Approximation Algorithms: Load Balancing
Approximations

- The word *approximation* is used for several flavors of algorithms
- Approximating optimization problems
  - An algorithm is a $c$-approximation if its cost is $c \cdot \text{OPT}$, where OPT is the optimum cost; ($c < 1$ or a maximization problem, $c > 1$ for min)
- Approximations in an online setting
  - An online algorithm is a $c$-approximation/$1$-competitive, if it has cost $c \cdot \text{OPT}$, where OPT is the cost of an offline algorithm (that knows the entire input ahead of time); ($c < 1$ or a max problem, $c > 1$ for min)
- The word approximate is also used to indicate that the algorithm is permitted to make one-sided errors (such false positive)
  - Bloom filter
  - Approximate hitter algorithm (Problem 5(b) on assignment)
Challenges: Approximation Algorithms

- Approximating problems that are NP hard
  - Main challenge is showing that the algorithm performs close to optimal when the optimal solution is not known/NP hard
  - Usually done by lower (upper) bounding the cost of the optimal solution for minimization (maximization) problems

- Approximation for online algorithms
  - High benchmark. Comparison against an optimal that knows the entire future, while the algorithm does not even know the next element
  - Sometimes dealt with using “resource augmentation”—allowing the algorithm some flexibility compared to the optimal
Online: Ski Rental Problem

• Assume that you are taking ski lessons
• After each lesson you decide (depending on how much you liked it and how cold you are) whether to continue to ski or to quit entirely
• Question: rent or buy?
• Cost of renting $1$ (say)
• Cost of buying $B$

- **Offline strategy.** If you knew in advance how many times you would ski, say $t$ times, what is the best strategy?
  • If $t \geq B$ times, then buy, else rent
  • In other words, optimal offline cost is $\min\{t, B\}$
Online strategy. We need to figure out a decision point, a number $k$ such you buy skis on the $k$th visit (renting before then)

**Claim.** If we set $k = B$ (the cost of buying skis), we are guaranteed to never pay more than twice of the best offline optimal strategy

That is, buying on the $B$th ski visit is 2—competitive

Even if you quit right after the $B$th visit, $t \geq B$

Offline cost is $\min\{t, B\} = B$

Online strategy’s cost?

- $(k - 1) \cdot 1 + B = (B - 1) + B = 2B - 1$

Competitive/Approximation ratio?

- $2B - 1/B = 2 - 1/B \leq 2$
Load Balancing

- **Input.** $m$ identical machines; $n$ jobs, and processing times $t_1, \ldots, t_m$, where job $j$ has processing time $t_j$ (on any machine).
- Job $j$ must run contiguously on one machine.
- A machine can process at most one job at a time.
- Let $S[i]$ be the subset of jobs assigned to machine $i$.

The **load of machine** $i$ is $L[i] = \sum_{j \in S[i]} t_j$ (total processing time).

```
---    ---
  0     time
```

<table>
<thead>
<tr>
<th>Machine 1</th>
<th>Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>d</td>
<td>c</td>
</tr>
<tr>
<td>f</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td>g</td>
</tr>
</tbody>
</table>
Load Balancing

- The **makespan** of an algorithm is the maximum load on any machine
  \[ L = \max_i L[i] \]

- **Load balancing Problem.** Assign jobs to machines so as to minimize makespan.

- **Claim.** Load balancing is NP hard even with \( m = 2 \) machines

- **Proof.** Reduction from PARTITION problem.

- We will design an approximation algorithm for this problem

- [Greedy returns!] Consider the following greedy strategy:
  - Fix some order on the jobs
  - Assign job \( j \) to machine \( i \) whose load is smallest so far
Load Balancing: Greedy

**LIST-SCHEDULING** \((m, n, t_1, t_2, \ldots, t_n)\)

For \(i = 1\) to \(m\)

\[
L[i] \leftarrow 0. \quad \text{load on machine } i
\]

\[
S[i] \leftarrow \emptyset. \quad \text{jobs assigned to machine } i
\]

For \(j = 1\) to \(n\)

\[
i \leftarrow \arg\min_{k} L[k]. \quad \text{machine } i \text{ has smallest load}
\]

\[
S[i] \leftarrow S[i] \cup \{j\}. \quad \text{assign job } j \text{ to machine } i
\]

\[
L[i] \leftarrow L[i] + t_j. \quad \text{update load of machine } i
\]

**RETURN** \(S[1], S[2], \ldots, S[m]\).

- Running time?
  - \(O(n \log m)\) using a priority queue for loads \(L[k]\)
Load Balancing: Greedy Analysis

- **Claim.** Greedy algorithm is a 2-approximation.

- To show this, we need to show greedy solution never more than a factor two worse than the optimal

- **Challenge.** We don’t know the optimal solution. In fact, finding the optimal is NP hard.

- Technique used in approximation algorithm (minimization problem)
  - Lower bound the cost of optimal solution
  - A good enough lower bound can help show that our algorithm cannot be too much worse than the optimal

- In our problem, what are some lower bounds on the makespan of even an optimal algorithm?
Load Balancing: Greedy Analysis

• Let OPT be the optimal makespan.

  • **Lemma.** $\text{OPT} \geq \max_j t_j$.

• **Proof.** Some machine must process the most time-consuming job.

• Any other lower bounds?

  • **Lemma.** $\text{OPT} \geq \frac{1}{m} \sum_j t_j$

  • **Proof.**

    • The total processing time is $\sum_j t_j$

    • Some machine must do a $1/m$ fraction of the total work.
Greedy is a 2-Approximation

- **Proof.** Consider load $L[i]$ of bottleneck machine $i$
  - Let $j$ be the last scheduled job on machine $i$
  - When job $j$ was assigned to machine $i$, $i$ must have had the smallest load
  - That is, $L[i] - t_j \leq L[k] \ \forall 1 \leq k \leq m$
Greedy is a 2-Approximation

- **Proof.** Consider load $L(i)$ of bottleneck machine $i$
  - Let $j$ be the last scheduled job on machine $i$
  - When job $j$ was assigned to machine $i$, $i$ must have had the smallest load
  - That is, $L[i] - t_j \leq L[k] \ \forall 1 \leq k \leq m$
  - Summing over all $k$ and diving by $m$
    
    $$L[i] - t_j \leq \frac{1}{m} \sum_{k} L[k]$$
    $$\leq \frac{1}{m} \sum_{k} t_k$$
    $$\leq \text{OPT}$$
Greedy is a 2-Approximation

Proof.

• Consider load $L(i)$ of bottleneck machine $i$

$$L[i] - t_j \leq \frac{1}{m} \sum_k L[k]$$

$$\frac{1}{m} \sum_k t_k \leq OPT$$

• We know that $t_j \leq OPT$

• Thus, $L = L[i] \leq OPT + t_j$

$$\leq 2OPT \qed$$
Greedy is a 2-Approximation

• Is our analysis tight?
• Close to it.
• Consider $m(m-1)$ jobs of length 1 + 1 job of length $m$
• How would greedy schedule these jobs?
  • Greedy will evenly divide the first $m(m-1)$ jobs among $m$ machines, will place the final long job on any one machine
  • Makespan: $m - 1 + m = 2m - 1$
• How would optimal schedule it?
  • Give the long job to one machine, the rest split the other small jobs with a makespan $m$
• Ratio: $(2m - 1)/m \approx 2$
Greedy is Online

- Notice that our greedy algorithm is an online algorithm
- Assigns jobs to machines in the order they arrive
  - Does not depend on future jobs
- Online approximation algorithms are very useful as often the entire input is not known ahead of time
- In online settings, it may be impossible to compute an optimum solution in polynomial time, even when the offline problem is polynomial time solvable
- Can we do better, if we assume all jobs are available at start time?
  - **Offline.** Slight modification of greedy gets better approximation!
Improving on Online Greedy

- Worst case of our greedy algorithm: spreading jobs out evenly when a giant job at the end screwed things up
- What can we do to avoid this?
  - Idea: deal with larger jobs first
  - Small jobs can only hurt so much
- Turns out this improves our approximation factor
- **Longest-processing-time (LPT) first.** Sort $n$ jobs in decreasing order of processing times; then run the greedy algorithm on them
- **Claim.** LPT has a makespan at most $1.5 \cdot \text{OPT}$
- **Observation.** If we have fewer than $m$ jobs, then the greedy solution is clearly optimal (as it puts each job on its own machine)
LPT-first is a 1.5-Approximation

- **Lemma.** LPT-first has a makespan at most $1.5 \cdot \text{OPT}$
- **Observation.**
  - If we have fewer than $m$ jobs, then the greedy solution is clearly optimal (as it puts each job on its own machine)
- **Claim.** If more than $m$ jobs then, $\text{OPT} \geq 2 \cdot t_{m+1}$
- **Proof.** Consider the first $m + 1$ jobs in sorted order.
  - They each take at least $t_{m+1}$ time
  - $m + 1$ jobs and $m$ machines, there must be a machine with at least two jobs
  - Thus the optimal makespan $\text{OPT} \geq 2 \cdot t_{m+1}$
LPT-first is a 1.5-Approximation

- **Lemma.** LPT-first has a makespan at most $1.5 \cdot \text{OPT}$
- **Proof.** Similar to our original proof. Consider the machine $M_i$ that has the maximum load
- If $M_i$ has a single job, then our algorithm is optimal
- Suppose $M_i$ has at least two jobs and let $t_j$ be the last job assigned to the machine, note that $j \geq m + 1$ (why?)
- Thus, $t_j \leq t_{m+1} \leq \frac{1}{2}\text{OPT}$
LPT-first is a 1.5-Approximation

- **Lemma.** LPT-first has a makespan at most $1.5 \cdot \text{OPT}$

- **Proof.** Similar to our original proof. Consider the machine $M_i$ that has the maximum load

  - If $M_i$ has a single job, then our algorithm is optimal

  - Suppose $M_i$ has at least two jobs and let $t_j$ be the last job assigned to the machine, note that $j \geq m + 1$ (why?)

  - Thus, $t_j \leq t_{m+1} \leq \frac{1}{2} \text{OPT}$

  - $T_i - t_j \leq \text{OPT}$

  - $T_i \leq \frac{3}{2} \text{OPT}$

  $\blacksquare$
Is our 1.5-Approximation tight?

- **Question.** Is our 3/2-approximation analysis tight?
  - Turns out, no

- **Theorem [Graham 1969].** LPT-first is a 4/3-approximation.
  - Proof via a more sophisticated analysis of the same algorithm

- **Question.** Is the 4/3-approximation analysis tight?
  - Pretty much.

- Example
  - $m$ machines, $n = 2m + 1$ jobs
  - 2 jobs each of length $m, m + 1, \ldots, 2m - 1$ + one job of length $m$
  - Approximation ratio $= (4m - 1)/3m \approx 4/3$
Formal Definition

- Consider an arbitrary optimization problem
- Let $\text{OPT}(X)$: the cost of the optimal solution on a given input $X$
- Let $A(X)$: the cost of algorithm $A$ on the same input $X$
- $A$ is a $\alpha(n)$-approximation iff
  \[
  \frac{\text{OPT}(X)}{A(X)} \leq \alpha(n) \quad \text{and} \quad \frac{A(X)}{\text{OPT}(X)} \leq \alpha(n)
  \]
  for all input $X$ of size $n$
- **Maximization problem**: second inequality is trivial, first matters
- **Minimization problem**: first inequality is trivial, second matters
- **Goal**: Find a useful function of the input to upper and lower on the cost of $\text{OPT}$ and $A$, e.g. $\text{OPT}(X) \geq f(X)/2$ and $A(X) \leq 4f(X)$ means $A$ is a 8-approximation
Acknowledgments

- Some of the material in these slides are taken from
  - Jeff Erickson’s Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)
  - Lecture slides: https://web.stanford.edu/class/archive/cs/cs161/cs161.1138/