Greedy Algorithms

Lecture 11
March 1, 2011

Part I

Problems and Terminology
Problem Types

- **Decision Problem**: Is the input a YES or NO input?
  Example: Given graph $G$, nodes $s, t$, is there a path from $s$ to $t$ in $G$?

- **Search Problem**: Find a *solution* if input is a YES input.
  Example: Given graph $G$, nodes $s, t$, find an $s$-$t$ path.

- **Optimization Problem**: Find a *best* solution among all solutions for the input.
  Example: Given graph $G$, nodes $s, t$, find a shortest $s$-$t$ path.

Terminology

- A problem $\Pi$ consists of an *infinite* collection of inputs $\{I_1, I_2, \ldots, \}$. Each input is referred to as an *instance*.

- The *size* of an instance $I$ is the number of bits in its representation.

- For an instance $I$, $\text{sol}(I)$ is a set of *feasible solutions* to $I$.
  *Typical implicit assumption*: given instance $I$ and $y \in \Sigma^*$, there is a way to check (efficiently!) if $y \in \text{sol}(I)$. In other words, problem is in $\text{NP}$.

- For optimization problems each solution $s \in \text{sol}(I)$ has an associated *value*. *Typical implicit assumption*: given $s$, can compute value efficiently.
Problem Types

- **Decision Problem**: Given \( I \) output whether \( \text{sol}(I) = \emptyset \) or not.
- **Search Problem**: Given \( I \), find a solution \( s \in \text{sol}(I) \) if \( \text{sol}(I) \neq \emptyset \).
- **Optimization Problem**: Given \( I \),
  - Minimization problem. Find a solution \( s \in \text{sol}(I) \) of minimum value
  - Maximization problem. Find a solution \( s \in \text{sol}(I) \) of maximum value
- **Notation**: \( \text{opt}(I) \): interchangeably (when there is no confusion) used to denote the value of an optimum solution or some fixed optimum solution.

Part II

Greedy Algorithms: Tools and Techniques
What is a Greedy Algorithm?

No real consensus on a universal definition.

Greedy algorithms:
- make decision incrementally in small steps \textit{without backtracking}
- decision at each step is based on improving \textit{local or current} state in a myopic fashion without paying attention to the \textit{global} situation
- decisions often based on some fixed and simple \textit{priority} rules

Pros and Cons of Greedy Algorithms

Pros:
- Usually (too) easy to design greedy algorithms
- Easy to implement and often run fast since they are simple
- Several important cases where they are effective/optimal
- Lead to a first-cut heuristic when problem not well understood

Cons:
- \textbf{Very often} greedy algorithms don’t work. Easy to lull oneself into believing they work
- Many greedy algorithms possible for a problem and no structured way to find effective ones

\textbf{CS 473: Every greedy algorithm needs a proof of correctness}
Greedy Algorithm Types

Crude classification:
- **Non-adaptive**: fix some ordering of decisions a priori and stick with the order
- **Adaptive**: make decisions adaptively but greedily/locally at each step

Plan:
- See several examples
- Pick up some proof techniques

Interval Scheduling

**Input** A set of jobs with start and finish times to be scheduled on a resource (example: classes and class rooms)

**Goal** Schedule as many jobs as possible
- Two jobs with overlapping intervals cannot both be scheduled!
Greedy Template

\[ R \] is the set of all requests
\[ X \] is empty (* \( X \) will store all the jobs that will be scheduled *)

\textbf{while} \( R \) is not empty \textbf{do}
  \textbf{choose} \( i \in R \)
  \textbf{add} \( i \) to \( X \)
  \textbf{remove from} \( R \) all requests that overlap with \( i \)
\textbf{return} the set \( X \)

\textbf{Main task:} Decide the order in which to process requests in \( R \)

Earliest Start Time

Process jobs in the order of their starting times, beginning with those that start earliest.

Figure: Counter example for earliest start time
Smallest Processing Time

Process jobs in the order of processing time, starting with jobs that require the shortest processing.

Figure: Counter example for smallest processing time

Fewest Conflicts

Process jobs in that have the fewest “conflicts” first.

Figure: Counter example for fewest conflicts
Earliest Finish Time

Process jobs in the order of their finishing times, beginning with those that finish earliest.

Optimal Greedy Algorithm

\( R \) is the set of all requests
\( X \) is empty (\(*\ X \) will store all the jobs that will be scheduled \(*\))

**while** \( R \) is not empty

- choose \( i \in R \) such that finishing time of \( i \) is least
- add \( i \) to \( X \)
- remove from \( R \) all requests that overlap with \( i \)

**return** \( X \)

**Theorem**

*The greedy algorithm that picks jobs in the order of their finishing times is optimal.*
Proving Optimality

- **Correctness:** Clearly the algorithm returns a set of jobs that does not have any conflicts
- For a set of requests $R$, let $O$ be an optimal set and let $X$ be the set returned by the greedy algorithm. Then $O = X$? Not likely!

Instead we will show that $|O| = |X|$

---

Proof of Optimality: Key Lemma

**Lemma**

Let $i_1$ be first interval picked by Greedy. There exists an optimum solution that contains $i_1$.

**Proof.**

Let $O$ be an arbitrary optimum solution. If $i_1 \in O$ we are done.

**Claim:** If $i_1 \not\in O$ then there is exactly one interval $j_1 \in O$ that conflicts with $i_1$. (proof later)

- Form a new set $O'$ by removing $j_1$ from $O$ and adding $i_1$, that is $O' = (O - \{j_1\}) \cup \{i_1\}$.
- From claim, $O'$ is a feasible solution (no conflicts).
- Since $|O'| = |O|$, $O'$ is also an optimum solution and it contains $i_1$. 

---
Claim

If $i_1 \not\in O$ then there is exactly one interval $j_1 \in O$ that conflicts with $i_1$.

Proof.

- Suppose $j_1, j_2 \in O$ such that $j_1 \neq j_2$ and both $j_1$ and $j_2$ conflict with $i_1$.
- Since $i_1$ has earliest finish time, $j_1$ and $i_1$ overlap at $f(i_1)$.
- For same reason $j_2$ also overlaps with $i_1$ at $f(i_1)$.
- Implies that $j_1, j_2$ overlap at $f(i_1)$ contradicting the feasibility of $O$.

See figure in next slide.

Figure: Since $i_1$ has the earliest finish time, any interval that conflicts with it does so at $f(i_1)$. This implies $j_1$ and $j_2$ conflict.
Proof of Optimality of Earliest Finish Time First

Proof by Induction on number of intervals.

**Base Case:** \( n = 1 \). Trivial since Greedy picks one interval.

**Induction Step:** Assume theorem holds for \( i < n \).

Let \( I \) be an instance with \( n \) intervals

\( I' \): \( I \) with \( i_1 \) and all intervals that overlap with \( i_1 \) removed

\( G(I), G(I') \): Solution produced by Greedy on \( I \) and \( I' \)

From Lemma, there is an optimum solution \( O \) to \( I \) and \( i_1 \in O \).

Let \( O' = O - \{i_1\} \). \( O' \) is a solution to \( I' \).

\[
|G(I)| = 1 + |G(I')| \quad \text{(from Greedy description)}
\]
\[
\geq 1 + |O'| \quad \text{(By induction, } G(I') \text{ is optimum for } I')
\]
\[
= |O|
\]

Implementation and Running Time

Initially \( R \) is the set of all requests

\( X \) is empty (\(* X \) will store all the jobs that will be scheduled *)

while \( R \) is not empty

choose \( i \in R \) such that finishing time of \( i \) is least

if \( i \) does not overlap with requests in \( X \)

add \( i \) to \( X \)

remove \( i \) from \( R \)

return the set \( X \)

- Presort all requests based on finishing time. \( O(n \log n) \) time
- Now choosing least finishing time is \( O(1) \)
- Keep track of the finishing time of the last request added to \( A \).
  Then check if starting time of \( i \) later than that
- Thus, checking non-overlapping is \( O(1) \)
- Total time \( O(n \log n + n) = O(n \log n) \)
Comments

- Interesting Exercise: smallest interval first picks at least half the optimum number of intervals.
- Instead of maximizing the total number of requests, associate value/weight with each job that is scheduled. Try to schedule jobs to maximize total value/weight. No greedy algorithm. Will be seen later in this course to illustrate dynamic programming.
- All requests need not be known at the beginning. Such online algorithms are a subject of research.

Scheduling all Requests

**Input** A set of lectures, with start and end times

**Goal** Find the minimum number of classrooms, needed to schedule all the lectures such two lectures do not occur at the same time in the same room.

**Figure:** A schedule requiring 4 classrooms

**Figure:** A schedule requiring 3 classrooms
Greedy Algorithm

Initially \( R \) is the set of all requests
\[
d = 0 (* \text{ number of classrooms } *)
\]
\textbf{while} \( R \) is not empty
\hspace{1em} choose \( i \in R \) such that start time of \( i \) is earliest
\hspace{1em} if \( i \) can be scheduled in some class-room \( k \leq d \)
\hspace{1em} schedule lecture \( i \) in class-room \( k \)
\hspace{1em} else
\hspace{1.5em} allocate a new class-room \( d + 1 \) and schedule lecture \( i \) in \( d + 1 \)
\hspace{1em} \( d = d + 1 \)

What order should we process requests in? According to start times (breaking ties arbitrarily)

Depth of Lectures

\textbf{Definition}

- For a set of lectures \( R, k \) are said to be in conflict if there is some time \( t \) such that there are \( k \) lectures going on at time \( t \).
- The depth of a set of lectures \( R \) is the maximum number of lectures in conflict at any time.
Lemma

For any set $R$ of lectures, the number of class-rooms required is at least the depth of $R$.

Proof.

All lectures that are in conflict must be scheduled in different rooms.

Number of Class-rooms used by Greedy Algorithm

Lemma

Let $d$ be the depth of the set of lectures $R$. The number of class-rooms used by the greedy algorithm is $d$.

Proof.

1. Suppose the greedy algorithm uses more than $d$ rooms. Let $j$ be the first lecture that is scheduled in room $d + 1$.
2. Since we process lectures according to start times, there are $d$ lectures that start (at or) before $j$ and which are in conflict with $j$.
3. Thus, at the start time of $j$, there are at least $d + 1$ lectures in conflict, which contradicts the fact that the depth is $d$. 

\[
\square
\]
Correctness

Observation

The greedy algorithm does not schedule two overlapping lectures in the same room.

Theorem

The greedy algorithm is correct and uses the optimal number of class-rooms.
Implementation and Running Time

Initially $R$ is the set of all requests

$d = 0$ (* number of classrooms *)

while $R$ is not empty

choose $i \in R$ such that start time of $i$ is earliest

if $i$ can be scheduled in some class-room $k \leq d$

schedule lecture $i$ in class-room $k$

else

allocate a new class-room $d + 1$ and schedule lecture $i$ in $d + 1$

$$d = d + 1$$

Presort according to start times. Picking lecture with earliest start time can be done in $O(1)$ time.

Keep track of the finish time of last lecture in each room.

Checking conflict takes $O(d)$ time. With priority queues, checking conflict takes $O(\log d)$ time.

Total time

$$= O(n \log n + nd) = O(n \log n + n \log d) = O(n \log n)$$

Scheduling to Minimize Lateness

* Given jobs with deadlines and processing times to be scheduled on a single resource.
* If a job $i$ starts at time $s_i$ then it will finish at time $f_i = s_i + t_i$, where $t_i$ is its processing time. $d_i$: deadline.
* The lateness of a job is $l_i = \max(0, f_i - d_i)$.
* Schedule all jobs such that $L = \max l_i$ is minimized.
A Simpler Feasibility Problem

- Given jobs with deadlines and processing times to be scheduled on a single resource.
- If a job $i$ starts at time $s_i$ then it will finish at time $f_i = s_i + t_i$, where $t_i$ is its processing time.
- Schedule all jobs such that each of them completes before its deadline (in other words $L = \max_i l_i = 0$).

**Definition**

A schedule is **feasible** if all jobs finish before their deadline.

**Greedy Template**

Initially $R$ is the set of all requests

$\text{curr\_time} = 0$

$\text{while } R \text{ is not empty} \text{ do}$

$\text{choose } i \in R$

$\text{curr\_time} = \text{curr\_time} + t_i$

$\text{if } (\text{curr\_time} > d_i) \text{ then}$

$\text{return} \text{ 'no feasible schedule'}$

$\text{return} \text{ 'found feasible schedule'}$

**Main task:** Decide the order in which to process jobs in $R$
Three Algorithms

- Shortest job first — sort according to $t_i$.
- Shortest slack first — sort according to $d_i - t_i$.
- EDF = Earliest deadline first — sort according to $d_i$.

Counter examples for first two: exercise

---

Earliest Deadline First

**Theorem**

*Greedy with EDF rule for picking requests correctly decides if there is a feasible schedule.*

Proof via an exchange argument.

Idle time: time during which machine is not working.

**Lemma**

*If there is a feasible schedule then there is one with no idle time before all jobs are finished.*
Inversions

Definition
A schedule $S$ is said to have an inversion if there are jobs $i$ and $j$ such that $S$ schedules $i$ before $j$, but $d_i > d_j$.

Claim
If a schedule $S$ has an inversion then there is an inversion between two adjacently scheduled jobs.

Proof: exercise.

Main Lemma

Lemma
If there is a feasible schedule, then there is one with no inversions.

Proof Sketch.
Let $S$ be a schedule with minimum number of inversions.

- If $S$ has 0 inversions, done.
- Suppose $S$ has one or more inversions. By claim there are two adjacent jobs $i$ and $j$ that define an inversion.
- Swap positions of $i$ and $j$.
- New schedule is still feasible. (Why?)
- New schedule has one fewer inversion — contradiction!
Goal: schedule to minimize $L = \max_i l_i$.

How can we use algorithm for simpler feasibility problem?

Given a lateness bound $L$, can we check if there is a schedule such that $\max_i l_i \leq L$?

Yes! Set $d'_i = d_i + L$ for each job $i$. Use feasibility algorithm with new deadlines.

How can we find minimum $L$? Binary search!

**Binary search for finding minimum lateness**

$L = L_{\text{min}} = 0$
$L_{\text{max}} = \sum_i t_i$ // why is this sufficient?

While $L_{\text{min}} < L_{\text{max}}$ do

\[ L = \lfloor (L_{\text{max}} + L_{\text{min}})/2 \rfloor \]

check if there is a feasible schedule with lateness $L$

if ‘‘yes’’ then $L_{\text{max}} = L$

else $L_{\text{min}} = L + 1$

end while

return $L$

**Running time:** $O(n \log n \cdot \log T)$ where $T = \sum_i t_i$

- $O(n \log n)$ for feasibility test (sort by deadlines)
- $O(\log T)$ calls to feasibility test in binary search
Do we need binary search?

What happens in each call?

EDF algorithm with deadlines $d'_i = d_i + L$.

Greedy with EDF schedules the jobs in the same order for all $L$!!!

Maybe there is a direct greedy algorithm for minimizing maximum lateness?

---

Greedy Algorithm for Minimizing Lateness

Initially $R$ is the set of all requests

$\text{curr\_time} = 0$

$\text{curr\_late} = 0$

while $R$ is not empty

choose $i \in R$ with earliest deadline

$\text{curr\_time} = \text{curr\_time} + t_i$

$\text{late} = \text{curr\_time} - d_i$

$\text{curr\_late} = \max(\text{late}, \text{curr\_late})$

return $\text{curr\_late}$

Exercise: argue directly that above algorithm is correct (see book).

Can be easily implemented in $O(n \log n)$ time after sorting jobs.
Greedy Analysis: Overview

- **Greedy’s first step leads to an optimum solution.** Show that there is an optimum solution leading from the first step of Greedy and then use induction. Example, Interval Scheduling.
- **Greedy algorithm stays ahead.** Show that after each step the solution of the greedy algorithm is at least as good as the solution of any other algorithm. Example, Interval scheduling.
- **Structural property of solution.** Observe some structural bound of every solution to the problem, and show that greedy algorithm achieves this bound. Example, Interval Partitioning.
- **Exchange argument.** Gradually transform any optimal solution to the one produced by the greedy algorithm, without hurting its optimality. Example, Minimizing lateness.

Takeaway Points

- Greedy algorithms come naturally but often are incorrect. A proof of correctness is an absolute necessity.
- *Exchange* arguments are often the key proof ingredient. Focus on why the first step of the algorithm is correct: need to show that there is an optimum/correct solution with the first step of the algorithm.
- Thinking about correctness is also a good way to figure out which of the many greedy strategies is likely to work.