CONSTRAINT PROCESSING
FOR PLANNING & SCHEDULING

Roman Barták
Charles University, Prague (CZ)

roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

WHAT AND WHY?

- What is the topic of the tutorial?
  - constraint satisfaction techniques useful for P&S
- What is constraint satisfaction?
  - technology for modeling and solving combinatorial optimization problems
- Why should one look at constraint satisfaction?
  - powerful solving technology
  - planning and scheduling are coming together and constraint satisfaction may serve as a bridge between them
- Why should one understand insides of constraint satisfaction algorithms?
  - better exploitation of the technology
  - design of better (solvable) constraint models

TUTORIAL OUTLINE

- Constraint satisfaction in a nutshell
  - domain filtering and local consistencies
  - search techniques
- Constraints for planning
  - constraint models
  - temporal reasoning
- Constraints for scheduling
  - a base constraint model
  - resource constraints
  - branching schemes

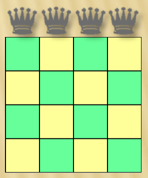
CONSTRAINT SATISFACTION IN A NUTSHELL

STARTING SIMPLE

CONSTRAINT SATISFACTION

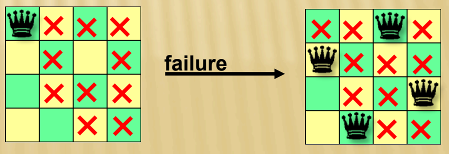
Modeling (problem formulation)

- N queens problem
- **decision variables** for positions of queens in rows
  r(i) in {1,...,N}
- **constraints** describing (non-)conflicts
  $\forall i \neq j \quad r(i) \neq r(j)$ & $|i-j| \neq |r(i)-r(j)|$



Search and **inference** (propagation)

- **backtracking** (assign values and return upon failure)
- infer consequences of decisions
  via maintaining **consistency**
  of constraints



CONSTRAINT SATISFACTION

based on **declarative problem description** via:

- **variables with domains** (sets of possible values)
  describe **decision points** of the problem with possible **options** for the decisions
  e.g. the start time of activity with time windows
- **constraints** restricting combinations of values,
  describe arbitrary **relations** over the set of variables
  e.g. end(A) < start(B)

A **feasible solution** to a constraint satisfaction problem is a complete assignment of variables satisfying all the constraints.

An **optimal solution** to a CSP is a feasible solution minimizing/maximizing a given objective function.

CONSTRAINT SATISFACTION

CONSISTENCY TECHNIQUES

DOMAIN FILTERING

- Example:
  - $D_a = \{1,2\}$, $D_b = \{1,2,3\}$
  - $a < b$
  - ⇘ Value 1 can be safely removed from D_b.
- Constraints are used **actively to remove inconsistencies** from the problem.
  - **inconsistency** = a value that cannot be in any solution
- This is realized via a procedure FILTER that is attached to each constraint.

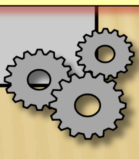
FILTER

- Removes all values violating a given constraint.
  - for each value we need to find values (**support**) in domains of other variables such that the tuple satisfies the constraint
  - filter for constraints specified using a table of compatible tuples

```
procedure c.FILTER(OrigD)
    NewD ← OrigD
    for each X in scope(c) do
        for each v in $NewD_X$ do
            if there is no support for v in c then
                $NewD_X$ ← $NewD_X$ - {v}
        end for
    end for
    return NewD
end FILTER
```

Constraint scope is a set of constrained variables

Support is a tuple of values from variables' domains that satisfies the constraint



ARC-CONSISTENCY

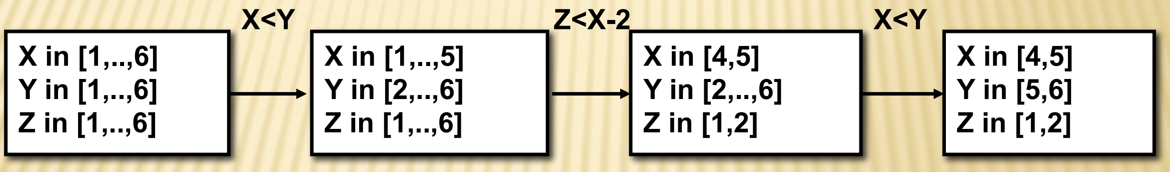
- We say that a **constraint** is **arc consistent** (AC) if for any value of the variable in the constraint there exists a value (a **support**) for the other variable(s) in such a way that the constraint is satisfied (we say that the value is supported).
  Unsupported values are filtered out of the domain.

- A **CSP** is **arc consistent** if all the constraints are arc consistent.

MAKING PROBLEMS AC

- How to **establish arc consistency** in a CSP?
- Every constraint must be made AC!

Example: X in [1,..,6], Y in [1,..,6], Z in [1,..,6], X<Y, Z<X-2



➯ Filtering through every constraint just once is not enough!

- Filtering must be repeated until any domain is changed (AC-1).

ALGORITHM AC-3

- Uses a **queue of constraints** that should be checked for AC.
- When a domain of variable is changed, only the constraints over this variable are added back to the queue for filtering.

```
procedure AC-3(V,D,C)
    $Q \leftarrow C$
    while non-empty Q do
        select c from Q
        $D' \leftarrow$ c.FILTER(D)
        if any domain in D' is empty then return (fail,D')
        $Q \leftarrow Q \cup \{c' \in C \mid \exists x \in var(c')\ D'_x \neq D_x\} - \{c\}$
        $D \leftarrow D'$
    end while
    return (true,D)
end AC-3
```

AC IN PRACTICE

- Uses a **queue of variables** with changed domains.
  - Users may specify for each constraint when the filtering should be done depending on the domain change.
- The algorithm is sometimes called **AC-8**.

```
procedure AC-8(V,D,C)
    Q ← V
    while non-empty Q do
        select v from Q
        for c∈C such that v is constrained by c do
            D' ← c.FILTER(D)
            if any domain in D' is empty then return (fail,D')
            Q ← Q ∪ {u∈V | $D'_u≠D_u$}
            D ← D'
        end for
    end while
    return (true,D)
end AC-8
```

ARC-B-CONSISTENCY

- Sometimes, making the problem arc-consistent is costly (for example, when domains of variables are large).
- In such a case, a weaker form of arc-consistency might be useful.
- We say that a constraint is **arc-b-consistent** (bound consistent) if for any bound values of the variable in the constraint there exists a value for the other variable(s) in such a way that the constraint is satisfied.
  - a bound value is either a minimum or a maximum value in domain
  - domain of the variable can be represented as an interval
  - for some constraints (like x<y) it is equivalent to AC

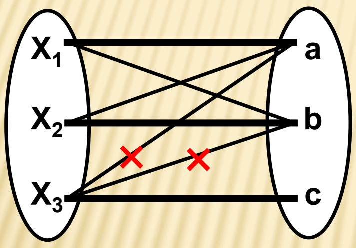
```
procedure (x<y).FILTER(OrigD)
    $NewD_X ← OrigD_X ∩ (inf .. max(OrigD_Y)-1)$
    $NewD_Y ← OrigD_Y ∩ (min(OrigD_X)+1 .. sup)$
    $∀Z≠X,Y\ NewD_Z ← OrigD_Z$
    return NewD
end FILTER
```

PITFALLS OF AC

- Disjunctive constraints
  - A, B in $\{1,\dots,10\}$, $A = 1 \vee A = 2$
  - no filtering (whenever $A \neq 1$ then deduce $A = 2$ and vice versa)
  - ↳ constructive disjunction
- Detection of inconsistency
  - A, B, C in $\{1,\dots,10000000\}$, $A < B$, $B < C$, $C < A$
  - long filtering (4 seconds)
  - ↳ a different model
- Weak filtering
  - A, B in $\{1,2\}$, C in $\{1,2,3\}$, $A \neq B$, $A \neq C$, $B \neq C$
  - weak filtering (it is arc-consistent)
  - ↳ global constraints

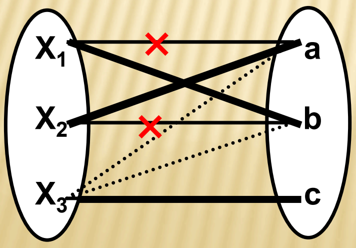
GLOBAL CONSTRAINTS (INSIDE ALL-DIFFERENT)

- a set of binary inequality constraints among all variables
  $X_1 \neq X_2, X_1 \neq X_3, \dots, X_{k-1} \neq X_k$
- $\text{all\_different}(\{X_1,\dots,X_k\}) = \{(d_1,\dots,d_k) \mid \forall i\ d_i \in D_i\ \&\ \forall i \neq j\ d_i \neq d_j\}$
- better pruning based on **matching theory over bipartite graphs**



Initialization:

1. compute maximum matching
2. remove all edges that do not belong to any maximum matching



Propagation of deletions ($X_1 \neq a$):

1. remove discharged edges
2. compute new maximum matching
3. remove all edges that do not belong to any maximum matching

META CONSISTENCY

Can we **strengthen any filtering technique**?

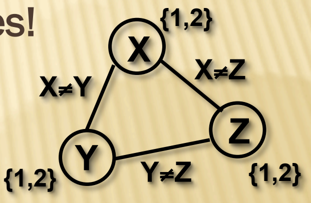
YES! Let us assign a value and make the rest of the problem consistent.

- **singleton consistency** (Prosser et al., 2000)
  - try each value in the domain
- **shaving**
  - try only the bound values
- **constructive disjunction**
  - propagate each constraint in disjunction separately
  - make a union of obtained restricted domains

PATH CONSISTENCY

Arc consistency does not detect all inconsistencies!

Let us look at several constraints together!



- The **path** $(V_0,V_1,\ldots, V_m)$ is **path consistent** iff for every pair of values $x \in D_0$ a $y \in D_m$ satisfying all the binary constraints on V_0,V_m there exists an assignment of variables $V_1,\ldots,V_{m-1}$ such that all the binary constraints between the neighboring variables V_i,V_{i+1} are satisfied.
- **CSP is path consistent** iff every path is consistent.

Some notes:

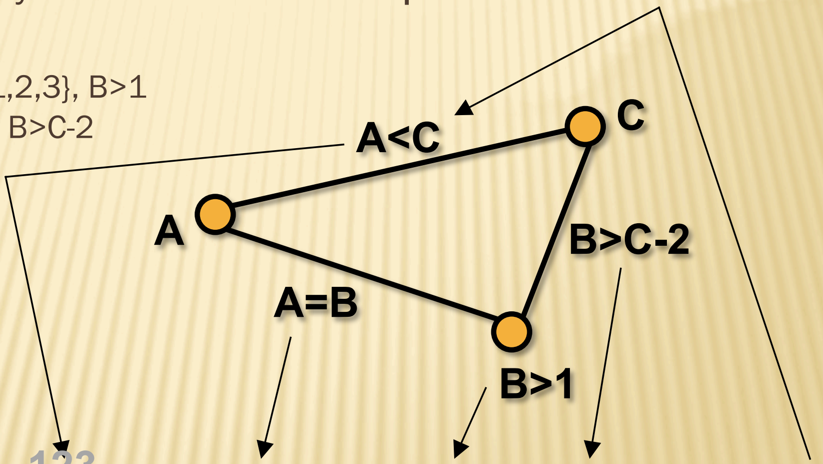
- only the **constraints between the neighboring variables** must be satisfied
- it is enough to explore **paths of length 2** (Montanary, 1974)

PATH REVISION

Constraints **represented extensionally** via matrixes.
Path consistency is realized via **matrix operations**

Example:

- A,B,C in {1,2,3}, B>1
- A<C, A=B, B>C-2



$$\begin{array}{c|c} & 123 \\ \hline 1 & 011 \\ 2 & 001 \\ 3 & 000 \end{array} \;\&\; \begin{pmatrix} 100 \\ 010 \\ 001 \end{pmatrix} * \begin{pmatrix} 000 \\ 010 \\ 001 \end{pmatrix} * \begin{pmatrix} 110 \\ 111 \\ 111 \end{pmatrix} = \begin{pmatrix} 000 \\ 001 \\ 000 \end{pmatrix}$$

CONSTRAINT SATISFACTION

SEARCH TECHNIQUES

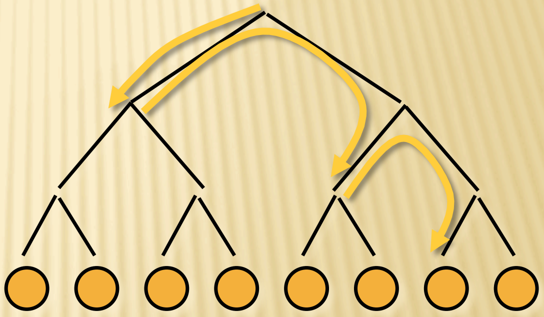
SEARCH / LABELING

Inference techniques are (usually) incomplete.

↳ We need a **search algorithm** to resolve the rest!

Labeling

- depth-first search
  - assign a value to the variable
  - propagate = make the problem locally consistent
  - backtrack upon failure



- X in 1..5 ≈ X=1 v X=2 v X=3 v X=4 v X=5 (**enumeration**)

In general, search algorithm resolves remaining disjunctions!

- X=1 v X≠1 (**step labeling**)
- X<3 v X≥3 (**domain splitting**)
- X<Y v X≥Y (variable ordering)

LABELING SKELETON

- Search is combined with filtering techniques that prune the search space.
- Look-ahead technique (MAC)

```
procedure labeling(V,D,C)
    if all variables from V are assigned then return V
    select not-yet assigned variable x from V
    for each value v from D_x do
        (TestOK,D') ← consistent(V,D,C∪{x=v})
        if TestOK=true then R ← labeling(V,D',C)
            if R ≠ fail then return R
    end for
    return fail
end labeling
```

BRANCHING SCHEMES

- Which variable should be assigned first?
  - **fail-first principle**
    - prefer the variable whose instantiation will lead to a failure with the highest probability
    - variables with the smallest domain first (**dom**)
    - the most constrained variables first (**deg**)
  - defines the **shape of the search tree**
- Which value should be tried first?
  - **succeed-first principle**
    - prefer the values that might belong to the solution with the highest probability
    - values with more supports in other variables
    - usually problem dependent
  - defines the **order of branches** to be explored

HEURISTICS IN SEARCH

Observation 1:

The **search space** for real-life problems is so **huge** that it cannot be fully explored.

- **Heuristics** - a guide of search
  - **value ordering heuristics** recommend a value for assignment
  - quite often lead to a solution
- What to do upon a **failure of the heuristic**?
  - BT cares about the end of search (a bottom part of the search tree)
    so it rather repairs later assignments than the earliest ones
    thus BT assumes that the heuristic guides it well in the top part

Observation 2:

The **heuristics** are **less reliable in the earlier parts** of the search tree (as search proceeds, more information is available).

Observation 3:

The number of **heuristic violations** is usually **small**.

DISCREPANCIES

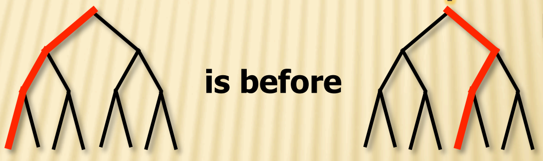
Discrepancy

= the heuristic is not followed

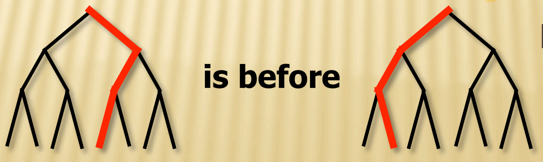
Basic principles of discrepancy search:

change the order of branches to be explored

+ prefer branches with **less discrepancies**

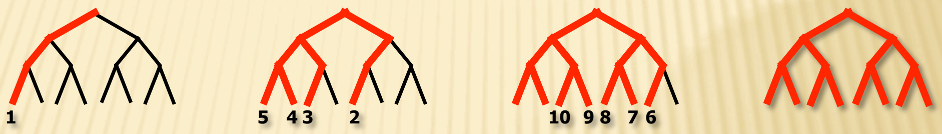


+ prefer branches with **earlier discrepancies**

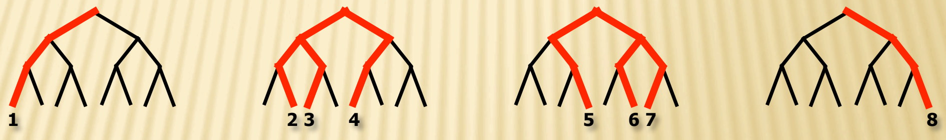


DISCREPANCY SEARCH

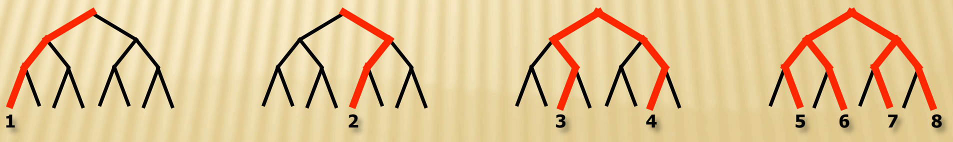
- Limited Discrepancy Search (Harvey & Ginsberg, 1995)
  + restricts a maximal number of discrepancies in the iteration



- Improved LDS (Korf, 1996)
  + restricts a given number of discrepancies in the iteration



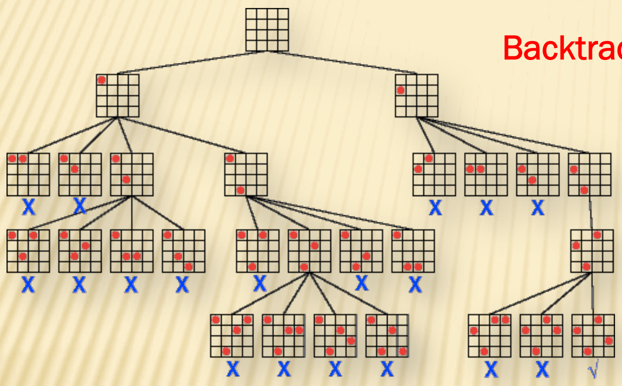
- Depth-bounded Discrepancy Search (Walsh, 1997)
  + restricts discrepancies till a given depth in the iteration



- …

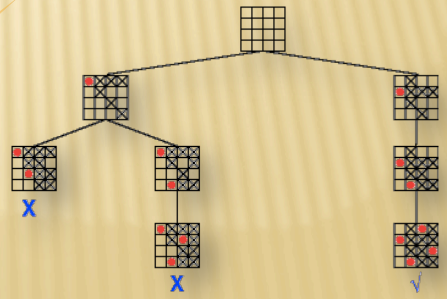
* **heuristic = go left**

4-QUEENS PROBLEM
CP IS NOT (ONLY) SEARCH!

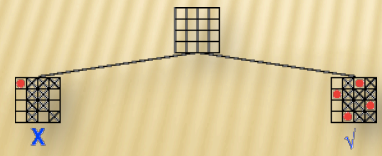


Backtracking is not very good
19 attempts

Forward checking is better
3 attempts



And the winner is Look Ahead
2 attempts



CONSTRAINT SATISFACTION
EXTENSIONS

OPTIMIZATION PROBLEMS

- **Constraint optimization problem** (COP)
  = CSP + objective function
- **Objective function** is encoded in a constraint.
  - V = objective(Xs)
  - heuristics for bound-estimate encoded in the filter

```
Branch and bound technique
    find a complete assignment (defines a new
          bound)
    store the assignment
    update bound (post the constraint that restricts
          the objective function to be better than a
          given bound which causes failure)
    continue in search (until total failure)
    restore the best assignment
```

SOFT PROBLEMS

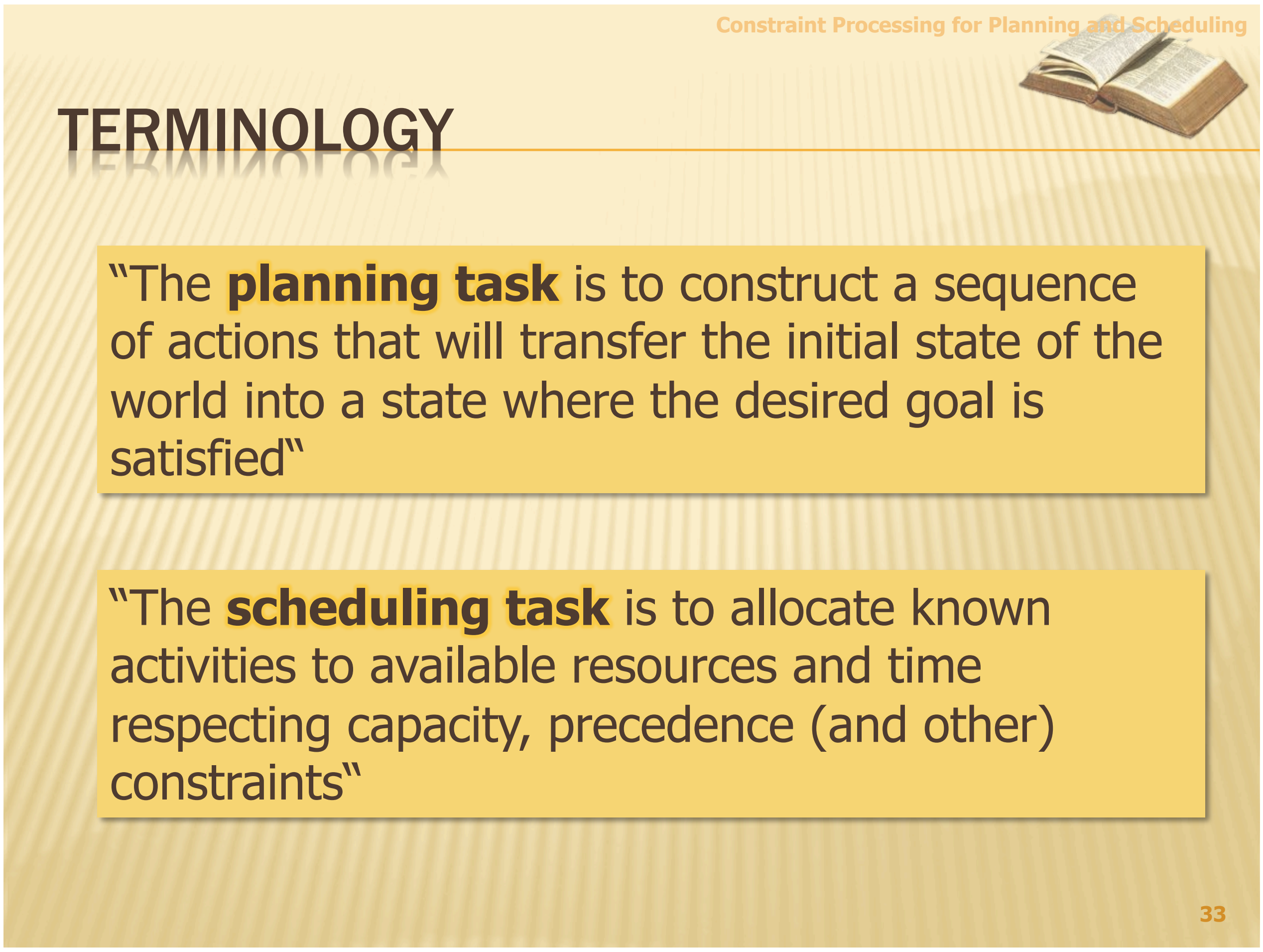
- **Hard constraints** express restrictions.
- **Soft constraints** express preferences.
- Maximizing the number of satisfied soft constraints
- Can be solved via **constraint optimization**
  - Soft constraints are encoded into an objective function
- Special frameworks for soft constraints
  - **Constraint hierarchies** (Borning et al., 1987)
    - symbolic preferences assigned to constraints
  - **Semiring-based CSP** (Bistarelli, Montanary, and Rossi, 1997)
    - semiring values assigned to tuples (how well/badly a tuple satisfies the constraint)
    - soft constraint propagation

DYNAMIC PROBLEMS

- **Internal dynamics** (Mittal & Falkenhainer, 1990)
  - planning, configuration
  - variables can be active or inactive, only active variables are instantiated
  - **activation (conditional) constraints**
    - cond(x_1,..., x_n) → activate(x_j)
  - solved like a standard CSP (a special value in the domain to denote inactive variables)

- **External dynamics** (Dechter & Dechter, 1988)
  - on-line systems
  - **sequence of static CSPs**, where each CSP is a result of the addition or retraction of a constraint in the preceding problem
  - Solving techniques:
    - reusing solutions
    - **maintaining dynamic consistency** (DnAC-4, DnAC-6, AC|DC).

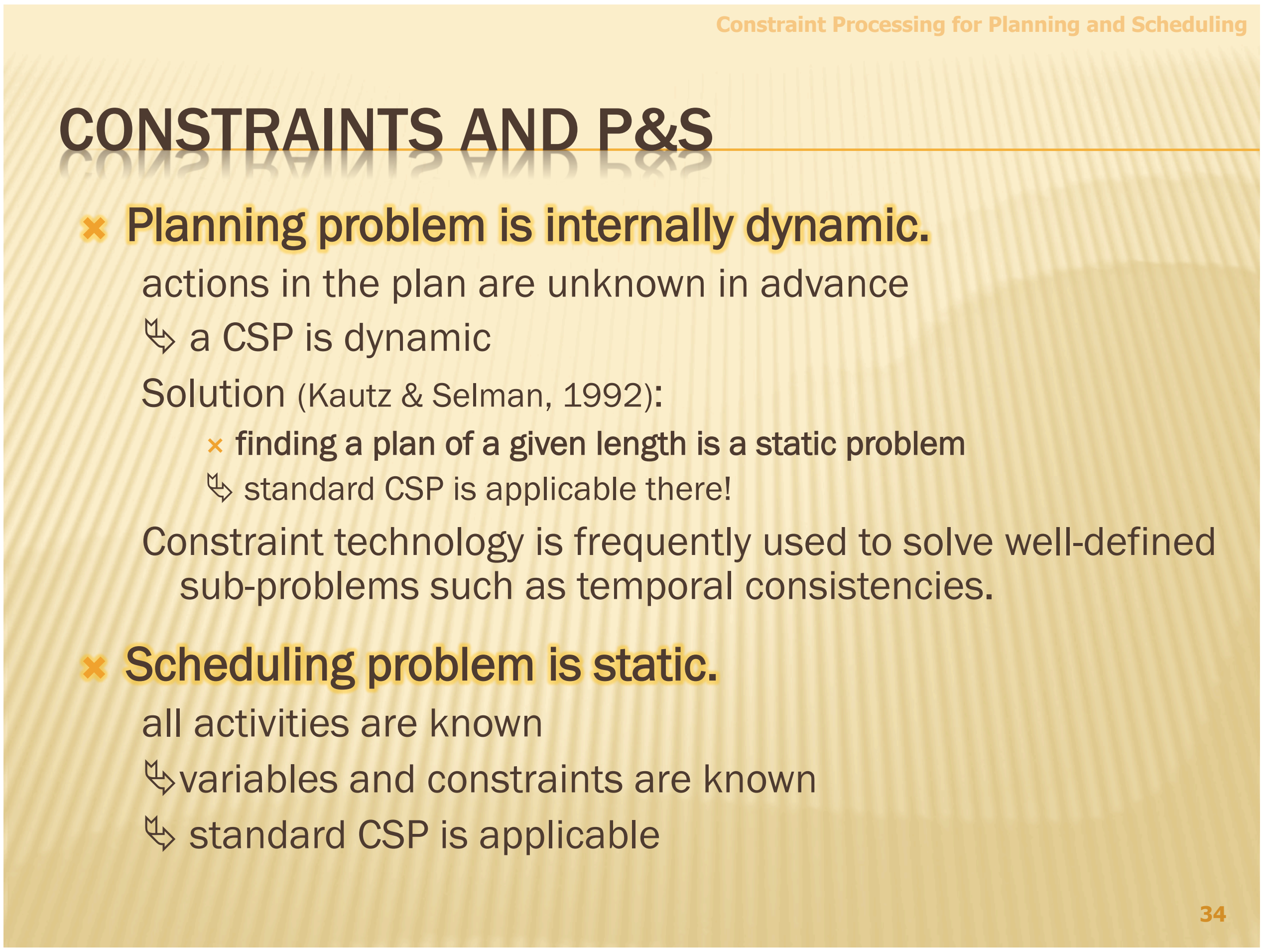
CONSTRAINTS FOR PLANNING AND SCHEDULING



TERMINOLOGY

"The **planning task** is to construct a sequence of actions that will transfer the initial state of the world into a state where the desired goal is satisfied‟

"The **scheduling task** is to allocate known activities to available resources and time respecting capacity, precedence (and other) constraints‟



CONSTRAINTS AND P&S

- **Planning problem is internally dynamic.**
  - actions in the plan are unknown in advance
  - ↳ a CSP is dynamic
  - Solution (Kautz & Selman, 1992):
    - finding a plan of a given length is a static problem
    - ↳ standard CSP is applicable there!
  - Constraint technology is frequently used to solve well-defined sub-problems such as temporal consistencies.
- **Scheduling problem is static.**
  - all activities are known
  - ↳variables and constraints are known
  - ↳ standard CSP is applicable

P&S VIA CSP?

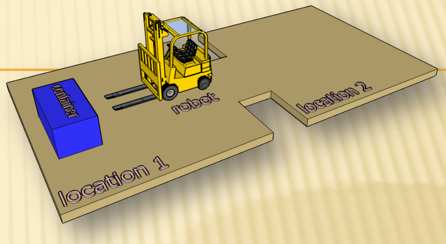
- **Exploiting state of the art constraint solvers!**
  - faster solver ⇒ faster planner
- **Constraint model is extendable!**
  - it is possible immediately to add other variables and constraints
  - modeling numerical variables, resource and precedence constraints for planning
  - adding side constraints to base scheduling models
- **Dedicated solving algorithms** encoded in the filtering algorithms for constraints!
  - fast algorithms accessible to constraint models

CONSTRAINTS FOR PLANNING
CONSTRAINT MODELS

PLANNING PROBLEM

- We deal with **classical AI planning**
  - looking for the shortest sequence of actions (a **plan**) transferring the initial state of world to the state satisfying some goal condition
  - **state** is described using a set of **multi-valued variables**
  - (grounded) **action** is specified by:
    - **precondition** (required values of some state variables before action execution)
    - **effect** (values of some state variables after action execution)

EXAMPLE PROBLEM



State Variables

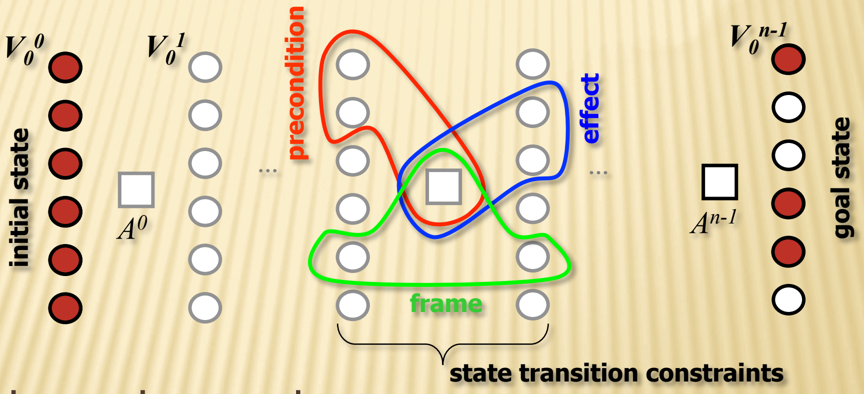
rloc ∈ {loc1,loc2} ;; robot's location
cpos ∈ {loc1,loc2,r} ;; container's position

Actions

move(r, loc1, loc2) ;; robot r at location loc1 moves to location loc2
  Precond: *rloc* = loc1
  Effects: *rloc* ← loc2
move(r, loc2, loc1) ;; robot r at location loc2 moves to location loc1
  Precond: *rloc* = loc2
  Effects: *rloc* ← loc1
load(r, c, loc1) ;; robot r loads container c at location loc1
  Precond: *rloc* = loc1, *cpos* = loc1
  Effects: *cpos* ← r
load(r, c, loc2) ;; robot r loads container c at location loc2
  Precond: *rloc* = loc2, *cpos* = loc2
  Effects: *cpos* ← r
unload(r, c, loc1) ;; robot r unloads container c at location loc1
  Precond: *rloc* = loc1, *cpos* = r
  Effects: *cpos* ← loc1
unload(r, c, loc2) ;; robot r unloads container c at location loc2
  Precond: *rloc* = loc2, *cpos* = r
  Effects: *cpos* ← loc2

SOLVING APPROACH

- Formulating the problem as a CSP
- Iterative extension of the plan length



- Backward search
  - instantiation of action variables
  - only actions relevant to the (sub)goal are tried

STRAIGHTFORWARD MODEL

original formulation

- action constraints

  $A^s = act \rightarrow \mathrm{Pre}(act)^s$, $\forall act \in \mathrm{Dom}(A^s)$

  $A^s = act \rightarrow \mathrm{Eff}(act)^{s+1}$, $\forall act \in \mathrm{Dom}(A^s)$

A^s = move21 → rloc^s = loc2
A^s = move21 → rloc^{s+1} = loc1

- frame constraint

  $A^s \in \mathrm{NonAffAct}(V_i) \rightarrow V_i^s = V_i^{s+1}$, $\forall i \in \langle 0, v\text{-}1 \rangle$

A^s = move21→ cpos^s = cpos^{s+1}

- **problems**
  - disjunctive constraints do no propagate well
    - do not prune well the search space
  - a huge number of constraints (depend on the number of actions)
    - the propagation loop takes a lot of time

MODEL REFORMULATION

- idea
  - encapsulate the logical constraints into a table constraint describing allowed tuples of values
  - be careful about the size of the table!

reformulated straightforward model

- action constraint = a single table

A^s	rloc^s	cpos^s	rloc^{s+1}	cpos^{s+1}
move21	loc2		loc1	
move12	loc1		loc2	
load1	loc1	loc1	loc1	r
…				

- frame constraint

  $A^s \in \text{NonAffAct}(V_i) \rightarrow V_i^s = V_i^{s+1}, \forall i \in \langle 0, v\text{-}1\rangle$

GP-CSP

- for each state variable V_i^s there is a supporting action variable S_i^s describing the action which sets the state variable (no-op action if the variable is not changed)

original model

- action constraints

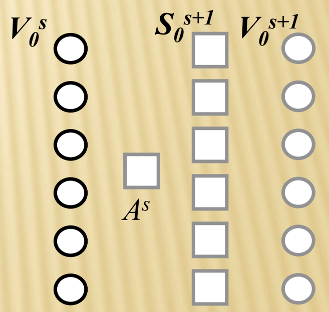
  $A^s = act \rightarrow \text{Pre}(act)^s$, $\forall act \in \text{Dom}(A^s)$

  $S_i^s = act \rightarrow V_i^s = val, \forall act \in \text{Dom}(S_i^s)$
- frame constraint

  $S_i^{s+1} = no\text{-}op \rightarrow V_i^s = V_i^{s+1}$.
- channeling constraint

  $A^s \in \text{AffAct}(V_i) \leftrightarrow S_i^{s+1} = A^s$, and

  $A^s \in \text{NonAffAct}(V_i) \leftrightarrow S_i^{s+1} = no\text{-}op$



reformulated model

- using a single table constraint instead of action constraints
- using a table constraint for a pair of channeling constraint
- frame constraints are kept in the logical form

CSP-PLAN

- idea
  - focus on modeling the reason for a value of the state variable (effect and frame constraints are merged)

original model

- precondition constraint
  - $A^s = act \rightarrow \text{Pre}(act)^s$, $\forall act \in \text{Dom}(A^s)$
- successor state constraint
  - $V_i^s = val \leftrightarrow A^{s-1} \in C(i,val) \vee (V_i^{s-1} = val \wedge A^{s-1} \in N(i))$
    - $C(i,val)$ = the set of actions containing $V_i = val$ among their effects
    - $N(i) = \text{NonAffAct}(V_i)$

reformulated model

- use a single table constraint to describe preconditions
- use ternary table constraints to describe successor state constraints (one table per state variable)

MODEL COMPARISON

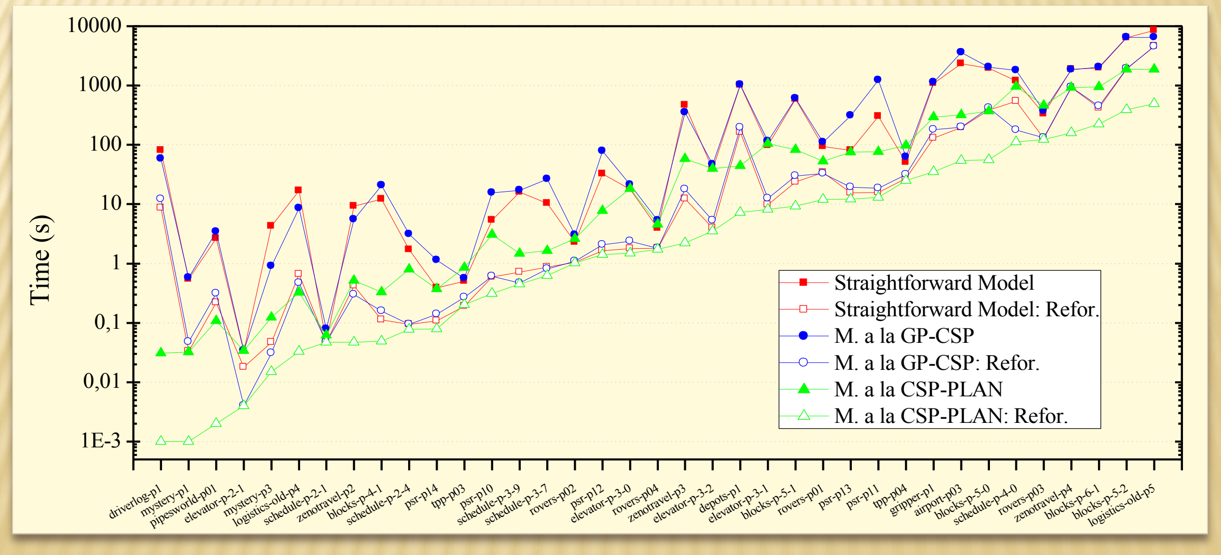
The total **number of constraints**

	original	**reformulated**
straightforward	n(ap+ae+v)	n(1+v)
GP-CSP	n(ap+ae+3v)	n(1+3v)
CSP-Plan	n(ap+vd)	n(1+v)

n - number of actions in the plan
a - number of grounded actions in the problem
v - number of multi-valued variables
p - average number of preconditions per action
e - average number of effects per action

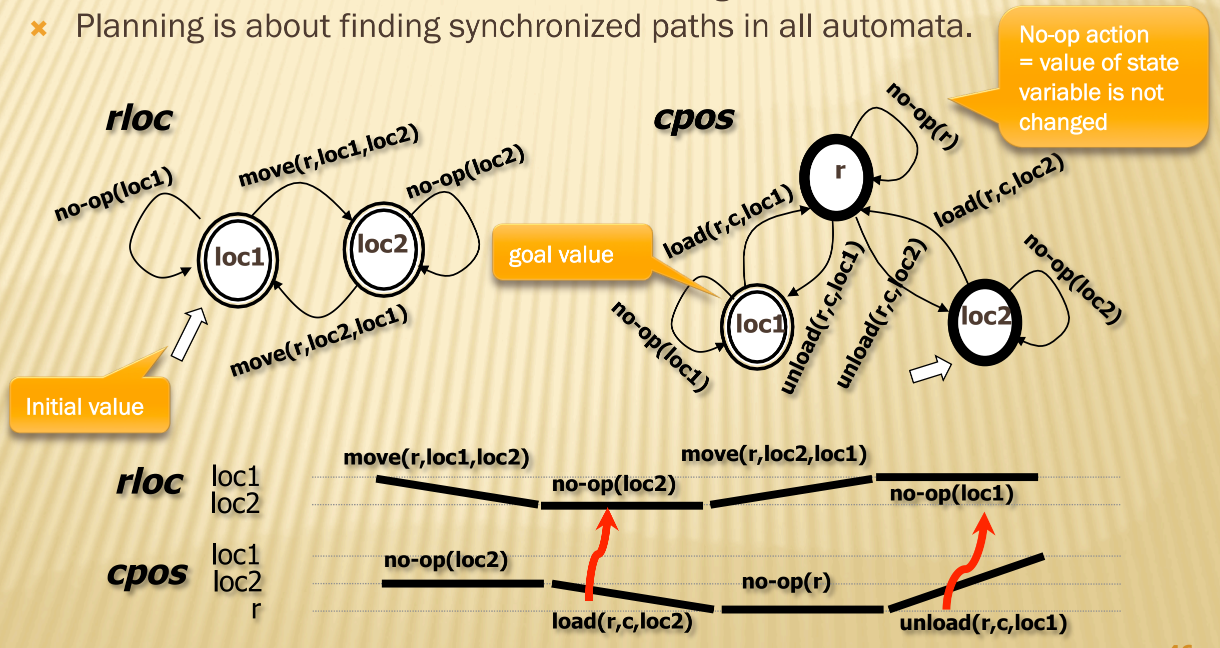
MODEL COMPARISON

The **runtime** to solve selected problems from IPC 1-5 (logarithmic scale)



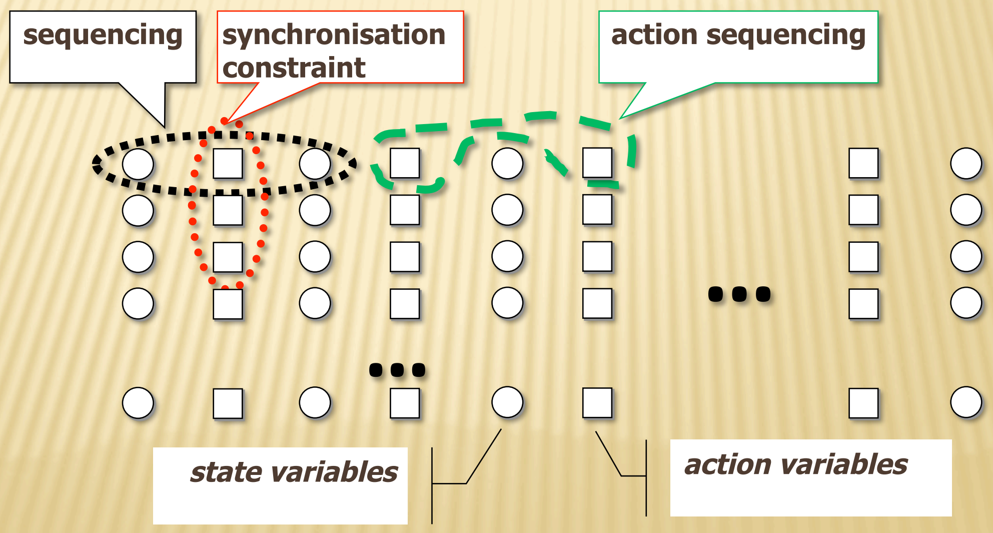
TIMELINES

- Planning can also be seen as **synchronized changes of state variables**.
- Evolution of each variable is described using finite state automaton.
- Planning is about finding synchronized paths in all automata.



CONSTRAINT MODEL (OVERVIEW)

- timeline model
- state and action variables organized to „layers“



SEARCH STRATEGY

- a more or less standard CP labeling procedure
- instantiating (by the search algorithm) only the action variables
  - the state variables are instantiated by inference
- variable selection
  - min-dom heuristic (only variables with real action in their domain are assumed)
- value selection (in two steps)
  - split the domain into no-op actions (explored first) and real actions
  - domains with real actions only are enumerated then

SUMMARY RESULTS (SOLVED PROBLEMS)

planning domain	SeP	PaP
airport (15)	4	6
blocks (16)	7	7
depots (10)	2	2
driverlog (15)	4	12
elevator (30)	30	27
freecell (10)	1	3
openstacks (7)	5	0
rovers (10)	4	6
tpp (15)	4	8
zenotravel (15)	6	11

problems from International Planning Competition, runtime limit 30 minutes

DETAILED RESULTS (RUNTIMES)

problem	plan length			runtime (ms)	
	SeP	PaP		SeP	PaP
		par	seq		
zenotravel-p01	1	1	1	10	20
zenotravel-p02	6	5	6	60	50
zenotravel-p03	6	5	9	300	130
zenotravel-p04	8	5	11	970	130
zenotravel-p05	11	5	14	153 990	240
zenotravel-p06	11	5	12	530 390	510
zenotravel-p07	≥12	6	16	-	560
zenotravel-p08	≥10	5	15	-	1 690
zenotravel-p09	≥11	6	24	-	145 760
zenotravel-p10	≥12	6	24	-	252 040
zenotravel-p11	≥9	6	16	-	41 780

CONSTRAINTS FOR PLANNING
TEMPORAL REASONING

FOUNDATIONS

What is **time**?

The mathematical structure of time is generally a **set with transitive and asymmetric ordering operation**.

The set can be continuous (reals) or discrete (integers).

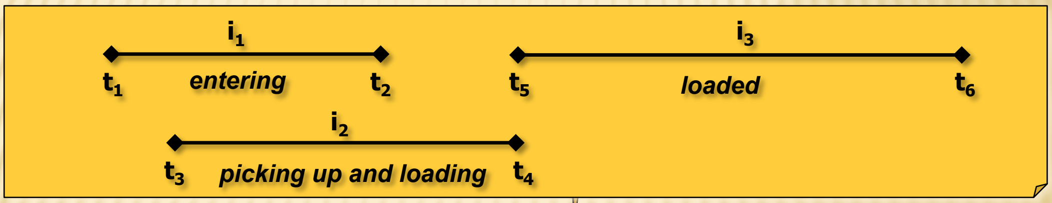
The planning/scheduling systems need to **maintain consistent information about time relations.**

We can see time relations:

- **qualitatively**
  - relative ordering (A finished before B)
  - typical for modeling causal relations in planning
- **quantitatively**
  - absolute position in time (A started at time 0)
  - typical for modeling exact timing in scheduling

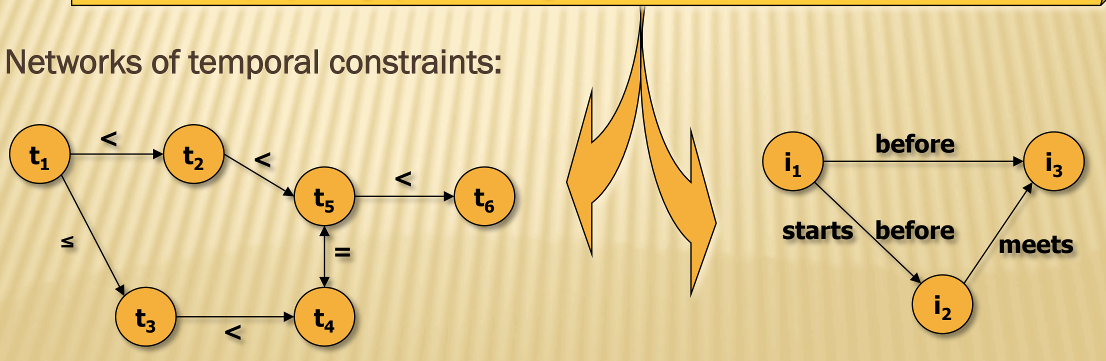
QUALITATIVE APPROACH (EXAMPLE)

- Robot starts entering a loading zone at time t_1 and stops there at time t_2.
- Crane starts picking up a container at t_3 and finishes putting it down at t_4.
- At t_5 the container is loaded onto the robot and stays there until time t_6.



Networks of temporal constraints:

$$\begin{tikzcd}
t_1 \arrow[r, "<"] \arrow[d, "\leq"'] & t_2 \arrow[r, "<"] & t_5 \arrow[r, "<"] \arrow[d, leftrightarrow, "="] & t_6 \\
t_3 \arrow[rr, "<"'] & & t_4 &
\end{tikzcd}$$



$$\begin{tikzcd}
i_1 \arrow[rr, "\text{before}"] \arrow[dr, "\text{starts}\ \text{before}"'] & & i_3 \\
& i_2 \arrow[ur, "\text{meets}"'] &
\end{tikzcd}$$

QUALITATIVE APPROACH (FORMALLY)

When **modeling time** we are interested in:

- **temporal references**
  (when something happened or hold)
  - **time points** (instants) when a state is changed
    instant is a variable over the real numbers
  - **time periods** (intervals) when some proposition is true
    interval is a pair of variables (x,y) over the real numbers, such that x<y
- **temporal relations** between the temporal references
  - **ordering** of temporal references

POINT ALGEBRA

symbolic calculus modeling relations between instants
without necessarily ordering them or allocating to exact times

There are three possible **primitive relations** between instants t_1 and t_2:

+ $[t_1 < t_2]$, $[t_1 > t_2]$, $[t_1 = t_2]$

× A set of primitives, meaning a disjunction of primitives, can describe any (even incomplete) relation between instants:
  + R = { {}, {<}, {=}, {>}, {<,=}, {>,=}, {<,>}, {<,=,>} }
    × {} means failure
    × {<,=,>} means that no ordering information is available
  + useful operations on R:
    × **set operations** ∩ (conjunction), ∪ (disjunction)
    × **composition operation** • ($[t_1 < t_2]$ and $[t_2 =< t_3]$ gives $[t_1 < t_3]$)

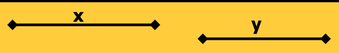
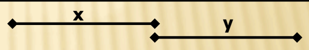
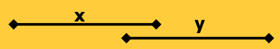
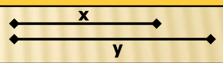
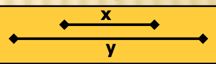
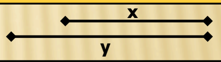
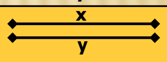
Consistency:

+ The **PA network** consisting of instants and relations between them is **consistent** when it is possible to assign a real number to each instant in such a way that all the relations between instants are satisfied.
+ To make the PA network consistent it is enough to make its transitive closure, for example using techniques of **path consistency**.
  × $[t_1\ r\ t_2]$ and $[t_1\ q\ t_3]$ and $[t_3\ s\ t_2]$ gives $[t_1\ r \cap (q \bullet s)\ t_2]$

INTERVAL ALGEBRA

symbolic calculus modeling relations between intervals
(interval is defined by a pair of instants i^- and i^+, $[i^- < i^+]$)

There are thirteen primitives:

x **before** y	$x^+ < y^-$	
x **meets** y	$x^+ = y^-$	
x **overlaps** y	$x^- < y^- < x^+$ & $x^+ < y^+$	
x **starts** y	$x^- = y^-$ & $x^+ < y^+$	
x **during** y	$y^- < x^-$ & $x^+ < y^+$	
x **finishes** y	$y^- < x^-$ & $x^+ = y^+$	
x **equals** y	$x^- = y^-$ & $x^+ = y^+$	
b′, m′, o′, s′, d′, f′	symmetrical relations	

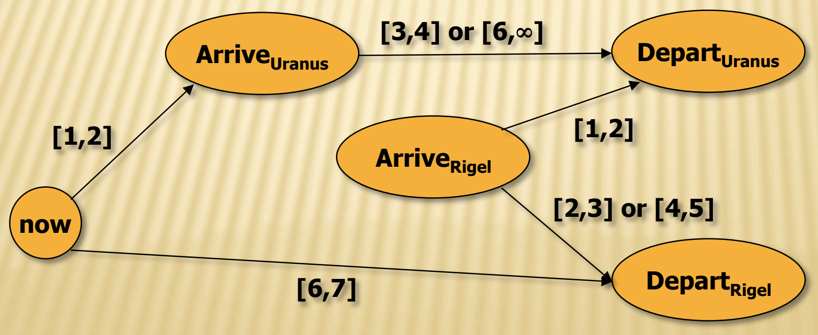
Consistency:

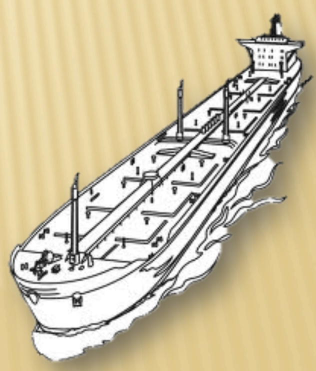
+ The **IA network** is **consistent** when it is possible to assign real numbers to x_i^-, x_i^+ of each interval x_i in such a way that all the relations between intervals are satisfied.
+ Consistency-checking problem for IA networks is an NP-complete problem.

Ghallab et al. (2004)

QUALITATIVE APPROACH (EXAMPLE)

- Two ships, Uranus and Rigel, are directing towards a dock.
- The Uranus arrival is expected within one or two days.
- Uranus will leave either with a light cargo (then it must stay in the dock for three to four days) or with a full load (then it must stay in the dock at least six days).
- Rigel can be serviced either on an express dock (then it will stay there for two to three days) or on a normal dock (then it must stay in the dock for four to five days).
- Uranus has to depart one to two days after the arrival of Rigel.
- Rigel has to depart six to seven days from now.


$\text{Arrive}_{\text{Uranus}}$
[3,4] or [6,∞]
$\text{Depart}_{\text{Uranus}}$
[1,2]
$\text{Arrive}_{\text{Rigel}}$
[1,2]
[2,3] or [4,5]
now
[6,7]
$\text{Depart}_{\text{Rigel}}$



QUALITATIVE APPROACH (FORMALLY)

- The basic temporal primitives are again **time points**, but now the relations are numerical.
- **Simple temporal constraints** for instants t_i and t_j:
  - unary: $a_i \leq t_i \leq b_i$
  - binary: $a_{ij} \leq t_i - t_j \leq b_{ij}$,

  where a_i, b_i, a_{ij}, b_{ij} are (real) constants

Notes:

- Unary relation can be converted to a binary one, if we use some fix origin reference point t_0.
- $[a_{ij}, b_{ij}]$ denotes the constraint between instants t_i a t_j.
- It is possible to use disjunction of simple temporal constraints.

STN

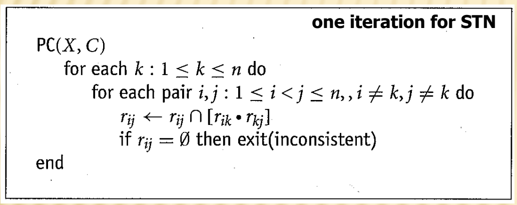
Simple Temporal Network (STN)

- only simple temporal constraints $r_{ij} = [a_{ij}, b_{ij}]$ are used
- **operations:**
  - composition: $r_{ij} \bullet r_{jk} = [a_{ij}+a_{jk}, b_{ij}+b_{jk}]$
  - intersection: $r_{ij} \cap r'_{ij} = [\max\{a_{ij}, a'_{ij}\}, \min\{b_{ij}, b'_{ij}\}]$
- **STN** is **consistent** if there is an assignment of values to instants satisfying all the temporal constraints.
- **Path consistency** is a complete technique making STN consistent (all inconsistent values are filtered out, one iteration is enough). Another option is using all-pairs minimal distance **Floyd-Warshall algorithm.**

ALGORITHMS

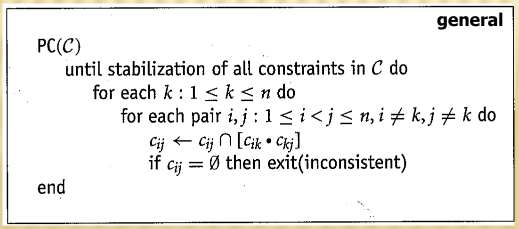
Path consistency

- finds a transitive closure of binary relations r
- one iteration is enough for STN (in general, it is iterated until any domain changes)
- works incrementally



one iteration for STN

```
PC($X, C$)
    for each $k : 1 \le k \le n$ do
        for each pair $i, j : 1 \le i < j \le n, , i \ne k, j \ne k$ do
            $r_{ij} \leftarrow r_{ij} \cap [r_{ik} \bullet r_{kj}]$
            if $r_{ij} = \emptyset$ then exit(inconsistent)
end
```

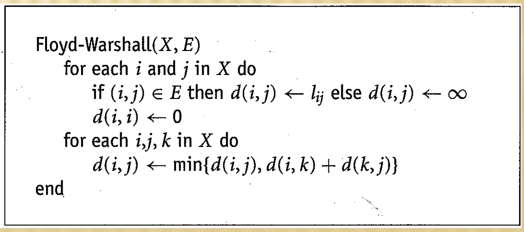


general

```
PC($C$)
    until stabilization of all constraints in $C$ do
        for each $k : 1 \le k \le n$ do
            for each pair $i, j : 1 \le i < j \le n, i \ne k, j \ne k$ do
                $c_{ij} \leftarrow c_{ij} \cap [c_{ik} \bullet c_{kj}]$
                if $c_{ij} = \emptyset$ then exit(inconsistent)
end
```

Floyd-Warshall algorithm

- finds minimal distances between all pairs of nodes
- First, the temporal network is converted into a directed graph
  - there is an arc from i to j with distance b_{ij}
  - there is an arc from j to i with distance $-a_{ij}$.
- STN is consistent iff there are no negative cycles in the graph, that is, $d(i,i) \ge 0$



```
Floyd-Warshall($X, E$)
    for each $i$ and $j$ in $X$ do
        if $(i, j) \in E$ then $d(i, j) \leftarrow l_{ij}$ else $d(i, j) \leftarrow \infty$
        $d(i, i) \leftarrow 0$
    for each $i, j, k$ in $X$ do
        $d(i, j) \leftarrow \min\{d(i, j), d(i, k) + d(k, j)\}$
end
```

TCSP

Temporal Constraint Network (TCSP)

+ It is possible to use **disjunctions of simple temporal constraints.**
+ Operations • and ∩ are being done over the sets of intervals.
+ **TCSP** is **consistent** if there is an assignment of values to instants satisfying all the temporal constraints.
+ Path consistency does not guarantee in general the consistency of the TCSP network!
+ A straightforward **approach** (constructive disjunction):
  - decompose the temporal network into several STNs by choosing one disjunct for each constraint
  - solve obtained STN separately (find the minimal network)
  - combine the result with the union of the minimal intervals

CONSTRAINTS FOR SCHEDULING

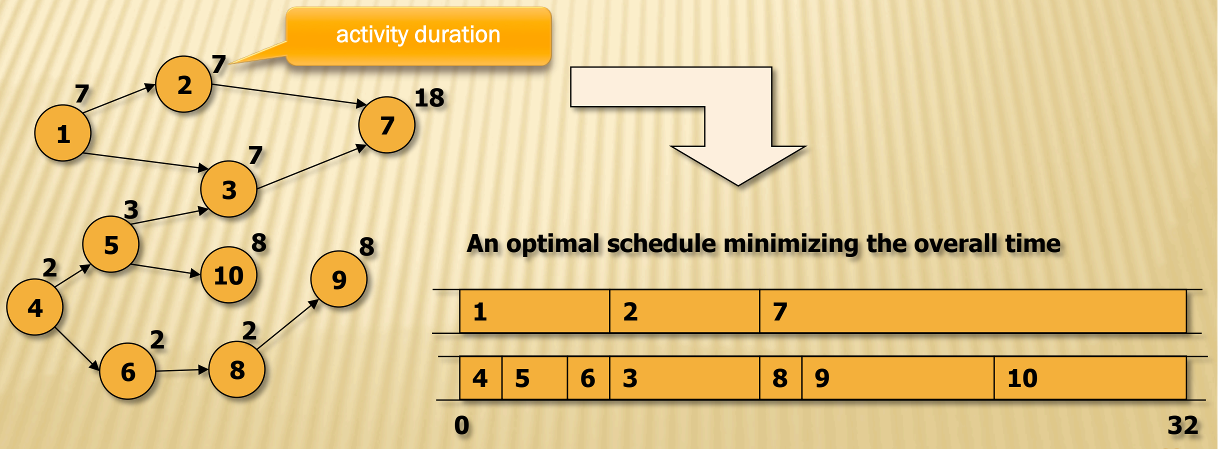
BASE CONSTRAINT MODEL

SCHEDULING PROBLEM

Scheduling deals with **optimal resource allocation** of a given set of activities **in time**.

Example (two workers building a bicycle):

+ activities have a fixed duration, cannot be interrupted and the precedence constraints must be satisfied



SCHEDULING MODEL

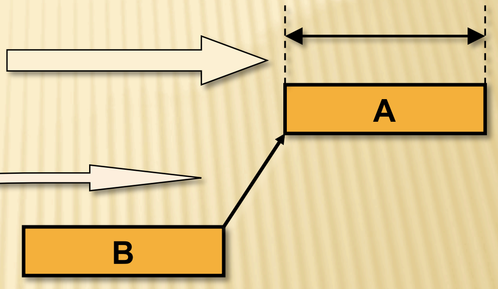
- **Scheduling problem** is static so it can be directly **encoded as a CSP.**
- Constraint technology is used for **full scheduling.**

Constraint model:

+ Variables
  - position of activity A in time and space
  - time allocation: **start(A), [p(A), end(A)]**
  - resource allocation: **resource(A)**
+ Domain
  - **release times** and **deadlines** for the time variables
  - **alternative resources** for the resource variables
+ Constraints
  - sequencing and resource capacities

SCHEDULING MODEL (CONSTRAINTS)

- Time relations
  - start(A) + p(A) = end(A)
  - sequencing
    - B « A
    - ↳ end(B) ≤ start(A)



- Resource capacity constraints
  - unary resource (activities cannot overlap)
    - A « B ∨ B « A (∨ resource(A) ≠ resource(B))
    - ↳ end(A) ≤ start(B) ∨ end(B) ≤ start(A)

CONSTRAINTS FOR SCHEDULING

RESOURCE CONSTRAINTS

RESOURCES

- Resources are used in slightly different meanings in planning and scheduling!

resources in scheduling

= **machines** (space) for processing the activities

resources in planning

= consumed/produced **material** by the activities

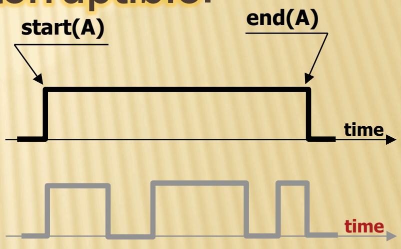
+ resource in the scheduling sense is often handled via the logical precondition (e.g. hand is free)

RESOURCE TYPES

- **unary (disjunctive) resource**
  + a single activity can be processed at any time
- **cumulative (discrete) resource**
  + several activities can be processed in parallel if capacity is not exceeded.
- **producible/consumable resource**
  + activity consumes/produces some quantity of the resource
  + minimal capacity is requested (consumption) and maximal capacity cannot be exceeded (production)

UNARY RESOURCES

- Activities **cannot overlap.**
- We assume that activities are **uninterruptible**.
  - **uninterruptible** activity occupies the resource from its start till its completion
  - **interruptible** (preemptible) activity can be interrupted by another activity

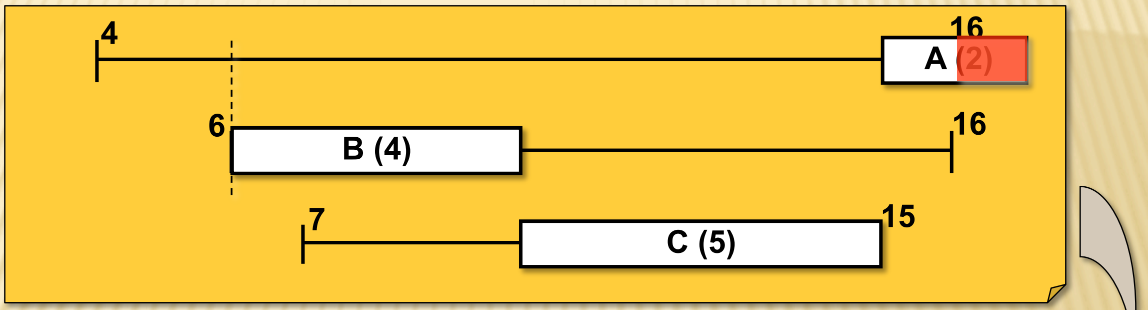


Note:
  There exists variants of the presented filtering algorithms for interruptible activities.

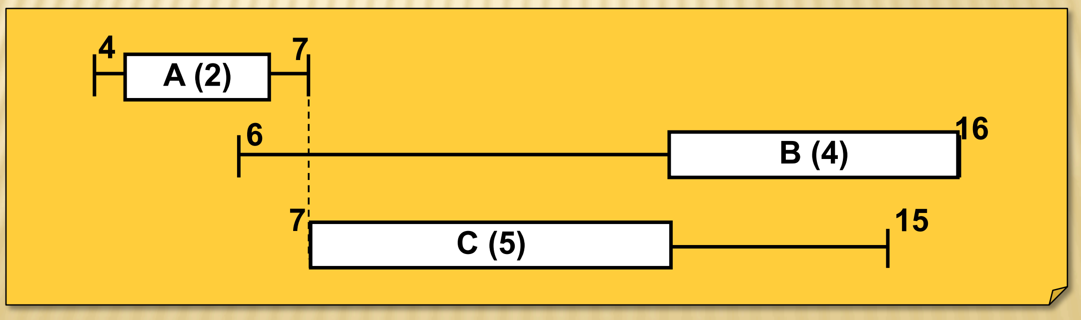
- A simple model with **disjunctive constraints**
  - A « B v B « A
    - end(A) ≤ start(B) v end(B) ≤ start(A)

EDGE FINDING

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 (INFERENCE RULES)

The inference rules:

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

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

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

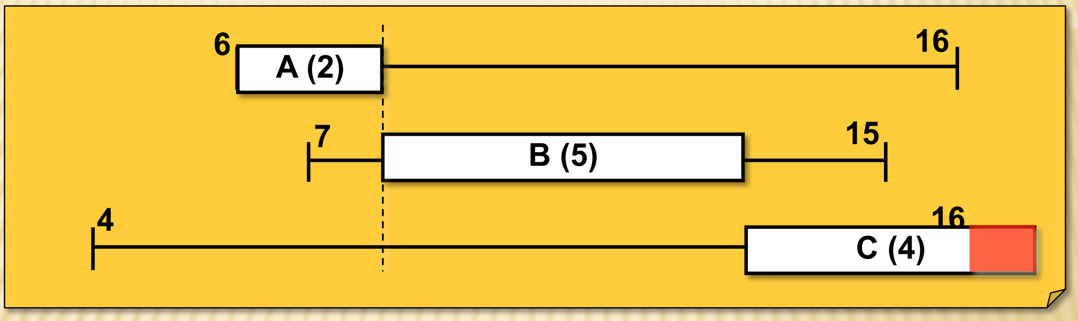
$\Omega \ll A \Rightarrow start(A) \geq \max\{ est(\Omega') + p(\Omega') \mid \Omega' \subseteq \Omega \}$

In practice:

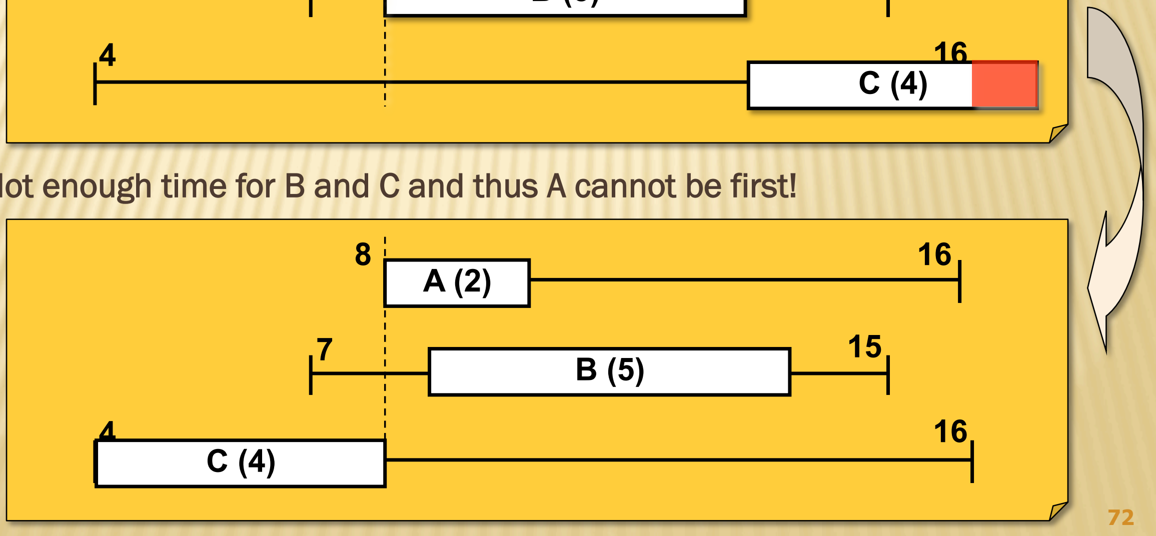
- there are $n.2^n$ pairs (A,Ω) to consider (too many!)
- instead of Ω use so called **task intervals** [X,Y]
  $\{C \mid est(X) \leq est(C) \wedge lct(C) \leq lct(Y)\}$
  - time complexity $O(n^3)$, frequently used incremental algorithm
- there are also $O(n^2)$ and $O(n.\log n)$ algorithms

NOT-FIRST/NOT-LAST

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 (INFERENCE RULES)

Not-first inference 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-last (symmetrical) inference rules:

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

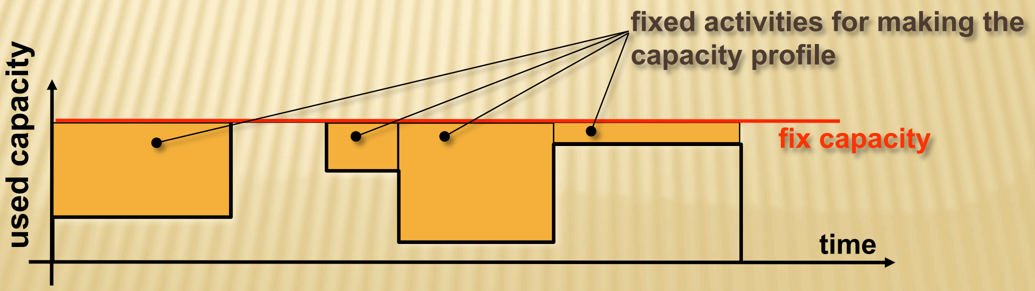
$\neg \Omega \ll A \Rightarrow end(A) \leq \max\{ lst(B) \mid B \in \Omega \}$

In practice:

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

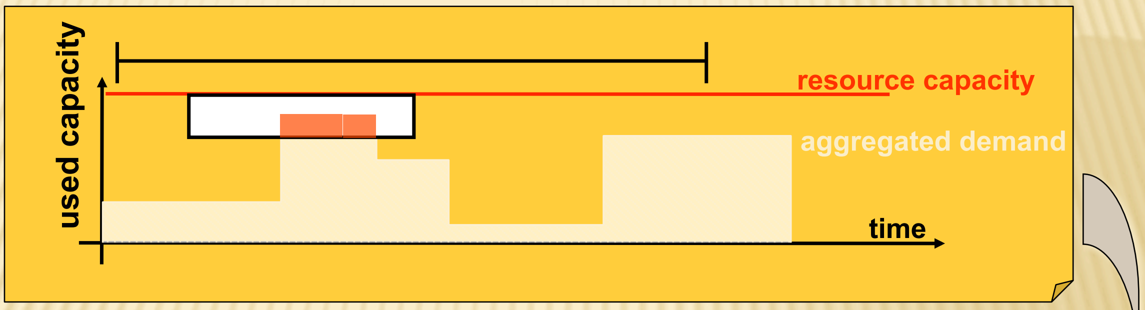
CUMULATIVE RESOURCES

- Each **activity uses some capacity** of the resource – **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

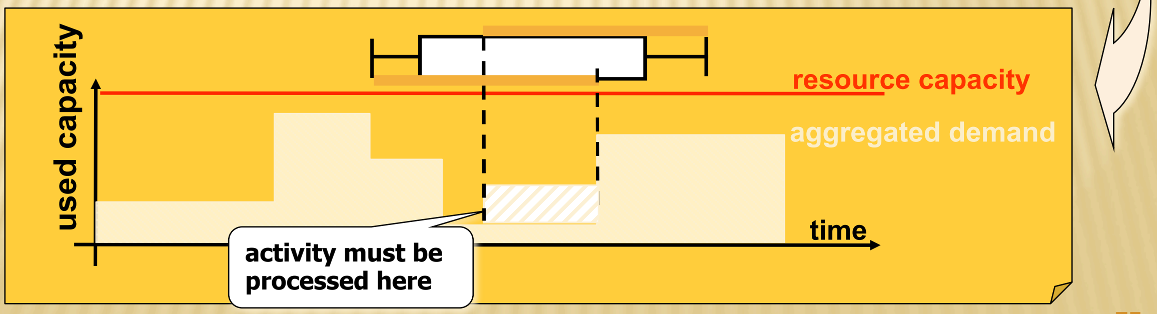


AGGREGATED DEMANDS

Where is enough capacity for processing the activity?



How the aggregated demand is constructed?



TIMETABLE CONSTRAINT

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

$$\forall t \quad \sum_{start(A_i) \le t < end(A_i)} cap(A_i) \le cap$$

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

$$\forall t \ \sum_{A_i} X(A_i,t) \cdot cap(A_i) \le cap$$

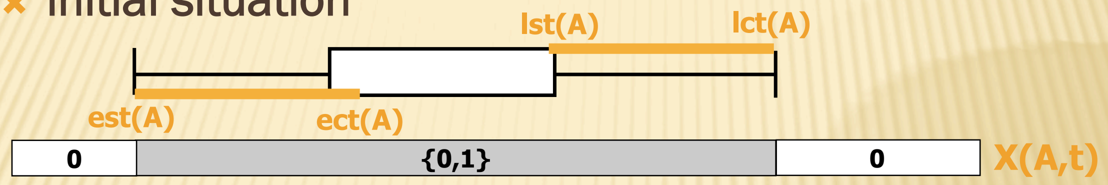
cap=1
in unary resource

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

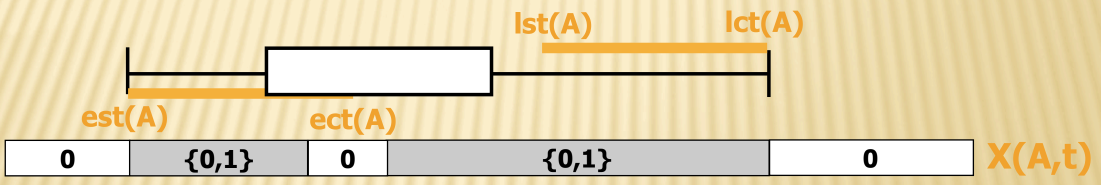
* discrete time is expected

TIMETABLE CONSTRAINT (FILTERING EXAMPLE)

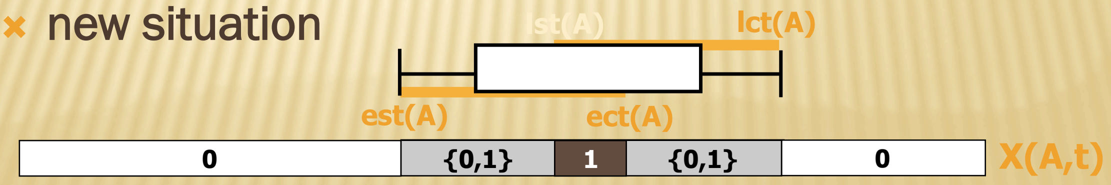
- initial situation



- some positions forbidden due to capacity



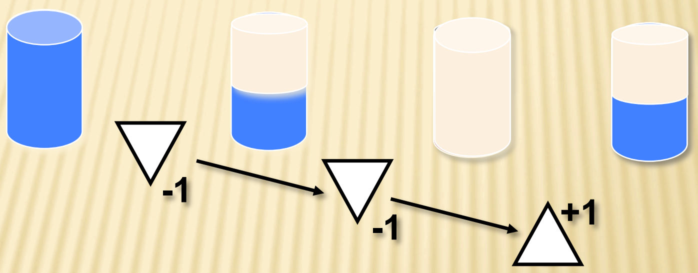
- new situation



RESERVOIRS

Producible/consumable resource

- Each event describes how much it increases or decreases the level of the resource.



- Cumulative resource can be seen as a special case of producible/consumable resource (reservoirs).
  + Each activity consists of consumption event at the start and production event at the end.

RELATIVE ORDERING

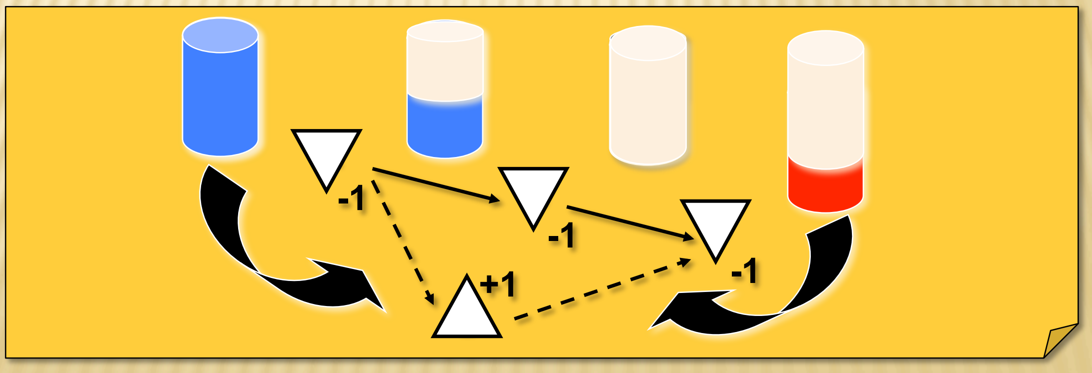
When time is relative (ordering of activities)

then edge-finding and aggregated demand deduce nothing

We can still use information about ordering of events and resource production/consumption!

Example:

Reservoir: events consume and supply items



RESOURCE PROFILES

- Event A „produces“ **prod(A)** quantity:
  - positive number means **production**
  - negative number means **consumption**
- optimistic resource profile (orp)
  - maximal possible level of the resource when A happens
  - events known to be before A are assumed together with the production events that can be before A

  orp(A) = InitLevel + prod(A) + $\sum_{B \ll A}$ prod(B) + $\sum_{B?A \wedge \mathrm{prod}(B)>0}$ prod(B)
- pessimistic resource profile (prp)
  - minimal possible level of the resource when A happens
  - events known to be before A are assumed together with the consumption events that can be before A

  prp(A) = InitLevel + prod(A) + $\sum_{B \ll A}$ prod(B) + $\sum_{B?A \wedge \mathrm{prod}(B)<0}$ prod(B)

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

ORP FILTERING (INFERENCE RULES)

$orp(A) < MinLevel \Rightarrow fail$

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

$orp(A) - prod(B) - \sum_{B«C \wedge C?A \wedge prod(C)>0} prod(C) < MinLevel \Rightarrow B«A$
for any B such that B?A and prod(B)>0

+ "if production in B is planned after A and the minimal required level in the resource is not reached then B must be before A"

PRP FILTERING (INFERENCE RULES)

$prp(A) > MaxLevel \Rightarrow fail$

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

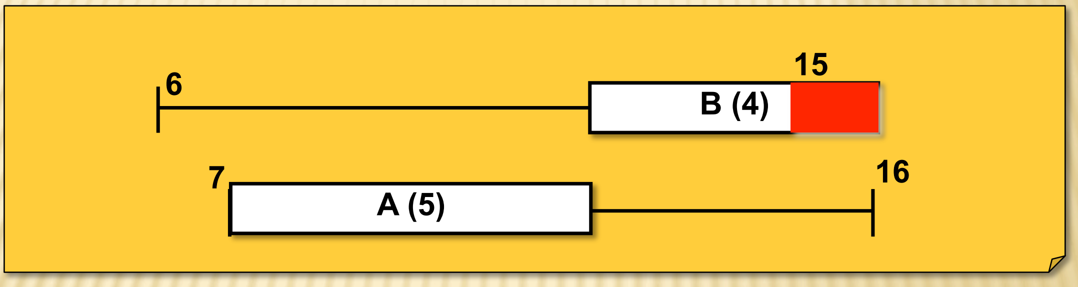
$prp(A) - prod(B) - \sum_{B«C \wedge C?A \wedge prod(C)<0} prod(C) > MaxLevel \Rightarrow B«A$
for any B such that B?A and prod(B)<0

+ "if consumption in B is planned after A and the maximal required level in the resource is exceeded then B must be before A"

FROM TIME WINDOWS TO ORDERING

DETECTABLE PRECEDENCE

What happens if activity A is processed before B?



- Restricted time windows can be used to deduce new precedence relations.

  est(A)+p(A)+p(B) > lct(B) ⇒ B « A

ALTERNATIVE RESOURCES

- How to model alternative resources for a given activity?
- Use a **duplicate activity** for each resource.
  - duplicate activity participates in the 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 the precedence constraints (e.g. within a job)
  - restricted times of duplicate activities are propagated to the original activity and vice versa.

ALTERNATIVE RESOURCES (FILTERING RULES)

- Let A_u be the duplicate activity of A allocated to resource $u \in res(A)$.

  $u \in resource(A) \Rightarrow start(A) \leq start(A_u)$

  $u \in resource(A) \Rightarrow end(A_u) \leq end(A)$

  $start(A) \geq \min\{start(A_u) : u \in resource(A)\}$

  $end(A) \leq \max\{end(A_u) : u \in resource(A)\}$

  failure related to $A_u \Rightarrow resource(A) \backslash \{u\}$

  Actually, it is maintaining **constructive disjunction** between the alternative activities.

CONSTRAINTS FOR SCHEDULING

SEARCH STRATEGIES

BRANCHING SCHEMES

Branching = resolving disjunctions

Traditional scheduling approaches:

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

How to describe criticality and flexibility formally?

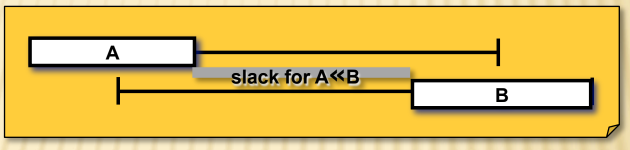
Smith and Cheng (1993)

SLACK

Slack is a formal description of flexibility

- Slack for **a given order of two activities**

  „free time for shifting the activities“



  slack(A « B) = max(end(B)) - min(start(A)) - p({A,B})

- Slack for **two activities**

  slack({A,B}) = max{ slack(A « B), slack(B « A) }

- Slack for **a group of activities**

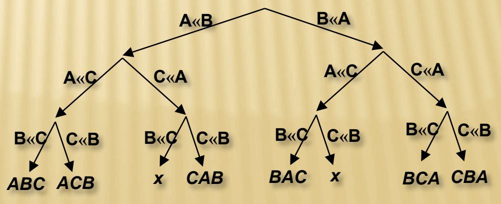
  slack(Ω) = max(end(Ω)) - min(start(Ω)) - p(Ω)

Smith and Cheng (1993)

ORDER BRANCHING

A « B v ¬ A « B

- **Which activities should be ordered first?**
  - the most critical pair (first-fail)
  - the pair with **the minimal slack({A,B})**
- **Which order should be selected?**
  - the most flexible order (succeed-first)
  - the order with **the maximal slack(A??B)**
- $O(n^2)$ choice points

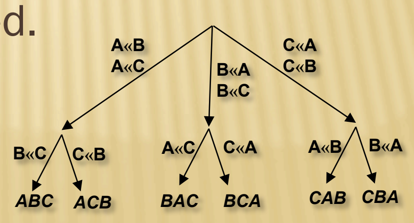


Baptiste et al. (1995)

FIRST/LAST BRANCHING

(A « Ω v ¬A « Ω) or (Ω « A v ¬ Ω « A)

- **Should we look for the first or for the last activity?**
  - select a **smaller set** among possible first or possible last activities (first-fail)
- **Which activity should be selected?**
  - If first activity is being selected then the activity with the **smallest min(start(A))** is preferred.
  - If last activity is being selected then the activity with the **largest max(end(A))** is preferred.
- O(n) choice points



RESOURCE SLACK

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

CONCLUSIONS

SUMMARY (CONSTRAINT SATISFACTION)

Basic constraint satisfaction framework:

- **local consistency** connecting filtering algorithms for individual constraints
- **search** resolves remaining disjunctions

Problem solving:

- **declarative modeling** of problems as a CSP
- **dedicated algorithms** encoded in constraints
- special **search strategies**

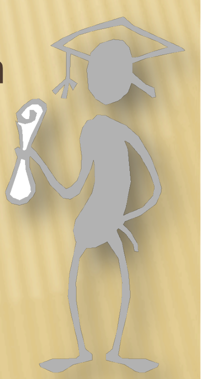
SUMMARY (CONSTRAINTS IN PLANNING AND SCHEDULING)

Constraint satisfaction techniques are used

- for **solving particular sub-problems** (temporal and resource consistency)
- for **modeling and solving a complete problem**

It is possible

- to **exploit constraint satisfaction principles** in own algorithms
- to **use an existing constraint solver** (modeling, adding specific inference techniques, and customizing search strategies)



CONSTRAINT SOLVERS

- It is not necessary to program all the presented techniques from scratch!
- Use existing constraint solvers (packages)!
  - provide **implementation of data structures** for modeling variables' domains and constraints
  - provide a basic **consistency framework**
  - provide **filtering algorithms** for many constraints (including global constraints)
  - provide basic **search strategies**
  - usually **extendible** (new filtering algorithms, new search strategies)

Some systems with **constraint satisfaction packages:**

- **Prolog:** SICStus Prolog, ECLiPSe, CHIP, Prolog IV, GNU Prolog, IF/Prolog
- **C/C++:** ILOG CP Optimizer, Gecode, CHIP++
- **Java:** Choco, JCK, JCL, Koalog
- **Oz:** Mozart

COMMENTED BIBLIOGRAPHY

Allen, J.F. (1983). Maintaining knowledge about temporal intervals. Communications of the ACM, 21(11): 832-843.
Introduction of interval algebra and description of path-consistency filtering algorithm for handling it.

Baptiste, P. and Le Pape, C. (1996). Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. Proceedings of the Fifteenth Workshop of the U.K. Planning Special Interest Group (PLANSIG).
Description of edge-finding rules for non-preemptive disjunctive scheduling, preemptive and mixed disjunctive scheduling, and non-preemptive cumulative scheduling, and a quadratic algorithm for not-first/not-last rules.

Baptiste, P.; Le Pape, C.; Nuijten, W. (2001). Constraint-based Scheduling: Applying Constraints to Scheduling Problems. Kluwer Academic Publishers, Dordrecht.
A comprehensive text on using constraint satisfaction techniques in scheduling with detailed description of many filtering algorithms for resource constraints.

Barták, R. (2005). Constraint Satisfaction for Planning and Scheduling. In Ionannis Vlahavas, Dimitris Vrakas (eds.): Intelligent Techniques for Planning, Idea Group, 2005, pp. 320-353
An introductory and survey text about constraint satisfaction techniques for planning and scheduling.

Barták, R. (2011). A Novel Constraint Model for Parallel Planning. In Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2011), AAAI Press, pp. 9-14.
Description of planning constraint model based on timelines (finite state automata).

Barták, R. and Čepek, O. (2005). Incremental Propagation Rules for a Precedence Graph with Optional Activities and Time Windows. In Proceedings of The 2nd Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA), Volume II, Stern School of Business, New York, 552-560.
Description of incremental algorithms for maintaining a transitive closure of the precedence graph with optional activities and realising the energy precedence constraint on unary resources.

Barták, R. and Toropila, D. (2010). Solving Sequential Planning Problems via Constraint Satisfaction. In Fundamenta Informaticae, Volume 99, Number 2, IOS Press, pp. 125-145.
Comparison of several constraint models for planning problems. New models are based on state variables and tabular constraints.

Brucker, P. (2001). Scheduling Algorithms. Springer Verlag.
A comprehensive book on traditional scheduling algorithms including detailed classification of many scheduling problems.

Carlier, J. and Pinson, E. (1994). Adjustment of heads and tails for the job-shop problem. European Journal of Operational Research 78(2), 146-161.
Description of first O(n.log n) algorithm for edge-finding, but this algorithm requires complex data structures.

Caseau, Y. and Laburthe, F. (1995). Disjunctive scheduling with task intervals. LIENS Technical Report 95-25, Laboratoire d'Informatique de l'Ecole Normale Superieure.
Description of incremental $O(n^3)$ algorithm for edge-finding using task intervals.

Cesta, A. and Stella, C. (1997). A Time and Resource Problem for Planning Architectures. Recent Advances in AI Planning (ECP'97), LNAI 1348, Springer Verlag, 117-129.
Description of resource profiles and orp/prp filtering rules.

Dechter, R.; Meiri, I.; Pearl, J. (1991). Temporal Constraint Networks. Artificial Intelligence 49: 61-95.
Introduction of Temporal Constraint Networks and Simple Temporal Problems.

Dechter, R. (2003). Constraint Processing. Morgan Kaufmann.
A comprehensive book on constraint satisfaction techniques, including a detailed description of temporal constraint networks.

Focacci, F.; Laborie, P.; Nuijten, W. (2000). Solving scheduling problems with setup times and alternative resources. In Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS). AAAI Press, 92-101.
Description of path optimization constraint for minimizing setup times/costs in problems with alternative resources.

Ghallab, M.; Nau, D.; Traverso, P. (2004). Automated Planning: Theory and Practice. Morgan Kaufmann.
A comprehensive book on planning, including a description of constraint satisfaction techniques for planning.

Laborie, P. (2003). Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. Artificial Intelligence 143, 151-188.
Introduction of filtering rules for energy precedence and balance constraints (algorithms are not described).

Lhomme, O. (1993). Consistency techniques for numeric CSPs. In Proc. 13th International Joint Conference on Artificial Intelligence.
Description of arc-B-consistency algorithm.

Mackworth, A.K. (1977). Consistency in Networks of Relations. Artificial Intelligence 8, 99-118.
Description of the basic arc and path consistency algorithms – AC-1, AC-2, AC-3, PC-1, PC-2.

Marriott, K. and Stuckey, P.J. (1998). Programming with Constraints: An Introduction. MIT Press.
A practically oriented book on using constraint satisfaction technology for problem solving.

Martin, P. and Shmoys, D.B. (1996). A new approach to computing optimal schedules for the job-shop scheduling problem. Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization. LNCS 1084, Springer Verlag, 389-403.
Description of alternative formulation of edge-finding rules.

Montanari, U. (1974). Networks of constraints: fundamental properties and applications to picture processing. Information Sciences 7, 95-132.
Introduction and formalization of constraint networks, defining path-consistency and algorithm for PC.

Nuijten, W.P.M. (1994). Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach. PhD thesis, Eindhoven University of Technology.
Description of several filtering algorithms for scheduling problems including the cumulative version of edge-finding and not-first/not-last rules.

Prosser, P.; Stergiou, K.; Walsh, T. (2000). Singleton Consistencies. Proceedings Principles and Practice of Constraint Programming (CP2000), 353-368.
Description and a theoretical study of singleton consistency techniques.

Phan-Huy, T. (2000). Constraint Propagation in Flexible Manufacturing. LNEMS 492, Springer Verlag.
PhD thesis with the description of constraint propagation algorithms for disjunctive scheduling problems.

Régin, J.-Ch. (1994). A filtering algorithm for constraints of difference in CSPs. Proceedings of 12th National Conference on Artificial Intelligence, AAAI Press, 362-367.
Description of the filtering algorithm behind the all-different constraint – based on matching over bipartite graphs.

Schulte, Ch. (2002). Programming Constraint Services. LNAI 2302, Springer Verlag.
A book describing insides of constraint solvers.

Torres, P. and Lopez, P. (2000). On Not-First/Not-Last conditions in disjunctive scheduling. European Journal of Operational Research 127, 332-343.
Description of $O(n^2)$ filtering algorithms for not-first/not-last rules.

Tsang, E. (1995). Foundations of Constraint Satisfaction. Academic Press, London.
A comprehensive book on foundational constraint satisfaction techniques with description of many consistency algorithms and their theoretical study.

Vilain, M. and Kautz, H. (1986). Constraint propagation algorithms for temporal reasoning. In Proceedings of the National Conference on Artificial Intelligence (AAAI), 377-382.
Introduction of point algebra and proof that consistency-checking problem of the IA problem is an NP-complete problem, while PA is a tractable problem.

Vilím, P. and Barták, R. (2002). Filtering Algorithms for Batch Processing with Sequence Dependent Setup Times. Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS). AAAI Press, 312-320.
Description of edge-finding and not-first/not-last algorithms for batch processing with sequence dependent setup times.

Vilím, P. (2002). Batch processing with sequence dependent setup times: New results. In Proceedings of the 4th Workshop on Constraint Programming for Decision and Control (CPDC), Gliwice, 53-58.
Description of edge-finding and not-first/not-last algorithms for problems with sequence dependent setup times, introduction of detectable precedences.

Vilím, P. (2004). O(n log n) Filtering Algorithms for Unary Resource Constraint. Proceedings of CPAIOR 2004. LNCS 3011, Springer Verlag, 335-347.
Description of O(n.log n) algorithms for not-first/not-last rules and detectable precedences.

Vilím, P.; Barták, R.; Čepek, P. (2005). Extension of O(n log n) filtering algorithms for the unary resource constraint to optional activities. Constraints, 10(4): 403-425.
Description of O(n.log n) versions of filtering algorithms for edge finding, not-first/not-last, and detectable precedences and their extension to optional activities.

Wallace, M. (1994). Applying Constraints for Scheduling. In Mayoh B. and Penjaak J. (eds.), Constraint Programming. NATO ASI Series, Springer Verlag.
A survey text on using constraint satisfaction technology in scheduling.