Chapter 2
Greedy Algorithms

Algorithm Theory
WS 2017/18

Fabian Kuhn
Greedy Algorithms

• No clear definition, but essentially:

  In each step make the choice that looks best at the moment!

• Depending on problem, greedy algorithms can give
  – Optimal solutions
  – Close to optimal solutions
  – No (reasonable) solutions at all

• If it works, very interesting approach!
  – And we might even learn something about the structure of the problem

Goal: Improve understanding where it works (mostly by examples)
Interval Scheduling

- **Given:** Set of intervals, e.g. 
  
  \([0,10],[1,3],[1,4],[3,5],[4,7],[5,8],[5,12],[7,9],[9,12],[8,10],[11,14],[12,14]\)

- **Goal:** Select largest possible non-overlapping set of intervals
  
  - For simplicity: overlap at boundary ok  
    (i.e., \([4,7]\) and \([7,9]\) are non-overlapping)

- **Example:** Intervals are room requests; satisfy as many as possible
Greedy Algorithms

• Several possibilities...

Choose first available interval:

Choose shortest available interval:
Greedy Algorithms

Choose available request with earliest finishing time:

\[ R := \text{set of all requests}; \quad S := \text{empty set}; \]

while \( R \) is not empty do
    choose \( r \in R \) with smallest finishing time
    add \( r \) to \( S \)
    delete all requests from \( R \) that are not compatible with \( r \)
end

// \( S \) is the solution
Earliest Finishing Time is Optimal

• Let $O$ be the set of intervals of an optimal solution

• Can we show that $S = O$?
  – No…

• Show that $S = O$.

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

```
[0,10] [1,3] [4,7] [7,9] [9,12] [1,4] [5,8] [8,10] [12,14] [3,5] [5,12]
```

Greey Solution Alternative Optimal Sol.

• Show that $|S| = |O|$. 
Greedy Stays Ahead

- Greedy solution $S$: 
  \[ [a_1, b_1], [a_2, b_2], \ldots, [a_{|S|}, b_{|S|}], \text{ where } b_i \leq a_{i+1} \]

- Some optimal solution $O$: 
  \[ [a_1^*, b_1^*], [a_2^*, b_2^*], \ldots, [a_{|O|}^*, b_{|O|}^*], \text{ where } b_i^* \leq a_{i+1}^* \]

- Define $b_i := \infty$ for $i > |S|$ and $b_i^* := \infty$ for $i > |O|$ 

Claim: For all $i \geq 1$, $b_i \leq b_i^*$
Greedy Stays Ahead

Claim: For all $i \geq 1$, $b_i \leq b_i^*$

Proof (by induction on $i$):

Corollary: Earliest finishing time algorithm is optimal.
Weighted Interval Scheduling

Weighted version of the problem:
• Each interval has a weight
• Goal: Non-overlapping set with maximum total weight

Earliest finishing time greedy algorithm fails:
• Algorithm needs to look at weights
• Else, the selected sets could be the ones with smallest weight...

No simple greedy algorithm:
• We will see an algorithm using another design technique later.
Interval Partitioning

• **Schedule all intervals**: Partition intervals into as few as possible non-overlapping sets of intervals
  – Assign intervals to different resources, where each resource needs to get a non-overlapping set

• Example:
  – Intervals are requests to use some room during this time
  – Assign all requests to some room such that there are no conflicts
  – Use as few rooms as possible

• Assignment to 3 resources:

  - [1,3]  
  - [4,7]  
  - [9,12]

  - [1,4]  
  - [5,8]  
  - [9,11]  
  - [12,14]

  - [2,4]  
  - [5,12]
Depth

Depth of a set of intervals:
- Maximum number passing over a single point in time
- Depth of initial example is 4 (e.g., [0,10],[4,7],[5,8],[5,12]):

![Diagram showing intervals and depth]

Lemma: Number of resources needed $\geq$ depth
Greedy Algorithm

Can we achieve a partition into “depth” non-overlapping sets?

• Would mean that the only obstacles to partitioning are local...

Algorithm:
• Assign labels 1, ..., to the intervals; same label → non-overlapping

1. sort intervals by starting time: \( I_1, I_2, ..., I_n \)
2. \textbf{for} \( i = 1 \) \textbf{to} \( n \) \textbf{do}
3. assign smallest possible label to \( I_i \)
   (possible label: different from conflicting intervals \( I_j, j < i \))
4. \textbf{end}
Interval Partitioning Algorithm

Example:

• Labels:

  - Number of labels = depth = 4

  **Diagram:**

[Visual representation of the interval partitioning algorithm with labeled intervals]

• Number of labels = depth = 4
**Interval Partitioning: Analysis**

**Theorem:**

a) Let $d$ be the depth of the given set of intervals. The algorithm assigns a label from $1, \ldots, d$ to each interval.

b) Sets with the same label are non-overlapping

**Proof:**

• b) holds by construction

• For a):
  
  – All intervals $I_j, j < i$ overlapping with $I_i$, overlap at the beginning of $I_i$

  – At most $d - 1$ such intervals $\implies$ some label in $\{1, \ldots, d\}$ is available.
Traveling Salesperson Problem (TSP)

Input:
• Set $V$ of $n$ nodes (points, cities, locations, sites)
• Distance function $d: V \times V \rightarrow \mathbb{R}$, i.e., $d(u, v)$: dist. from $u$ to $v$
• Distances usually symmetric, asymm. distances $\rightarrow$ asymm. TSP

Solution:
• Ordering/permutation $v_1, v_2, \ldots, v_n$ of nodes
• Length of TSP path: $\sum_{i=1}^{n-1} d(v_i, v_{i+1})$
• Length of TSP tour: $d(v_n, v_1) + \sum_{i=1}^{n-1} d(v_i, v_{i+1})$

Goal:
• Minimize length of TSP path or TSP tour
Example

Optimal Tour:
Length: 86

Greedy Algorithm?
Length: 121
Nearest Neighbor (Greedy)

- Nearest neighbor can be arbitrarily bad, even for TSP paths
TSP Variants

• Asymmetric TSP
  – arbitrary non-negative distance/cost function
  – most general, nearest neighbor arbitrarily bad
  – NP-hard to get within any bound of optimum

• Symmetric TSP
  – arbitrary non-negative distance/cost function
  – nearest neighbor arbitrarily bad
  – NP-hard to get within any bound of optimum

• Metric TSP
  – distance function defines metric space: symmetric, non-negative, triangle inequality: \( d(u, v) \leq d(u, w) + d(w, v) \)
  – possible to get close to optimum (we will later see factor \( \frac{3}{2} \))
  – what about the nearest neighbor algorithm?
Metric TSP, Nearest Neighbor

Optimal TSP tour:

Nearest-Neighbor TSP tour:
Metric TSP, Nearest Neighbor

Optimal TSP tour:

Nearest-Neighbor TSP tour:
cost = 24
Metric TSP, Nearest Neighbor

Triangle Inequality:

optimal tour on remaining nodes \( \leq \) overall optimal tour
Metric TSP, Nearest Neighbor

Analysis works in **phases**:

- In each phase, assign each optimal edge to some greedy edge
  - Cost of greedy edge \( \leq \) cost of optimal edge
- Each greedy edge gets assigned \( \leq 2 \) optimal edges
  - At least half of the greedy edges get assigned
- At end of phase:
  Remove points for which greedy edge is assigned
  Consider optimal solution for remaining points

- **Triangle inequality**: remaining opt. solution \( \leq \) overall opt. sol.

- Cost of greedy edges assigned in **each phase** \( \leq \) opt. cost
- **Number of phases** \( \leq \log_2 n \)
  - +1 for last greedy edge in tour
Metric TSP, Nearest Neighbor

- Assume:
  \[ \text{NN: cost of greedy tour, \quad \text{OPT: cost of optimal tour}} \]

- We have shown:
  \[ \frac{\text{NN}}{\text{OPT}} \leq 1 + \log_2 n \]

- Example of an approximation algorithm

- We will later see a \( \frac{3}{2} \)-approximation algorithm for metric TSP
Back to Scheduling

• Given: $n$ requests / jobs with deadlines:

| length $t_1 = 10$ | deadline $d_1 = 11$
|-------------------|---------------------
| $t_2 = 7$         | $d_2 = 10$          
| $t_3 = 3$         | $d_3 = 13$          
| $t_4 = 5$         | $d_4 = 7$          

• Goal: schedule all jobs with minimum lateness $L$
  
  – Schedule: $s(i), f(i)$: start and finishing times of request $i$
  
  Note: $f(i) = s(i) + t_i$

• Lateness $L := \max \{0, \max_i \{f(i) - d_i\}\}$
  
  – largest amount of time by which some job finishes late

• Many other natural objective functions possible...
Greedy Algorithm?

Schedule jobs in order of increasing length?

• Ignores deadlines: seems too simplistic...
• E.g.:

\[
\begin{align*}
t_1 &= 10 \\
t_2 &= 2
\end{align*}
\]
deadline \( d_1 = 10 \)

Schedule: \( t_2 = 2 \) \( t_1 = 10 \)

Schedule by increasing slack time?

• Should be concerned about slack time: \( d_i - t_i \)

\[
\begin{align*}
t_1 &= 10 \\
t_2 &= 2 \quad d_2 = 3
\end{align*}
\]
deadline \( d_1 = 10 \)

Schedule: \( t_1 = 10 \) \( t_2 = 2 \)
Greedy Algorithm

Schedule by earliest deadline?

- Schedule in increasing order of \( d_i \)
- Ignores lengths of jobs: too simplistic?
- Earliest deadline is optimal!

Algorithm:

- Assume jobs are reordered such that \( d_1 \leq d_2 \leq \cdots \leq d_n \)
- Start/finishing times:
  - First job starts at time \( s(1) = 0 \)
  - Duration of job \( i \) is \( t_i \): \( f(i) = s(i) + t_i \)
  - No gaps between jobs: \( s(i + 1) = f(i) \)

(idle time: gaps in a schedule \(\rightarrow\) alg. gives schedule with no idle time)
Example

Jobs ordered by deadline:

- $t_1 = 5 \quad | \quad d_4 = 7$
- $t_2 = 3 \quad | \quad d_2 = 10$
- $t_3 = 7 \quad | \quad d_1 = 11$
- $t_3 = 3 \quad | \quad d_3 = 13$

Schedule:

- $t_1 = 5 \quad | \quad t_2 = 3 \quad | \quad t_3 = 7 \quad | \quad t_3 = 3$

Lateness: job 1: 0, job 2: 0, job 3: 4, job 4: 5
Basic Facts

1. There is an optimal schedule with no idle time
   – Can just schedule jobs earlier...

2. Inversion: Job $i$ scheduled before job $j$ if $d_i > d_j$
   Schedules with no inversions have the same maximum lateness
Earliest Deadline is Optimal

**Theorem:**
There is an optimal schedule \(\mathcal{O}\) with no inversions and no idle time.

**Proof:**
- Consider some schedule \(\mathcal{O}'\) with no idle time.
- If \(\mathcal{O}'\) has inversions, \(\exists\) pair \((i, j)\), s.t. \(i\) is scheduled immediately before \(j\) and \(d_j < d_i\).

  - Claim: Swapping \(i\) and \(j\) gives a schedule with
    1. Fewer inversions
    2. Maximum lateness no larger than in \(\mathcal{O}'\)
Earliest Deadline is Optimal

**Claim:** Swapping $i$ and $j$: maximum lateness no larger than in $O'$
Exchange Argument

• General approach that often works to analyze greedy algorithms

• Start with any solution

• Define basic exchange step that allows to transform solution into a new solution that is not worse

• Show that exchange step move solution closer to the solution produced by the greedy algorithm

• Number of exchange steps to reach greedy solution should be finite...
Another Exchange Argument Example

- **Minimum spanning tree (MST) problem**
  - Classic graph-theoretic optimization problem

- **Given**: weighted graph

- **Goal**: spanning tree with min. total weight

- Several greedy algorithms work

- **Kruskal’s algorithm**:
  - Start with empty edge set
  - As long as we do not have a spanning tree:
    - **add minimum weight edge that doesn’t close a cycle**
Kruskal Algorithm: Example