Greedy Algorithms

Example

Interval Scheduling

- Job \( j \) starts at time \( s_j \) and finishes as time \( f_j \)
- Two jobs are compatible if they don’t overlap
- Goal: find maximum subset of mutually compatible jobs

```
for job j = 1 to n
  A ← φ
  if job j is compatible with A
    A ← A ∪ \{j\}
return A
```

Counterexample for earliest finish time first algorithm

```
SORT jobs by finish time so that \( f_1 \leq f_2 \leq \ldots \leq f_n \)
A ← φ
for job j = 1 to n
  if job j is compatible with A
    A ← A ∪ \{j\}
return A
```

```
least interval
earliest start time
fewest conflicts
```
**Generic greedy strategy:** Consider jobs in some natural order. Take each job provided it’s compatible with the ones already taken

- *Earliest start time:* Consider jobs in ascending order of \( s_j \)
- *Earliest finish time:* Consider jobs in ascending order of \( f_j \)
- *Shortest interval:* Consider jobs in ascending order of \( f_j - s_j \)
- *Fewest conflicts:* For each job \( j \), count the number of conflicting jobs \( c_j \). Schedule in ascending order of \( c_j \)
Interval scheduling: greedy algorithms

Consider jobs in some natural order. Make each job provided it is compatible with the ones already taken:

- Earliest start time [earliest start time]
- Shortest interval [shortest interval]
- Fewest conflicts

A proposition that implements earliest finish time first takes \( O(n \log n) \) time.

counterexample for earliest start time

counterexample for shortest interval

counterexample for fewest conflicts
Interval scheduling: “earliest finish time first” algorithm

Sort jobs by finishing time so that $f_1 \leq f_2 \leq \cdots \leq f_n$

$A \leftarrow \emptyset$  // set of jobs selected

for $j$ in $[1..n]$ do
    if job $j$ compatible with $A$ then
        $A \leftarrow A \cup \{j\}$

return $A$

$O(n \log n)$ time implementation:

• Keep track of last job $j^*$ added to $A$
• Job $j$ compatible with $A \iff s_j \geq f_{j^*}$
• Sorting takes $O(n \log n)$ time
**Theorem**

“earliest finish time first” algorithm is optimal

**Proof (by contradiction)**

Assume greedy not optimal

Suppose for some input of size $n$:

- greedy produces solution $j_1 < j_2 < \ldots < j_k$
- there is a better solution $j'_1 < j'_2 < \ldots < j'_\ell$, where $\ell > k$

Assume this is the smallest $n$ for which greedy is not optimal
• greedy produces solution $j_1 < j_2 < \cdots < j_k$
• there is a better solution $j'_1 < j'_2 < \cdots < j'_\ell$, where $\ell > k$
• $n$ chosen minimally

Let $C$ be the set of jobs that are compatible with job $j_1$

$C = \text{set of jobs with start time } \geq f_{j_1}$

Example:

```
Example:           greedy: $(j_1,j_2,j_3) = (b,e,h)$

C = \{e,f,g,h\}
```

The sequence $j_2, \ldots, j_k$ is a greedy solution for $C$

Since $f_{j'_1} \geq f_{j_1}$, jobs $j'_2, \ldots, j'_\ell$ are compatible with job $j_1$

The sequence $j'_2, \ldots, j'_\ell$ is a better solution for $C$

This contradicts the minimality of $n$
A general approach to proving optimality of greedy algorithms

Proof by contradiction: assume that greedy is not optimal

So there is a counter-example: an input $I$ and a strategy $S$ such that $S$ beats greedy on input $I$

Assume that the counter-example is as small as possible (with respect to some natural notion of input size)

Construct a smaller counter-example: a smaller input $I'$ such and a strategy $S'$ that beats greedy on input $I'$

- This contradicts the minimality of the counter-example
- This is usually done by transforming $(I, S)$ into $(I', S')$
Limitations of greedy approach

- Minor variations may not be solvable by greedy approach
  - *Example*: suppose each job in the interval scheduling problem has a non-negative “value”
    - We want to maximize the total value of all scheduled jobs (rather than number of jobs)
    - Greedy approach fails (homework)
    - Dynamic programming applies (homework)
- Greedy algorithms can be tricky to design
- Proofs of correctness can be tricky to get right
Example

Huffman Encoding Problem

Let $w_1, \ldots, w_n$ be non-negative weights.

Let $T$ be a binary tree, with each $w_i$ labeling some leaf of depth $d_i$.

Define $\text{Cost}(T) := \sum_i w_id_i$.

Problem: given $w_1, \ldots, w_n$, find a minimal cost $T$.

Without loss of generality, we may assume $T$ is a full binary tree, i.e., each non-leaf has two children.
Example:

Application: optimal prefix-free binary encoding

- $w_i$ represents probability of symbol $\sigma_i$
- path in tree represents a bit string encoding
- $\text{Cost}(T)$ is expected encoding length
- prefix-free property allows for unambiguous parsing of strings


Encodings: $A \Rightarrow 000, B \Rightarrow 001, C \Rightarrow 010, D \Rightarrow 011, E \Rightarrow 1$

$A \Rightarrow 010, B \Rightarrow 10, C \Rightarrow 011, D \Rightarrow 111, E \Rightarrow 00$
For a tree $T$, define its \textit{weight} to be the sum of weights of its leaves

Greedy Algorithm:

- Start with a forest of $n$ leaves
- Repeat $n - 1$ times:
  - Take two trees $T_1, T_2$ in the forest of least weight, and join them:

Implement using a heap. Running time: $O(n \log n)$
**Theorem**
This greedy algorithm produces a least-cost tree.

**Lemma 1**
Let $T$ be a full binary tree with weights $w(\nu)$ assigned to leaves $\nu$. Suppose $\nu_1, \nu_2$ are leaves of smallest weight. We can construct a new tree $T'$ from $T$ such that

1. the leaves of $T'$ and $T$ are the same,
2. $\nu_1$ and $\nu_2$ are siblings in $T'$, and
3. $\text{Cost}(T') \leq \text{Cost}(T)$.
Proof of Lemma 1. Assume $v_1, v_2$ not siblings in $T$

Let $d_i := \text{depth of } v_i \text{ in } T \text{ for } i = 1, 2$

Assume $d_1 \geq d_2$ and let $\Delta := d_1 - d_2$

Moving $v_2$ down increases cost by $\Delta \cdot w(v_2)$

All leaves in $T_1$ have weight $\geq w(v_2)$ (because $v_1, v_2$ have least weight), and so moving $T_1$ up decreases cost by at least $\Delta \cdot w(v_2)$
**Lemma 2**

Let $T$ be a full binary tree with weights $w(v)$ assigned to leaves $v$. Suppose $v_1, v_2$ are leaves that are siblings in $T$ with parent $v_3$, and that we create a new tree $\tilde{T}$ by deleting $v_1$ and $v_2$ and set $w(v_3) := w(v_1) + w(v_2)$:

$$w(v_3) = w(v_1) + w(v_2).$$

Then $\text{Cost}(\tilde{T}) = \text{Cost}(T) - w(v_1) - w(v_2)$.

**Proof.** Let $d =$ depth of $v_3$ in $T$

$v_1$ and $v_2$ contribute $(d + 1)(w(v_1) + w(v_2))$ to $\text{Cost}(T)$

$v_3$ contributes $d(w(v_1) + w(v_2))$ to $\text{Cost}(\tilde{T})$
Proof of Theorem (by contradiction)

Suppose that on some input of size \( n \), the greedy algorithm produces the tree \( T \), but there exists a tree \( X \) with \( \text{Cost}(X) < \text{Cost}(T) \)

Assume that this counter-example is chosen with \( n \) as small as possible

We must have \( n > 2 \) (otherwise, no better solution)

Consider the first step of the greedy algorithm, which joined two leaves \( v_1, v_2 \) of smallest weight

\( v_1 \) and \( v_2 \) are siblings in \( T \)
Apply Lemma 1 to $X$, obtaining a new tree $X'$ in which $v_1$ and $v_2$ are siblings, and $\text{Cost}(X') \leq \text{Cost}(X)$

Apply Lemma 2 to both $T$ and $X'$, removing $v_1$ and $v_2$, obtaining trees $\tilde{T}$ and $\tilde{X}'$ such that

$$\text{Cost}(\tilde{T}) = \text{Cost}(T) - s, \quad \text{Cost}(\tilde{X}') = \text{Cost}(X') - s$$

where $s := w(v_1) + w(v_2)$

$\tilde{T}$ is also a tree that would be produced the greedy algorithm

It follows that

$$\text{Cost}(\tilde{X}') = \text{Cost}(X') - s \leq \text{Cost}(X) - s < \text{Cost}(T) - s = \text{Cost}(\tilde{T})$$

This contradicts the assumption that $n$ was chosen as small as possible