of the $k$-th and last element that the fractional greedy algorithm tries to include). Also, we know that the greedy output plus $v_k$ is at least $\text{OPT}'$ by Lemma 12. Hence, greedy achieves a profit of at least $\text{OPT}' - \epsilon \text{OPT}$, so our scheme achieves at least $(1 - \epsilon) \text{OPT}$. 

References


