CMPUT 498/501: Advanced Algorithms **Fall 2025**

Lecture 16 (Oct 28, 2025): Set Cover, Knapsack, and Bin Packing

Lecturer: Mohammad R. Salavatipour *Scribe: Churong Yu, Keven Qiu*

16.1 Set Cover & Max Coverage

The *Set Cover* problem is one of the most fundamental NP-hard optimization problems. We are given a universe $U = \{e_1, \ldots, e_n\}$ of n elements and a collection of subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ where each $S_i \subseteq U$ has an associated cost $c_i > 0$. The goal is to choose a subcollection $\mathcal{C} \subseteq \mathcal{S}$ such that $\bigcup_{S_i \in \mathcal{C}} S_i = U$ and $\sum_{S_i \in \mathcal{C}} c_i$ is minimized:

$$\min_{\mathcal{C} \subseteq \mathcal{S}} \sum_{S_i \in \mathcal{C}} c_i \quad \text{s.t.} \quad \bigcup_{S_i \in \mathcal{C}} S_i = U.$$

Example and connection to Vertex Cover. *Vertex Cover* is a special case of Set Cover. Each edge of the graph is an element of U, and for each vertex v, we define a set S_v containing all edges incident to v. Choosing a collection of vertices whose incident sets cover all edges is exactly a Set Cover instance.

Frequency. For each element $e \in U$, define its *frequency* $f(e)$ as the number of sets that contain it. In the Vertex Cover example, every edge belongs to exactly two sets, so the frequency is 2.

A unified greedy strategy works for both problems:

- At each step, pick the set that covers the maximum number of uncovered elements per unit cost.
- For Set Cover, continue until all elements are covered.
- For Max Coverage, stop after selecting k sets.

Theorem 1 *Greedy is a* $(\ln n + 1)$*-approximation for* Set Cover *and a*

$$1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - \frac{1}{e}$$

approximation for Max Coverage.

Intuition. The same principle minimizes the cost-effectiveness ratio $\frac{c_i}{|S_i \cap U'|}$ at each iteration. For Set Cover we use a harmonic-series charging argument; for Max Coverage we show each iteration covers at least a $\frac{1}{k}$-fraction of the remaining optimum elements.

16.1.1 Max Coverage

Problem. Given a universe $U = \{e_1, \ldots, e_n\}$ and m sets $\mathcal{S} = \{S_1, \ldots, S_m\}$, the goal is to choose at most k sets to maximize the number of covered elements:

$$\max_{|\mathcal{C}| \leq k} \Big| \bigcup_{S_i \in \mathcal{C}} S_i \Big|.$$

Analysis. Let OPT^* be an optimal solution with value OPT. Define:

$$a_i = \text{newly covered elements at iteration } i, \quad b_i = \sum_{j=1}^{i} a_j, \quad c_i = \mathrm{OPT} - b_i.$$

Initially, $a_0 = b_0 = 0$ and $c_0 = \mathrm{OPT}$.

Lemma 1 *For all $0 \le i < k$,*

$$a_{i+1} \ge \frac{c_i}{k}.$$

Proof. Consider iteration $i+1$. Up to the end of iteration i, b_i elements have been covered by the greedy algorithm, so there remain c_i elements of the optimal solution OPT^* that are still uncovered. Let Z^* denote these uncovered elements of OPT^*, so $|Z^*| \ge c_i$.

Since OPT^* uses exactly k sets to cover all elements of Z^*, by the pigeonhole principle one of these k sets must cover at least $|Z^*|/k \ge c_i/k$ elements of Z^*. Hence, there exists a set in OPT^* that can cover at least c_i/k of the still-uncovered elements.

Because the greedy algorithm always chooses the set that covers the largest number of uncovered elements, the set chosen in iteration $i+1$ must cover at least this many. Therefore,

$$a_{i+1} \ge \frac{c_i}{k}.$$

■

Lemma 2 *For all $i > 0$,*

$$c_i \le (1 - \tfrac{1}{k})^i \cdot \mathrm{OPT}.$$

Proof. Recall a_i is the number of new elements covered at iteration i, $b_i = \sum_{j=1}^{i} a_j$, and $c_i = \mathrm{OPT} - b_i$. We prove by induction on i.

Base case ($i = 0$):

$$c_0 = \mathrm{OPT} = \left(1 - \frac{1}{k}\right)^0 \cdot \mathrm{OPT}.$$

Induction step. Assume $c_i \le \left(1 - \frac{1}{k}\right)^i \cdot \mathrm{OPT}$. By Lemma 1, $a_{i+1} \ge \frac{c_i}{k}$. Hence

$$\begin{aligned} c_{i+1} &= c_i - a_{i+1} \\ &\le c_i - \frac{c_i}{k} \\ &= \left(1 - \frac{1}{k}\right) c_i \\ &\le \left(1 - \frac{1}{k}\right)\left(1 - \frac{1}{k}\right)^i \cdot \mathrm{OPT} \\ &= \left(1 - \frac{1}{k}\right)^{i+1} \cdot \mathrm{OPT}. \end{aligned}$$

This completes the induction. ■

After k iterations.

$$\begin{aligned} c_k &\le \left(1 - \frac{1}{k}\right)^k \cdot \mathrm{OPT}, \\ b_k &= \mathrm{OPT} - c_k \\ &\ge \mathrm{OPT} - \left(1 - \frac{1}{k}\right)^k \cdot \mathrm{OPT} \\ &= \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot \mathrm{OPT} \\ &\ge \left(1 - \frac{1}{e}\right) \cdot \mathrm{OPT}. \end{aligned}$$

Therefore, the Greedy algorithm achieves a $(1 - 1/e)$-approximation for Max Coverage.

Extensions. If each set has cost and total budget B, choose sets minimizing cost per new element covered; the same $(1 - 1/e)$ bound holds.

Tightness. The $(1 - 1/e)$ factor is optimal unless P = NP.

16.1.2 Set Cover

Algorithm 1 Greedy Set Cover (SC1)

$C \leftarrow \emptyset$ {covered elements}
$\mathcal{S}' \leftarrow \emptyset$ {chosen sets}
while $C \neq U$ **do**
  Choose $S_i \in \mathcal{S}$ minimizing $\alpha = \dfrac{c(S_i)}{|\, S_i \setminus C \,|}$ {cost per new element}
  $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{S_i\}$
  $N \leftarrow S_i \setminus C$ {newly covered elements (computed using the old C)}
  for each $e \in N$ **do** price$(e) \leftarrow \alpha$ **end for**
  $C \leftarrow C \cup S_i$
end while
return $\mathcal{S}'$

Each element is charged α, the cost per new element; total charge equals total cost of the greedy solution.

Lemma 3

$$\sum_{e \in U} \mathit{price}(e) \le H_n \cdot \mathrm{OPT}, \qquad H_n = 1 + \tfrac{1}{2} + \cdots + \tfrac{1}{n} \le \ln n + 1.$$

Proof. Order the elements of U in the order that they are first covered by SC1:

$$e_1, e_2, \ldots, e_n.$$

(If several elements are covered at the same step, break ties arbitrarily.)

Consider the moment just before element e_k is covered. At this time there are exactly $n - k + 1$ uncovered elements remaining. Even if the greedy algorithm has made poor choices so far, we could still cover all the remaining elements using the sets of an optimal solution at a total cost of at most OPT.

By an averaging argument, there must exist at least one of these optimal sets whose cost-effectiveness ratio (cost per newly covered element) is no more than

$$\frac{\text{OPT}}{n-k+1}.$$

Since the greedy algorithm chooses the set with the smallest available ratio, the charge (price) assigned to e_k satisfies

$$\text{price}(e_k) \;\leq\; \frac{\text{OPT}}{n-k+1}.$$

Summing over all n elements yields

$$\sum_{e \in U} \text{price}(e) \leq \sum_{k=1}^{n} \frac{\text{OPT}}{n-k+1} \;=\; \text{OPT} \cdot \sum_{k=1}^{n} \frac{1}{n-k+1} \;=\; H_n \cdot \text{OPT}.$$

■

Since SC1 charges each newly covered element exactly the per-new-element cost α of the chosen set, the total charge equals the algorithm's cost:

$$\text{cost(SC1)} = \sum_{e \in U} \text{price}(e) \leq H_n \cdot \text{OPT} \leq (\ln n + 1) \cdot \text{OPT}.$$

Tightness. This H_n factor is essentially tight: it can be shown that no $(1-\varepsilon)\ln n$-approximation for Set Cover exists for any $\varepsilon > 0$, unless P = NP.

16.2 Knapsack

The knapsack problem: Given n items where each item i has size $s_i \geq 0$, profit/value $v_i \geq 0$, and a knapsack capacity of B. We want to find a subset S of the items so that the size of S is at most B, i.e. $\sum_{i \in S} s_i \leq B$, and we want to maximize $\sum_{i \in S} v_i$.

We assume that each $s_i \leq B$, otherwise we can just throw them out since they will never go in our knapsack.

If we allow fractional items, we can solve this using greedy by sorting the items based on its value per size v_i/s_i.

$$\frac{v_1}{s_1} \geq \frac{v_2}{s_2} \geq \cdots \geq \frac{v_n}{s_n}$$

and pack items in this order. This will get you an optimal solution.

The integer knapsack is NP-hard. Our fractional approach can be arbitrarily bad by the following example: Let there be 2 items, with $s_1 = 1, s_2 = B, v_1 = 2, v_2 = B$. The greedy algorithm would get us a knapsack of value 2, but $\text{OPT} = B$.

We can modify greedy to get a simple 2-approximation. Consider

Algorithm 2 Choosing the better of two solutions

1: Pick the better of

- OPT of greedy, or
- Most profitable item.

Lemma 4 *Suppose greedy picks items* $1, \ldots, i-1$*, but cannot pick item* i*, where*

$$\frac{v_1}{s_1} \geq \frac{v_2}{s_2} \geq \cdots \geq \frac{v_{i-1}}{s_{i-1}} \geq \frac{v_i}{s_i}$$

Then

$$v_1 + v_2 + \cdots + v_{i-1} + \frac{B - (s_1 + \cdots + s_{i-1})}{s_i} v_i \geq \text{OPT}$$

Corollary 1 *At least one of* $\{v_1 + \cdots + v_{i-1}, v_i\} \geq \frac{\text{OPT}}{2}$.

Proof. By an averaging argument, since the sum of all items $1, \ldots, i$ is $\geq$ OPT by the lemma, one of the sets must have $\geq \text{OPT}/2$ value. ■

Corollary 2 *If all* $v_i \leq \varepsilon \cdot \text{OPT}$ *for all* i*, then Greedy is a* $(1-\varepsilon)$*-approximation for Knapsack.*

16.2.1 Pseudopolynomial-time for Knapsack

We can solve knapsack using dynamic programming. Let $A[i, V]$ be the size of the smallest knapsack using items $1, \ldots, i$ with total value V. Note that $V \leq n \cdot v_{\max}$.

We want to find the largest v such that $A[n, v] \leq B$. The DP recurrence is

$$A[i, v] = \min\{\underbrace{A[i-1, v]}_{\text{don't pick } i}, \underbrace{A[i-1, v-v_i] + s_i}_{\text{pick item } i}\}, \quad \forall v_i$$

If $V = \max_i v_i$, then the runtime is $O(n^2 V)$ to fill in the table, since the largest profit the table can go to is $\leq nV$. This is not polynomial time, because V could be arbitrarily large. We can scale and round to turn this into a PTAS. Instead of considering all values $0, \ldots, V$, we focus on poly-size many values by scaling.

Algorithm 3 FPTAS for Knapsack

1: Let $K = \frac{V}{n/\varepsilon}$.
2: $v_i' = \left\lfloor \frac{v_i}{K} \right\rfloor$.
3: Run DP with (s_i, v_i') to get set of items S'.
4: **return** S'

Theorem 2 *This is an FPTAS for Knapsack.*

Proof. Let S^* be an optimal solution with value OPT^*. By the definition of v_i', $Kv_i' \leq v_i \leq K(v_i' + 1)$ by the scaling of v_i'. Now

$$\text{OPT}^* = \sum_{i \in S^*} v_i \leq \sum_{i \in S^*} Kv_i' + nK$$

Note that S' is OPT under value v'_i. So we have the bound $\sum_{i\in S'} v'_i \geq \sum_{i\in S^*} v'_i$. Putting all these together,

$$\sum_{i\in S'} v_i \geq \sum_{i\in S'} Kv'_i \geq \sum_{i\in S^*} Kv'_i \geq \text{OPT}^* - nK = \text{OPT}^* - \varepsilon V \geq (1-\varepsilon)\text{OPT}^*$$

since $V \leq \text{OPT}$ in the last inequality.

So the solution found is $\geq (1-\varepsilon)\text{OPT}^*$. The runtime is $O(n^2\lfloor V/k\rfloor) = O(n^3/\varepsilon)$. ■

16.3 Bin Packing

Given a set of items $0 < s_n \leq s_{n-1} \leq \cdots \leq s_1 < 1$. We want to find a minimum number of unit-sized bins into which all items can be packed. This is equivalent to partitioning the items into k groups $G_1, \ldots, G_k$ such that $\sum_{i\in G_j} s_i \leq 1$ for all j.

Theorem 3 *There is no α-approximation for bin packing for any $\alpha < \frac{3}{2}$ unless $P = NP$.*

Proof. We can do a reduction from Partition. The partition problem is to partition the set of integers $s_1, \ldots, s_n$ into two parts of equal sum.

Suppose $\sum s_i = 2$, then a perfect packing into 2 bins is a partition. If there is an α-approximation for bin packing with $\alpha < \frac{3}{2}$, then you can decide between needing ≥ 3 bins or needing ≤ 2 bins. ■

16.3.1 Greedy Bin Packing

We show a greedy algorithm that gets within $2\text{OPT} + 1$. Start putting items into bins by trying the first bin into which you can fit the item and use a new bin if needed.

Theorem 4 *Cost of First-Fit is $\leq 2\text{OPT} + 1$.*

Proof. Let $S_I = \sum s_i$ for instance I. We know $\text{OPT} \geq S_i$.

We claim that in the solution of first-fit, at most one bin is $\leq \frac{1}{2}$ full, otherwise the items in another bin $\leq \frac{1}{2}$ full can be placed in an earlier bin of $\leq \frac{1}{2}$ full. The theorem follows. ■

Theorem 5 *(Johnson '73) If items are sorted in nonincreasing order, then First-Fit uses at most $\frac{11}{9}S_I + 4$ bins.*

First-Fit can be improved if all items are small, i.e. $\leq \varepsilon$. All bins except possibly the last are $\geq (1-\varepsilon)$ full, so

$$F.F.(I) \leq \frac{1}{1-\varepsilon}S_I + 1 \leq (1+2\varepsilon)\text{OPT} + 1$$

Lemma 5 *Let $\varepsilon > 0$ be fixed and suppose all items have size $\geq \varepsilon$ and there are at most g (g is a fixed constant) distinct item sizes, then this can be solved optimally in polynomial time.*

Proof. Note that each bin can have $m \leq \lceil 1/\varepsilon\rceil$ items. The number of different bin types is at most $g^m = R$ ($O(1)$). Since $\text{OPT} \leq n$, the number of of different feasible solutions is at most number of nonnegative solutions to $x_1 + x_2 + \cdots + x_R = n \leq n^{O(R)}$. So this case can be solved using exhaustive search. ■