1 The Cable Guy Problem

- You are an installer for Charter Cable.
- Your job is to do installations by appointment.
- Appointments have a fixed length (30 minutes) and must start on a 30-minute boundary.
- Each new customer specifies an earliest and latest possible start time for her appointment.
- **Goal**: schedule as many appointments as possible in a day.

More formally...

- Given set $P$ of unit-duration jobs that must be scheduled at integer times ("time slots") 0, 1, 2, \ldots
- Only one job may be scheduled per slot.
- Job $i$ may start in any slot from $s_i$ to $e_i$, inclusive.
- **Goal**: find a schedule that maximizes number of jobs scheduled.

2 A Greedy Algorithm

OK, here's a simple algorithm.

- Let $P$ be input set of jobs.
- Let $t$ be latest free slot in which some job from $P$ may be scheduled.
- Let $\Sigma \subseteq P$ be set of all jobs in $P$ that can be scheduled at time $t$.
- Choose job $i \in \Sigma$ with latest start time $s_i$.
- Schedule job $i$ at time $t$.
- Recur on remaining job set $P - \{i\}$, until this set is empty or no jobs can run.
Example:

3 Optimality

Let’s implement our three-part proof...

- **Greedy Choice Property**: let \( i \) be the job chosen first by greedy algo, and let \( \hat{t} \) be the time at which algo scheduled it. Then ...

- ...there exists an optimal solution that makes the greedy choice, that is, a soln that schedules \( i \) at time \( \hat{t} \).

- *(Note: it is not enough to find opt soln that uses \( i \) – it must be placed at time \( \hat{t} \), since this is what greedy choice does!)*

OK, on with the proof!

- **Pf**: Let \( \Pi \) be any optimal solution.

- If \( \Pi \) schedules \( i \) at time \( \hat{t} \), great.

- Otherwise, we have two cases.

- **Case 1**: Suppose \( \Pi \) doesn’t schedule \( i \) at all.

- If slot \( \hat{t} \) is empty in \( \Pi \), can simply add \( i \) for better soln.

- If slot \( \hat{t} \) is occupied by job \( j \), throw out \( j \) and put \( i \) there. Soln has same size as before.

- **Case 2**: Suppose \( \Pi \) schedules \( i \) at time \( t \neq \hat{t} \)

- If slot \( \hat{t} \) is empty in \( \Pi \), move \( i \) to \( \hat{t} \).

- Otherwise, some job \( j \) occupies slot \( \hat{t} \).

- Observe that \( t < \hat{t} \), since by choice of \( \hat{t} \), no job in \( P \) can be scheduled after time \( \hat{t} \).

- Observe also that by choice of job \( i \),

\[
    s_i \geq s_j.
\]

- Since \( t \geq s_i \), job \( j \) can run at time \( t \).

- Hence, we simply exchange slots of jobs \( i \) and \( j \).
• In all cases, new soln has size at least that of old and so is optimal. QED

One down, two to go.

• **Inductive Structure**: after making greedy choice, we are left with smaller instance of scheduling problem with no external constraints.

• **Pf**: Let $P' = P - \{i\}$.

• $P'$ is clearly a smaller set of jobs to be scheduled.

But what about constraints?

• First time greedy schedules a job, it can use latest slot that any job can take.

• For subsequent choices, this property does not hold! Consider our simple example.

• It seems we are more constrained on recursive calls than on original call! This breaks inductive structure.

• *Two solutions here.* Either extend the problem, or more carefully define the recursive call.

• **Problem extension**: input to problem includes a list of blocked slots. On recursion, also block $i$.

• Check that my algo description and proof of GCP need not change under this extension!

• (This is nice because you can allow the cable guy to take a lunch break.)

• **Recursion Defn**: as it happens, times are filled in from latest to earliest.

• So, trim ends of all unscheduled jobs back to at most $t - 1$.

• Also, remove any jobs $j$ for which $s_j \geq t$.

• For remaining jobs, algo never looks beyond $t - 1$, so does not see additional constraints.

• (Might be able to simplify GCP proof a bit, but forbids lunch breaks.)

• With one of the above two hacks, adding $i$ in slot $\hat{t}$ will be possible no matter how the subproblem is solved, so subsolution plus greedy choice is a feasible solution.

Two down, one to go.

• **Optimal Substructure**: let $\Pi'$ be opt solution to subproblem $P'$. Then $\Pi'$ together with $i$ at time $\hat{t}$ is opt solution to $P$.

• Value of $\Pi$ is value of $\Pi'$, plus one.

• Apply usual contradiction argument. QED

**Moral**: be careful what your subproblem is. You may need to generalize your problem defn to make an inductive proof go through.
4 A Fast Implementation

How can we implement this algorithm efficiently?

- Let \( t \) be latest free slot.
- Observe that \( t \) decreases monotonically – we always schedule something in latest free slot for any job.
- Each time \( t \) decreases, it may pass \( e_i \) for one or more jobs \( i \). These jobs join eligible set \( \Sigma \).
- \( t \) may also pass \( s_i \) for one or more unscheduled jobs. These jobs are deleted from \( \Sigma \) and are unschedulable.
- At any time, we want the job in \( \Sigma \) with largest \( s_i \).

These observations suggest the right data structure.

- Make sorted list \( L \) of jobs in \( P \) in decreasing order by \( e_i \).
- \( t \leftarrow \infty \)
- Let \( Q \) be a max-first priority queue, keyed by job start times.
- Repeat following loop until \( t = 0 \) or \( L \) is empty:
  - If \( Q \) is empty, \( t \leftarrow \) ending time of next job in \( L \).
  - Otherwise, \( t \leftarrow t - 1 \).
  - Move all jobs \( j \) with \( e_j = t \) from \( L \) to \( Q \).
  - \( i \leftarrow Q.\text{ExtractMax}() \)
  - Schedule job \( i \) at time \( t \)
  - Extract and discard any jobs with start time \( t \) from \( Q \).

Running time?

- Let \( n \) be number of jobs in \( P \).
- Each job is added to \( Q \) exactly once and removed exactly once.
- Total cost: \( O(n \log n) \) for binary heap.
- \( t \) is decremented only as many times as a job is scheduled.
- Total cost: \( O(n) \).
- List must initially be sorted.
- Total cost: \( O(n \log n) \).
- Conclude that overall cost is \( O(n \log n) \).