Constraint-Based Scheduling

Joseph Scott

Department of Information Technology
Uppsala University

Constraint Programming, HT’13
1. What is Scheduling?
   Example scheduling problems
   The general case

2. Resource Constrained Scheduling
   Introduction
   Global constraint: \textit{cumulative}

3. Propagation of the cumulative constraint
   Time Table
   Overload Checking
   Edge-Finding
   Other \textit{cumulative} propagation algorithms

4. Conclusion
1. What is Scheduling?
   Example scheduling problems
   The general case

2. Resource Constrained Scheduling
   Introduction
   Global constraint: cumulative

3. Propagation of the cumulative constraint
   Time Table
   Overload Checking
   Edge-Finding
   Other cumulative propagation algorithms

4. Conclusion
From the Modelling lecture:

Example (The Sport Scheduling Problem, SSP)

Find schedule in $Periods \times Weeks \rightarrow Teams \times Teams$ for:

- $|Teams| = n$
- $|Weeks| = n - 1$
- $|Periods| = n/2$

subject to the following constraints:

- Each team plays exactly once against each other team.
- Each team plays exactly once per week.
- Each team plays at most twice per period.

Intuitive idea for a matrix model and a solution for $n = 8$:

<table>
<thead>
<tr>
<th></th>
<th>Wk 1</th>
<th>Wk 2</th>
<th>Wk 3</th>
<th>Wk 4</th>
<th>Wk 5</th>
<th>Wk 6</th>
<th>Wk 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>P 1</td>
<td>1 vs. 2</td>
<td>1 vs. 3</td>
<td>2 vs. 6</td>
<td>3 vs. 5</td>
<td>4 vs. 7</td>
<td>4 vs. 8</td>
<td>5 vs. 8</td>
</tr>
<tr>
<td>P 2</td>
<td>3 vs. 4</td>
<td>2 vs. 8</td>
<td>1 vs. 7</td>
<td>6 vs. 7</td>
<td>6 vs. 8</td>
<td>2 vs. 5</td>
<td>1 vs. 4</td>
</tr>
<tr>
<td>P 3</td>
<td>5 vs. 6</td>
<td>4 vs. 6</td>
<td>3 vs. 8</td>
<td>1 vs. 8</td>
<td>1 vs. 5</td>
<td>3 vs. 7</td>
<td>2 vs. 7</td>
</tr>
<tr>
<td>P 4</td>
<td>7 vs. 8</td>
<td>5 vs. 7</td>
<td>4 vs. 5</td>
<td>2 vs. 4</td>
<td>2 vs. 3</td>
<td>1 vs. 6</td>
<td>3 vs. 6</td>
</tr>
</tbody>
</table>
A different scheduling model

Example (The Job-Shop Scheduling Problem)

- \( m \) machines, each processing one operation at a time
- \( n \) jobs, \( \text{job}_i = \langle \text{op}_1^i, \text{op}_2^i, \ldots, \text{op}_m^i \rangle \)
  - a job is a sequence of operations, \( \text{op}_j^i \) where each:
    - executes on a specific machine
    - lasts a fixed time
  - operation order is fixed: \( \prec \ldots \prec \text{op}_m^i \)
A different scheduling model

Example (The Job-Shop Scheduling Problem)

- \( m \) machines, each processing one operation at a time
- \( n \) jobs, \( job_i = \langle op_{i1}, op_{i2}, \ldots, op_{im} \rangle \)
  - a job is a sequence of operations, \( op_{ij} \) where each:
    - executes on a specific machine
    - lasts a fixed time
  - operation order is fixed: \( \prec \ldots \prec op_{im} \)
A different scheduling model

Example (The Job-Shop Scheduling Problem)

- **m** machines, each processing one operation at a time
- **n** jobs, \( \text{job}_i = \langle \text{op}_1^i, \text{op}_2^i, \ldots, \text{op}_m^i \rangle \)
  - a job is a sequence of operations, \( \text{op}_j^i \)
    - executes on a specific machine
    - lasts a fixed time
  - operation order is fixed: \( \text{op}_1^i \prec \text{op}_2^i \prec \ldots \prec \text{op}_m^i \)
  - \( a \prec b \) means “b cannot start until a ends”
A different scheduling model

Example (The Job-Shop Scheduling Problem)

- \( m \) machines, each processing one operation at a time
- \( n \) jobs, \( \text{job}_i = \langle \text{op}_1^i, \text{op}_2^i, \ldots, \text{op}_m^i \rangle \)
  - a job is a sequence of operations, \( \text{op}_j^i \)
    - executes on a specific machine
    - lasts a fixed time
  - operation order is fixed: \( \text{op}_1^i \prec \text{op}_2^i \prec \ldots \prec \text{op}_m^i \)

\( a \prec b \) means “\( b \) cannot start until \( a \) ends”
A different scheduling model

Example (The Job-Shop Scheduling Problem)

- **m** machines, each processing one operation at a time
- **n** jobs, $job_i = \langle op^i_1, op^i_2, \ldots, op^i_m \rangle$
  - a job is a sequence of operations, $op^i_j$
    - executes on a specific machine
    - lasts a fixed time
  - operation order is fixed: $op^i_1 \prec op^i_2 \prec \ldots \prec op^i_m$

$a \prec b$ means "$b$ cannot start until $a$ ends"
A different scheduling model

Example (The Job-Shop Scheduling Problem)

- \( m \) machines, each processing one operation at a time
- \( n \) jobs, \( \text{job}_i = \langle \text{op}_1^i, \text{op}_2^i, \ldots, \text{op}_m^i \rangle \)
  - a job is a sequence of operations, \( \text{op}_j^i \)
    - executes on a specific machine
    - lasts a fixed time
  - operation order is fixed: \( \text{op}_1^i < \text{op}_2^i \prec \ldots \prec \text{op}_m^i \)

\( a \prec b \) means “\( b \) cannot start until \( a \) ends”
1. What is Scheduling?
   Example scheduling problems
   The general case

2. Resource Constrained Scheduling
   Introduction
   Global constraint: $cumulative$

3. Propagation of the cumulative constraint
   Time Table
   Overload Checking
   Edge-Finding
   Other $cumulative$ propagation algorithms

4. Conclusion
What is scheduling?
[Baker & Trietsch, 2009]

- **Given:**
  - Set of tasks,
  - each of some duration,
  - sharing one or more finite resources.

- **Need:**
  - A feasible execution sequence
  - that respects the limitations of the resources.

- **Additional Constraints:**
  - precedence: $a$ must finish before $b$ begins
  - sequence: task uses several resources in fixed order
  - objective: minimize makespan, minimize simultaneous resource usage, etc.
  - . . .
What is scheduling?
[Baker & Trietsch, 2009]

■ Given:
  • Set of tasks,
  • each of some duration,
  • sharing one or more finite resources.

■ Need:
  • A feasible execution sequence
  • that respects the limitations of the resources.

■ Additional Constraints:
  • precedence: $a$ must finish before $b$ begins
  • sequence: task uses several resources in fixed order
  • objective: minimize makespan, minimize simultaneous resource usage, etc.
  • ...
What is scheduling?
[Baker & Trietsch, 2009]

- Given:
  - Set of tasks,
  - each of some duration,
  - sharing one or more finite resources.

- Need:
  - A feasible execution sequence
  - that respects the limitations of the resources.

- Additional Constraints:
  - precedence: $a$ must finish before $b$ begins
  - sequence: task uses several resources in fixed order
  - objective: minimize makespan, minimize simultaneous resource usage, etc.
  - ...
What is (scheduling)?

- **Sequencing**
  - relax the condition that tasks have a duration
  - Instead of execution times, just compute an ordering.

- **Planning**
  - Many possible tasks, must select which ones to execute.
  - Goal can be reached by multiple combinations of tasks.
  - (Usually) does not consider durations.
Outline

1. What is Scheduling?
   Example scheduling problems
   The general case

2. Resource Constrained Scheduling
   Introduction
   Global constraint: cumulative

3. Propagation of the cumulative constraint
   Time Table
   Overload Checking
   Edge-Finding
   Other cumulative propagation algorithms

4. Conclusion
The Resource Constrained Scheduling Problem (RCSP)

- A finite, discrete resource
  - Examples
    - machine with limited processing capacity
    - fixed number of available employees
    - etc.
  - Resource is limited, but not consumable
    - capacity limits the number of tasks processed at one time
    - the resource is not depleted over time

- Each task:
  - requires part of the resource’s capacity,
  - lasts for some amount of time,
  - has a domain of valid start times.
The Resource Constrained Scheduling Problem (RCSP)

- A finite, discrete resource
  - Examples
    - machine with limited processing capacity
    - fixed number of available employees
    - etc.
  - Resource is limited, but not consumable
    - capacity limits the number of tasks processed at one time
    - the resource is not depleted over time

- Each task:
  - requires part of the resource’s capacity,
  - lasts for some amount of time,
  - has a domain of valid start times.
Variants of RCSP

- **Capacity of Resource**
  - disjunctive: only one task executes at a time
  - cumulative: resource has a capacity that can never be exceeded

- **Elasticity of Tasks**
  - Inelastic: duration and resource requirements are fixed
  - Elastic: resource usage and/or duration are flexible

- **Interruptibility of Tasks**
  - Preemptive: tasks may interrupt each other
  - Non-preemptive: once started, a task continues until completion

**Today**
non-preemptive, inelastic, cumulative scheduling
Variants of RCSP

- **Capacity of Resource**
  - disjunctive: only one task executes at a time
  - cumulative: resource has a capacity that can never be exceeded

- **Elasticity of Tasks**
  - Inelastic: duration and resource requirements are fixed
  - Elastic: resource usage and/or duration are flexible

- **Interruptibility of Tasks**
  - Preemptive: tasks may interrupt each other
  - Non-preemptive: once started, a task continues until completion

**Today**

non-preemptive, inelastic, cumulative scheduling
Notation for cumulative scheduling problems
Part 1: Tasks

Notation
Specific tasks are written $A, B, \ldots$, while variables referring to some task are written $i, j, \ldots$

- Set $Tasks$ of $n$ tasks, where for $i \in Tasks$:
  - fixed resource requirement: $use_i$
  - fixed duration: $dur_i$
  - energy: $energy_i = use_i \cdot dur_i$

- One shared resource of constant capacity $C$. 

![Diagram showing time axis and resource usage](image-url)
Notation for cumulative scheduling problems
Part 1: Tasks

Notation

Specific tasks are written \( A, B, \ldots \),
while variables referring to some task are written \( i, j, \ldots \)

- Set \( \text{Tasks} \) of \( n \) tasks, where for \( i \in \text{Tasks} \):
  - fixed resource requirement: \( \text{use}_i \)
  - fixed duration: \( \text{dur}_i \)
  - energy: \( \text{energy}_i = \text{use}_i \cdot \text{dur}_i \)

- One shared resource of constant capacity \( C \).

Diagram:

- Time axis from 0 to 10
- Resource axis from 0 to 2
- Task \( D \) with duration \( \text{dur}_D \) and energy \( \text{energy}_D \)
- Resource usage \( \text{use}_D \)
Notation for cumulative scheduling problems
Part 1: Tasks

Notation

Specific tasks are written $A, B, \ldots$, while variables referring to some task are written $i, j, \ldots$

- Set $Tasks$ of $n$ tasks, where for $i \in Tasks$:
  - fixed resource requirement: $use_i$
  - fixed duration: $dur_i$
  - energy: $energy_i = use_i \cdot dur_i$
- One shared resource of constant capacity $C$.

![Diagram of cumulative scheduling](image)
Notation for cumulative scheduling problems
Part 2: Start Times

- Task $i$ has interval of feasible start times $\text{start}_i$
  - bounds: earliest start time ($\text{est}_i$), latest start time ($\text{lst}_i$)
  - $\text{start}_i \in [\text{est}_i .. \text{lst}_i]$
  - Prune $\text{start}_i$ by strengthening $\text{est}_i$ and/or $\text{lst}_i$

- $\text{dur}_i$ is fixed, relates start times to completion times
  - latest completion time ($\text{lct}_i$)

Important

Strengthening $\text{lct}_i$ is symmetric to strengthening $\text{est}_i$. 

![Diagram showing start and end times with intervals and bounds]
Notation for cumulative scheduling problems
Part 2: Start Times

- Task $i$ has interval of feasible start times $start_i$
  - bounds: earliest start time ($est_i$), latest start time ($lst_i$)
  - $start_i \in [est_i .. lst_i]$
  - Prune $start_i$ by strengthening $est_i$ and/or $lst_i$
- $dur_i$ is fixed, relates start times to completion times
  - latest completion time ($lct_i$)

**Important**

Strengthening $lct_i$ is symmetric to strengthening $est_i$. 

![Diagram showing task scheduling with start and completion times](image-url)
Outline

1. What is Scheduling?
   - Example scheduling problems
   - The general case

2. Resource Constrained Scheduling
   - Introduction
   - Global constraint: cumulative

3. Propagation of the cumulative constraint
   - Time Table
   - Overload Checking
   - Edge-Finding
   - Other cumulative propagation algorithms

4. Conclusion
The cumulative constraint

- **Decision variables:** \( \forall i \in Tasks : start_i \)

- **Constraint:**

  \[
  \forall t \in time : \quad \sum_{i \in Tasks} use_i \leq C \\
  \text{subject to: } \quad \text{start}_i \leq t \leq \text{start}_i + \text{dur}_i
  \]

  - Time is discrete, not continuous.
  - Interested in enforcing bounds consistency only.

- Could decompose this into a series of linear constraints; prefer to use a global constraint to capture the structure of the problem.
Outline

1. What is Scheduling?
   - Example scheduling problems
   - The general case

2. Resource Constrained Scheduling
   - Introduction
   - Global constraint: cumulative

3. Propagation of the cumulative constraint
   - Time Table
   - Overload Checking
   - Edge-Finding
   - Other cumulative propagation algorithms

4. Conclusion
Time tabling reasons on required parts

![Diagram showing time tabling with estimated start (est) and latest completion (lct) for tasks A and B.](image)
What if there is no required part?

\[ \text{est}_A = 0 \quad \text{lct}_A = 6 \]

Diagram showing time intervals with labeled 'A'.
Outline

1. What is Scheduling?
   Example scheduling problems
   The general case

2. Resource Constrained Scheduling
   Introduction
   Global constraint: cumulative

3. Propagation of the cumulative constraint
   Time Table
   Overload Checking
   Edge-Finding
   Other cumulative propagation algorithms

4. Conclusion
Notation for cumulative scheduling problems
Part 3: Sets of Tasks

Sets of tasks (e.g., \(\{A, B, C\}\)) are denoted \(\omega, \theta,\) etc.

- Raise several of these concepts to apply to sets of tasks

\[
est_\omega = \min_{i \in \omega} (est_i) \quad lct_\omega = \max_{i \in \omega} (lct_i) \quad energy_\omega = \sum_{i \in \omega} (energy_i)
\]

\[
est_{A,D} = 0 \quad lct_{A,B,C} = 5 \quad lct_D = 10
\]

\(est\) and \(lct\) values for tasks A, B, C, and D.
Notation for cumulative scheduling problems
Part 3: Sets of Tasks

Notation
Sets of tasks (e.g., \{A, B, C\}) are denoted \( \omega, \theta, \) etc.

- Raise several of these concepts to apply to sets of tasks

\[
est_\omega = \min_{i \in \omega} (est_i) \quad lct_\omega = \max_{i \in \omega} (lct_i) \quad energy_\omega = \sum_{i \in \omega} (energy_i)
\]

\[
est_{\{A,B,C\}} = 0 \quad lct_{\{A,B,C\}} = 5 \quad lct_D = 10
\]
e-feasibility and overload checking

- What if $use_i$ and $dur_i$ were not fixed, but the energy was?
  - Recall the elastic problem type
  - Same area, different shape

Overload Rule:
\[
\forall \theta \subseteq tasks: energy_\theta > C(lct_\theta - est_\theta) \implies \text{Overload}
\]

- **e-feasible**: no overload for any $\theta \subseteq Tasks$
Is e-feasibility stronger than time tabling?

- $est_{A,B} = 0$
- $lct_{A,B} = 4$

- $est_{A,B} = 0$
- $lct_{A,B} = 4$
Is time tabling stronger than e-feasibility?

A trivial overload, not e-feasible, but ignored by time tabling.
Is time tabling stronger than e-feasibility?

A trivial overload, not e-feasible, but ignored by time tabling.
Is time tabling stronger than e-feasibility?

A trivial overload, not e-feasible, but ignored by time tabling.
How do we make an effective propagator for cumulative?

- time table and e-feasibility miss different overload conditions
  - time table considers tasks exactly, but in isolation
  - e-feasibility considers tasks in combination, but approximately

- A *cumulative* propagator should consider **both**

- Most common solution:
  - Run several different propagation algorithms in sequence
  - Gecode’s *cumulative*: time table, overload checking, plus edge-finding...
Outline

1. What is Scheduling?
   Example scheduling problems
   The general case

2. Resource Constrained Scheduling
   Introduction
   Global constraint: cumulative

3. Propagation of the cumulative constraint
   Time Table
   Overload Checking
   Edge-Finding
   Other cumulative propagation algorithms

4. Conclusion
Improving on overload checking

- extend the idea of e-feasibility:
  1. Start with a set of tasks that is e-feasible.
  2. Add another task with overlapping start times.
  3. Limit the new task to the domain of the set, and check for e-feasibility again.
  4. If the new set is infeasible, prune the domain of the new task.

- Basis for a set of filtering algorithms used to propagate cumulative:
  - edge-finding
  - extended edge-finding
  - not-first/not-last
  - timetable edge-finding
Edge-finding: deducing new precedence relations

- Given a set of tasks, determine one task such that:
  - for every feasible solution,
  - that task must come first (or last).
- Is there a feasible schedule in which \( \omega = \{A, B, C\} \) and D both end by \( lct_\omega \)?
  - If not, D must end after \( lct_\omega \).

\[
\text{energy}_\omega + \text{energy}_D > C \cdot (lct_\omega - est_{\omega \cup \{D\}}) \implies \omega < D
\]

Running example due to [VILÍM, 2009]

```
est_\omega = 0  \quad lct_\omega = 5  \quad lct_D = 10
```

```
\begin{array}{c}
\text{A} \\
\text{B} \\
\text{C} \\
\text{D}
\end{array}
```

*Running example due to [VILÍM, 2009]*
Edge-finding: deducing new precedence relations

- Given a set of tasks, determine one task such that:
  - for every feasible solution,
  - that task must come first (or last).
- Is there a feasible schedule in which $\omega = \{A, B, C\}$ and $D$ both end by $lct_\omega$?
  - If not, $D$ must end after $lct_\omega$.

$$energy_\omega + energy_D > C \cdot (lct_\omega - est_{\omega \cup \{D\}}) \implies \omega < D$$

Running example due to [VIĹM, 2009]
Edge-finding: deducing new precedence relations

- Given a set of tasks, determine one task such that:
  - for every feasible solution,
  - that task must come first (or last).
- Is there a feasible schedule in which \( \omega = \{A, B, C\} \) and D both end by \( \text{lct}_\omega \)?
  - If not, D must end after \( \text{lct}_\omega \).

\[
\text{energy}_\omega + \text{energy}_D > C \cdot (\text{lct}_\omega - \text{est}_{\omega \cup \{D\}}) \quad \Rightarrow \quad \omega \lessdot D
\]

"D ends after the end of all tasks in \( \omega \)"

Running example due to [VILÍM, 2009]
How much can the bound be updated?

Consider $energy_\omega$ in two parts:
- energy which may be scheduled without affecting $D$,
- and the $rest$, which must affect $D$.

$$est'_i \geq est_\omega + \left\lceil \frac{rest(\omega, use_i)}{use_i} \right\rceil$$

<table>
<thead>
<tr>
<th>$est_\omega = 0$</th>
<th>$lct_\omega = 5$</th>
<th>$lct_D = 10$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>$rest(\omega, use_D)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$C - use_D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

$$rest(\omega, use_i) = \begin{cases} energy_\omega - (C - use_i)(lct_\omega - est_\omega) & \text{if } \omega \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$
How much can the bound be updated?

- Consider \( \text{energy}_\omega \) in two parts:
  - energy which may be scheduled without affecting \( D \),
  - and the \( \text{rest} \), which \textit{must} affect \( D \).

\[
est'_i \geq \text{est}_\omega + \left\lceil \frac{\text{rest}(\omega, \text{use}_i)}{\text{use}_i} \right\rceil
\]

\[
est\omega = 0 \quad \text{lct}_\omega = 5 \quad \text{lct}_D = 10
\]

\[
est\omega - \text{lct}\omega \quad \text{rest}(\omega, \text{use}_D)
\]

\[	ext{rest}(\omega, \text{use}_i) = \begin{cases} 
\text{energy}_\omega - (C - \text{use}_i)(\text{lct}_\omega - \text{est}_\omega) & \text{if } \omega \neq \emptyset \\
0 & \text{otherwise}
\end{cases}
\]
How much can the bound be updated?

Consider $energy_\omega$ in two parts:
- energy which may be scheduled without affecting $D$,
- and the rest, which must affect $D$.

$$est'_i \geq est_\omega + \left\lceil \frac{rest(\omega, use_i)}{use_i} \right\rceil$$

<table>
<thead>
<tr>
<th>$est_\omega = 0$</th>
<th>$est'_D = 3$</th>
<th>$lct_\omega = 5$</th>
<th>$lct_D = 10$</th>
</tr>
</thead>
</table>

$rest(\omega, use_i) = \begin{cases} energy_\omega - (C - use_i)(lct_\omega - est_\omega) & \text{if } \omega \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$
Minimum Slack

- Slack measures the capacity not used by a task set $\omega$

$$\text{Slack}(\omega) = C(lct_{\omega} - est_{\omega}) - e_{\omega}$$

- If $energy_i > \text{Slack}(\omega)$, then part of $i$ falls outside $[est_{\omega}, lct_{\omega}]$

Conjecture
For a fixed $est$ and $lct$, the set of tasks with the minimum slack is the most likely to conflict with the scheduling of other tasks.
Task Intervals

- There are only $O(n)$ meaningful values for each bound
- Task intervals: sets of tasks, bounded by tasks

$$\omega^U_L = \{ i \in T | est_i \geq est_L \land lct_i \leq lct_U \}$$

- $\omega_A^B = \{ A, B, C \}$
- Removing task $C$ reduces the energy, but not the available capacity
Task Intervals

- There are only $O(n)$ meaningful values for each bound
- Task intervals: sets of tasks, bounded by tasks

$$\omega^U_L = \{i \in T \mid est_i \geq est_L \land lct_i \leq lct_U\}$$

- $\omega_A^B = \{A, B, C\}$
- Removing task $C$ reduces the energy, but not the available capacity
Using intervals to perform edge-finding
[Baptiste et al., 2001]

- General task interval based edge-finding algorithm:

  for every upper bound $U \in T$ do
  
  for every task $i \in T$ by decreasing $est_i$ do
  
  if $lct_i \leq lct_U$ then
  
  if $Slack(\omega_i^U) < Slack(\omega_L^U)$ then
  
  $L \leftarrow i$

  else

  if $\omega_L^U \leq i$ then

  $est_i = \text{update based on } \omega_L^U$

- Two problems:
  - Computing the update to $est_i$ is (generally) not $O(n)$.
  - The minimum slack interval is not always the best interval.
Using intervals to perform edge-finding
[Baptiste et al., 2001]

- General task interval based edge-finding algorithm:
  
  for every upper bound $U \in T$ do
  
      for every task $i \in T$ by decreasing $est_i$ do
        if $lct_i \leq lct_U$ then
          if $Slack(\omega_i^U) < Slack(\omega_L^U)$ then
            $L \leftarrow i$
          else
            if $\omega_L^U \leq i$ then
              $est_i = \text{update based on } \omega_L^U$
        
  - Two problems:
    - Computing the update to $est_i$ is (generally) not $O(n)$.
    - The minimum slack interval is not always the best interval.
Using intervals to perform edge-finding
[Baptiste et al., 2001]

- General task interval based edge-finding algorithm:

\[
\text{for every upper bound } U \in T \text{ do} \\
\quad \text{for every task } i \in T \text{ by decreasing } est_i \text{ do} \\
\quad \quad \text{if } lct_i \leq lct_U \text{ then} \\
\quad \quad \quad \text{if } \text{Slack}(\omega_i^U) < \text{Slack}(\omega_L^U) \text{ then} \\
\quad \quad \quad \quad L \leftarrow i \\
\quad \quad \text{else} \\
\quad \quad \quad \text{if } \omega_L^U \leq i \text{ then} \\
\quad \quad \quad \quad est_i = \text{update based on } \omega_L^U
\]

- Two problems:
  - Computing the update to \( est_i \) is (generally) not \( O(n) \).
  - The minimum slack interval is not always the best interval.
Arrange all tasks as the leaves of a balanced, binary tree.

- Sorted by increasing \( est \).

Let \( U \) be the task with the largest \( lct \)

- Notice: for any \( L \), the interval \( \omega^U_L \) is all tasks to the right.
Arrange all tasks as the leaves of a balanced, binary tree.

- Sorted by increasing $est$.

Let $U$ be the task with the largest $lct$

- Notice: for any $L$, the interval $\omega^U_L$ is all tasks to the right.
Goal: use interior nodes to compute the minimum slack of any subset of tasks in that node’s subtree
  • Find the best lower bound, $L$, for the current upper bound, $U$.

Minimum slack set is always a task interval, so only two choices for each node:
  • minimum slack set of right child, or
  • minimum slack of left child, and all tasks under right child.
Minimum Slack Becomes Energy Envelope

■ Rewrite the edge-finding rule:

\[
\begin{align*}
\text{energy}_{\theta \cup \{i\}} & > C (\text{lct}_\theta - \text{est}_{\theta \cup \{i\}}) \implies \theta < i \\
C \cdot \text{est}_{\theta \cup \{i\}} + \text{energy}_{\theta \cup \{i\}} & > C \cdot \text{lct}_\theta \implies \theta < i
\end{align*}
\]

■ Can compute the energy envelope of a node \( v \) (\( Env_v \)) by comparing child nodes, \( \ell \) and \( r \):

\[
Env_v = \begin{cases} 
C \cdot \text{est}_x + \text{energy}_x & \text{if } v \text{ is a leaf, for task } x \\
\max\{Env_r, Env_\ell + \text{energy}_r\} & \text{if } v \text{ is not a leaf}
\end{cases}
\]

\[
\text{energy}_v = \begin{cases} 
\text{energy}_x & \text{if } v \text{ is a leaf, for task } x \\
\text{energy}_r + \text{energy}_\ell & \text{if } v \text{ is not a leaf}
\end{cases}
\]

■ Maximizing energy envelope = minimizing slack
Tree Structure Allows for Efficient Recomputation

- So far, the tree requires more work, not less!
- Make $U$ a zero-energy task, then recompute $O(\log n)$ ancestor nodes.
- $n$ upper bounds, so $O(n \log n)$ to check them all.
  - Unfortunately, that only holds for unary resources...
Tree Structure Allows for Efficient Recomputation

- So far, the tree requires more work, not less!
- Make $U$ a zero-energy task, then recompute $O(\log n)$ ancestor nodes.
- $n$ upper bounds, so $O(n \log n)$ to check them all.
  - Unfortunately, that only holds for unary resources...
Tree Structure Allows for Efficient Recomputation

- So far, the tree requires more work, not less!
- Make $U$ a zero-energy task, then recompute $O(\log n)$ ancestor nodes.
- $n$ upper bounds, so $O(n \log n)$ to check them all.
  - Unfortunately, that only holds for unary resources...
So far, the tree requires more work, not less!
Make $U$ a zero-energy task, then recompute $O(\log n)$
ancestor nodes.
$n$ upper bounds, so $O(n \log n)$ to check them all.
• Unfortunately, that only holds for unary resources...
For cumulative resources, must consider subsets of $\omega$

- $\{B, C\}$ not energetic enough to require $\{B, C\} \preceq D$.
- $\{A, B, C\} \preceq D$, and $lct_{A,B,C} \geq lct_{B,C}$,
  - therefore $D$ must end after $lct_{B,C}$.

\[\begin{align*}
est_D &= 0 & \nest_{\{B,C\}} &= 2 & \lct_{\{B,C\}} &= 5 & \lct_D &= 10
\end{align*}\]

- Must find $\theta^u_\ell \subseteq \omega^U_L$ that yields the strongest update.
For cumulative resources, must consider subsets of $\omega$

- $\{B, C\}$ not energetic enough to require $\{B, C\} \preceq D$.
- $\{A, B, C\} \preceq D$, and $lct_{A,B,C} \geq lct_{B,C}$, therefore $D$ must end after $lct_{B,C}$.

![](image)

- Must find $\theta^u_\ell \subseteq \omega^U_L$ that yields the strongest update.
For cumulative resources, must consider subsets of $\omega$

- $\{B, C\}$ not energetic enough to require $\{B, C\} \preceq D$.
- $\{A, B, C\} \preceq D$, and $\text{lct}_{A,B,C} \geq \text{lct}_{B,C}$,
  - therefore $D$ must end after $\text{lct}_{B,C}$.

$$\begin{align*}
est_{\{A,B,C\}} &= \nest_D = 0 & \text{lct}_{\{A,B,C\}} &= \text{lct}_{\{B,C\}} = 5 & \text{lct}_D &= 10 \\
A &\quad & C &\quad & D \\
B &\quad & \text{Must find } \theta^u_{\ell} \subseteq \omega^U_L \text{ that yields the strongest update.} \end{align*}$$
For cumulative resources, must consider subsets of $\omega$

- $\{B, C\}$ not energetic enough to require $\{B, C\} \preceq D$.
- $\{A, B, C\} \preceq D$, and $lct_{A,B,C} \geq lct_{B,C}$,
  - therefore $D$ must end after $lct_{B,C}$.

\[
\begin{align*}
est_{\{A,B,C\}} &= \nest_D = 0 & lct_{\{A,B,C\}} &= lct_{\{B,C\}} = 5 & lct_D &= 10 \\
\end{align*}
\]

\[\text{est}'_D = 3\]

- Must find $\theta^U_\ell \subseteq \omega^U_L$ that yields the strongest update.
For cumulative resources, must consider subsets of $\omega$:

- $\{B, C\}$ not energetic enough to require $\{B, C\} \preceq D$.
- $\{A, B, C\} \preceq D$, and $lct_{A,B,C} \geq lct_{B,C}$,
  - therefore $D$ must end after $lct_{B,C}$.

Must find $\theta_{\ell}^u \subseteq \omega_L^U$ that yields the strongest update.
Impact on Overall Complexity

- For a given $U$, the minimum slack interval does correctly check the edge-finding condition.
- It does not always find the subset that produces the strongest bound update.
  - [Mercier & Van Hentenryck, 2008]
  - Includes dynamic programming approach with $O(kn^2)$ complexity,
    where $k$ is the number of distinct capacity requirements.
- [Vilím, 2009] shows how to use the $\Theta$-tree method outlined here to find the strongest subset with $O(kn \log n)$ complexity.
  - It is significantly more complicated than what you have seen today.
**Definition: Density**

- For a task interval $\theta$, the **density** of $\theta$ is given by:

$$Dens(\theta) = \frac{energy_\theta}{lct_\theta - est_\theta}$$

- In (most) cases where the interval responsible for the strongest update is not the interval of minimum slack, it is the interval of maximum density.
min slack/max density edge-finding algorithm
[Kameugne et al., 2011]

\[
\begin{align*}
\text{for } U \in T \text{ do} \\
\quad \text{for } i \in T \text{ by decreasing } est_i \text{ do} \\
\quad \quad \text{if } lct_i \leq lct_U \text{ then} \\
\quad \quad \quad \text{if } Dens(\omega^U_i) > Dens(\omega^U_L) \text{ then} \\
\quad \quad \quad \quad L \leftarrow i \\
\quad \quad \quad \text{else} \\
\quad \quad \quad \quad Dupd_i \leftarrow \text{update based on } \omega^U_L \\
\quad \quad \text{for } i \in T \text{ by increasing } est_i \text{ do} \\
\quad \quad \text{if } Slack(\theta^U_i) < Slack(\theta^U_\ell) \text{ then} \\
\quad \quad \quad \ell \leftarrow i \\
\quad \quad \quad \text{if } lct_i > lct_U \text{ then} \\
\quad \quad \quad \quad SLupd_i \leftarrow \text{update based on } \theta^U_\ell \\
\quad \quad \quad \text{if } \theta^U_\ell < i \text{ then} \\
\quad \quad \quad \quad est'_i \leftarrow \max(Dupd_i, SLupd_i)
\end{align*}
\]
A comparison of the algorithms

- $O(n^2)$ does not strictly dominate the $O(kn \log n)$ complexity of $\Theta$-tree edge-finding, especially for low values of $k$.
  - Since $k$ is bounded by $n$, $\Theta$-tree complexity varies from $O(n \log n)$ to $O(n^2 \log n)$.

- Furthermore, the maximum density algorithm may not always make the strongest update on the first iteration.
  - If an edge finding update exists, will always make some update.
  - Always makes the strongest update when:
    - $est_\theta \leq est_i$, or
    - $est_i < est_\theta \leq est_\rho$.
  - Number of weaker updates bounded in $O(n)$.

- A form of “lazy” evaluation
  - Makes sense to prune fast, then let a lower complexity propagator run again.
  - With the guarantee that the edge finder can then enforce the stronger update on a later iteration, if necessary.
Outline

1. **What is Scheduling?**
   - Example scheduling problems
   - The general case

2. **Resource Constrained Scheduling**
   - Introduction
   - Global constraint: *cumulative*

3. **Propagation of the cumulative constraint**
   - Time Table
   - Overload Checking
   - Edge-Finding
   - Other *cumulative* propagation algorithms

4. **Conclusion**
Extended edge-finding

- Recall the edge-finding rule:

\[ energy_\omega + energy_i > C \cdot (lct_\omega - est_{\omega \cup \{i\}}) \implies \omega < i \]

- In this example
  - no feasible schedule where I starts before 3,
  - but the edge-finding condition is not satisfied for \( \omega = \{G, H\} \) and \( i = I \).

\[ est_I = 0 \quad est_{\{G, H\}} = 2 \]

\[ lct_I = 10 \]

\[ lct_{\{G, H\}} = 4 \]

0 3 6 9
Extended edge-finding

- Recall the edge-finding rule:

\[ \text{energy}_\omega + \text{energy}_i > C \cdot (\text{lct}_\omega - \text{est}_\omega \cup \{i\}) \Rightarrow \omega < i \]

- In this example
  - no feasible schedule where I starts before 3,
  - but the edge-finding condition is not satisfied for \( \omega = \{G, H\} \) and \( i = I \).

\[ \text{est}_{\{G,H,I\}} = 0 \]

\[ lct_I = 10 \]

\[ lct_{\{G,H\}} = 4 \]
Extended edge-finding

- Recall the edge-finding rule:

\[\text{energy}_\omega + \text{energy}_i > C \cdot (lct_\omega - \text{est}_\omega \cup \{i\}) \implies \omega < i\]

- In this example
  - no feasible schedule where I starts before 3,
  - but the edge-finding condition is not satisfied for \(\omega = \{G, H\}\) and \(i = I\).

\[\text{est}_{\{G, H, I\}} = 0\]

\[lct_I = 10\]
Extended edge-finding

- Recall the edge-finding rule:

\[ energy_\omega + energy_i > C \cdot (lct_\omega - est_\omega \cup \{i\}) \implies \omega < i \]

- In this example
  - no feasible schedule where I starts before 3,
  - but the edge-finding condition is not satisfied for \( \omega = \{G, H\} \) and \( i = I \).

\[ est_{\{G,H,I\}} = 0 \]

\[ lct_I = 10 \]

\[ lct_{\{G,H\}} = 4 \]
Extended edge-finding

- Recall the edge-finding rule:

\[ \text{energy}_\omega + \text{energy}_i > C \cdot (\text{lct}_\omega - \text{est}_{\omega \cup \{i\}}) \implies \omega < i \]

- In this example
  - no feasible schedule where \( I \) starts before 3,
  - but the edge-finding condition is not satisfied for \( \omega = \{G, H\} \) and \( i = I \).

\[ \text{est}_I = 0 \quad \text{est}_{\{G,H\}} = 2 \]

\[ \text{lct}_I = 10 \]

Edge-finding is too loose a relaxation for tasks with an \( \text{est} \) in this range.
Extended edge-finding

The extended edge-finding rule (for $est_i < est_\omega$):

$$energy_\omega + energy_i - use_i(est_\omega - est_i) > C \cdot (lct_\omega - est_\omega \cup \{i\}) \Rightarrow \omega < i$$

Intuition

Split task $i$ into two parts: the largest part that could execute before $est_\omega$, and the remainder. Check the edge-finding condition on $\omega$ and the remainder.
Extended edge-finding

The extended edge-finding rule (for $est_i < est_\omega$):

$$energy_\omega + energy_i - \text{use}_i(est_\omega - est_i) > C \cdot (lct_\omega - est_\omega \cup \{i\}) \Rightarrow \omega < i$$

Intuition

Split task $i$ into two parts: the largest part that could execute before $est_\omega$, and the remainder. Check the edge-finding condition on $\omega$ and the remainder.
Extended edge-finding

The extended edge-finding rule (for \( est_i < est_\omega \)):

\[
energy_\omega + energy_i - \text{use}_i(\text{est}_\omega - est_i) > C \cdot (\text{lct}_\omega - est_{\omega \cup \{i\}}) \Rightarrow \omega \leq i
\]

\[
est_I = 0 \quad \text{est}_{\{G,H\}} = 2
\]

\[
\text{lct}_I = 10
\]

\[
\text{lct}_{\{G,H\}} = 4
\]

Intuition

Split task \( i \) into two parts: the largest part that could execute before \( est_\omega \), and the remainder. Check the edge-finding condition on \( \omega \) and the remainder.
■ In edge-finding, looking for a task that must come first or last in a set of tasks

■ Not-first/not-last looks for a task that cannot be the first or last in a set
  • There is at least one activity in $\theta$ that must be scheduled before (after) $i$.
  • Requires reasoning on the minimum earliest completion time of any task in $\theta$.
  • A “lazy” $O(n^2 \log n)$ algorithm reported at CP 2010.
Energetic Reasoning

- Substantially stronger filtering than edge-finding or not-first/not-last.
- Considers required energy consumption over various time periods, tries to deduce when a task must be shifted earlier or later.
- Best algorithm is $\mathcal{O}(n^3)$!
  - Is it worth it? Maybe, maybe not.

- More recently, [VILÍM, 2011] presented a “Timetable Edge-Finding” algorithm
  - Required parts plus edge-finding, in $\mathcal{O}(n^2)$ time.
  - Incorporates some deductions made by energetic reasoning.
Further Reading I

Baker K.R., Trietsch D.
*Principles of Sequencing and Scheduling.*
Comprehensive overview of scheduling, from OR perspective.

Baptiste P., Le Pape C., Nuijten W.P.M.
*Constraint-based scheduling: applying constraint programming to scheduling problems.*
Algorithms out of date, but remains the most complete reference.

Vilím P.
Edge finding filtering algorithm for discrete cumulative resources in $O(kn \log n)$.
A challenging paper, but a wonderful algorithm (used in the current release of Gecode). Winner, “Best Paper, CP 2009.”.

Scott J.
*Filtering Algorithms for Discrete Cumulative Resources.*
Details omitted by [Vilím, 2009], plus explanations and corrections.
Further Reading II

Mercier L., Van Hentenryck P.
**Edge finding for cumulative scheduling.**
More recent algorithms are faster, but this remains the best theoretical discussion.

Kameugne R., Fotso L.P., Scott J., Ngo-Kateu Y.
**A quadratic edge-finding filtering algorithm for cumulative resource constraints.**
Simpler to implement than \( \Theta \)-tree filtering, and generally faster in practice. Planned for inclusion in the next major release of Gecode.

Vilím P.
**Timetable edge finding filtering algorithm for discrete cumulative resources.**
Interesting hybrid of edge-finding, time tabling, and parts of energetic reasoning, all in \( \mathcal{O}(n^2) \) time.