CMSC 451: Interval Scheduling

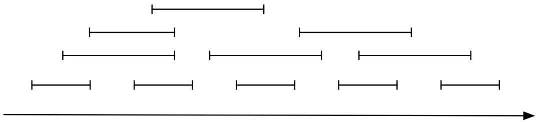
Slides By: Carl Kingsford

Department of Computer Science
University of Maryland, College Park

Based on Section 4.1 of *Algorithm Design* by Kleinberg & Tardos.

Interval Scheduling

- You want to schedule jobs on a supercomputer.
- Requests take the form (s_i, f_i) meaning a job that runs from time s_i to time f_i.
- You get many such requests, and you want to process as many as possible, but the computer can only work on one job at a time.



Interval Scheduling

Given a set $J = \{(s_i, f_i) : i = 1, \ldots, n\}$ of job intervals, find the largest $S \subset J$ such that no two intervals in S overlap.

Greedy Algorithms

- Not easy to define what we mean by "greedy algorithm"
- Generally means we take little steps, looking only at our local choices
- Often among the first reasonable algorithms we can think of
- Frequently doesn't lead to optimal solutions, but sometimes it does.

Example Greedy Algorithms

- `TreeGrowing` is an example of a greedy framework for most choices of `nextEdge` functions.
- Topological sort & testing bipartiteness were greedy algorithms
- Interval Scheduling turns out to have a nice greedy algorithm that works.

Ideas for Interval Scheduling

A greedy framework:

```
S = set of input intervals (s_i, f_i)
While S is not empty:
    q = nextInterval(S)
    Output interval q
    Remove intervals that overlap with q from S
```

What are possible rules for `nextInterval`?

Ideas for Interval Scheduling

What are possible rules for `nextInterval`?

1. *Choose the interval that starts earliest.*
   Rationale: start using the resource as soon as possible.

2. *Choose the smallest interval.*
   Rationale: try to fit in lots of small jobs.

3. *Choose the interval that overlaps with the fewest remaining intervals.*
   Rationale: keep our options open and eliminate as few intervals as possible.

Rules That Don't Work

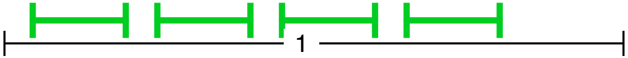
1. Earliest start time

2. Shortest job

3. Fewest conflicts

Rules That Don't Work

1. Earliest start time

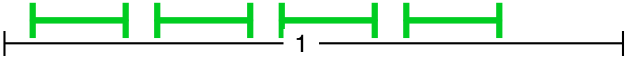


2. Shortest job

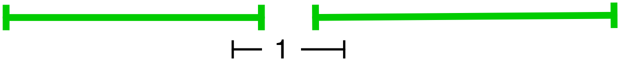
3. Fewest conflicts

Rules That Don't Work

1. Earliest start time



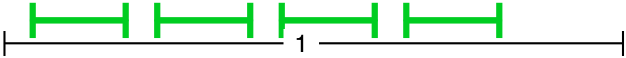
2. Shortest job



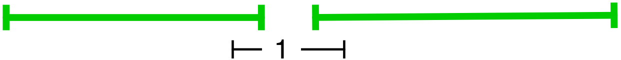
3. Fewest conflicts

Rules That Don't Work

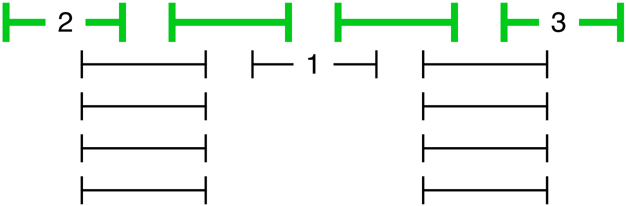
1. Earliest start time



2. Shortest job



3. Fewest conflicts



Optimal Greedy Algorithm

- Choose the interval with the earliest finishing time.
  Rationale: ensure we have as much of the resource left as possible.

```
S = set of input intervals {(s[i], f[i])}
While S is not empty:
    q = a request in S that has the
                soonest finishing time
    Output interval q
    Remove intervals that overlap with q from S
```

- This algorithm chooses a compatible set of intervals.

How to Prove Optimality

How can we prove the schedule returned is optimal?

- Let A be the schedule returned by this algorithm.
- Let OPT be some optimal solution.

Might be hard to show that $A = OPT$, instead we need only to show that $|A| = |OPT|$.

Note the distinction: instead of proving directly that a choice of intervals A is the same as an optimal choice, we prove that it has the same *number* of intervals as an optimal. Therefore, it is optimal.

Notation

Let these be the schedules of A and OPT:

$$\begin{array}{rl} \text{A:} & i_1, i_2, \ldots, i_k \\ \text{OPT:} & j_1, j_2, \ldots, j_m \end{array}$$

Let $f(i)$ be the finishing time of job i.

Theorem

For all $r \leq k$ we have $f(i_r) \leq f(j_r)$.

Proof.

By induction. True when $r = 1$ because we chose greedily.

Assume $f(i_{r-1}) \leq f(j_{r-1})$. Then we have

$$f(i_{r-1}) \leq f(j_{r-1}) \leq s(j_r),$$

where $s(j_r)$ is the start time of job j_r. So, job j_r is available when the greedy algorithm makes its choice. Hence, $f(i_r) \leq f(j_r)$. □

Greedy is Optimal

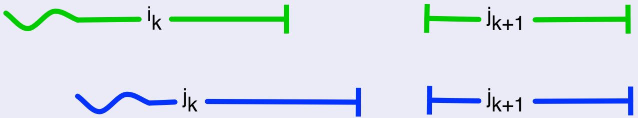
Theorem

This greedy algorithm is optimal for Interval Scheduling.

Proof.

Suppose not. Then if A has k jobs, OPT has $m > k$ jobs.

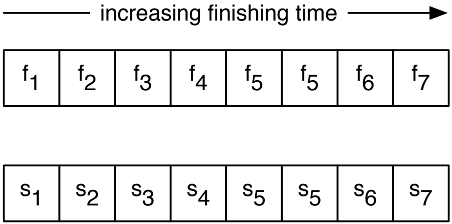
By our lemma, after k jobs we have this situation:



So, job j_{k+1} must have been available to the greedy algorithm. $\square$

Implementation

1. Sort the intervals based on f_i — takes $O(n \log n)$.
2. Scan down this list, output the first element that starts after the finishing time of the last item output — takes $O(n)$.



LatestFinishingTime = finishing time of last scheduled interval

Extensions

- Online algorithms: What if you don't know all the intervals at the start? Current active area of research.

- What if some intervals are more important than others? Weighted interval scheduling. — We'll see this later.

Interval Partitioning Problem

Other rules of scheduling intervals also lead to nice greedy algorithms. For example:

Interval Partitioning Problem

Interval Partitioning Problem

Interval Partitioning Problem

Given intervals (s_i, f_i) assign them to processors so that you schedule every interval and use the smallest # of processors.

Now, we're trying to minimize the # of processors (or rooms) used.

Another way to think of it: Each processor corresponds to a color. We're trying to color intervals with the fewest # of colors so that no two overlapping intervals get the same color.

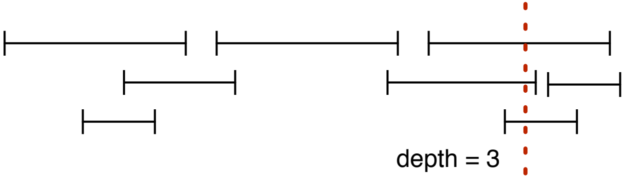
Interval Partitioning Problem

Definition

Depth is the maximum number of intervals passing over any time point.

Theorem

The number of processors needed is at least the depth of the set of intervals.



We need at least depth processors.

Can we find a schedule with no more than depth processors?

Greedy Alg for Interval Partitioning

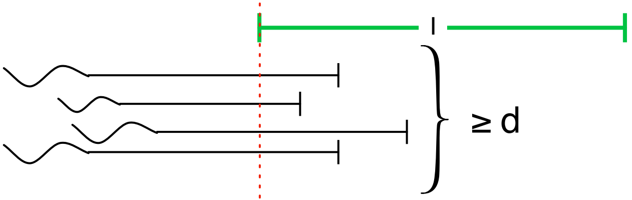
Yes: Let $\{1, \ldots, d\}$ be a set of labels, where $d = \text{depth}$.

```
Sort intervals by start time
For j = 1,2,3,...,n
    Let Q = set of labels that haven't been
        assigned to a preceding interval that
        overlaps Ij
    If Q is not empty,
        Pick any label from Q and assign it to Ij
    Else
        Leave Ij unlabeled
Endfor
```

Every interval gets a label

No overlapping intervals get the same label because we exclude the labels that have already been used.

Every interval gets a label: The only way it wouldn't is if we've run out of labels. That would mean, when coloring interval I that $\geq d$ intervals with start times before I overlap I:

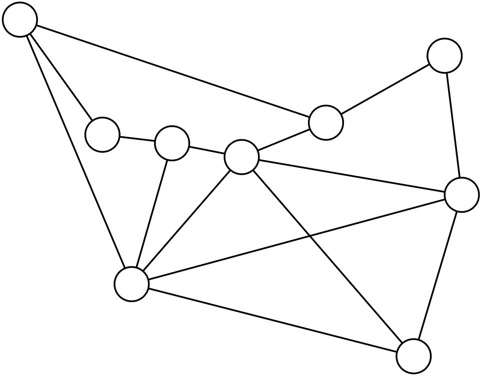


So, when coloring i, there must be a color free.

Graph Coloring

Graph Coloring

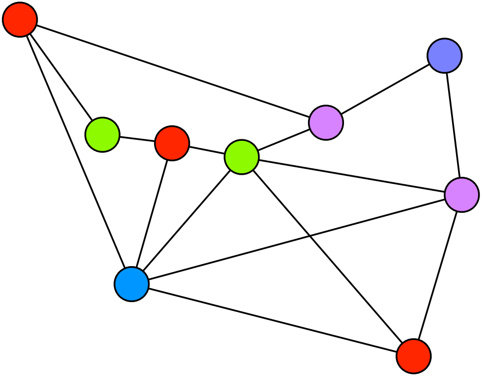
Given a graph G and a number k, color the nodes with k colors such that no edge connects 2 vertices of the same color (or report that it can't be done).



Graph Coloring

Graph Coloring

Given a graph G and a number k, color the nodes with k colors such that no edge connects 2 vertices of the same color (or report that it can't be done).



Interval Graph Coloring

When $k \geq 3$, Graph Coloring is hard in general.

We saw an algorithm for Graph Coloring when $k = 2$.

Interval Graph Coloring

Given a graph G derived from a set of intervals and a number k, color the nodes with k colors such that no edge connects 2 vertices of the same color (or report that it can't be done).

Now we've seen an algorithm for solving Graph Coloring on the restricted class of graphs called "Interval Graphs."