

**5.175 k\_alldifferent**

	DESCRIPTION	LINKS	GRAPH
<b>Origin</b>	[133]		
<b>Constraint</b>	<code>k_alldifferent(VARS)</code>		
<b>Synonyms</b>	<code>k_alldiff</code> , <code>k_alldistinct</code> , <code>some_different</code> .		
<b>Type</b>	<code>X</code> : <code>collection(x-dvar)</code>		
<b>Argument</b>	<code>VARS</code> : <code>collection(vars - X)</code>		
<b>Restrictions</b>	<code>required(X, x)</code> <code>required(VARS, vars)</code> $ X  > 0$ $ VARS  > 0$		
<b>Purpose</b>	For each collection of variables depicted by an item of VARS, enforce their corresponding variables to take distinct values.		
<b>Example</b>	$\left( \left\langle \text{vars} - \left\langle \begin{array}{c} x - 5, \\ x - 6, \\ x - 0, \\ x - 9, \\ x - 3 \end{array} \right\rangle, \right\rangle \right)$ $\text{vars} - \langle 5, 6, 1, 2 \rangle$		
	The <code>k_alldifferent</code> constraint holds since all the values 5, 6, 0, 9 and 3 are distinct and since all the values 5, 6, 1 and 2 are distinct as well.		
<b>Typical</b>	$ X  > 1$ $ VARS  > 1$		
<b>Symmetries</b>	<ul style="list-style-type: none"> <li>• Items of VARS are <a href="#">permutable</a>.</li> <li>• Items of VARS.vars are <a href="#">permutable</a>.</li> <li>• All occurrences of two distinct values of VARS.vars.x can be <a href="#">swapped</a>; all occurrences of a value of VARS.vars.x can be <a href="#">renamed</a> to any unused value.</li> </ul>		
<b>Usage</b>	Systems of <code>alldifferent</code> constraints sharing variables occurs frequently in practice. We give 4 typical problems that can be modelled by a combination of <code>alldifferent</code> constraints as well as one problem where a system of <code>alldifferent</code> constraints provides a necessary condition.		

- The *graph colouring* problem is to colour with a restricted number of colours the vertices of a given undirected graph in such a way that adjacent vertices are coloured with distinct colours. The problem can be modelled by a system of **alldifferent** constraints. All the next problems can be seen as graph colouring problems where the graphs have some specific structure.
- A *Latin square of order  $n$*  is an  $n \times n$  array in which  $n$  distinct numbers in  $[1, n]$  are arranged so that each number occurs once in each row and column. The problem is to complete a partially filled Latin square. Part (A) of Figure 5.354 gives a partially filled Latin square, while part (B) provides a possible completion.

1			
			3
3			
			1

(A)

1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

(B)

Figure 5.354: A partially filled Latin square and a possible completion

- A *Sudoku* is a Latin square of order  $9 \times 9$  such that the numbers in each major  $3 \times 3$  block are distinct. As for the Latin square problem, the problem is to complete a partially filled board. Part (A) of Figure 5.355 gives a partially filled Sudoku board, while part (B) provides a possible completion. A constraint programming approach for solving Sudoku puzzles is depicted in [350]. It shows how to generate redundant constraints as well as shaving [247] in order to find a solution without guessing.

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

(A)

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

(B)

Figure 5.355: A partially Sudoku square and a possible completion

- A *task assignment* problem consists to assign a given set of non-preemptive tasks, which are fixed in time (i.e., the origin, duration and end of each task are fixed), to a set of resources so that, tasks that are assigned to the same resource do not overlap in time. Each task can be assigned to a predefined set of resources. Problems like *aircraft stand allocation* [124], [349] or *air traffic flow management* [19] correspond to an example of a real-life task **assignment** problem. *Assignment of service professionals* [12] is yet another industrial example where professionals have to be



The `tree_precedence` constraint specifies that its associated digraph  $\mathcal{G}$  should be a forest that fulfils the precedence constraints. Formally a ground instance of a `tree_precedence(NTREE, VERTICES)` constraint is satisfied if and only if the following conditions hold:

1.  $\forall i \in [1, n] : \text{VERTICES}[i].\text{index} = i,$
2. Its associated digraph  $\mathcal{G}$  consists of `NTREE` connected components,
3. Each connected component of  $\mathcal{G}$  does not contain any circuit involving more than one vertex,
4. For every vertex `VERTICES`[ $i$ ] such that  $j \in \text{VERTICES}[i].\text{preds}$  there must be an elementary path in  $\mathcal{G}$  from `VERTICES`[ $j$ ] to `VERTICES`[ $i$ ].

We can build the following system of `alldifferent` constraints that corresponds to a necessary condition for the `tree_precedence` constraint: To each vertex  $v$  of  $\mathcal{G}$ , which both has no predecessors and cannot be the root of a tree, we generate an `alldifferent` constraint involving the father variables of those descendants of  $v$  in  $\mathcal{G}$  that cannot be the root of a tree.

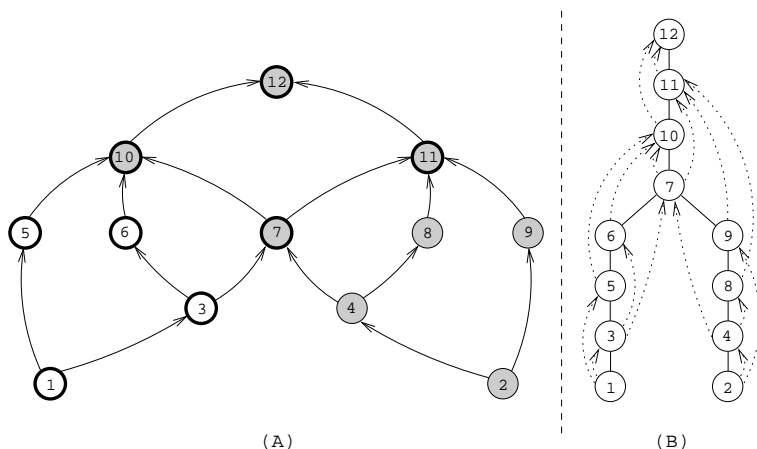


Figure 5.357: A set of precedences and a corresponding feasible tree

For the set of precedences depicted by part (A) of Figure 5.357<sup>9</sup>, where we assume that `VERTICES`[12] is the only vertex that can be a root and where  $F_i$  denotes the father variable associated with `VERTICES`[ $i$ ], we get the following system of `alldifferent` constraints:

- `alldifferent`( $\langle F_1, F_3, F_5, F_6, F_7, F_{10}, F_{11} \rangle$ ),
- `alldifferent`( $\langle F_2, F_4, F_7, F_8, F_9, F_{10}, F_{11} \rangle$ ).

The variables of these two `alldifferent` constraints respectively correspond to the descendants of the two source vertices (i.e.,  $F_1$  and  $F_2$ ) of the precedence graph depicted by part (A) of Figure 5.357. On part (A) of Figure 5.357 the descendants of  $F_1$  and  $F_2$  are respectively depicted with a thick line and a grey circle. Their intersection,  $\{F_7, F_{10}, F_{11}, F_{12}\}$ , from which we remove

<sup>9</sup>The number in a vertex gives the value of the `index` attribute of the corresponding item.

$F_{12}$  belong to the two `alldifferent` constraints. In fact,  $F_{12}$  is not mentioned in the two `alldifferent` constraints since its corresponding vertex is the root of a tree. Part (B) of Figure 5.357 gives a possible tree satisfying all the precedences constraints expressed by part (A), where precedences are depicted with a dotted line. It corresponds to the following ground solution:

```
tree_precedence((
  index - 1   father - 3   preds - {},
  index - 2   father - 4   preds - {},
  index - 3   father - 5   preds - {1},
  index - 4   father - 8   preds - {2},
  index - 5   father - 6   preds - {1},
  index - 6   father - 7   preds - {3},
  index - 7   father - 10  preds - {3, 4},
  index - 8   father - 9   preds - {4},
  index - 9   father - 7   preds - {2},
  index - 10  father - 11  preds - {5, 6, 7},
  index - 11  father - 12  preds - {7, 8, 9},
  index - 12  father - 12  preds - {10, 11} ))
```

**Remark**

It was shown in [134] that, finding out whether a system of two `alldifferent` constraints sharing some variables has a solution or not is NP-hard. This was achieved by reduction from `set packing`.

A slight variation in the way of describing the arguments of the `k_alldifferent` constraint appears in [324] under the name of `some_different`: the set of disequalities is described by a set of pairs of variables, where each pair corresponds to a disequality constraint between two given variables.

Within the context of `linear programming`, a relaxation of the `k_alldifferent` constraint is provided in [7]. The special case where  $k = 2$  is discussed in [8].

**Algorithm**

Even if there is no filtering algorithm for the `k_alldifferent` constraint, one can enforce redundant constraints for the following patterns:

- Within the context of graph colouring, one can state an `nvalue` constraint for every cycle of odd length of the graph to colour enforcing that the corresponding variables have to be assigned to at least three distinct values.
- Within the context of Latin squares, one can state a `colored_matrix` constraint enforcing that each value is used exactly once in each row and column.
- Within the context of two `alldifferent` constraints `alldifferent`(( $U_1, \dots, U_n, V_1, \dots, V_m$ )) and `alldifferent`(( $U_1, \dots, U_n, W_1, \dots, W_m$ )) where the domain of all variables  $U_1, \dots, U_n, V_1, \dots, V_m, W_1, \dots, W_m$  is included in the interval  $[1, n + m]$ , one can state a `same_and_global_cardinality` constraint stating that the variables  $V_1, \dots, V_m$  should correspond to a permutation of the variables  $W_1, \dots, W_m$  and that the variables  $V_1, \dots, V_m$  should be assigned to distinct values.
- In the general case of two `alldifferent` constraints `alldifferent`(( $U_1, \dots, U_n, V_1, \dots, V_m$ )) and `alldifferent`(( $U_1, \dots, U_n, W_1, \dots, W_o$ )), one can state an `nvalue` constraint involving the variables  $V_1, \dots, V_m$  and  $W_1, \dots, W_o$  enforcing that these variables should not use more than  $s - n$  distinct values, where  $s$  denotes the cardinality of the union of the domains of the variables  $U_1, \dots, U_n, V_1, \dots, V_m, W_1, \dots, W_o$ .

Several propagation rules for the `k_alldifferent` constraint are also described in [226].

**Reformulation**

Given two `alldifferent` constraints that share some variables, a reformulation preserving `bound-consistency` was introduced in [67]. This reformulation is based on an extension of Hall's theorem that is presented in the same paper.

**See also**

**common keyword:** `colored_matrix` (*system of constraints*).

**generalisation:** `diffn`, `geost` (*tasks for which the start attribute is not fixed*).

**part of system of constraints:** `alldifferent`.

**related:** `nvalue` (*implied by two overlapping `alldifferent`*), `same_and_global_cardinality` (*implied by two overlapping `alldifferent` and restriction on values*).

**Keywords**

**application area:** air traffic management, assignment.

**characteristic of a constraint:** all different, disequality.

**combinatorial object:** permutation, Latin square.

**complexity:** set packing.

**constraint type:** system of constraints, overlapping `alldifferent`, value constraint, decomposition.

**filtering:** bound-consistency, duplicated variables.

**problems:** graph colouring.

**puzzles:** Sudoku.

	For all items of VARS:
<b>Arc input(s)</b>	<code>VARS.vars</code>
<b>Arc generator</b>	<code>CLIQUE</code> $\mapsto$ <code>collection(x1, x2)</code>
<b>Arc arity</b>	2
<b>Arc constraint(s)</b>	<code>x1.x = x2.x</code>
<b>Graph property(ies)</b>	<u><code>MAX_NSCC</code> <math>\leq</math> 1</u>
<b>Graph model</b>	For each collection of variables depicted by an item of VARS we generate a <i>clique</i> with an <i>equality</i> constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one.

20050618

1115