

Local Consistency, Constraint Propagation & Global Constraints



Constraint Propagation

- Examination of the constraints to **remove incompatible (inconsistent) values** from the domains of the **future (unexplored) variables**.
 - Values that cannot be part of any solution.

Local Consistency

- A form of inference which **detects inconsistent partial assignments**.
 - Local, because we examine individual constraints.
- Popular local consistencies are domain-based.
 - They detect inconsistent partial assignments of the form $X_i = j$, hence:
 - j can be **removed** from $D(X_i)$ via **propagation**;
 - propagation can be implemented easily.

Generalised Arc Consistency (GAC)

- Also referred to as hyper-arc or domain consistency.
- A constraint C defined on k variables $C(X_1, \dots, X_k)$ gives the set of allowed combinations of values (i.e. tuples).
 - $C \subseteq D(X_1) \times \dots \times D(X_k)$
 - Each allowed tuple $(d_1, \dots, d_k) \in C$ is a **support** for C .
- $C(X_1, \dots, X_k)$ is GAC iff:
 - for all X_i in $\{X_1, \dots, X_k\}$, for all $v \in D(X_i)$, v belongs to a support.
- Arc Consistency (AC) when $k = 2$.
- A CSP is GAC iff all its constraints are GAC.

Examples

- $D(X_1) = \{1,2,3\}$, $D(X_2) = \{2,3,4\}$, **C**: $X_1 = X_2$
 - $AC(\mathbf{C})?$
 - $1 \in D(X_1)$ and $4 \in D(X_2)$ do not have a support.
 - $X_1 = 1$ and $X_2 = 4$ are inconsistent partial assignments.
- $D(X_1) = \{1,2,3\}$, $D(X_2) = \{1,2\}$, $D(X_3) = \{1,2\}$,
C: **alldifferent** $([X_1, X_2, X_3])$
 - $GAC(\mathbf{C})?$
 - $1 \in D(X_1)$ and $2 \in D(X_1)$ do not have support.
 - $X_1 = 1$ and $X_1 = 2$ are inconsistent partial assignments.

Constraint Propagation

- A **local consistency** notion defines the properties that a constraint **C** must satisfy **after constraint propagation**.
 - The operational behaviour is completely left open (any algorithm could do).
 - The only requirement is to achieve the required property on **C**.
- A constraint propagation algorithm achieves a certain level of consistency by removing the inconsistent values from the domains of the variables in **C**.

Examples

- $D(X_1) = \{1,2,3\}$, $D(X_2) = \{2,3,4\}$, $C: X_1 = X_2$
 - $AC(C)$?
 - $1 \in D(X_1)$ and $4 \in D(X_2)$ do not have a support.
 - $X_1 = 1$ and $X_2 = 4$ are inconsistent partial assignments.
 - $D(X_1) = \{\cancel{1},2,3\}$, $D(X_2) = \{2,3,\cancel{4}\}$, C is now AC.
- $D(X_1) = \{1,2,3\}$, $D(X_2) = \{1,2\}$, $D(X_3) = \{1,2\}$,
 $C: \text{alldifferent}([X_1, X_2, X_3])$
 - $GAC(C)$?
 - $1 \in D(X_1)$ and $2 \in D(X_1)$ do not have support.
 - $X_1 = 1$ and $X_1 = 2$ are inconsistent partial assignments.
 - $D(X_1) = \{\cancel{1},\cancel{2},3\}$, $D(X_2) = D(X_3) = \{1,2\}$, C is now GAC.

Constraint Propagation

- The level of consistency depends on **C**.
 - GAC if an efficient propagation algorithm can be developed.
 - Otherwise BC or a lower level of consistency.

Bounds Consistency (BC)

- Defined for totally ordered (e.g. integer) domains.
- Relaxes the domain of X_i from $D(X_i)$ to $[\min(X_i)..\max(X_i)]$.
 - E.g., $D(X_i) = \{1,3,5\} \rightarrow [1..5]$
- A **bound support** is a tuple $(d_1, \dots, d_k) \in \mathbf{C}$ where $d_i \in [\min(X_i)..\max(X_i)]$.
- $\mathbf{C}(X_1, \dots, X_k)$ is BC iff:
 - For all X_i in $\{X_1, \dots, X_k\}$, $\min(X_i)$ and $\max(X_i)$ belong to a bound support.

Bounds Consistency (BC)

- Disadvantage

- BC might not detect all GAC inconsistencies in general.
 - We need to search more.

- Advantages

- Might be easier to look for a support in a range than in a domain.
- Achieving BC is often cheaper than achieving GAC.
 - Of interest in arithmetic constraints defined on integer variables with large domains.
- Achieving BC is enough to achieve GAC for monotonic constraints.

GAC vs BC

- $D(X_1) = D(X_2) = D(X_3) = \{1,3\}$
C: alldifferent($[X_1, X_2, X_3]$)
 - BC(**C**)
 - All $\min(X_i)$ and $\max(X_i)$ belong to a bound support.
 - No domain reduction with BC propagation.
 - Not GAC(**C**)
 - None of $\min(X_i)$ and $\max(X_i)$ belongs to a support.
 - **C** fails with GAC propagation.
- $D(X_1) = [2..6]$, $D(X_2) = [1..5]$
C: $X_1 \leq X_2$
 - All values of $D(X_1) \leq \max(X_2)$ are AC.
 - All values of $D(X_2) \geq \min(X_1)$ are AC.
 - Enough to adjust $\max(X_1)$ and $\min(X_2)$.
 - $\max(X_1) \leq \max(X_2)$
 - $\min(X_1) \leq \min(X_2)$
 - $D(X_1) = [2..5]$, $D(X_2) = [2..5]$
 - AC(**C**), BC(**C**)



Propagation Algorithms

- When solving a CSP with multiple constraints:
 - propagation algorithms interact;
 - a propagation algorithm can wake up an already propagated constraint to be propagated again!
 - in the end, propagation reaches a fixed-point and all constraints reach a level of consistency;
 - the whole process is referred as **constraint propagation**.

$$\begin{array}{lcl}
 X_1 & : & \{\cancel{1}, 2, 3\} \\
 X_2 & : & \{1, 2\} \\
 X_3 & : & \{1, 2\}
 \end{array}
 \quad
 \begin{array}{ccc}
 X_1 & X_2 & X_3 \\
 3 & 1 & 2 \\
 3 & 2 & 1
 \end{array}$$

Example

- $D(X_1) = D(X_2) = D(X_3) = \{1, 2, 3\}$
 C_1 : alldifferent($[X_1, X_2, X_3]$) C_2 : $X_2 < 3$ C_3 : $X_3 < 3$
- Let's assume:
 - the order of propagation is C_1, C_2, C_3 ;
 - propagation algorithms maintain (G)AC.
- Propagation of C_1 :
 - nothing happens, C_1 is GAC.
- Propagation of C_2 :
 - $D(X_2) = \{1, 2, \cancel{3}\}$, C_2 is now AC.
- Propagation of C_3 :
 - $D(X_3) = \{1, 2, \cancel{3}\}$, C_3 is now AC.
- C_1 is not GAC anymore, because the supports of $\{1, 2\} \in D(X_1)$ in $D(X_2)$ and $D(X_3)$ are removed by the propagation of C_2 and C_3 .
- Re-propagation of C_1 :
 - $D(X_1) = \{\cancel{1}, \cancel{2}, 3\}$, C_1 is now GAC.

Properties of Propagation Algorithms

- It may not be enough to remove inconsistent values from domains once.
- A propagation algorithm **must wake up when necessary**, otherwise may not achieve the desired local consistency property.
- Events that may trigger a constraint propagation:
 - when a variable is assigned
 - when the domain of a variable changes (for GAC);
 - when the domain bounds of a variable changes (for BC);
 - ...

$$X_1 : \{\cancel{1}, 2, \cancel{3}\}$$

$$X_2 : \{1, \cancel{2}, 3\}$$

$$X_3 : \{1, \cancel{2}, 3\}$$

Example

- $D(X_1) = D(X_2) = D(X_3) = \{1, 2, 3\}$
 C_1 : alldifferent($[X_1, X_2, X_3]$) C_2 : $X_2 \neq 2$ C_3 : $X_3 \neq 2$
- Let's assume:
 - the order of propagation is C_1, C_2, C_3 ;
 - the alldifferent propagation algorithm maintains BC.
- Propagation of C_1 :
 - nothing happens, C_1 is GAC.
- Propagation of C_2 :
 - $D(X_2) = \{1, \cancel{2}, 3\}$, C_2 is now AC.
- Propagation of C_3 :
 - $D(X_3) = \{1, \cancel{2}, 3\}$, C_3 is now AC.
- Does the propagator of C_1 wake up again?
- What happens if search assigns $X_1 = 1$?

NO, MIN, MAX DON'T CHANGE

Complexity of Propagation Algorithms

- Assume $|D(X_i)| = d$.
- Following the definition of the local consistency property:
 - one time AC propagation on a $C(X_1, X_2)$ takes $O(d^2)$ time.
- We can do better!

Examples

- **C**: $X_1 = X_2$
 - $D(X_1) = D(X_2) = D(X_1) \cap D(X_2)$
 - Complexity: the cost of the set intersection operation
 - When should the propagation algorithm wake up?
- **C**: $X_1 \neq X_2$
 - When $D(X_i) = \{v\}$, remove v from $D(X_j)$.
 - Complexity: $O(1)$
 - When should the propagation algorithm wake up?
- **C**: $X_1 \leq X_2$
 - $\max(X_1) \leq \max(X_2), \min(X_1) \leq \min(X_2)$
 - Complexity: $O(1)$
 - When should the propagation algorithm wake up?

Specialized Propagation

- Propagation **specific** to a given **constraint**.
- Advantages
 - Exploits the constraint semantics.
 - Potentially much more efficient than a general propagation approach.
- Disadvantages
 - Limited use.
 - Not always easy to develop one.
- Worth developing for recurring constraints.

Global Constraints

- Capture **complex, non-binary and recurring combinatorial substructures** arising in a variety of applications.
- Embed **specialized propagation** which exploits the substructure.

Benefits of Global Constraints

- Modelling benefits
 - Reduce the gap between the problem statement and the model.
 - May allow the expression of constraints that are otherwise not possible to state using primitive constraints (**semantic**).
- Solving benefits
 - Strong inference in propagation (**operational**).
 - Efficient propagation (**algorithmic**).

Some Groups of Global Constraints

- Counting
- Sequencing
- Scheduling
- Ordering
- Balancing
- Distance
- Packing
- Graph-based
- ...

Counting Constraints

- Restrict the number of variables satisfying a condition or the number of times values are taken.

Alldifferent Constraint

- **alldifferent**($[X_1, X_2, \dots, X_k]$) holds iff
$$X_i \neq X_j \text{ for } i < j \in \{1, \dots, k\}$$
 - permutation constraint with $|D(X_i)| = k$.
 - **alldifferent**([3,5,2,1,4])
- Useful in a variety of context, like:
 - puzzles and graph problems (e.g., sudoku and map colouring);
 - timetabling (e.g. allocation of activities to different slots);
 - tournament scheduling (e.g. a team can play at most once in a week);
 - configuration (e.g. a particular product cannot have repeating components).

Global Cardinality Constraint

- Constrains the number of times each value is taken by the variables.
- $\text{gcc}([X_1, X_2, \dots, X_k], [v_1, \dots, v_m], [O_1, \dots, O_m])$ iff
for all $j \in \{1, \dots, m\}$ $O_j = |\{X_i \mid X_i = v_j, 1 \leq i \leq k\}|$
– $\text{gcc}([1, 1, 3, 2, 3], [1, 2, 3, 4], [2, 1, 2, 0])$
– alldifferent when $O_j \leq 1$.
- Useful e.g. in:
 - resource allocation (e.g. limit the usage of each resource).

THE NUMBER OF VALUES IN {CERTAIN VALUES}
MUST BE AMONG A LOWER AND UPPER
BOUND

Among Constraint

- Constrains the number of variables taking certain values.
- **among**($[X_1, X_2, \dots, X_k], \textcircled{v}, \textcircled{l}, \textcircled{u}$) iff
$$l \leq |\{i \mid X_i \in v, 1 \leq i \leq k\}| \leq u$$

LOWER BOUND
UPPER BOUND
CERTAIN VALUES

 - **among**($[1, 5, 3, 2, 5, 4], \{1, 2, 3, 4\}, 3, 4$)
- Useful in sequencing problems, as we see next.

Sequencing Constraints

- Ensure a sequence of variables obey certain patterns.

THE AMONG FUNCTION IS VALID FOR
EVERY SUBSEQUENCE OF THE SET
→ WITH THE GIVEN LENGTH

Sequence Constraint

- Constrains the number of values taken from a given set in any subsequence of q variables. LENGTH OF THE SUBSEQUENCE
- **sequence**($l, u, q, [X_1, X_2, \dots, X_k], v$) iff
among($[X_i, X_{i+1}, \dots, X_{i+q-1}], v, l, u$) for $1 \leq i \leq k-q+1$
 - Known also as **amongseq** constraint.
 - **sequence**(1, 2, 3, [1,0,2,0,3], {0,1})
- Useful e.g. in:
 - rostering (e.g. every employee has 2 days off in any 7 day of period);
 - production line (e.g. at most 1 in 3 cars along the production line can have a sun-roof fitted).

Scheduling Constraints

- Help schedule tasks with respective release times, duration, and deadlines, using limited resources in a time interval D .

Disjunctive Resource Constraint

- Requires that tasks do not overlap in time.
 - Known also as **noOverlap** constraint.
- Given tasks t_1, \dots, t_k , each associated with a start time S_i and duration D_i :

disjunctive($[S_1, \dots, S_k], [D_1, \dots, D_k]$) iff for all $i < j$
 $(S_i + D_i \leq S_j) \vee (S_j + D_j \leq S_i)$

- Useful when a resource
can execute at most one
task at a time.

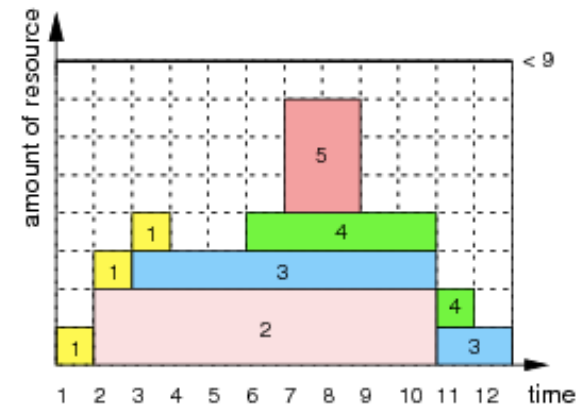


Cumulative Resource Constraint

- Constrains the usage of a shared resource.
- Given tasks t_1, \dots, t_k , each associated with a start time S_i , duration D_i , resource requirement R_i , and a resource with a capacity C :

cumulative $([S_1, \dots, S_k], [D_1, \dots, D_k], [R_1, \dots, R_k], C)$ iff
 $\sum_{i | S_i \leq u < S_i + D_i} R_i \leq C$ for all u in D

- Useful when a fixed-capacity resource can execute multiple tasks at a time.



Ordering Constraints

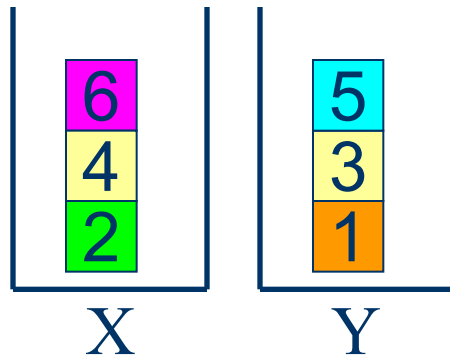
- Enforce an ordering between the variables or the values.

Lexicographic Ordering Constraint

- Requires a sequence of variables to be lexicographically less than or equal to another sequence of variables.
- $\text{lex}\leq([X_1, X_2, \dots, X_k], [Y_1, Y_2, \dots, Y_k])$ holds iff:
 - $X_1 \leq Y_1 \wedge$
 - $(X_1 = Y_1 \rightarrow X_2 \leq Y_2) \wedge$
 - $(X_1 = Y_1 \wedge X_2 = Y_2 \rightarrow X_3 \leq Y_3) \dots$
 - $(X_1 = Y_1 \wedge X_2 = Y_2 \dots \wedge X_{k-1} = Y_{k-1} \rightarrow X_k \leq Y_k)$
 - $\text{lex}\leq([1, 2, 4], [1, 3, 3])$
- Useful for breaking symmetry.
 - $\text{lex}\leq(X, \pi(X))$ for all π
 - $\text{lex}\leq(X, Y)$

Lexicographic Ordering Constraint

- Consider the assignment of items to bins, which can be modelled by a matrix of Boolean variables:

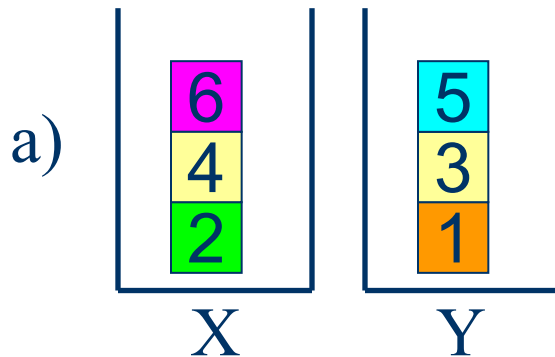


	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	0	1	0	1
Y	1	0	1	0	1	0

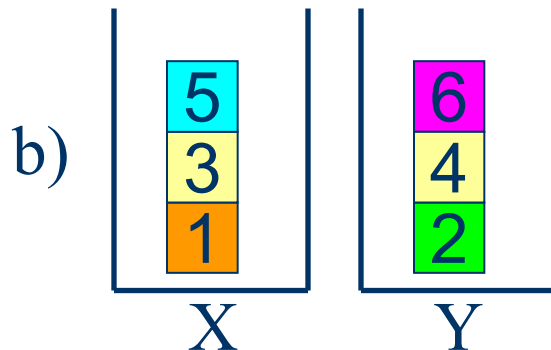
- What if the bins are symmetric?

Lexicographic Ordering Constraint

- Their item assignments can be permuted.



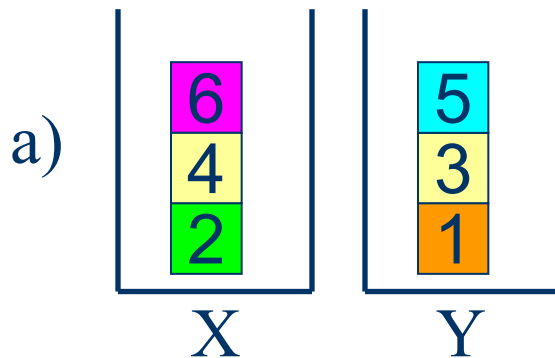
	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	0	1	0	1
Y	1	0	1	0	1	0



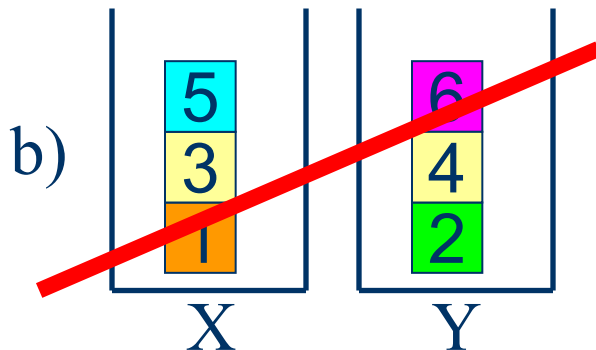
	i_1	i_2	i_3	i_4	i_5	i_6
X	1	0	1	0	1	0
Y	0	1	0	1	0	1

Lexicographic Ordering Constraint

- $\text{lex} \leq (X, Y)$ avoids the symmetric assignments.



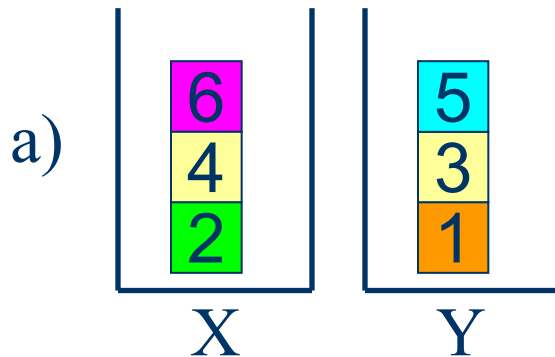
	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	0	1	0	1
Y	1	0	1	0	1	0



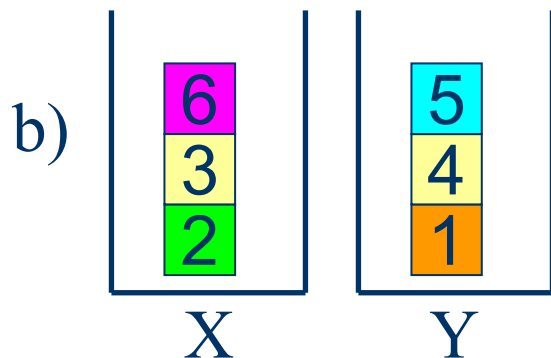
	i_1	i_2	i_3	i_4	i_5	i_6
X	1	0	1	0	1	0
Y	0	1	0	1	0	1

Lexicographic Ordering Constraint

- What if items 3 and 4 are symmetric too?



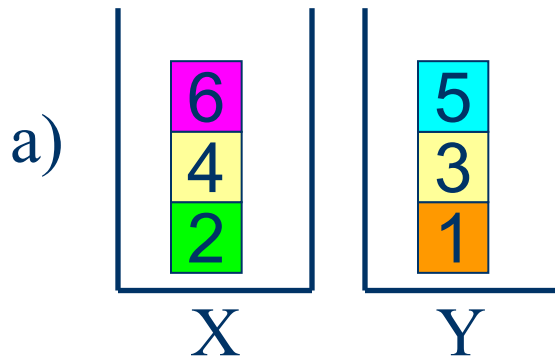
	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	0	1	0	1
Y	1	0	1	0	1	0



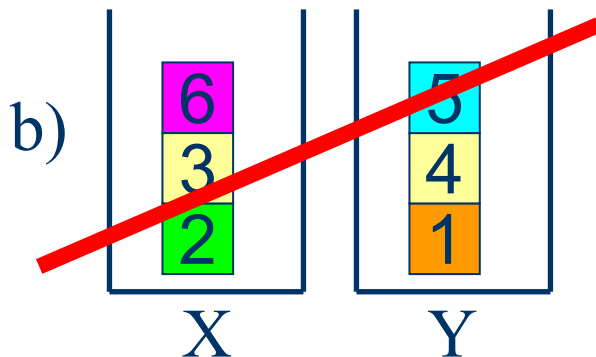
	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	1	0	0	1
Y	1	0	0	1	1	0

Lexicographic Ordering Constraint

- $\text{lex} \leq(i_3, i_4)$ avoids the symmetric assignments.



	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	0	1	0	1
Y	1	0	1	0	1	0



	i_1	i_2	i_3	i_4	i_5	i_6
X	0	1	1	0	0	1
Y	1	0	0	1	1	0

Specialized Propagation for Global Constraints

- How do we develop specialized propagation for global constraints?
- Two main approaches:
 - constraint decomposition;
 - dedicated propagation algorithm.

Constraint Decomposition

- A global constraint is decomposed into smaller and simpler constraints, each of which has a known propagation algorithm.
- Propagating each of the constraints gives a propagation algorithm for the original global constraint.
 - A very effective and efficient approach for some global constraints.

A Decomposition of Among

- **among**($[X_1, X_2, \dots, X_k], v, N$) iff
$$N = |\{i \mid X_i \in v, 1 \leq i \leq k\}|$$
 - Decomposition as a conjunction of logical constraints and a sum constraint:
$$B_i \text{ with } D(B_i) = \{0, 1\} \text{ for } 1 \leq i \leq k$$
$$C_i: B_i = 1 \Leftrightarrow X_i \in v \text{ for } 1 \leq i \leq k$$
$$C_{k+1}: \sum_i B_i = N$$
 - AC(C_i) for all i and BC(C_{k+1}) ensures GAC on **among**.

Constraint Decompositions

- May not always provide an effective propagation.
- Often GAC on the original constraint is stronger than (G)AC on the constraints in the decomposition.

A Decomposition of Alldifferent

- **alldifferent**($[X_1, X_2, \dots, X_k]$)
 - Decomposition as a conjunction of difference constraints C_{ij} : $X_i \neq X_j$ for $i < j \in \{1, \dots, k\}$
 - $AC(C_{ij})$ for all $i < j$ is weaker than GAC on **alldifferent**.
 - E.g., **alldifferent**($[X_1, X_2, X_3]$) with $D(X_1) = D(X_2) = D(X_3) = \{1, 2\}$.
 - **alldifferent** is not GAC but the decomposition does not prune anything.

A Decomposition of Disjunctive

- **disjunctive**([S_1, \dots, S_k], [D_1, \dots, D_k])
 - Decomposition as a conjunction of disjunctive constraints C_{ij} : $(S_i + D_i \leq S_j) \vee (S_j + D_j \leq S_i)$
for $i < j \in \{1, \dots, k\}$
 - Is $GAC(C_{ij})$ for all $i < j$ weaker than GAC on **disjunctive**?

Dedicated Propagation Algorithms

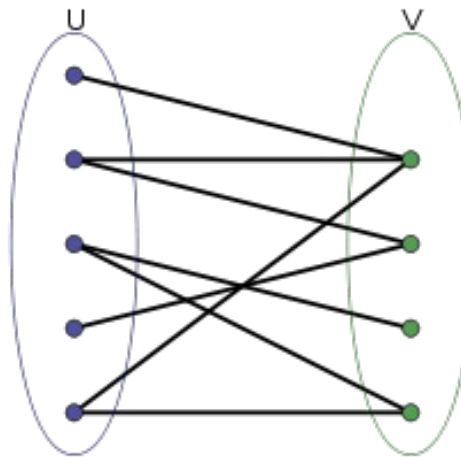
- Dedicated algorithms provide **effective** and **efficient** propagation.
- Often:
 - GAC is maintained in polynomial time;
 - many more inconsistent values are detected compared to the decompositions;
 - computation is done incrementally.

A Propagation Algorithm for alldifferent

- Jean-Charles Régin, “A Filtering Algorithm for Constraints of Difference in CSPs”, in the Proc. of AAAI’1994
 - Maintains GAC on **alldifferent**($[X_1, X_2, \dots, X_k]$) and runs in polynomial time.
 - Establishes a relation between the solutions of the constraint and the properties of a graph.
 - **Maximal matching** in a **bipartite graph**.
- A similar algorithm can be obtained with the use of flow theory.

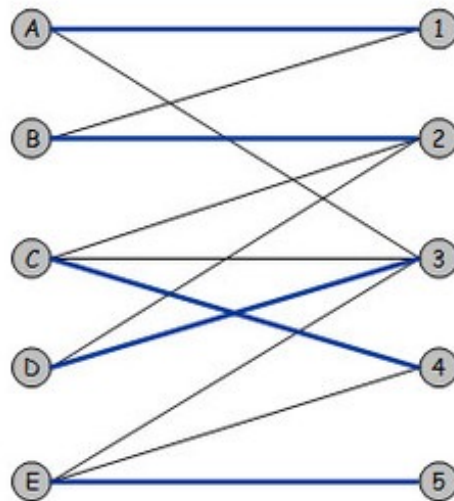
A Propagation Algorithm for all different

- A **bipartite graph** is a graph whose vertices are divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V .



A Propagation Algorithm for all different

- A **matching** in a graph is a subset of its edges such that no two edges have a node in common.
 - **Maximal matching** is the largest possible matching.
- In a bipartite graph, maximal matching covers one set of nodes.



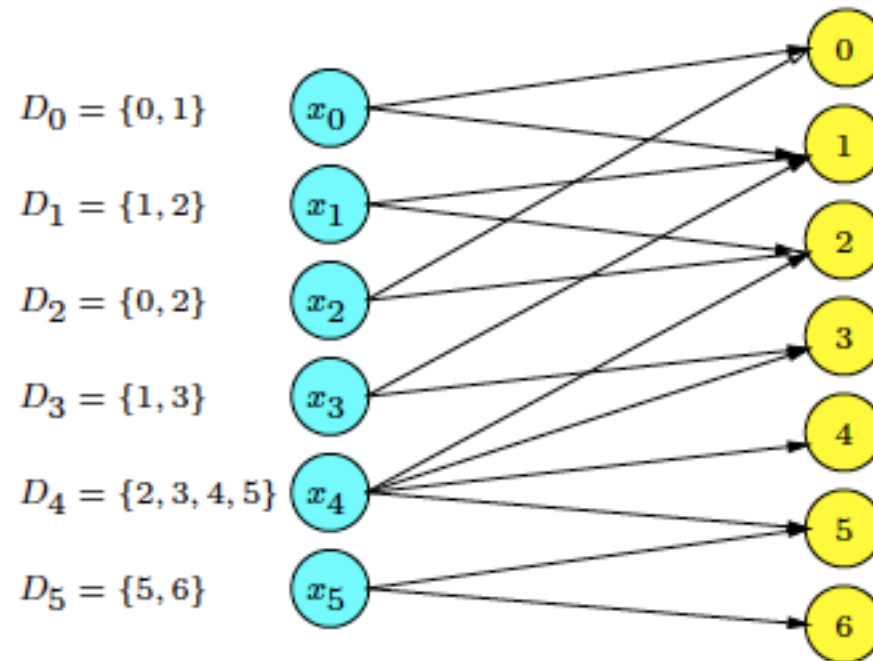
A Propagation Algorithm for all different

- Observation

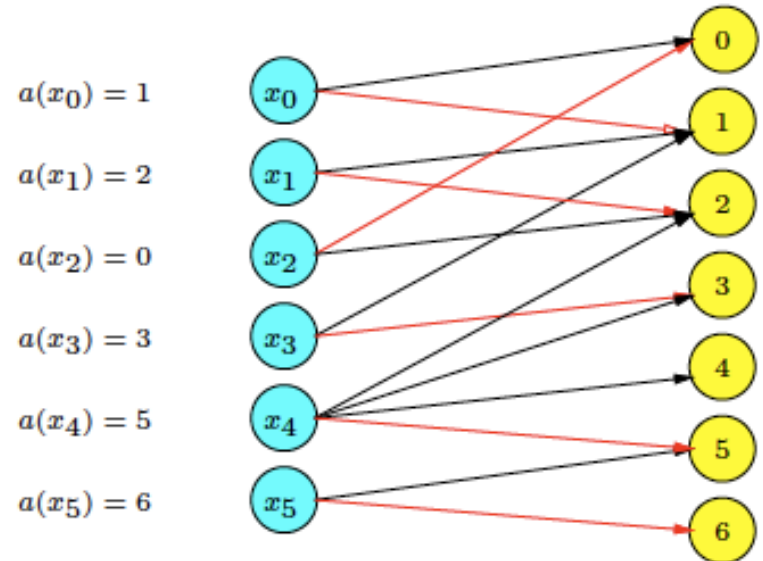
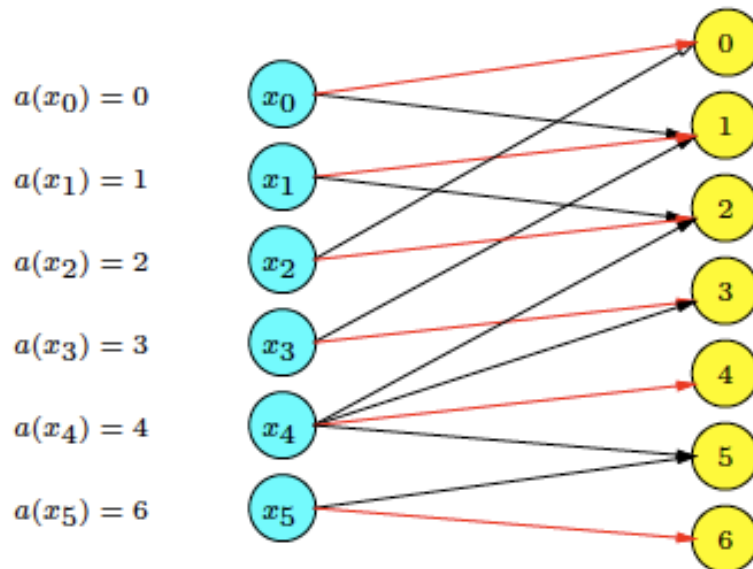
- Given a **bipartite graph** G constructed between the variables $[X_1, X_2, \dots, X_k]$ and their possible values (**variable-value graph**),
- an **assignment** of values to the variables is a solution iff it corresponds to a **maximal matching** in G .
 - A maximal matching covers all the variables.
- By computing **all maximal matchings**, we can find the consistent partial assignments.

Example

Variable-value graph



Some Maximal Matchings



All Maximal Matchings

- Inefficient to compute them naively.
- Theoretical results from matching theory to compute them efficiently.
 - One maximal matching can describe all maximal matchings!

Dedicated Propagation Algorithms

- GAC may as well be NP-hard!
 - E.g., gcc using variables for occurrences.
 - Algorithms which maintain weaker consistencies are of interest.
 - BC.
 - Between GAC and BC.
 - GAC on some variables, BC on others.
 - ...

Dedicated Propagation Algorithms

- What if it is difficult to:
 - decompose a constraint;
 - build an efficient and effective dedicated algorithm?
- Consult global constraints for generic purposes!
 - E.g., table constraint.
 - Many solvers have efficient GAC algorithms.
 - Need to keep the table size small.

Crossword Puzzle Generation

- Valid words are defined in a table of compatible letters (i.e. dictionary).
 - `table([X1,X2,X3], dictionary)`
 - `table([X1,X13,X16], dictionary)`
 - `table([X4,X5,X6,X7], dictionary)`
 - ...
- No simple way to decide acceptable words other than to put them in a table.

1	C	2	A	3	T		4	T	5	S	6	N	7	I		8	P	9	E	10	R	11	C	12	H
13	E	C	A				14	H	T	O	G				15	T	U	R	T	L	E				
16	S	H	I				17	B	A	I	N	U			18	O	R	R			19	O	R		
			20	L	A	I	C				21	A	22	B	E	R			23	F	W	D			
24	B	O	W	L				26	K	A	N	E			28	S	H	E	D	I					
30	S	W	A	L			31	C			32	R	A	S	33	P			34	O	W	E	N		
35	E	N	G				36	H	A	M	S	T	E	R	38	S			39	R	G				
			40	S	S	I	M							42	T	A	E	43	M						
44	S	F				46	P	A	R	A	K	E	E	T				50	U	S	A				
53	C	E				54	I	C			55	E	Y	E	S			56	S	K	I	N	S		
58	R	E	T	A		59	W				60	A	N	E	61	W		62	E	R	E	H			
63	A	D	S			64	H	A	H	N					66	O	K	R	A						
68	T	E				69	A	E	S					70	E	U	K	A	N	U	72	B	A		
74	C	R	A	T	E	S								76	L	A	E	R			77	Q	U	O	
78	H	S	A	E	L									79	S	E	N	T			80	A	T	L	