Interval scheduling

- Job $j$ starts at $s_j$ and finishes at $f_j$.
- Two jobs compatible if they don’t overlap.
- Goal: find maximum subset of mutually compatible jobs.

![Interval scheduling diagram]

Interval scheduling: greedy algorithms

**Greedy template.** 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

**Greedy template.** Consider jobs in some natural order. Take each job provided it’s compatible with the ones already taken.

- **Counterexample for earliest start time**
- **Counterexample for shortest interval**
- **Counterexample for fewest conflicts**

Interval scheduling: earliest-finish-time-first algorithm

**EARELIEST-FINISH-TIME-FIRST** \((n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)\)

**SORT** jobs by finish time so that \(f_1 \leq f_2 \leq \ldots \leq f_n\)

\[ A \leftarrow \emptyset \quad \text{set of jobs selected} \]

**FOR** \(j = 1 \text{ TO } n\)

**IF** job \(j\) is compatible with \(A\)

\[ A \leftarrow A \cup \{j\} \]

**RETURN** \(A\)

**Proposition.** Can implement earliest-finish-time first in \(O(n \log n)\) time.

- Keep track of job \(j^*\) that was added last to \(A\).
- Job \(j\) is compatible with \(A\) iff \(s_j \geq f_{j^*}\).
- Sorting by finish time takes \(O(n \log n)\) time.
Interval scheduling: analysis of earliest-finish-time-first algorithm

**Theorem.** The earliest-finish-time-first algorithm is optimal.

**Pf.** [by contradiction]
- Assume greedy is not optimal, and let’s see what happens.
- Let \( i_1, i_2, \ldots, i_k \) denote set of jobs selected by greedy.
- Let \( j_1, j_2, \ldots, j_m \) denote set of jobs in an optimal solution with \( i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r \) for the largest possible value of \( r \).

![Diagram showing greedy and optimal schedules]

Such a job can’t exist!

---

Interval scheduling: analysis of earliest-finish-time-first algorithm

**Theorem.** The earliest-finish-time-first algorithm is optimal.

**Pf.** [by contradiction]
- Assume greedy is not optimal, and let’s see what happens.
- Let \( i_1, i_2, \ldots, i_k \) denote set of jobs selected by greedy.
- Let \( j_1, j_2, \ldots, j_m \) denote set of jobs in an optimal solution with \( i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r \) for the largest possible value of \( r \).

![Diagram showing greedy and optimal schedules]

Solution still feasible and optimal (but contradicts maximality of \( r \))
Interval partitioning

Interval partitioning.

- Lecture \( j \) starts at \( s_j \) and finishes at \( f_j \).
- Goal: find minimum number of classrooms to schedule all lectures
  so that no two lectures occur at the same time in the same room.

Ex. This schedule uses 4 classrooms to schedule 10 lectures.

---

Interval partitioning

Interval partitioning.

- Lecture \( j \) starts at \( s_j \) and finishes at \( f_j \).
- Goal: find minimum number of classrooms to schedule all lectures
  so that no two lectures occur at the same time in the same room.

Ex. This schedule uses 3 classrooms to schedule 10 lectures.
Interval partitioning: greedy algorithms

**Greedy template.** Consider lectures in some natural order. Assign each lecture to an available classroom (which one?); allocate a new classroom if none are available.

- **[Earliest start time]** Consider lectures in ascending order of $s_i$.
- **[Earliest finish time]** Consider lectures in ascending order of $f_i$.
- **[Shortest interval]** Consider lectures in ascending order of $f_i - s_i$.
- **[Fewest conflicts]** For each lecture $i$, count the number of conflicting lectures $c_i$. Schedule in ascending order of $c_i$.

---

**Interval partitioning: greedy algorithms**

**Greedy template.** Consider lectures in some natural order. Assign each lecture to an available classroom (which one?); allocate a new classroom if none are available.

**Counterexample for earliest finish time**

```
3  
2  
1  
```

**Counterexample for shortest interval**

```
3  
2  
1  
```

**Counterexample for fewest conflicts**

```
3  
2  
1  
```
Interval partitioning: earliest-start-time-first algorithm

**EARLIEST-START-TIME-FIRST** \((n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)\)

**Algorithm**
- Sort lectures by start time so that \(s_1 \leq s_2 \leq \ldots \leq s_n\).
- \(d \leftarrow 0\) \hspace{1cm} number of allocated classrooms
- For \(j = 1\) to \(n\)
  - If lecture \(j\) is compatible with some classroom
    - Schedule lecture \(j\) in any such classroom \(k\).
  - Else
    - Allocate a new classroom \(d + 1\).
    - Schedule lecture \(j\) in classroom \(d + 1\).
    - \(d \leftarrow d + 1\)
- **RETURN** schedule.

**Proposition.** The earliest-start-time-first algorithm can be implemented in \(O(n \log n)\) time.

**Proof.** Store classrooms in a priority queue (key = finish time of its last lecture).
- To determine whether lecture \(j\) is compatible with some classroom, compare \(s_j\) to key of min classroom \(k\) in priority queue.
- To add lecture \(j\) to classroom \(k\), increase key of classroom \(k\) to \(f_j\).
- Total number of priority queue operations is \(O(n)\).
- Sorting by start time takes \(O(n \log n)\) time.

**Remark.** This implementation chooses the classroom \(k\) whose finish time of its last lecture is the earliest.
Interval partitioning: lower bound on optimal solution

**Def.** The depth of a set of open intervals is the maximum number that contain any given time.

**Key observation.** Number of classrooms needed $\geq$ depth.

**Q.** Does number of classrooms needed always equal depth?  
**A.** Yes! Moreover, earliest-start-time-first algorithm finds one.

![Interval partitioning diagram](image)

Interval partitioning: analysis of earliest-start-time-first algorithm

**Observation.** The earliest-start-time first algorithm never schedules two incompatible lectures in the same classroom.

**Theorem.** Earliest-start-time-first algorithm is optimal.  
**Pf.**
- Let $d$ = number of classrooms that the algorithm allocates.
- Classroom $d$ is opened because we needed to schedule a lecture, say $j$, that is incompatible with all $d-1$ other classrooms.
- These $d$ lectures each end after $s_j$.
- Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than $s_j$.
- Thus, we have $d$ lectures overlapping at time $s_j + \varepsilon$.
- Key observation $\Rightarrow$ all schedules use $\geq d$ classrooms.

Greedy analysis strategies

**Greedy algorithm stays ahead.** Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.
Greedy analysis strategies

Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.

Structural. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

Other greedy algorithms. Gale-Shapley, Kruskal, Prim, Dijkstra, Huffman, ...