Design and Analysis & of Algorithms

Hoeteck Wee · hoeteck@cs.qc.cuny.edu
http://www.cs.qc.edu/~hoeteck/f09/
Interval scheduling

INTERVAL SCHEDULING.

- Input: set of jobs with start and finish times.
- Find max cardinality subset of compatible (i.e. non-overlapping) jobs.

GREEDY WORKS. earliest finish time first
Interval partitioning

INTERVAL PARTITIONING.

- **Input:** set of jobs with start and finish times.
- **Goal:** schedule all jobs on the **minimal** number of resources so that each resource runs at most 1 job at any time.
- **Example:** scheduling classrooms for lectures

**DEFINITION.** **Depth** is maximum number of jobs at any given time

- structural property: number of resources needed $\geq$ depth
- does there exist a schedule using with resource count equal to depth?
Interval partitioning

**GREEDY ALGORITHM.** assign each lecture to first available classroom

```python
d = 0
for j in range(1,n+1):
    if some classroom is available for lecture j:
        schedule lecture j in that classroom
    else:
        d = d+1
        schedule lecture j in classroom d
return d
```

**VALIDITY.** no overlapping lectures in any classroom

**OPTIMALITY.**

- algorithmic guarantee: depth $\geq d$
- structural property: number of classrooms needed $\geq$ depth
Interval partitioning
Interval partitioning

NAIVE IMPLEMENTATION.
- maintain a list of classrooms + all lectures assigned to that classroom
- checking availability for the $j$’th lecture takes time: $O(j)$
- total running time: $O(n^2)$

(MILDLY) IMPROVED IMPLEMENTATION.
- pre-processing: sort lectures in order of increasing start time
- maintain a list of classrooms + last lecture assigned to that classroom
- checking availability for the $j$’th lecture takes time: $O(d)$

BETTER IMPLEMENTATION.
- store classrooms using a heap with key: finish time of last lecture
- checking availability for the $j$’th lecture takes time: $O(\log n)$
  (use FINDMIN/EXTRACTMIN)
- use INSERT to schedule a new lecture
Scheduling to minimize lateness

MINIMIZING LATENESS.

- Setup: single resource processes one job at a time.
- Input: set of jobs with processing time $t_j$ and deadline $d_j$.
- Definition: lateness $\ell_j = \max\{0, f_j - d_j\}$, where $f_j = s_j + t_j$
- Goal: schedule all jobs to minimize maximum lateness $L = \max \ell_j$

**Greedy works.**

- Interval scheduling: earliest finish time first
- Here: earliest deadline first
Minimizing lateness

GREEDY ALGORITHM. earliest deadline first

sort jobs by deadline in increasing order

\[
\begin{align*}
time, s, f &= 0, [], [] \\
\text{for } j \text{ in range}(1,n+1): \\
    &\quad \text{# assign job } j \text{ to the interval } [time, time+t_j] \\
    &\quad s.\text{append}(time), f.\text{append}(time + t_j) \\
    &\quad time = t + t_j \\
\text{return } s, f
\end{align*}
\]

RUNNING TIME. \( O(n \log n) \)
PROOF TECHNIQUE. exchange argument
- gradually transform an optimal schedule into a “greedy schedule”
- perform a series of exchanges that preserve optimality

DEFINITION. an inversion occurs when a job is scheduled before another with an earlier deadline
- job $i$ scheduled before job $j$ but $d_j < d_i$
- a greedy schedule has no inversions

EXCHANGE ARGUMENT.
- compress optimal schedule to remove any idle time
- repeatedly swap adjacent inversions until no inversions left

CLAIMS.
1. if there exists an inversion, there exists an adjacent inversion.
2. swapping adjacent inversions reduces total # of inversions by one.
3. swapping adjacent inversions does not increase max lateness.
4. all schedules with no inversions and no idle time have same max lateness.
CLAIM. swapping adjacent inversions does not increase max lateness.

- only finish times of jobs $i$ and $j$ are affected
  $\Rightarrow$ lateness for all other jobs remain unchanged
- new $\ell_i \leq \text{old } \ell_j$
- new $\ell_j \leq \text{old } \ell_j$
Greedy algorithms

GREEDY TEMPLATE.

▶ find the “right” rule (if one exists)
▶ establish optimality

PROOF TECHNIQUES.

▶ greedy algorithm stays ahead (c.f. INTERVAL SCHEDULING)
  after each step, partial solution achieved by greedy is as good as optimal
▶ match structural bound (c.f. INTERVAL PARTITIONING)
  show that greedy achieves some structural bound
▶ exchange argument (c.f. MINIMIZING LATENESS)
  transform optimal into greedy via optimality-preserving exchanges
THE END