More Approximation Algorithms

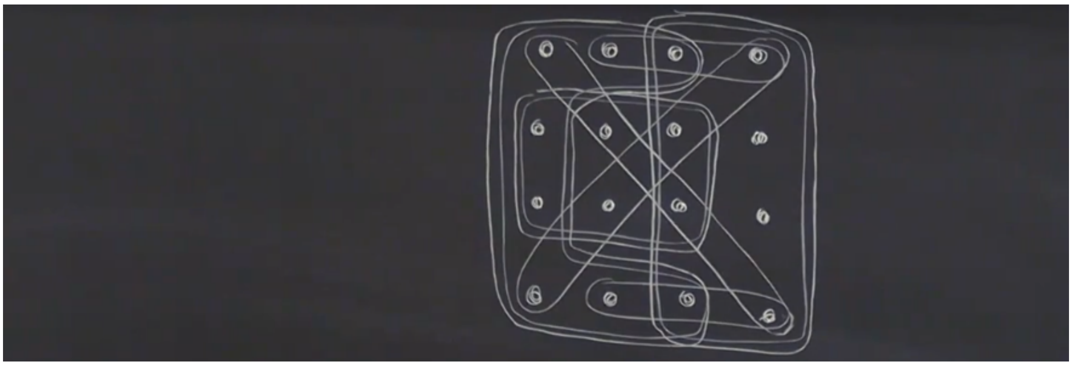
Frank

December 13, 2021

1 Set Cover Problem

1.1 Generalized Problem

- **Input**:
  - Universe $U = \{a_1, \ldots, a_n\}$
  - Set $S = \{A_1, \ldots, A_m\}$ where $A_i \subseteq U$ and $\cup_{i=1}^{m} A_i = U$
- **Output**:
  - Define cover C as a collection of sets in S s.t. $\cup_{A \in C} A = U$, then output is a **minimum cardinality cover** C
- Example:



Where dots are elements of the universe, and blobs are sets of elements. Double blobs are included in the cover. Then minimum cardinality cover has cardinality of 3.

1.2 Special Case: Vertex Cover

- Given graph $G = (V, E)$

  $U = E$ and $\forall v \in V$, define $A_v = \{e : v \text{ is an endpoint of } e\}$

- It turns out that the special case is an "easier" problem than the general problem (which is usually the case). In this case, we are able to find a 2-approximation for this special case, whereas we can do at best $O(\log n)$ for the general case.

1.3 Approximation Algorithm: Greedy

- The greedy approach is to choose the set that covers the most uncovered elements at each step.

---

procedure GreedySetCover
    $C \leftarrow \emptyset$
    $R \leftarrow U$ ▷ Remaining Uncovered
    while $(R \neq \emptyset)$ **do**
        $A \leftarrow$ set in $S - C$ s.t. $|A \cap R|$ is maximum
        $C \leftarrow C \cup \{A\}$
        $R \leftarrow R - A$
    return C

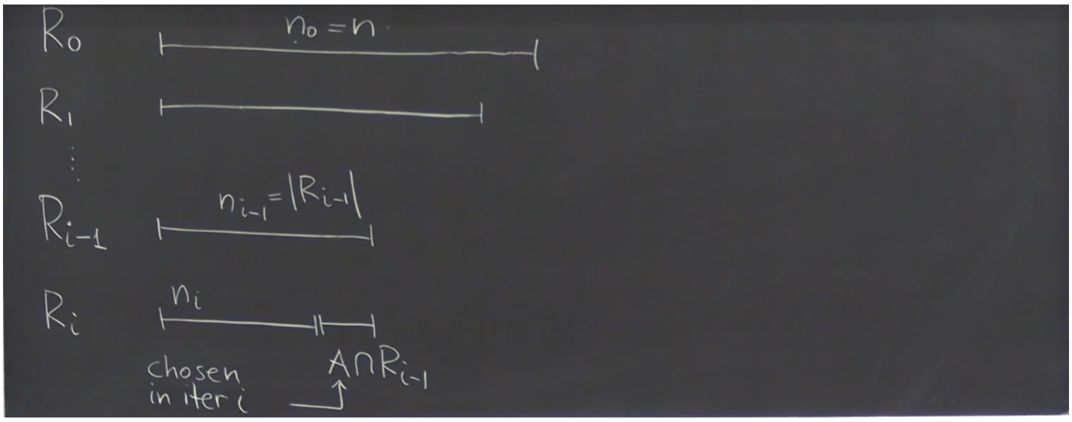
- So how much bigger is C compared to optimal cover in the worst case?

1.4 Error Analysis

- Let $n = |U|$, $k = |C|$, $l =$ size of optimal cover $C^\star$

  Fact: $k \leq \ln n \cdot l$ where $\ln n$ is our approximation ratio

- Let R_i be the number of uncovered elements after iteration i



- Note that $C^\star$ contains sets that cover all of R_{i-1}, since it is a cover. These sets are not in C_{i-1} and there are at most l of these sets. $\implies$ at least one of these sets must cover $\geq \frac{n_{i-1}}{l}$ elements in R_{i-1}, because taking the opposite, where all of these sets must cover $< \frac{n_{i-1}}{l}$ elements in R_{i-1} leads to a contradiction that $C^\star$ is a cover.

  $\implies A \cap R_{i-1}$ covers at least $\frac{n_{i-1}}{l}$ elements of R_{i-1}

  $\implies n_i \leq n_{i-1} - \frac{n_{i-1}}{l}$

  $\implies n_i \leq n_{i-1}(1 - \frac{1}{l})$

  $\implies n_i \leq n(1 - \frac{1}{l})^i$

  Consider that $1 - x \leq e^{-x}$ where equality only when $x = 0$, but $\frac{1}{l} \neq 0$, thus $n_i < n \cdot e^{-\frac{i}{l}}$

  Take $i = l \cdot \ln n \implies n_i < n \cdot e^{-\ln n}$

$\implies n_i < n \cdot \frac{1}{n}$

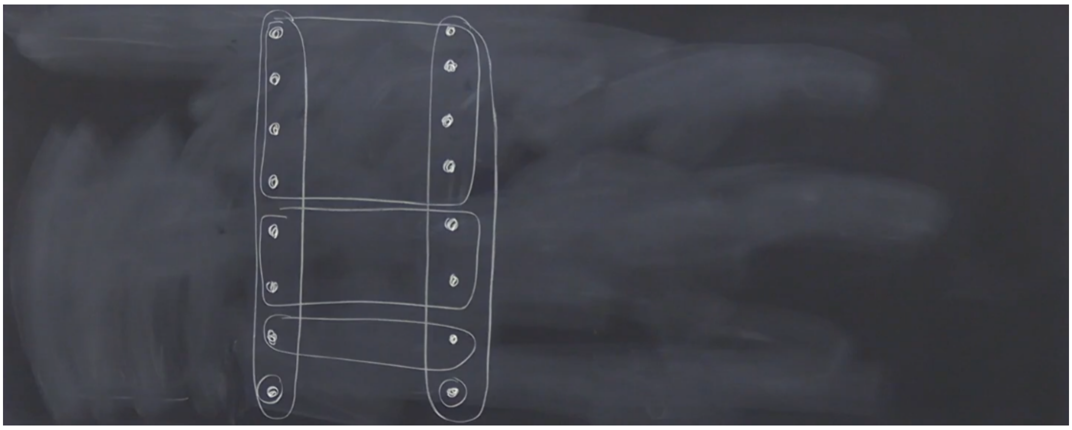
$\implies n_i < 1 \implies n_i = 0$

- Thus, after $l \cdot \ln n$ iterations, we have 0 uncovered elements, and the algorithm stops.

  $\implies$ we will have taken at most $l \cdot \ln n$ sets into our cover

  $\implies k \leq l \cdot \ln n$, and we are done.

- Consider this example, where our greedy algorithm chooses at most $\log_2 n$ more elements than the optimal solution (by choosing the rectangles that decreases by a factor of 2 each time)



  This means the algorithm cannot do much better asymptotically. We improved to base 2 log instead of base e log.

- Another fact (not going to be proven here): Unless $P = NP$, any polytime approximation algorithm for set cover has approximation ratio $\Omega(\log n)$

  However, remember the special case of set cover, vertex cover, has a 2-approximation algorithm, which is constant.

2 Weighted Set Cover

2.1 Problem

- Each set A has weight $w(A)$.
- Want set cover C of minimum total weight.

2.2 Modification to our algorithm

- Instead of maximizing $|A \cap R|$, most uncovered elements, we maximize the ratio $\frac{|A \cap R|}{w(A)}$ and approach greedily that way.
- And it turns out that this is also bounded by $\log n \cdot l$ in terms of the size of the cover

3 Minimum Makespan (load balancing)

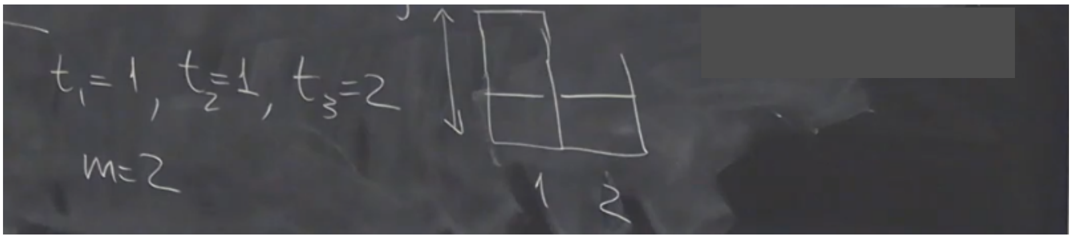
3.1 Problem

- **input**:
  - m machines $1, 2, \ldots, m$
  - n jobs $1, 2, \ldots, n$
  - where job j takes time t_j on any machine
  - Assignment $A(i) \subseteq \{1, 2, \ldots, n\}$
    where $\cup_{1 \leq i \leq m} A(i) = \{1, 2, \ldots, n\}$ and $A(i) \cap A(i') = \emptyset \; \forall i \neq i'$
  - Load of machine i (under assignment A) $= \sum_{j \in A(i)} t_j$
  - Makespan of A = max load of any machine
- **Output**:
  - An assignment with the minimum makespan.
- NP-hard problem

3.2 Algorithm

1. Put next job in least loaded machine (load balancing)

- Not optimal, consider



If we had been patient, we would have stacked job 1 and 2 together for a makespan of 2, but instead we have 3.

3.3 Error Analysis

- Let L^g = makespan of greedy algorithm assignment
- Let $L^\star$ = makespan of optimal algorithm assignment
- **Thm 1**: $L^g \leq (2 - \frac{1}{m}) \cdot L^\star$

  Proof:
  - Let i = most loaded machine (under greedy assignment)
  - l = last job assigned to i (i was a least loaded machine before the job l was assigned to it, by algorithm)

- Then after l is assigned to i,

$$L^g \leq \frac{1}{m}\left[\sum_{j=1}^{n} t_j - t_l\right] + t_l$$

Where $\frac{1}{m} \cdot \sum_{j=1}^{n} t_j - t_l$ was the maximum possible load of i when l was added to it. This is taking all the load (without job l assigned) and distributing it amongst all machines, and because i was the least loaded machine, we cannot have that the least machine has more than $\frac{1}{m} \cdot \sum_{j=1}^{n} t_j - t_l$ load, otherwise we would have more load than we actually have.

Of course, we then add l to the least loaded machine, and by our choice, i becomes the most loaded machine.

Continuing, we have

$$\begin{aligned} L^g &\leq \left(\frac{1}{m}\sum_{j=1}^{n} t_j\right) + \left(1 - \frac{1}{m}\right) t_l \\ &\leq \frac{1}{m} \cdot m \cdot L^\star + \left(1 - \frac{1}{m}\right) \cdot t_l \end{aligned}$$

Since the sum of all the loads is bounded by the number of machines times the optimal makespan (minimized maximum load).

Finally,

$$\begin{aligned} L^g &\leq L^\star + \left(1 - \frac{1}{m}\right) \cdot L^\star \\ &\leq L^\star \left(2 - \frac{1}{m}\right) \end{aligned}$$

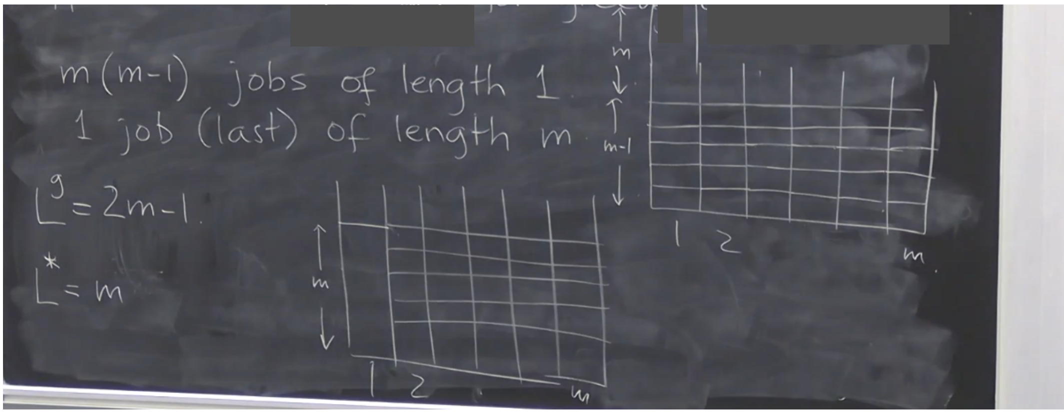
Since t_l is at most $L^\star$.

- We have also proven that

$$L^g \leq L^\star + \left(1 - \frac{1}{m}\right) \cdot t_l \quad (\dagger)$$

3.4 Tight Approximation Ratio Bound?

- Yes, the approximation ratio $\left(2 - \frac{1}{m}\right)$ for greedy algorithm is tight. Consider the example:



And we have found an example that is as bad as our bound

For instance:

$$L^g = 2m - 1 = L^\star \cdot \left(2 - \frac{1}{m}\right)$$

3.5 Longest-First (LF) Greedy

- Sort the jobs in non-increasing length, then put the jobs in order, into the least loaded machine at every instance.
- Let L^{lf} = makespan of assignment by LF greedy algorithm
- **Thm 2**: $L^{lf} \leq \left(\frac{3}{2} - \frac{1}{2m}\right) \cdot L^\star$

  Proof:

  - Let i be the most loaded machine under LF greedy algorithm
  - Let l be the last job assigned to i by FL greedy
  - **Case 1**: l is the only job assigned to i

    $\implies L^{lf} = t_l \leq L^\star$ since it is the only job

    $\implies L^{lf} = L^\star$, since it cannot be better than optimal
  - **Case 2**: Machine i is assigned ≥ 2 jobs

    $\implies l \geq m + 1$, because the least machine has at least 1 job at the time of assignment of l, this means that every machine must already have a job, then l is at least the $(m+1)^{\text{st}}$ job

    Also, we must have that $L^\star \geq 2 \cdot t_{m+1}$, because any assignment (in particular the optimal one) must assign two of the jobs $1, 2, \ldots, m+1$ to the same machine and we have **ordered the jobs in non-increasing length**, and thus each of these two jobs has length $\geq t_{m+1}$

    $\implies L^\star \geq 2 \cdot t_{m+1} \geq 2 \cdot t_l$

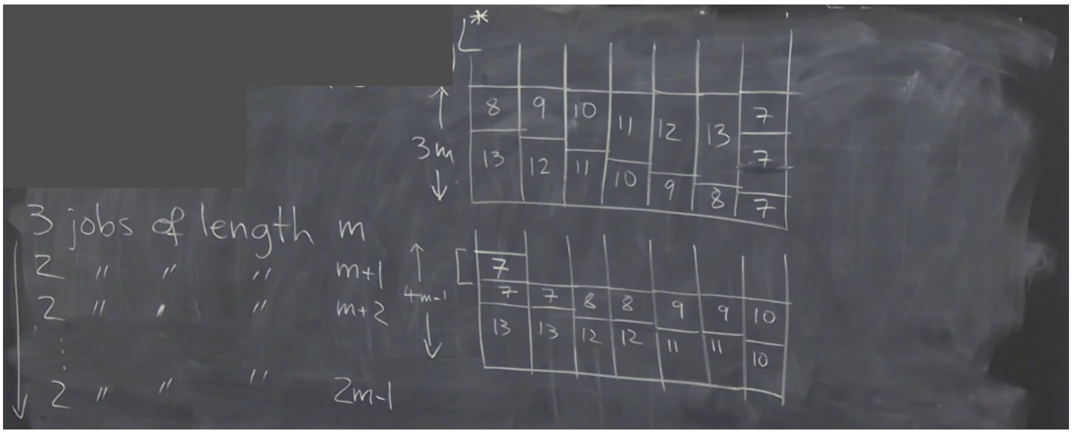
    $\implies t_l \leq \frac{L^\star}{2}$

– Now, by (†),

$$
\begin{aligned}
L^{lf} &\leq L^{\star} + \left(1 - \frac{1}{m}\right) \cdot t_l \\
&\leq L^{\star} + \left(1 - \frac{1}{m}\right) \cdot \frac{L^{\star}}{2} \\
&= L^{\star}\left(\frac{3}{2} - \frac{1}{2m}\right)
\end{aligned}
$$

- And thus we have achieved an improvement.
- However, we cannot find an example that achieves this bound, because we have been sloppy for this proof.
- Fact: $L^{lf} \leq \left(\frac{4}{3} - \frac{1}{3m}\right) \cdot L^{\star}$ is the tight bound. Which will not be proven here.

  Example that achieves this bound is:



- Note that this is algorithm is better, but this requires prior knowledge of the jobs, whereas the first algorithm can be "online" (see input one at a time)