Edge finding for disjunctive and cumulative constraints

Petr Vilím

CPAIOR 2012
Outline

- Introduction, Job Shop Problem
- Unary resource
  - Overload
  - Computation of earliest completion times
  - Sketch of propagation algorithms
- Cumulative resource
  - Timetable Edge Finding
- Optional Activities
Introduction
Job Shop Problem
Activities

- Our task is to find a schedule for set of activities (operations, tasks).
- Each activity is characterized by:
  - Processing time (duration) \( p_i \)
  - Earliest start time \( \text{est}_i \)
    - For jobshop, initially \( \text{est}_i = 0 \)
  - Latest completion time (deadline) \( \text{lct}_i \)
    - For jobshop, initially \( \text{lct}_i = \infty \)
- Activities cannot be preempted.
Machines (resources)

- Each activity requires during execution one particular machine from a set of machines.
  - Resource (machine, worker, workplace, ...)
- Machine cannot execute more than one activity a time
  - Resource capacity $= 1 \rightarrow \textit{unary (disjunctive) resource}$
Precedences

- Activities are split into jobs.
- Job is a sequence of activities that must be executed in a particular order:
Objective

- Find shortest possible schedule.
- Optimal solution of problem instance la19:
CP model: activities

- Activities are decision variables (their start and end times).
  - Each activity \( i \) is represented by values \( \text{est}_i \), \( \text{lct}_i \), \( p_i \).
  - \( \text{est}_i + p_i \leq \text{lct}_i \)
  - When \( \text{est}_i + p_i = \text{lct}_i \) then activity is bound.
  - When \( \text{est}_i + p_i > \text{lct}_i \) then there's no solution (fail).
  - Interval \( [\text{est}_i, \text{lct}_i] \) is called \textit{time window} of activity \( i \).
  - Usually, domain is represented only by time window (holes in domain are not maintained).

Propagation aims to increase \( \text{est}_i \) and decrease \( \text{lct}_i \)
CP model: precedences

- For every two consecutive activities a, b in a job:
  - \( \text{est}_a + p_a \leq \text{est}_b \)
  - \( \text{lct}_a \leq \text{lct}_b - p_b \)
CP model: resources

- Resources are represented by resource constraints.
- Resource constraint consists of:
  - A set of activities.
  - Resource capacity.
- Resource constraint propagates on activities: increases values est$_i$ and decrease lct$_i$. 

![Diagram showing resource constraints and their propagation on activities](image-url)
CP model: search

- Maintain a domain (time window) for each activity.
- After each search decision, *filtering algorithms* are used to reduce time windows (*propagation*).
- Typically, filtering algorithms are repeated until *fixpoint* is found (no algorithm can propagate any more).
- Propagation can lead to a fail (no solution exists).
Unary Resource Constraint
Propagation algorithms

- **Overload Checking** (fail detection)
- **Edge-Finding**
  - $O(n \log n)$: [Carlier & Pinson 1994], [Vilím]
  - $O(n^2)$: [Martin & Shmoys 96], [Wolf 2003], [Nuijten].
- **Not-Fist/Not-Last**
  - $O(n^2)$: [Baptiste & Le Pape 1996], [Torres & Lopez 1999], [Wolf 2003]
  - $O(n \log n)$: [Vilím]
- **Detectable Precedences**
  - $O(n \log n)$: [Vilím]
- …

Each algorithm removes different type of inconsistent values, therefore they can be used together to achieve better pruning.
Fixpoint

- Overload Checking
  - inconsistent
  - consistent
- Detectable Precedences
- Not-First/Not-Last
- Edge Finding
- Precedence Graph
  - no change
- Fixpoint

14
Example: no solution (overload)

Traditional explanation:

- Union of time windows of \{B, C, D\} is [25, 43], its length is 18.
- Total duration of \{B, C, D\} is $6 + 4 + 10 = 20$.
- $18 < 20 \rightarrow$ no solution.

Leads to $O(n^2)$ algorithm.
Example: no solution (overload)

Alternative explanation (leads to $O(n \log n)$ algorithm):

- Lets relax the problem by ignoring deadlines (all $lct_i = \infty$).
Example: no solution (overload)

Alternative explanation (leads to $O(n \log n)$ algorithm):

- Lets relax the problem by ignoring deadlines (all $lct_i = \infty$).
- With this relaxation, what is *earliest completion time of set* $\{A, B, C, D\}$?
Example: no solution (overload)

Alternative explanation (leads to $O(n \log n)$ algorithm):

- Lets relax the problem by ignoring all deadlines (assuming all $lct_i = \infty$).
- With this relaxation, what is \textit{earliest completion time of set \{A, B, C, D\}}?
  - $est_B + p_B + p_C + p_D = 25 + 6 + 4 + 10 = 45$
Example: no solution (overload)

Alternative explanation (leads to $O(n \log n)$ algorithm):

- Lets relax the problem by ignoring deadlines (all $lct_i = \infty$).
- With this relaxation, what is earliest completion time of set $\{A, B, C, D\}$?
  - $\text{est}_B + p_B + p_C + p_D = 25 + 6 + 4 + 10 = 45$
- But what is the deadline for $\{A, B, C, D\}$?
  - $lct_{\{A,B,C,D\}} = \max\{lct_A, lct_B, lct_C, lct_D\} = \max\{28, 36, 42, 43\} = 43$
- $43 > 45 \rightarrow$ no solution.
What is the difference?

- Classical explanation does not detect a problem for set \{A, B, C, D\}.
  - It have to check also subset \{B, C, D\} to recognize infeasibility.
  - There is \(O(n^2)\) sets to check this way
    - One set for every combination of \(\text{est}_X\) and \(\text{lct}_Y\).
- Alternative explanation correctly recognize problem for \{A, B, C, D\}.
  - There is \(O(n)\) sets to check this way
    - One for every \(\text{lct}_Y\).

However, how to compute earliest completion times effectively?
Let $\Omega$ is a set of activities.

- Earliest start time of $\Omega$ is $\text{est}_\Omega = \min\{\text{est}_i, i \in \Omega\}$
- Latest completion time of $\Omega$ is $\text{lct}_\Omega = \max\{\text{lct}_i, i \in \Omega\}$
- Total duration of $\Omega$ is $p_\Omega = \max\{\text{lct}_i, i \in \Omega\}$

For $\Omega = \{B, C, D\}$:
- $\text{est}_\Omega = 35$
- $\text{lct}_\Omega = 43$
- $p_\Omega = 20$
Let $\Omega$ be a set of activities.

- Earliest start time of $\Omega$ is $est_\Omega = \min\{est_i, i \in \Omega\}$
- Latest completion time of $\Omega$ is $lct_\Omega = \max\{lct_i, i \in \Omega\}$
- Total duration of $\Omega$ is $p_\Omega = \max\{lct_i, i \in \Omega\}$
- Earliest completion time of (another set of activities) $\Theta$ is:

$$ECT_\Theta = \max\{est_\Omega + p_\Omega, \Omega \subseteq \Theta\}$$

For $\Theta = \{A, B, C, D\}$ the best $\Omega$ is $\{B, C, D\}$ and $ECT_\Theta = 25 + 20 = 45$. 
Overload rule

\[ ECT_\Theta > lct_\Theta \implies \text{fail} \]
Computation of earliest completion time

- The goal is to quickly recompute $\text{ECT}_\Theta$ after a change of $\Theta$ such as:
  - addition of an activity into $\Theta$
  - removal of an activity from $\Theta$
- The idea: represent $\Theta$ by a binary tree.
**Θ-Tree**

- Activities are represented by leaves
  - sorted by $est_i$
- Each node holds:
  - $\Sigma P$: total duration of activities in the subtree
  - ECT: earliest completion time of the subtree
- ECT of $\Theta$ is in the root node.
Θ-Tree: recursive computation

\[ \Sigma P_v = \Sigma P_{\text{left}(v)} + \Sigma P_{\text{right}(v)} \]

\[ \begin{align*}
\Sigma P &= 25 \\
\text{ECT} &= 45 \\
\Sigma P &= 11 \\
\text{ECT} &= 31 \\
\Sigma P &= 14 \\
\text{ECT} &= 44 \\
est_A &= 15 \\
p_A &= 5 \\
\Sigma P &= 5 \\
\text{ECT} &= 20 \\
est_B &= 25 \\
p_B &= 6 \\
\Sigma P &= 6 \\
\text{ECT} &= 31 \\
est_C &= 30 \\
p_C &= 4 \\
\Sigma P &= 4 \\
\text{ECT} &= 34 \\
est_D &= 32 \\
p_D &= 10 \\
\Sigma P &= 10 \\
\text{ECT} &= 42
\end{align*} \]
\( \Theta\text{-Tree: recursive computation} \)

\[
ECT_v = \max \left\{ ECT_{\text{right}(v)}, \ ECT_{\text{left}(v)} + \Sigma P_{\text{right}(v)} \right\}
\]

\[
ECT_\Theta = \max \left\{ \text{est}_\Omega + p_\Omega, \ \Omega \subseteq \Theta \right\}
\]
Θ-Tree: time complexities

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Θ := ∅</td>
<td>Θ(1) or Θ(n log n)</td>
</tr>
<tr>
<td>Θ := Θ ∪ {i}</td>
<td>Θ(log n)</td>
</tr>
<tr>
<td>Θ := Θ \ {i}</td>
<td>Θ(log n)</td>
</tr>
<tr>
<td>ECT_Θ</td>
<td>Θ(1)</td>
</tr>
</tbody>
</table>

ΣP = 11
ECT = 31

ΣP = 14
ECT = 44

ΣP = 5
ECT = 20

est_A = 15
p_A = 5
ΣP = 5
ECT = 31

est_B = 25
p_B = 6
ΣP = 6
ECT = 31

est_C = 30
p_C = 4
ΣP = 4
ECT = 44

est_D = 32
p_D = 10
ΣP = 10
ECT = 42

A

B

C

D

15 20 25 30 35 40 45

© 2011 IBM Corporation
Overload checking algorithm

\begin{algorithm}
\begin{algorithmic}
\STATE $\Theta := \emptyset$;
\FOR{$j \in T$ in non-decreasing order of $\text{lct}_j$}
\STATE $\Theta := \Theta \cup \{j\}$;
\IF{$\text{ECT}_\Theta > \text{lct}_j$}
\STATE \textbf{fail}; \quad \{ No solution exists \}
\ENDIF
\ENDFOR
\end{algorithmic}
\end{algorithm}

Time complexity is $O(n \log n)$.
Example of implementation of Θ-Tree

• Tree is stored in an array (similar to array representation of a heap).
• Tree doesn't change its shape. Instead of node addition/removal nodes are turned on/off.
• Node turned off:
  • $\sum P = 0$
  • ECT = $-\infty$

Activity is not in $\Theta$

Root node | Internal nodes | All activities sorted by $est_i$ | Unused
Edge Finding

- Edge finding improve bounds by removing values that would lead to overflow.
- Scheduling activity C before 18 would lead to overflow.
  - \( \text{est}_C := 18 \)
Edge Finding

- Remember the overflow rule:

  \[ \text{ECT}_\Theta > \text{lct}_\Theta \implies \text{fail} \]

- Edge finding rule is:

  \[ \text{ECT}_{\Theta \cup \{i\}} > \text{lct}_\Theta \implies \Theta \ll i \implies \text{est}_i := \max \{\text{est}_i, \text{ECT}_\Theta\} \]

- Setting \( \text{lct}_\Theta \) as deadline for activity \( i \) would cause overflow.

  - Therefore \( i \) can start only after all activities from \( \Theta \) finish.
Edge Finding: idea of the algorithm

- Consider some deadline $t$.
- $\Theta =$ all activities that must finish before $t$.
- $\Lambda =$ all activities that can start before $t$ but can finish after $t$.
- If we can add one activity from $\Lambda$ into $\Theta$, how big earliest completion time we can make?
- Is it bigger than $t$?
- If yes, activity we used from $\Lambda$ can be updated and removed from $\Lambda$.

- for example $t = \text{lct}_D = 18$
- $\Theta = \{D, E, F\}$
- $\Lambda = \{C\}$
- $\text{ECT}_{\{C,D,E,F\}} = 19$
- Yes: $19 > 18$
- $\text{est}_C := 18$
The concept of Θ-tree is extended to compute:

$$\text{ECT}(\Theta, \Lambda) = \max (\{\text{ECT}_\Theta\} \cup \{\text{ECT}_{\Theta \cup \{i\}}, \ i \in \Lambda\})$$

\[\begin{array}{ll}
\Sigma P &= 21 \\
\text{ect} &= 44 \\
\overline{\Sigma P} &= 26 \\
\overline{\text{ect}} &= 49 \\
\end{array}\]

\[\begin{array}{ll}
\Sigma P &= 11 \\
\text{ect} &= 34 \\
\overline{\Sigma P} &= 11 \\
\overline{\text{ect}} &= 34 \\
\end{array}\]

\[\begin{array}{ll}
est_a &= 0 \\
p_a &= 5 \\
\Sigma P_a &= 5 \\
ect_a &= 5 \\
\overline{\Sigma P}_a &= 5 \\
\overline{\text{ect}}_a &= 5 \\
\end{array}\]

\[\begin{array}{ll}
est_b &= 25 \\
p_b &= 9 \\
\Sigma P_b &= 9 \\
ect_b &= 34 \\
\overline{\Sigma P}_b &= 9 \\
\overline{\text{ect}}_b &= 34 \\
\end{array}\]

\[\begin{array}{ll}
est_c &= 30 \\
p_c &= 5 \\
\Sigma P_c &= 0 \\
ect_c &= -\infty \\
\overline{\Sigma P}_c &= 5 \\
\overline{\text{ect}}_c &= 35 \\
\end{array}\]

\[\begin{array}{ll}
est_d &= 32 \\
p_d &= 10 \\
\Sigma P_d &= 10 \\
ect_d &= 42 \\
\overline{\Sigma P}_d &= 10 \\
\overline{\text{ect}}_d &= 42 \\
\end{array}\]
**Θ-Λ-Tree: time complexities**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>((Θ, \Lambda) := (\emptyset, \emptyset))</td>
<td>(O(1))</td>
</tr>
<tr>
<td>((Θ, \Lambda) := (T, \emptyset))</td>
<td>(O(n \log n))</td>
</tr>
<tr>
<td>((Θ, \Lambda) := (Θ \setminus {i}, \Lambda \cup {i}))</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>(Θ := Θ \cup {i})</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>(Λ := Λ \cup {i})</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>(Λ := Λ \setminus {i})</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>(\text{ECT}(Θ, Λ))</td>
<td>(O(1))</td>
</tr>
<tr>
<td>responsible for (\text{ECT}(Θ, Λ))</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>(\text{ECT}_Θ)</td>
<td>(O(1))</td>
</tr>
</tbody>
</table>
Edge Finding algorithm

1 (Θ, Λ) := (T, ∅);
2 Q := queue of all activities j ∈ T in non-increasing order of lct_j;
3 j := Q.first;
4 while Q.size > 1 do begin
5   if ECT_Θ > lct_j then
6     fail;  {Resource is overloaded}
7     (Θ, Λ) := (Θ \ {j}, Λ ∪ {j});
8     Q.dequeue;
9     j := Q.first;
10    while ECT(Θ, Λ) > lct_j do begin
11      i := gray activity responsible for ECT(Θ, Λ);
12      est_i := max{est_i, ECT_Θ};
13      Λ := Λ \ {i};
14    end;
15  end;

Time complexity is O(n log n).
Symmetry

- Just presented algorithm updates only est\(_i\), not lct\(_i\).
- Algorithm to update lct\(_i\) is symmetrical.
- There are two ways to implement it:
  - Write the algorithm twice ("forward" and "backward" versions).
  - Write the algorithm only once but feed it with symmetrical data.
Not-First / Not-Last

- Let $\Theta = \{A, B\}$.
- $ECT_\Theta = ect_A + p_A + p_B = 0 + 11 + 10 = 21$
- If $\Theta$ is scheduled before C then $\Theta$ would have to end before $lct_C - p_C = 20 - 2 = 18$
  - This is not possible because $21 > 18$
- At least one activity from $\Theta$ must be after C.
- $lct_C \leq \max(lct_A - p_A, lct_B - p_B) = 17$

Propagation rule:

$$ECT_\Theta > lct_i - p_i \implies i \text{ can’t be last} \implies lct_i := \max\{lct_j - p_j, \; j \in \Theta\}$$
Not-Last algorithm

1 for $i \in T$ do
2 \hspace{1em} lct' := lct_i;
3 \hspace{1em} \Theta := \emptyset;
4 \hspace{1em} Q := \text{queue of all activities } j \in T \text{ in non-decreasing order of } lct_j - p_j;
5 for $i \in T$ in non-decreasing order of lct_i do begin
6 \hspace{2em} while lct_i > lct_{Q.first} - p_{Q.first} do begin
7 \hspace{3em} j := Q.first;
8 \hspace{3em} \Theta := \Theta \cup \{j\};
9 \hspace{3em} Q.dequeue;
10 end;
11 if $\text{ECT}_{\Theta \setminus \{i\}} > lct_i - p_i$ then
12 \hspace{1em} lct' := \min\{lct_j - p_j, lct'\};
13 end;
14 for $i \in T$ do
15 \hspace{1em} lct_i := lct'_i;

Time complexity is $O(n \log n)$. 
Detectable precedences

- C doesn't fit before B. Therefore B is before C: $B \ll C$
- Similarly, C doesn't fit before A. Therefore A is before C: $A \ll C$
- $\{A, B\} \ll C$ therefore C cannot start before $ECT_{\{A,B\}} = 10$. 
Detectable precedences

- Detectable precedence:
  \[ \text{est}_i + p_i > lct_j - p_j \quad \Rightarrow \quad j \ll i \]

The algorithm:
- Take an activity i
- Let \( \Theta \) are detectable predecessors of i: \( \Theta = \{j, j \ll i\} \).
- Then i cannot start before \( \text{ECT}_\Theta \).
Detectable Precedences algorithm

\begin{algorithmic}
\State $\Theta := \emptyset$;
\State $Q := \text{queue of all activities } j \in T \text{ in non-decreasing order of } \text{lct}_j - p_j$;
\For{$i \in T$ in non-decreasing order of $\text{est}_i + p_i$} do begin
\While{$\text{est}_i + p_i > \text{lct}_{Q,\text{first}} - p_{Q,\text{first}}$} do begin
\State $\Theta := \Theta \cup \{Q,\text{first}\}$;
\State $Q$.dequeue;
\End;\end{algorithmic}

\State $\text{est}'_i := \max\{\text{est}_i, \text{ECT}_{\Theta \setminus \{i\}}\}$;
\End;
\For{$i \in T$} do
\State $\text{est}_i := \text{est}'_i$;
\End;

Time complexity is $O(n \log n)$.
Cumulative Resource: Timetable Edge Finding
Cumulative Resource
Filtering Algorithms for Cumulative Resource

Classical Filtering Algorithms:
- Timetable propagation
- Edge Finding:
  - $O(kn^2)$
  - $O(kn \log n)$
- Extended Edge Finding
  - $O(kn^2)$
- Not-First / Not-Last
  - $O(n^2 \log n)$, lazy
- Energetic Reasoning
  - $O(n^3)$

These algorithms are independent and could/should be used together.

Timetable Edge Finding:
- Inspired by all the algorithms on the left.
- Reuses/shares data structure with timetable propagation.
- Stronger propagation than both Edge Finding and Extended Edge Finding.
- Limited Not-First / Not-Last and Energetic Reasoning.
  - $O(n^2)$
  - Lazy propagation: may need more iterations to reach fixpoint.
Timetable Propagation

- If for activity $i$ holds $lct_i - p_i < est_i + p_i$ then the activity necessarily use the resource during interval $[lct_i - p_i, est_i + p_i]$.

- In this case we split the interval into fixed and free parts:

- Fixed parts are aggregated into timetable (graph of minimum resource usage):

- The timetable is used to improve bounds of activities.
Overload Checking

- Similar to disjunctive case. $O(n^2)$ and $O(n \log n)$ versions.
- It is the cornerstone of all Edge Finding algorithms.
- The idea is to choose an interval $[X, Y]$ and compare:

\[
C(Y - X) < \sum_{\text{est}_i \geq X, \ lct_i \leq Y} c_i p_i \quad \Rightarrow \quad \text{fail}
\]
Standard and Extended Edge Finding Algorithms

Informally speaking, these algorithms update time windows in such a way that scheduling any activity $i$ on its earliest starting time $est_i$ does not lead to immediate overload.

- In this example, $est_D$ can be updated from 0 to 4.
- Otherwise, either interval $[0, 5]$ or $[2, 5]$ would be overloaded.
Energetic Reasoning Algorithm

- Energy computation in Edge Finding takes into account only activities which are completely inside the interval \([X, Y]\).
- Therefore it misses cases when only a part of the activity must be executed inside \([X, Y]\). For example, activity \(i\) in the following picture consumes at least 3 energy units during \([1, 5]\):

![Diagram showing energy computation](image)

- There is Energetic Reasoning algorithm, which takes this energy into account, but it is \(O(n^3)\).
- But there are some simple cases where we can improve energy computation without increasing time complexity.
- In particular, the idea is to take into account timetable.
Timetable Edge Finding

The idea is to split energy computation into two parts:

**energy from fixed parts**

- Fixed part, $p_i^{TT} = 2$
- $est_i = 0$
- $lct_i = 6$

**energy from free parts**

- Free part, $p_i^{EF} = 2$

This energy can be easily computed from timetable:

Computed by standard Edge Finding way, but only from free parts:
Example of energy computation

What is the minimal energy contribution of activity $i$ to interval $[1, 5]$?

- Energetic reasoning: 3
  - Exact computation, but slow.
- Edge Finding: 0
  - Activity $i$ is not completely inside $[1, 5]$ therefore it is ignored.
- Timetable Edge Finding: 2
  - Fast, but not exact.
Example of energy computation

Timetable Edge Finding splits activity $i$ into two fixed part (duration 2) and free part (also duration 2):

For interval $[1,5]$, TTEF takes fixed part into account, but ignores free part (because it is not completely inside $[1,5]$). Total contribution counted is 2 energy units.

Note that for fixed activities, TTEF computes the same value as Energetic Reasoning.
Timetable Edge Finding algorithm

\begin{verbatim}
1. \quad i \in T^{EP}
2. \quad \text{est}_1 := \text{est}_i;
3. \quad b \in T^{EP}
4.   // Cases “Inside” and “Right”
5.   \text{eEF} := 0;
6.   \quad \iota := -1;
7.   \quad a \in T^{EP} \text{ such that est}_a < \text{lct}_b, \text{ in non-increasing order by est}_a
8.   \quad \text{lct}_a \leq \text{lct}_b
9.   \quad \text{eEF} := \text{eEF} + e^{EP}_b;
10.  \quad \iota := \iota - 1
11.  \quad \text{reserve} := C (\text{lct}_b - \text{est}_a) - \text{eEF} - (\text{ttAfterEst}[a] - \text{ttAfterLct}[b]);
12.  \quad \iota \neq -1
13.  \quad \text{reserve} < \text{min}(e^{EP}_b, c_i (\text{lct}_b - \text{est}_a))
14.  \quad \text{est}_i' := \max (\text{est}_i, \text{lct}_b - \text{mandatoryIn} (\text{est}_a, \text{lct}_b, \iota) - \lceil \text{reserve} / c_i \rceil);
15.  \\
16.   // Case “Through”
17.   \quad \iota := -1;
18.   \quad a \in T^{EP} \text{ in non-decreasing order by est}_a,
19.   \quad \text{break ties by non-increasing est}_a + p^{EP}_a
20.   \quad \text{lct}_a \leq \text{lct}_b
21.   \quad \text{reserve} := C (\text{lct}_b - \text{est}_a) - \text{eEF} - (\text{ttAfterEst}[a] - \text{ttAfterLct}[b]);
22.   \quad \iota \neq -1
23.   \quad \text{reserve} < c_i (\text{lct}_b - \text{est}_a)
24.   \quad \text{est}_i' := \max (\text{est}_i', \text{lct}_b - \text{mandatoryIn} (\text{est}_a, \text{lct}_b, \iota) - \lceil \text{reserve} / c_i \rceil);
25.   \quad \text{eEF} := \text{eEF} - e^{EP}_b;
26.   \\
27.   \quad \text{est}_a + p^{EP}_a \geq \text{lct}_b \quad (\iota = -1 \quad c_i > c_i)
28.   \\
29.   \quad \iota := \iota - 1;
30.  \\
31.   // Case “Left”
32.   \quad a \in T^{EP}
33.   \quad \text{eEF} := 0;
34.   \quad \iota := -1;
35.   \quad \text{Q} := \text{queue of activities } i \in T^{EP} \text{ sorted by non-decreasing est}_i + p^{EP}_i;
36.   \quad b \in T^{EP} \text{ in non-decreasing order by est}_i
37.   \quad \text{est}_a \leq \text{est}_b
38.   \quad \text{eEF} := \text{eEF} + e^{EP}_b;
39.   \quad \text{est}_{Q, \text{top}} + p^{EP}_{Q, \text{top}} < \text{lct}_b
40.   \quad i := \text{Q.\text{dequeue}};
41.   \quad \text{est}_i < \text{est}_a \quad \text{est}_a < \text{est}_i + p^{EP}_i
42.   \quad (\iota = -1, c_i (\text{est}_i + p^{EP}_i - \text{est}_a) > c_i (\text{est}_a + p^{EP}_i - \text{est}_a))
43.   \quad \iota := i;
44.   \quad \text{reserve} := C (\text{lct}_b - \text{est}_a) - \text{eEF} - (\text{ttAfterEst}[a] - \text{ttAfterLct}[b]);
45.   \quad \iota \neq -1
46.   \quad \text{reserve} < c_i (\text{est}_i + p^{EP}_i - \text{est}_a)
47.   \quad \text{est}_i' := \max (\text{est}_i', \text{lct}_b - \text{mandatoryIn} (\text{est}_a, \text{lct}_b, \iota) - \lceil \text{reserve} / c_i \rceil);
48.   \\
49.   \quad i \in T^{EP}
50.   \quad \text{est}_i := \text{est}_i';
\end{verbatim}

Time complexity is O(n^2).
Optional activities
Alternatives

Let activity C represent my travel from Prague to Nantes. I can travel by:

- train
- plane
- or car.

This decision affects:

- duration
- departure time
- cost
- resource usage

Traditionally way is to use meta-constraints to describe the problem:

- Either (train) duration = 8h and departure in {9:00, 13:40, .. } and cost = 170€
- Or (plane) duration = 3h and departure in { 9:20, 12:30, .. } and cost = 250€
- Or (car) duration = 11h and cost = 200€
Alternatives: new approach

The idea is to represent not only C as activity, but also its alternatives (modes).

- C is **present** activity.
- Its alternatives are **optional** activities.
- Optional activities doesn't have to appear in the schedule.
- If they don't appear then their start is undefined.

The solver must make a decision which one of the activities C:Train, C:Plane and C:Car will be **present** in the solution. The remaining two activities will be **absent**.
Optional Interval Variable

Extension of classical CSP with a new type of decision variable: 

Optional Interval Variable a:

\[
\text{Domain}(a) \subseteq \{ \bot \} \cup \{ [s,e) \mid s, e \in \mathbb{Z}, s \leq e \}
\]

The interval can be either:

- present, then it starts at time \( s \) and ends at time \( e \),
- or absent (\( \bot \)), and then it doesn't have any start or end.

Most constraints ignore absent intervals. For example:

- Precedence \( \text{endsBeforeStart}(a, b) \) is automatically satisfied if \( a \) or \( b \) are absent.
- Resource requirements of absent intervals are ignored.
Semantics of the alternative constraint

\texttt{alternative}(C, \{C:Train, C:Plane, C:Car\})

- If C is present then:
  - Exactly one of C:Train, C:Plane, C:Car is present.
  - C and the chosen alternative start together and end together.
- If C is absent then C:Train, C:Plane and C:Car are also absent.
Semantics of alternative constraint

\[ \text{alternative}(C, \{C: \text{Train}, C: \text{Plane}, C: \text{Car}\}) \]

- If \( C \) is present then:
  - Exactly one of \( C: \text{Train}, C: \text{Plane}, C: \text{Car} \) is present.
  - \( C \) and the chosen alternative starts together and end together.
- If \( C \) is absent then \( C: \text{Train}, C: \text{Plane} \) and \( C: \text{Car} \) are also absent.

This allows to easily constraints both on master interval \( C \) and its modes like \( C: \text{Car} \).

After arrival to Nantes, I'll check in to the hotel:
- \( \text{endBeforeStart}(C, \text{HotelCheckin}) \)

If I use plane then I have to buy tickets at least 10 days ahead:
- \( \text{presenceOf(BuyPlaneTickets)} = \text{presenceOf(C:Plane)} \)
- \( \text{endsBeforeStart(BuyPlaneTickets, C, 10)} \)
Propagation of alternative constraint

\text{alternative}(C, \{\text{C:Train, C:Plane, C:Car}\})

- For optional activities, we maintain their time window \([\text{est}_i, \text{lct}_i]\) for the case they will become present.
- For example:
  - \(\text{est}_{C:Train} = 9:00\) (first train)
  - \(\text{est}_{C:Plane} = 9:20\) (first plane)
  - \(\text{est}_{C:Car} = 8:00\) (I refuse to get up early)
- Earliest starting time of master activity \(C\) is the minimum of available alternatives:
  - \(\text{est}_C = 8:00\)
Propagation of alternative constraint

alternative(C, {C:Train, C:Plane, C:Car})

- For optional activities, we maintain their time window [est\_i, lct\_i] for the case they will become present.

- For example:
  - est\_C:Train = 9:00 (first train)
  - est\_C:Plane = 9:20 (first plane)
  - est\_C:Car = 8:00 (I refuse to get up early)
- Earliest starting time of master activity C is the minimum of available alternatives:
  - est\_C = 8:00

- However, car is a disjunctive resource that cannot be used by more than one driver at a time.

- My wife occupies the resource until 21:00 (present activity).
  - Resource constraint propagates: est\_C:Car = 21:00.

- But that's too late: lct\_C:Car ≤ lct\_C = 19:00. Therefore C:Car becomes absent.
  - If C:Car wouldn't be optional then it would mean a fail.
- As a result, alternative constraint propagates est\_C = 9:00.
How to handle optional activities in resource constraints?

The general rules are:

- Present activities influence all other activities on the resource including optional ones.
  - My wife blocked the car.
- Absent activities are ignored.
  - Once I decided not to use the car, car is not affected by my travel to Nantes at all.
- Optional activities does not affect any other activity on the resource.
  - While I was only speculating about using the car, I couldn't postpone ride of my wife.
Disjunctive Edge Finding with optional activities

- Remember Edge Finding propagation rule:

\[ \text{ECT}_{\Theta \cup \{i\}} > \text{lct}_\Theta \Rightarrow \Theta \ll i \Rightarrow \text{est}_i := \max \{\text{est}_i, \ ECT_\Theta\} \]

- Set \( \Theta \) cannot contain any optional (or absent) interval.
  - Otherwise optional activity would affect other activity \(- i \-\) on the resource.
  \( \rightarrow \) Never add optional activity into \( \Theta \).
- Note that \( i \) could be optional activity.
Disjunctive Edge Finding with optional activities

- Remember Edge Finding propagation rule:

\[ ECT_{\Theta \cup \{i\}} > lct_\Theta \Rightarrow \Theta \ll i \Rightarrow est_i := \max \{est_i, ECT_\Theta\} \]

Another approach:

Use classical EF algorithm (unaware of optional activities) but pretend (just for the algorithm) that all optional activities have \( lct_i = \infty \).

- If optional activity \( l \) is in \( \Theta \) then \( lct_\Theta = \infty \) and therefore the inequality doesn't hold.

→ It is not necessary to write new version of EF algorithm.

- It works for cumulative Edge Finding too.
Implementation of EF with optional activities

Optional decision variables

Interface for symmetry and optionality

Edge Finding algorithm

It works for Edge Finding, but not for (for example) Not-First / Not-Last.
Questions?