CSE 421

Approximation Alg

Shayan Oveis Gharan
Midterm

Congratulations! You did great in the midterm
Median ~ 70%
• I did very well in the midterm; so I’ll get a 4.0, Yaay! (not really)
  Final is harder and has a significant impact on your final gpa
• I did terrible in midterm, can I still get 3.9 or 4.0? Yes!
• If you are way below median below 50% (midterm grade <35) try harder
• Will I pass this course, Typically only 4/200 may get below 3.0.
  Final will be harder
Problem 2: Alg: Find a cycle C in G. Let e=(u,v) an edge in C. G-e is a tree, and we can color by 2 colors (proved in class). Use the third color for vertex u.

Correctness: G has n vertices and n edges and connected=>G has a cycle C (section 2), Let e=(u,v) an edge in C, G-e is a tree (proved in class). G-e can be colored with 2 colors (proved in class). e is the only non-tree edge and a possible violation so using the third color on v makes sure all adjacent vertices have distinct colors.

P3) Alg: If G has a vertex of deg 0 output no, o.w. output yes

Correctness: If G has a deg 0 vertex, that is always a source and sink so impossible. Otherwise, since all vertices has even degree, G can be partitioned into disjoint cycles (homework 2). Every vertex has deg>=2, so shows up in at least one cycle. Orienting every cycle clockwise every vertex will have indegree,outdegree>=1 so not a source nor a sink.
Possible Wrong answers

Problem 2:
• Delete arbitrary vertex \( v \), color the rest of the graph, then add back \( v \) and color \( v \) with a color not used on neighbors.
• Choose \( v \) with minimum degree, color \( v \) with an available color, then delete \( v \) and color the rest of the graph

Problem 3:
• Same as HW: Orient all edges of the tree away from the root, orient the rest of the edges arbitrarily.
Q/A

• HW problems are too hard for me
  • We have resources to prepare for HW
    • section, OH, …
    • Exercises in the book.
    • USA Olympiad training website: https://train.usaco.org
  • Difficult HW problems prepare you for real world algorithm problems

• Grading rules are too strict
  • Every week I spent hours to train TAs how to grade. The well-defined rubric is my effort to have a systematic grading guidelines that all TAs can follow. Without it everybody grades arbitrarily.
  • Everything is not about grade! We are here to learn.

• TAs have not responded to my re-grade requests
  • TAs are also humans; give them sometime.
  • Send me an email or come to OH, I’ll look into your request

• What is the point of this course after all? Why do you have to prove correctness of an algorithm?
  • Often algorithms that we design are incorrect.
Approximation Alg Summary

• To design approximation Alg, always find a way to lower bound OPT.

• The best known approximation Alg for vertex cover is the greedy.
  – It has been open for 50 years to obtain a polynomial time algorithm with approximation ratio better than 2.

• The best known approximation Alg for set cover is the greedy.
  – It is NP-Complete to obtain better than \ln n approximation ratio for set cover.
Dynamic Programming
Algorithmic Paradigm

**Greedy:** Build up a solution incrementally, myopically optimizing some local criterion.

**Divide-and-conquer:** Break up a problem into two sub-problems, solve each sub-problem independently, and combine solution to sub-problems to form solution to original problem.

**Dynamic programming.** Break up a problem into a series of overlapping sub-problems, and build up solutions to larger and larger sub-problems. **Memorize** the answers to obtain polynomial time ALG.
Dynamic Programming History

Bellman. Pioneered the systematic study of dynamic programming in the 1950s.

Etymology.

Dynamic programming = planning over time.

Secretary of Defense was hostile to mathematical research.

Bellman sought an impressive name to avoid confrontation.

• "it's impossible to use dynamic in a pejorative sense"
• "something not even a Congressman could object to"
Dynamic Programming Applications

Areas:
- Bioinformatics
- Control Theory
- Information Theory
- Operations Research
- Computer Science: Theory, Graphics, AI, …

Some famous DP algorithms
- Viterbi for hidden Markov Model
- Unix diff for comparing two files.
- Smith-Waterman for sequence alignment.
- Bellman-Ford for shortest path routing in networks.
- Cocke-Kasami-Younger for parsing context free grammars.
Dynamic programming is nothing but algorithm design by induction!

We just "remember" the subproblems that we have solved so far to avoid re-solving the same sub-problem many times.
Weighted Interval Scheduling
Interval Scheduling

- Job j starts at $s(j)$ and finishes at $f(j)$ and has weight $w_j$
- Two jobs compatible if they don’t overlap.
- Goal: find maximum weight subset of mutually compatible jobs.

![Diagram of Interval Scheduling]
Unweighted Interval Scheduling: Review

Recall: Greedy algorithm works if all weights are 1:
• Consider jobs in ascending order of finishing time
• Add job to a subset if it is compatible with prev added jobs.

OBS: Greedy ALG fails spectacularly (no approximation ratio) if arbitrary weights are allowed:

![Diagram showing the impact of weight on scheduling decisions.](image-url)
Weighted Job Scheduling by Induction

Suppose $1, \ldots, n$ are all jobs. Let us use induction:

**IH (strong ind):** Suppose we can compute the optimum job scheduling for $< n$ jobs.

**IS: Goal:** For any $n$ jobs we can compute $OPT$.

**Case 1:** Job $n$ is not in $OPT$.
-- Then, just return $OPT$ of $1, \ldots, n-1$.

**Case 2:** Job $n$ is in $OPT$.
-- Then, delete all jobs not compatible with $n$ and recurse.

Q: Are we done?
A: No, How many subproblems are there? Potentially $2^n$ all possible subsets of jobs.
A Bad Example

Consider jobs n/2+1,…,n. These decisions have no impact on one another.

How many subproblems do we get?

Time

1
n/2+1
2
n/2+2
3
n/2+3

n/2
n
Sorting to Reduce Subproblems

**IS:** For jobs 1,…,n we want to compute OPT

**Sorting Idea:** Label jobs by finishing time $f(1) \leq \cdots \leq f(n)$

**Case 1:** Suppose OPT has job n.
- So, all jobs $i$ that are not compatible with n are not OPT
- Let $p(n) =$ largest index $i < n$ such that job $i$ is compatible with n.
- Then, we just need to find OPT of 1, ..., $p(n)$

```
1

P(n)
P(n)+1

\ldots

n-2
n-1
n
```
Sorting to reduce Subproblems

IS: For jobs 1,...,n we want to compute OPT

Sorting Idea: Label jobs by finishing time \( f(1) \leq \cdots \leq f(n) \)

Case 1: Suppose OPT has job n.
- So, all jobs \( i \) that are not compatible with n are not OPT
- Let \( p(n) = \) largest index \( i < n \) such that job \( i \) is compatible with n.
- Then, we just need to find OPT of 1, 2, ..., \( p(n) \)

Case 2: OPT does not select job n.
- Then, OPT is just the optimum 1, ..., \( n - 1 \)

Q: Have we made any progress (still reducing to two subproblems)?
A: Yes! This time every subproblem is of the form 1, ..., \( i \) for some \( i \)
So, at most \( n \) possible subproblems.
Bad Example Review

How many subproblems do we get in this sorted order?
Weighted Job Scheduling by Induction

Sorting Idea: Label jobs by finishing time $f(1) \leq \cdots \leq f(n)$

Let $\text{OPT}(j)$ denote the OPT solution of $1, \ldots, j$

To solve $\text{OPT}(j)$:

Case 1: $\text{OPT}(j)$ has job $j$.

• So, all jobs $i$ that are not compatible with $j$ are not $\text{OPT}(j)$
• Let $p(j) =$ largest index $i$ such that job $i$ is compatible with $j$.
• So $\text{OPT}(j) = \text{OPT}(p(j)) \cup \{j\}$.

Case 2: $\text{OPT}(j)$ does not select job $j$.

• Then, $\text{OPT}(j) = \text{OPT}(j - 1)$

\[
\text{OPT}(j) = \begin{cases} 
0 & \text{if } j = 0 \\
\max \left( w_j + \text{OPT}(p(j)), \text{OPT}(j - 1) \right) & \text{o. w.}
\end{cases}
\]
**Algorithm**

**Input:** $n$, $s(1),\ldots,s(n)$ and $f(1),\ldots,f(n)$ and $w_1,\ldots,w_n$.

Sort jobs by finish times so that $f(1) \leq f(2) \leq \cdots f(n)$.

Compute $p(1), p(2), \ldots, p(n)$

Compute-Opt$(j)$ {
   if $(j = 0)$
      return 0
   else
      return max($w_j + $Compute-Opt$(p(j))$, $Compute-Opt(j-1)$)
}
Recursive Algorithm Fails

Even though we have only $n$ subproblems, we do not store the solution to the subproblems.

- So, we may re-solve the same problem many many times.

**Ex.** Number of recursive calls for family of "layered" instances grows like Fibonacci sequence.

\[ p(1) = 0, \quad p(j) = j - 2 \]
Algorithm with Memoization

**Memoization.** Compute and Store the solution of each sub-problem in a cache the first time that you face it. lookup as needed.

**Input:** \( n, s(1), \ldots, s(n) \) and \( f(1), \ldots, f(n) \) and \( w_1, \ldots, w_n \).

Sort jobs by finish times so that \( f(1) \leq f(2) \leq \cdots f(n) \).

Compute \( p(1), p(2), \ldots, p(n) \)

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

\[ M[j] = \text{empty} \]
\[ M[0] = 0 \]

\( M-\text{Compute-Opt}(j) \) {
  \text{if (} M[j] \text{ is empty)}
  \[ M[j] = \max(w_j + M-\text{Compute-Opt}(p(j)), M-\text{Compute-Opt}(j-1)) \]
  return \( M[j] \)
}

You can also avoid recursion

- recursion may be easier conceptually when you use induction

**Input**: \( n, \ s(1), \ldots, s(n) \) and \( f(1), \ldots, f(n) \) and \( w_1, \ldots, w_n \).

**Sort** jobs by finish times so that \( f(1) \leq f(2) \leq \ldots f(n) \).

**Compute** \( p(1), p(2), \ldots, p(n) \)

**Iterative-Compute-Opt**

\[
\begin{align*}
M[0] &= 0 \\
\text{for } j &= 1 \text{ to } n \\
M[j] &= \max(w_j + M[p(j)], M[j-1])
\end{align*}
\]

Output \( M[n] \)

**Claim**: \( M[j] \) is value of \( \text{OPT}(j) \)

**Timing**: Easy. Main loop is \( O(n) \); sorting is \( O(n \log n) \)
Example

Label jobs by finishing time: \( f(1) \leq \cdots \leq f(n) \).

\( p(j) = \) largest index \( i < j \) such that job \( i \) is compatible with \( j \).
Example

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.

$p(j) = \text{largest index } i < j \text{ such that job } i \text{ is compatible with } j.$

<table>
<thead>
<tr>
<th>j</th>
<th>$w_j$</th>
<th>p(j)</th>
<th>OPT(j)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Example

Label jobs by finishing time: \( f(1) \leq \cdots \leq f(n) \).

\( p(j) = \) largest index \( i < j \) such that job \( i \) is compatible with \( j \).

\begin{tabular}{|c|c|c|c|}
\hline
\( j \) & \( w_j \) & \( p(j) \) & \( \text{OPT}(j) \) \\
\hline
0 & & & 0 \\
1 & 3 & 0 & 3 \\
2 & 4 & 0 & 4 \\
3 & 1 & 0 & & \\
4 & 3 & 1 & & \\
5 & 4 & 0 & & \\
6 & 3 & 2 & & \\
7 & 2 & 3 & & \\
8 & 4 & 5 & & \\
\hline
\end{tabular}
Example

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.
p(j) = largest index $i < j$ such that job $i$ is compatible with $j$. 

<table>
<thead>
<tr>
<th>j</th>
<th>$w_j$</th>
<th>p(j)</th>
<th>OPT(j)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Example

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.

$p(j) =$ largest index $i < j$ such that job $i$ is compatible with $j$. 

<table>
<thead>
<tr>
<th>$j$</th>
<th>$w_j$</th>
<th>$p(j)$</th>
<th>$\text{OPT}(j)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Example

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.

$p(j) =$ largest index $i < j$ such that job $i$ is compatible with $j$. 

<table>
<thead>
<tr>
<th>$j$</th>
<th>$w_j$</th>
<th>$p(j)$</th>
<th>OPT($j$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Example

Label jobs by finishing time: \( f(1) \leq \cdots \leq f(n) \).

\( p(j) = \) largest index \( i < j \) such that job \( i \) is compatible with \( j \).

<table>
<thead>
<tr>
<th>( j )</th>
<th>( w_j )</th>
<th>( p(j) )</th>
<th>( \text{OPT}(j) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.

$p(j) =$ largest index $i < j$ such that job $i$ is compatible with $j$.

<table>
<thead>
<tr>
<th>j</th>
<th>$w_j$</th>
<th>p(j)</th>
<th>OPT(j)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Example

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.
p(j) = largest index $i < j$ such that job $i$ is compatible with $j$.

<table>
<thead>
<tr>
<th>j</th>
<th>$w_j$</th>
<th>p(j)</th>
<th>OPT(j)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>\emptyset</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>
Example

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.
$p(j) =$ largest index $i < j$ such that job $i$ is compatible with $j$.

<table>
<thead>
<tr>
<th>$j$</th>
<th>$w_j$</th>
<th>$p(j)$</th>
<th>OPT($j$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>