CONSTRAINT PROGRAMMING APPROACH TO AI APPLICATIONS

Michela Milano

DEIS, Università di Bologna

mmilano@deis.unibo.it

Tutorial 3: AI*IA99 Bologna September 99

OVERVIEW

- Constraint Satisfaction (Optimization) Problems
- Constraint (Logic) Programming
  - *language and tools*
- AI Applications: modelling and solving
  - *Scheduling - Timetabling - Resource Allocation*
  - *Routing*
  - *Packing - Cutting*
  - *Graphics - Vision*
  - *Planning*
- Advantages and Limitations of CP: extensions

OVERVIEW

- Constraint Satisfaction (Optimization) Problems
- Constraint (Logic) Programming
  - *language and tools*
- AI Applications: modelling and solving
  - *Scheduling - Timetabling - Resource Allocation*
  - *Routing*
  - *Packing - Cutting*
  - *Graphics - Vision*
  - *Planning*
- Advantages and Limitations of CP: extensions

CONSTRAINT SATISFACTION PROBLEMS

- A CSP consists of:

– *a set of variables* (V_1, V_2…V_n)

– *a discrete domain* (D_1,D_2…D_n) *for each variable*

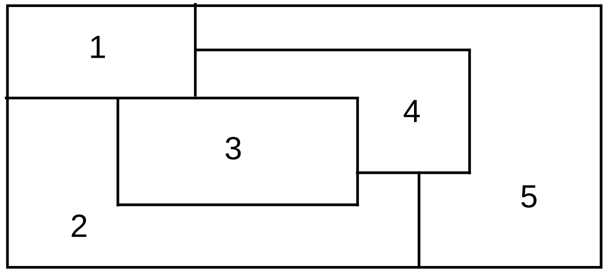
– *a set of constraints on those variables* - relations among variables which represent a subset of the Cartesian product of the domains $D_1 x D_2 x ... x D_n$

Binary constraints are posted between couples of variables

Solution of a CSP: an assignment of values to variables consistent with problem constraints

E.Tsang: "Foundations of Constraint Satisfaction" Academic Press, 1992.

EXAMPLE: Map Coloring



- Aim: find an assignments of colours to zones s.t. no two adjacent zones are coloured with the same colour

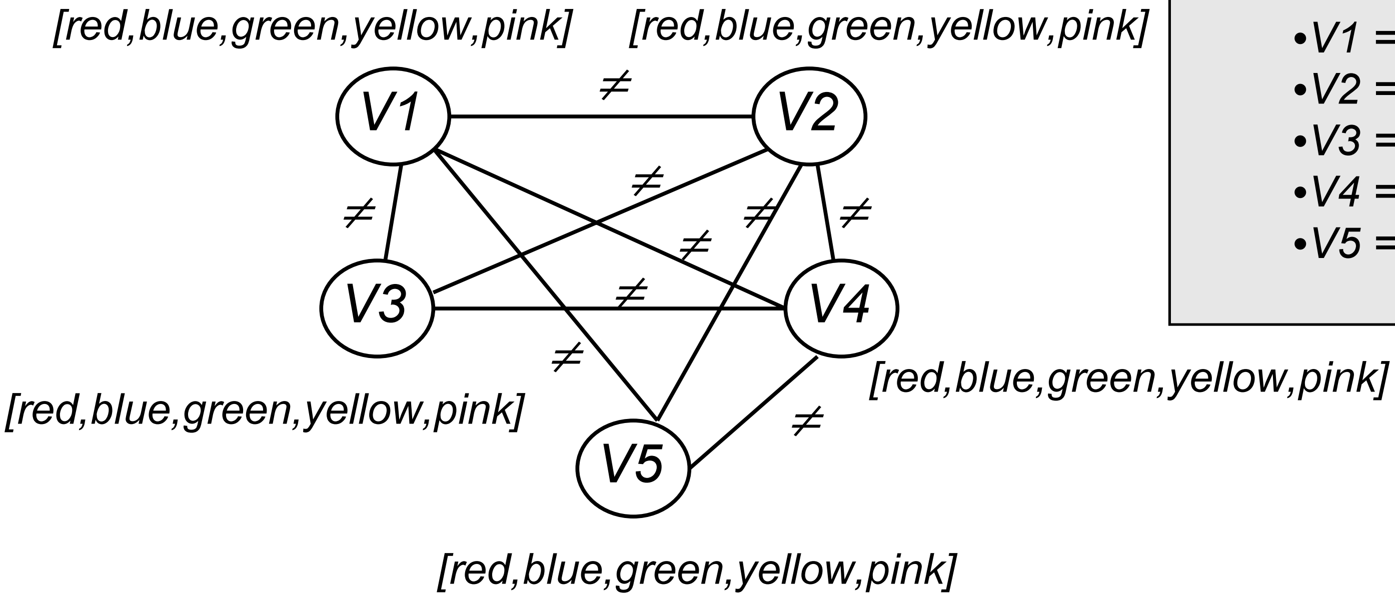
– *variables V1, V2, V3, V4, V5: zones*

– *domains D1, D2, D3, D4, D5: [red, blue, green, yellow,pink]*

– *constraints: near(Vi, Vj)* $\Rightarrow$ *Vi* $\neq$ *Vj*

CONSTRAINT GRAPHS

- A CSP can be represented by a constraint graph:
  - *variables* ⟺ *nodes*
  - *constraints* ⟺ *(hyper)-arcs*


[red,blue,green,yellow,pink]
[red,blue,green,yellow,pink]
V1
V2
≠
≠
≠
≠
≠
≠
V3
V4
≠
≠
[red,blue,green,yellow,pink]
[red,blue,green,yellow,pink]
≠
V5
[red,blue,green,yellow,pink]

- *Feasible Solution:*
  - *V1 = red*
  - *V2 = green*
  - *V3 = blue*
  - *V4 = yellow*
  - *V5 = pink*

CONSTRAINT OPTIMIZATION PROBLEMS

- A COP consists of:

– *a set of variables* (V_1, $V_2 \ldots V_n$)

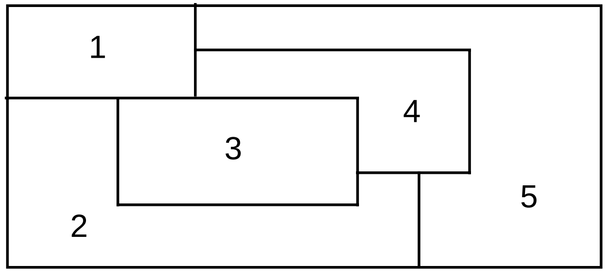
– *a discrete domain* (D_1,$D_2 \ldots D_n$) *for each variable*

– *a set of constraints on those variables* - relations among variables which represent a subset of the Cartesian product of the domains $D_1 x D_2 x \ldots x D_n$

– *an objective function* $f(V_1, V_2 \ldots V_n)$

Solution of a COP: an assignment of values to variables consistent with problem constraints, which optimizes the objective function

EXAMPLE: Map Coloring

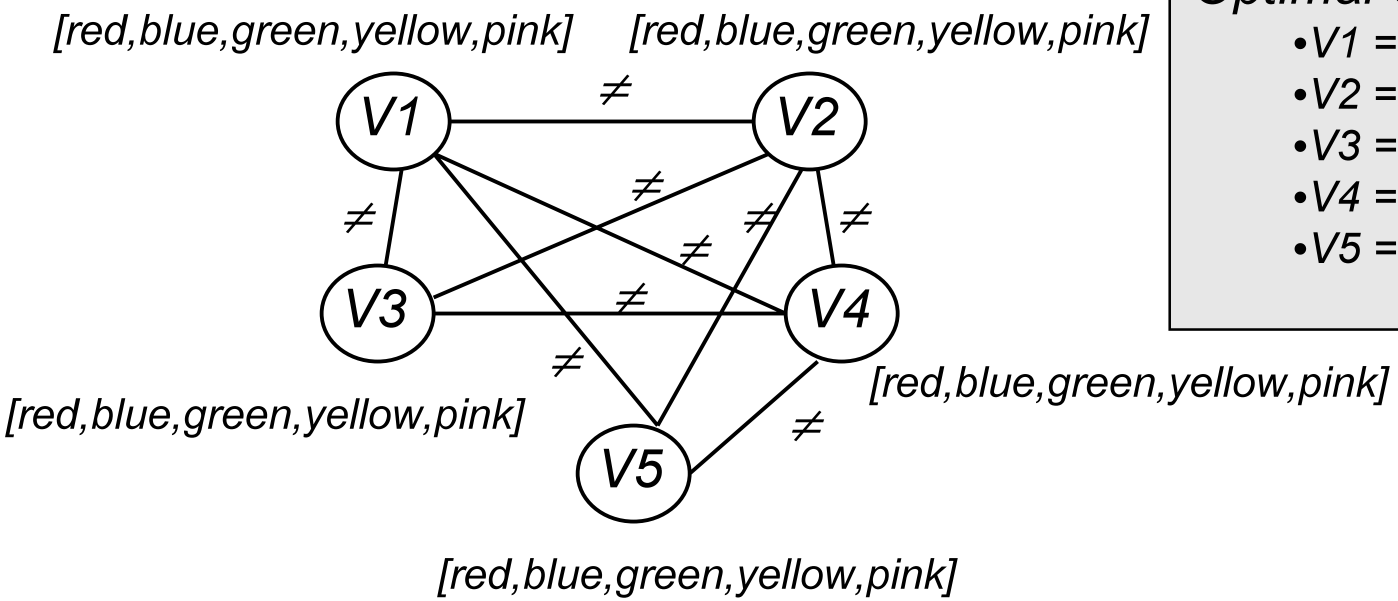


- Aim: find an assignments of colours to zones s.t. no two adjacent zones are coloured with the same colour MINIMIZING the number of colours used

– *variables V1, V2, V3, V4, V5: zones*
– *domains D1, D2, D3, D4, D5: [red, blue, green, yellow,pink]*
– *constraints: near(Vi, Vj)* $\Rightarrow$ *Vi* $\neq$ *Vj*

CONSTRAINT GRAPHS

- A COP can be represented by a constraint graph:
  - *variables* ⟺ *nodes*
  - *constraints* ⟺ *(hyper)-arcs*


[red,blue,green,yellow,pink]
[red,blue,green,yellow,pink]
V1
V2
V3
V4
V5
≠
≠
≠
≠
≠
≠
≠
≠
≠
≠
[red,blue,green,yellow,pink]
[red,blue,green,yellow,pink]
[red,blue,green,yellow,pink]

- *Optimal Solution:*
  - *V1 = red*
  - *V2 = green*
  - *V3 = blue*
  - *V4 = yellow*
  - *V5 = blue*

CONSTRAINT MODELLING

- Properties of constraints
  - *Declarative (invariant) relations among objects*
    - *X > Y*
  - *Addictive: the order of imposition does not matter*
    - *X + Y <= Z, X + Y >=Z*
  - *Non-directional*
    - *a constraint between X and Y can be used to infer information on Y given information on X and viceversa.*
  - *Rarely independent*
    - *shared variables as communication mechanism*
  - *Incremental operational behaviour*
    - *each time new information available, the computation does not start from scratch*

CONSTRAINT SOLVING

- Enumeration: backtracking algorithms

– Assign a tentative value and test the constraints
– Inefficiency due to the dimensions of the search space
– Trashing

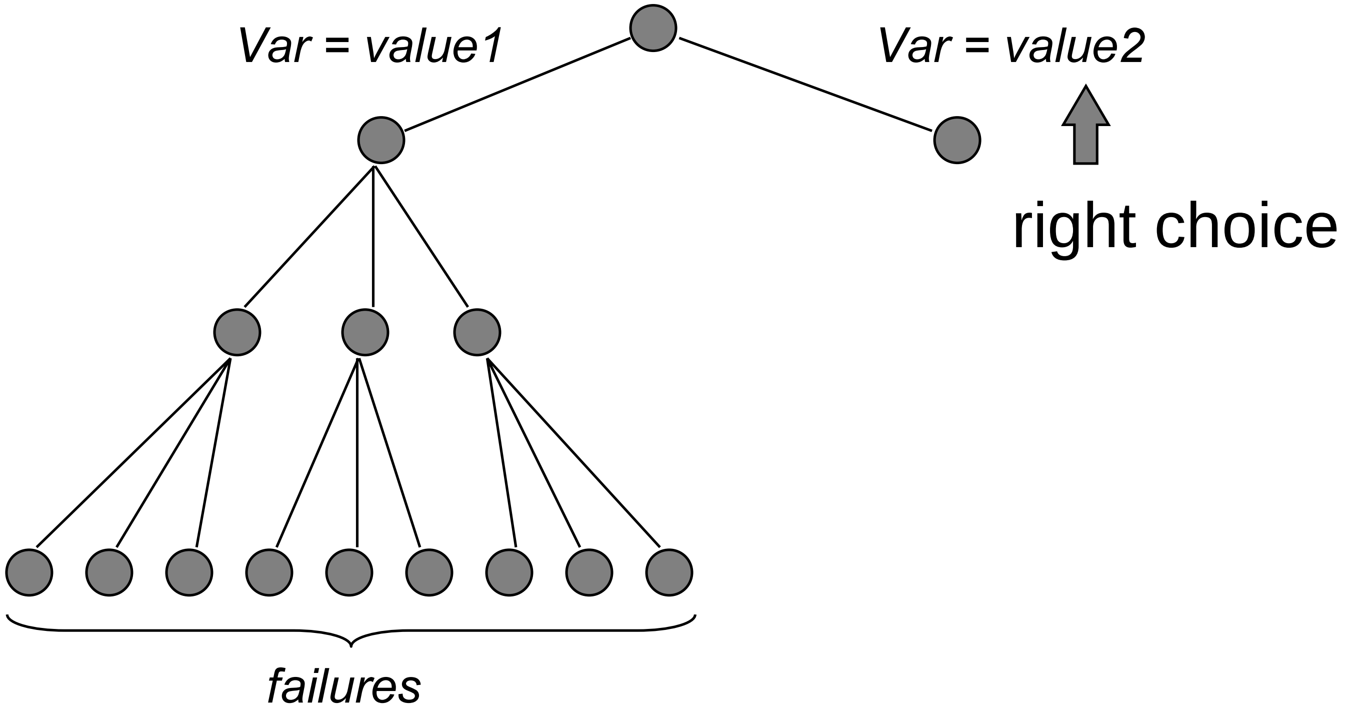
- Constraint Propagation algorithms

– aimed at reducing the search space
– constraint propagation algorithms A PRIORI remove combinations of assignments which cannot lead to a consistent solution

BACKTRACKING ALGORITHMS

Intuitive way of solving CSP: backtracking algorithms

Trashing



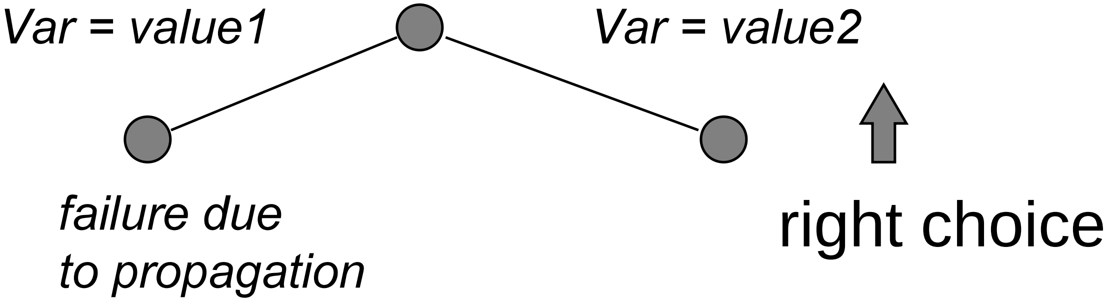
Var = value1

Var = value2

right choice

failures

PROPAGATION ALGORITHMS



Propagation algorithms: avoid failures instead of recovering from them

Based on the concept of consistency properties

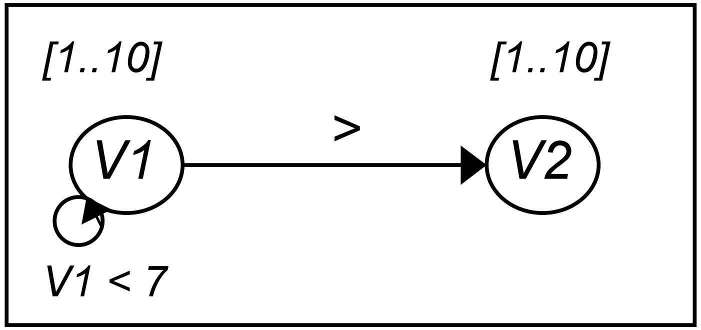
CONSISTENCY PROPERTIES

- NODE CONSISTENCY

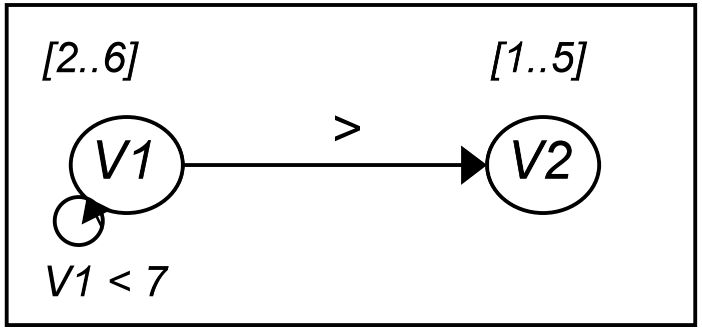
– a network is node consistent if in each node domain each value is consistent with unary constraints

- ARC CONSISTENCY

– a network is arc consistent if for each arc connecting variables Vi and Vj for each value in the domain of Vi there exists a value in the domain of Vj consistent with binary constraints



Not Node consistent
Not Arc consistent



Node consistent
Arc consistent

ENFORCING CONSISTENCY PROPERTIES

- NODE CONSISTENCY: trivial

- ARC CONSISTENCY

– *Many algorithms proposed*

– *AC1 - AC2 - AC3* ***[Mackworth AIJ (8), 77] [Montanari Inf.Sci (7), 74],*** *AC4* ***[Mohr, Henderson AIJ(28), 86],*** *AC5* ***[Van Hentenryck, Deville and Teng AIJ(58), 92],*** AC6 ***[Bessiere AIJ(65), 94],*** AC7 ***[Bessiere, Freuder, Regin AIJ(107), 99]***

– *Many variants: DAC* ***[Detcher,Pearl IJCAI85]***

MAC ***[Bessiere, Freuder, Regin, IJCAI95]***

– Bound Consistency *[****Van Hentenryck, Saraswat, Deville TR Brown, CS-93-02, 93]***

– Complexity: ***[Machworth, Freuder AIJ(25), 85], [Mohr, Henderson AIJ(28), 86], [Detcher,Pearl AIJ (34), 88] [Han, Lee AIJ(36), 88], [Cooper AIJ (41), 89]***

- PATH CONSISTENCY

– *PC1 - PC2 [****Mackworth AIJ (8), 77]***

– *PC3 [****Mohr, Henderson AIJ(28), 86]***

– *PC4* ***[Han, Lee AIJ(36), 88]***

INCOMPLETENESS of CONSISTENCY ALGORITHMS

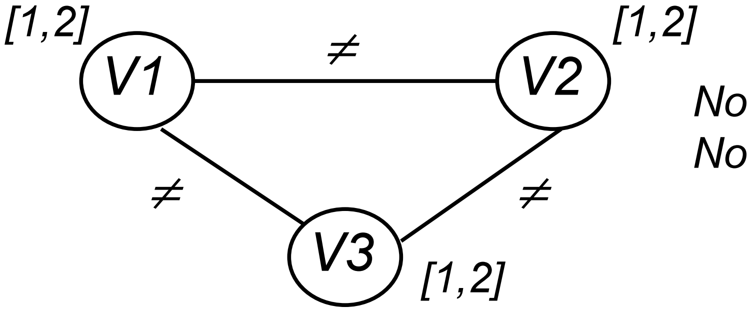
- NODE, ARC and PATH CONSISTENCY are in general not COMPLETE
  - complete for some problems with particular structures

    [Freuder JACM (29), 82], [Freuder JACM (32), 85]

- Complete algorithm: N-CONSISTENCY for N variable problems.
  Exponential complexity

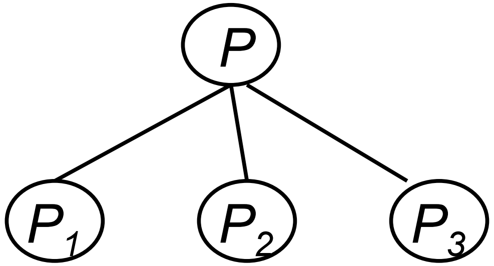
    [Freuder CACM (21), 78], [Cooper AIJ (41), 89]

- Example:



Node + Arc consistent network
No feasible solutions exist

SEARCH NEEDED



- After propagation:
  - *Solution Found*
  - *Failure*
  - *Search space to be explored: divide Problem P into easier sub-problems*

- Exploring the search space
  - *Variable selection: which is the next variable to assign ?*
  - *Value selection: which value to assign next ?*

- Search Strategies
  - *Static heuristics*
  - *Dynamic heuristics*

SEARCH STRATEGIES: general criteria

- Variable Selection
  - *First Fail - Most Constraining Principle*
  - *Select first more “difficult” variables*

- Value Selection
  - *Least Constraining Principle*
  - *Select first more “promising” values*

- Problem dependent search strategies
  - *Branching rules ensuring that the set of resulting subproblems partitions the original problem P*

OVERVIEW

- Constraint Satisfaction (Optimization) Problems
- Constraint (Logic) Programming
  - *language and tools*
- AI Applications: modelling and solving
  - *Scheduling - Timetabling - Resource Allocation*
  - *Routing*
  - *Packing - Cutting*
  - *Graphics - Vision*
  - *Planning*
- Advantages and Limitations of CP: extensions

CONSTRAINT (LOGIC) PROGRAMMING

- Aim: language for modelling and solving CSPs and COPs
  - *Constraint: basic language structure*
    - *declarative semantics*
    - *notion of consistency-entailment-optimization*
    - *operational semantics: propagation algorithms*
    - *incrementality*
    - *each constraint considered as a sub-problem*
  - *Search strategies easily implementable*

CONSTRAINT PROGRAMMING ORIGINS AND HISTORY

- Earliest ideas: field of AI Constraint Satisfaction (‘60, ‘70)
  - ***[Montanari Inf.Sci (7), 74]**, **[Waltz,75]**: constraints as matrices*
  - *REF-ARF **[Fikes PhD 68]**, ALICE **[Lauriere AIJ (10) 78]:** simple but powerful constraint languages. Customized constraint solvers*
  - *Early applications: graphics (SKETCHPAD **[Sutherland, Spring Joint Computer Conf. 63]**, ThingLab **[Borning ACM Trans. Progr. Lang and Sys. 81]**), circuit analysis (EL **[Stallman,Sussman AIJ(9),77]**)*
  - *CONSTRAINTS **[Steels PhD 80], [Sussman, Steels AIJ(14), 80]** first explicit effort of developing a constraint language*
  - CLP ***[Jaffar, Lassez POPL87], [Jaffar, Maher JLP(19-20) 94]:***

    Logic Programming as Constraint Programming
    - *unification as constraint solving*
    - *general framework CLP(X)*

CONSTRAINT PROGRAMMING ORIGINS AND HISTORY

- Constraint languages
  - *CLP(R)* ***[Jaffar et al. Trans. Progr. Lang and Sys. 92]***, *Prolog III* ***[Colmerauer CACM(33) 90]***, *CHIP* ***[Dincbas et al., JICSLP88]***, *CLP(PB)* ***[Bockmayer ICLP95]***, *Ecl^{i}ps^e* ***[Wallace et al. 97]***
- Concurrent constraint languages (‘90)
  - ***[Maher ICLP87]*** *entailment as the heart of synchronisation mechanisms*
  - *Concurrent Constraint Programming* ***[Saraswat MIT Press 93]***
  - *Oz* ***[Smolka LNCS1000, 95]***, *AKL* ***[Carlson, Haridi, Janson ICLP 94]***, *CIAO* ***[Hermenegildo et al. PPCP94]***
- Other programming languages constraint-based extensions
  - *ILOG* ***[Puget SPICIS94], [Puget, Leconte ILPS95]****: based on C**++***
  - *CHARME* ***[Bull Corporation 90****]*

FINITE DOMAIN CP

- One to one mapping with CSP concepts
- Problem modelling
  - *Variables range on a finite domain of objects of arbitrary type*
  - *Constraints among variables*
    - *mathematical constraints*
    - *symbolic constraints*
- Problem solving
  - *Propagation algorithms embedded in constraints*
  - *Arc consistency as standard propagation*
  - *More sophisticated propagation for global constraints*
  - *Search strategies*

CP: PROBLEM MODELLING

- ONE to ONE mapping between CSP (COP) concepts and CP syntactic structures

- A problem should be modelled in terms of

– Variables	→	Problem entities
– Domains	→	Possible Values
– Constraints	→	Relations among variables
– Objective function (if any)	→	Optimization Criteria

CP: PROBLEM CONSTRAINTS

- Mathematical constraints: =, >, <, ≠, ≥, ≤
  - Propagation: arc-consistency

- Symbolic Constraints ***[Beldiceanu, Contejean, Math.Comp.Mod. 94]***
  - Embed more powerful propagation algorithms
  - More concise formulation
    - `alldifferent([X`$_1$`,...X`$_m$`])`
      all variables have different values
    - `element(N,[X`$_1$`,...X`$_m$`],Value)`
      the n-th element of the list should be equal to Value
    - `cumulative([S`$_1$`,...S`$_m$`], [D`$_1$`,...D`$_n$`], [R`$_1$`,...R`$_n$`], L)`
      used for capacity constraints
    - disjunctive constraints

CP: PROBLEM MODELLING (Example)

- Map Colouring

-
  ```
  map_colouring([V1,V2,V3,V4,V5]):-
       V1::[red,green,yellow,blue,pink],
       V2::[red,green,yellow,blue,pink],
       V3::[red,green,yellow,blue,pink],
       V4::[red,green,yellow,blue,pink],
       V5::[red,green,yellow,blue,pink],
       V1≠V2, V1≠V3, V1≠V4, V1≠V5, V2≠V3,
       V2≠V4, V2≠V5, V3≠V4, V4≠V5,
       .....
  ```

variables & domains

constraints

Or alternatively

```
     alldifferent([V1,V2,V3,V4]),
     alldifferent([V1,V2,V4,V5]).
```

CP: PROBLEM SOLVING

- Notion of Consistency:
  - Is the set of constraint consistent ?
  - Does it exist a solution ?

- Constraint Propagation: inference mechanism
  - Remove from domain inconsistent values
  - Infer new constraints

- Search: branching strategies
  - Variable selection
  - Value selection

CP: CONSISTENCY - ENTAILMENT

- A set of constraint is CONSISTENT if it admits at least one solution

- COMPLETE solvers are able to decide if the set of constraint is satisfiable (real numbers)

- INCOMPLETE solvers detect some form of inconsistency, but they are not able to decide satisfiability (finite domains)
  - the inconsistency is detected when one variable domain becomes empty

- A set of constraint C ENTAILS a constraint c if: C $\models$ c. Some solvers are based on entailment

CONSTRAINT PROPAGATION

- Mathematical Constraints:
  - Example 1
    - `X::[1..10], Y::[5..15], X>Y`

    Arc-consistency: for each value v of variable X, if a value for Y compatible with constraints does not exists, v is deleted from the domain of X and vice versa.
    - `X::[6..10], Y::[5..9]` after propagation
  - Example 2
    - `X::[1..10], Y::[5..15], X=Y`
    - `X::[5..10], Y::[5..10]` after propagation
  - Example 3
    - `X::[1..10], Y::[5..15], X≠Y`
    - No propagation

CONSTRAINT PROPAGATION

- Symbolic Constraints:
  - Each Constraint has an associated propagation algorithm called *FILTERING ALGORITHM*
  - Complex propagation algorithms using also Operations Research techniques
  - Propagation ends when a state of quiescence is reached: the constraint graph is stable
  - Incremental propagation

CONSTRAINT PROPAGATION

- Symbolic Constraint: example 1
  - **alldifferent([X_1,...X_n])**

    all variables have different values

  Declaratively equivalent to a set of binary constraints

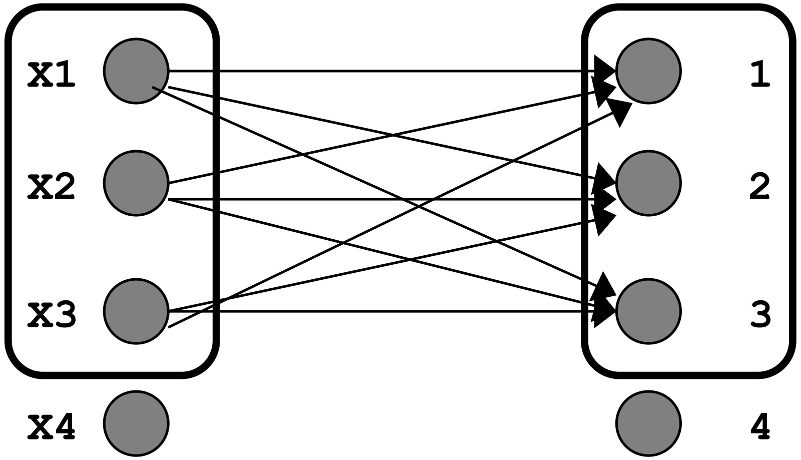
  alldifferent([X_1,...X_n]) $\leftrightarrow X_1 \neq X_2, X_1 \neq X_3, \ldots, X_{n-1} \neq X_n$

  Operationally more powerful constraints

- **X1::[1,2,3],X2::[1,2,3],X3::[1,2,3],X4::[1,2,3,4],**
- *Arc consistency: NO PROPAGATION*
- *Filtering algorithm **[Regin AAAI94]**: values 1 2 3 removed from* **X4**

CONSTRAINT PROPAGATION

- Symbolic Constraint: example 1 (continues)
- **`X1::[1,2,3],X2::[1,2,3],X3::[1,2,3],X4::[1,2,3,4],`**
- *Filtering algorithm: values 1 2 3 removed from* **`X4`**



Set of variables whose cardinality is 3 ranging on the same set of values whose cardinality is 3

X4::[~~1~~,~~2~~,~~3~~,4]

CONSTRAINT PROPAGATION

- Symbolic Constraint: example 2
  - **element(N,[X_1,...X_m],Value)**

    the n-th element of the list should be equal to Value

- propagation from **N** to **Value** :
  - **N=i** → **X_i = Value**

- propagation from **Value** to **N** and **X_i** :
  - **Value= x** → **N=1 and X_1=x or**

    N=2 and X_2=x or....

    N=m and X_m=x

CONSTRAINT PROPAGATION

- Symbolic Constraint: disjunctive constraints
  - suppose we have two different lessons to be given by the same teacher. We have the starting times of the lessons: **`L1Start`** and **`L2Start`** and their duration **`Duration1`** and **`Duration2`**.
  - Clearly, the two lessons cannot be scheduled at the same time:

```
L1Start + Duration1 ≤ L2Start
            OR
L2Start + Duration2 ≤ L1Start
```

  - Two INDEPENDENT CSPs one for each size of the disjunction.

CONSTRAINT PROPAGATION

- Symbolic Constraint: disjunctive constraints (2)

  – Two INDEPENDENT CSPs one for each size of the disjunction: one choice does not affect the other → Trashing

  – Exponential number of problems:
    - N disjunction → 2^N Problems
    - Main source of complexity in real world problems

  – Solutions:
    - constructive disjunction ***[Van Hentenryck, Saraswat, Deville, TR Brown 93]***
    - cardinality operator ***[P. Van Hentenryck, Y. Deville ICLP91]***
    - meta-constraints ***[Carlson, Haridi, Janson ILPS94], [Smolka LNCS1000, 95]***, ***[Lamma, Mello, Milano JLP99]***

CONSTRUCTIVE DISJUNCTION

- ***P. Van Hentenryck, V. Saraswat, Y. Deville,***
  Design, Implementation and Evaluation of the Constraint Language cc(FD),
  Tech. Rep. Brown University, CS-93-02, 1993.

- Exploits the disjunction to prune the search space

- Idea: add to the store constraints entailed by all possible alternatives

- Example: `X::[5..10], Y::[7..11], Z::[1..20], (Z=X OR Z=Y)`
  - `Z=X` would reduce the domain of `Z` to `[5..10]`
  - `Z=X` would reduce the domain of `Z` to `[7..11]`
  - result of the constructive disjunction: `Z::[5..11]`

CARDINALITY OPERATOR

- Symbolic Constraint: cardinality operator
  - **#(l,u,[c_1,…,c_n])** holds ⇔ the number **k** of constraints c_i (**1≤i≤n**) satisfiable is not less than **l** and no more than **u**
  - How to model disjunctive constraints with the cardinality operator

```
#(1,1,[L1Start+Duration1 ≤ L2Start, L2Start+Duration2 ≤ L1Start])
```

META-CONSTRAINTS

- Symbolic Constraint: meta-constraints
  - Reified Constraints: each constraint is associated to a boolean variable **`B`**. If **`B=1`** the constraint holds, if **`B=0`** the constraint does not.
  - **`c`** $\Leftrightarrow$ **`B`**
  - How to model disjunctive constraints with reified constraint

```
B1 ::[0,1], B2::[0,1],
L1Start+Duration1 ≤ L2Start ⇔ B1,
L2Start+Duration2 ≤ L1Start ⇔ B2,
B1 + B2 = 1
```

CONSTRAINT PROPAGATION

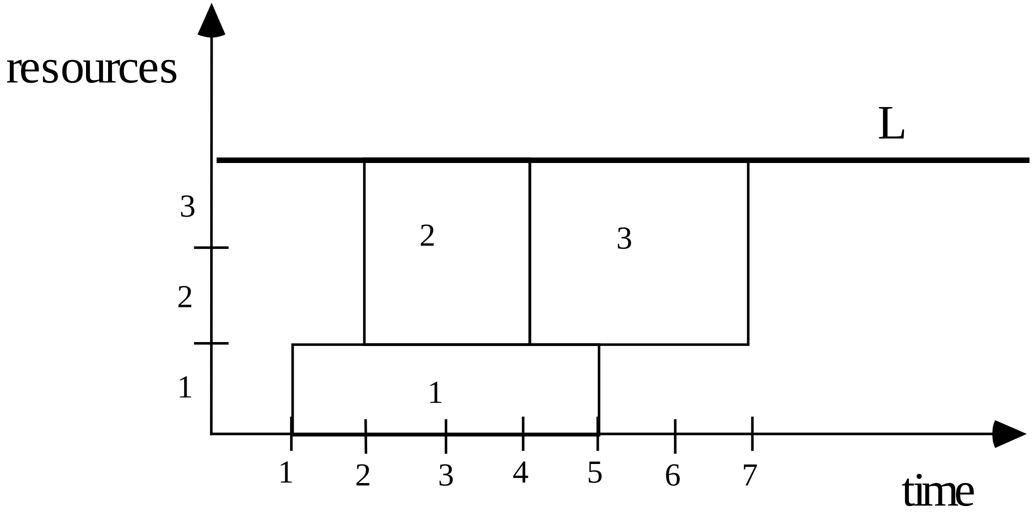
- Symbolic Constraint: example 3
  - **`cumulative([`S_1`,...`S_m`], [`D_1`,...`D_n`], [`R_1`,...`R_n`], L)`**
    - **S_1,...S_m** are starting times (domain variables)
    - **D_1,...D_n** are durations (domain variables)
    - **R_1,...R_n** are required resources (domain variables)
    - **L** resource capacity limit (also time variant)

- Given the interval [min,max] where min = *min*$_i$ {**S_i**}, max = max{**S_i+D_i**} - 1, the cumulative constraint ensures that

$$\max\{\sum_{j|S_j \leq i \leq S_j+D_j} R_i\} \leq L$$

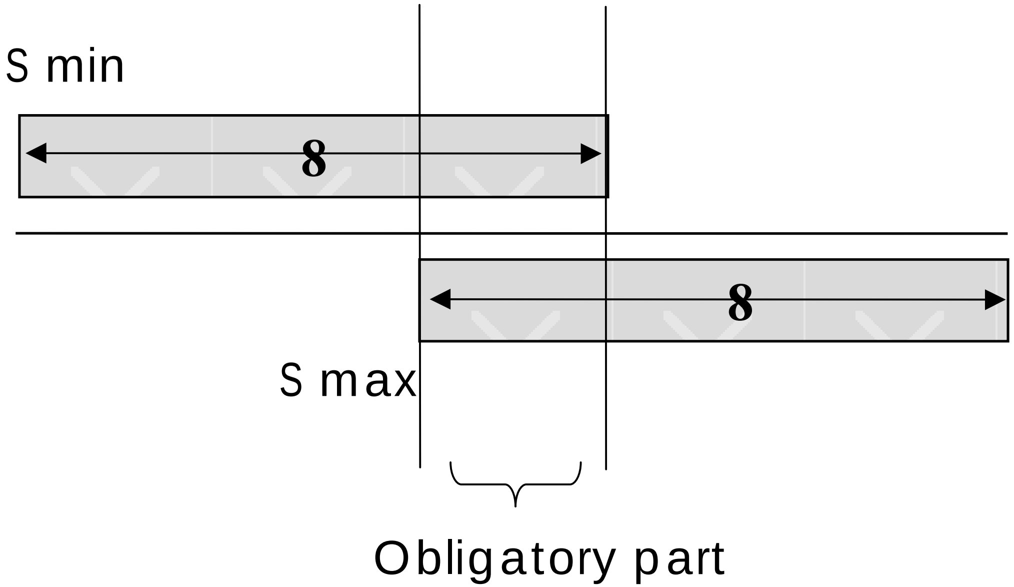
CONSTRAINT PROPAGATION

- Symbolic Constraint: example 3 (continues)
  - **`cumulative([1,2,4],[4,2,3],[1,2,2],3)`**



CONSTRAINT PROPAGATION

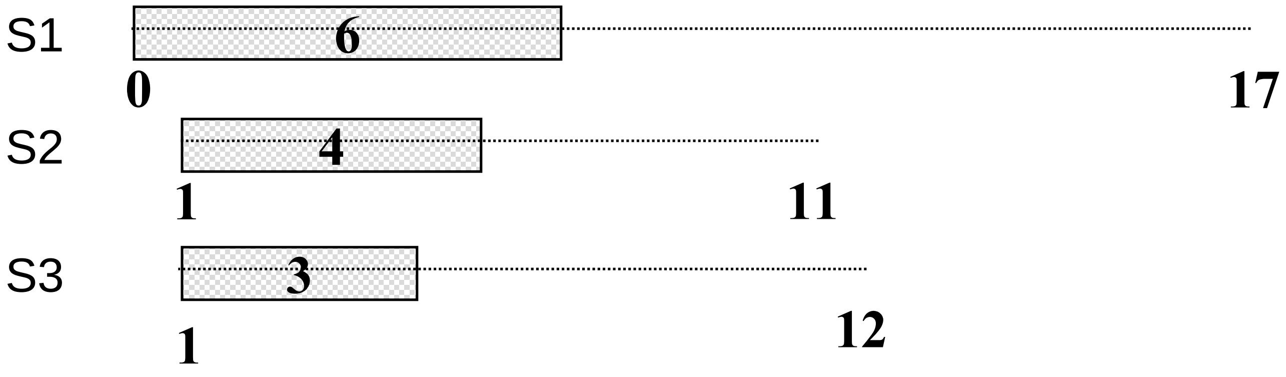
- Symbolic Constraint: example 3 (continues)
  - a propagation example used in the resource constraint is that based on the *obligatory parts*



CONSTRAINT PROPAGATION

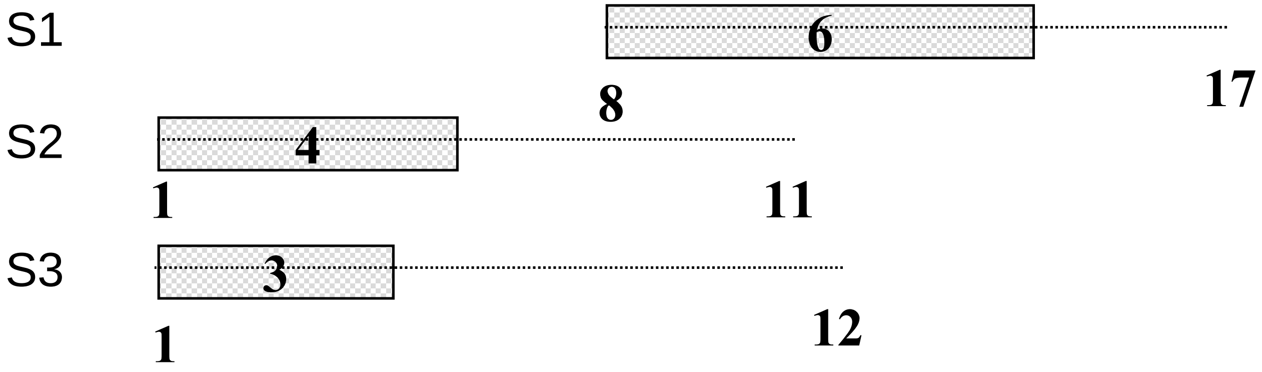
- Symbolic Constraint: example 3 (continues)
  - another propagation example used in the resource constraint is that based on the *edge finding* ***[Baptiste, Le Pape, Nuijten, IJCAI95]***

Consider a unary resource and three activities.



CONSTRAINT PROPAGATION

- Symbolic Constraint: example 3 (continues)



We can deduce that earliest start time of S1 is 8.

This is based on the fact that S1 must be scheduled after S2 and S3.

Global reasoning: suppose either S2 or S3 is scheduled after S1. Then the maximum of the completion times of S2 and S3 is at least 13 (out of the domain of S2 and S3).

CONSTRAINT PROPAGATION

- Global reasoning: example 3 (continues)

Basic Theorem: [*Carlier, Pinson, Man.Sci.95]*

Let o be an activity and S a set of activities all to be scheduled on the same unary resource (o not in S). The earliest start time is e, the sum of durations is D and the latest completion time C. If

e(S+{o}) + D(S+{o}) > C(S)

then no schedule exists in which o precedes any of the operations in S. This implies that the earliest start time of o can be set to

$$\max_{(S' \subseteq S)} \{e(S') + D(S')\}.$$

CONSTRAINTS: general remarks

- Symbolic Constraints are available in most CP tools

- Local vs. Global reasoning ⇒ powerful propagation
- Local vs. Global reasoning ⇒ computational effort

} Tradeoff

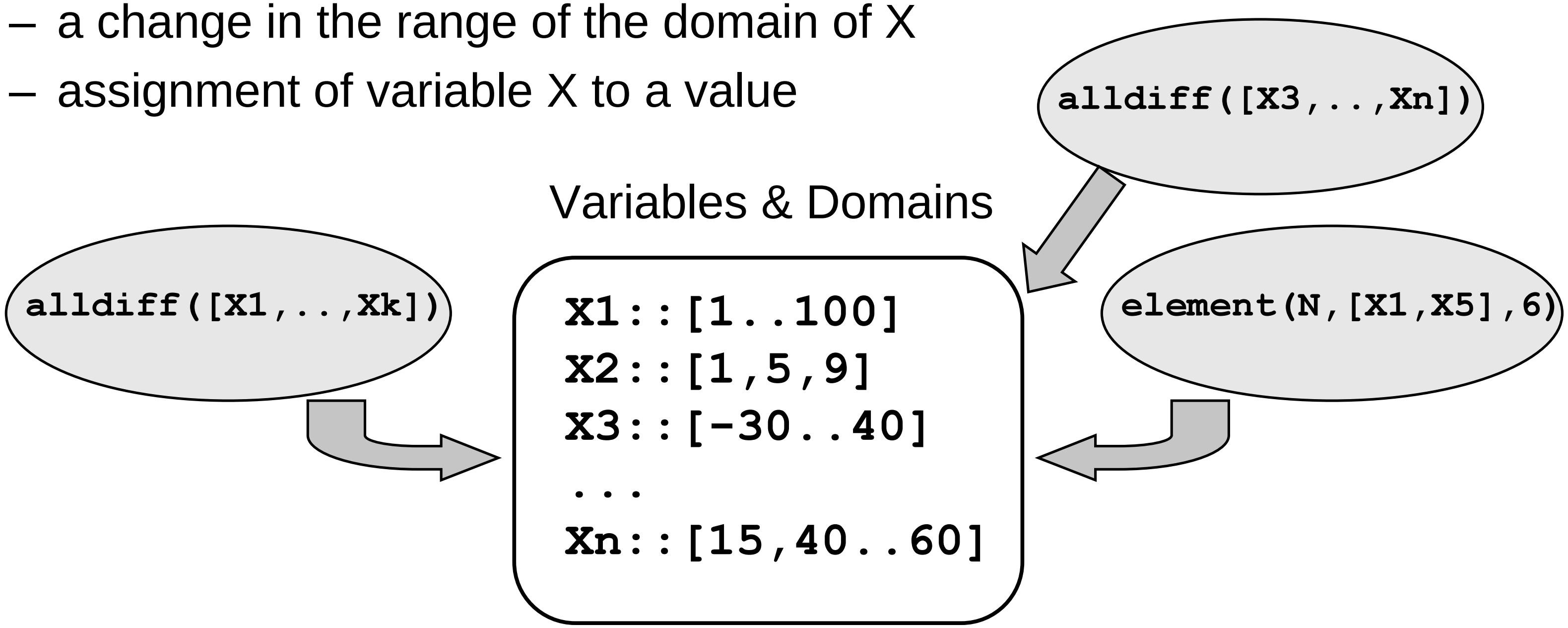
- Generalization of frequently found constraints

- Concise and easily understandable code

- Symbolic constraints represent independent subproblems (relaxations of the original problem)

INTERACTIONS AMONG CONSTRAINTS

- Interact each other through shared variables in the constraint store
- Trigger propagation each time an event is raised on one variable X
  - a change in the domain of X
  - a change in the range of the domain of X
  - assignment of variable X to a value

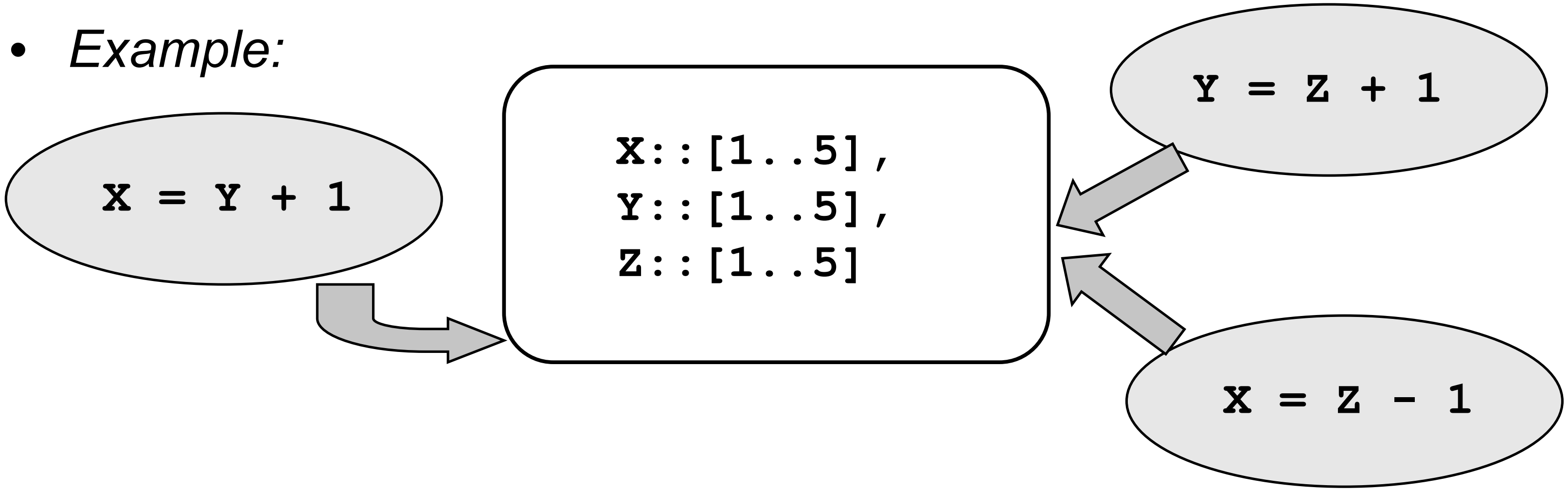


INTERACTIONS AMONG CONSTRAINTS

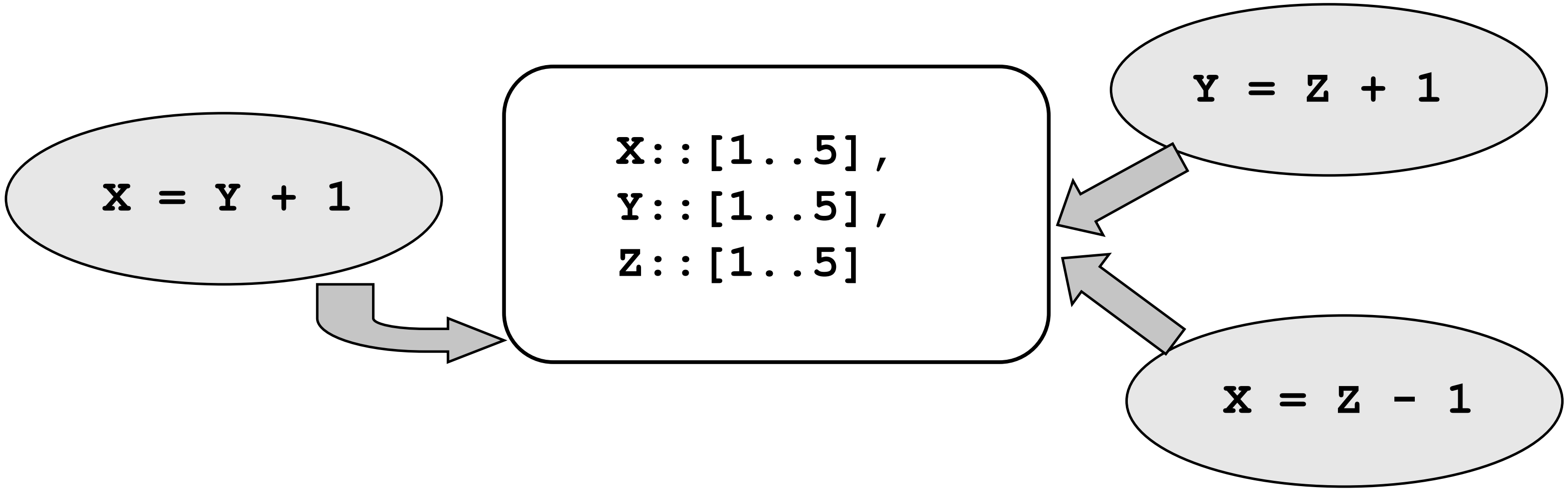
- In general each variable is involved in many constraints. Consequently, each change in variable domains as a result of propagation may result in further propagations to other variables.

- Constraints agents view: during their lifetime they alternate between suspended and waking states (triggered by events).

- *Example:*



INTERACTIONS AMONG CONSTRAINTS



```
X = Y + 1
```

```
X::[1..5],
Y::[1..5],
Z::[1..5]
```

```
Y = Z + 1
```

```
X = Z - 1
```

- First propagation of ***X = Y + 1*** yields to

```
X::[2..5]
Y::[1..4]
Z::[1..5]
```

X = Y + 1 is suspended

INTERACTIONS AMONG CONSTRAINTS

- Second propagation of ***`Y = Z + 1`*** yields to

```
X::[2..5]
Y::[2..4]
Z::[1..3]
```

`Y = Z + 1` is suspended

- The domain of ***`Y`*** has changed ***`X = Y + 1`*** *is awakened*

```
X::[3..5]
Y::[2..4]
Z::[1..3]
```

`X = Y + 1` is suspended

INTERACTIONS AMONG CONSTRAINTS

- Third propagation of ***Z = X - 1*** yields to

```
X::[]
Y::[2..4]
Z::[1..3]
```

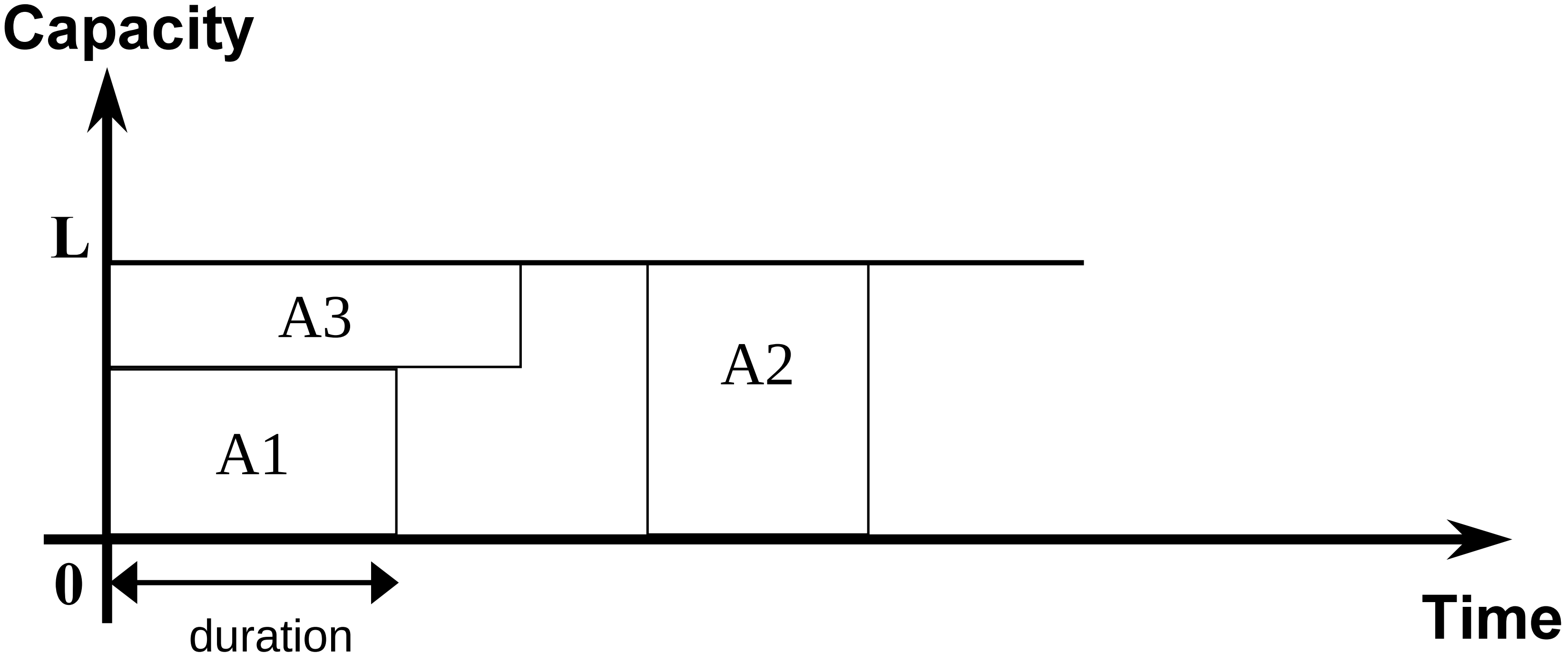
FAILURE detected

- The order in which constraints are considered (delayed and awakened) does not affect the propagation results, BUT can affect the performances of the algorithm.

CONSTRAINTS: GENERAL STRUCTURES

- Cumulative constraint on different examples

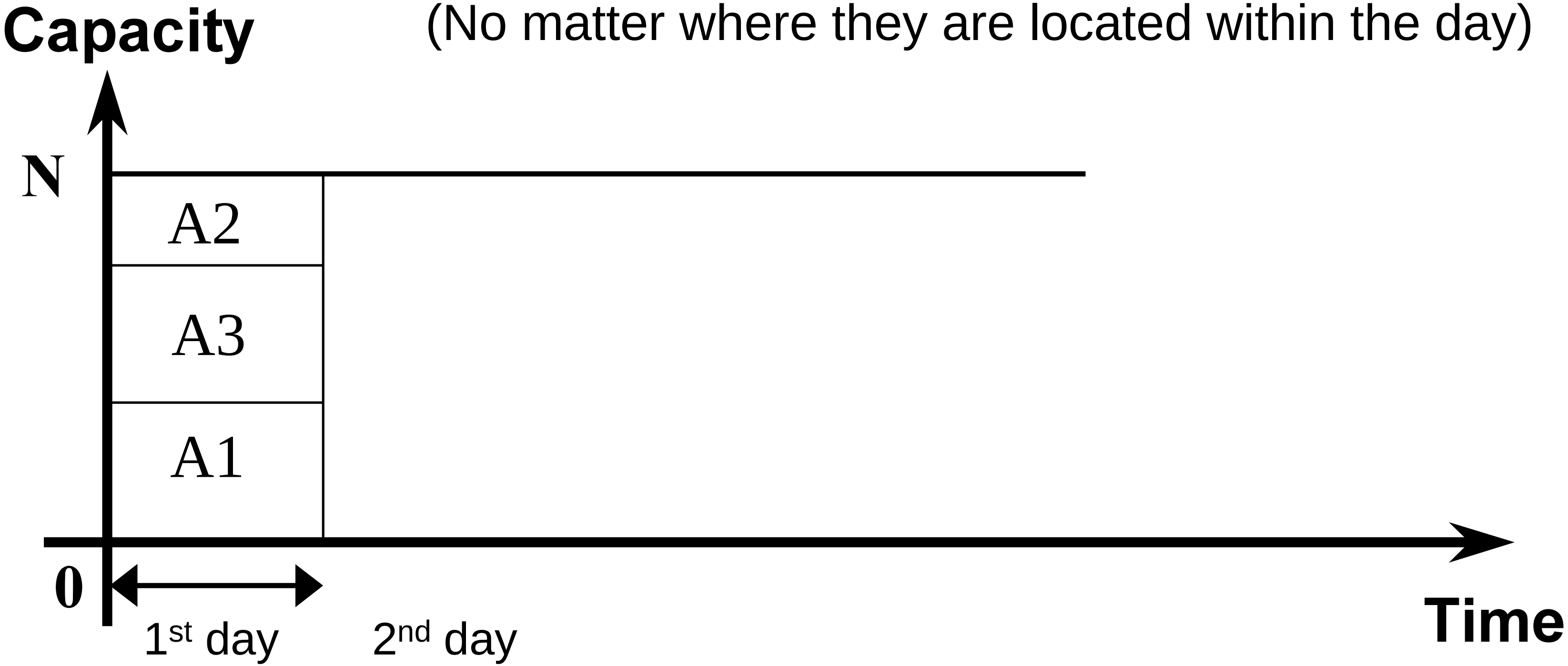
- Scheduling (1): Tasks A1, A2, A3 sharing the same resource with limited capacity. Duration on X and Resource use on Y



CONSTRAINTS: GENERAL STRUCTURES

- Cumulative constraint on different examples
- Scheduling (2): *Limited number of resources per day = N.* Day on X and Resource number used on Y

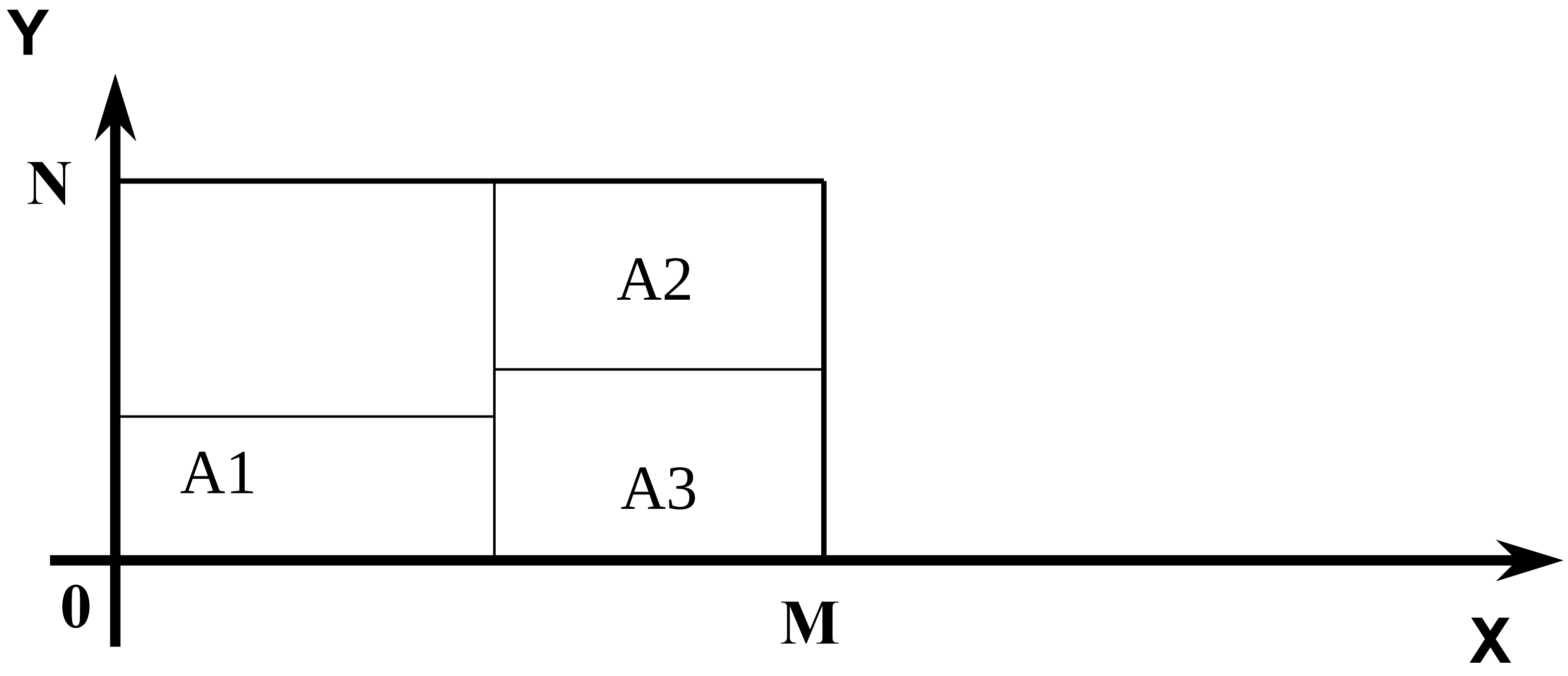
(No matter where they are located within the day)



CONSTRAINTS: GENERAL STRUCTURES

- Cumulative constraint on different examples

- Packing: *Box whose dimensions are M x N. Pieces to be packed.*



REDUNDANT CONSTRAINTS

- Propagation is in general not complete: inconsistent values are left in domains

- Redundant constraints can be useful for reducing the search space even if they introduce some overhead (treadeoff).

- A redundant constraint C is one which is already entailed by other constraints {C1…Ck}, but this entailment is not found by the constraint solver due to its incomplete nature.

REDUNDANT CONSTRAINTS (2)

- Example: Magic sequence
- Given a set of n+1 variables $X_0,\dots,X_n$. Each X_i should respect the following constraints:
  - 0 appears X_0 times in the solution
  - 1 appears X_1 times
  - …
  - n appears X_n times.

- **magic_sequence([X_0,…,X_n]):-**

```
    X0,…,Xn ::[0..n],
    exactly(X0,[X0,…,Xn], 0),
    exactly(X1,[X0,…,Xn], 1),
    ...,
    exactly(Xn,[X0,…,Xn], n),
    ...
```

REDUNDANT CONSTRAINTS (3)

- Redundant constraint: note that the sum of all variables multiplied by their value (index) is equal to the number of cells in the sequence. Thus, variables satisfy the constraint:

  - $X_1 + 2*X_2 + \ldots + N*X_n = N + 1$

-
  ```
  magic_sequence([X_0,…,X_n]):-
      X_0,…,X_n ::[0..n],
      exactly(X_0,[X_0,…,X_n], 0),
      exactly(X_1,[X_0,…,X_n], 1),
      ...,
      exactly(X_n,[X_0,…,X_n], n),
      X_1+ 2*X_2 +…+ N*X_n = N + 1,
      ...
  ```

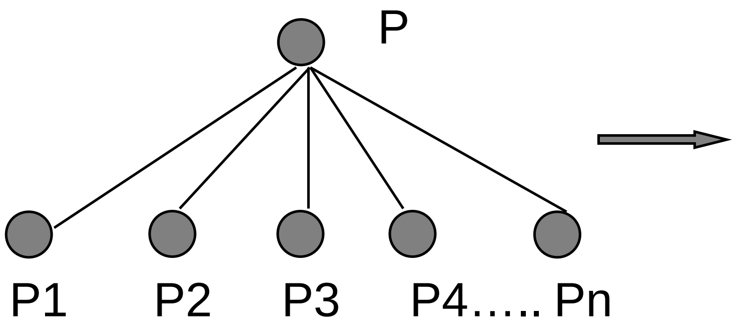
SEARCH

- Propagation is, in general, not complete. After propagation:
  - Solution found → stop
  - Failure detected → backtracking
  - Domains contain some values → SEARCH

- SEARCH: Basic idea
  - Divide the problem into subproblems and solve them independently
  - Subproblems must partition the original problem

- AIM: maintain the search space as small as possible
  - conventionally, left branches are explored first.

SEARCH



Branching Strategy
GUESS

- Branching strategies define the way of partitioning the problem P into easier subproblems P1, P2, …, Pn.

- To each subproblem: apply again propagation. New branches can be pruned thanks to the new information derived from the branching.

SEARCH

- Most popular branching in CP: *labelling*

- LABELLING:
  - Select one VARIABLE
  - Select one VALUE in its domain
  - Assign the VALUE to the VARIABLE

- The order in which variables and values are chosen (i.e., the search strategy) greatly affects the performances of the search algorithm.

- Find good strategies

SEARCH STRATEGIES: GENERAL CRITERIA

- VARIABLE CHOICE: more difficult variables first
  - FIRST FAIL: select first the variable with the smallest domain
  - MOST CONSTRAINING PRINCIPLE: select first the variable involved in the greatest number of constraints
  - HYBRID APPROACH: combination of the two

- VALUE CHOICE: more promising values first
  - LEAST CONSTRAINING PRINCIPLE.

- PROBLEM DEPENDENT STRATEGIES

SEARCH STRATEGIES: HOW TO CHOOSE

- Given the variety of problems, there do not exist definitive rules for choosing the best strategy
- CRITERIA: the sooner the search strategy prunes branches of the search space, the more efficiently it works
- PARAMETERS to be taken into account:
  - computational time
  - number of failures

CP: PROBLEM MODELLING (Example)

- Map Colouring: complete code

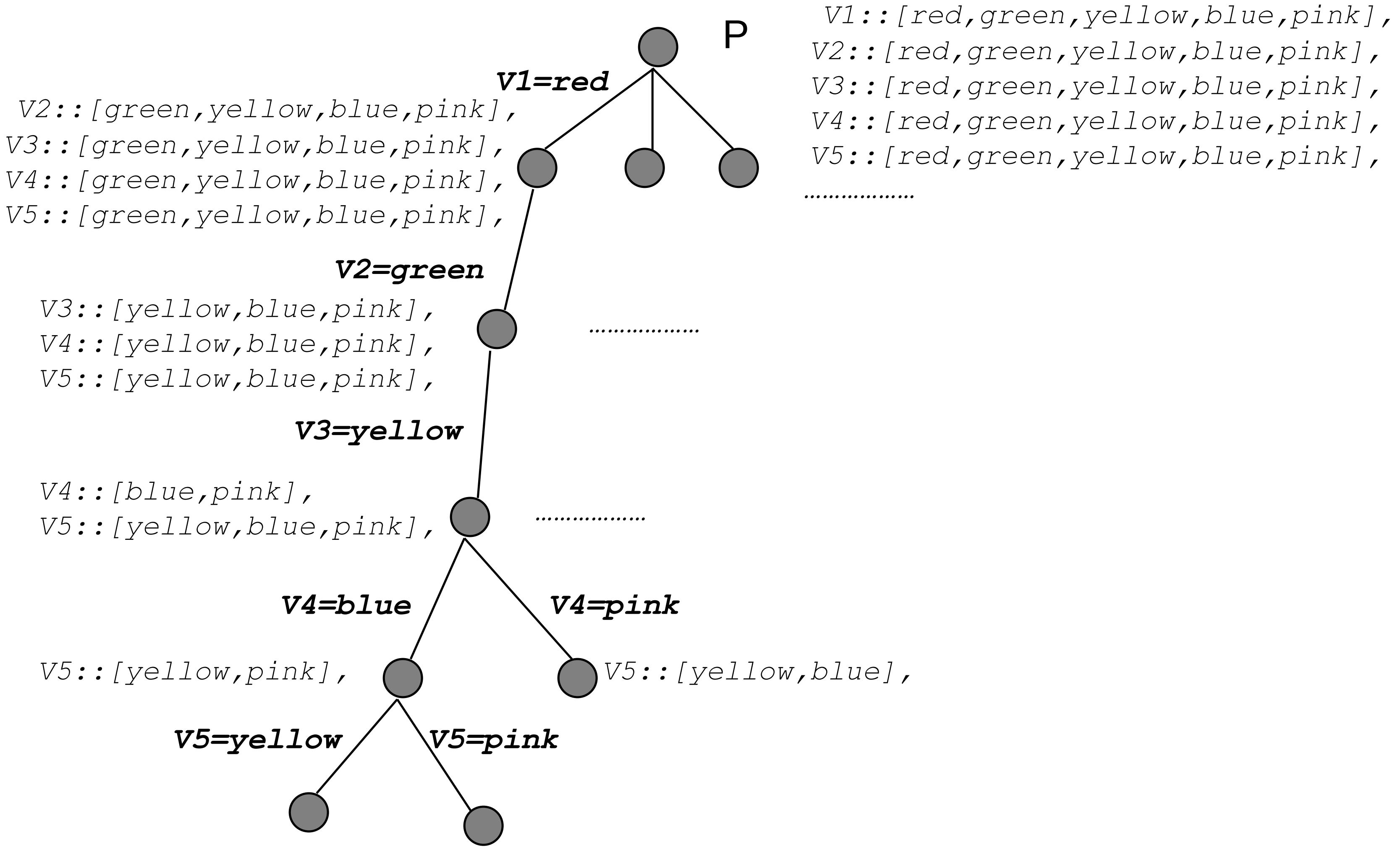
-
  ```
  map_colouring([V1,V2,V3,V4,V5]):-
        V1::[red,green,yellow,blue,pink],
        V2::[red,green,yellow,blue,pink],
        V3::[red,green,yellow,blue,pink],
        V4::[red,green,yellow,blue,pink],
        V5::[red,green,yellow,blue,pink],
        V1≠V2, V1≠V3, V1≠V4, V1≠V5, V2≠V3,
        V2≠V4, V2≠V5, V3≠V4, V4≠V5,
        labelling([V1,V2,V3,V4,V5]).
  ```

variables & domains

constraints

search

SEARCH SPACE



P

V1::[red,green,yellow,blue,pink],
V2::[red,green,yellow,blue,pink],
V3::[red,green,yellow,blue,pink],
V4::[red,green,yellow,blue,pink],
V5::[red,green,yellow,blue,pink],
………………

V1=red

V2::[green,yellow,blue,pink],
V3::[green,yellow,blue,pink],
V4::[green,yellow,blue,pink],
V5::[green,yellow,blue,pink],

V2=green

V3::[yellow,blue,pink],
V4::[yellow,blue,pink],
V5::[yellow,blue,pink],

………………

V3=yellow

V4::[blue,pink],
V5::[yellow,blue,pink],

………………

V4=blue

V4=pink

V5::[yellow,pink],

V5::[yellow,blue],

V5=yellow

V5=pink

OPTIMIZATION

- In some applications, we are not interested in a feasible solution but in the OPTIMAL solution according to a given criterion
- ENUMERATION ⇨ inefficient
  - find all feasible solutions
  - chose the best one
- Constraint Programming tools in general embed a simple form of Branch and Bound
  - each time a solution is found whose cost is C*, impose a constraint on the remaining search tree, stating that further solutions (whose cost is C) should be better than the best one found so far

C < C*

OPTIMIZATION

- Operations Research (OR) has a long tradition in optimization problems.

- OR Branch & Bound methods are based on the optimal solution of a relaxation of the original problem → BOUND
  - relaxation: same problem with some constraints relaxed

- Trend: embed OR techniques in CP for improving the performances of the search algorithm

CONSTRAINT PROGRAMMING TOOLS

– CLP(R) ***[Jaffar et al. Trans. Progr. Lang and Sys. 92],***

– Prolog III ***[Colmerauer CACM(33) 90]***,

– CHIP ***[Dincbas et al., JICSLP88],***

– CLP(PB) ***[Bockmayer ICLP95]***,

– Ecl^{i}ps^{e} ***[Wallace et al.97],*** Conjunto ***[Gervet, Constraints(1), 97]***

– *Oz* ***[Smolka JLP 91]****,*

– *AKL* ***[Carlson, S.Haridi, S.Janson ILPS94],***

– *CIAO* ***[Hermenegildo et al. PPCP94]***

– *ILOG* ***[Puget SPICIS94], [Puget, Leconte ILPS95]***

– *CHARME* ***[Bull Corporation 90]***

– *many others……….*

OVERVIEW

- Constraint Satisfaction (Optimization) Problems
- Constraint (Logic) Programming
  - *language and tools*
- AI Applications: modelling and solving
  - *Scheduling - Timetabling - Resource Allocation*
  - *Routing*
  - *Packing - Cutting*
  - *Graphics - Vision*
  - *Planning*
- Advantages and Limitations of CP: extensions

AI APPLICATIONS: MODELLING and SOLVING

- We will focus on Constraint Programming on finite domains

- For each application, we will present a problem description, the CP modelling and solving part.

- Applications discussed:
  - *Scheduling - Timetabling - Resource Allocation*
  - *Routing*
  - *Packing - Cutting*

  optimization

  - *Graphics - Vision*
  - *Planning*

  feasibility

SCHEDULING - TIMETABLING - RESOURCE ALLOCATION

- Three applications with analogous features/constraints:
  - we will focus on scheduling. Same considerations for timetabling and resource allocation (easier problem)

- Scheduling is probably one of the most successful applications of CP to date
  - flexibility
  - generality
  - easy code

- NP-complete problem studied by the AI community since 80s

SCHEDULING: problem definition

- Scheduling concerns the assignment of limited resources (machines, money, personnel) to activities (project phases, manufacturing, lessons) over time
- Constraints
  - temporal restrictions
    - ordering among activities
    - due dates - release dates
  - resource capacity
    - different types of resources
    - consumable/renewable resources
- Optimization Criteria
    - makespan
    - resource balance
    - lateness on due dates
    - resource assignment cost

SCHEDULING: Activities

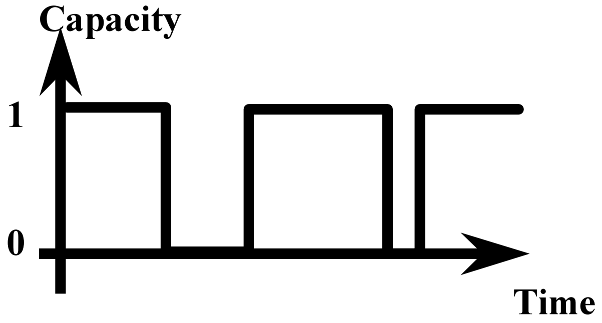
- Decision variables:
  - Activity start times
  - Activity end times
  - Activity resource assignments
  - Alternative activities (alternative routing)

- Activity types:
  - interval activity: cannot be interrupted
  - breakable activity: can be interrupted by breaks
  - preemptable activity: can be interrupted by other activities

SCHEDULING: Resources

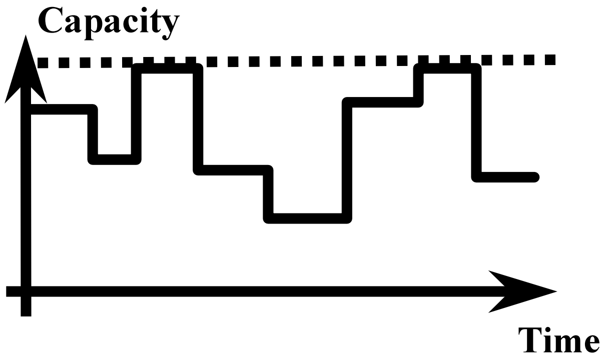
- Resource types:
  - 1. Unary resources
  - 2. Discrete resources
  - 3. State resources
  - 4. Energy resources
  - 5. Stock

1.

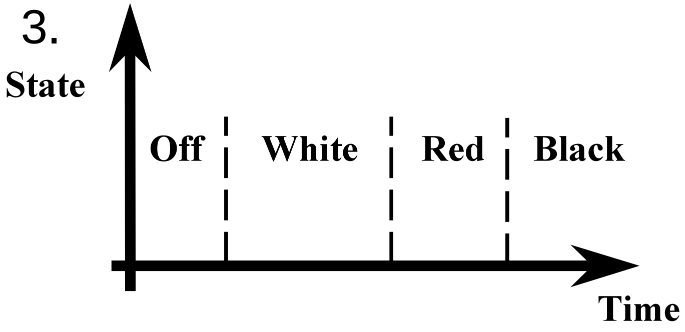


2.

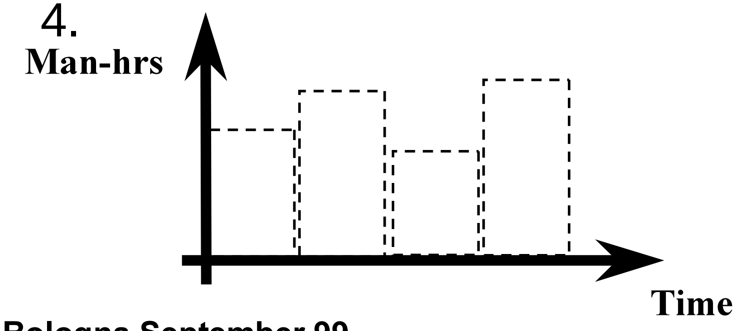
5.



3.



4.



SCHEDULING: Simple Example

- 6 activities: each activity described by a predicate

  task(NAME,DURATION,LISTofPRECEDINGTASKS,MACHINE).

```
task(j1,3,[],m1).
task(j2,8,[],m1).
task(j3,8,[j4,j5],m1).
task(j4,6,[],m2).
task(j5,3,[j1],m2).
task(j6,4,[j1],m2).
```

- Machines are unary resources.

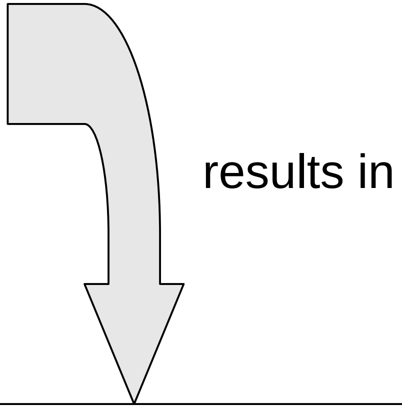
- A maximum ending time End is required.

SCHEDULING: Simple Example

```
schedule(Data, End, TaksList):-
    makeTaskVariables(Data,End,TaskList),
    precedence(Data, TaskList),
    machines(Data, TaskList),
    minimize(labelTasks(TaskList),End).

makeTaskVariables([],_,[]).
makeTaskVariables([task(N,D,_,_)|T],End,[Start|Var]):-
    Start::[0..End-D],
    makeTaskVariables(T,End,Var).

precedence([task(N,_,Prec,_)|T], [Start|Var]):-
   select_preceding_tasks(Prec,T,Var,PrecVars,PrecDurations),
   impose_constraints(Start,PrecVars,PrecDurations),
   precedence(T,Var).

impose_constraints(_,[],[]).
impose_constraints(Start,[Var|Other],[Dur|OtherDur]):-
   Var + Dur <= Start
   impose_constraints(Start,Other,OtherDur).
```

SCHEDULING: Simple Example

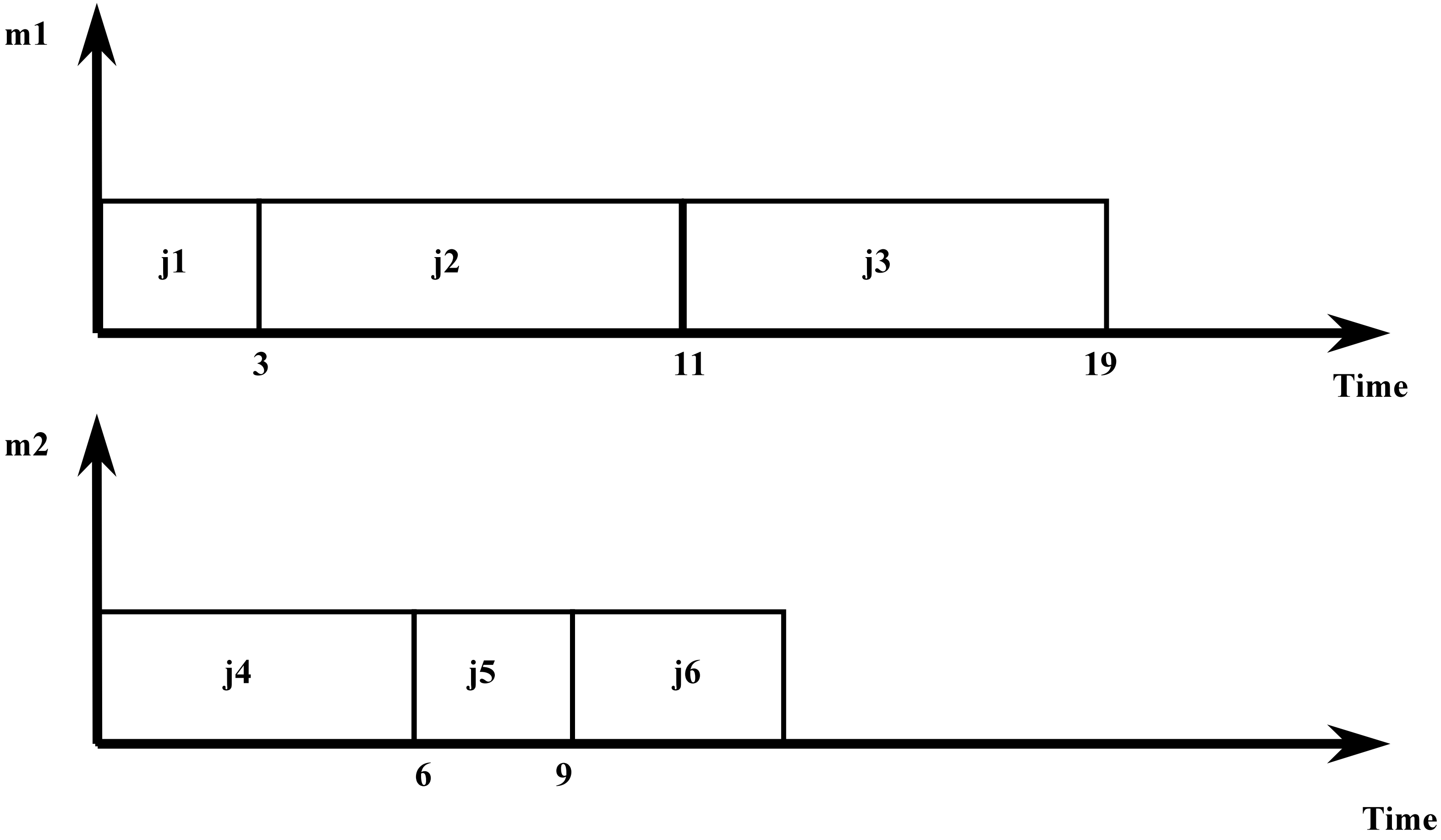
```
machines(Data, TaskList):-
   tasks_sharing_resource(Data,TaskList,SameResource,Durations),
   impose_cumulative(SameResource,Durations,Use).

impose_cumulative([],[],_).
impose_cumulative([ListSameRes|LSR],[Dur|D],[Use|U]):-
    cumulative(ListSameRes,Dur,Use,1),
    impose_cumulative(LSR,D,U).

labelTasks([]).
labelTasks([Task|Other]):-
    indomain(Task),
    labelTasks(Other).
```



results in

```
cumulative([Start1,Start2,Start3],[3,8,8],[1,1,1],1)
cumulative([Start4,Start5,Start6],[6,3,4],[1,1,1],1)
```

SCHEDULING: Optimal Solution



SCHEDULING: Optimality

- **`minimize`**: finds the optimal solution (simple Branch & Bound)

- Minimization of the makespan: an heuristic which always selects the task which can be assigned first and assigns to the task the minimal bound is in general a good heuristics. As a choice point, delay the task.

```
labelTasks([]).
labelTasks(TaskList):-
    find_min_start(TaskList,First,MinStart,Others),
    label_earliest(TaskList,First,MinStart,Others).

label_earliest(TaskList,First,Min,Others):- % schedule the task
    First = Min,
    labelTasks(Others).
label_earliest(TaskList,First,Min,Others):- % delay the task
    First ≠ Min,
    labelTasks(TaskList).
```

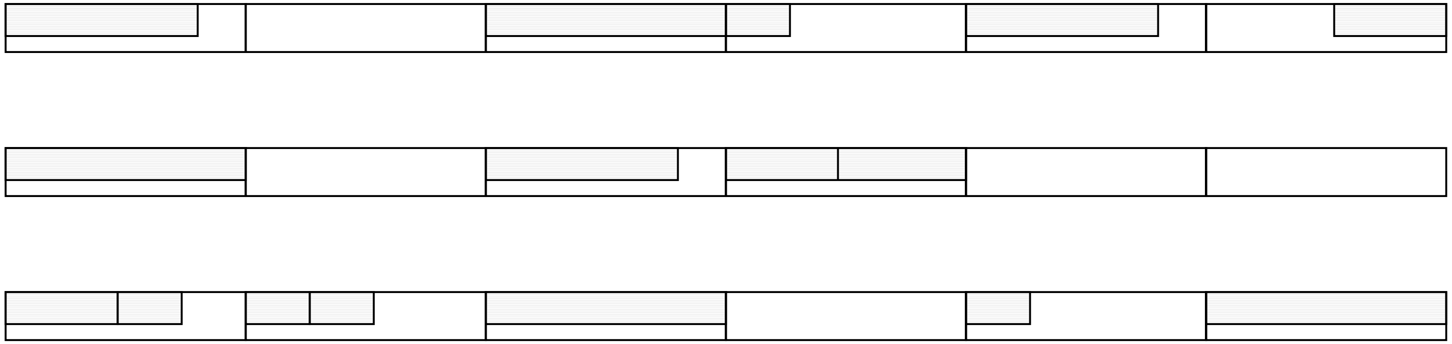
TIMETABLING: problem definition

- Timetabling concerns the definitions of agenda (similar to scheduling)

- Constraints
  - temporal restrictions
    - ordering among activities
    - due dates - release dates
  - resource capacity
    - discrete resources
- Optimization Criteria
    - cost/preferences
    - resource balance

TIMETABLING: simple example

- 4-Hours Slots - 1 to 4 Hours Courses
- Two courses cannot overlap
- A course must be contained in a single slot
- Preferences are associated with
  - Course-Slot assignments
  - Maximize Sum of preferences



TIMETABLING: code with redundant constraints

```
timetable(Data,Tasks,MaxTime,Costs):-
        define_variable_start(Tasks,MaxTime),
        define_variable_singleHours(Data,SingleHours),
        define_variable_courses3_4Hours(Data, Courses34Hours),
        impose_cumulative(Tasks),
        alldifferent(SingleHours),
        alldifferent(Courses34Hours),
        minimize(labelling(Tasks),Cost).
```

redundant constraints

- Redundant variables:
  - start times
  - single hours
  - courses lasting 3 or 4 hours

linked each other:

exchange propagation results

TIMETABLING: optimality & search

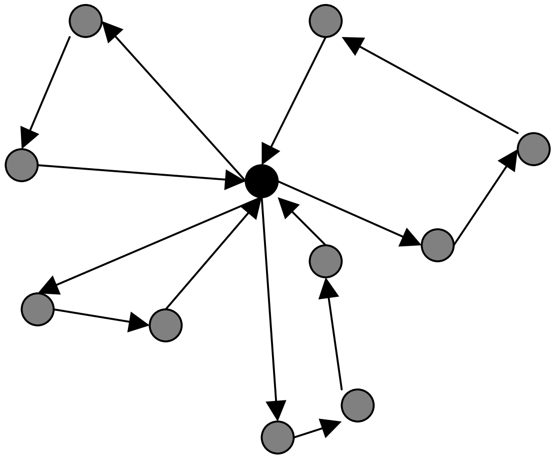
- Search strategy: when coping with objective functions, we can exploit information on costs for defining a good search strategy.

- Example:
  - Choose the variable with max value of regret
    - Regret: difference between the first and the second best on each row of the cost matrix.
    - Combination of regret and first-fail
  - Choose the value associated with the minimum cost
- Example: based on the optimal solution of a relaxation
  - Choose the variable with max value of regret
  - Choose the solution of the relaxation

ROUTING: problem definition

- Routing concerns the problem of finding a set of routes to be covered by a set of vehicles visiting a set of cities/customers once starting and ending at one (n) depot(s).

- Constraints
  - temporal restrictions:
    - time windows
    - maximal duration of a travel
  - vehicle capacity
  - customer demands
- Optimization Criteria
  - number of vehicles
  - travel cost
  - travel time



ROUTING: problem definition

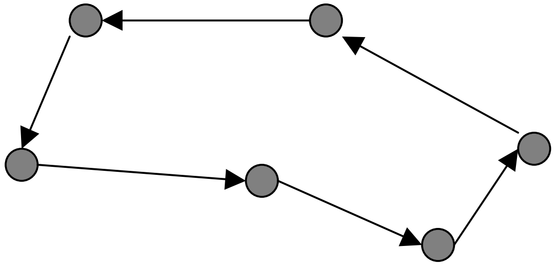
- Routing has been solved within OR community by using
  - Branch & Bound approaches
  - Dynamic Programming
  - Local Search techniques
  - Branch & Cut
  - Column generation

- Routing has been solved within CP community by using
  - Branch & Bound approaches
  - Local Search techniques embedded in CP

- Basic component: Travelling Salesman Problem (TSP) and its time constrained variant.

TSP: problem definition

- TSP is the problem of finding a minimum cost tour covering a set of nodes once.



- No subtours are allowed
- TSPTW: Time windows are associated to each node. Early arrival is allowed. Late arrival is not permitted
- Even finding an Hamiltonian Circuit (no costs) is NP-complete

TSP: CP model

- Variable associated to each node. The domain of each variable contains possible next nodes to be visited

- N nodes → N + 1 variables $\mathtt{Next}_{\mathtt{i}}$ (duplicate the depot)

- For all $\mathtt{i}$: $\mathtt{Next}_{\mathtt{i}} \neq \mathtt{i}$
- $\mathtt{nocycle([Next}_0\mathtt{,}\ldots\mathtt{Next}_n\mathtt{])}$
- $\mathtt{alldifferent([Next}_0\mathtt{,}\ldots\mathtt{Next}_n\mathtt{])}$
- Cost $\mathtt{c}_{\mathtt{ij}}$ if $\mathtt{Next}_{\mathtt{i}} = \mathtt{j}$

- In some models, we can find the redundant variables **`Prev`** indicating a node predecessor.

TSP: code

```
tsp(Data,Next,Costs):-
    remove_arcs_to_self(Next),
    nocycle(Next),
    alldifferent(Next),
    create_objective(Next,Costs,Z),
    minimize(labelling(Next),Z).
```

- **`nocycle`**: symbolic constraint that ensures that no subtour is present in the solution.
- **`create_objective`**: creates **`Costs`** variables, imposes an **`element`** constraint between the set of **`Next`** variables and **`Costs`** variables, and creates a variable **`Z`** representing the objective function summing costs corresponding to assignments

TSP: results

- Pure CP implementations: still far from the state of the art OR approaches.

- Integration of OR techniques in CP: better results
  - local search
  - optimal solution of relaxations
    - Lagrangean relaxation
    - Assignment Problem
    - Minimum Spanning Arborescence
  - search strategies based on these relaxations
    - subtour elimination

- Addition of Time Windows in OR approaches requires to re-write major code parts while in CP comes for free.

CUTTING & PACKING: problem definition

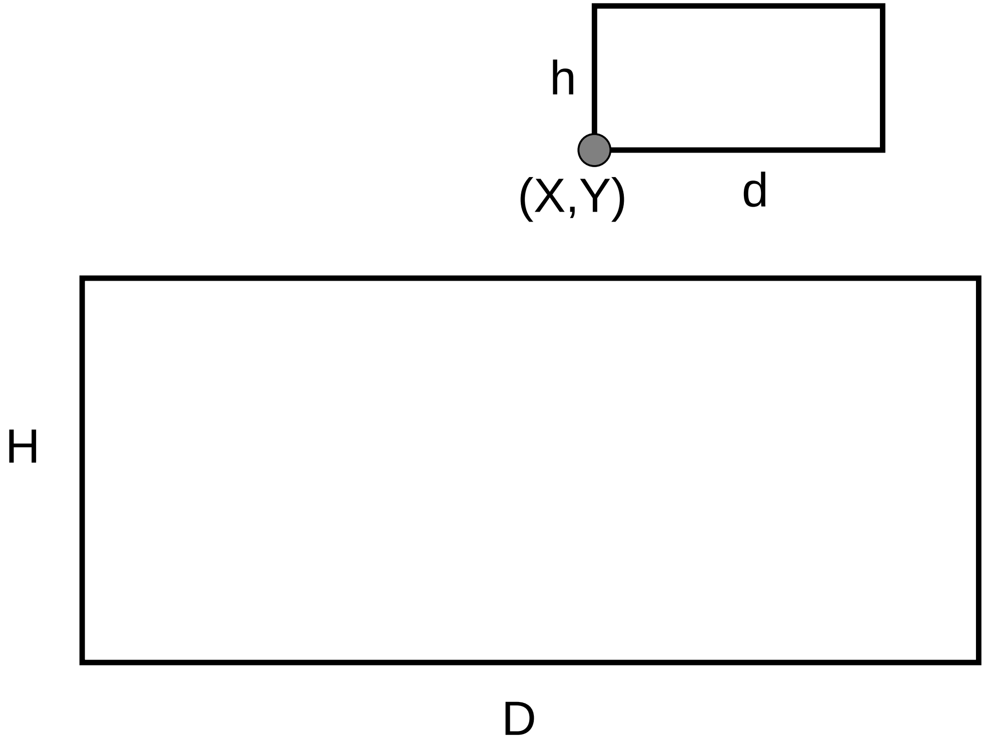
- Packing concerns the placement of a number of squares (of different sizes) in one or more larger boxes in such a way that squares do not overlap and minimizing the empty space

- Cutting is the problem of finding cuts of a master piece in order to obtain a given number of pieces with fixed dimensions, minimizing residues

- Many variants:
  - strip packing
  - guillottine cuts
  - rotations allowed
  - 1 dimension - 2 dimensions - 3 dimensions - 4 dimensions

2-D PACKING: CP model

- For each square to be packed, we have a couple of variables representing the coordinates of the bottom-left point of the square



Pieces:

`X::[0..D-d]`

`Y::[0..H-h]`

masterPiece or
bin

2-D PACKING: CP model

- Constraints:
  - non overlapping constraints: given two squares whose coordinates are (**`X1`**, **`Y1`**) and (**`X2`**, **`Y2`**) and dimensions **`D1`**, **`H1`** and **`D2`**, **`H2`** respectively

- **`X1+D1 ≤ X2 OR Y1+H1 ≤ Y2 OR X2+D2 ≤ X1 OR Y2+H2 ≤ Y1`**

- Very hard form of disjunction: no propagation even after instantiation

- Redundant constraints can help:
- **`cumulative(Xcoordinates,XDimension,Ydimension,H)`**
  `cumulative(Ycoordinates,YDimension,Xdimension,D)`

PACKING: code

```
packing(Data,Xcoords,Ycoords,D,H):-
    create_variables(Data,Xcoords,Ycoords,D,H),
    state_disjunctive(Data,Xcoords,Ycoords),
    state_cumulatives(Data,Xcoords,Ycoords,D,H),
    create_objective(Xcoords,Ycoords,D,H,Z),
    minimize(label_squares(Xcoords,Ycoords),Z).
```

- **`create_objective`**: creates a variable representing the spare space (or the number of bins if more than one is present)

- **`label_squares`** selects bigger squares first and assigns the coordinates in order to minimize spare space.

MODEL BASED VISION
OBJECT RECOGNITION : definition

- Object recognition in model based vision concerns the problem of recognizing an object in a scene given its model

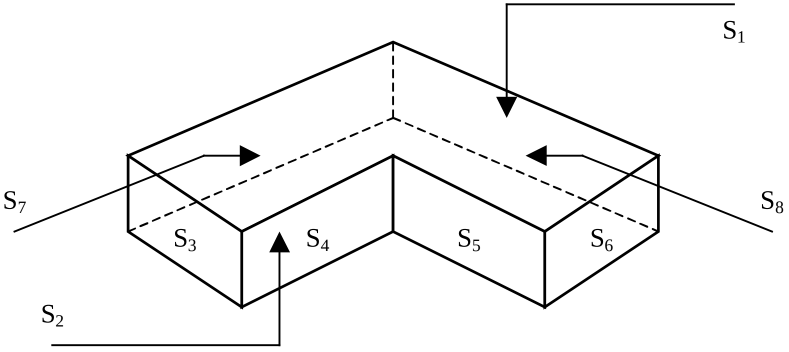
- How to describe the model
- How to perform the mapping

Both problems can be solved with constraints

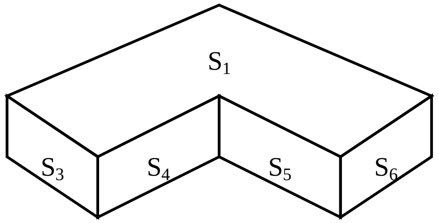
- MODEL: Constraint graph
  - Nodes: object parts
  - Arcs (constraints): their relations
- RECOGNITION: Constraint satisfaction

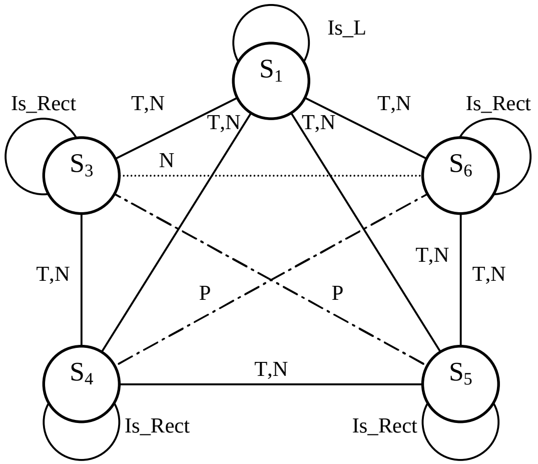
OBJECT RECOGNITION

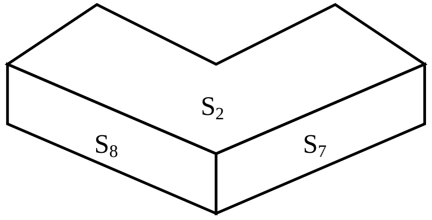
- 3-D OBJECT

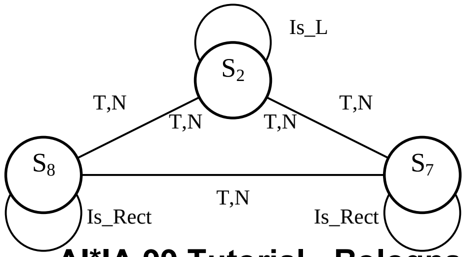


- Two views and the corresponding models









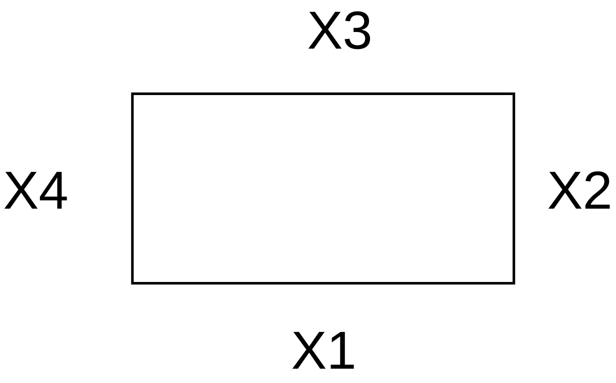
CONSTRAINT-BASED OBJECT RECOGNITION

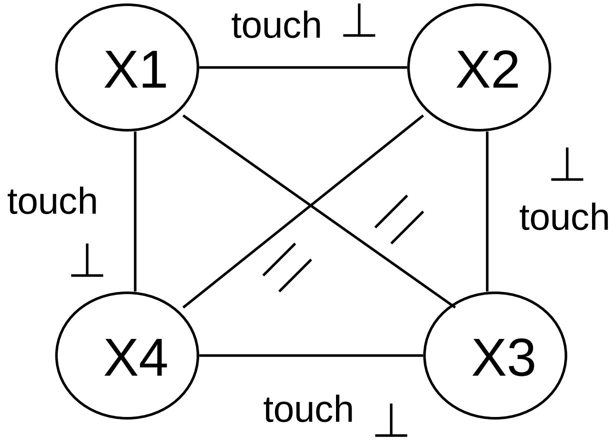
- In order to recognize an object, a low level vision system should extract from the image some visual features (surfaces/edges)
- Constraint satisfaction techniques can be applied in order to recognize the object in the scene.
- The object is recognized if the extracted features satisfy the constraints contained in the model.
- Constraints allow to reduce the search space to be explored.

EXAMPLE

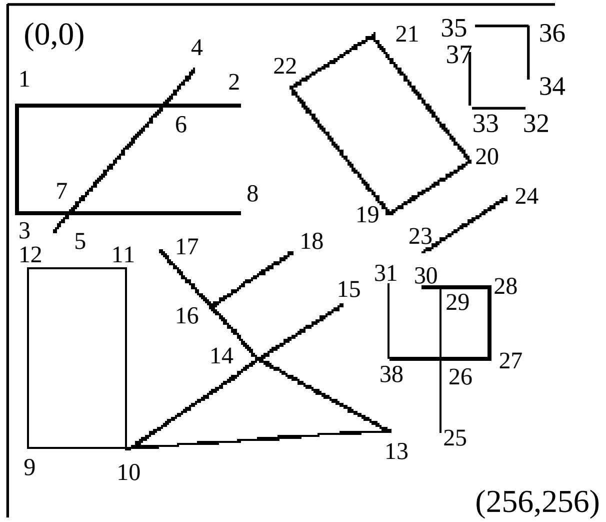
- Rectangle shape

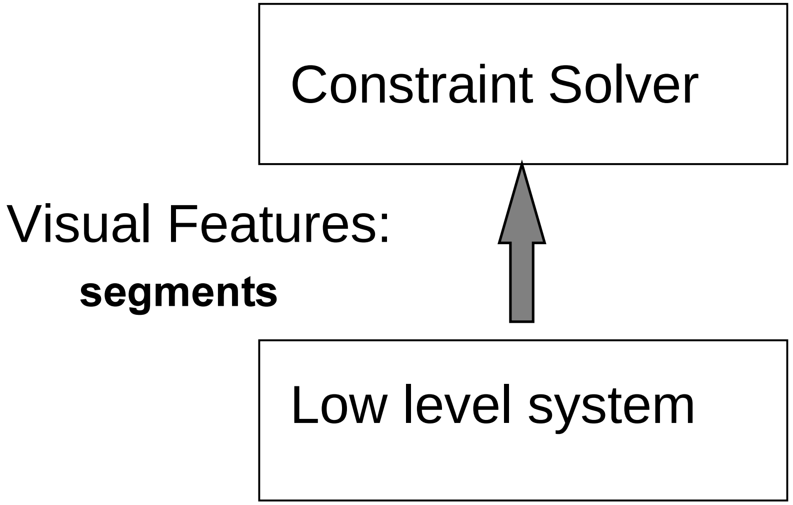
  Four straight edges (variables), parallel two by two, which mutually touch themselves with a 90 degree angle ...





EXAMPLE (2)





Constraint Solver

Visual Features:

segments

Low level system

rectangle CP model

```
touch(X1,X2), touch(X2,X3), touch(X3,X4), touch(X1,X4),
perpend(X1,X2), perpend(X2,X3), perpend(X3,X4), perpend(X1,X4),
same_len(X2,X4), same_len(X1,X3), parallel(X2,X4), parallel(X1,X3)
```

CP model: USER DEFINED CONSTRAINTS

rectangle CP model

```
recognize([X1,X2,X3,X4]):-
    X1,X2,X3,X4::[s1,s2,…,sn],
    touch(X1,X2), touch(X2,X3), touch(X3,X4), touch(X1,X4),
    perpend(X1,X2), perpend(X2,X3), perpend(X3,X4), perpend(X1,X4),
    same_len(X2,X4), same_len(X1,X3),
    parallel(X2,X4), parallel(X1,X3),
    labeling([X1,X2,X3,X4]).
```

- Give the declarative and operational semantics of the constraints: segments are described as facts: **`segment(name,X1,Y1,X2,Y2)`**

- In all CP languages there are tools that allow new constraints to be defined.

- An example in the CLP(FD) library of ECL^{i}PS^{e}

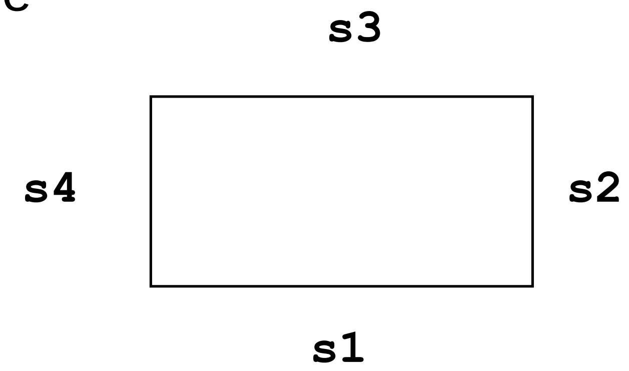
CP model: USER DEFINED CONSTRAINTS

```
touch(X1,X2) :-
   dvar_domain(X1,D1),
   dvar_domain(X2,D2),
   arc_cons_1(D1,D2,D1new),  % user defined propagation
   (dom_compare(>,D1,D1new) -> dvar_update(X1,D1new); true),
    arc_cons_2(D1new,D2,D2new), % user defined propagation
   (dom_compare(>,D2,D2new) -> dvar_update(X2,D2new); true),
   (var(X1),var(X2) ->
           (make_suspension(touch(X1,X2),3,Susp),
           insert_suspension((X1,X2), Susp, any of fd, fd))
   ;       true),
   wake.
```

- After the propagation, the constraint if not solved is suspended and awaked each time an event ***`any of fd`*** on one of the variables **`(X1,X2)`** happens.

CP model: SYMMETRIES

- Problem symmetries: arise when some permutations of the variables map a solution onto another solution.
- Four segments forming a rectangle
- One solution:
  - **X1 = s1**
  - **X2 = s2**
  - **X3 = s3**
  - **X4 = s4**



- Other identical solutions:
  - **X1 = s2   X1 = s3   X1 = s4**
  - **X2 = s3   X2 = s4   X2 = s1**
  - **X3 = s4   X3 = s1   X3 = s2**
  - **X4 = s1   X4 = s2   X4 = s3**

Time lost to look for already found solutions . Remove symmetries by imposing additional constraints

[Freuder AAAI91], [Puget ISMIS93], [Crawford et al. KR96], [Meseguer, Torras IJCAI99].

AI PLANNING: definition

- Planning concerns the problem of finding a chain of actions that achieve a given goal starting from a well known initial state. Actions are described with a set of preconditions (requirements) and postconditions (effects)

- Constraints
  - Plan constraints
    - temporal constraints (ordering)
    - designation and co-designation constraints
    - resource constraints
    - domain dependent constraints
  - Plan construction constraints
    - threats
    - open condition achievement

CONSTRAINTS ON THE PLAN

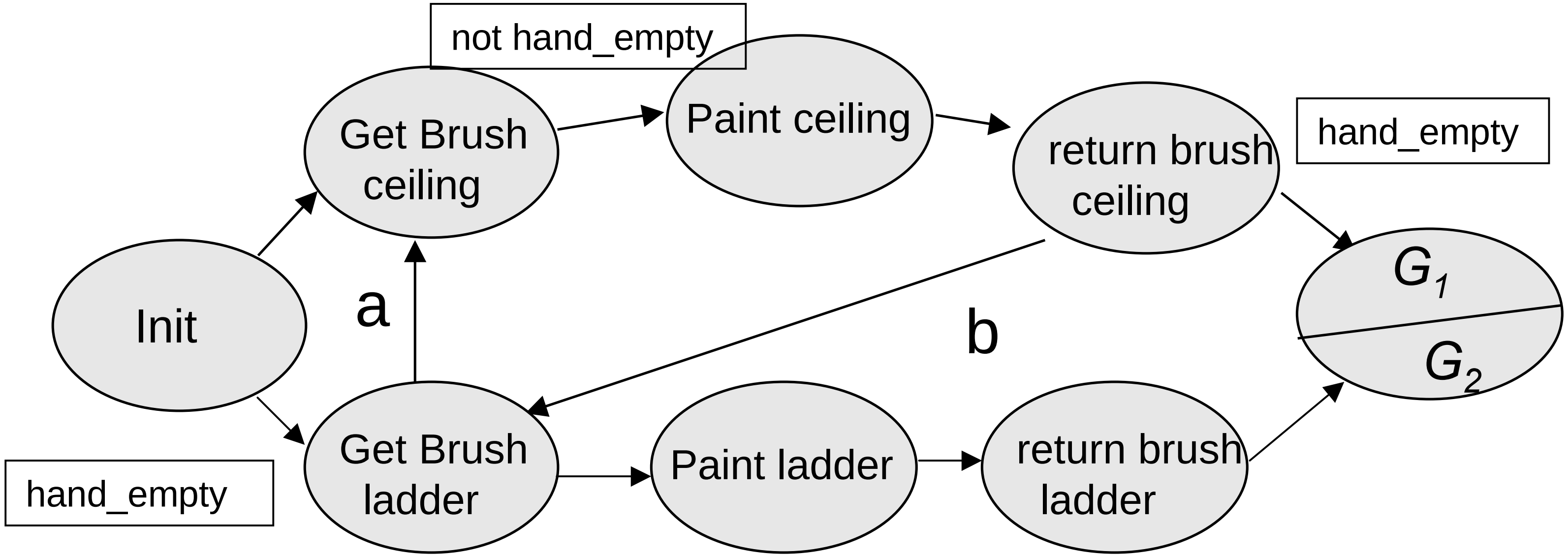
- Temporal constraints
  - qualitative → action A before action B
  - quantitative → action A in [10..30]
- Resource constraints
  - consumable resources
  - shared resources
  - renewable resources
- Domain-dependent constraints
  - priority among resources
  - destructive actions on object A always after any other action on A

CONSTRAINTS ON PLAN CONSTRUCTION

- During the plan construction a set of decisions should be taken:
  - threat resolution
  - open condition achievement
- Good technique: least commitment. Decisions are delayed as much as possible in order to perform only consistent choices.
- Passive postponement vs. active postponement
- Each decision is represented by a variable whose domain contains possible solution to the pending decision. Constraints take into account interaction among decisions.

THREAT RESOLUTION

- Threat: conflict occurring when the effects of an action A threat the preconditions of an action B



Robot with one hand: threat

a promotion: getBrushLadder < getBrushCeiling

b demotion: returnBrushCeiling < getBrushLadder

THREAT RESOLUTION

- Variable associated with the threat:
  - T1 :: [*getBrushLadder* < *getBrushCeiling,*

    returnBrushCeiling < *getBrushLadder*]
  - Threat variables linked by:
    - *incompatibility constraints*
    - *subsumption constraints*
  - Example: if a threat T2 can be solved only by the constraint *getBrushCeiling* < *getBrushLadder,* the only way of solving T1 becomes *returnBrushCeiling* < *getBrushLadder*

- Propagation of consequences of the decisions. Reduction of the search space and trashing avoided

- More complex solver: user defined constraints.

PLANNERS based on CP/CSPs

- ParcPlan ***[Lever, Richards, ISMIS94]***
- PlaNet ***[Barruffi, Milano ECAI98], [Barruffi et. Al. ECP99]***
- O-Plan ***[Tate, Drabble, Dalton, TR Univ. Edinburg 95]***
- Molgen ***[Stefik AIJ(16), 81]***
- DEVISER ***[Vere IJCAI85]***
- Descartes ***[Joslin PhD, 95], [Joslin, Pollac, EWSP96]***
- WatPlan ***[Yang, AIJ(58), 92]***
- SLNP ***[McAllester, Rosenblitt Nat.Conf AI 91]***
- FSNLP ***[Yang, Chan AIPS94]***
- Kambampati ***[TR Arizona State Univ. 96]***
- SIPE ***[Wilkins AI (22) 84]***

OTHER APPLICATIONS of CP

- Constraint Databases
- Spreadsheet
- Robotics/Control
- Diagnosis
- Test data generation
- Circuit Verification
- Natural Language
- Graphical Interfaces
- Graphical Editors
- Biology (DNA sequencing)
- Qualitative reasoning
- Temporal reasoning
- SAT
- Other LSCO problems

OVERVIEW

- Constraint Satisfaction (Optimization) Problems
- Constraint (Logic) Programming
  - *language and tools*
- AI Applications: modelling and solving
  - *Scheduling - Timetabling - Resource Allocation*
  - *Routing*
  - *Packing - Cutting*

  - *Graphics - Vision*
  - *Planning*
- Advantages and Limitations of CP: extensions

ADVANTAGES OF CP

- Easy problem modelling
- Constraints provide a natural way of implementing propagation rules
- Flexible treatment of variants of original problems:
  - easy introduction of new constraints
  - transparent interaction with other constraints
- Easy control of the search

LIMITATIONS of PURE CP

- Optimization side not very effective
- Over-Constrained problems:
  - no effective way of relaxing constraints
  - hard/soft constraints
- Dynamic Changes:
  - addition/deletion of variables
  - addition/deletion of domain values
  - addition/deletion of constraints

CP EXTENSION FOR OPTIMIZATION

- Integration of OR techniques in CP tools:
  - MP based solvers: 2LP ***[McAloon, Tretkoof PPCP94],*** OPL ***[Van Hentenryck, 99],*** Planner ***[ILOG Planner Manual]***
    - Integration of CPLEX and XPRESS in FD solvers
  - Integration of specialized algorithms for:
    - computing bounds
    - using reduced costs

    [Caseau, Laburthe ICLP97 and CP97], [Focacci, Lodi, Milano ICLP99 and CP99],
  - Improvement of CP branch and bound
    - ***[Rodosek, Wallace, Hajian Annals OR 97], [Caseau, Laburthe ICLP94 and JICSLP96], [Beringer, DeBacker, LP Formal Method and Pract. Appl. 95]***
  - Integration of local search techniques
  - ***[DeBacker, Furnon,Shaw CPAIOR99], [Caseau, Laburthe CPAIOR99], [Gendreau, Pesant, Rousseau, Transp. Sci. 98]***
  - Integration of branch and cut in a logical setting
    - ***[Bockmayr ICLP95], [Kasper PhD, 99]***

CP EXTENSION FOR OVER-CONSTRAINED PROBLEMS

- HCSP:
  - Implementation of CP solvers exploiting Hierarchical CSP framework
  - Meta programming

[A. Borning OOPSLA87], [A. Borning et al. ICLP89], [M.Jamper PhD, 96]

CP EXTENSION FOR DYNAMIC CHANGES

- DCSP
- ATMS-based solvers
- Interactive Constraint Satisfaction

(DCSP, ATMS-based solvers:) Complex data-structures

[R.Dechter, A.Dechter, AAAI88], [Verfaillie, Schiex AAAI94], [Bessiere, AAAI91], [Lamma et al. IJCAI99]

TO KNOW MORE…..

- *Conferences:*
  - *International Conference on Principles and Practice of Constraint Programming CP*
  - *International Conference on Practical Applications of Constraint Technology PACT (this year PACLP)*
  - *Logic programming conferences (ILPS - ICLP - JICSLP)*
  - *AI Conferences (ECAI - AAAI - IJCAI)*
  - *Operations research conferences (INFORMS - IFORS)*
  - *New Workshop (CP-AI-OR)*

- *Book : K. Marriott and P. Stuckey*
  - *Programming with constraints: An Introduction*
  - *MIT Press*

TO KNOW MORE…..

- *Journals:*
  - *Constraint - An International Journal*
  - *AI - LP - OR Journals*

- *Industrial Applications:*
  - *COSYTEC, ILOG, ECRC, SIEMENS, BULL*

- *News group:* **`comp.constraints`**

- *Mailing lists:* **`CPWORLD@gmu.edu`**

- *Constraint Archive:* **`http://www.cs.unh.edu/ccc/archive/`**