

Constraint-based Scheduling

Hana Rudová

Fakulty of Informatics, Masaryk University
Brno, Czech Republic

<http://www.fi.muni.cz/~hanka>

Some topics are described in

- Philippe Baptiste, Philippe Laborie, Claude Le Pape, Wim Nuijten: Constraint-based Scheduling and Planning. Chapter 22 in Handbook of Constraint programming, pages 761-799, Elsevier, 2006.
- Roman Barták: Filtering Techniques in Planning and Scheduling, ICAPS 2006, June 6-10, 2006, Cumbria, England
<http://www.plg.inf.uc3m.es/icaps06/preprints/i06-tu2-allpapers.pdf>
- Philippe Baptiste, Claude Le Pape, Wim Nuijten: Constraint-based Scheduling, Kluwer Academic Publishers, 2001.

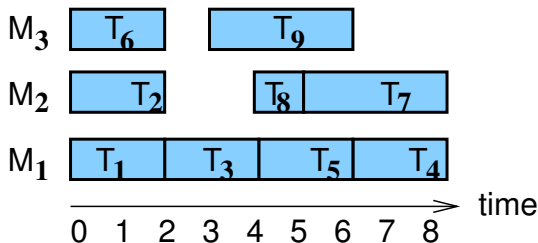
Constraint-based Scheduling: Introduction

- 1 Scheduling
- 2 CSP model
- 3 Resources
- 4 Optimization
- 5 Problem: Basic Class Timetabling

Scheduling

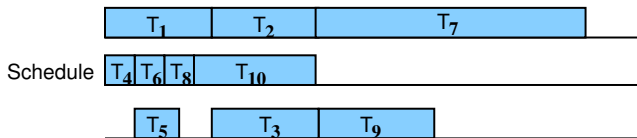
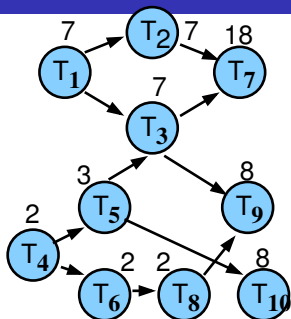
- Scheduling
optimal resource allocation of a given set of activities in time
 - resource or machine
 - activity or task
- Machine $M_j, j = 1, \dots, 3$ Task $T_i, i = 1, \dots, 9$

Machine-oriented Gantt chart



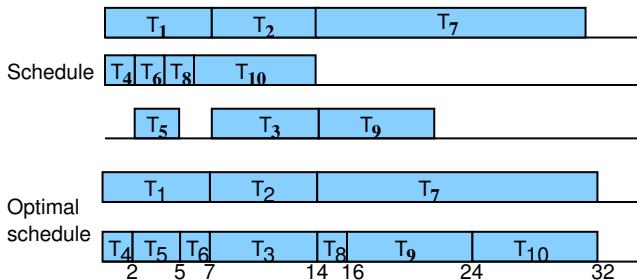
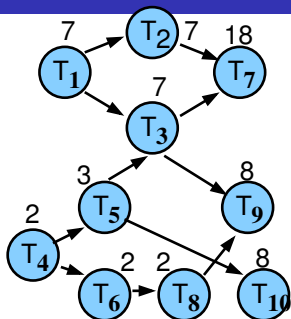
Example: Bicycle Assembly

- 3 workers who can perform tasks
- 10 tasks with its own duration
- Precedence constraints ($T_i \ll T_j$)
 - activity must be processed before other activity
- No preemption
 - activity cannot be interrupted during processing



Example: Bicycle Assembly

- 3 workers who can perform tasks
- 10 tasks with its own duration
- Precedence constraints ($T_i \ll T_j$)
 - activity must be processed before other activity
- No preemption
 - activity cannot be interrupted during processing



Example: Classroom allocation

- One day seminar with several courses to be presented in several available rooms
- 8:00am – 4:00pm (periods 1,2,...,8)
- 14 courses (A,B, ... N)
each course has several meetings with pre-assigned time periods
- 5 rooms (1,2,3,4,5) ... resources
- Find suitable room for each meeting

Demands:	Course	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	Periods	2	8	4	1	3	6	7	2	1	5	6	3	8	2
				5	2	4		8	3	2		7	4		3
						5									4

Solution = Schedule/Timetable:	Periods	1	2	3	4	5	6	7	8
	Room 1	D	D		C	C	F		B
	Room 2	I	I	E	E	E		G	G
	Room 3		H	H		J	K	K	
	Room 4		N	N	N				M
	Room 5		A	L	L				

Scheduling problems

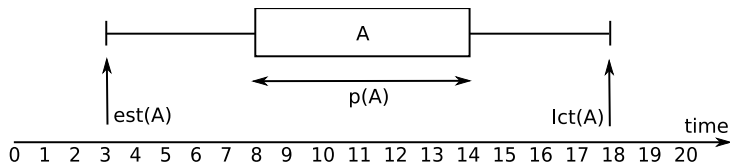
- Project planning and scheduling
 - software project planning
- Machine scheduling
 - allocation of jobs to computational resources
- Scheduling of flexible assembly systems
 - car production
- Employees scheduling
 - nurse rostering
- Transport scheduling
 - gate assignment for flights
- Sports scheduling
 - schedule for NHL
- Educational timetabling
 - timetables at school
- ...

Scheduling as a CSP: time assignment

Activity A is an entity occupying some space (resources) and time

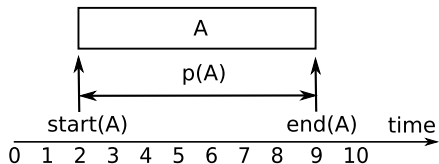
Variables and their domains for each activity for time assignment

- $start(A)$: start time of the activity
 - activity cannot start before its **release date**
 - $est(A) = \min(start(A))$, earliest start time
- $end(A)$: completion time of the activity
 - activity must finish before the **deadline**
 - $lct(A) = \max(end(A), \text{latest completion time})$
- $p(A)$: processing time (duration) of the activity
 - $start(A) = \{est(A), \dots, (lct(A)-p(A))\}$
 - $end(A) = \{(est(A)+p(A)), \dots, lct(A)\}$



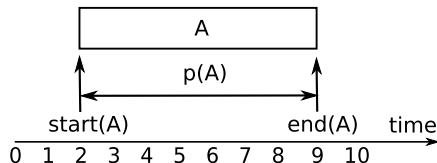
Scheduling as a CSP: basic constraints I.

- Non-preemptive activity: no interruption during processing
 - $\text{start}(A) + p(A) = \text{end}(A)$

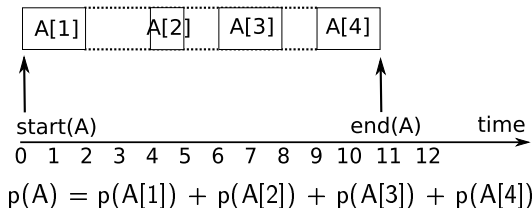


Scheduling as a CSP: basic constraints I.

- **Non-preemptive activity**: **no interruption** during processing
 - $\text{start}(A) + p(A) = \text{end}(A)$

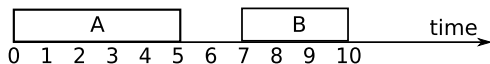


- **Preemptible activity**: **can be interrupted** during its processing
 - $\text{start}(A) + p(A) \leq \text{end}(A)$



Scheduling as a CSP: basic constraints II.

- Sequencing $A \ll B$ of activities A, B
(also: precedence constraint between activities A, B)
 - $\text{end}(A) \leq \text{start}(B)$



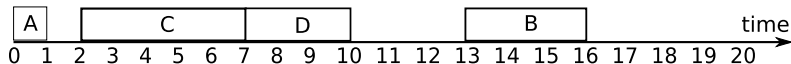
- Disjunctive constraint: non-overlapping of activities A, B
 - non-preemptive activities
 - $A \ll B$ or $B \ll A$
 - $\text{end}(A) \leq \text{start}(B)$ or $\text{end}(B) \leq \text{start}(A)$
 - related with the idea of unary resource

Domain variables for resources

- $cap(A)$: requested capacity of the resource
 - unary resources
 - cumulative resources
 - producible/consumable resources
- $resource(A)$: alternative resources for A

Unary (disjunctive) resources

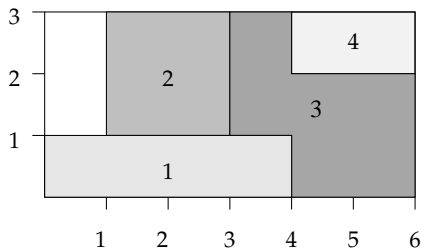
- Each activity requests unary capacity of the resource: $cap(A)=1$
- Single activity can be processed at given time
- Any two non-preemptive activities are related by the disjunctive constraint $A \ll B$ or $B \ll A$



- Example: one machine with jobs running on it

Cumulative (discrete) resources

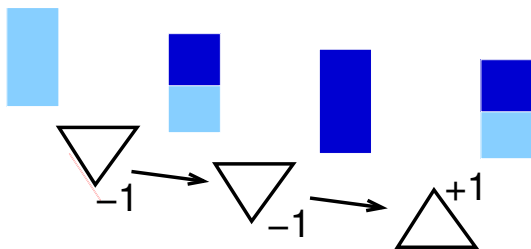
- Each activity uses some capacity of the resource $cap(A)$
- Several activities can be processed in parallel if a resource capacity is not exceeded



- Example: multi-processor computer with parallel jobs

Producible/consumable resources

- Resource = reservoir
- Activity consumes some quantity of the resource $cap(A) < 0$ or activity produces some quantity of the resource $cap(A) > 0$
- Minimal capacity is requested (consumption) and maximal capacity cannot be exceeded (production)



- Example: inventory for some products, activities producing them and activities using them in other production

- Activity can be processed on a set of alternative resources
 - defined by the domain variable $\text{resource}(A)$
- One of them is selected for the activity
- Alternative unary resources
 - activity can be processed on any of the unary resources
 - can be modeled as one cumulative resource with resource capacity corresponding to the number of alternative unary resources
 - suitable for symmetric unary resources
- Example: any of the persons can process set of tasks

Various criteria and objective function

Common criteria: **makespan**

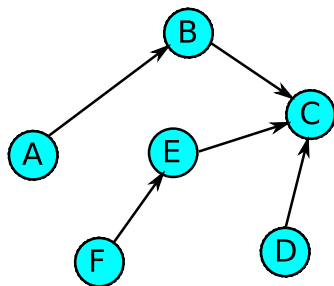
- completion time of the last activity
- modeling
 - introduced a new additional activity L , $p(L)=0$
 - added precedence constraint
for each activity T with no successor: $T \ll L$

Optimization

Various criteria and objective function

Common criteria: [makespan](#)

- completion time of the last activity
- modeling
 - introduced a new additional activity L, $p(L)=0$
 - added precedence constraint for each activity T with no successor: $T \ll L$

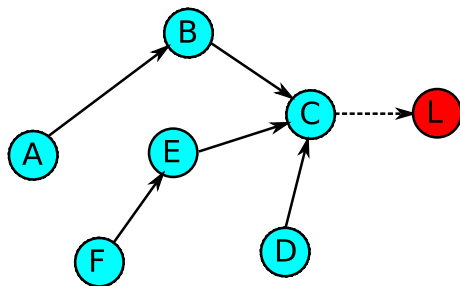


Optimization

Various criteria and objective function

Common criteria: **makespan**

- completion time of the last activity
- modeling
 - introduced a new additional activity L, $p(L)=0$
 - added precedence constraint for each activity T with no successor: $T \ll L$



- makespan = start(L)

Problem: Basic Class Timetabling

Create a schedule for one hour classes with given earliest and latest starting time. All classes are taught in one classroom. Physical training and drawing should be taught as latest as possible.

Class	Min	Max
Drawing	3	6
Physical T.	3	4
Chemistry	2	5
Math	2	4
Biology	3	4
Computers	1	6

CSP Model: Basic Class Timetabling

- Variables:

- for start time of each class: Drawing, PhysicalT, Chemistry, ...

- Domain:

- earliest and latest starting time

Drawing={3...6}, PhysicalT={3...4}, Chemistry={2...5}

Math={2...4}, Biology={3...4}, Computers={1...6}

- Classroom = unary resource

- with all start times for classes
- with unit processing time

or better with: all-different constraint over start time variables

- Optimization: maximize (Drawing+PhysicalT)

- Solutions:

Drawing=6, PhysicalT=3, Chemistry=5, Math=2, Biology=4, Computers=1

Drawing=6, PhysicalT=4, Chemistry=5, Math=2, Biology=3, Computers=1

Optimal solution:

Drawing=6, PhysicalT=4, Chemistry=5, Math=2, Biology=3, Computers=1

6 Machine Scheduling

- Machine scheduling with unary resource
- Scheduling with cumulative resource
- Job-shop problem

7 Timetabling

8 Employees Scheduling

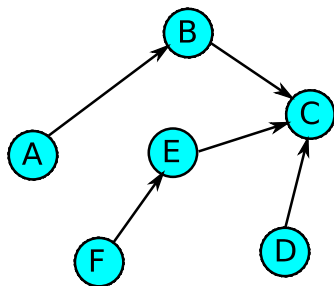
Machine scheduling with unary resource: problem & example

Problem: Create a schedule for several tasks with

- earliest (est) and latest completion time (lct)
- processing time (p)
- precedence constraints given by the graph

on machine of unit capacity such that the makespan is minimized

Task T	est(T)	lct(T)	p(T)
A	0	10	2
B	0	15	3
C	5	25	4
D	0	20	1
E	10	25	5
F	0	5	3



Machine scheduling with unary resource: variables

- Start time variables $\text{start}(T)$ for each task T
- $\text{start}(T) = \{\text{est}(T), \dots, \text{lct}(T) - p(T)\}$
- Example:

Task T	$\text{est}(T)$	$\text{lct}(T)$	$p(T)$
A	0	10	2
B	0	15	3
C	5	25	4
D	0	20	1
E	10	25	5
F	0	5	3

$A = \{0..8\}$, $B = \{0..2\}$, $C = \{5..21\}$,
 $D = \{0..19\}$, $E = \{10..20\}$, $F = \{0..2\}$

Machine scheduling with unary resource: constraints

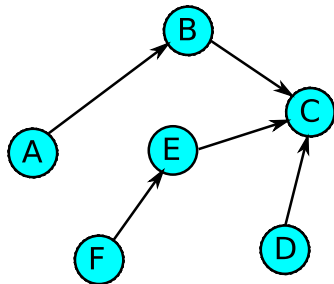
Precedence constraints for each tasks $T_1 \ll T_2$

- $\text{start}(T_1) + p(T_1) \leq \text{start}(T_2)$
- example: $A+2 \leq B$, $B+3 \leq C$,
 $F+3 \leq E$, $E+5 \leq C$, $D+1 \leq C$,

Unary resource for all tasks T given by

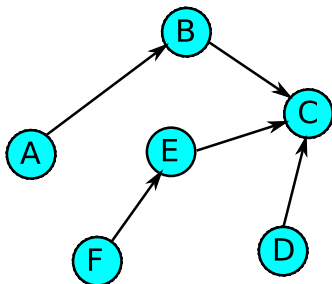
- start time variables $\text{start}(T)$
- duration $p(T)$
- example: $\text{serialized}([A,B,C,D,E,F],[2,3,4,1,5,3])$

Task T	est(T)	lct(T)	p(T)
A	0	10	2
B	0	15	3
C	5	25	4
D	0	20	1
E	10	25	5
F	0	5	3



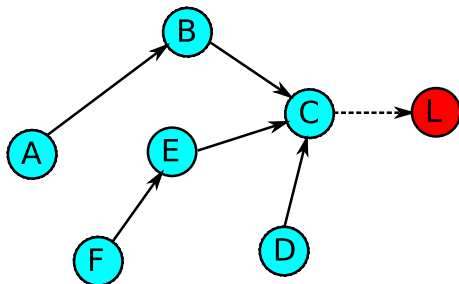
Scheduling with unary resource: optimization

- New task L with $p(L)=0$ added
- Precedence constraints
 between L and tasks with no successor added
- Example: $C+4 \leq L$



Scheduling with unary resource: optimization

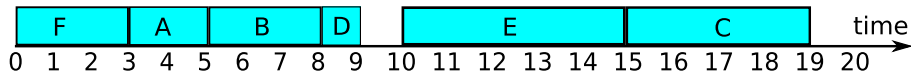
- New task L with $p(L)=0$ added
- Precedence constraints
 between L and tasks with no successor added
- Example: $C+4 \leq L$



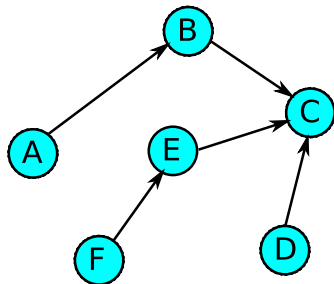
- $\text{minimize}(\text{makespan}) = \text{minimize}(\text{start}(L))$

Scheduling with unary resource: solution

Solution:



Task T	est(T)	lct(T)	p(T)
A	0	10	2
B	0	15	3
C	5	25	4
D	0	20	1
E	10	25	5
F	0	5	3



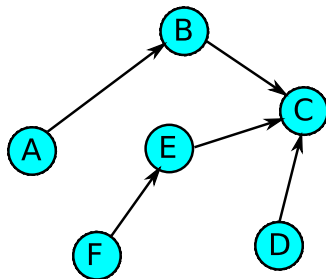
Scheduling with cumulative resource: problem & example

Problem: Create a schedule for several tasks with

- earliest (est) and latest completion time (lct)
- processing time (p)
- requested capacity of the resource (cap)
- precedence constraints given by the graph

on machine of capacity 3 such that the makespan is minimized

Task T	est(T)	lct(T)	p(T)	cap(T)
A	0	10	2	1
B	0	15	3	2
C	5	25	4	2
D	0	20	1	3
E	10	25	5	2
F	0	5	3	2



Scheduling with cumulative resource: modeling

Same model as for scheduling with unary resource with

- unary resource replaced by cumulative resource

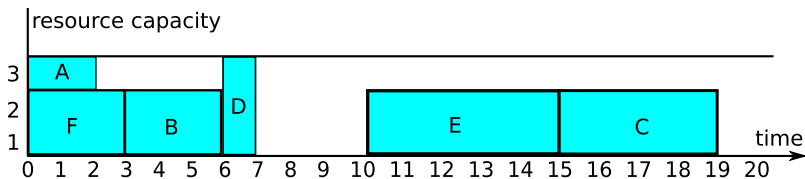
Cumulative resource for all tasks T given by

- start time variables $\text{start}(T)$
- duration $p(T)$
- requested capacity of the resource
- example: $\text{cumulative}([A,B,C,D,E,F],[2,3,4,1,5,3],[1,2,2,3,2,2],3)$

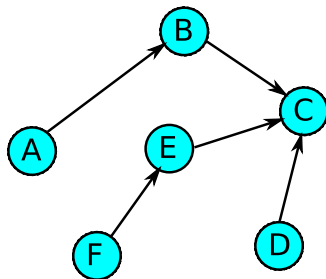
Task T	$\text{est}(T)$	$\text{lct}(T)$	$p(T)$	$\text{cap}(T)$
A	0	10	2	1
B	0	15	3	2
C	5	25	4	2
D	0	20	1	3
E	10	25	5	2
F	0	5	3	2

Scheduling with cumulative resource: solution

Solution:



Task T	est(T)	lct(T)	p(T)	cap(T)
A	0	10	2	1
B	0	15	3	2
C	5	25	4	2
D	0	20	1	3
E	10	25	5	2
F	0	5	3	2



Job-shop problem: problem

Create a schedule for several tasks such that

- each task consists of several jobs
- ordering of jobs for each task is fixed
- jobs of each tasks are processed on different dedicated machine
- machines have unit capacity
- makespan is minimized

Job-shop problem: example

- Machines: M1, M2, M3
- Tasks T1, T2 and T3 with jobs noted by (machine,task)

T1: (3,1)«(2,1)«(1,1)

T2: (1,2)«(3,2)

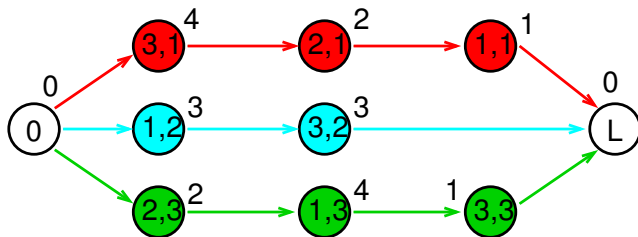
T3: (2,3)«(1,3)«(3,3)

- Processing times:

$p(31)=4, p(21)=2, p(11)=1$

$p(12)=3, p(32)=3$

$p(23)=2, p(13)=4, p(33)=1$



Additional first and last activities O and L with $p(0)=p(L)=0$

Job-shop problem: modeling

Variables and domains

- $\text{start}(i,j)$ start time variables for j -th task running on machine i

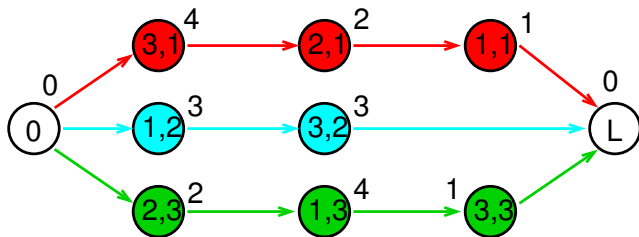
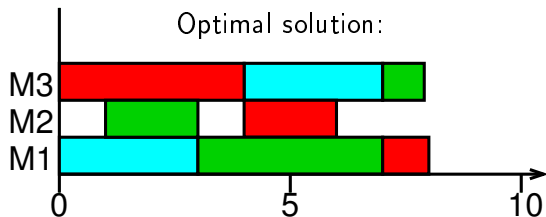
Constraints

- ordering of jobs modeled through precedence constraints
- unary resource constraint for each machine i with jobs (i,j) for all tasks j

Optimization

- $\text{minimize}(\text{makespan}) = \text{minimize}(\text{start}(L))$

Job-shop problem: solution



Class Timetabling: problem

Create schedule for N periods for classes with

- given duration
- given teacher
- given number of students
- prohibited time periods

Several classrooms (M) with specified number of seats are given.

There are sets of classes creating a curriculum

- no overlaps within curricula allowed

Class Timetabling: variables and domains

- Class represents activity with given duration
- Start time variables for each class $start(A)$
 - $start(A) = \{0, \dots, N-1\}$
 - $start(A) \neq prohibited(A)$

(N number of periods)

Class Timetabling: variables and domains

- Class represents activity with given duration
- Start time variables for each class $start(A)$
 - $start(A) = \{0, \dots, N-1\}$ (N number of periods)
 - $start(A) \neq prohibited(A)$
- Classroom represent resource
- Classrooms are ordered by the number of seats
 - smallest classroom = 0
 - largest classroom = M-1 (M number of rooms)
- Classroom variable for each class $resource(A)$
 - $resource(A) = \{K, \dots, M-1\}$ such that K is the smallest classroom where the class fits by the number of students
 - example
 - 4 classrooms with sizes 20, 20, 40, 80 corresponding to 0,1,2,3
 - class A wants a room of the size 20: $resource(A)=\{0,1,2,3\}$
 - class B wants a room of the size 40: $resource(B)=\{2,3\}$
 - class C wants a room of the size 80: $resource(C)=3$

Teacher represents a unary resource

- classes of one teacher cannot overlap
- all classes of each teacher are constrained by unary resource constraint
- classes are represented with their $start(A)$ and $p(A)$ variables

Curriculum represents a unary resource

- classes of one curriculum cannot overlap
- classes of one curriculum define one unary resource constraint
- classes are represented with their $start(A)$ and $p(A)$ variables

Class Timetabling: time and classrooms

Constraint:

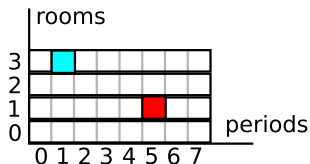
- Each time at most one course must be taught at any classroom

All classrooms together represents one unary resource

- all classes request this resource
- each class is encoded by activity with the starting time

$$\text{start-resource}(A) = \text{start}(A) + \text{resource}(A) * N$$

and duration $p(A)$



Class Timetabling: time and classrooms

Constraint:

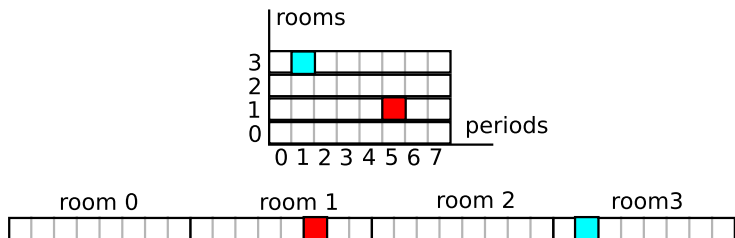
- Each time at most one course must be taught at any classroom

All classrooms together represents one unary resource

- all classes request this resource
- each class is encoded by activity with the starting time

$$\text{start-resource}(A) = \text{start}(A) + \text{resource}(A) * N$$

and duration $p(A)$



Class Timetabling: time and classrooms

Constraint:

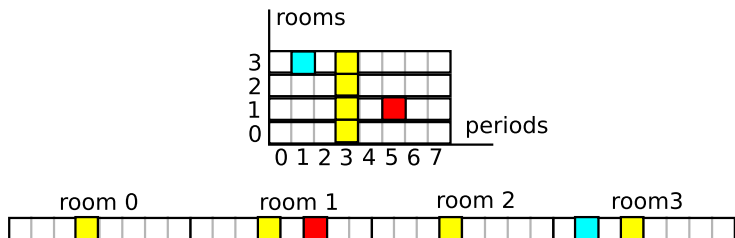
- Each time at most one course must be taught at any classroom

All classrooms together represents one unary resource

- all classes request this resource
- each class is encoded by activity with the starting time

$$\text{start-resource}(A) = \text{start}(A) + \text{resource}(A) * N$$

and duration $p(A)$



Employees Scheduling (rostering): problem

Create a one week schedule for employees working on shifts with

- several shift types
- minimal and maximal number of employees per shift
- minimal and maximal number of shift type per employee
- minimal and maximal number of working shifts per employee
- cost for each shift type to be paid to employee working on it
- minimal cost

Employees scheduling: example

- Employees Peter, Paul, Mary, Jane, Keith, Alex, Anne
- One week schedule, i.e. 7 days
- Shift types: M morning, A afternoon, N night
- Each shift – M:3 employees, A:2-3 employees, N:1-2 employees
- Each employee – M: 2-3 shifts, A:1-3 shifts, N: 1-2 shifts
- Working shifts: 4-6
- Cost – CostM=10, CostA=11, CostN=13

Schedule	Mo	Tue	Wed	Thu	...
Peter	M	M	N	-	
Paul	A	A	-	M	
Mary	N	-	M	A	
...					

Matrix of variables corresponding to shifts of employees:

- PeterMo, PeterTue, . . . , PeterSun,
PaulMo, PaulTue, . . . , PaulSun,
MaryMo, MaryTue, . . . , MarySun,
. . .

Domains:

- new shift type F (free) added to record free time shifts
- M, A, N, F corresponds to 1,2,3,4
- domain of variables corresponds to $\{1\dots 4\}$

Global constraint `global_cardinality(List, KeyCounts)`

- List: list of domain variables
- KeyCounts: list of Key-Count tuples with
 - Key: unique integer in the list of keys
 - Count: domain variable (or natural number)
- Each Key is contained in List with cardinality Count
- Example:
 - A in 1..3, B in 1..3, `global_cardinality([A,B], [1-N,2-2])`.
 - ⇒ A = 2, B = 2, N = 0

Minimal and maximal number constraints I.

Schedule	Mo	Tue	Wed	Thu	...
Peter	M	M	N	F	...
Paul	A	A	F	M	
Mary	N	F	M	A	
...	...				

Minimal and maximal number of employees per shift

- new domain variables representing these numbers
 $M1 = \{\text{Min}M1, \dots, \text{Max}M1\}$, $A1 = \{\text{Min}A1, \dots, \text{Max}A1\}$, $N1 = \{\text{Min}N1, \dots, \text{Max}N1\}$
- global cardinality constraint for each day
`global_cardinality([PeterMo,PaulMo,MaryMo,...], [1-M1,2-A1,3-N1])`

Minimal and maximal number constraints I.

Schedule	Mo	Tue	Wed	Thu	...
Peter	M	M	N	F	...
Paul	A	A	F	M	
Mary	N	F	M	A	
...	...				

Minimal and maximal number of employees per shift

- new domain variables representing these numbers
 $M1 = \{\text{Min}M1, \dots, \text{Max}M1\}$, $A1 = \{\text{Min}A1, \dots, \text{Max}M1\}$, $N1 = \{\text{Min}N1, \dots, \text{Max}N2\}$
- global cardinality constraint for each day
`global_cardinality([PeterMo,PaulMo,MaryMo,...], [1-M1,2-A1,3-N1])`

Minimal and maximal number of shift type per employee

- new domain variables representing these numbers
 $M2 = \{\text{Min}M2, \dots, \text{Max}M2\}$, $A2 = \{\text{Min}A2, \dots, \text{Max}M2\}$, $N2 = \{\text{Min}N2, \dots, \text{Max}N2\}$
- global cardinality constraint for each employee
`global_cardinality([PeterMo,PeterTue,...,PeterSun], [1-M2,2-A2,3-N2])`

Minimal and maximal number constraints II.

Minimal (MinW) and maximal number (MaxW)
of working shifts per employee

- $\text{MinF} = (\text{Days} - \text{MaxW})$... minimal number of free time shifts
- $\text{MaxF} = (\text{Days} - \text{MinW})$... maximal number of free time shifts
- example
 - $\text{Days}=7, \text{MaxW}=6 \Rightarrow \text{MinF}=1$
 - $\text{Days}=7, \text{MinW}=4 \Rightarrow \text{MaxF}=3$
- new domain variable $F = \{(\text{Days} - \text{MaxW}, \dots, \text{Days} - \text{MinW})\}$
- `global_cardinality([PeterMo, PeterTue, ..., PeterSun], [4-F])`

Minimal and maximal number constraints II.

Minimal (MinW) and maximal number (MaxW)
of working shifts per employee

- $\text{MinF} = (\text{Days} - \text{MaxW})$... minimal number of free time shifts
- $\text{MaxF} = (\text{Days} - \text{MinW})$... maximal number of free time shifts
- example
 - $\text{Days}=7, \text{MaxW}=6 \Rightarrow \text{MinF}=1$
 - $\text{Days}=7, \text{MinW}=4 \Rightarrow \text{MaxF}=3$
- new domain variable $F = \{(\text{Days} - \text{MaxW}, \dots, \text{Days} - \text{MinW})\}$
- `global_cardinality([PeterMo,PeterTue,...,PeterSun], [4-F])`
- can be added to "Minimal and maximal number of shift type per employee" global cardinality constraint
`global_cardinality([PeterMo,PeterTue,...,PeterSun], [1-M2,2-A2,3-N2,4-F])`

Cost minimization

Schedule	Mo	Tue	Wed	Thu	...
Peter	M	M	N	F	...
Paul	A	A	F	M	
Mary	N	F	M	A	
...	...				

- M_1, A_1, N_1 represent number of particular shifts on Monday
- All M_1 variables for particular days can be summarized into M
... and same for A_1 and A , N_1 and N
- Total schedule cost corresponds to
$$\text{Cost} = M * \text{Cost}_M + A * \text{Cost}_A + N * \text{Cost}_N$$

Cost minimization

Schedule	Mo	Tue	Wed	Thu	...
Peter	M	M	N	F	...
Paul	A	A	F	M	
Mary	N	F	M	A	
...	...				

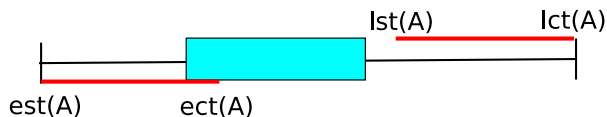
- M_1, A_1, N_1 represent number of particular shifts on Monday
- All M_1 variables for particular days can be summarized into M
... and same for A_1 and A , N_1 and N
- Total schedule cost corresponds to
$$\text{Cost} = M * \text{Cost}_M + A * \text{Cost}_A + N * \text{Cost}_N$$

Alternatively

- M_2, A_2, N_2 represent number of particular shifts for Peter
- All M_2 variables for all employees can be summarized into M
... and same for A_2 and A , N_2 and N

Constraint-based Scheduling: Propagation

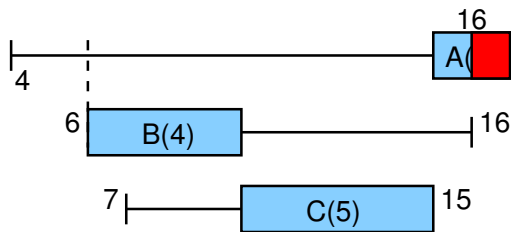
- 9 Unary resources
- 10 Cumulative resources
- 11 Alternative resources
- 12 Producible/consumable resources



- $est(A)$ earliest start time of activity A
 - $ect(A)$ earliest completion time of activity A
 - $lst(A)$ latest start time of activity A
 - $lct(A)$ latest completion time of activity A
-
- Ω is the set of activities
 - $p(\Omega) = \sum_{A \in \Omega} p(A)$
 - $est(\Omega) = \min\{est(A) \mid A \in \Omega\}$
 - $lct(\Omega) = \max\{lct(A) \mid A \in \Omega\}$

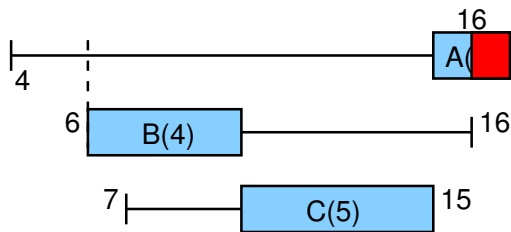
Edge finding: example

- What happens if activity A is not processed first?

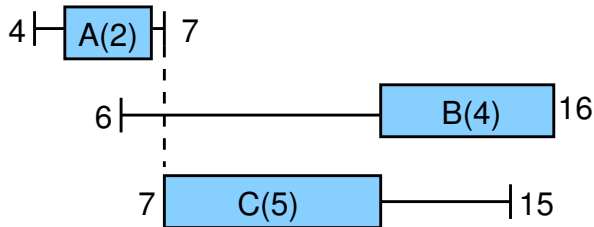


Edge finding: example

- What happens if activity A is not processed first?

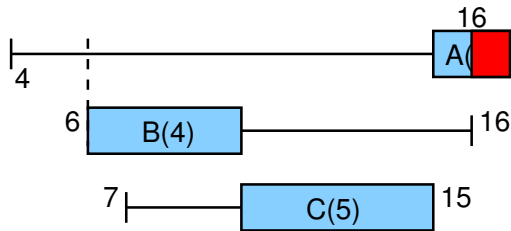


- Not enough time for A, B, and C and thus A must be first!



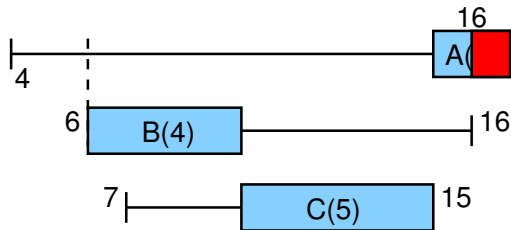
Edge finding: example with filtering rules

- $p(\Omega \cup \{A\}) > lct(\Omega \cup \{A\}) - est(\Omega) \Rightarrow A \ll \Omega$

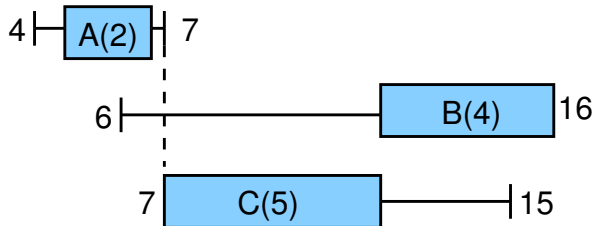


Edge finding: example with filtering rules

- $p(\Omega \cup \{A\}) > lct(\Omega \cup \{A\}) - est(\Omega) \Rightarrow A \ll \Omega$



- $A \ll \Omega \Rightarrow end(A) \leq \min\{lct(\Omega') - p(\Omega') \mid \Omega' \subseteq \Omega\}$



Edge finding: all filtering rules

- Edge-finding rules

- $p(\Omega \cup \{A\}) > lct(\Omega \cup \{A\}) - est(\Omega)$
 $\Rightarrow A \ll \Omega$

- $A \ll \Omega \Rightarrow$
 $end(A) \leq \min\{lct(\Omega') - p(\Omega') \mid \Omega' \subseteq \Omega\}$

- Edge-finding (symmetrical) rules

- $p(\Omega \cup \{A\}) > lct(\Omega) - est(\Omega \cup \{A\})$
 $\Rightarrow \Omega \ll A$

- $\Omega \ll A \Rightarrow$
 $start(A) \leq \max\{est(\Omega') + p(\Omega') \mid \Omega' \subseteq \Omega\}$

Edge finding: all filtering rules

- Edge-finding rules

- $p(\Omega \cup \{A\}) > lct(\Omega \cup \{A\}) - est(\Omega)$
 $\Rightarrow A \ll \Omega$
- $A \ll \Omega \Rightarrow$
 $end(A) \leq \min\{lct(\Omega') - p(\Omega') \mid \Omega' \subseteq \Omega\}$

- Edge-finding (symmetrical) rules

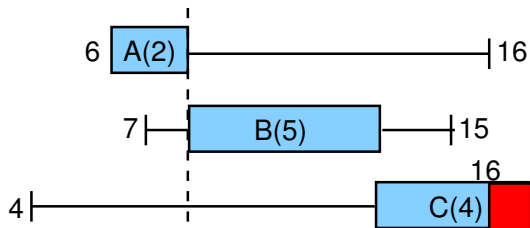
- $p(\Omega \cup \{A\}) > lct(\Omega) - est(\Omega \cup \{A\})$
 $\Rightarrow \Omega \ll A$
- $\Omega \ll A \Rightarrow$
 $start(A) \leq \max\{est(\Omega') + p(\Omega') \mid \Omega' \subseteq \Omega\}$

- In practice:

- there are $n \cdot 2^n$ pairs (A, Ω) to consider (too many!)
- instead of Ω use so called **task intervals** $[A, B]$
 $\{C \mid est(A) \leq est(C) \wedge lct(C) \leq lct(B)\}$
time complexity $O(n^3)$, frequently used incremental algorithm
- there are also $O(n^2)$ and $O(n \log n)$ algorithms

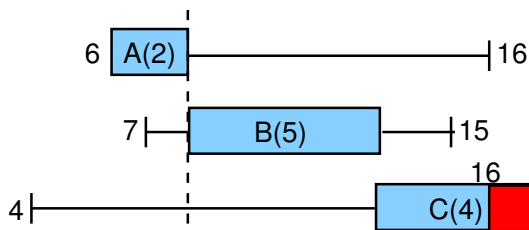
Not-first/not-last: example

- What happens if activity A is processed first?

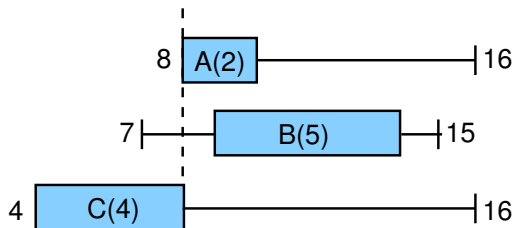


Not-first/not-last: example

- What happens if activity A is processed first?

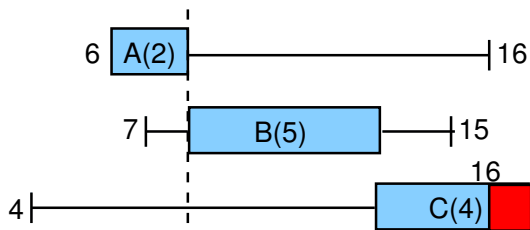


- Not enough time for B and C and thus A cannot be first!



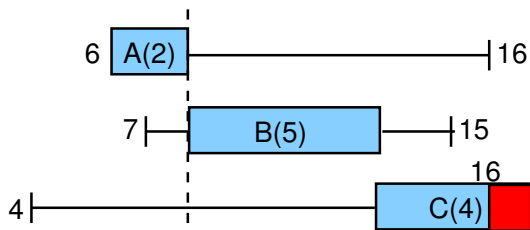
Not-first/not-last: example with filtering rules

- $p(\Omega \cup \{A\}) > lct(\Omega) - est(A) \Rightarrow \neg A \ll \Omega$

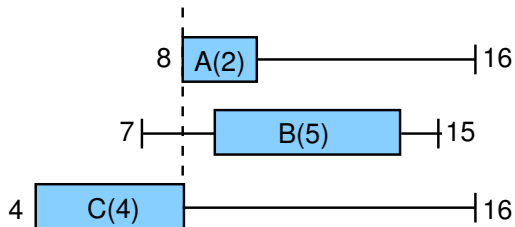


Not-first/not-last: example with filtering rules

- $p(\Omega \cup \{A\}) > lct(\Omega) - est(A) \Rightarrow \neg A \ll \Omega$



- $\neg A \ll \Omega \Rightarrow start(A) \geq \min\{ect(B) \mid B \in \Omega\}$



- Not-first rules:

- $p(\Omega \cup \{A\}) > lct(\Omega) - est(A)$
 $\Rightarrow \neg A \ll \Omega$
- $\neg A \ll \Omega$
 $\Rightarrow start(A) \geq \min\{ect(B) | B \in \Omega\}$

- Not-last (symmetrical) rules:

- $p(\Omega \cup \{A\}) > lct(A) - est(\Omega)$
 $\Rightarrow \neg \Omega \ll A$
- $\neg \Omega \ll A \Rightarrow$
 $end(A) \leq \max\{lst(B) | B \in \Omega\}$

Not-first/not-last: all filtering rules

- Not-first rules:

- $p(\Omega \cup \{A\}) > lct(\Omega) - est(A)$
 $\Rightarrow \neg A \ll \Omega$
- $\neg A \ll \Omega$
 $\Rightarrow start(A) \geq \min\{ect(B) | B \in \Omega\}$

- Not-last (symmetrical) rules:

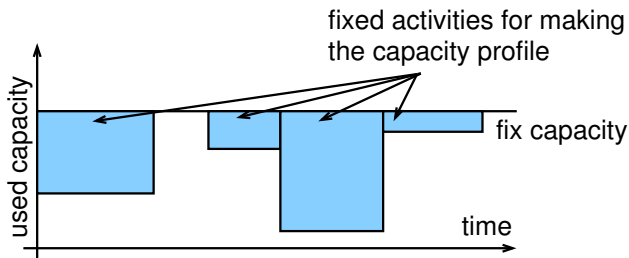
- $p(\Omega \cup \{A\}) > lct(A) - est(\Omega)$
 $\Rightarrow \neg \Omega \ll A$
- $\neg \Omega \ll A \Rightarrow$
 $end(A) \leq \max\{lst(B) | B \in \Omega\}$

- In practice:

- can be implemented with
time complexity $O(n^2)$ and space complexity $O(n)$

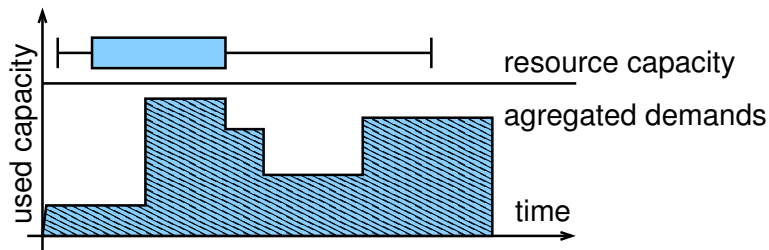
Cumulative resources

- Each **activity uses some capacity** of the resource $\text{cap}(A)$
- Activities can be **processed in parallel**, if a resource capacity is not exceeded
- Resource capacity **may vary in time**
 - modeled via fix capacity over time and fixed activities consuming the resource until the requested capacity level is reached



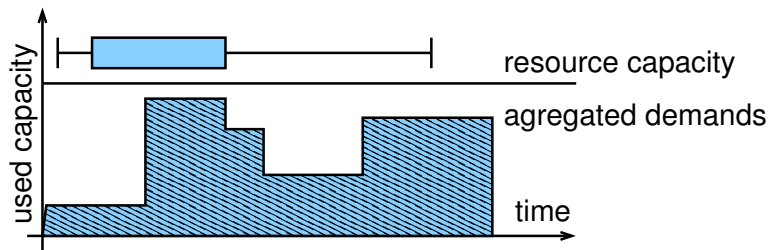
Aggregated demands

- Where is enough capacity for processing the activity?

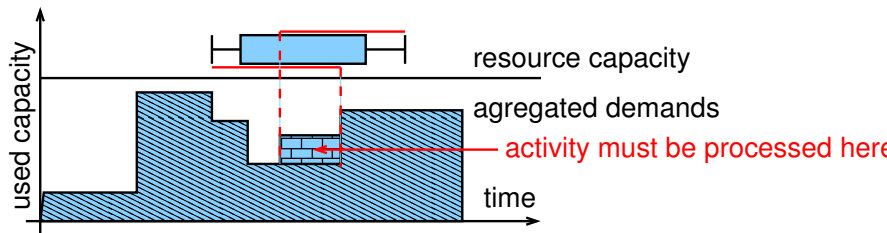


Aggregated demands

- Where is enough capacity for processing the activity?



- How aggregated demand is constructed?



Timetable constraint

- Discrete time is expected
- How to ensure that capacity is not exceeded at any time point?

$$\forall t \quad \sum_{start(A_i) \leq t \leq end(A_i)} cap(A_i) \leq MaxCapacity$$

- Discrete time is expected
- How to ensure that capacity is not exceeded at any time point?

$$\forall t \sum_{start(A_i) \leq t \leq end(A_i)} cap(A_i) \leq MaxCapacity$$

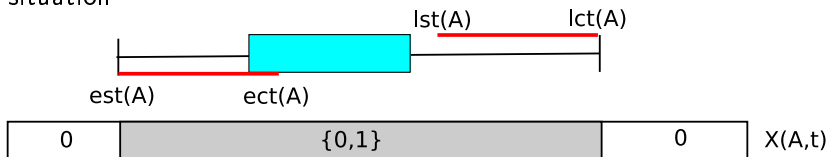
- **Timetable** for activity A is a set of Boolean domain variables $X(A, t)$ indicating whether A is processed in time t

$$\forall t \sum_{A_i} X(A_i, t) * cap(A_i) \leq MaxCapacity$$

$$\forall t, i \quad start(A_i) \leq t < end(A_i) \Leftrightarrow X(A_i, t)$$

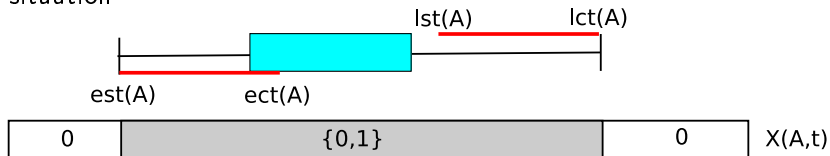
Timetable constraint: filtering example

Initial situation

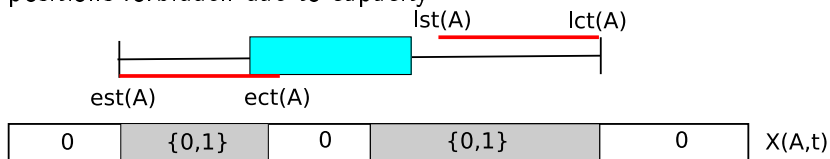


Timetable constraint: filtering example

Initial situation

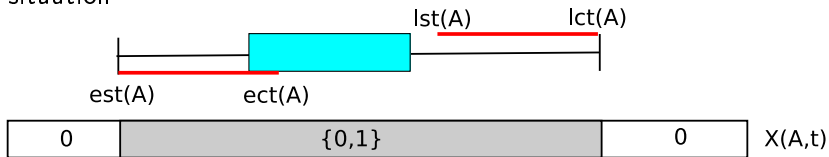


Some positions forbidden due to capacity

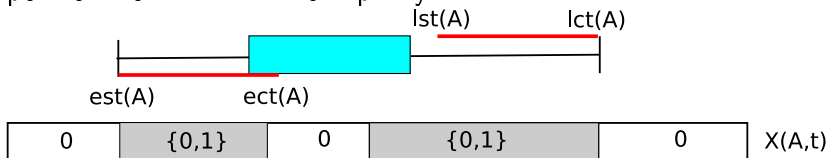


Timetable constraint: filtering example

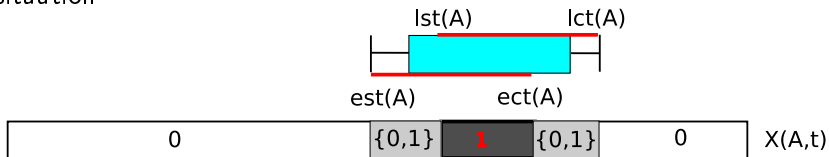
Initial situation



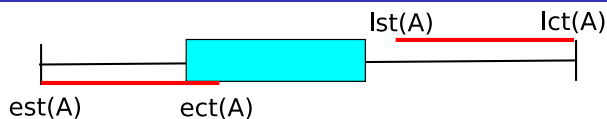
Some positions forbidden due to capacity



New situation



Timetable constraint: filtering rules



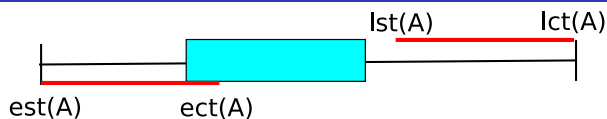
How to do filtering through the constraint

$$\forall t, i \text{ start}(A_i) \leq t < \text{end}(A_i) \Leftrightarrow X(A_i, t) ?$$

Problem: t serves as an index and as a variable

- $\text{start}(A) \geq \min\{t \mid 1 \in X(A, t)\}$
- $\text{end}(A) \leq 1 + \max\{t \mid 1 \in X(A, t)\}$

Timetable constraint: filtering rules



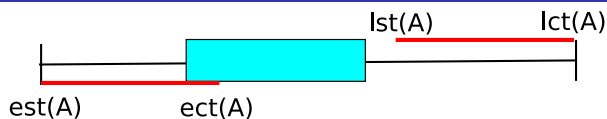
How to do filtering through the constraint

$$\forall t, i \text{ start}(A_i) \leq t < \text{end}(A_i) \Leftrightarrow X(A_i, t) ?$$

Problem: t serves as an index and as a variable

- $\text{start}(A) \geq \min\{t \mid 1 \in X(A, t)\}$
- $\text{end}(A) \leq 1 + \max\{t \mid 1 \in X(A, t)\}$
- $X(A, t) = 0 \wedge t < \text{ect}(A) \Rightarrow \text{start}(A) > t$
- $X(A, t) = 0 \wedge \text{lst}(A) \leq t \Rightarrow \text{end}(A) \leq t$

Timetable constraint: filtering rules



How to do filtering through the constraint

$$\forall t, i \text{ start}(A_i) \leq t < \text{end}(A_i) \Leftrightarrow X(A_i, t) ?$$

Problem: t serves as an index and as a variable

- $\text{start}(A) \geq \min\{t \mid 1 \in X(A, t)\}$
- $\text{end}(A) \leq 1 + \max\{t \mid 1 \in X(A, t)\}$
- $X(A, t) = 0 \wedge t < \text{ect}(A) \Rightarrow \text{start}(A) > t$
- $X(A, t) = 0 \wedge \text{lst}(A) \leq t \Rightarrow \text{end}(A) \leq t$
- $\text{lst}(A) \leq t \wedge t < \text{ect}(A) \Rightarrow X(A, t) = 1$

How to model alternative resources for a given activity?

Use a **duplicate activity** for each resource

- duplicate activity participates in a respective resource constraint but does not restrict other activities there
 - "failure" means removing the resource from the domain of variable $\text{resource}(A)$
 - deleting the resource from the domain of variable $\text{resource}(A)$ means "deleting" the respective duplicate activity

How to model alternative resources for a given activity?

Use a **duplicate activity** for each resource

- duplicate activity participates in a respective resource constraint but does not restrict other activities there
 - "failure" means removing the resource from the domain of variable resource(A)
 - deleting the resource from the domain of variable resource(A) means "deleting" the respective duplicate activity
- original activity participates in precedence constraints (e.g., within a job)
- restricted times of duplicate activities are propagated to the original activity and vice versa

Let A_u be the duplicate activity of A allocated to resource $u \in \text{resource}(A)$

- $u \in \text{resource}(A) \Rightarrow \text{start}(A) \leq \text{start}(A_u)$
- $u \in \text{resource}(A) \Rightarrow \text{end}(A_u) \leq \text{end}(A)$
- $\text{start}(A) \geq \min\{\text{start}(A_u) : u \in \text{resource}(A)\}$
- $\text{end}(A) \leq \max\{\text{end}(A_u) : u \in \text{resource}(A)\}$
- failure related to $A_u \Rightarrow \text{resource}(A) \setminus \{u\}$

Disjunctive constraint

- unary resources, non-preemptive activities
- extensions: preemptive activities, cumulative resources

Edge finding

- unary resources, non-preemptive activities
- extensions: preemptive activities, cumulative resources

Not-first/not-last

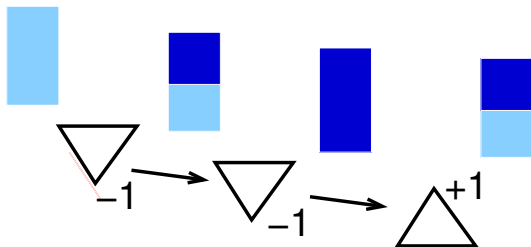
- unary resources, non-preemptive activities
- extensions: cumulative resources

Timetable constraint

- cumulative resource, non-preemptive activities
- extensions: preemptive activities

Producible/consumable resources (revision)

- Resource = reservoir
- Activity consumes some quantity of the resource $cap(A) < 0$ or activity produces some quantity of the resource $cap(A) > 0$
- Minimal capacity is requested (consumption) and maximal capacity cannot be exceeded (production)



- Cumulative resource can be seen as a special case of reservoir
 - each activity consumes $cap(A)$ at its start and produces $cap(A)$ at its end

Relative ordering

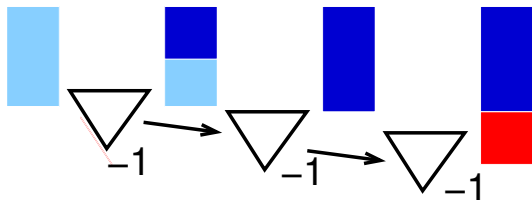
When time is relative (ordering of activities)

- then edge-finding and aggregated demands deduce nothing

We can still use information about

ordering of events and resource production/consumption!

Example: reservoir with events consuming and supplying items



Relative ordering

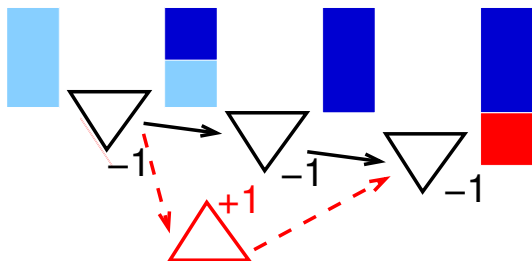
When time is relative (ordering of activities)

- then edge-finding and aggregated demands deduce nothing

We can still use information about

ordering of events and resource production/consumption!

Example: reservoir with events consuming and supplying items



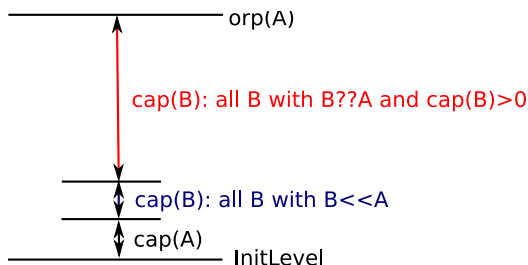
We can deduce needs for additional activity supplying items

Optimistic resource profile (*orp*)

- *orp*: maximal possible level of the resource when A happens
- Activities **known to be before A** are assumed together with the production activities events that **can be before A**

$$\text{orp}(A) = \text{InitLevel} + \text{cap}(A) + \sum_{B \ll A} \text{cap}(B) + \sum_{B ?? A \ \& \ \text{cap}(B) > 0} \text{cap}(B)$$

- Example:

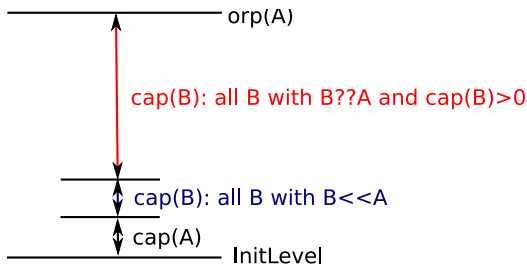


B ?? A means that order of A and B is unknown yet

orp filtering rules I.

$orp(A) < MinLevel \Rightarrow fail$

- despite the fact that all production is planned before A, the minimal required level in the resource is not reached



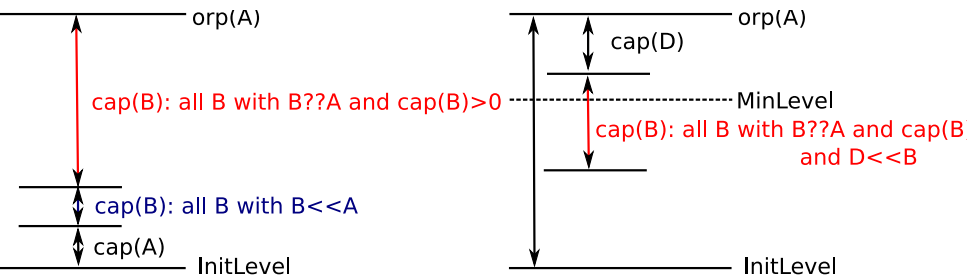
$$orp(A) = InitLevel + cap(A) + \sum_{B < A} cap(B) + \sum_{B ?? A \ \& \ cap(B) > 0} cap(B)$$

orp filtering rules II.

$$\text{orp}(A) - \text{cap}(D) - \sum_{D \ll B \ \& \ B??A \ \& \ \text{cap}(B) > 0} \text{cap}(B) < \text{MinLevel} \\ \Rightarrow D \ll A$$

Think about the time when D happens

- for any such D that $D??A$ and $\text{cap}(D) > 0$
if production in D is planned after A and the minimal required level in the resource is not reached then D must be before A

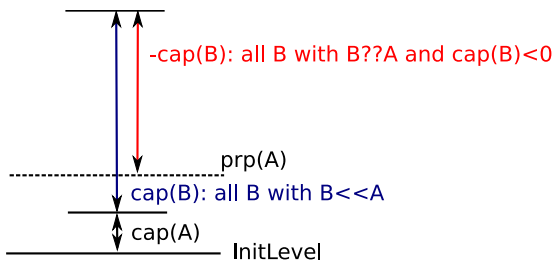


Pesimistic resource profile (*prp*)

- *prp*: minimal possible level of the resource when A happens
- Activities **known to be before A** are assumed together with the consumption activities that **can be before A**

$$prp(A) = InitLevel + cap(A) + \sum_{B \ll A} cap(B) + \sum_{B \text{??} A \ \& \ cap(B) < 0} cap(B)$$

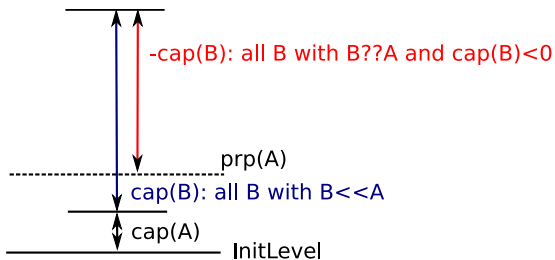
- Example:



prp filtering rules I.

$prp(A) > MaxLevel \Rightarrow fail$

- despite the fact that all consumption is planned before A, the maximal required level (resource capacity) in the resource is exceeded



$$prp(A) = InitLevel + cap(A) + \sum_{B \ll A} cap(B) + \sum_{B \text{ ?? } A \text{ \& } cap(B) < 0} cap(B)$$

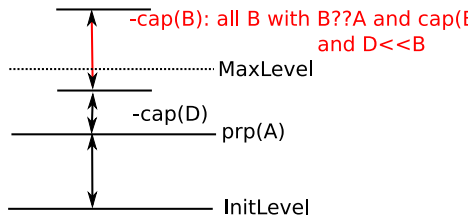
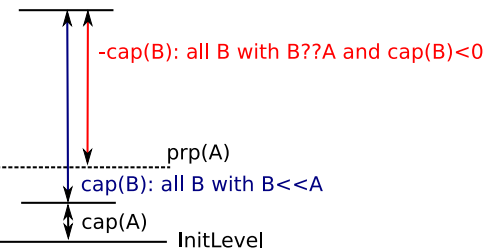
prp filtering rules II.

$$prp(A) - cap(D) - \sum_{D \ll B \ \& \ B \ ?? \ A \ \& \ cap(B) < 0} cap(B) > MaxLevel$$

$$\Rightarrow D \ll A$$

Think about the time just before D happens

- for any D such that $D \ ?? \ A$ and $cap(D) < 0$
- if consumption in D is planned after A and the maximal required level in the resource is exceeded then D must be before A



13 Search (revision)

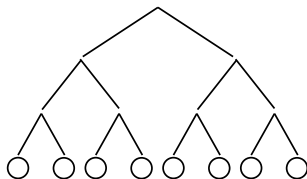
14 Search Strategies for Scheduling

Search (revision)

Constraint propagation techniques are (usually) incomplete
⇒ search algorithm is needed to solve the "rest"

Labeling

- depth-first search (DFS/BT)
 - assign value to variable
 - propagate = make the problem locally consistent
 - backtrack in case of failure

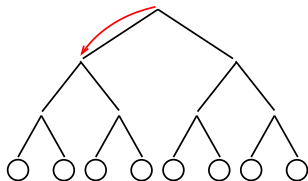


Search (revision)

Constraint propagation techniques are (usually) incomplete
⇒ search algorithm is needed to solve the "rest"

Labeling

- depth-first search (DFS/BT)
 - assign value to variable
 - propagate = make the problem locally consistent
 - backtrack in case of failure

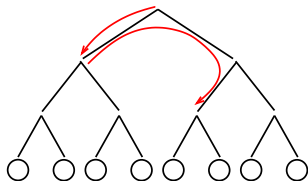


Search (revision)

Constraint propagation techniques are (usually) incomplete
⇒ search algorithm is needed to solve the "rest"

Labeling

- depth-first search (DFS/BT)
 - assign value to variable
 - propagate = make the problem locally consistent
 - backtrack in case of failure

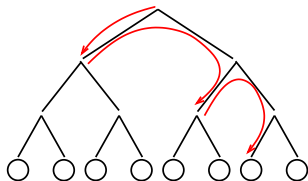


Search (revision)

Constraint propagation techniques are (usually) incomplete
⇒ search algorithm is needed to solve the "rest"

Labeling

- depth-first search (DFS/BT)
 - assign value to variable
 - propagate = make the problem locally consistent
 - backtrack in case of failure

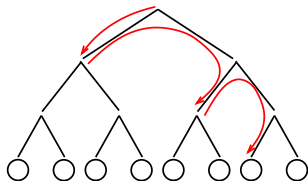


Search (revision)

Constraint propagation techniques are (usually) incomplete
⇒ search algorithm is needed to solve the "rest"

Labeling

- depth-first search (DFS/BT)
 - assign value to variable
 - propagate = make the problem locally consistent
 - backtrack in case of failure



- $X \text{ in } 1..5 \quad \equiv \quad X=1 \vee X=2 \vee X=3 \vee X=4 \vee X=5$

Generally search algorithm solves remaining disjunctions

- $X=1 \vee X \neq 1$ standard assignment
- $X < 3 \vee X \geq 3$ domain splitting
- $X < Y \vee X \geq Y$ variable ordering (scheduling: tasks ordering)

Search: variable ordering (revision)

Which variable should be assigned first?

First-fail principle

- prefer variable with the hardest assignment
- for example variable with the smallest domain:
domain can be emptied easily
- or variable with the most constraints:
assignment of other variables constrain and
make the domain smaller easily

Variable ordering defines **shape of the search tree**

- selection of variable with small domain size:
small branching on this level more options left for later
- selection of variable with large domain size:
large branching on this level less options left for later

What value should be chosen first?

Succeed-first principle

- prefers values with probably belongs to the solution
- for example the values with most supports in neighbouring variables
- this heuristic is usually problem specific

Value ordering defines the order how the branches are explored

Branching=resolving disjunctions

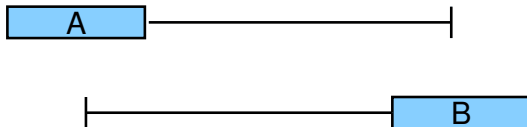
Traditional scheduling approaches

- take the **critical decisions first**
 - resolve bottlenecks, . . .
 - defines the **shape** of the search tree
 - recall the **first-fail** principle
- prefer an **alternative leaving more flexibility**
 - defines **order** of branches to be explored
 - recall **succeed-first** principle

How to describe criticality and flexibility formally?

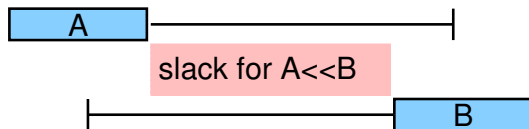
Slack

- **Slack** is a formal description of flexibility
- Slack for a **given order of two activities**
"free time for shifting the activities"



Slack

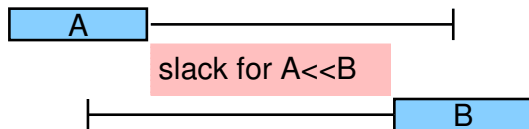
- **Slack** is a formal description of flexibility
- Slack for a given order of two activities
"free time for shifting the activities"



$$\text{slack}(A \ll B) = \max(\text{end}(B)) - \min(\text{start}(A)) - p(A) - p(B)$$

Slack

- **Slack** is a formal description of flexibility
- Slack for a **given order of two activities**
"free time for shifting the activities"

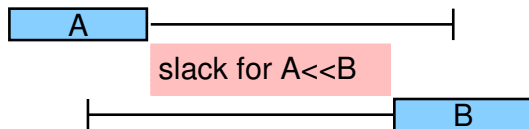


$$\text{slack}(A \ll B) = \max(\text{end}(B)) - \min(\text{start}(A)) - p(A) - p(B)$$

- Slack for **two activities** (without any ordering)
 $\text{slack}(\{A, B\}) = \max(\text{slack}(A \ll B), \text{slack}(B \ll A))$

Slack

- **Slack** is a formal description of flexibility
- Slack for a **given order of two activities**
"free time for shifting the activities"



$$\text{slack}(A \ll B) = \max(\text{end}(B)) - \min(\text{start}(A)) - p(A) - p(B)$$

- Slack for **two activities** (without any ordering)
 $\text{slack}(\{A, B\}) = \max(\text{slack}(A \ll B), \text{slack}(B \ll A))$
- Slack for a **group of activities**
 $\text{slack}(\Omega) = \max(\text{end}(\Omega)) - \min(\text{start}(\Omega)) - p(\Omega)$

Order branching

$$A \ll B \quad \vee \quad \neg A \ll B$$

What activities A, B should be ordered first?

- the most critical pair (first-fail)
- the pair with the minimal slack($\{A, B\}$)

Order branching

$$A \ll B \quad \vee \quad \neg A \ll B$$

What activities A, B should be ordered first?

- the most critical pair (first-fail)
- the pair with the minimal slack($\{A, B\}$)

What order of activities A and B should be selected?

- the most flexible order
- the order with with the maximal slack($A??B$)

$O(n^2)$ choice points

First/last branching

$$(A \ll \Omega \vee \neg A \ll \Omega) \quad \vee \quad (\Omega \ll A \vee \neg \Omega \ll A)$$

Should we look for the first or last activity?

- look to the set of possible candidates for first activity and to the set of possible candidates for last activities
- select a **smaller set** from these (first-fail)
 - smaller number of candidates means that it is harder to find a suitable candidate

First/last branching

$$(A \ll \Omega \vee \neg A \ll \Omega) \quad \vee \quad (\Omega \ll A \vee \neg \Omega \ll A)$$

Should we look for the first or last activity?

- look to the set of possible candidates for first activity and to the set of possible candidates for last activities
- select a **smaller set** from these (first-fail)
 - smaller number of candidates means that it is harder to find a suitable candidate

What activity should be selected?

- if first activity is being selected then the activity with the **smallest** $\min(\text{start}(A))$ is preferred
- if last activity is being selected then the activity with the **largest** $\max(\text{end}(A))$ is preferred

$O(n)$ choice points

Resource slack is defined as
a slack of the set of activities processed by the resource

How to use a resource slack?

- choosing a resource on which **the activities will be ordered** first
 - resource with a minimal slack (**bottleneck**) preferred
- choosing a resource on which the **activity will be allocated**
 - resource with a maximal slack (**flexibility**) preferred