Approximate Set Cover

Set Cover

- **Set Cover (Optimization version)**. Given a set U of n elements, a collection $\mathcal{S}$ of subsets of U, find the minimum number of subsets from $\mathcal{S}$ whose union covers U.

$$U = \{\ 1, 2, 3, 4, 5, 6, 7\ \}$$

$$S_a = \{\ 3, 7\ \} \qquad S_b = \{\ 2, 4\ \}$$

$$S_c = \{\ 3, 4, 5, 6\ \} \qquad S_d = \{\ 5\ \}$$

$$S_e = \{\ 1\ \} \qquad S_f = \{\ 1, 2, 6, 7\ \}$$

$$k = 2$$

a set cover instance

Greedy Algorithm

- Greedily pick sets that maximize coverage until done

- Greedy Cover($\mathcal{U}, \mathcal{S}$):
    - Initially all elements of $\mathcal{U}$ are marked uncovered
    - $C \leftarrow \emptyset$ (Initialize cover)
    - While there is an uncovered element in $\mathcal{U}$
        - Pick the set S_m from $\mathcal{S} \setminus C$ that maximizes the number of uncovered elements
        - $C \leftarrow C \cup \{S_m\}$
        - Mark elements of S_m as covered

Analyzing Greedy

- **Claim**. Greedy set cover is a $\ln n$-approximation, that is, greedy uses at most $k(\ln n + 1)$ sets where k is the size of the optimal set cover.

Main observations behind proof:

- If there exists k subsets whose union covers all n elements, then there exists a subset that covers $1/k$ fraction of elements
- Greedy always picks subsets that maximize remaining uncovered elements
- In each iteration, greedy's choice must cover at least $1/k$ fraction of the remaining elements
- Such a subset must always exist since the remaining elements can also be covered by at most k subsets

Analyzing Greedy

- **Claim.** Greedy set cover is a $(\ln n+1)$-approximation—greedy uses at most $k(\ln n + 1)$ sets where k is the size of the optimal set cover.
- **Proof.**
- Let E_t be the set of elements still uncovered after tth iteration.
- The optimal solution covers E_t with no more than k sets
- Greedy always picks the subset that covers most of E_t in step $t + 1$
- Selected subset must cover at least $|E_t|/k$ elements of E_t
- Thus $|E_{t+1}| \leq |E_t| \left(1 - 1/k\right)$ and as $E_0 = n$, inductively we have $|E_t| \leq n(1 - 1/k)^t$
- When $|E_t| < 1$, we are done

Analyzing Greedy

- **Claim.** Greedy set cover is a $(\ln n + 1)$-approximation—greedy uses at most $k(\ln n + 1)$ sets where k is the size of the optimal set cover.
- **Proof.** (Cont.)
- $|E_t| \leq n(1 - 1/k)^t$
- When $|E_t| \leq 1$, we are done
- Setting $t = k \ln n$, we get $|E_t| = n\left(1 - \frac{1}{k}\right)^{k \ln n} \leq n \cdot \frac{1}{n} = 1$
- Thus, greedy finishes in $k \ln n + 1$ steps where k is the optimal-set cover size, so it uses at most $k \ln n + 1$ sets.
- We can tighten the analysis by considering when there are at most k uncovered elements

$\left(1 - \frac{1}{x}\right)^x \leq \frac{1}{e}$ for $x > 0$

Analyzing Greedy

- **Claim**. If the optimal set cover has size k then the greedy set cover has size at most $k(1 + \ln(n/k))$.
- **Proof**. (Cont.)
- $|E_t| \leq n(1 - 1/k)^t$
- When $|E_t| \leq k$, we are done
- Setting $t = k\ln(n/k)$, we get $|E_t| = n\left(1 - \frac{1}{k}\right)^{k\ln(n/k)}$

$$\leq n \cdot k/n = k$$

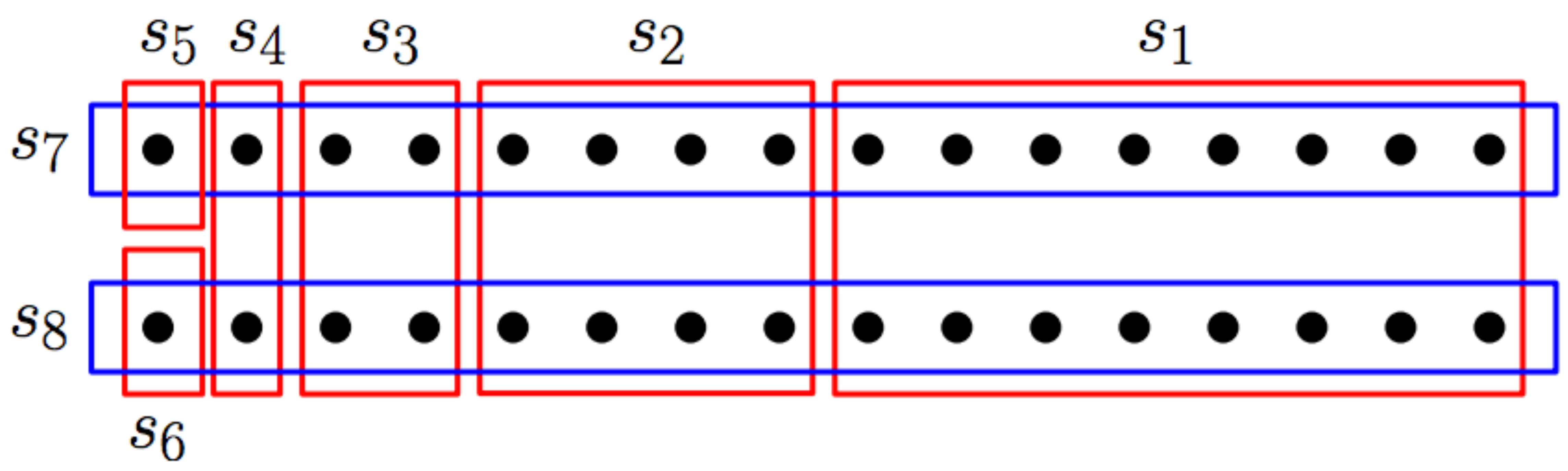
- Greedy needs at most k more sets to cover remaining k elements and thus uses at most $k + k\ln(n/k)$ sets in total.

Special Case

- We can do slightly better for special input
- **Claim**. If the maximum size of any subset in $\mathcal{S}$ is B then the greedy algorithm is $(\ln B + 1)$-approximation
- **Proof**.
- If each subset has almost B elements and the optimal set cover has k subsets then $k \geq n/B$
- Substituting $n/k \leq B$ shows that greedy is $(\ln B + 1)$ approximation

Tight Approximation

- Is the greedy approximation tight?
- Essentially, yes
- Consider the following example with $n = 2^5$ elements
- s_1, s_7, s_8 each have $n/2$ elements but greedy can pick the worst: s_1
- Example can be extended to any $n = 2^k$ where optimal cost is 2 and greedy cost is $O(\ln n)$



Approximating Vertex Cover

- We know that vertex cover reduces to set cover
- $\mathscr{U} = E$ and $\mathscr{S} = \{S_v \mid v \in V\}$ where
  $S_v = \{e \in E \mid e \text{ incident to } v\}$
- Thus the greedy approximation algorithm for set cover also gives an approximation algorithm for vertex cover
- Greedy picks vertices that cover maximum number of edges (i.e., vertices with max degrees w.r.t. uncovered edges)
- Greedy vertex cover is thus a $(\ln \Delta + 1)$ approximation where Δ is maximum degree of any vertex
- The seemingly stupider algorithm on assignment 9 is better than greedy—2-approximation is best known
- Finding a $(2 - \varepsilon)$-approximation of VC is a big open problem!

Approximate Weighted Set Cover

Weighted Set Cover

- In the weighted-version of the set cover problem, each subset $S_i \in \mathcal{S}$ has a weight w_i associated with it
- The goal is to find the a collection of subsets $C = \{S_1, \ldots, S_k\}$ such that they cover $\mathcal{U}$ and $\sum_{S_i \in C} w(S_i)$ is minimized
- We extend the greedy algorithm to the weighted case
- What should we be greedy about?

Weighted Case: Greedy

- In the weighted-version of the set cover problem, each subset $S_i \in \mathcal{S}$ has a weight w_i associated with it
- Each potential set that can be added to the solution has some "benefit" (elements it covers) and some "cost" (its weight)
- We can be greedy in terms of the cost/benefit or the "amortized cost" of choosing set S_i
- Greedy algorithm.
    - Begin with an empty cover and continue until all elements covered
    - In each iteration choose the set S_i that minimizes amortized cost w_i/e, where e is the # of new elements covered by S_i

Weighted Case: Greedy

- How good is the greedy strategy for the weighted case?
- **Claim.** Greedy is a $O(\log n)$-approximation for weighted set cover.
- We prove this by proving a **different claim**:
  for any subset $S_i \in \mathcal{S}$, the greedy algorithm covers the elements of S_i with a cost no greater than $O(\log n)$ times w_i (the cost of choosing S_i itself)
- Thus, no matter what collection of subsets $O = \{S_1, \ldots, S_k\}$ the optimal solution picks, the greedy algorithm covers them at cost $O(\log n)$ times $\sum_{S_j \in O} w(S_j)$ (the cost of the optimal solution)
- This would complete the proof that greedy is a $O(\log n)$-approximation

Weighted Greedy: Analysis

- **Claim**. For any subset $S_i \in \mathcal{S}$, the greedy algorithm covers the elements of S_i with a cost no greater than $O(\log n)$ times w_i (the cost of choosing S_i itself)
- **Proof**. Order the elements of $S_i = \{a_1, a_2, \ldots, a_d\}$ in the order in which they were covered by the greedy algorithm (if more than one are covered at the same time, break ties arbitrarily)
- Consider the time the element a_d is covered: the available sets to cover a_d include S_i itself
- Covering a_d with S_i would incur an amortized cost of w_i or less (if a_d is the only new element covered by S_i or less otherwise)
- Greedy picks the set with least amortized cost so its cost is at most w_i to cover a_d

Weighted Greedy: Analysis

- **Claim**. For any subset $S_i \in \mathcal{S}$, the greedy algorithm covers the elements of S_i with a cost no greater than $O(\log n)$ times w_i (the cost of choosing S_i itself)
- **Proof**.
  Now look at when a_{d-1} is covered, at this time, it is possible to select S_i and cover both a_{d-1} and a_d incurring an amortized cost of $w_i/2$ or less (if more elements are covered)
- Greedy picks the set with least amortized cost so its cost to cover a_{d-1} is at most $w_i/2$
- Similarly a_{d-2} is covered at amortized cost at most $w_i/3$. Each element a_j incurs an amortized cost at most$w_i/(d-j+1)$ up until a_1 which is covered at amortized cost w_i/d

Weighted Set Cover

- **Claim**. For any subset $S_i \in \mathcal{S}$, the greedy algorithm covers the elements of S_i with a cost no greater than $O(\log n)$ times w_i (the cost of choosing S_i itself)
- **Proof**.
- Each element a_j incurs an amortized cost at most $w_i/(d - j + 1)$ up until a_1 which is covered at amortized cost w_i/d
- Thus the greedy algorithm covers all elements of S_i at an amortized cost of

$$w_i \left(\sum_{j=1}^{d} \frac{1}{n - j + 1} \right) = w_i \cdot O(\log d) = w_i \cdot O(\log n)$$

- This analysis can be shown to be tight as well

Wrapping Up Approximations

- Set Cover. Can we do better than $O(\log n)$?
- [Raz & Safra 1997]. There exists a constant $c > 0$, there is no polynomial-time $c \ln n$-approximation algorithm, unless P = NP.'
- **Approximation schemes**.
  - Let X be a minimization problem, an approximation scheme for X is a family of $(1 + \varepsilon)$-approximation algorithms for $0 < \varepsilon < 1$
  - If the running time is polynomial in n (not in ε) we call it a polynomial-time approximation scheme (PTAS)
  - If the running time is polynomial in both n and ε, we call it a fully polynomial-time approximation scheme (FPTAS)
  - FPTAS for NP hard problems such Knapsack and Subset-Sum

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)