

# Approximating Set Cover

# Admin

- Assignment 10 (Practical Problems) out; do not have to submit
- Review hours next week:
  - 2-3.30 pm Monday, **7-9 pm Tuesday**, 1.30-3 pm Wednesday
  - Ask questions in general or about practice problems
- 24-hour final will be available on Gradescope from Thursday May 20 8.30 am and must be submitted by May 28, 8.30 pm
- Open book, open notes, but nothing else (not allowed to google terms)

# CS Colloquium Today: 3.15 pm

Rediet Abebe, UC Berkeley

## “Modeling the Impact of Shocks on Poverty”

The dynamic nature of poverty presents a challenge in designing effective assistance policies. A significant gap in our understanding of poverty is related to the role of income shocks in triggering or perpetuating cycles of poverty. Such shocks can constitute unexpected expenses — such as a medical bill or a parking ticket — or an interruption to one’s income flow. Shocks have recently garnered increased public attention, in part due to prevalent evictions and food insecurity during the COVID-19 pandemic. However, shocks do not play a corresponding central role in the design and evaluation of poverty-alleviation programs.

To bridge this gap, we present a model of economic welfare that incorporates dynamic experiences with shocks and pose a set of algorithmic questions related to subsidy allocations. We then computationally analyze the impact of shocks on poverty using a longitudinal, survey-based dataset. We reveal insights about the multi-faceted and dynamic nature of shocks and poverty. We discuss how these insights can inform the design of poverty-alleviation programs and highlight directions at this emerging interface of algorithms, economics, and social work.

**Incentive:** 2 pt **extra credit** on HW 9 for attending



# 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  $U$ .

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

$$S_a = \{ 3, 7 \}$$

$$S_b = \{ 2, 4 \}$$

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

$$S_d = \{ 5 \}$$

$$S_e = \{ 1 \}$$

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

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

# 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

$$U = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$$

$$S_a = \{ 1, 2, 3 \}$$

$$S_b = \{ 4, 5 \}$$

$$S_c = \{ 3, 5, 7 \}$$

$$S_d = \{ 6 \}$$

$$S_e = \{ 1, 8 \}$$

$$S_f = \{ 2, 4, 6 \}$$

# 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

$$U = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$$

$$S_a = \{ 1, 2, 3 \}$$

$$S_b = \{ 4, 5 \}$$

$$S_c = \{ 3, 5, 7 \}$$

$$S_d = \{ 6 \}$$

$$S_e = \{ 1, 8 \}$$

$$S_f = \{ 2, 4, 6 \}$$

# Analyzing Greedy

- **Claim.** Greedy set cover is a  $\ln n$ -approximation, that is, greedy uses at most  $k \ln n$  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  $\geq 1/k$  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  $\ln n$ -approximation, that is, greedy uses at most  $k \ln n$  sets where  $k$  is the size of the optimal set cover.
- **Proof.**
- Let  $E_t$  be the set of elements still uncovered after  $t$  th 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  $|E_{t+1}| \leq |E_t| (1 - 1/k)$  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  $\ln n$ -approximation—greedy uses at most  $k \ln n$  sets where  $k$  is the size of the optimal set cover.

- **Proof.** (Cont.)

- $|E_t| \leq n(1 - 1/k)^t$

- When  $|E_t| < 1$ , we are done

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

- Setting  $t = k \ln n$ , we get  $|E_t| = n \left(1 - \frac{1}{k}\right)^{k \ln n} < n \cdot \frac{1}{n} = 1$

- Thus, greedy finishes in  $k \ln n$  steps where  $k$  is the optimal-set cover size, so it uses at most  $k \ln n$  sets.
- We can tighten the analysis by considering when there are at most  $k$  uncovered elements

# Analyzing Greedy

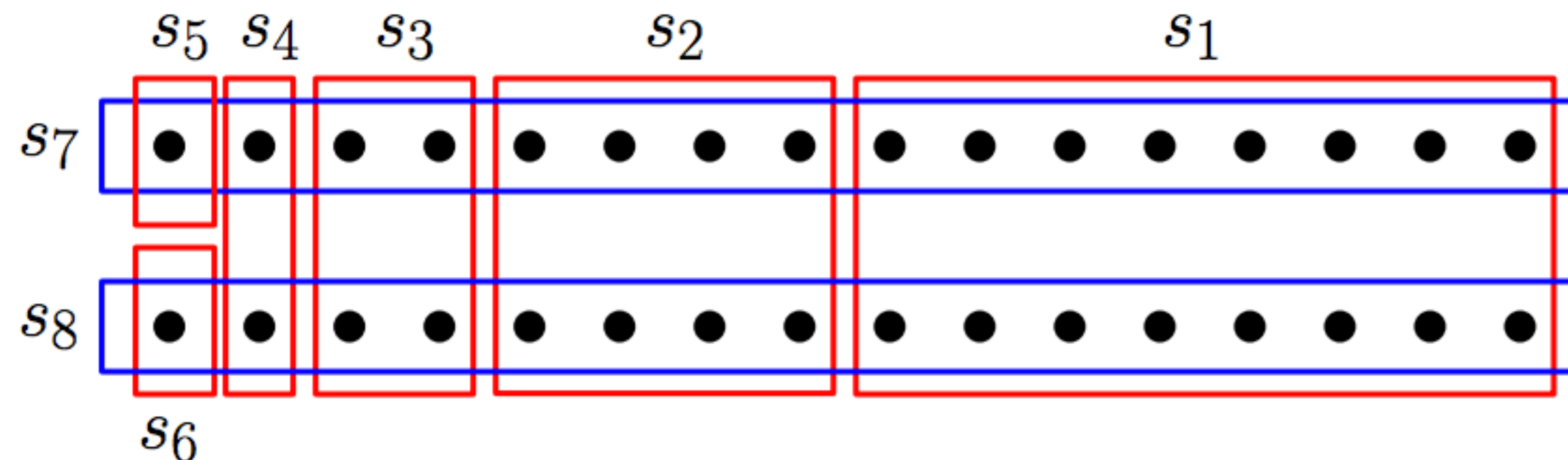
- **Claim.** If the optimal set cover has size  $k$  then the greedy set cover has size at most  $k(1 + \ln(n/k))$ .
- **Proof.** (Cont.)
- $|E_t| \leq n(1 - 1/k)^t$
- When  $|E_t| \leq k$ , we finish after selecting at most  $k$  more sets
- Setting  $t = k \ln(n/k)$ , we get  $|E_t| = n \left(1 - \frac{1}{k}\right)^{k \ln(n/k)} \leq n \cdot k/n = k$
- Greedy 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, then  $k$  must be at least
    - $k \geq n/B$
- Substituting  $n/k \leq B$  shows that greedy is  $(\ln B + 1)$  approximation

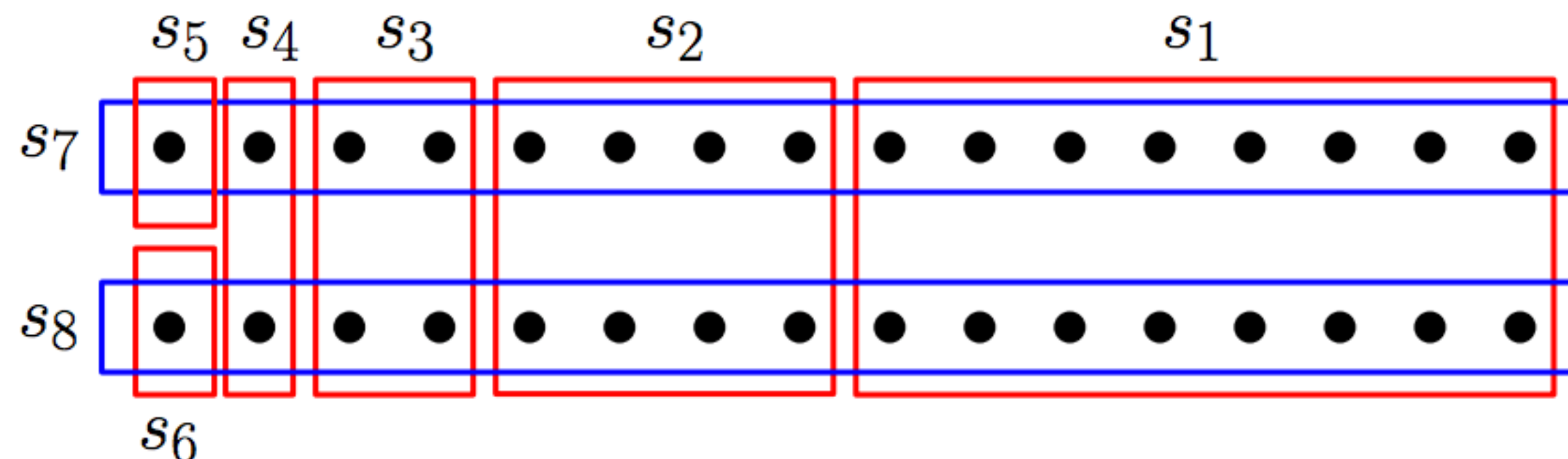
# Tight Approximation

- Is the greedy approximation tight?
  - Essentially, yes
- Consider the following example with  $n = 2^5$  elements
- $s_1$  has 16 elements, but  $s_7$  and  $s_8$  each have 15, so greedy chooses  $s_1$



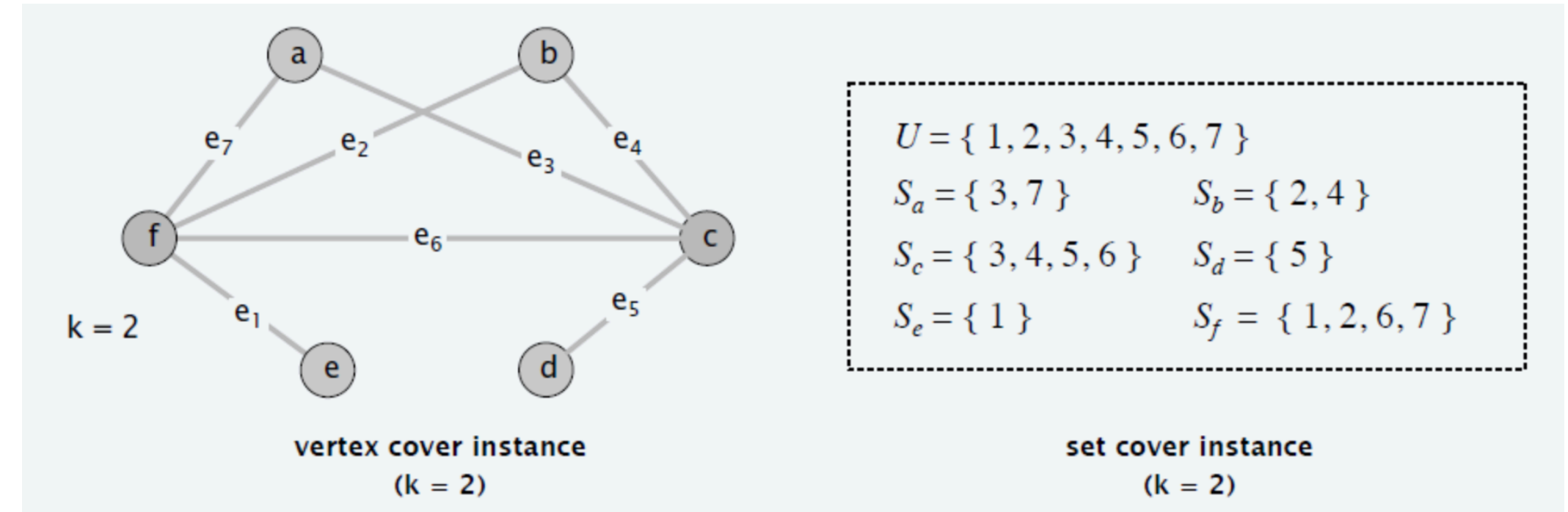
# Tight Approximation

- Consider the following example with  $n = 2^5$  elements
- Then,  $s_2$  has 8 elements, but  $s_7$  and  $s_8$  each have 7, so greedy chooses  $s_2$  ...
- This happens  $\log_2(n/2) = \log_2 16 = 4$  times
- Only two sets  $s_7, s_8$  are enough so, approximation ratio  $(\log_2 \frac{n}{2})/2$



# Approximating Vertex Cover

- We know that vertex cover reduces to set cover
- $\mathcal{U} = E$  and  $\mathcal{S} = \{S_v \mid v \in V\}$  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  $\Delta$  is maximum degree of any vertex
- The **seemingly stupider algorithm on assignment 9 is better than greedy**—2-approximation is best known
- Finding a  $(2 - \epsilon)$ -approximation of VC is a big open problem!
  - Can't be done under “unique games conjecture”



This won't work for all reductions

# Set Cover Summary

- Weighted set cover:  $1 + \ln$  approximation
- [Raz & Safra 1997]. There exists a constant  $c > 0$ , there is no polynomial-time  $c \ln n$ -approximation algorithm, unless  $P = NP$ .
- [Dinur & Steurer 2014] No polynomial time  $(1 - \epsilon) \ln n$  approximation for any constant  $\epsilon > 0$  unless  $P = NP$



# 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 a collection of subsets  $C = \{S_1, \dots, S_k\}$  such that they cover  $\mathcal{U}$  and  $\sum_{S_i \in C} w(S_i)$  is minimized

$$U = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$$

$$\$5 \ S_a = \{ 1, 2, 3 \}$$

$$\$4 \ S_b = \{ 4, 5 \}$$

$$\$13 \ S_c = \{ 3, 5, 7 \}$$

$$\$3 \ S_d = \{ 6 \}$$

$$\$1 \ S_e = \{ 1, 8 \}$$

$$\$15 \ S_f = \{ 2, 4, 6 \}$$

$$k = 2$$

# Weighted Set Cover

- We extend the greedy algorithm to the weighted case
- What should we be greedy about?
  - What could happen if we pick the largest?
  - What could happen if we pick the cheapest?

$$U = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$$

$$\$5 \ S_a = \{ 1, 2, 3 \}$$

$$\$4 \ S_b = \{ 4, 5 \}$$

$$\$13 \ S_c = \{ 3, 5, 7 \}$$

$$\$3 \ S_d = \{ 6 \}$$

$$\$1 \ S_e = \{ 1, 8 \}$$

$$\$15 \ S_f = \{ 2, 4, 6 \}$$

$$k = 2$$

# 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  $S_i$ —how much are we spending per new item covered?

$U = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$					
1.67	\$5	$S_a = \{ 1, 2, 3 \}$	\$4	$S_b = \{ 4, 5 \}$	2.00
4.33	\$13	$S_c = \{ 3, 5, 7 \}$	\$3	$S_d = \{ 6 \}$	3.00
.50	\$1	$S_e = \{ 1, 8 \}$	\$15	$S_f = \{ 2, 4, 6 \}$	5.00
$k = 2$					

# 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

- 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  $S_i$
- How good is this strategy?

$U = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$				
1.67	\$5	$S_a = \{ 1, 2, 3 \}$	\$4	$S_b = \{ 4, 5 \}$ 2.00
4.33	\$13	$S_c = \{ 3, 5, 7 \}$	\$3	$S_d = \{ 6 \}$ 3.00
.50	\$1	$S_e = \{ 1, 8 \}$	\$15	$S_f = \{ 2, 4, 6 \}$ 5.00
$k = 2$				

# Weighted Case: Greedy

- How good is the greedy strategy for the weighted case?
- **Claim.** Greedy is a  $(1 + \ln n)$ -approximation for weighted set cover.
- We prove this by proving a **different claim**:
- Let  $\mathcal{S}^*$  be the subsets picked by the optimal set cover
- For any  $S_i \in \mathcal{S}^*$ , greedy may not have picked  $S_i$  but
  - We show that greedy covers elements in  $S_i$  at cost no greater than  $O(\log n) \cdot w(S_i)$  (than choosing  $S_i$  itself)
- Every element is in some subset of  $\mathcal{S}^*$
- Thus, summing over subsets, we get that greedy covers all elements at cost no greater than  $O(\log n) \cdot \sum_i w(S_i) = O(\log n) \cdot \text{OPT}$



# Weighted Set Cover: 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  $S_i = \{a_1, a_2, \dots, a_d\}$  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  $a_d$ . Therefore  $c_d \leq w_i$



# Weighted Set Cover: 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  $w_i/2$ , therefore  $c_{d-1} \leq w_i/2$
- Similarly  $a_{d-2}$  is covered at amortized cost at most  $w_i/3$ . Each element  $a_j$  incurs an amortized cost at most  $c_j \leq w_i/(d - j + 1)$  up until  $a_1$  which is covered at amortized cost  $c_1 \leq w_i/d$

# Weighted Set Cover: 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.**

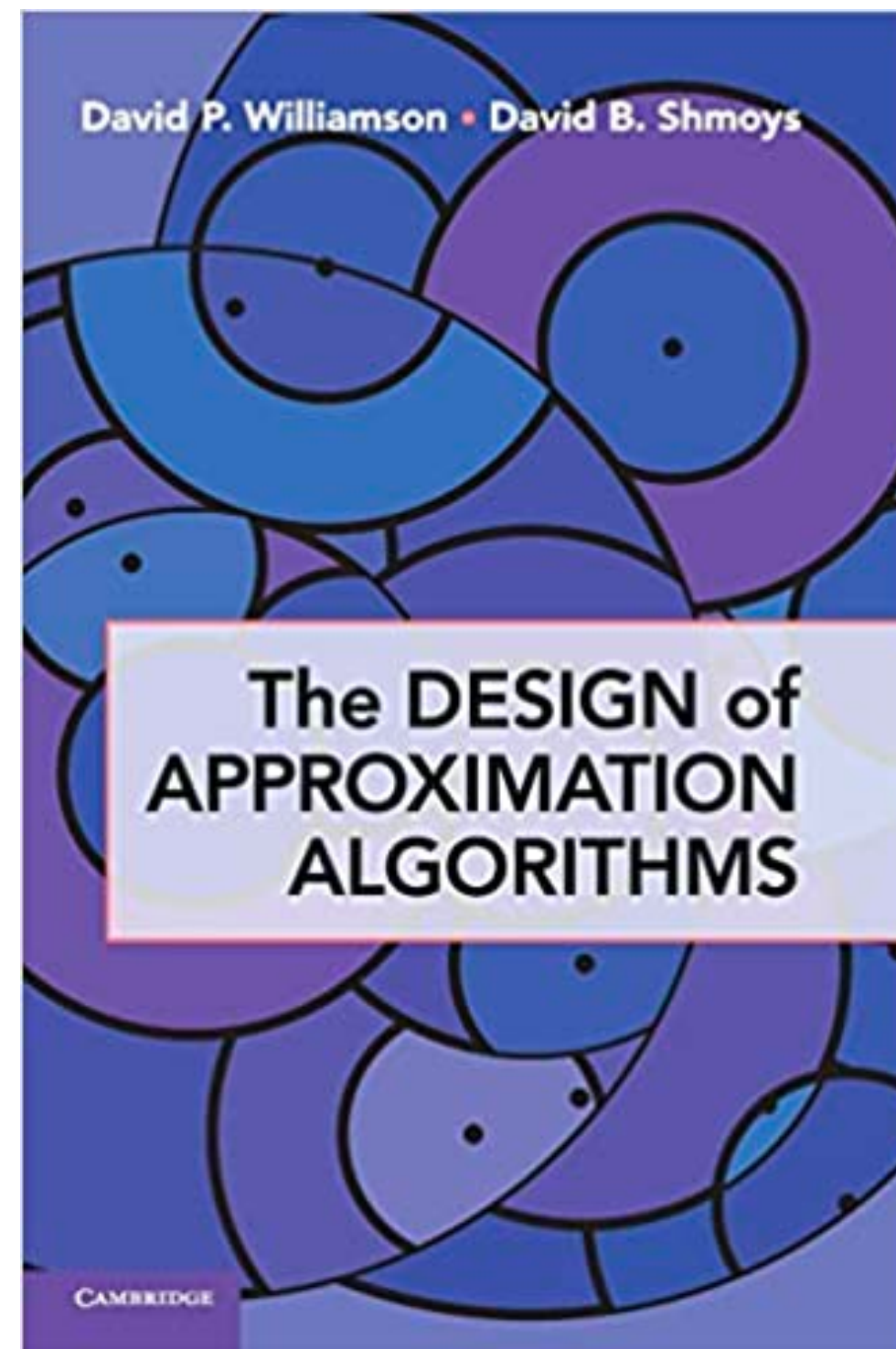
- Each element  $a_j$  incurs an amortized cost at most  $w_i/(d - j + 1)$
- Thus the total amortized cost of all elements in  $S_i$  is

$$\sum_{\ell \in S_i} c_\ell \leq w_i \left( \sum_{j=1}^d \frac{1}{n - j + 1} \right) = w_i H_d \leq w_i H_n \leq w_i (1 + \ln n)$$

- This analysis can be shown to be essentially 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  $P = NP$ .



# 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>)