Constraint Programming and Scheduling

Materials from the course taught at HTWG Constanz, Germany

Hana Rudová

Faculty of Informatics, Masaryk University
Brno, Czech Republic

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

May 2009
Constraint Programming and Scheduling: Outline

- Constraint Programming
  - Introduction
  - Modeling
  - Propagation
  - Search

- Constraint-based Scheduling
  - Introduction
  - Modeling
  - Propagation
  - Search
Constraint Programming: Introduction

1. CSP

2. CP Approach

3. Complexity
Materials on Constraint Programming

- Rina Dechter: Constraint processing.  
  - http://www.ics.uci.edu/~dechter/books/

- Francesca Rossi, Peter Van Beek, Toby Walsh (editors):  
  Handbook of Constraint Programming, Elsevier, 2006

  - full text available:  
    http://cswww.essex.ac.uk/Research/CSP/edward/FCS.html

- Roman Barták: On-line guide to constraint programming.  

- Constraint Programming Online  
**Constraint**

- **Given**
  - set of **domain variables** \( Y = \{y_1, \ldots, y_k\} \)
  - finite set of values (**domain**) \( D = D_1 \cup \ldots \cup D_k \)

  **constraint** \( c \) defined on \( Y \) is a subset of \( D_1 \times \ldots \times D_k \) i.e. **relation**
  - it constrains values which can variables take at the same time

- **Example:**
  - variables: \( A, B \)
  - domains: \( \{0,1\} \) for \( A \), \( \{1,2\} \) for \( B \)
  - constraint: \( A \neq B \) or \( (A,B) \in \{(0,1),(0,2),(1,2)\} \)

**Constraint** \( c \) defined on variables \( y_1, \ldots, y_k \) is **satisfied**, if \( (d_1, \ldots, d_k) \in c \) holds for values \( d_1 \in D_1, \ldots, d_k \in D_k \)

- **Example (continues):**
  - constraint is satisfied for \( (0,1), (0,2), (1,2) \)
  - and unsatisfied for \( (1,1) \)
Constraint Satisfaction Problem (CSP)

Given

- finite set of variables $X = \{x_1, \ldots, x_n\}$
- finite set of values (domain) $D = D_1 \cup \ldots \cup D_n$
- finite set of constraints $C = \{c_1, \ldots, c_m\}$
  - each constraint defined over subset of $X$

Constraint satisfaction problem (CSP) is the tripple $(X, D, C)$

Example:

- variables: A, B, C
- domains: {0,1} for A, {1} for B, {0,1,2} for C
- constraints: $A \neq B$, $B \neq C$
Solution of CSP

Example: A in 0..1, B \# = 1, C in 0..2, A \# \\= B, B \# \\= C

Partial assignment of variables \((d_1, \ldots, d_k), k < n\)
- some variables have assigned a value
  - A=1, B=1

Complete assignment of variables \((d_1, \ldots, d_n)\)
- all variables have assigned a value
  - A=1, B=1, C=0

Solution of CSP
- complete assignment of variables satisfying all the constraints
- \((d_1, \ldots, d_n) \in D_1 \times \ldots \times D_n\) is solution of \((X, D, C)\)
  - \(x_{i_1}, \ldots x_{i_k}\) holds for each \(c_i \in C\) \((d_{i_1}, \ldots d_{i_k}) \in c_i\)
  - A=0, B=1, C=2

Constraint Optimization Problem
- objective function \(F\) defined on variables from \(X\)
  - B+C
- search for optimal solution of \(F\)
  - minimal: B=1, C=0
Constraint Programming (CP) Approach

**Formulation** of given problem using constraints: **modeling**

**Solving** of given formulation using

- domain specific methods
- general methods
General Methods

Example: A in 0..2, B ≠ 1, C in 0..2, A \≠ B, B \≠ C, A \≠ C

Constraint propagation algorithms
- allow to remove inconsistent values from domains of variables
  - values A=1 and C=1 are removed
- simplify problem
- maintain equivalency between original and simplified problem
- for computation of local consistency
- approximates global consistency

Search algorithms
- search through solution space
  1. select value 0 for A, as a consequence C=2
  2. select value 2 for A, as a consequence C=0
- example: backtracking, branch & bound
Domain Specific Methods

Specialized algorithms

Called **constraint solvers**

Examples:

- program for solving system of linear equations
- libraries for linear programming
- graph algorithm implementing special relation/constraint

Constraint programming

- broad term including many areas
- Artificial Intelligence, Linear Algebra, Global Optimization, Linear and Integer Programming, ...

Existence of domain specific methods ⇒ must replace general methods

- goal: find domain specific methods to replace general methods
Polynomial and NP-complete Problems

Polynomial problems
- there is a polynomial complexity algorithm solving the problem

NP-complete problem
- solvable by non-deterministic polynomial algorithm
- potential solution can be verified in polynomial time
- exponential complexity in the worst case (if P=NP does not hold)
Complexity: Polynomial Problems

Linear equalities over real numbers
- variables with domains from $\mathbb{R}$, constraints: linear equalities
- Gaussova eliminace
- polynomial complexity

Linear inequalities over real numbers
- linear programming, simplex method
- polynomial complexity is often sufficient
Complexity: NP-complete Problems

Boolean constraints

- 0/1 variables
- constraint $\equiv$ Boolean formula (conjunction, disjunction, implication, ...)
  - example: variables $A, B, C$, domains 0..1
    - constraints: $(A \lor B), (C \Rightarrow A)$
    - CSP: $(A \lor B) \land (C \Rightarrow A)$

- satisfiability problem of Boolean formula (SAT problem): NP-complete
- $n$ variables: $2^n$ possibilities

Constraints over finite domains

- general CSP
- satisfiability problem of general relations
- NP-complete problem
- $n$ variables, $d$ maximal domain size: $d^n$ possibilities
Complexity and Completeness

Complete vs. incomplete algorithms

- complete algorithm searches through the complete solution space
- incomplete algorithm: does not search the whole solution space
  - "don’t know" as a possible response, effectiveness could be the gain
- example:
  - incomplete polynomial algorithm for NP-complete problem

Solver complexity

- Gauss elimination (P), SAT solver (NP), general CSP solver (NP)

Constraint propagation algorithms

- mostly incomplete polynomial algorithms

Search algorithms

- complete algorithms, examples: backtracking, generate & test
- incomplete algorithms, example: randomized time limited searches
Constraint Programming: Modeling

4 Constraints

5 Examples

6 Exercises
  - Cryptogram
  - Timetable
  - Magic Sequence
  - Assignment Problem
Systems for Constraint Programming

- **ILOG, constraints in C++** 1987
  - commercial product, originally from France
  - implementation of constraints based on object oriented programming

- **Swedish Institute of Computer Science: SICStus Prolog** 1985
  - strong CLP(FD) library
  - commercial product, wide academic use

- **Choco, constraints in Java** 1999 (C++), 2003 (Java)
  - open-source software
  - development at: École des Mines de Nantes, Cork Constraint Computation Center, Bouygues e-lab, Amadeus SA

- For more see: **Constraint Programming Online**
Domain Representation

domain( Variables, Min, Max)

- example: domain( [A,B,C], 1,3).

Variable in Min..Max

- example: A in 1..3, B in -3..10

General representation: subset of intervals of integers

- \( l_1 \leq l_1u < l_2 \leq l_2u < \ldots < l_n \leq l_nu \)
- example: \( 1 \leq 3 < 8 \leq 10 < 20 \leq 20 < 30 \leq 33 \)
  - corresponds to values: 1,2,3, 8,9,10, 20, 30,31,32,33

Variable in Range

- example: A in (1..3) \( \setminus (8..10) \setminus \{20\} \setminus (30..33)\)
- B in (1..3) \( \setminus (8..15) \setminus (5..9) \setminus \{100\} \)
  - results for B: \( 1 \leq 3 < 5 \leq 15 < 100 \leq 100 \)
Arithmetic Constraints

Expr RelOp Expr

RelOp -> #= | #\=< | #< | #=< | #> | #>=

- examples: A + B #=< 3
- A #\=< (C - 4) * (D - 5)
- A/2 #= 4

sum(Variables,RelOp,Suma)

- example: domain([A,B,C,F],1,3), sum([A,B,C], #= ,F)

scalar_product(Coeffs,Variables,RelOp,ScalarProduct)

- example: domain([A,B,C,F],1,6),
  scalar_product([1,2,3],[A,B,C], #< ,F)
Global Constraints I.

Global Constraint Catalog: 313 constraints
- maintained by Beldiceanu & Carlsson & Rampon
- http://www.emn.fr/x-info/sdemasse/gccat

all_different( Variables )
- each variable must take unique value among variables
- example: domain([A,B,C],1,4), all_different([A,B,C]), A #!= 1, B #> 3
  results in: A=1, B=4, C in 2..3

table(Tuples, Extension)
- defines $n$-ary constraint by extension
- Extension: list of lists of integers, each of length $n$
- Tuples: list of lists of domain variables (or integers), each of length $n$
- example: table([A,B,C], [[1,2,3],[2,3,3]])
  results in: A in 1..2, B in 2..3, C=3
  table([A,B],[C,D], [[1,2],[2,3]])
  results in: A in 1..2, B in 2..3, C in 1..2, D in 2..3
Global Constraints II.

```
element(N,List,X)
  • N-th element of the List equals to X
  • A in 2..10, B in 1..3, element( N, [A,B], X ), X #< 2.
    results in: B = 1, N = 2, X = 1, A in 2..10

global_cardinality(List, KeyCounts)
  • for each member Key-Count of the list KeyCounts holds: Count members of the List equals to Key
  • each Key is integer and exists at most one among keys
  • A in 1..3, B in 1..3, global_cardinality( [A,B], [1-N,2-2]).
    results in: A = 2, B = 2, N = 0
```
Cryptogram

- Assign ciphers 0, ... 9 to letters S, E, N, D, M, O, R, Y such that

  \[
  \begin{align*}
  &\text{SEND} \\
  + &\text{MORE} \\
  \hline
  &\text{MONEY}
  \end{align*}
  \]

- different letters have assigned different ciphers
- S and M are not 0

- Unique solution:

  \[
  \begin{align*}
  &9567 \\
  + &1085 \\
  \hline
  &10652
  \end{align*}
  \]

- **Variables:** S, E, N, D, M, O, R, Y
- **Domains:** \( \text{domain}([S,M],1,9) \) \( \text{domain}([E,N,D,O,R,Y],0,9) \)
Cryptogram: equality constraints

1 equality constraint

\[
1000 \times S + 100 \times E + 10 \times N + D \quad \text{SEND} \\
+ \quad 1000 \times M + 100 \times O + 10 \times R + E \quad \text{MORE} \\
\#= 10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y \quad \text{MONEY}
\]

OR 5 equality constraints

use of "transfer" variables \( P_1, P_2, P_3, P_4 \) with domains \([0..1]\)

\[
D + E \#= 10 \times P_1 + Y, \\
P_1 + N + R \#= 10 \times P_2 + E, \\
P_2 + E + O \#= 10 \times P_3 + N, \\
P_3 + S + M \#= 10 \times P_4 + O \\
P_4 \#= M
\]
Cryptogram: unequality constraints

- 28 unequality constraints
  \[ X \neq Y \text{ for } X,Y \in \{S,E,N,D,M,O,R,Y\}, \ X \prec Y \]

- OR 1 unequality constraint
  \[ \text{all\_different([S,E,N,D,M,O,R,Y])} \]
N Queens

Place $N$ queens to $N \times N$ chessboard such that they do not attack each other.

- **Variables:** $X_1, \ldots, X_N$  
  each variable for each column

- **Domains:** $\text{domain}(X_1, \ldots, X_N, 1, N)$

- **Constraints:** for $i \in [1..(N - 1)]$ and $j \in [(i + 1)..N]$:
  - $X_i \#\not= X_j$ (rows) $\Rightarrow$ all different $([X_1, \ldots, X_N])$
  - $X_i - X_j \#\not= i - j$ (diagonal)
    - e.g. $X_1=1$ and $X_2=2$ prohibited ... diagonal through $(1,1)$ a $(2,2)$
  - $X_i - X_j \#\not= j - i$ (diagonal)
    - e.g. $X_1=3$ and $X_2=2$ prohibited ... diagonal through $(1,3)$ a $(2,2)$
Knapsack Problem

There is a knapsack of the size $M$ and there are $N$ articles of given size $S_i$ and cost $C_i$. Choose such articles to place into the knapsack to maximize the cost of articles in the knapsack.

- knapsack of the size $M$
- articles of the size $S_1, \ldots, S_N$ and cost $C_1, \ldots, C_N$

- **Variables:** $X_1, \ldots, X_N$
- **Domains:** $\text{domain}([X_1, \ldots, X_N], 0, 1)$
- **Constraint:**
  \[
  \text{scalar\_product}([S_1, \ldots, S_N], [X_1, \ldots, X_N], \#=<, M)
  \]
- **Objective function:**
  \[
  \text{scalar\_product}([C_1, \ldots, C_N], [X_1, \ldots, X_N], \#=, F)
  \]
- **Goal:** maximize($F$)
Exercises

Define domain variables with their domains and write a set of constraints together with possible optimization criteria to describe the model of particular constraint satisfaction problems in the following exercises.
Cryptogram

Assign different ciphers to letters such that the equation

\[ DONALD + GERALD = ROBERT \]

holds. Be careful that zero cannot be assigned to leading letters \( D, G, R \) to make the formula well readable.
Cryptogram: Solution

Variables:

- \text{domain}([O,N,A,L,E,B,T],0,9)
- \text{domain}([D,G,R],1,9)

Constraints:

- \text{all\_different}([D,G,R,O,N,A,L,E,B,T])
- \begin{align*}
& 100000 \times D + 10000 \times O + 1000 \times N + 100 \times A + 10 \times L + D \\
& + 100000 \times G + 10000 \times E + 1000 \times R + 100 \times A + 10 \times L + D \\
& \neq 100000 \times R + 10000 \times O + 1000 \times B + 100 \times E + 10 \times R + T
\end{align*}
Timetable

Find time for classes of given teachers such that each teacher is expected to teach within specified interval:

<table>
<thead>
<tr>
<th>teacher</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Jane</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Anne</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Yan</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Dave</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Mary</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

All classes are taught at the same room and take one hour.
Classes of women should be taught as earliest as possible.

Hint: all_different
Timetable: Solution

Variables for starting time of classes:

- Peter in 3..6, Jane in 3..4, Anne in 2..5, Yan in 2..4, Dave in 3..4, Mary in 1..6

Constraints:

- all_different([Peter, Jane, Anne, Yan, Dave, Mary])

Optimization:

- constraint: Jane + Anne + Mary ≠ F
- minimize(F)
Magic Sequence

A magic sequence of length $n$ is a sequence of integers $x_0 \ldots x_{n-1}$ between 0 and $n-1$, such that for all $i$ in 0 to $n-1$, the number $i$ occurs exactly $x_i$ times in the sequence.

For instance, 6,2,1,0,0,0,1,0,0,0 is a magic sequence since 0 occurs 6 times in it, 1 occurs twice, ...

Find magic sequence of the length $n$.

Solution:

$n=0,1,2$ a 5: no solution

$n=3$: [1,2,1,0] and [2,0,2,0]

$n=4$: [2,1,2,0,0]

$n=6$: [3,2,1,1,0,0,0]

for $n \geq 6$: $[n-3,2,1,\ldots,1,0,0,0]$

Hint: global_cardinality
Magic Sequence: Solution

M = N - 1 domain variables for a sequence of numbers:

- domain([X0,...XM], 0, M)

Constraints:

- global_cardinality([X0,X1,...,XM],[0-X0,1-X1,...,M-XM])

Additional (redundant) constraints to improve propagation:

- sum([X0,...XM],(M+1))
- scalar_product([0,1,...,M],[X0,X1,...,XM], #=,(M+1))
Assignment Problem

Assign four workers W1, W2, W3, W4 to four products such that each worker works on one product and each product is produced by one worker. Effectiveness of production is given by the following table (e.g., worker W1 produces P1 with effectiveness 7) and the total effectiveness must be 19 at least.

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>W2</td>
<td>8</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>W3</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>W4</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

Hint: element(W1, [7,1,3,4], EW1)
Assignment Problem: Solution

Variables for each worker representing a product to be produced by him:

- \text{domain}([W1, W2, W3, W4], 1, 4)

Constraints:

- all\_different([W1, W2, W3, W4])
- element(W1, [7, 1, 3, 4], EW1),
  element(W2, [8, 2, 5, 1], EW2),
  element(W3, [4, 3, 7, 2], EW3),
  element(W4, [3, 1, 6, 3], EW4)
- EW1 + EW2 + EW3 + EW4 \geq 19
Constraint Programming: Propagation

7 Representation of CSP

8 Arc Consistency

9 k-consistency

10 Non-binary Constraints

11 Bounds Consistency
Graph Representation of CSP

- **Constraint representation**
  - using mathematical/logical formula
  - by extension (list of compatible k-tuples, 0-1 matrix)

- **Graph**: nodes, edges (edge connects two nodes)
- **Hypergraph**: nodes, hyper-edges (hyper-edge connects set of nodes)

- CSP representation through **constraint hypergraph**
  - node = variable, hyper-edge = constraint

- **Example**
  - variables $X_1, \ldots, X_6$ with domain 0..1
  - constraints:
    - $c_1 : X_1 + X_2 + X_6 = 1$
    - $c_2 : X_1 - X_3 + X_4 = 1$
    - $c_3 : X_4 + X_5 - X_6 > 0$
    - $c_4 : X_2 + X_5 - X_6 = 0$
Binary CSP

Binary CSP

- CSP with binary constraints only
- unary constraints encoded into the domain

Constraint graph for binary CSP

- hypergraph not necessary
  graph sufficient (constraint defined on two nodes only)

Each CSP can be transformed to "corresponding" binary CSP

Binarization: pros and cons

- getting of unified CSP, many algorithms proposed for binary CSPs
- unfortunately significant increase of the problem size

Non-binary constraints

- more complex propagation algorithms
- semantics of constraints allows for stronger propagation
  example: all_different vs. set of inequality constraints
Node and Arc Consistency

Node consistency (NC)
- each value from the domain of variable $V_i$ satisfies all unary constraints over $V_i$

Arc consistency (AC) for binary CSP
- arc $(V_i, V_j)$ is arc consistent, iff there is a value $y$ for each value $x$ from the domain of $V_i$ such that the assignment $V_i = x, V_j = y$ satisfies all binary constraints over $V_i, V_j$.

- arc consistency is **directional**
  - consistency of arc $(V_i, V_j)$ does not guarantee consistency of $(V_j, V_i)$

\[
\begin{align*}
A & \xrightarrow{3..7} A < B & 1..5 & B \\
A & \xrightarrow{3..4} A < B & 1..5 & B \\
A & \xrightarrow{3..4} A < B & 4..5 & B
\end{align*}
\]
- CSP is arc consistent, iff all arcs (in both directions) are arc consistent
Arc Revision

- How to make arc \((V_i, V_j)\) arc consistent?
- Remove such values \(x\) from \(D_i\) that are inconsistent with current domain of \(V_j\) (there is no value \(y\) for \(V_j\) such that assignment \(V_i = x\) and \(V_j = y\) satisfies all binary constraints between \(V_i\) and \(V_j\))

procedure revise((\(V_i, V_j\)))
Deleted := false
for \(\forall x\) in \(D_i\) do
  if there is no \(y \in D_j\) such that \((x, y)\) is consistent
    then \(D_i := D_i - \{x\}\)
       Deleted := true
  end if
return Deleted
end revise

- domain([\(V_1, V_2]\),2,4), \(V_1 \neq < V_2\)
  revise((\(V_1, V_2\))) removes 4 from \(D_1\), \(D_2\) is not changed
Computation of Arc Consistency

How to make CSP arc consistent?

- revisions must be repeated if the domain of some variable was changed
- more effective: revisions can be done with the help of queue
  - we add to queue arcs which consistency could have been violated by value removal(s)

What arcs must be exactly revised after domain reduction of $V_k$?

- arcs $(V_i, V_k)$ leading to variable $V_k$ with reduced domain

  $V_{k} \leftarrow V_{m}$

- let $(V_k, V_m)$ caused domain reduction of $V_k$: then arc $(V_m, V_k)$ leading from $V_m$ may not be revised (change does not influence it)
AC-3 Algorithm

procedure AC-3(G)
Q := \{(V_i, V_j) \mid (V_i, V_j) \in \text{edges}(G), \ i \neq j\} % list of arcs to revise
while Q non empty do
    choose and remove (V_k, V_m) from Q
    if revise((V_k, V_m)) then % additions of arcs
        Q := Q \cup \{(V_i, V_k) \in \text{edges}(G), \ i \neq k, \ i \neq m\} % still not in queue
    end if
end while
end AC-3

Example:
A<B, B<C: (3..7,1..5,1..5) \rightarrow A(B,3..4,1..5,1..5) \rightarrow B(A,3..4,4..5,1..5)
                      \rightarrow B(C,3..4,4,1..5) \rightarrow C(B,3..4,4,5) \rightarrow (3,4,5)

AC-3 is the most common today but it is still not optimal!

Excercise: What will be domains of A,B,C after AC-3 for:

- domain([A,B,C],1,10), A \# = B + 1, C \#< B
Is it arc consistency sufficient?

AC removes many inconsistent values

- do we get solution of the problem then? NO
- do we know that solution of the problem exists? NO

  domain([X,Y,Z],1,2), X #\(\neq\) Y, Y #\(\neq\) Z, Z #\(\neq\) X

  - arc consistent
  - no solution exists

Why we use AC then?

- sometimes we obtain solution directly
  
    - some domain is emptied \(\Rightarrow\) no solution exists
  
    - all domains have one element only \(\Rightarrow\) we have solution

- general case: size of the solution search is decreased
**k-consistency**

Have NC and AC anything in common?

- NC: consistency of one variable
- AC: consistency of two variables
- ... an we can continue

CSP is **k-consistent** iff, any consistent assignment of \((k-1)\) variables can be extended to any \(k\)-th variable

k-consistency defined for general CSPs including non-binary constraints

\[
1,2,3 
eq 1,2,3 
eq 1,2,3 
eq 4
\]

4-consistent graph
Strong $k$-consistency

3-consistent graph

(1, 1) can be extended to (1, 1, 1)
(2, 2) can be extended to (2, 2, 2)
(1, 3) and (2, 3) are not consistent tuples (we do not extend those)

CSP is **strongly $k$-consistent** iff it is $j$-consistent for each $j \leq k$

- strong $k$-consistency $\Rightarrow$ $k$-consistency
- strong $k$-konzistence $\Rightarrow$ $j$-consistency $\forall j \leq k$
- $k$-consistency $\nRightarrow$ strong $k$-consistency

- NC = strong 1-consistency = 1-consistency
- AC = (strong) 2-consistency
Consistency for Finding of the Solution

If we have a graph with $n$ nodes, how strong consistency is necessary to obtain a solution directly?

- strong $n$-consistency is necessary for the graph with $n$ nodes
  - $n$-consistency does not suffice (see earlier example)
  - strong $k$-consistency for $k < n$ does not suffice too

Strong $n$-consistency is necessary for the graph with $n$ nodes

- exponential complexity!
Non-binary Constraints

k-consistency have exponential complexity, it is not used in practice

n-ary constraints are used directly

Constraint is **generally arc consistent (GAC)** iff for each variable $V_i$ from this constraint and for each its value $x \in D_i$, there is an assignment $y \in D_j$ for each remaining variable $V_j$ in the constraint such that it is satisfied

- $A + B = C$, $A$ in 1..3, $B$ in 2..4, $C$ in 3..7 is GAC

Semantics of constraints is used

- special type of consistency for global constraints
  - e.g. all_different

- bounds consistency can be used
  - propagation when the smallest or largest domain value changes

One constraint may use different types of consistency

- $A \#\neq B$: arc consistency, bounds consistency
Consistency Algorithm for Non-binary Constraints
Algorithm with queue of variables (sometimes also called AC-8)

procedure AC-8(Q)
while Q non empty do
    choose and remove $V_j \in Q$
    for \( \forall \) $c$ such that $V_j \in \text{scope}(c)$ do
        $W := \text{revise}(V_j, c)$
        // $W$ is the set of variables with changed domain
        if \( \exists \) $V_i \in W$ such that $D_i = \emptyset$ then return fail
    $Q := Q \cup \{W\}$
end AC-8

- revisions are repeated until there are domain changes
- $\text{scope}(c)$: set of variables on which $c$ is defined

Implementation
- set of constraints to be propagated maintained for each variable
- user defines \text{REVISE} procedures based on the constraint type
Bounds Consistency

Constraint is **bounds consistent (BC)** iff for each variable $V_i$ from this constraint and for each its value $x \in D_i$, there is an assignment of remaining variables in the constraint such that it is satisfied and $\min(D_i) \leq y_i \leq \max(D_i)$ holds for selected assignment $y_i$ of $V_i$

Bounds consistency: weaker than generalized arc consistency

Propagation when **minimal and maximal value (bounds)** changed only

**Bounds consistency for inequalities**

- $A \nRightarrow B \Rightarrow \min(A) \geq \min(B)+1, \max(B) \leq \max(A)-1$
- example: $A$ in 4..10, $B$ in 6..18, $A \nRightarrow B$
  \[
  \min(A) \geq 6+1 \Rightarrow A \text{ in 7..10}
  \]
  \[
  \max(B) \leq 10-1 \Rightarrow B \text{ in 6..9}
  \]
- and similar for: $A \nleftarrow B$, $A \nRightarrow\nRightarrow B$, $A \nLeftarrow\nLeftarrow B$
Bounds Consistency and Arithmetic Constraints

\[ A \neq B + C \Rightarrow \min(A) \geq \min(B) + \min(C), \max(A) \leq \max(B) + \max(C) \]
\[ \min(B) \geq \min(A) - \max(C), \max(B) \leq \max(A) - \min(C) \]
\[ \min(C) \geq \min(A) - \max(B), \max(C) \leq \max(A) - \min(B) \]

- change of \( \min(A) \) causes the change of \( \min(B) \) and \( \min(C) \) only
- change of \( \max(A) \) causes the change of \( \max(B) \) a \( \max(C) \) only, ...

Example:

A in 1..10, B in 1..10, A \#= B + 2, A \#> 5, A \#\leq 8

A \#= B + 2 \Rightarrow \min(A) \geq 1+2, \max(A) \leq 10+2 \Rightarrow A in 3..10

\Rightarrow \min(B) \geq 1-2, \max(B) \leq 10-2 \Rightarrow B in 1..8

A \#> 5 \Rightarrow \min(A) \geq 6 \Rightarrow A in 6..10

\Rightarrow \min(B) \geq 6-2 \Rightarrow B in 4..8 \quad \text{(new propagation for A \#= B + 2)}

A \#\leq 8 \Rightarrow A in (6..7) \lor (9..10)

(bounds same, no propagation for A \#= B + 2)
Exercises: Bounds and Arc Consistency

1. Define rules for bounds consistency of the constraints
   \[ A \#= B - C, A \#>= B + C \]

2. What are rules for bounds consistency of the constraint \( X \#= Y+5 \)?
   
   How propagations are processed in the following example?
   
   \( X \) in 1..20, \( Y \) in 1..20, \( X \#= Y + 5, Y \#> 10 \).

3. What is the difference between bounds and arc consistency? Show it on the example.

4. How arc consistency is achieved in the following example?
   
   \( \text{domain}([X,Y,Z],1,5]), X \#< Y, Z \#= Y+1 \)
Constraint Programming: Search

12 Depth First Search

13 Backtracking

14 Forward Checking

15 Looking Ahead

16 Summary and Exercises
Search & Consistency

Constraint satisfaction through search of the solution space
- constraints used passively as a test
- assign values and try what happens
- examples: backtracking, generate & test (trivial)
- complete methods (either solution is find or inconsistency is proved)
- too slow (exponential): search through "clearly" bad assignments

Consistency/propagation techniques
- allow to remove inconsistent values from domains
- incomplete methods (some inconsistent values still in domains)
- relatively fast (polynomial)

Combination of both methods used
- subsequent assignment of values to variables
- after assignment, inconsistent values removed by consistency techniques
Depth First Search (DFS)

Depth first search of the solution space:
base search algorithm for CSPs

Two phases of the search

- **forward phase**: variables subsequently selected, partial assignment extended by assignment of consistent value (if exists) to another variable
  - after value selection, consistency tests are processed
- **backward phase**: if there is no consistent value for current variable, algorithm backtracks to earlier assigned value

Types of variables

- **past**: already selected variable (have assigned value)
- **current**: currently selected variable to be assigned a value
- **future**: variables which will be selected in the future
Core search procedure: DFS

Variables numbered for simplicity, assignment processed in given order

Initial call: labeling(G,1)

procedure labeling(G, a)
if \( a > |\text{edges}(G)| \) then return \text{nodes}(G)
for \( \forall x \in D_a \) do
    if consistent(G, a) then % consistent(G,a) replaced by FC(G,a), LA(G,a), ...
        \( R := \text{labeling}(G, a + 1) \)
        if \( R \neq \text{fail} \) then return \( R \)
return fail
end labeling

R: assignment of variables or fail

Procedures consistent(G,i) will be described for binary constraints only
Backtracking (BT)

Backtracking verifies consistency of constraints leading from past variables to current variable at each step.

Backtracking maintains consistency of constraints:
- on all past variables
- on past and current variable

procedure BT(G,a)
Q:=\{(V_i, V_a) \in \text{edges}(G), i < a\} \quad \% edges from past to current variable
Consistent := true
while non empty Q \land \text{Consistent} do
    choose and remove any edge \((V_k, V_m)\) from Q
    Consistent := not revise\((V_k, V_m)\) \quad \% if value is removed, domain is empty!
return Consistent
end BT
Example: Backtracking

Constraints: $V_1, V_2, V_3$ in $1 \ldots 3$, \hspace{1cm} V_1 \# = 3 \times V_3

Solution space:

- red boxes: failed attempt for assignment, no solution
- empty circles: solution found
- black circles: inner node representing partial assignment
**Forward Checking (FC)**

FC is an extension of backtracking

In addition, FC maintains consistency between current and future variables

procedure FC(G,a)
Q:=\{(V_i, V_a) \in edges(G), i > a\} % addition of arcs from future to current variable
Consistent := true
while non empty Q \land Consistent do
    choose and remove any edge (V_k, V_m) from Q
    if revise((V_k, V_m)) then
        Consistent := (|D_k| > 0) % empty domain means fail
return Consistent
end FC

Edges from past to current variables is not necessary to test
Example: Forward Checking

Constraints: \( V_1, V_2, V_3 \) in 1 \ldots 3, \quad c : V_1\# = 3 \times V_3

Solution space:
Looking Ahead (LA)

LA is an extension of FC, LA maintains arc consistency
In addition, LA maintains consistency between all future variables

procedure LA(G,a)
Q := \{(V_i, V_a) \in edges(G), i > a\} % start with edges leading to a
Consistent := true
while non empty Q \land Consistent do
    choose and remove any edge (V_k, V_m) from Q
    if revise((V_k, V_m)) then
        Q := Q \cup \{(V_i, V_k)|(V_i, V_k) \in edges(G), i \neq k, i \neq m, i > a\}
        Consistent := (|D_k| > 0)
return Consistent
end LA

- edges from past variables to current variable are not necessary to test again
- this LA procedure is based on AC-3, other AC algorithms can be also applied

LA maintains arc consistency: since LA(G,a) applies AC-3,
initial consistency must be computed by AC-3 before search starts
Example: Looking Ahead (with AC-3)

Constraints: \( V_1, V_2, V_3 \) in 1 \ldots 4, \quad c1 : V_1 \# > V_2, \quad c2 : V_2 \# = 3 \times V_3

Solution space:

- initial consistency is computed (by AC-3 algorithm) before search

\[
\begin{align*}
&c1 \Rightarrow V_1 \text{ in } 2..4 \\
&V_2 \text{ in } 1..3 \\
&c2 \Rightarrow V_2 = 3 \\
&V_3 = 1 \\
&c1 \Rightarrow V_1 = 4
\end{align*}
\]
Example: Looking Ahead with AC-1

Constraints: $V_1, V_2, V_3$ in $1 \ldots 4$, \( c1 : V_1 \# > V_2 \), \( c2 : V_2 \# = 3 \times V_3 \)

Solution space (when AC-1 is used instead of AC-3):

- **initial consistency before search is not computed**
- AC-1 algorithm repeats revisions of all arcs in cycles unless domains of all variables are stable (do not change)

$\Rightarrow$ AC-1 makes the problem arc consistent as soon as the value of current variable is assigned

\[
\begin{array}{c}
\text{V1} \\
1 \quad \text{c1 => fail} \\
\quad \text{c2 => fail} \\
\text{V2} \\
2 \quad \text{c1 => V2 = 1} \\
\quad \text{c2 => fail} \\
\text{V3} \\
3 \quad \text{c1 => V2 = 1..2} \\
\quad \text{c2 => fail} \\
\text{V4} \\
4 \quad \text{c1 => V2 = 1..3} \\
\quad \text{c2 => V2 = 3} \\
\quad \text{V3 = 1}
\end{array}
\]
Summary of Algorithms

Backtracking (BT) maintains at step $a$ constraints
\[ c(V_1, V_a), \ldots, c(V_{a-1}, V_a) \]
from past variables to current variable

Forward checking (FC) maintains at step $a$ constraints
\[ c(V_{a+1}, V_a), \ldots, c(V_n, V_a) \]
from future variables to current variable

Looking Ahead (LA) maintains at step $a$ constraints
\[ \forall l (a \leq l \leq n), \forall k (a \leq k \leq n), k \neq l : c(V_k, V_l) \]
from future variables to current variable
between future variables
Exercise 1.

1. Write solution space for constraints
A in 1..4, B in 3..4, C in 3..4, c1: B ≠< C, c2: A ≠= C
when using forward checking and ordering of variables A,B,C. Explain what
types of propagation happens in each node.

Solution:
Exercise 2.

2. Write solution space for constraints
A in 1..4, B in 3..4, C in 3..4, c1: B \#< C, c2: A \#= C
when using looking ahead and ordering of variables A,B,C. Explain what types of propagation happens in each node.

Solution: (initial propagation computed by AC-3)
Exercises 3. and 4.

3. Write and compare solution spaces for constraints domain([A,B,C],0,1), c1: A ≠ B-1, c2: C ≠ A*A when using backtracking and looking ahead. Explain what types of propagation happens in each node.

Solution:

4. Present on some example differences between forward checking and looking ahead.
Constraint-based Scheduling: Introduction

17 Scheduling

18 CSP model

19 Resources

20 Optimization
Materials on Constraint-based Scheduling (CBS)

Some topics are described in


- 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

Scheduling

- Scheduling
  - optimal resource allocation of a given set of activities in time
    - resource or machine
    - activity or task

- Machine $M_j, j = 1, \ldots, 3$
- Task $T_i, i = 1, \ldots, 9$

Machine–oriented Gantt chart

- $M_1$
  - $T_1$
  - $T_3$
  - $T_5$
  - $T_4$

- $M_2$
  - $T_2$
  - $T_8$
  - $T_7$

- $M_3$
  - $T_6$
  - $T_9$

0 1 2 3 4 5 6 7 8  

Time
Example: Bicycle Assembly

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

![Activity Diagram]

Schedule:

<table>
<thead>
<tr>
<th>Task</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_4$</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$T_6$</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>$T_8$</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>$T_{10}$</td>
<td>14</td>
<td>24</td>
</tr>
<tr>
<td>$T_5$</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$T_3$</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>$T_9$</td>
<td>14</td>
<td>24</td>
</tr>
<tr>
<td>$T_1$</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>$T_7$</td>
<td>16</td>
<td>32</td>
</tr>
</tbody>
</table>

Optimal schedule:

<table>
<thead>
<tr>
<th>Task</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_4$</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$T_5$</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>$T_6$</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>$T_3$</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>$T_8$</td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td>$T_9$</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>$T_{10}$</td>
<td>24</td>
<td>32</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Course</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periods</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td>8</td>
<td>3</td>
<td>2</td>
<td></td>
<td>7</td>
<td>4</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

Demands:

<table>
<thead>
<tr>
<th>Periods</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Room 1</td>
<td>D</td>
<td>D</td>
<td>C</td>
<td>C</td>
<td>F</td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Room 2</td>
<td>I</td>
<td>I</td>
<td>E</td>
<td>E</td>
<td>G</td>
<td>G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Room 3</td>
<td>H</td>
<td>H</td>
<td>J</td>
<td>K</td>
<td>K</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Room 4</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
<td>M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Room 5</td>
<td>A</td>
<td>L</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solution = Schedule/Timetable:
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

- **StartA**: start time of the activity
  - activity cannot start before its **release date**
  - \( \text{est}(A) = \min(\text{StartA}) \), earliest start time

- **EndA**: completion time of the activity
  - activity must finish before the **deadline**
  - \( \text{lct}(A) = \max(\text{EndA}) \), latest completion time

- **PA**: processing time (duration) of the activity
  - \( \text{StartA} = \{\text{est}(A), \ldots, (\text{lct}(A)-\text{PA})\} \)
  - \( \text{EndA} = \{(\text{est}(A)+\text{PA}), \ldots, \text{lct}(A)\} \)
Scheduling as a CSP: basic constraints I.

- **Non-preemptive activity:** no interruption during processing
  - \( \text{StartA} + \text{PA} \neq \text{EndA} \)

- **Preemptible activity:** can be interrupted during its processing
  - \( \text{StartA} + \text{PA} \neq < \text{EndA} \)

\[
\begin{align*}
\text{PA} &= \text{PA}[1] + \text{PA}[2] + \text{PA}[3] + \text{PA}[4]
\end{align*}
\]
Scheduling as a CSP: basic constraints II.

- **Sequencing** $A \ll B$ of activities $A, B$
  - (also: *precedence constraint* between activities $A, B$)
  - $\text{End}_A \neq < \text{Start}_B$

![Diagram](image)

- **Disjunctive constraint:** non-overlapping of activities $A, B$
  - non-preemptive activities
  - $A \ll B$ or $B \ll A$
  - $(\text{End}_A \neq < \text{Start}_B) \neq \lor (\text{End}_B \neq < \text{Start}_A)$
  - related with the idea of unary resource
Domain variables for resources

- **CapA**: requested capacity of the resource
  - unary resources
  - cumulative resources
  - producible/consumable resources
- **ResourceA**: alternative resources for A
Unary (disjunctive) resources

- Each activity requests unary capacity of the resource: $\text{CapA}=1$
- 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

$$\text{cumulative}([\text{task}(\text{StartA}_1, \text{PA}_1, \text{EndA}_1, 1, \text{A}_1), \ldots, \text{task}(\text{StartA}_n, \text{PA}_n, \text{EndA}_n, 1, \text{A}_n)], \text{Options})$$

$A_1, \ldots, A_n$: activity identifiers

Options: options for different propagation algorithms

4th parameter of the task $= 1$: unit consumption of the resource
Cumulative (discrete) resources

- Each activity uses some capacity of the resource $\text{CapA}$
- Several activities can be processed in parallel if a resource capacity is not exceeded

Example: multi-processor computer with parallel jobs

```plaintext
cumulative([task(StartA1,PA1,EndA1,CapA1,A1), ..., task(StartAn,PAn,EndAn,CapAn,An)], [limit(L)|Options])
```

A1, ..., An: activity identifiers

- limit(L): available capacity of the resource is L
- Options: options for different propagation algorithms
Producible/consumable resources

- Resource = reservoir
- **Activity consumes some quantity** of the resource $\text{CapA} < 0$ or **activity produces some quantity** of the resource $\text{CapA} > 0$
- 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
Alternative resources

- Activity can be processed on a set of alternative resources
  - defined by the domain variable ResourceA

- 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
Optimization
Various criteria and objective function

Common criteria: **makespan**

- completion time of the last activity
- modeling
  - introduced a new additional activity L, PL=0
  - added precedence constraint
    for each activity T with no successor: T « L

![](image)

- makespan = StartL
Constraint-based Scheduling: Modeling

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

22 Timetabling

23 Employees Scheduling

24 Exercises
- Operator Scheduling
- Meeting Scheduling
- Scheduling of Computational Jobs
- Room Assignment
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

<table>
<thead>
<tr>
<th>Task T</th>
<th>est(T)</th>
<th>lct(T)</th>
<th>PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>10</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
Machine scheduling with unary resource: variables

- Start time variables \( \text{StartT} \) for each task \( T \)
- \( \text{lst}(T) = \text{lct}(T) - PT \)
- \( \text{StartT} \) in \( \text{est}(T) .. \text{lst}(T) \)

Example:

<table>
<thead>
<tr>
<th>Task T</th>
<th>est(T)</th>
<th>lct(T)</th>
<th>PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>10</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

\( \text{StartA} \) in 0..8, \( \text{StartB} \) in 0..12, \( \text{StartC} \) in 5..21,
\( \text{StartD} \) in 0..19, \( \text{StartE} \) in 10..20, \( \text{StartF} \) in 0..2
Machine scheduling with unary resource: constraints

Precedence constraints for each tasks $T_1 \prec T_2$

- $\text{Start}T_1 + \text{PT}_1 \preceq \text{Start}T_2$
- $\text{Start}A + 2 \preceq \text{Start}B$, $\text{Start}B + 3 \preceq \text{Start}C$
- $\text{Start}F + 3 \preceq \text{Start}E$, $\text{Start}E + 5 \preceq \text{Start}C$, $\text{Start}D + 1 \preceq \text{Start}C$

Unary resource for all tasks $T$ given by

- start time variables $\text{Start}T$
- end time variables $\text{End}T \preceq \text{Start}T + \text{PT}$

$cumulative([\text{task}(\text{Start}A, 2, \text{End}A, 1, A), \ldots, \text{task}(\text{Start}F, 3, \text{End}F, 1, F)], \text{Options})$

<table>
<thead>
<tr>
<th>Task $T$</th>
<th>$est(T)$</th>
<th>$lct(T)$</th>
<th>$PT$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>10</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
Scheduling with unary resource: optimization

- New task L with PL=0 added
- Precedence constraints between L and tasks with no successor added
- Example: \( \text{StartC} + 4 \neq < \text{StartL} \)

\[
\text{minimize(Makespan)} = \text{minimize(StartL)}
\]
Scheduling with unary resource: solution

Solution:

<table>
<thead>
<tr>
<th>Task T</th>
<th>est(T)</th>
<th>lct(T)</th>
<th>PT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>10</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Task T</th>
<th>est(T)</th>
<th>lct(T)</th>
<th>PT</th>
<th>CapT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>15</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>25</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>20</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>10</td>
<td>25</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
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 StartT
- duration PT
- requested capacity of the resource
- example: cumulative([task(StartA,2,EndA,1,A), ..,
  task(StartF,3,EndF,2,F)], [limit(3)|Options])

<table>
<thead>
<tr>
<th>Task T</th>
<th>est(T)</th>
<th>lct(T)</th>
<th>PT</th>
<th>CapT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>15</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>25</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>20</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>10</td>
<td>25</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
Scheduling with cumulative resource: solution

Solution:

<table>
<thead>
<tr>
<th>Task</th>
<th>est(T)</th>
<th>lct(T)</th>
<th>PT</th>
<th>CapT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>15</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>25</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>20</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>10</td>
<td>25</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
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:
  P31=4, P21=2, P11=1
  P12=3, P32=3
  P23=2, P13=4, P33=1

Additional first and last activities O and L with P0 = PL = 0
Job-shop problem: modeling

Variables and domains
- $\text{Start}_{IJ}$ start time variables for $J$ task running on machine $I$

Constraints
- ordering of jobs modeled through precedence constraints
  $\text{Start}_{31} + P_{31} \leq \text{Start}_{21}$, $\text{Start}_{21} + P_{21} \leq \text{Start}_{11}$, ...
- unary resource constraint for each machine $I$
  with jobs $(I,J)$ for all tasks $J$
  1st machine: $\text{cumulative([task(Start_{11},1,End_{11},1,11)},$
  $\text{task(Start_{12},3,End_{12},1,12)}, \text{task((Start_{13},4,End_{13},1,13)}],\text{Options})$, 
  2nd machine: $\text{cumulative(...)}$, 3d machine: $\text{cumulative(...)}$

Optimization
- precedence constraints: $\text{Start}_{11} + 1 \leq \text{Start}_L$, $\text{Start}_{32} + 3 \leq \text{Start}_L$, $\text{Start}_{33} + 1 \leq \text{Start}_L$
- minimize($\text{Makespan}) = \text{minimize(Start}_L)$
Job-shop problem: solution

Optimal solution:

M3
M2
M1

L0
0 0
4 2 1
3 3
2 4 1

Hana Rudová (FI MU, CR)  Constraint Programming and Scheduling  May 2009  93 / 132
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 $\text{StartA}$
  - $\text{StartA}$ in $0..(N-1)$ (N number of periods)
  - for each prohibited time period $i$ of the class A:
    - $\text{StartA} \neq \text{ProhibitedAi}$

- Classroom represents 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 $\text{ResourceA}$
  - $\text{ResourceA}$ in $K..(M-1)$ 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: $\text{ResourceA}$ in 0..3
    - class B wants a room of the size 40: $\text{ResourceB}$ in 2..3
    - class C wants a room of the size 80: $\text{ResourceC} = 3$
Class Timetabling: resource constraints

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 StartA and PA variables
- EndA ≠ StartA+PA
- for each teacher I and all his classes I1,...:
  cumulative([task(StartI1,PI1,EndI1,1,I1), ...], Options)

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 StartA and PA variables
- EndA ≠ StartA+PA
- for each curriculum J and all its classes J1,...:
  cumulative([task(StartJ1,PI1,EndJ1,1,J1), ...], Options)
Class Timetabling: time and classrooms

Constraint:

- Each time at most one course must be taught at any classroom.
- All classrooms together represent one unary resource.
- All classes request this resource.
- Each class is encoded by an activity with the starting time
  \[ \text{StartResourceA} \#= \text{StartA} + \text{ResourceA} \times N \]
  and duration \( PA \)

For all classes \( A_1, \ldots \):

\[
\text{cumulative([task(StartResourceA1,PA1,(StartResourceA1+PA1),1,A1),\ldots],Options)}
\]
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 types 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
- Working shifts: 4-6
- Cost – CostM=10, CostA=11, CostN=13

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Mo</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>M</td>
<td>M</td>
<td>N</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Paul</td>
<td>A</td>
<td>A</td>
<td>-</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>Mary</td>
<td>N</td>
<td>-</td>
<td>M</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Employees scheduling: variables & domains

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..4
Global cardinality constraint (revision)

Global constraint \texttt{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
Minimal and maximal number constraints I.

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Mo</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>M</td>
<td>M</td>
<td>N</td>
<td>F</td>
<td>...</td>
</tr>
<tr>
<td>Paul</td>
<td>A</td>
<td>A</td>
<td>F</td>
<td>M</td>
<td>...</td>
</tr>
<tr>
<td>Mary</td>
<td>N</td>
<td>F</td>
<td>M</td>
<td>A</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Minimal and maximal number of employees per shift

- new domain variables representing these numbers
  M1 in MinM1..MaxM1, A1 in MinA1..MaxM1, N1 in MinN1..MaxN2
- global cardinality constraint for each day
  \[\text{global_cardinality}( [\text{PeterMo},\text{PaulMo},\text{MaryMo},\ldots], [1-M1,2-A1,3-N1] ) \]

Minimal and maximal number of shift types per employee

- new domain variables representing these numbers
  M2 in MinM2..MaxM2, A2 in MinA2..MaxM2, N2 in MinN2..MaxN2
- global cardinality constraint for each employee
  \[\text{global_cardinality}( [\text{PeterMo},\text{PeterTue}.,\ldots,\text{PeterSun}], [1-M2,2-A2,3-N2] ) \]
Minimal and maximal number constraints II.

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

- MinF = Days - MaxW \ldots \text{minimal number of free time shifts}
- MaxF = Days - MinW \ldots \text{maximal number of free time shifts}

Example

- Days=7, MaxW=6 \Rightarrow MinF=1
- Days=7, MinW=4 \Rightarrow MaxF=3

- new domain variable \( F \) in MinF..MaxF
- \textbf{global_cardinality( [PeterMo,PeterTue,...,PeterSun], [4-F] )}

- better to add to "Minimal and maximal number of shift types per employee" global cardinality constraint
  \[ \text{global_cardinality([PeterMo,PeterTue,...,PeterSun], [1-M2,2-A2,3-N2,4-F])} \]
Cost minimization

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Mo</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>M</td>
<td>M</td>
<td>N</td>
<td>F</td>
<td>...</td>
</tr>
<tr>
<td>Paul</td>
<td>A</td>
<td>A</td>
<td>F</td>
<td>M</td>
<td>...</td>
</tr>
<tr>
<td>Mary</td>
<td>N</td>
<td>F</td>
<td>M</td>
<td>A</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **M1, A1, N1** represent number of particular shifts on Monday
- All M1 variables for particular days can be summarized into M and same for A1 and A, N1 and N
  
  \[ \text{sum}([M1\text{Mon}, M1\text{Tue}, \ldots, M1\text{Sun}], \#= , \text{CostM}), \ldots \]

- Total schedule cost corresponds to
  
  \[ \text{Cost } \#= M \times \text{CostM} + A \times \text{CostA} + N \times \text{CostN} \]

Alternatively

- **M2, A2, N2** represent number of particular shifts for Peter
- All M2 variables for all employees can be summarized into M ... and same for A2 and A, N2 and N
Exercises

Define domain variables with their domains and write a set of constraints together with possible optimization criteria to describe the model of particular constraint satisfaction problems in the following excercises.
There are N tasks and M operators. Tasks should be scheduled within 4 weeks, each week starts on Monday and ends on Friday. Each task I has specified constant number of days DI to be processed. Find starting day of all tasks with given prerequisites:

1. Each day can be processed Limit tasks at most.
2. Each operator J has a list of tasks J1,...,JKJ to work on. In addition, each operator can process only one task at any time.
3. Each task I has specified weak WI to be completed at the latest.
4. Last task N must processed after completion of three first tasks 1,2,3.
Operator Scheduling: Solution

Variables for starting times:

- domain([T1,...,TN],1,20)

Constraints:

1. cumulative([task(T1,D1,(T1+D1),1,1),...,task(TN,DN,(TN+DN),1,N],
   [limit(Limit)|Options])

2. M constraints:
   cumulative([task(TJ1,DJ1,(JT1+JD1),1,1),...,task(TJKJ,DJKJ,(TKJ+DKJ),1,KJ], Options)

3. N constraints: T1+D1 #=< WI*5

4. T1+D1 #=< TN, T2+D2 #=< TN, T3+D3 #=< TN
Meeting Scheduling

Find time and room for \( N \) meetings and \( K \) persons. Meetings will run from 8:00 to 17:00 and each meeting \( J \) has specified its constant duration \( DJ \).

1. Meetings will take place in \( M \) rooms. One meeting can place at each time and room at most.
2. Each person \( I \) will attend 4 meetings \( I1, I2, I3, I4 \).
3. Meeting \( A \) must take before meeting \( B \).
4. Meeting \( C \) must take after meeting \( D \).
5. There are no meetings during lunch time from 12:00 to 13:00.
Meeting Scheduling: Solution

Variables for starting time and room:

- domain([T1,...,TN], 0, 8)
  0 corresponds to 8:00, 8 corresponds to 16:00
- domain([R1,...,RN], 0, M-1)

Constraints:

1. variable for each meeting l: TimeRooml #= RI*9 + TI
   cumulative([task(TimeRoom1,D1,TimeRoom1+D1,1,1),..., task(TimeRoomN,DN,TimeRoomN+DN,1,N)], Options)

2. K constraints:
   cumulative([task(T1,I1,T1+I1,1,I1),..., task(T4,I4,T4+I4,1,I4)], Options)

3. TA+DA #=< TB

4. TD+DD #=< TC

5. N constraints: TI #\leq 4 (4 corresponds to 12:00)
Scheduling of Computational Jobs

There are N jobs to be processed on M processor cluster. Find starting time of all jobs given duration of the job I equal to DI and required number of processors by the job RI.

1. Each processor can process one job at any time.

2. Each job J requires memory EJ specified in GB. Any time the total allocated memory must be smaller or equal to 128GB available on the cluster.

3. Some jobs needs output of other jobs, so that they needs to be processed after completion of given jobs. Specifically, job U uses output of jobs V and job V uses output of W. In addition, job R uses output of S and T.

Also completion time of all jobs must be minimized.
Scheduling of Computational Jobs: Solution

Variables for starting time:

- \( \text{sum}([D_1, \ldots, D_N], \# = \text{Sum}), \text{domain}([T_1, \ldots, T_N], 0, (\text{Sum}-1)) \)
  - starting time of all jobs will be certainly smaller than sum of their durations

Constraints:

1. \( \text{cumulative}([\text{task}(T_1, D_1, (T_1+D_1), R_1, 1), \ldots, \text{task}(T_N, D_N, (T_N+D_N), R_N, N], [\text{limit}(M) | \text{Options}]) \)

2. \( \text{cumulative}([\text{task}(T_1, D_1, (T_1+D_1), E_1, 1), \ldots, \text{task}(T_N, D_N, (T_N+D_N), E_N, N], [\text{limit}(128) | \text{Options}]) \)

3. \( TU \ #\geq TV+DV, \ TV \ #\geq TW+DW \)
   \( TR \ #\geq TS+DS, \ TR \ #\geq TT+DT \)

Optimization:

- N constraints: \( \text{End} \ #\geq TI+DI \)
- minimize(End)
Room Assignment

Find rooms for 6 courses A,B,C,D,E,F taking from 9 am to 4 am. Each course consists of several lectures and each lecture takes one hour. Each lecture has specified its time (starting time of all lectures for each course is given in the table). All lectures of one course must be at the same room. There are three rooms available and one lecture can be at any room at most.

<table>
<thead>
<tr>
<th>Course/Time period</th>
<th>9 am</th>
<th>10 am</th>
<th>11 am</th>
<th>12 am</th>
<th>1 pm</th>
<th>2 pm</th>
<th>3 pm</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>+</td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td>+</td>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
Room Assignment: Solution

Domain variables:

- domain([RoomA, RoomB, RoomC, RoomD, RoomE, RoomF], 1, 3)

Constraints:

- 10 am: all_different([RoomA, RoomB, RoomC])
- 11 am: all_different([RoomA, RoomB, RoomC])
  
  (not necessary, same as for 10 am)
- 12 am: all_different([RoomA, RoomD, RoomE])
- 1 pm: all_different([RoomD, RoomE, RoomF])
- 2 pm: RoomD ≠ RoomF (not necessary, included in 1 pm)
25 Unary resources

26 Cumulative resources
Notation

- start(A) domain variable for starting time of activity A
- end(A) domain variable for completion time of activity A
- p(A) domain variable for processing time of activity A
- 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
- \( p(Ω) = \sum_{A \in Ω} p(A) \)
- \( est(Ω) = \min \{ est(A) \mid A \in Ω \} \)
- \( lct(Ω) = \max \{ lct(A) \mid A \in Ω \} \)
Unary Resource for Unit Time Activities: all\_different

\[ U = \{X2, X4, X5\}, \ \text{dom}(U) = \{2, 3, 4\}: \]
\[ \{2,3,4\} \text{ impossible for } X1, X3, X6 \Rightarrow \]
\[ X1 \text{ in } 5..6, \ X3 = 5, \ X6 \text{ in } \{1\} \backslash (5..6) \]

Consistency:
\[ \forall \{X_1, \ldots, X_k\} \subset V: \text{card}\{D_1 \cup \cdots \cup D_k\} \geq k \]

Inference rule
- \[ U = \{X_1, \ldots, X_k\}, \ \text{dom}(U) = \{D_1 \cup \cdots \cup D_k\} \]
- \[ \text{card}(U) = \text{card}(\text{dom}(U)) \Rightarrow \forall v \in \text{dom}(U), \forall X \in (V - U), X \neq v \]
- values in \text{dom}(U) unavailable for other variables

Complexity
- \[ O(2^n) \text{: search through all subsets of the set of } n \text{ variables} \]
- \[ O(n \log n) \text{: changes of bounds propagated only (1998)} \]
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

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

\[ \begin{array}{c}
\text{4} \quad \text{6} \quad \text{16}
\end{array} \]

\[ \begin{array}{c}
A(2) \quad B(4) \quad C(5)
\end{array} \]

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

\[ \begin{array}{c}
\text{4} \quad \text{7}
\end{array} \]

\[ \begin{array}{c}
A(2) \quad B(4) \quad C(5)
\end{array} \]
Edge finding: all filtering rules

- Edge-finding rules
  - \( p(\Omega \cup \{A\}) > lct(\Omega \cup \{A\}) - est(\Omega) \)
    \[ \Rightarrow A << \Omega \]
  - \( A << \Omega \Rightarrow \)
    \[ end(A) \leq \min\{lct(\Omega') - p(\Omega') | \Omega' \subseteq \Omega\} \]

- Edge-finding (symmetrical) rules
  - similar rules for \( \Omega << A \) and \( \text{start}(A) \)

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

- How aggregated demand is constructed?
Timetable constraint

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

\[ \forall t \sum_{\text{start}(A_i) \leq t \leq \text{end}(A_i)} \text{cap}(A_i) \leq \text{MaxCapacity} \]

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

\[ \forall t \sum_{A_i} X(A_i, t) \times \text{cap}(A_i) \leq \text{MaxCapacity} \]

\[ \forall t, i \ \text{start}(A_i) \leq t < \text{end}(A_i) \Leftrightarrow X(A_i, t) \]
Timetable constraint: filtering example

Initial situation

Some positions forbidden due to capacity

New situation

Hana Rudová (FI MU, CR)  Constraint Programming and Scheduling  May 2009 123 / 132
Search

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

- $X$ in 1..5 $\equiv X=1 \lor X=2 \lor X=3 \lor X=4 \lor X=5$

Generally search algorithm solves remaining disjunctions

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

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
Search: value ordering

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 schemes in scheduling

**Branching** = resolving disjunctions

**Traditional scheduling approaches**

- take the **critical decisions first**
  - resolve bottlenecks, . . .
  - defines the **shape** of the search treee
  - 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"

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

- 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

\[ A \ll B \lor \neg A \ll B \]

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

\[(A << \Omega \lor \neg A << \Omega) \lor (\Omega << A \lor \neg \Omega << A)\]

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 \[\min(start(A))\] 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 a minimal slack (bottleneck) preferred

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