CSC373

Week 2: Greedy Algorithms

Nisarg Shah
Recap

• Divide & Conquer
  - Master theorem
  - Counting inversions in $O(n \log n)$
  - Finding closest pair of points in $\mathbb{R}^2$ in $O(n \log n)$
  - Fast integer multiplication in $O(n^{\log_2 3})$
  - Fast matrix multiplication in $O(n^{\log_2 7})$
  - Finding $k^{th}$ smallest element (in particular, median) in $O(n)$
Greedy Algorithms

- **Greedy (also known as myopic) algorithm outline**
  - We want to find a solution $x$ that maximizes some objective function $f$
  - But the space of possible solutions $x$ is too large
  - The solution $x$ is typically composed of several parts (e.g. $x$ may be a set, composed of its elements)
  - Instead of directly computing $x$...
    - Compute it one part at a time
    - Select the next part “greedily” to get maximum immediate benefit (this needs to be defined carefully for each problem)
    - May not be optimal because there is no foresight
    - But sometimes this can be optimal too!
Interval Scheduling

• Problem
  ➢ Job $j$ starts at time $s_j$ and finishes at time $f_j$
  ➢ Two jobs are compatible if they don’t overlap
  ➢ Goal: find maximum-size subset of mutually compatible jobs
Interval Scheduling

• **Greedy template**
  - Consider jobs in some “natural” order
  - Take each job if it’s compatible with the ones already chosen

• **What order?**
  - Earliest start time: ascending order of $s_j$
  - Earliest finish time: ascending order of $f_j$
  - Shortest interval: ascending order of $f_j - s_j$
  - Fewest conflicts: ascending order of $c_j$, where $c_j$ is the number of remaining jobs that conflict with $j$
Example

- Earliest start time: ascending order of $s_j$
- Earliest finish time: ascending order of $f_j$
- Shortest interval: ascending order of $f_j - s_j$
- Fewest conflicts: ascending order of $c_j$, where $c_j$ is the number of remaining jobs that conflict with $j$
Interval Scheduling

- Does it work?

  Counterexamples for

  - earliest start time
  - shortest interval
  - fewest conflicts
Interval Scheduling

• Implementing greedy with earliest finish time (EFT)

➢ Sort jobs by finish time. Say $f_1 \leq f_2 \leq \cdots \leq f_n$

➢ When deciding on job $j$, we need to check whether it’s compatible with all previously added jobs

  o We only need to check if $s_j \geq f_{i^*}$, where $i^*$ is the last added job

  o This is because for any jobs $i$ added before $i^*$, $f_i \leq f_{i^*}$

  o So we can simply keep track of the finish time of the last added job

➢ Running time: $O(n \log n)$
Interval Scheduling

- Optimality of greedy with EFT
  - Suppose for contradiction that greedy is not optimal
  - Say greedy selects jobs $i_1, i_2, \ldots, i_k$ sorted by finish time
  - Consider the optimal solution $j_1, j_2, \ldots, j_m$ (also sorted by finish time) which matches greedy for as long as possible
    - That is, we want $j_1 = i_1, \ldots, j_r = i_r$ for greatest possible $r$
Interval Scheduling

• Optimality of greedy with EFT
  ➢ Both $i_{r+1}$ and $j_{r+1}$ were compatible with the previous selection ($i_1 = j_1, \ldots, i_r = j_r$)
  ➢ Consider the solution $i_1, i_2, \ldots, i_r, i_{r+1}, j_{r+2}, \ldots, j_m$
    o It should still be feasible (since $f_{i_{r+1}} \leq f_{j_{r+1}}$)
    o It is still optimal
    o And it matches with greedy for one more step (contradiction!)

Another standard method is induction
• **Problem**
  - Job $j$ starts at time $s_j$ and finishes at time $f_j$
  - Two jobs are compatible if they don’t overlap
  - **Goal:** group jobs into fewest partitions such that jobs in the same partition are compatible

• **One idea**
  - Find the maximum compatible set using the previous greedy EFT algorithm, call it one partition, recurse on the remaining jobs.
  - Doesn’t work (check by yourselves)
Interval Partitioning

• Think of scheduling lectures for various courses into as few classrooms as possible

• This schedule uses 4 classrooms for scheduling 10 lectures
Interval Partitioning

- Think of scheduling lectures for various courses into as few classrooms as possible

- This schedule uses 3 classrooms for scheduling 10 lectures
Interval Partitioning

• Let’s go back to the **greedy template**!
  ➢ Go through lectures in some “natural” order
  ➢ Assign each lecture to an *(arbitrary?)* compatible classroom, and create a new classroom if the lecture conflicts with every existing classroom

• **Order of lectures?**
  ➢ Earliest start time: ascending order of \( s_j \)
  ➢ Earliest finish time: ascending order of \( f_j \)
  ➢ Shortest interval: ascending order of \( f_j - s_j \)
  ➢ Fewest conflicts: ascending order of \( c_j \), where \( c_j \) is the number of remaining jobs that conflict with \( j \)
Interval Partitioning

- At least when you assign each lecture to an arbitrary compatible classroom, three of these heuristics do not work.

- The fourth one works! (next slide)
Interval Partitioning

\[ \text{EARLIEST\_START\_TIME\_FIRST}(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n) \]

\[ \text{SORT lectures by start time so that } s_1 \leq s_2 \leq \ldots \leq s_n. \]

\[ d \leftarrow 0 \quad \text{number of allocated classrooms} \]

\[ \text{FOR } j = 1 \text{ TO } n \]

\[ \text{IF lecture } j \text{ is compatible with some classroom} \]

\[ \text{Schedule lecture } j \text{ in any such classroom } k. \]

\[ \text{ELSE} \]

\[ \text{Allocate a new classroom } d + 1. \]

\[ \text{Schedule lecture } j \text{ in classroom } d + 1. \]

\[ d \leftarrow d + 1 \]

\[ \text{RETURN schedule.} \]
Interval Partitioning

- **Running time**
  - **Key step:** check if the next lecture can be scheduled at some classroom
  - Store classrooms in a priority queue
    - key = latest finish time of any lecture in the classroom
  - Is lecture $j$ compatible with some classroom?
    - Same as “Is $s_j$ at least as large as the minimum key?”
    - If yes: add lecture $j$ to classroom $k$ with minimum key, and increase its key to $f_j$
    - Otherwise: create a new classroom, add lecture $j$, set key to $f_j$
  - $O(n)$ priority queue operations, $O(n \log n)$ time
Interval Partitioning

• Proof of optimality (lower bound)
  ➢ # classrooms needed ≥ maximum “depth” at any point
    ▪ depth = number of lectures running at that time
  ➢ We now show that our greedy algorithm uses only these many classrooms!
Interval Partitioning

• Proof of optimality (upper bound)
  ➢ Let $d = \# \text{classrooms used by greedy}$
  ➢ Classroom $d$ was opened because there was a schedule $j$ which was incompatible with some lectures already scheduled in each of $d - 1$ other classrooms
  ➢ All these $d$ lectures end after $s_j$
  ➢ *Since we sorted by start time*, they all start at/before $s_j$
  ➢ So at time $s_j$, we have $d$ mutually overlapping lectures
  ➢ Hence, depth $\geq d$
  ➢ So all schedules use $\geq d$ classrooms. ■
Interval Graphs

• Interval scheduling and interval partitioning can be seen as graph problems

• Input
  ➢ Graph $G = (V, E)$
  ➢ Vertices $V =$ jobs/lectures
  ➢ Edge $(i, j) \in E$ if jobs $i$ and $j$ are incompatible

• Interval scheduling = maximum independent set (MIS)

• Interval partitioning = graph colouring
Interval Graphs

• MIS and graph colouring are NP-hard for general graphs

• But they’re efficiently solvable for “interval graphs”
  - Graphs which can be obtained from incompatibility of intervals
  - In fact, this holds even when we are not given an interval representation of the graph

• Can we extend this result further?
  - Yes! Chordal graphs
    - Every cycle with 4 or more vertices has a chord
Minimizing Lateness

• Problem
  ➢ We have a single machine
  ➢ Each job $j$ requires $t_j$ units of time and is due by time $d_j$
  ➢ If it’s scheduled to start at $s_j$, it will finish at $f_j = s_j + t_j$
  ➢ Lateness: $\ell_j = \max\{0, f_j - d_j\}$
  ➢ Goal: minimize the maximum lateness, $L = \max_j \ell_j$

• Contrast with interval scheduling
  ➢ We can decide the start time
  ➢ There are soft deadlines
Minimizing Lateness

- Example

Input

An example schedule

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$d_j$</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

Lateness = 2

Lateness = 0

Max lateness = 6
Minimizing Lateness

• Let’s go back to greedy template
  ➢ Consider jobs one-by-one in some “natural” order
  ➢ Schedule jobs in this order (nothing special to do here, since we have to schedule all jobs and there is only one machine available)

• Natural orders?
  ➢ Shortest processing time first: ascending order of processing time $t_j$
  ➢ Earliest deadline first: ascending order of due time $d_j$
  ➢ Smallest slack first: ascending order of $d_j - t_j$
Minimizing Lateness

• Counterexamples

- Shortest processing time first
  - Ascending order of processing time $t_j$

- Smallest slack first
  - Ascending order of $d_j - t_j$
Minimizing Lateness

• By now, you should know what’s coming...

• We’ll prove that earliest deadline first works!

\[
\text{EarliestDeadlineFirst}(n, t_1, t_2, \ldots, t_n, d_1, d_2, \ldots, d_n)
\]

Sort n jobs so that \(d_1 \leq d_2 \leq \ldots \leq d_n\).
\[t \leftarrow 0\]
\[
\text{For } j = 1 \text{ to } n
\]
Assign job \(j\) to interval \([t, t + t_j]\).
\[s_j \leftarrow t ; \quad f_j \leftarrow t + t_j\]
\[t \leftarrow t + t_j\]
\[
\text{Return}\, \text{intervals } [s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n].
\]
Minimizing Lateness

• Observation 1
  ➢ There is an optimal schedule with *no idle time*
Minimizing Lateness

• Observation 2
  ➢ Earliest deadline first has no idle time

• Let us define an “inversion”
  ➢ \((i, j)\) such that \(d_i < d_j\) but \(j\) is scheduled before \(i\)

• Observation 3
  ➢ By definition, earliest deadline first has no inversions

• Observation 4
  ➢ If a schedule with no idle time has an inversion, it has a pair of inverted jobs scheduled consecutively
Minimizing Lateness

• Observation 5
  ➢ Swapping adjacently scheduled inverted jobs doesn’t increase lateness but reduces #inversions by one

• Proof
  ➢ Let $L$ and $L'$ denote lateness before/after swap
  ➢ Clearly, $\ell_k = \ell'_k$ for all $k \neq i, j$
  ➢ Also, clearly, $\ell'_i \leq \ell_i$
Minimizing Lateness

• Observation 5
  ➢ Swapping adjacently scheduled inverted jobs doesn’t increase lateness but reduces #inversions by one

• Proof
  ➢ $\ell_j' = f_j' - d_j = f_i - d_j \leq f_i - d_i = \ell_i$
  ➢ $L' = \max \{\ell_i', \ell_j', \max_k \ell_k'\} \leq \max \{\ell_i, \ell_i, \max_k \ell_k\} \leq L$
Minimizing Lateness

• Observation 5
  ➢ Swapping adjacently scheduled inverted jobs doesn’t increase lateness but reduces #inversions by one

• Proof
  ➢ Check that swapping an adjacent inverted pair reduces the total #inversions by one
Minimizing Lateness

• Proof of optimality of earliest deadline first
  ➢ Suppose for contradiction that it’s not optimal
  ➢ Consider an optimal schedule $S^*$ with fewest inversions among all optimal schedules
    o WLOG, suppose it has no idle time
  ➢ Because EDF is not optimal, $S^*$ has inversions
  ➢ By Observation 4, it has an adjacent inversion $(i, j)$
  ➢ By Observation 5, swapping the adjacent pair keeps the schedule optimal but reduces the #inversions by 1
  ➢ Contradiction! ■
Lossless Compression

- **Problem**
  - We have a document that is written using $n$ distinct labels
  - Naïve encoding: represent each label using $\log n$ bits
  - If the document has length $m$, this uses $m \log n$ bits

- English document with no punctuations etc.
- $n = 26$, so we can use 5 bits
  - $a = 00000$
  - $b = 00001$
  - $c = 00010$
  - $d = 00011$
  - ...
Lossless Compression

• Is this optimal?
  ➢ What if \( a, e, r, s \) are much more frequent in the document than \( x, q, z \)?
  ➢ Can we assign shorter codes to more frequent letters?

• Say we assign...
  ➢ \( a = 0, b = 1, c = 01, \ldots \)
  ➢ See a problem?
    o What if we observe the encoding ‘01’?
    o Is it ‘ab’? Or is it ‘c’?
Lossless Compression

• To avoid conflicts, we need a *prefix-free encoding*
  - Map each label \( x \) to a bit-string \( c(x) \) such that for all distinct labels \( x \) and \( y \), \( c(x) \) is not a prefix of \( c(y) \)
  - Then it’s impossible to have a scenario like this

```
............................
     ||
  c(x)  c(y)
     ||
```

• So we can read left to right
  - Whenever the part to the left becomes a valid encoding, greedily decode it, and continue with the rest
Lossless Compression

• Formal problem
  ➢ Given $n$ symbols and their frequencies $(w_1, \ldots, w_n)$, find a prefix-free encoding with lengths $(\ell_1, \ldots, \ell_n)$ assigned to the symbols which minimizes $\sum_{i=1}^{n} w_i \cdot \ell_i$
  ○ Note that $\sum_{i=1}^{n} w_i \cdot \ell_i$ is the length of the compressed document

• Example
  ➢ $(w_a, w_b, w_c, w_d, w_e, w_f) = (42, 20, 5, 10, 11, 12)$
  ➢ No need to remember the numbers 😊
Lossless Compression

• Observation: prefix-free encoding = tree

\[ a \rightarrow 0, e \rightarrow 100, \\
 f \rightarrow 101, c \rightarrow 1100, \\
 d \rightarrow 1101, b \rightarrow 111 \]
Lossless Compression

• **Huffman Coding**
  - Build a priority queue by adding \((x, w_x)\) for each symbol \(x\)
  - While \(|\text{queue}| \geq 2\)
    - Take the two symbols with the lowest weight \((x, w_x)\) and \((y, w_y)\)
    - Merge them into one symbol with weight \(w_x + w_y\)

• Let’s see this on the previous example
Lossless Compression

c: 5  d: 10  e: 11  f: 12  b: 20  a: 42

e: 11  f: 12

c: 5  d: 10

15

b: 20  a: 42
Lossless Compression
Lossless Compression
Lossless Compression
Lossless Compression

![Tree Diagram]

- a: 42
- e: 11
- f: 12
- c: 5
- d: 10
- b: 20
- 23
- 15
- 35
- 58
Lossless Compression

- Final Outcome

\[ a \rightarrow 0, \quad e \rightarrow 100, \]
\[ f \rightarrow 101, \quad c \rightarrow 1100, \]
\[ d \rightarrow 1101, \quad b \rightarrow 111 \]
Lossless Compression

• **Running time**
  - $O(n \log n)$
  - Can be made $O(n)$ if the labels are given to you sorted by their frequencies
    - Exercise! Think of using two queues...

• **Proof of optimality**
  - Induction on the number of symbols $n$
  - **Base case:** For $n = 2$, both encodings which assign 1 bit to each symbol are optimal
  - **Hypothesis:** Assume it returns an optimal encoding with $n - 1$ symbols
Lossless Compression

• Proof of optimality
  ➢ Consider the case of $n$ symbols

  ➢ Lemma 1: If $w_x < w_y$, then $\ell_x \geq \ell_y$ in any optimal tree.

  ➢ Proof:
    o Suppose for contradiction that $w_x < w_y$ and $\ell_x < \ell_y$.
    o Swapping $x$ and $y$ strictly reduces the overall length as $w_x \cdot \ell_y + w_y \cdot \ell_x < w_x \cdot \ell_x + w_y \cdot \ell_y$ (check!)
    o QED!
Lossless Compression

• **Proof of optimality**
  - Consider the two symbols $x$ and $y$ with lowest frequency which Huffman combines in the first step
  - **Lemma 2:** $\exists$ optimal tree $T$ in which $x$ and $y$ are siblings (i.e. for some $p$, they are assigned encodings $p0$ and $p1$).
  - **Proof:**
    1. Take any optimal tree
    2. Let $x$ be the label with the lowest frequency.
    3. If $x$ doesn’t have the longest encoding, swap it with one that has
    4. Due to optimality, $x$ must have a sibling (check!)
    5. If it’s not $y$, swap it with $y$
    6. Check that Steps 3 and 5 do not change the overall length. ■
Lossless Compression

• **Proof of optimality**
  - Let $x$ and $y$ be the two least frequency symbols that Huffman combines in the first step into “$xy$”
  - Let $H$ be the Huffman tree produced
  - Let $T$ be an optimal tree in which $x$ and $y$ are siblings
  - Let $H'$ and $T'$ be obtained from $H$ and $T$ by treating $xy$ as one symbol with frequency $w_x + w_y$
  - Induction hypothesis: $\text{Length}(H') \leq \text{Length}(T')$
  - $\text{Length}(H) = \text{Length}(H') + (w_x + w_y) \cdot 1$
  - $\text{Length}(T) = \text{Length}(T') + (w_x + w_y) \cdot 1$
  - So $\text{Length}(H) \leq \text{Length}(T)$ ■
Other Greedy Algorithms

• If you aren’t familiar with the following algorithms, spend some time checking them out!
  ➢ Dijkstra’s shortest path algorithm
  ➢ Kruskal and Prim’s minimum spanning tree algorithms