4.1 Interval Scheduling
Interval Scheduling

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.

![Diagram of interval scheduling](image-url)
Interval Scheduling: Greedy Algorithms

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

- [Earliest start time] Consider jobs in ascending order of start time \( s_j \).
- [Earliest finish time] Consider jobs in ascending order of finish time \( f_j \).
- [Shortest interval] Consider jobs in ascending order of interval length \( f_j - s_j \).
- [Fewest conflicts] For each job, count the number of conflicting jobs \( c_j \). Schedule in ascending order of conflicts \( c_j \).
Interval Scheduling: Greedy Algorithms

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

- Breaks earliest start time
- Breaks shortest interval
- Breaks fewest conflicts
Greedy algorithm. Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

Sort jobs by finish times so that \( f_1 \leq f_2 \leq \ldots \leq f_n \).

```
A ← ∅
for j = 1 to n {
    if (job j compatible with A)
        A ← A ∪ {j}
}
return A
```

Implementation. \( O(n \log n) \).
- Remember job \( j^* \) that was added last to \( A \).
- Job \( j \) is compatible with \( A \) if \( s_j \geq f_{j^*} \).
Interval Scheduling

Time

0 1 2 3 4 5 6 7 8 9 10 11

A
B
C
D
E
F
G
H
Interval Scheduling
Interval Scheduling

Time

0 1 2 3 4 5 6 7 8 9 10 11

A

B

C

D

E

F

G

H

B

C
Interval Scheduling

Time

0 1 2 3 4 5 6 7 8 9 10 11

B C A E D F G H

B A
Interval Scheduling

- A
- B
- C
- D
- E
- F
- G
- H
Interval Scheduling

Time

0 1 2 3 4 5 6 7 8 9 10 11

A

B

C

D

E

F

G

H
Interval Scheduling

Time

0 1 2 3 4 5 6 7 8 9 10 11
Interval Scheduling
4.1 Interval Partitioning
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 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 occur at the same time in the same room.

Ex: This schedule uses only 3.
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.

**Ex:** Depth of schedule below = 3 \( \Rightarrow \) schedule below is optimal.

\[
\begin{array}{cccccc}
9 & 9:30 & 10 & 10:30 & 11 & 11:30 \\
\hline
9:30 & 10:30 & 11:30 & 12:30 & 1 & 1:30 \\
\hline
\end{array}
\]

a, b, c all contain 9:30

**Q.** Does there always exist a schedule equal to depth of intervals?
Interval Partitioning: Greedy Algorithm

**Greedy algorithm.** Consider lectures in increasing order of start time: assign lecture to any compatible classroom.

Sort intervals by starting time so that $s_1 \leq s_2 \leq \ldots \leq s_n$.

\[
d \leftarrow 0 \quad \text{—— number of allocated classrooms}
\]

\[
\text{for } j = 1 \text{ to } n \{
\quad \text{if (lecture } j \text{ is compatible with some classroom } k) }
\quad \text{schedule lecture } j \text{ in classroom } k
\text{ else }
\quad \text{allocate a new classroom } d + 1
\quad \text{schedule lecture } j \text{ in classroom } d + 1
\quad d \leftarrow d + 1
\}
\]

**Implementation.** $O(n \log n)$.
- For each classroom $k$, maintain the finish time of the last job added.
- Keep the classrooms in a priority queue.
**Interval Partitioning: Greedy Analysis**

**Observation.** Greedy algorithm never schedules two incompatible lectures in the same classroom.

**Theorem.** Greedy algorithm is optimal.

**Pf.**
- Let \( d \) = number of classrooms that the greedy algorithm allocates.

- Classroom \( d \) was allocated because we needed to schedule a job, say \( j \), that is incompatible with all \( d-1 \) other classrooms.

- 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 \( s_j + \epsilon \).

- Key observation \( \Rightarrow \) all schedules use \( \geq d \) classrooms.
Chapter 4

Greedy Algorithms

Algorithm Design
JON KLEINBERG • ÉVA TARDOS

Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.
4.4 Shortest Paths in a Graph

shortest path from Princeton CS department to Einstein's house
Shortest Path Problem

Shortest path network.
- Directed graph $G = (V, E)$.
- Source $s$, target $t$.
- Length $\ell_e$ = length of edge $e$.

Shortest path problem: find shortest directed path from $s$ to $t$.

Cost of path = sum of edge costs in path

Cost of path $s$-2-3-5-$t$ = $9 + 23 + 2 + 16$ = 50.
Dijkstra's Algorithm

**Dijkstra's algorithm.**

- Maintain a set of **explored nodes** $S$ for which we have determined the shortest path distance $d(u)$ from $s$ to $u$.

- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly (greedily) choose unexplored node $v \not\in S$ which minimizes

$$\partial(v) = \min_{e = (u, v) : u \in S} d(u) + \ell_e,$$

add $v$ to $S$, and set $d(v) = \partial(v)$.  

**Diagram:**
- Graph with nodes $s$, $u$, and $v$.
- Edge $\ell_e$ from $u$ to $v$.
- Gray shaded area represents explored nodes.
- Yellow highlight shows shortest path to some $u$ in explored part, followed by a single edge $(u, v)$. 


Dijkstra's Shortest Path Algorithm

Find shortest path from s to t.
Dijkstra's Shortest Path Algorithm

S = \{ \}
PQ = \{ s, 2, 3, 4, 5, 6, 7, t \}
Dijkstra's Shortest Path Algorithm

\[ S = \{ \} \]
\[ PQ = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s \} \]
\[ PQ = \{ 2, 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s \} \]
\[ PQ = \{ 2, 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2 \} \]
\[ PQ = \{ 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2 \} \]
\[ PQ = \{ 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2 \} \]
\[ PQ = \{ 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6 \} \]
\[ PQ = \{ 3, 4, 5, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6 \} \]

\[ PQ = \{ 3, 4, 5, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6, 7 \} \]
\[ PQ = \{ 3, 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6, 7 \} \]

\[ PQ = \{ 3, 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 6, 7 \} \]
\[ PQ = \{ 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

S = \{ s, 2, 3, 6, 7 \}
PQ = \{ 4, 5, t \}
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 5, 6, 7 \} \]
\[ PQ = \{ 4, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 5, 6, 7 \} \]
\[ PQ = \{ 4, t \} \]
Dijkstra's Shortest Path Algorithm

\[
S = \{ s, 2, 3, 4, 5, 6, 7 \}
PQ = \{ t \}
\]
Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 3, 4, 5, 6, 7 \}$

$PQ = \{ t \}$
Dijkstra's Shortest Path Algorithm

\[ S = \{s, 2, 3, 4, 5, 6, 7, t\} \]
\[ PQ = \{\} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
\[ PQ = \{ \} \]
**Dijkstra's Algorithm: Proof of Correctness**

**Invariant.** For each node $u \in S$, $d(u)$ is the length of the shortest $s$-$u$ path.

**Pf.** (by induction on $|S|$)

**Base case:** $|S| = 1$ is trivial.

**Inductive hypothesis:** Assume true for $|S| = k \geq 1$.

- Let $v$ be next node added to $S$, and let $(u,v)$ be the chosen edge.
- The shortest $s$-$u$ path plus $(u,v)$ is an $s$-$v$ path of length $\partial(v)$.
- Consider any $s$-$v$ path $P$. We'll see that it's no shorter than $\partial(v)$.
- Let $x$-$y$ be the first edge in $P$ that leaves $S$, and let $P'$ be the subpath to $x$.
- $P$ is already too long as soon as it leaves $S$.

$$l(P) \geq l(P') + l(x,y) \geq d(x) + l(x,y) \geq \partial(y) \geq \partial(v)$$

- nonnegative weights
- inductive hypothesis
- defn of $\partial(y)$
- Dijkstra chose $v$ instead of $y$
For each unexplored node \( v \notin S \), explicitly maintain
\[
\partial(v) = \min_{e = (u, v) : u \in S} d(u) + \ell_e,
\]

- Next node to explore = node \( v \notin S \) with minimum \( \partial(v) \).
- When exploring \( v \), for each incident edge \( e = (v, w) \), \( w \notin S \), update
\[
\partial(w) = \min \{ \partial(w), \partial(v) + \ell_e \}.
\]

**Efficient implementation.** Maintain a priority queue of unexplored nodes, prioritized by \( \partial(v) \).

<table>
<thead>
<tr>
<th>PQ Operation</th>
<th>Dijkstra</th>
<th>Array</th>
<th>Binary heap</th>
<th>d-way Heap</th>
<th>Fib heap †</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>( n )</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( d \log_d n )</td>
<td>( 1 )</td>
</tr>
<tr>
<td>ExtractMin</td>
<td>( n )</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( d \log_d n )</td>
<td>( \log n )</td>
</tr>
<tr>
<td>ChangeKey</td>
<td>( m )</td>
<td>( 1 )</td>
<td>( \log n )</td>
<td>( \log_d n )</td>
<td>( 1 )</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>( n )</td>
<td>( 1 )</td>
<td>( 1 )</td>
<td>( 1 )</td>
<td>( 1 )</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>( n^2 )</td>
<td>( m \log n )</td>
<td>( m \log_{m/n} n )</td>
<td>( m + n \log n )</td>
<td></td>
</tr>
</tbody>
</table>

† Individual ops are amortized bounds
Chapter 4

Greedy Algorithms

Algorithm Design

JON KLEINBERG · ÉVA TARDOS

Copyright © 2005 Pearson-Addison Wesley. All rights reserved.