

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: *cumulative*

3. Propagation of the cumulative constraint

Time Table

Overload Checking

Edge-Finding

Other *cumulative* propagation algorithms

4. Conclusion



Outline

Outline

Scheduling

Examples

General

RCSP

Propagation

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:

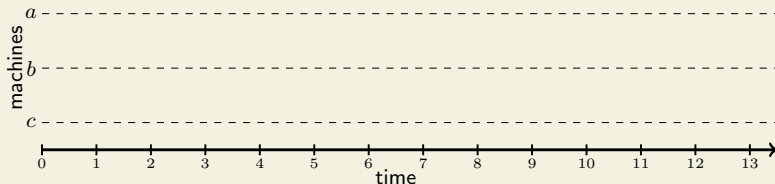
- Each team plays exactly once against each other team.
- Each team plays exactly once per week.
- Each team plays at most twice per period.

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

	Wk 1	Wk 2	Wk 3	Wk 4	Wk 5	Wk 6	Wk 7
P 1	1 vs. 2	1 vs. 3	2 vs. 6	3 vs. 5	4 vs. 7	4 vs. 8	5 vs. 8
P 2	3 vs. 4	2 vs. 8	1 vs. 7	6 vs. 7	6 vs. 8	2 vs. 5	1 vs. 4
P 3	5 vs. 6	4 vs. 6	3 vs. 8	1 vs. 8	1 vs. 5	3 vs. 7	2 vs. 7
P 4	7 vs. 8	5 vs. 7	4 vs. 5	2 vs. 4	2 vs. 3	1 vs. 6	3 vs. 6

A different scheduling model

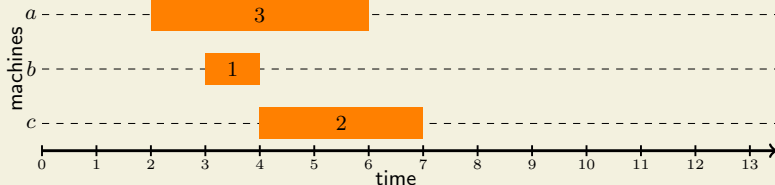
Example (The Job-Shop Scheduling Problem)



- m **machines**, each processing one operation at a time
- n **jobs**, $job_i = \langle op_1^i, op_2^i, \dots, op_m^i \rangle$
 - a job is a sequence of **operations**, op_j^i where each:
 - ▶ executes on a specific machine
 - ▶ lasts a fixed time
 - operation order is fixed: $\prec \dots \prec op_m^i$

A different scheduling model

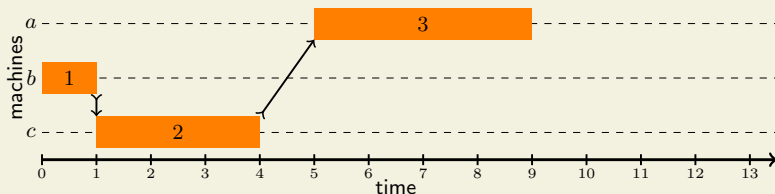
Example (The Job-Shop Scheduling Problem)



- m **machines**, each processing one operation at a time
- n **jobs**, $job_i = \langle op_1^i, op_2^i, \dots, op_m^i \rangle$
 - a job is a sequence of **operations**, op_j^i where each:
 - ▶ executes on a specific machine
 - ▶ lasts a fixed time
 - operation order is fixed: $\prec \dots \prec op_m^i$

A different scheduling model

Example (The Job-Shop Scheduling Problem)



■ m **machines**, each processing one operation at a time

■ n **jobs**, $job_i = \langle op_1^i, op_2^i, \dots, op_m^i \rangle$

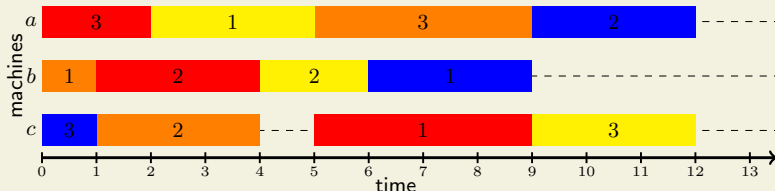
- a job is a sequence of **operations**, op_j^i
 - ▶ executes on a specific machine
 - ▶ lasts a fixed time

$a \prec b$ means
"b cannot start
until a ends"

- operation order is fixed: $op_1^i \prec op_2^i \prec \dots \prec op_m^i$

A different scheduling model

Example (The Job-Shop Scheduling Problem)



■ m **machines**, each processing one operation at a time

■ n **jobs**, $job_i = \langle op_1^i, op_2^i, \dots, op_m^i \rangle$

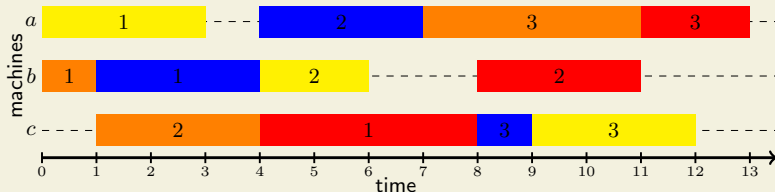
- a job is a sequence of **operations**, op_j^i
 - ▶ executes on a specific machine
 - ▶ lasts a fixed time

$a \prec b$ means
"b cannot start
until a ends"

- operation order is fixed: $op_1^i \prec op_2^i \prec \dots \prec op_m^i$

A different scheduling model

Example (The Job-Shop Scheduling Problem)



■ m machines, each processing one operation at a time

■ n jobs, $job_i = \langle op_1^i, op_2^i, \dots, op_m^i \rangle$

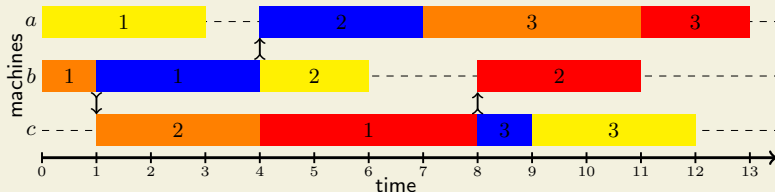
- a job is a sequence of operations, op_j^i
 - ▶ executes on a specific machine
 - ▶ lasts a fixed time

$a \prec b$ means
"b cannot start
until a ends"

- operation order is fixed: $op_1^i \prec op_2^i \prec \dots \prec op_m^i$

A different scheduling model

Example (The Job-Shop Scheduling Problem)



■ m **machines**, each processing one operation at a time

■ n **jobs**, $job_i = \langle op_1^i, op_2^i, \dots, op_m^i \rangle$

- a job is a sequence of **operations**, op_j^i
 - ▶ executes on a specific machine
 - ▶ lasts a fixed time

$a \prec b$ means
"b cannot start
until a ends"

- operation order is fixed: $op_1^i \prec op_2^i \prec \dots \prec op_m^i$



Outline

Outline

Scheduling

Examples

General

RCSP

Propagation

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



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 \neg (scheduling)?

Outline

Scheduling

Examples

General

RCSP

Propagation

Conclusion

■ 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

Outline

Scheduling

RCSP

Intro

cumulative

Propagation

Conclusion



The Resource Constrained Scheduling Problem (RCSP)

Outline

Scheduling

RCSP

Intro

cumulative

Propagation

Conclusion

■ 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)

Outline

Scheduling

RCSP

Intro

cumulative

Propagation

Conclusion

- 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

Outline

Scheduling

RCSP

Intro

cumulative

Propagation

Conclusion

■ 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

Outline

Scheduling

RCSP

Intro

cumulative

Propagation

Conclusion

■ 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



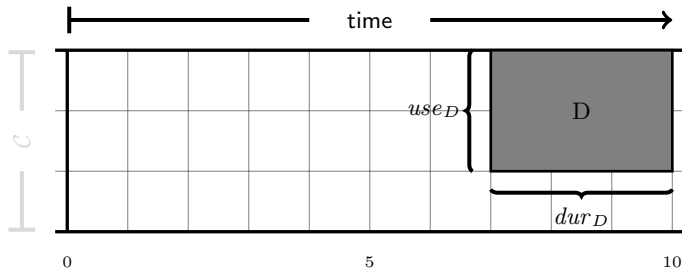
Notation for cumulative scheduling problems

Part 1: Tasks

Notation

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

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





Notation for cumulative scheduling problems

Part 1: Tasks

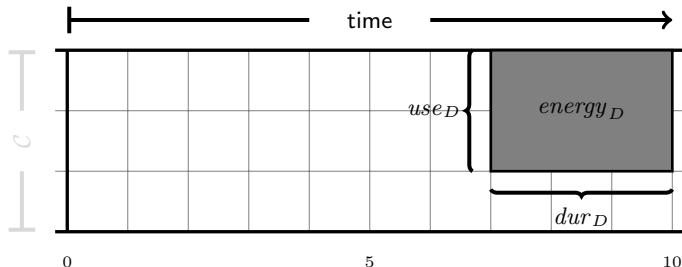
Notation

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

- Set *Tasks* of n tasks, where for $i \in \text{Tasks}$:

- fixed resource requirement: use_i
- fixed duration: dur_i
- energy: $energy_i = use_i \cdot dur_i$

- One shared resource of constant capacity C .





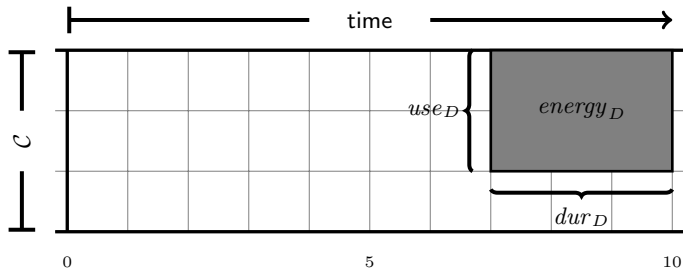
Notation for cumulative scheduling problems

Part 1: Tasks

Notation

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

- 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 .





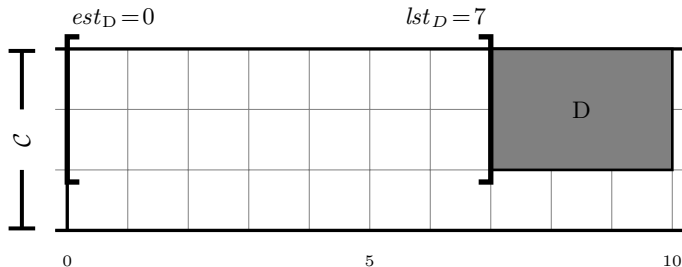
Notation for cumulative scheduling problems

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 .





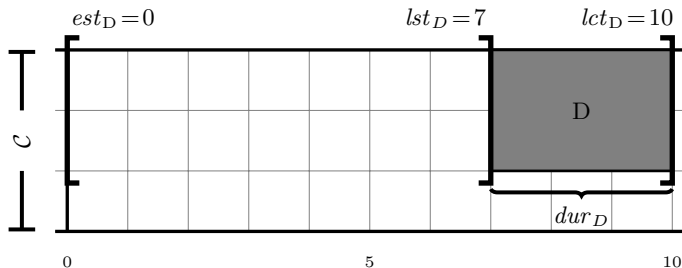
Notation for cumulative scheduling problems

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 .





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:**

$$\forall t \in time: \sum_{\substack{i \in Tasks \\ start_i \leq t \leq start_i + dur_i}} use_i \leq C$$

- 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

Outline

Scheduling

RCSP

Propagation

Time

Table

Overload

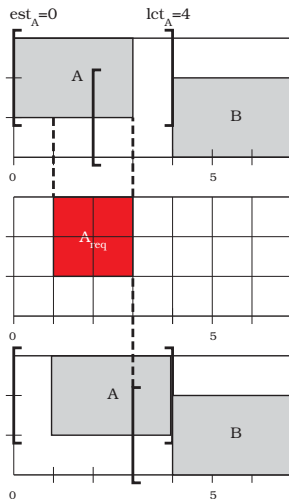
Edge-

Finding

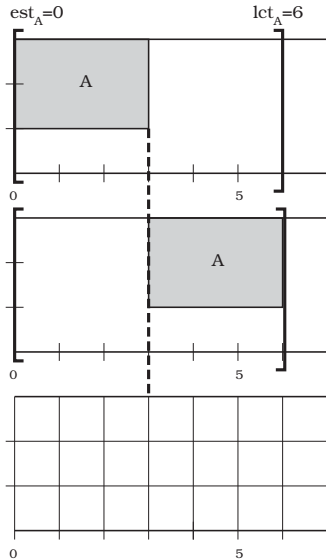
others

Conclusion

- Outline
- Scheduling
- RCSP
- Propagation
- Time Table**
- Overload
- Edge-Finding
- others
- Conclusion



What if there is no required part?





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

Outline

Scheduling

RCSP

Propagation

Time

Table

Overload

Edge-

Finding

others

Conclusion



Notation for cumulative scheduling problems

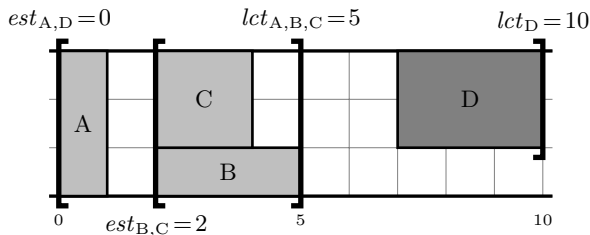
Part 3: Sets of Tasks

Notation

Sets of tasks (e.g., $\{A, B, C\}$) are denoted ω , θ , etc.

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

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





Notation for cumulative scheduling problems

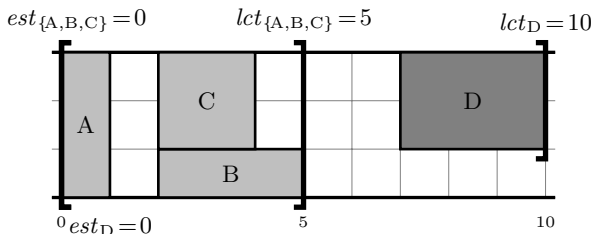
Part 3: Sets of Tasks

Notation

Sets of tasks (e.g., $\{A, B, C\}$) are denoted ω , θ , etc.

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

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

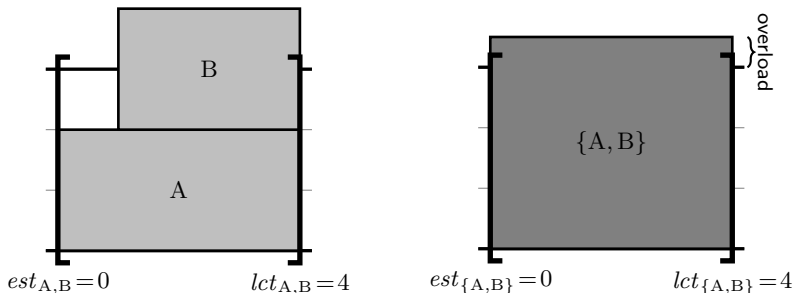


e-feasibility and overload checking

- 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:

$$\forall \theta \subseteq tasks: energy_{\theta} > \mathcal{C}(lct_{\theta} - est_{\theta}) \implies \text{Overload}$$



- **e-feasible**: no overload for any $\theta \subseteq Tasks$



Is e-feasibility stronger than time tabling?

Outline

Scheduling

RCSP

Propagation

Time

Table

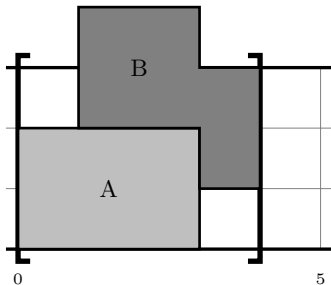
Overload

Edge-

Finding

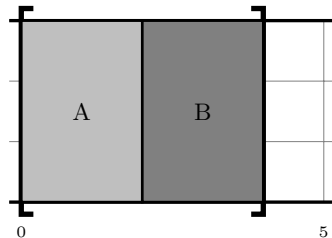
others

Conclusion



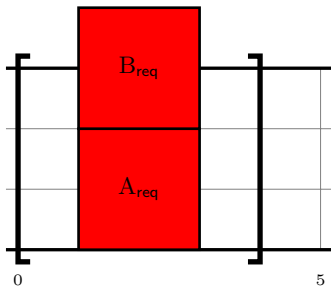
$$est_{\{A,B\}} = 0$$

$$lct_{\{A,B\}} = 4$$



$$est_{\{A,B\}} = 0$$

$$lct_{\{A,B\}} = 4$$





Is time tabling stronger than e-feasibility?

Outline

Scheduling

RCSP

Propagation

Time

Table

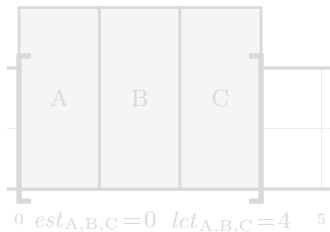
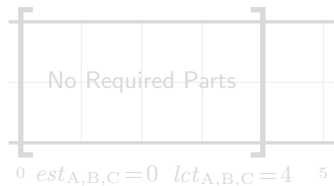
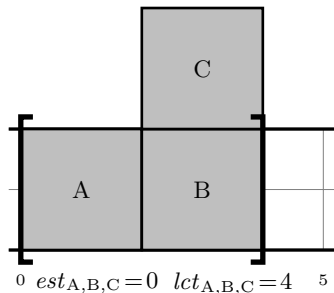
Overload

Edge-

Finding

others

Conclusion



A trivial overload,
not e-feasible,
but ignored by time tabling.



Is time tabling stronger than e-feasibility?

Outline

Scheduling

RCSP

Propagation

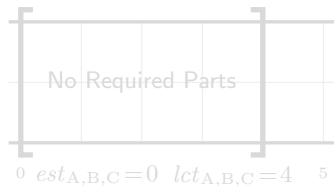
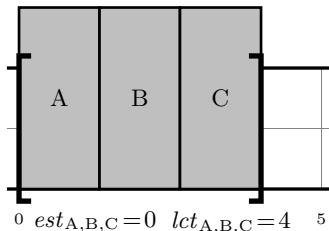
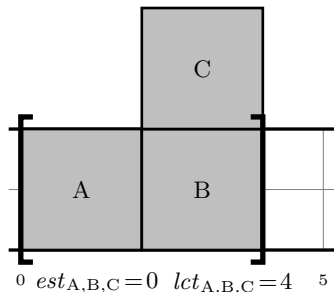
Time
Table

Overload

Edge-
Finding

others

Conclusion



A trivial overload,
not e-feasible,
but ignored by time tabling.



Is time tabling stronger than e-feasibility?

Outline

Scheduling

RCSP

Propagation

Time

Table

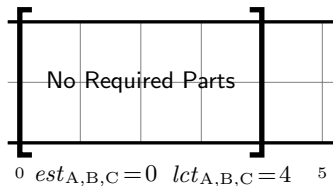
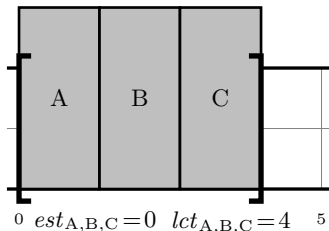
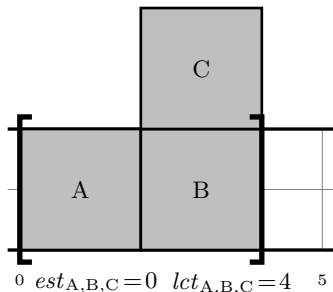
Overload

Edge-

Finding

others

Conclusion



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

Outline

Scheduling

RCSP

Propagation

Time

Table

Overload

Edge-
Finding

others

Conclusion



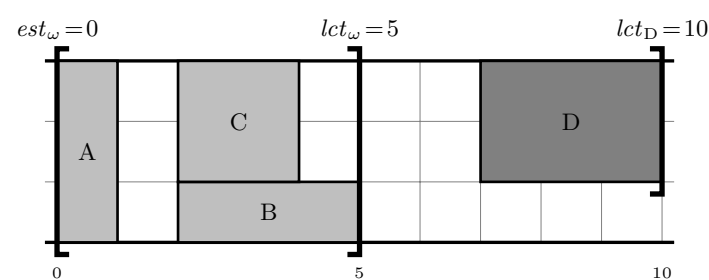
Improving on overload checking

- 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_ω ?
 - If not, D must end after lct_ω .

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

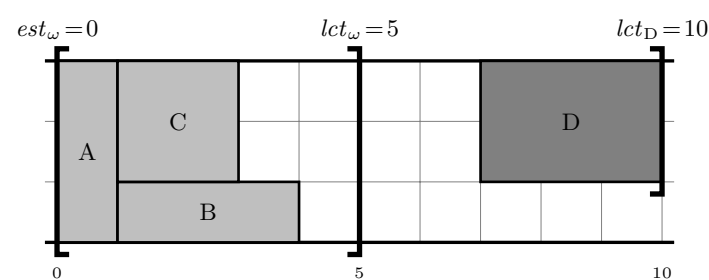


*Running example due to [VILÍM, 2009]

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_ω ?
 - If not, D must end after lct_ω .

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



*Running example due to [VILÍM, 2009]

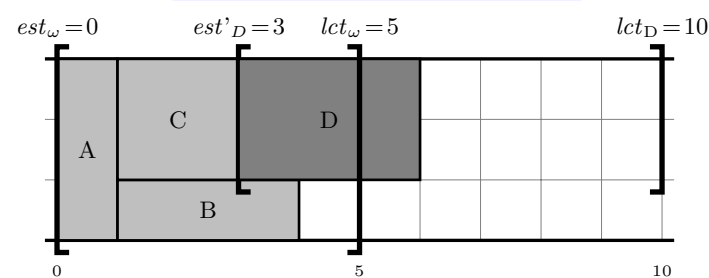


Edge-finding: deducing new precedence relations

- 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_ω ?
 - If not, D must end after lct_ω .

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

"D ends after the end of all tasks in ω "

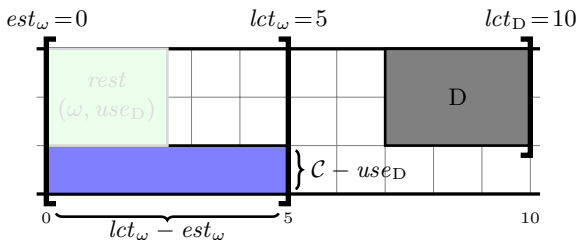


*Running example due to [VILÍM, 2009]

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.

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

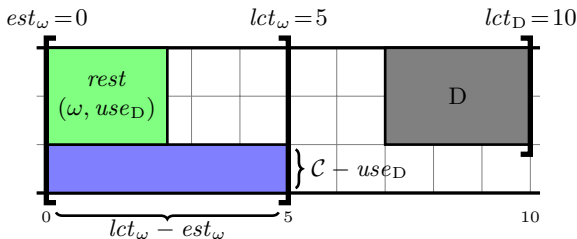


$$rest(\omega, use_i) = \begin{cases} energy_{\omega} - (C - use_i)(lct_{\omega} - est_{\omega}) & \text{if } \omega \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

How much can the bound be updated?

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

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

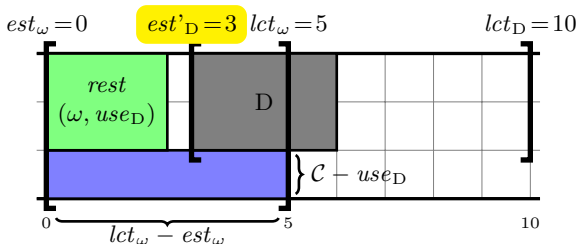


$$rest(\omega, use_i) = \begin{cases} energy_{\omega} - (C - use_i)(lct_{\omega} - est_{\omega}) & \text{if } \omega \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

How much can the bound be updated?

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

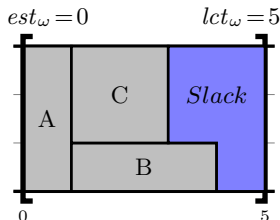
$$est'_i \geq est_{\omega} + \left\lceil \frac{rest(\omega, use_i)}{use_i} \right\rceil$$



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

Minimum Slack

- **Slack** measures the capacity not used by a task set ω



$$\omega = \{A, B, C\}$$

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

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

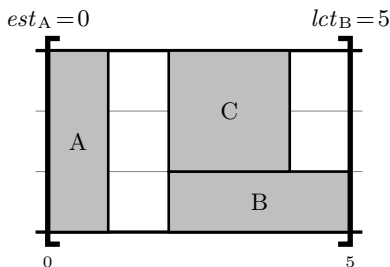
Conjecture

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 $\mathcal{O}(n)$ **meaningful** values for each bound
- Task intervals**: sets of tasks, bounded by tasks

$$\omega_L^U = \{i \in T \mid est_i \geq est_L \wedge lct_i \leq lct_U\}$$

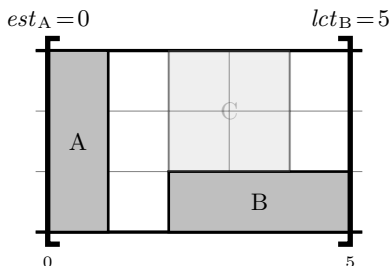


- $\omega_A^B = \{A, B, C\}$
- Removing task C reduces the energy, but not the available capacity

Task Intervals

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

$$\omega_L^U = \{i \in T \mid est_i \geq est_L \wedge lct_i \leq lct_U\}$$



- $\omega_A^B = \{A, B, C\}$
- 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  $\omega_L^U \triangleleft i$  then  
         $est_i = \text{update based on } \omega_L^U$ 
```

- Two problems:
 - Computing the update to est_i is (generally) not $\mathcal{O}(n)$.
 - 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  $\omega_L^U \triangleleft i$  then  
         $est_i = \text{update based on } \omega_L^U$ 
```

- Two problems:

- Computing the update to est_i is (generally) not $\mathcal{O}(n)$.
- 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  $\omega_L^U \triangleleft i$  then
           $est_i = \text{update based on } \omega_L^U$ 
```

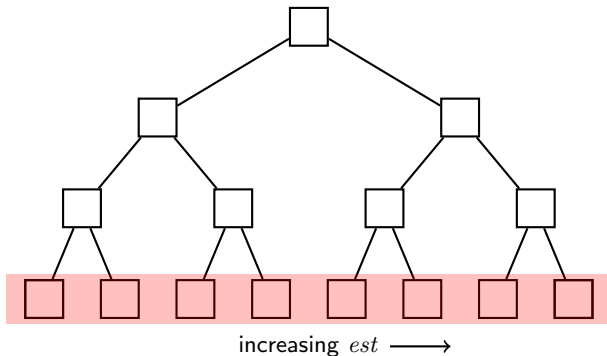
- Two problems:

- Computing the update to est_i is (generally) not $\mathcal{O}(n)$.
- The minimum slack interval is not always the best interval.



Θ -trees for Edge-Finding

[Vilím, 2009]

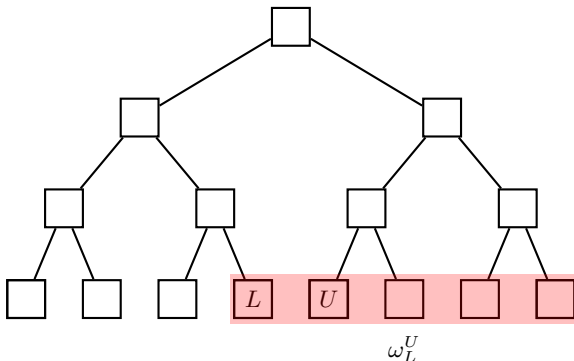


- Arrange all tasks as the leaves of a balanced, binary tree.
 - Sorted by increasing est .
- Let U be the task with the largest lct
 - Notice: for any L , the interval ω_L^U is all tasks to the right.



Θ -trees for Edge-Finding

[Vilím, 2009]

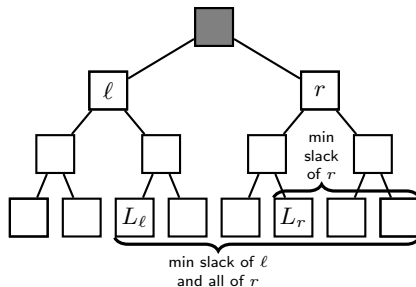


- Arrange all tasks as the leaves of a balanced, binary tree.
 - Sorted by increasing *est.*
- Let U be the task with the largest *lct*
 - Notice: for any L , the interval ω_L^U is all tasks to the right.



Θ -trees for Edge-Finding

[Vilím, 2009]



- 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:

$$energy_{\theta \cup \{i\}} > C(lct_{\theta} - est_{\theta \cup \{i\}}) \implies \theta \triangleleft i$$

$$\underbrace{C \cdot est_{\theta \cup \{i\}} + energy_{\theta \cup \{i\}}}_{Env_{\theta \cup \{i\}}} > C \cdot lct_{\theta} \implies \theta \triangleleft i$$

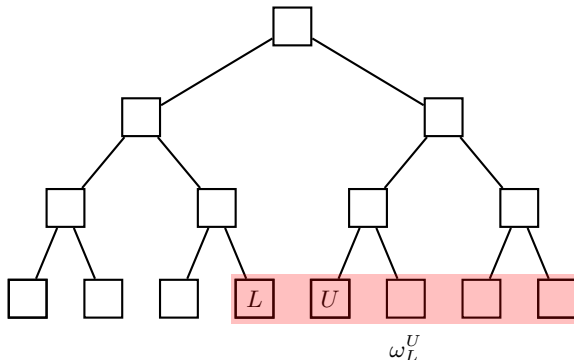
- Can compute the **energy envelope** of a node v (Env_v) by comparing child nodes, ℓ and r :

$$Env_v = \begin{cases} C \cdot est_x + energy_x & \text{if } v \text{ is a leaf, for task } x \\ \max\{Env_r, Env_{\ell} + energy_r\} & \text{if } v \text{ is not a leaf} \end{cases}$$

$$energy_v = \begin{cases} energy_x & \text{if } v \text{ is a leaf, for task } x \\ energy_r + energy_{\ell} & \text{if } v \text{ is not a leaf} \end{cases}$$

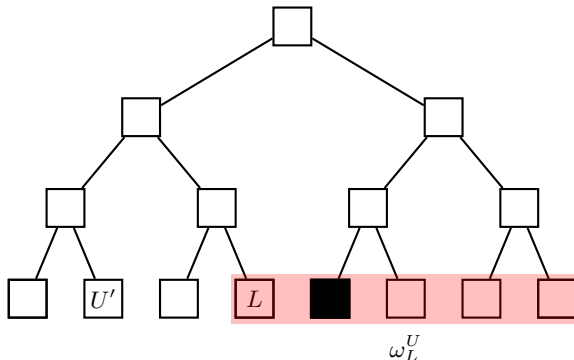
- Maximizing energy envelope = minimizing slack

Tree Structure Allows for Efficient Recomputation



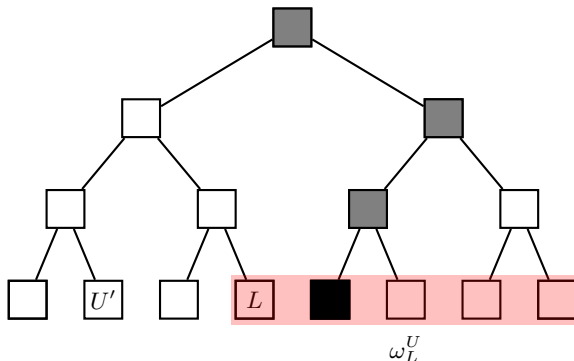
- So far, the tree requires more work, not less!
- Make U a zero-energy task, then recompute $\mathcal{O}(\log n)$ ancestor nodes.
- n upper bounds, so $\mathcal{O}(n \log n)$ to check them all.
 - Unfortunately, that only holds for unary resources...

Tree Structure Allows for Efficient Recomputation



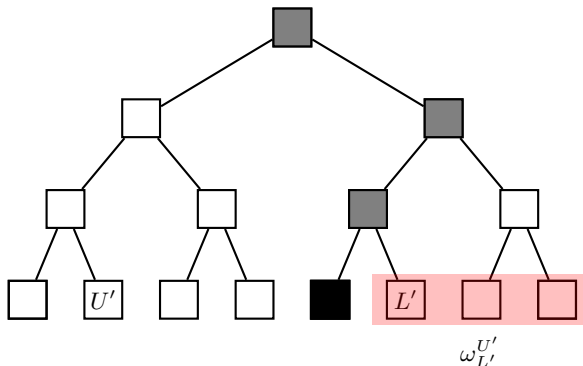
- So far, the tree requires more work, not less!
- Make U a zero-energy task, then recompute $\mathcal{O}(\log n)$ ancestor nodes.
- n upper bounds, so $\mathcal{O}(n \log n)$ to check them all.
 - Unfortunately, that only holds for unary resources...

Tree Structure Allows for Efficient Recomputation



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

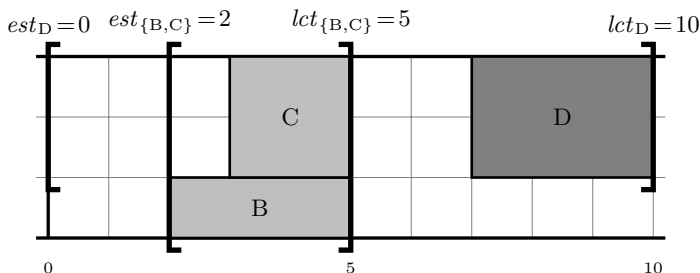
Tree Structure Allows for Efficient Recomputation



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

For cumulative resources, must consider subsets of ω

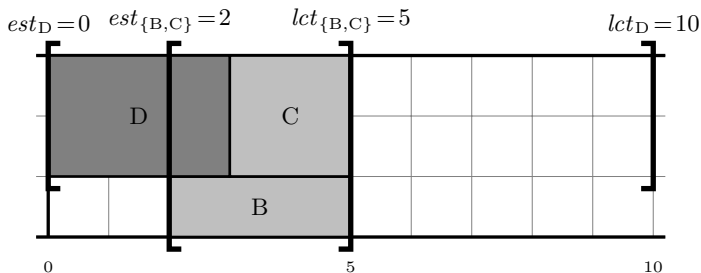
- $\{B, C\}$ not energetic enough to require $\{B, C\} \leq D$.
- $\{A, B, C\} \leq D$, and $lct_{A,B,C} \geq lct_{B,C}$,
 - therefore D must end after $lct_{B,C}$.



- Must find $\theta_\ell^u \subseteq \omega_L^U$ that yields the strongest update.

For cumulative resources, must consider subsets of ω

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

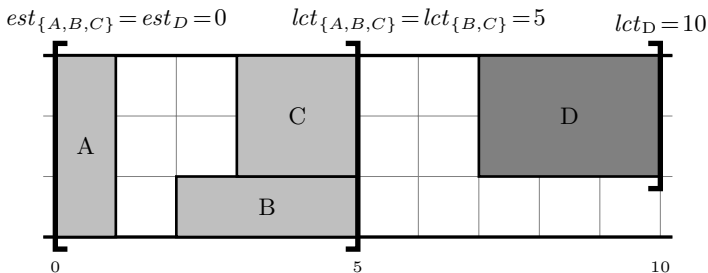


- Must find $\theta_\ell^u \subseteq \omega_L^U$ that yields the strongest update.



For cumulative resources, must consider subsets of ω

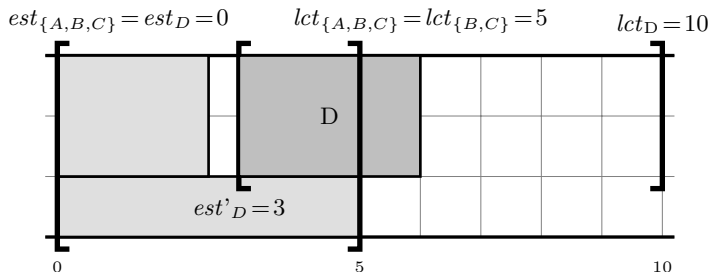
- $\{B, C\}$ not energetic enough to require $\{B, C\} \prec D$.
- $\{A, B, C\} \prec D$, and $lct_{A,B,C} \geq lct_{B,C}$,
 - therefore D must end after $lct_{B,C}$.



- Must find $\theta_\ell^u \subseteq \omega_L^U$ that yields the strongest update.

For cumulative resources, must consider subsets of ω

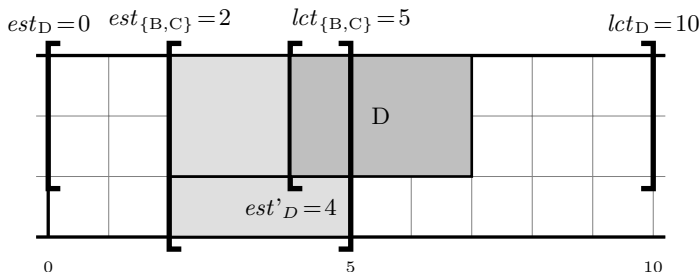
- $\{B, C\}$ not energetic enough to require $\{B, C\} \prec D$.
- $\{A, B, C\} \prec D$, and $lct_{A,B,C} \geq lct_{B,C}$,
 - therefore D must end after $lct_{B,C}$.



- Must find $\theta_\ell^u \subseteq \omega_L^U$ that yields the strongest update.

For cumulative resources, must consider subsets of ω

- $\{B, C\}$ not energetic enough to require $\{B, C\} \prec D$.
- $\{A, B, C\} \prec D$, 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

Outline

Scheduling

RCSP

Propagation

Time

Table

Overload

Edge-

Finding

others

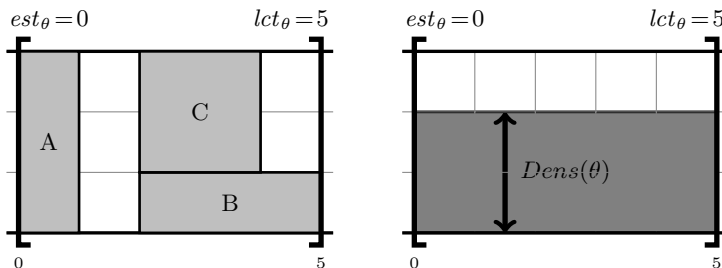
Conclusion

- 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 $\mathcal{O}(kn^2)$ complexity, where k is the number of **distinct** capacity requirements.
- [VILÍM, 2009] shows how to use the Θ -tree method outlined here to find the strongest subset with $\mathcal{O}(kn \log n)$ complexity.
 - It is significantly more complicated than what you have seen today.

Definition: Density

- For a task interval θ , the **density** of θ 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]

```
for  $U \in T$  do
  for  $i \in T$  by decreasing  $est_i$  do
    if  $lct_i \leq lct_U$  then
      if  $Dens(\omega_i^U) > Dens(\omega_L^U)$  then
         $L \leftarrow i$ 
      else
         $Dupd_i \leftarrow$  update based on  $\omega_L^U$ 
    for  $i \in T$  by increasing  $est_i$  do
      if  $Slack(\theta_i^U) < Slack(\theta_\ell^U)$  then
         $\ell \leftarrow i$ 
      if  $lct_i > lct_U$  then
         $SLupd_i \leftarrow$  update based on  $\theta_\ell^U$ 
        if  $\theta_\ell^U \leq i$  then
           $est'_i \leftarrow \max(Dupd_i, SLupd_i)$ 
```

Outline

Scheduling

RCSP

Propagation

Time

Table

Overload

Edge-
Finding

others

Conclusion



A comparison of the algorithms

Outline

Scheduling

RCSP

Propagation

Time

Table

Overload

Edge-
Finding

others

Conclusion

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

Outline

Scheduling

RCSP

Propagation

Time

Table

Overload

Edge-

Finding

others

Conclusion

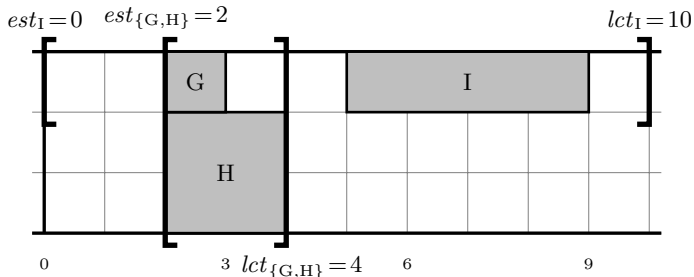
Extended edge-finding

- Recall the edge-finding rule:

$$energy_{\omega} + energy_i > C \cdot (lct_{\omega} - est_{\omega \cup \{i\}}) \implies \omega \prec i$$

- In this example

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





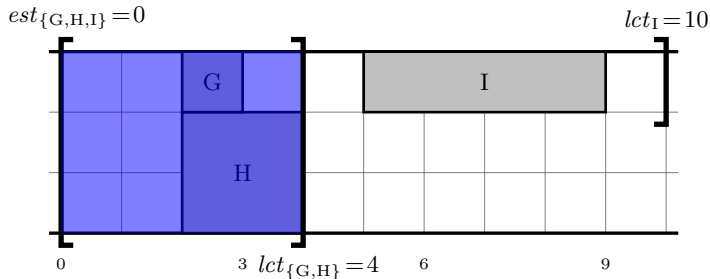
Extended edge-finding

- Recall the edge-finding rule:

$$energy_{\omega} + energy_i > C \cdot (lct_{\omega} - est_{\omega \cup \{i\}}) \implies \omega \prec i$$

- In this example

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





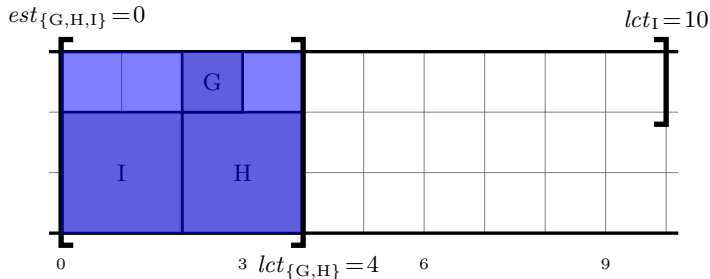
Extended edge-finding

- Recall the edge-finding rule:

$$energy_{\omega} + energy_i > C \cdot (lct_{\omega} - est_{\omega \cup \{i\}}) \implies \omega \prec i$$

- In this example

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



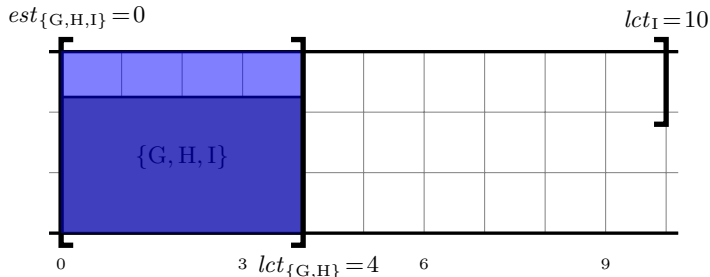
Extended edge-finding

- Recall the edge-finding rule:

$$energy_{\omega} + energy_i > C \cdot (lct_{\omega} - est_{\omega \cup \{i\}}) \implies \omega \prec i$$

- In this example

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



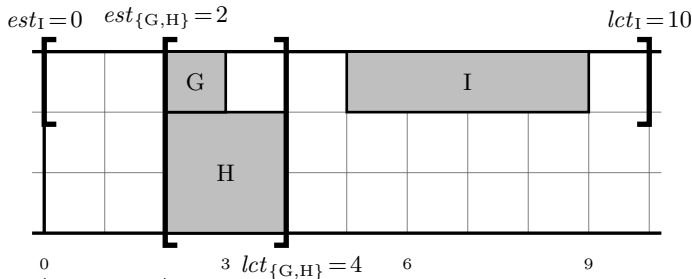
Extended edge-finding

- Recall the edge-finding rule:

$$energy_{\omega} + energy_i > C \cdot (lct_{\omega} - est_{\omega \cup \{i\}}) \implies \omega \prec i$$

- In this example

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



Edge-finding is too loose a relaxation
for tasks with an est in this range.



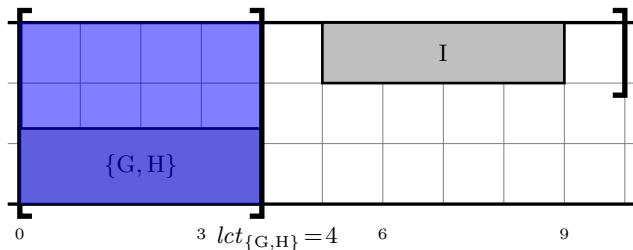
Extended edge-finding

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

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

$$est_{\{G,H,I\}} = 0$$

$$lct_I = 10$$



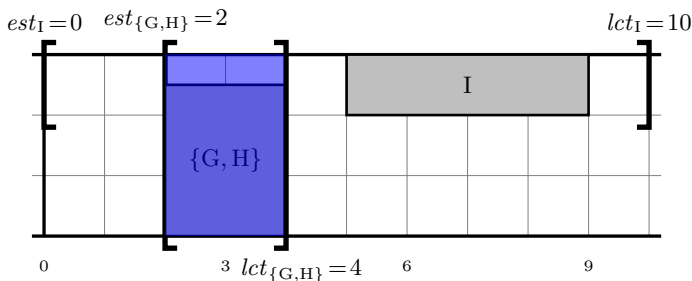
Intuition

Split task i into two parts: the largest part that could execute before est_ω , and the remainder. Check the edge-finding condition on ω and the remainder.

Extended edge-finding

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

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



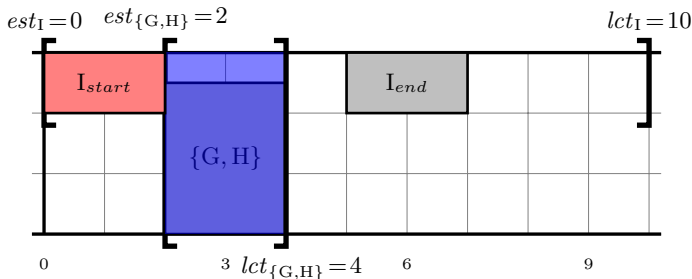
Intuition

Split task i into two parts: the largest part that could execute before est_ω , and the remainder. Check the edge-finding condition on ω and the remainder.

Extended edge-finding

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

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



Intuition

Split task i into two parts: the largest part that could execute before est_ω , and the remainder. Check the edge-finding condition on ω and the remainder.



Not-first/not-last

Outline

Scheduling

RCSP

Propagation

Time

Table

Overload

Edge-

Finding

others

Conclusion

- 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 θ that must be scheduled before (after) i .
 - Requires reasoning on the minimum earliest completion time of any task in θ .
 - A “lazy” $\mathcal{O}(n^2 \log n)$ algorithm reported at CP 2010.



Energetic Reasoning

Outline

Scheduling

RCSP

Propagation

Time

Table

Overload

Edge-
Finding

others

Conclusion

- 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

Outline

Scheduling

RCSP

Propagation

Conclusion



Baker K.R., Trietsch D.

Principles of Sequencing and Scheduling.

John Wiley & Sons, Hoboken, New Jersey (2009).

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.

Springer, Berlin / Heidelberg (2001).

Algorithms out of date, but remains the most complete reference.



Vilím P.

Edge finding filtering algorithm for discrete cumulative resources in $O(kn \log n)$.

Gent I.P., ed., *CP 2009, LNCS*, vol. 5732. Springer (2009).

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.

Master's thesis, Uppsala University (2010).

Details omitted by [VILÍM, 2009], plus explanations and corrections.

Further Reading II

Outline

Scheduling

RCSP

Propagation

Conclusion



Mercier L., Van Hentenryck P.

Edge finding for cumulative scheduling.

INFORMS Journal on Computing, 20:143 (2008).

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.

Lee J.H.M., ed., *CP 2011, LNCS*, vol. 6876. Springer (2011).

Simpler to implement than Θ -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.

Achterberg T., Beck J.C., eds., *CPAIOR 2011, LNCS*, vol. 6697, pp. 230–245. Springer (2011).

Interesting hybrid of edge-finding, time tabling, and parts of energetic reasoning, all in $\mathcal{O}(n^2)$ time.