Week 13 - Wednesday

COMP 4500

Last time

- What did we talk about last time?
- A little bit about computability
- Approximation algorithms
- Started load balancing

Questions?

Assignment 7

Logical warmup

- A sociologist named McSnurd gave a talk about a community that has clubs with the following attributes:
  - A person can belong to more than one club
  - Each club is named after a person
  - No two different clubs are named after the same person
  - Every person has a club named after him or her
  - A person may or may not belong to the club named after them
    - If they do, that person is called **sociable**
    - If they don't, that person is called **unsociable**
  - Perhaps surprisingly, there is a club whose entire membership is made up of all of the unsociable people
- Do you believe McSnurd's description?

Revenge of the logical warmup

- Another sociologist named McSnuff described a similar community:
  - As before, there is the same number of clubs as people, and each club is named after exactly one person
  - However, a person can be a member of a club openly or secretly
    - If a person is not openly a member of the club named after them, they are **suspicious**
    - If a person is secretly a member of the club named after them, they are a **spy**
  - Perhaps surprisingly, there is a club whose entire membership is made up of all the suspicious people
- Do you believe McSnuff's description?
- Do you know whether or not there is at least one spy in the community?

Load Balancing

Load balancing

- You have m machines $M_1, M_2, \ldots, M_m$
- You have n jobs
- Each job j has a processing time t_j
- We can assign jobs $A(i)$ to machine M_i
- The total time that M_i needs to work is:

$$T_i = \sum_{j \in A(i)} t_j$$

- We want to minimize the **makespan**, which is just the longest T_i
- In other words, we want the last machine running to stop running as early as possible
- Unfortunately, doing so in **NP-hard**

Greedy algorithm

- We can use a simple greedy algorithm for assigning jobs:
  - For each job ***j***, assign it to the machine that has the shortest completion time so far

Greedy algorithm gets a makespan $T \leq 2T^*$

- **Proof:**
  - Let M_i be the machine that get the maximum load T in the greedy assignment
  - Let j be the last job assigned to M_i
  - When j was assigned to M_i, it had the smallest load of any machine, namely $T_i - t_j$
  - Thus, every machine had load at least $T_i - t_j$

$$\sum_{k=1}^{m} T_k \geq m(T_i - t_j)$$

$$T_i - t_j \leq \frac{1}{m} \sum_{k=1}^{m} T_k$$

Proof continued

- Since $\sum_{k=1}^{m} T_k = \sum_{i=1}^{n} j_i$ and $\frac{1}{m}\sum_{i=1}^{n} j_i \leq T^*$

$$T_i - t_j \leq T^*$$

- But the optimal makespan must be at least as big as any job, thus $t_j \leq T^*$, thus:

$$T_i = (T_i - t_j) + t_j \leq T^* + T^* = 2T^*$$

- Since our makespan $T = T_i$, the proof is done.

■

Improved approximation algorithm

- We use a similar greedy algorithm
- However, we first sort all the jobs in descending order
- Now, $t_1 \geq t_2 \geq \ldots \geq t_n$
- If there are m jobs or fewer, our algorithm will be optimal, since each machine will get at most one job
- If there are more than m jobs, $T^* \geq 2t_{m+1}$
  - Consider the first $m + 1$ sorted jobs.
  - Each takes at least t_{m+1} time. Since there are at least $m + 1$ jobs and only m machines, one machine will get at least two of these jobs.
  - That machine will have processing time at least $2t_{m+1}$.

Sorted greedy algorithm gets a makespan $T \leq \frac{3}{2}T^*$

- **Proof:**
  - Let M_i be the machine that get the maximum load T in the greedy assignment
  - Let j be the last job assigned to M_i, and assume that M_i has at least 2 jobs
  - When j was assigned to M_i, it had the smallest load of any machine, namely $T_i - t_j$
  - Thus, every machine had load at least $T_i - t_j$

$$\sum_{k=1}^{m} T_k \geq m(T_i - t_j)$$

$$T_i - t_j \leq \frac{1}{m}\sum_{k=1}^{m} T_k$$

Proof continued

- Since $\sum_{k=1}^{m} T_k = \sum_{i=1}^{n} j_i$ and $\frac{1}{m}\sum_{i=1}^{n} j_i \leq T^*$

$$T_i - t_j \leq T^*$$

- Note that $\boldsymbol{j \geq m + 1}$, since the first $\boldsymbol{m}$ jobs will be put on $\boldsymbol{m}$ different machines
- Thus, $t_j \leq t_{m+1} \leq \frac{1}{2}T^*$
- But the optimal makespan must be at least as big as any job, thus $t_j \leq T^*$, thus:

$$T_i = (T_i - t_j) + t_j \leq T^* + \frac{1}{2}T^* = \frac{3}{2}T^*$$

- Since our makespan $T = T_i$, the proof is done.

■

Center Selection

- We have a set S of n sites, like towns
- We want to build k centers, like Starbucks
- We want to minimize the distance from a site to its closest facility, called the **covering radius**
- Distances obey the following rules:
  - $d(s, s) = 0$
  - $d(u, v) = d(v, u)$
  - $d(x, y) + d(y, z) \geq d(x, z)$

Bad greedy algorithm

- Put one facility in the middle of all the cities
- Keep adding centers to reduce the worst outlier
- First, it's not clear how to pick later centers
- Second, we can show that this could be arbitrarily bad with 2 cities and 2 centers
  - First Starbucks would go right between the two cities
  - Second one would go…where?
  - Obviously, the best locations would be right on top of the cities

Insight for better greedy algorithm

- Imagine that we knew that the maximum radius of cover was ***r***
- We could use this knowledge to get a covering radius of no more than 2***r***
- Algorithm:
  - Pick any city, put a Starbucks there
  - Remove any cities within 2***r*** of the city
  - Keep going as long as there are cities in the set

If more than *k* centers are returned, the best covering radius > *r*

- **Proof by contradiction:**
  - Suppose the opposite, that the algorithm returns more than k, but for optimal sites C^* of size k, the covering radius $r(C^*) \leq r$.
  - Now we want to consider the elements $c \in C$, the sites returned by the algorithm, and the elements $c^* \in C^*$, the optimal sites.
  - Say that centers c and c^* are **close** if $d(c, c^*) \leq r$.
  - Every center c was a site in the original problem, so there has to be at least one center c^* that is close.
  - We want to show that no optimal center c^* can be close to two different greedy centers c and c'.

Proof continued

- By the design of our algorithm, all centers c, $c' \in C$ are more than $2r$ away from each other.
- Because of the triangle inequality, $d(c, c^*) + d(c^*, c') \geq d(c, c') > 2r$.
- Thus, no c^* could be within r of both without breaking this inequality.
- That means every $c \in C$ must have at least one close c^* that no other c' does.
- Thus, each c has exactly one c^* not shared by any other, making $|C^*| \geq |C| > k$, contradicting the assumption that C^* has at most k centers.

■

Without knowing *r*

- We know that ***r*** > 0 and ***r*** is less than or equal to maximum distance between any two sites
  - We could binary search between those two values
- Instead, our algorithm that magically knew ***r*** only used it to pick sites 2***r*** or further from existing centers
- So…all we really need to do is pick sites that are far away from our existing centers

Updated greedy algorithm

- Assume $k < |S|$, otherwise pick all sites
- Select any site s to start with and let $C = \{s\}$
- While $|C| < k$
  - Find a site $s \in S$ that is as far away as possible from every element in C
  - Add s to C
- Return C as the selected set of centers

Proof of approximation bound

- **Claim:**
  - Our new greedy algorithm returns a set C of k points such that $r(C) \leq 2r(C^*)$ where C^* is an optimal set of k points.
- **Proof by contradiction:**
  - Assume we got a set C with k centers such that $r(C) > 2r$.
  - There is some site s that is more than $2r$ from every center in C.
  - At some point in the algorithm, we have only selected centers C'.
  - We are just about to add center c'.

Proof continued

- We claim that c' is at least $2r$ away from all sites in C' because of this inequality:
  - $d(c', C') \geq d(s, C') \geq d(s, C) > 2r$
- So our greedy algorithm follows the first k iterations of the algorithm that knew r since it always picks a center more than $2r$ from previously selected centers.
- But we proved that algorithm would only pick more than k centers if the optimal k centers did not have a covering radius of r.
- By the same contradiction, no site s can be further than $2r$ from a center, so $r(C) \leq 2r$.

■

Three Sentence Summary of Set Cover

Set cover (optimization version)

- Given:
  - Set U of n elements
  - Collection of sets $S_1, S_2, \ldots, S_m$ of subsets of U
  - Each subset S_i has a weight $w_i \geq 0$
- Find the subsets with smallest total weight whose union is equal to all of U

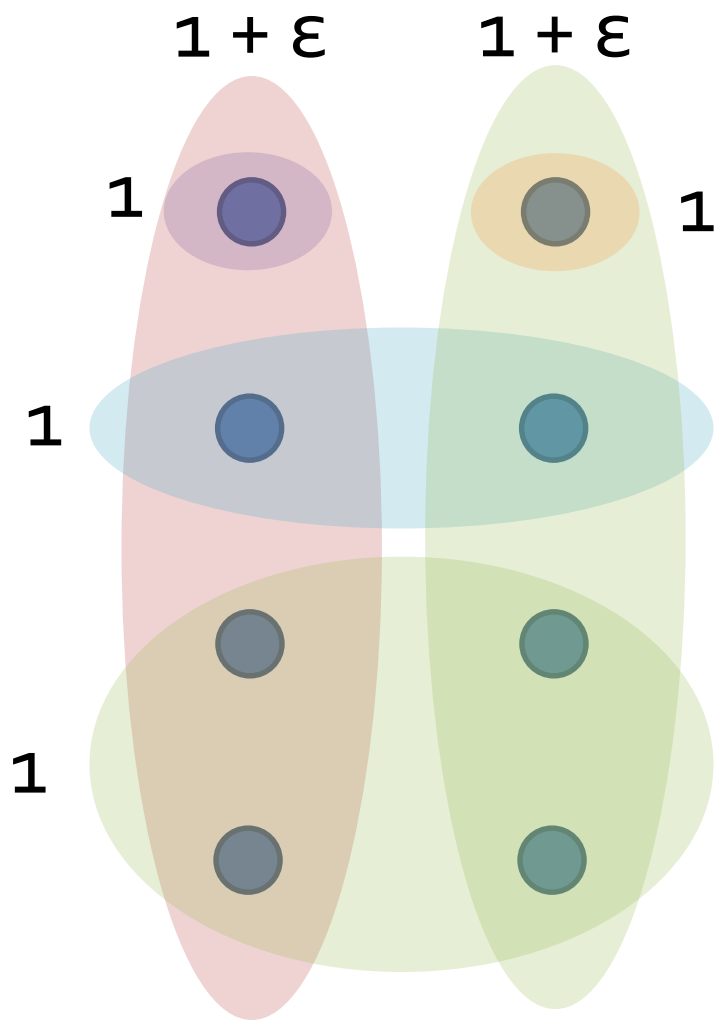
Algorithm design

- We want the most bang for our buck
- We want small weight sets with a lot of elements
  - In other words, low cost per element
- So, we look at the value $w_i/|S_i|$ for each set, and pick the lowest such value set
- We keep doing that, but we only "count" the elements in each set that still aren't covered

Greedy set cover algorithm

- Start with $\boldsymbol{R} = \boldsymbol{U}$ and no sets selected
- While $\boldsymbol{R} \neq \emptyset$
  - Select set $\boldsymbol{S_i}$ with minimum $\boldsymbol{w_i}/|\boldsymbol{S_i} \cap \boldsymbol{R}|$
  - Delete set $\boldsymbol{S_i}$ from $\boldsymbol{R}$
- Return the selected sets

Set cover example



Algorithm finds a total weight of 4

Optimal is a total weight of $2 + 2\varepsilon$

Analysis

- How good (or bad) is our set cover approximation in the worst case?
- Let's think about the cost per item incurred by each set we add:
  - $c_s = w_i/|S_i \cap R|$ for all $s \in S_i \cap R$
  - Imagine we assign that cost in the algorithm when we cover those elements
- Clearly, these c_s values end up being the total weight of our solution C:

$$\sum_{s_i \in C} w_i = \sum_{s \in U} c_s$$

Unfortunately: math

- To bound our analysis, we will use the idea of the harmonic function:

$$H(n) = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

- This function grows…slowly but infinitely
- You might recall that $H(n)$ is $\Theta(\log n)$

Bound on each set

- **Claim:**
  - For every set S_k, $\sum_{s \in S_k} c_s$ is at most $H(|S_k|)w_k$
- **Proof:**
  - For notation, assume $|S_k| = d$, and S_k = the first d elements of U
  - In other words, $S_k = \{s_1, s_2, \ldots, s_d\}$
  - Even better, let's assume the elements are labeled in the order that they are assigned a cost c_{s_j}

Proof continued

- Consider the iteration when s_j is covered by our algorithm, for some $j \leq d$.
- Before this iteration, $s_j, s_{j+1}, \ldots, s_d \in R$
- This implies that $|S_k \cap R|$ is at least $d - j + 1$, making the average cost of set S_k at most

$$\frac{w_k}{|S_k \cap R|} \leq \frac{w_k}{d - j + 1}$$

- On this particular iteration, the greedy algorithm selects a set S_i of minimum average cost
  - Thus, S_i has an average cost no more than S_k

Proof continued

- The average cost of S_i is what will get assigned to s_j, so

$$c_{s_j} = \frac{w_i}{|S_i \cap R|} \leq \frac{w_k}{|S_k \cap R|} \leq \frac{w_k}{d - j + 1}$$

- Summing up all the costs for every element $s \in S_k$

$$\sum_{s \in s_k} c_s = \sum_{j=1}^{d} c_{s_j} \leq \sum_{j=1}^{d} \frac{w_k}{d - j + 1}$$

$$= \frac{w_k}{d} + \frac{w_k}{d-1} + \cdots + \frac{w_k}{1} = H(d) \cdot w_k$$

■

Final approximation bound

- Let d^* be the size of the largest set
- **Claim:**
  - Set cover C found by our greedy algorithm has weight at most $H(d^*)$ times the optimal weight w^*
- **Proof:**
  - The optimal set cover C^* has weight $w^* = \sum_{S_i \in C^*} w_i$
  - By our previous proof:

$$w_i \geq \frac{1}{H(d^*)} \sum_{s \in S_i} c_s$$

Approximation bound continued

- Since C^* is a set cover

$$\sum_{S_i \in C^*} \sum_{s \in S_i} c_s = \sum_{s \in U} c_s$$

- Putting it all, insanely, together:

$$w^* = \sum_{S_i \in C^*} w_i \geq \sum_{S_i \in C^*} \frac{1}{H(d^*)} \sum_{s \in S_i} c_s \geq \frac{1}{H(d^*)} \sum_{s \in U} c_s = \frac{1}{H(d^*)} \sum_{S_i \in C} w_i$$

■

Log approximation

- All of that madness means that our approximation algorithm to set cover might return a set cover costing O(log ***d****) times the true optimal
- Worse, ***d**** could be some constant fraction of ***n***, making the approximation an O(log ***n***) times worse than optimal
- This approximation is worse than **any** constant approximation, since our approximation actually will degrade as ***n*** gets larger
- To top it off, there's even a proof that this is the best you can approximate set cover, unless **P** = **NP**

Upcoming

Next time…

- Approximating knapsack
- Read section 11.8

Reminders

- Work on Assignment 7
  - Due the last day of class