Global Constraints

Amira Zaki
Prof. Dr. Thom Frühwirth

University of Ulm

WS 2012/2013

Overview

Classes of Constraints

Global Constraints

Case Study 1: `all_different`

Global Constraint as Graph Problem

Case Study 2: `cumulative`

Combinatorial Problem Representation

- Many possible models for a given problem
- Represent the entities of the problem
  - Variables and domains
  - Constraints that apply to these variables
- Test the model and try to improve its solving efficiency, try:
  - Global constraints
  - Redundant constraints
  - Dual models

Expressing the Constraints

- Different ways can lead to different algorithm behavior
- Example of two programs having the same solutions but consistency propagators produce different results

P1:

```
X in 1..5,
Y in 0..7,
Z in 1..9,
X = Y,
X + Y = Z
```

$\Longrightarrow$ `X,Y in 1..5, Z in 2..9`

P2:

```
X in 1..5,
Y in 0..7,
Z in 1..9,
X = Y,
2*X = Z
```

$\Longrightarrow$ `X,Y in 1..4, Z in 2..8`

Expressing the Constraints

- Bring different viewpoints of the problem
- A "good" viewpoint
  - Problem is easily expressed
  - Few constraints with efficient propagators
    - The inequality constraint is rarely a good choice $\neq$
  - Knowing the different types of constraints offered by the solver
    - Global constraints tend to be more effective
    - Capture more semantics of the problem structure
  - Implied or redundant constraints

Classes of Constraints

1. Basic constraints
   - For which the solver is complete
   - Stored in the *constraint store*
   - $x \in D, x = v$
2. Primitive constraints
   - Cannot be meaningfully decomposed
   - Implemented as propagators
   - $x < y, x = y, x + y = z, \ldots$
3. Global Constraints

Global Constraints

- Global Constraints is one of the main trends in todays CP-research and practice
- Algorithms that can be implemented as incremental filtering mechanisms integrate well in to the general CP framework
- Many such algorithms from matching theory, flow optimization and graph algorithms have proved to be most effective as specialized modeling components in such frameworks

Global Constraints

- Captures a relation between a non-fixed number of variables
- Subsumes a set of basic and or primitive constraints, thus less constraints needed
- Encode complex or high level modeling concepts
- May give (significantly) stronger propagation than corresponding set of primitive constraints
- Case studies
  - `all_different`
  - `cumulative`

Global Constraint: all_different

`all_different(+Variables)`

where `Variables` is a list of domain variables with bounded domains or integers. Each variable is constrained to take a value that is unique among the variables, hence the variables are *pairwise distinct*
Or formally,

$$all\_different([x_1, \ldots, x_n]) = \{[d_1, \ldots, d_n] \mid \forall_i d_i \in D(x_i),\ \forall_{i=j}\ d_i \neq d_j\}.$$

all_different

It is a *filtering* constraint; all values that do not belong to any solution of the constraint are filtered from its domain

Example

`all_different`$([X, Y, Z]), X$ `in` $[1, 2], Y$ `in` $[1, 2], Z$ `in` $[1, 2, 3]$
The two variables X and Y take the values 1 and 2, while z cannot take these values. Thus by filtering, $Z = 3$

Sudoku

Fill a $9 * 9$ grid with digits from 1 to 9 so that each column, each row, and each of the nine 3×3 sub-grids that compose the grid contains each digit exactly once

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Sample Sudoku puzzle (Wikipedia)

Modeling of Sudoku

- Variables
  - One for each cell
  - A 9×9 matrix of variables for the Sudoku grid
- Domains
  - Each variable has a domain from 1 to 9
- Constraints
  - Pairwise inequality constraints $\neq$ among cells of the same row, column and 3×3 block
  - **Too many!**

Better Modeling of Sudoku

- One global constraint for each row, column and 3×3 block `all_different(Rows)`, `all_different(Columns)`, `all_different(Blocks)`
- Seek sets of values that must be matched to some variables; they can not be assigned to other variables
- Reformulate the constraints using a bipartite graph

Bipartite Graph Definition

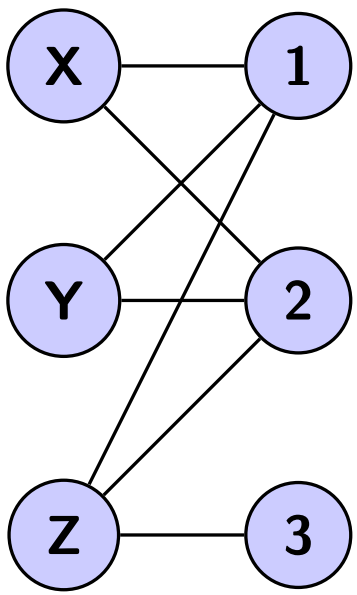
A graph G consists of a finite non-empty set of elements V called *nodes* and a set of unordered pairs of nodes E called *edges*. If the node set V can be partitioned into two disjoint non-empty sets X and Y such that all edges in E join a node from X to a node in Y, we call G bipartite with bipartition (X, Y)

$$G = (X, Y, E)$$

For Global Constraints,

- X = *Variables*
- Y = *Values*

all_different as Graph Problem



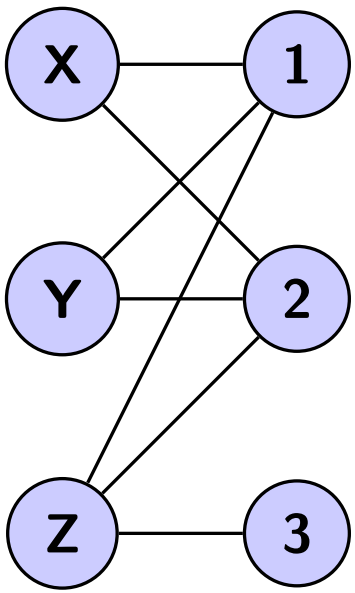

```
graph :-
  X in 1..2,
  Y in 1..2,
  Z in 1..3,
  all_different([X,Y,Z]).
```

Matching

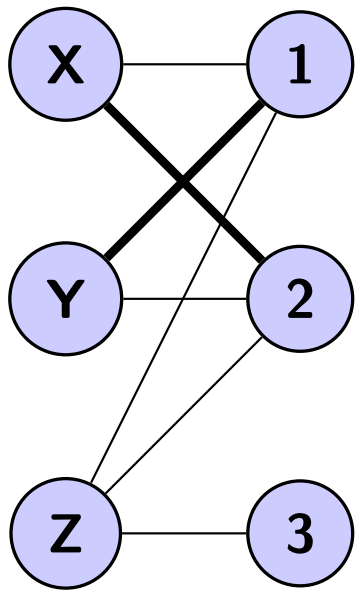
- Definitions
  - **Matching**: a subset of edges in a graph G where no two edges have a node in common
  - **Maximum matching**: a matching of maximum cardinality i.e. has largest set of edges in a graph that form a matching
- Every solution to the `all_different` corresponds to a maximal matching
- If a link does not belong to a maximum matching, it can be removed (domain reduction)

Example

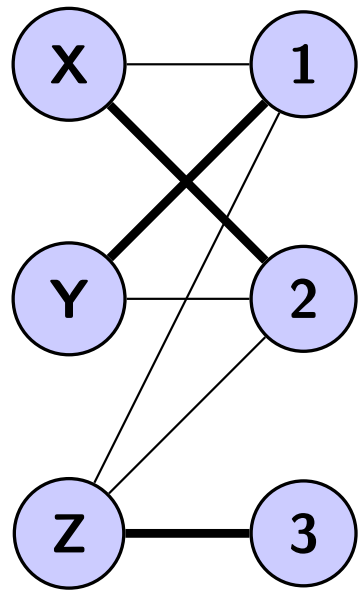
X in $[1, 2]$, Y in $[1, 2]$, Z in $[1, 2, 3]$



Bipartite Graph



Matching



Maximum Matching

Hall's Theorem

Hall's marriage Theorem (Hall, 1935)

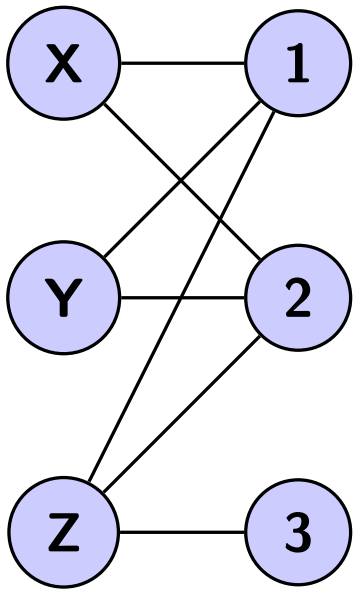
> *If a group of men and women marry only if they have been introduced to each other previously, then a complete set of marriages is possible if and only if every subset of men has collectively been introduced to at least as many women, and vice versa*

Meaning:
all_different$([x_1, \ldots, x_n])$ has a solution if and only if:

$$|K| \leq |D(K)| \text{ for all } K \subseteq \{x_1, \ldots, x_n\}$$

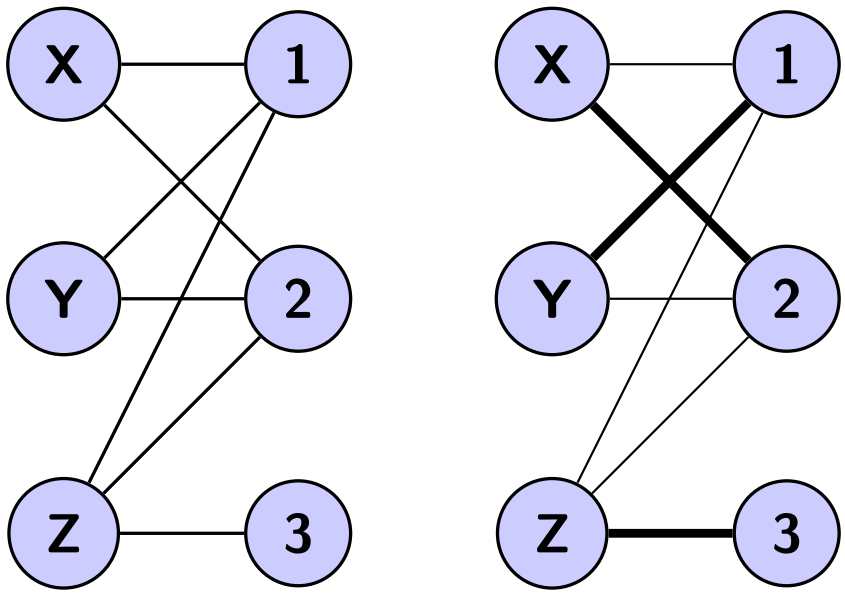
Step 1: Maximum Matching Computation

- Largest number of edges such that no two edges share an endpoint
- Take a bipartite graph produce maximum cardinality matching
- Increase the size of partial matching by finding alternating paths that starts from and ends on free (unmatched) nodes



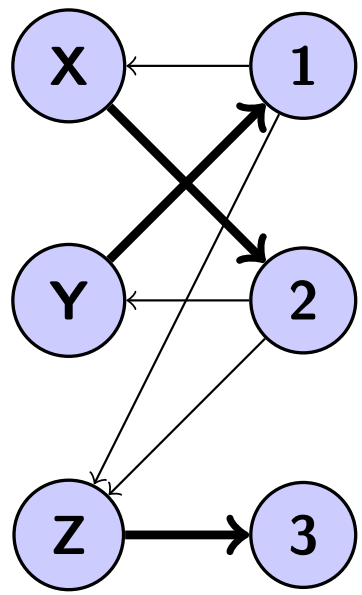
Step 1: Maximum Matching Computation

- Largest number of edges such that no two edges share an endpoint
- Take a bipartite graph produce maximum cardinality matching
- Increase the size of partial matching by finding alternating paths that starts from and ends on free (unmatched) nodes



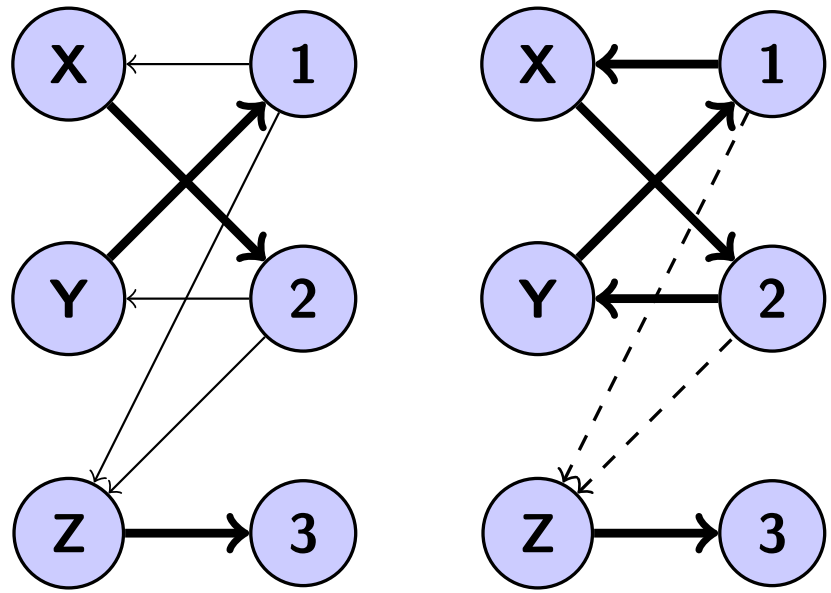
Step 2: Strongly Connected Components

1. Orient the edges (Matching edges from variables to values, other from values to variables)



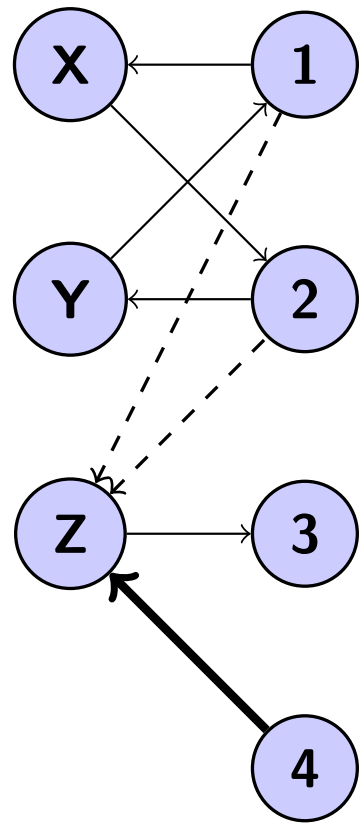
Step 2: Strongly Connected Components

1. Orient the edges (Matching edges from variables to values, other from values to variables)
2. Compute maximal subgraph of a directed graph in which every vertex is reachable from every other vertex: aka **strongly connected components**



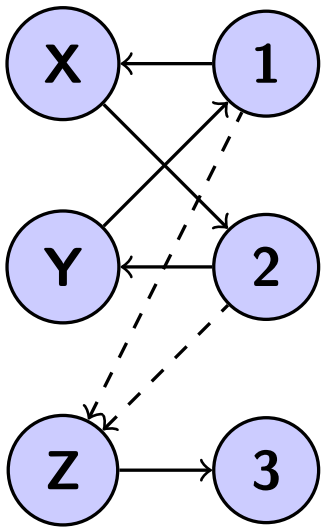
Step 3: Mark More Edges

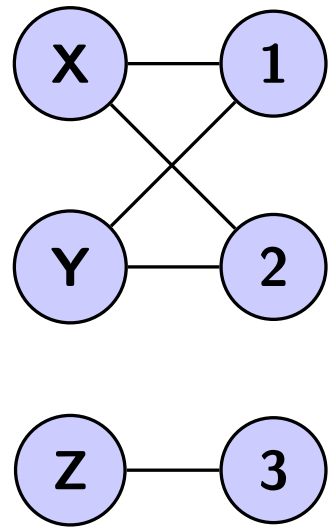
1. Find unmatched value nodes (here none)
2. Mark alternative paths for these nodes, regardless if these edges belonged or not to matching



Step 4: Pruning

- Remove all unmarked edges (i.e. dotted)





- Corresponds to pruning stronger than binary constraints:

$$X \text{ in } [1,2], Y \text{ in } [1,2], Z = 3$$

Resource Scheduling

- Planning of temporal order of *tasks (jobs)* in presence of limited resources
  - task, e.g., production step or lecture
  - *resource* e.g., a machine, electrical energy, or lecture room
  - tasks compete for limited resources
  - dependencies between tasks
  - find a schedule with an optimal value for a given objective function (measuring time or use of other resources)
- *job shop scheduling problem*
  - tasks have fixed duration and cannot be interrupted
  - resources are machines for at most one task at a time
  - objective is to minimize the overall production time that is needed to accomplish all the tasks

Global Constraint: cumulative

`cumulative(+Tasks,?Limit)`

- Tasks is a list of tasks, each of form $task(S_i, D_i, E_i, R_i, T_i)$. S_i denotes start time, D_i positive duration, E_i end time, R_i non-negative resource consumption, T_i task identifier
- Each of these arguments must be a finite domain variable with bounded domain or an integer
- The constraint holds if at any time during the start and end of each task, the total resource consumption of all tasks running at that time does not exceed the global resource limit stated

Global Constraint: cumulative

Given positive *Durations*, *Resources* and *Limit*, let:

$$a = min(S_1, ..., S_n)$$
$$b = max(S_1 + D_1, ..., S_n + D_n)$$

and

$$R_{ij} = \begin{cases} R_j & \text{if } S_j \leq i < S_j + D_j \\ 0 & \text{otherwise} \end{cases}$$

The constraint is True iff:

$$(\forall a \leq i < b) \left(\sum_{j=1}^{n} R_{ij} \right) \leq Limit$$

Moving Furniture Example

- We would like to move the following items:

Item	*Movers*	*Time/minutes*
Piano	3	30
Bed	3	15
Chair	1	10
Table	2	15

- We want to finish moving in one hour
- Only 4 people are available to move

Moving Furniture Example

Solution requires **one** cumulative constraint:

- Starts: [SP,SB,SC,ST]
- Ends: [EP,EB,EC,ET]
- Durations: [30,15,10,15]
- Resources: [3,3,1,2]
- TaskIds: [p,b,c,t]
- Limit: 4

```
:- use_module(library(clpfd)).   % with SWI-Prolog 6.3.8
move(Tasks):-
  Tasks = [task(SP,DP,EP,RP,TP),    task(SB,DB,EB,RB,TB),
           task(SC,DC,EC,RC,TC),    task(ST,DT,ET,RT,TT)],
  Starts = [SP,SB,SC,ST],    Ends = [EP,EB,EC,ET],
  Durations = [DP,DB,DC,DT], Durations = [30,15,10,15],
  Resources = [RP,RB,RC,RT], Resources = [3,3,1,2],
  TaskIds = [TP,TB,TC,TT],   TaskIds = [p,b,c,t],
  Limit = 4,
  Starts ins 0..60,
  Ends ins 0..60,
  cumulative(Tasks, [limit(Limit)]),
  label([SP,SB,SC,ST]).

 % a solution: [task(0,30,30,3,p), task(30,15,45,3,b),
                task(0,10,10,1,c), task(45,15,60,2,t)]
```