Load Balancing

Problem

Input: m identical machines, n jobs with ith job having processing time t_i

Goal: Schedule jobs to computers such that

- Jobs run contiguously on a machine
- A machine processes only one job a time
- Makespan or maximum load on any machine is minimized

Definition

Let $A(i)$ be the set of jobs assigned to machine i. The load on i is $T_i = \sum_{j \in A(i)} t_j$.

The makespan of A is $T = \max_i T_i$

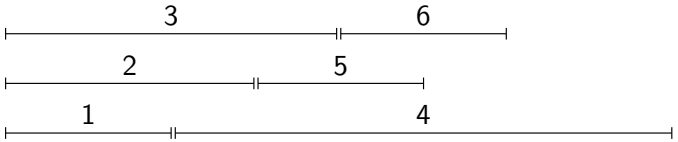
Load Balancing: Example

Example

Consider 6 jobs whose processing times is given as follows

Jobs	1	2	3	4	5	6
t_i	2	3	4	6	2	2

Consider the following schedule on 3 machines



The loads are: $T_1 = 8$, $T_2 = 5$, and $T_3 = 6$. So makespan of schedule is 8

Load Balancing is NP-Complete

Decision version: given n, m and $t_1, t_2, \ldots, t_n$ and a target T, is there a schedule with makespan at most T?

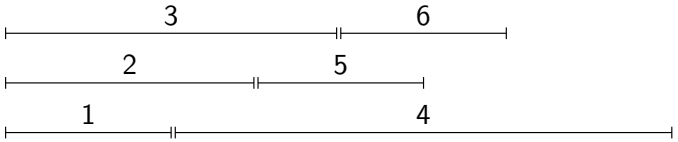
Problem is NP-Complete: reduce from Subset Sum.

Greedy Algorithm

1. Consider the jobs in some fixed order
2. Assign job j to the machine with lowest load so far

Example

Jobs	1	2	3	4	5	6
t_i	2	3	4	6	2	2



Putting it together

```
for each machine i
    T_i = 0 (* initially no load *)
    A(i) = ∅ (* initially no jobs *)
for each job j
    Let i be machine with smallest load
    A(i) = A(i) ∪ {j} (* schedule j on i *)
    T_i = T_i + t_j (* compute new load *)
```

Running Time

- First loop takes $O(m)$ time
- Second loop has $O(n)$ iterations
- Body of loop takes $O(\log m)$ time using a priority heap
- Total time is $O(n \log m + m)$

Optimality

Problem

Is the greedy algorithm optimal? No! For example, on

Jobs	1	2	3	4	5	6
t_i	2	3	4	6	2	2

the greedy algorithm gives schedule with makespan 8, but optimal is 7

In fact, the load balancing problem is *NP*-complete.

Quality of Solution

Theorem (Graham 1966)

The makespan of the schedule output by the greedy algorithm is at most 2 times the optimal make span. In other words, the greedy algorithm is a 2-approximation.

Challenge

How do we compare the output of the greedy algorithm with the optimal? How do we get the value of the optimal solution?

- We will obtain bounds on the optimal value

Bounding the Optimal Value

Lemma

$T^* \geq \max_j t_j$, *where* T^* *is the optimal makespan*

Proof.

Some machine will run the job with maximum processing time □

Lemma

$T^* \geq \frac{1}{m} \sum_j t_j$, *where* T^* *is the optimal makespan*

Proof.

- Total processing time is $\sum_j t_j$
- Some machine must do at least $\frac{1}{m}$ (or average) of the total work □

Analysis of Greedy Algorithm

Theorem

The greedy algorithm is a 2-approximation

Proof.

Let machine i have the maximum load T_i, and let j be the last job scheduled on machine i

- At the time j was scheduled, machine i must have had the least load; load on i before assigning job j is $T_i - t_j$
- Since i has the least load, we know $T_i - t_j \leq T_k$, for all k. Thus, $m(T_i - t_j) \leq \sum_k T_k$
- But $\sum_k T_k = \sum_\ell t_\ell$. So $T_i - t_j \leq \frac{1}{m} \sum_k T_k = \frac{1}{m} \sum_\ell t_\ell \leq T^*$
- Finally, $T_i = (T_i - t_j) + t_j \leq T^* + T^* = 2T^*$ □

Tightness of Analysis

Proposition

The analysis of the greedy algorithm is tight, i.e., there is an example on which the greedy schedule has twice the optimal makespan

Proof.

Consider problem of $m(m-1)$ jobs with processing time 1 and the last job with processing time m.
Greedy schedule: distribute first the $m(m-1)$ jobs equally among the m machines, and then schedule the last job on machine 1, giving a makespan of $(m-1) + m = 2m - 1$
The optimal schedule: run last job on machine 1, and then distribute remaining jobs equally among $m-1$ machines, giving a makespan of m □

Improved Greedy Algorithm

Modified Greedy

Sort the jobs in descending order of processing time, and process jobs using greedy algorithm

```
for each machine i
    T_i = 0 (* initially no load *)
    A(i) = ∅ (* initially no jobs *)
for each job j in descending order of processing time
    Let i be machine with smallest load
    A(i) = A(i) ∪ {j} (* schedule j on i *)
    T_i = T_i + t_j (* compute new load *)
```

Technical Lemma

Lemma

If there are more than m jobs then $T^* \geq 2t_{m+1}$

Proof.

Consider the first $m+1$ jobs

- Two of them must be scheduled on same machine, by pigeon-hole principle
- Both jobs have processing time at least t_{m+1}, since we consider jobs according to processing time, and this proves the lemma □

Analysis of Modified Greedy

Theorem

The modified greedy algorithm is a 3/2-approximation

Proof.

Once again let i be the machine with highest load, and let j be the last job scheduled

- If machine i has only one job then schedule is optimal
- If i has at least 2 jobs, then it must be the case that $j \geq m+1$. This means $t_j \leq t_{m+1} \leq \frac{1}{2} T^*$
- Thus, $T_i = (T_i - t_j) + t_j \leq T^* + \frac{1}{2} T^*$ □

Tightness of Analysis

Theorem (Graham)

Modified greedy is a 4/3-approximation

The 4/3-analysis is tight.

(Weighted) Set Cover Problem

Input Given a set U of n elements, a collection $S_1, S_2, \ldots S_m$ of subsets of U, with weights w_i

Goal Find a collection $\mathcal{C}$ of these sets S_i whose union is equal to U and such that $\sum_{i \in \mathcal{C}} w_i$ is minimized.

Example

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

$S_1 = \{1\}$	$w_1 = 1$	$S_2 = \{2\}$	$w_2 = 1$
$S_3 = \{3, 4\}$	$w_3 = 1$	$S_4 = \{5, 6, 7, 8\}$	$w_4 = 1$
$S_5 = \{1, 3, 5, 7\}$	$w_5 = 1 + \epsilon$	$S_6 = \{2, 4, 6, 8\}$	$w_6 = 1 + \epsilon$

$\{S_5, S_6\}$ is a set cover of weight $2 + 2\epsilon$

Greedy Rule

- Pick the next set in the cover to be the one that makes "most progress" towards the goal
  - Covers many (uncovered) elements
  - Has a small weight
- If R is the set of elements that aren't covered as yet, add set S_i to the cover, if it minimizes the quantity $\frac{w_i}{|S_i \cap R|}$

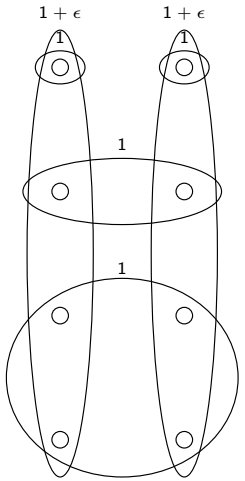
Greedy Algorithm

```
Initially R = U and C = ∅
while R ≠ ∅
    let S_i be the set that minimizes w_i/|S_i ∩ R|
    C = C ∪ {i}
    R = R \ S_i
return C
```

Running Time

- Main loop iterates for $O(n)$ time, where $|U| = n$
- Minimum S_i can be found in $O(\log m)$ time, using a priority heap, where there are m sets in set cover instance
- Total time is $O(n \log m)$

Example: Greedy Algorithm



Example

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

$$S_1 = \{1\} \quad S_2 = \{2\}$$
$$S_3 = \{3, 4\} \quad S_4 = \{5, 6, 7, 8\}$$
$$S_5 = \{1, 3, 5, 7\} \quad S_6 = \{2, 4, 6, 8\}$$

$w_1 = w_2 = w_3 = w_4 = 1$ and $w_5 = w_6 = 1 + \epsilon$
Greedy Algorithm first picks S_4, then S_3, and finally S_1 and S_2

Cost for covering an element

Definition

- Suppose the greedy algorithm selects sets $S^1, S^2, \ldots S^k$ (in that order) to form the set cover.
- Consider an element s that is first covered when S^i is picked
- Let R be the set of elements that are uncovered when S^i is picked
- The cost of covering s is $c_s = w(S^i)/|S^i \cap R|$, where $w(S^i)$ is weight of the set S^i

Cost of element: Example

Example

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

$$S_1 = \{1\} \quad S_2 = \{2\}$$
$$S_3 = \{3, 4\} \quad S_4 = \{5, 6, 7, 8\}$$
$$S_5 = \{1, 3, 5, 7\} \quad S_6 = \{2, 4, 6, 8\}$$

$w_1 = w_2 = w_3 = w_4 = 1$ and $w_5 = w_6 = 1 + \epsilon$. The greedy algorithm picks S_4, S_3, S_2, S_1 in that order. The costs of elements are given as follows

$$c_1 = 1 \quad c_2 = 1 \quad c_3 = 1/2 \quad c_4 = 1/2$$
$$c_5 = 1/4 \quad c_6 = 1/4 \quad c_7 = 1/4 \quad c_8 = 1/4$$

Costs of Covers and Elements

Proposition

If $\mathcal{C}$ is the set cover computed by the greedy algorithm then $\sum_{i \in \mathcal{C}} w_i = \sum_{s \in U} c_s$

Proof left as exercise

Main Idea

Upper bound the ratio $\frac{\sum_{s \in S_k} c_s}{w_k}$, i.e., to say "to cover a lot of cost, you need to use a lot of weight"

Bounding costs of sets

Lemma

For every set S_k, $\sum_{s \in S_k} c_s \leq H(|S_k|) \cdot w_k$, where $H(n) = \sum_{i=1}^{n} \frac{1}{i} = \Theta(ln\ n)$

Proof.

Let $S_k = \{s_1, \ldots s_d\}$, where s_i is covered before s_j if $i \leq j$

- Consider the iteration when s_j is covered; at this time $R \supseteq \{s_j, s_{j+1}, \ldots s_d\}$
- Average progress cost of S_k is $w_k/|S_k \cap R| \leq w_k/(d - j + 1)$
- Suppose s_j gets covered because S_i is selected by greedy algorithm. Then, $c_{s_j} = \frac{w_i}{|S_i \cap R|} \leq \frac{w_k}{|S_k \cap R|} \leq \frac{w_k}{d-j+1}$
- Hence, $\sum_{s \in S_k} c_s = \sum_{j=1}^{d} c_{s_j} \leq \sum_{j=1}^{d} \frac{w_i}{d-j+1} = H(d) w_k$ □

Analysis of the Greedy Algorithm

Theorem

The greedy algorithm for set cover is a $H(d^*)$-approximation, where $d^* = \max_i |S_i|$

Proof.

Let $\mathcal{C}^*$ be the optimal set cover, and $\mathcal{C}$ the set cover computed by greedy algorithm

- By previous lemma we know $w_i \geq \frac{1}{H(d^*)} \sum_{s \in S_i} c_{s_i}$ and so we have $w^* = \sum_{i \in \mathcal{C}^*} w_i \geq \sum_{i \in \mathcal{C}^*} \frac{1}{H(d^*)} \sum_{s \in S_i} c_{s_i}$
- Further, $\sum_{i \in \mathcal{C}^*} \sum_{s \in S_i} c_s \geq \sum_{s \in U} c_s$
- Thus, $w^* = \sum_{i \in \mathcal{C}^*} w_i \geq \sum_{i \in \mathcal{C}^*} \frac{1}{H(d^*)} \sum_{s \in S_i} c_{s_i} \geq \frac{1}{H(d^*)} \sum_{s \in U} c_s = \frac{1}{H(d^*)} \sum_{i \in \mathcal{C}} w_i$ □

Tightness of Analysis

Analysis Tight?

Does the Greedy Algorithm give better approximation guarantees? No!

Consider a generalization of the set cover example. Each column has 2^{k-1} elements, and there are two sets consisting of a column each with weight $1 + \epsilon$. Additionally there are $\log n$ sets of increasing size of weight 1. The greedy algorithm will pick these $\log n$ sets given weight $\log n$, while the best cover has weight $2 + 2\epsilon$

Best Algorithm for Set Cover

Theorem

If $P \neq NP$ then no polynomial time algorithm can achieve a better than $H(n)$ approximation.

Proof beyond the scope of this course.