5.205 k_alldifferent

DESCRIPTION LINKS GRAPH

Origin [151]

Constraint k_alldifferent(VARS)

Synonyms k_alldiff, k_alldistinct, some_different.

Type X : collection(x−dvar)

Argument VARS : collection(vars − X)

Restrictions
$|\mathtt{X}| \geq 1$
required(X, x)
required(VARS, vars)
$|\mathtt{VARS}| \geq 1$

Purpose For each collection of variables depicted by an item of VARS, enforce their corresponding variables to take distinct values. Usually some variables occur in several collections.

Example $(\langle \mathtt{vars} - \langle 5, 6, 0, 9, 3\rangle, \mathtt{vars} - \langle 5, 6, 1, 2\rangle\rangle)$

The k_alldifferent 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.

Typical
$|\mathtt{X}| > 1$
$|\mathtt{VARS}| > 1$

Symmetries

- Items of VARS are permutable.
- Items of VARS.vars are permutable.
- All occurrences of two distinct values of VARS.vars.x can be swapped; all occurrences of a value of VARS.vars.x can be renamed to any unused value.

Arg. properties Contractible wrt. VARS.

Usage Systems of alldifferent constraints sharing variables occurs frequently in practice. We give 4 typical problems that can be modelled by a combination of alldifferent constraints as well as one problem where a system of alldifferent 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 been 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 5.474 gives a partially filled Latin square, while part (B) provides a possible completion.

(A)

1			
			3
3			
			1

(B)

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

Figure 5.474: (A) A partially filled Latin square and (B) a possible completion

- A *Sudoku* is a Latin square of order 9×9 such that the numbers in each major 3×3 block are distinct. As for the Latin square problem, the problem is to complete a partially filled board. Part (A) of Figure 5.475 gives a partially filled Sudoku board, while part (B) provides a possible completion. A constraint programming approach for solving Sudoku puzzles is depicted in [384]. It shows how to generate redundant constraints as well as shaving [276] in order to find a solution without guessing.

(A)

	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	

(B)

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

Figure 5.475: (A) A partially filled Sudoku square and (B) its unique 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* [140], [383] or *air traffic flow management* [20] correspond to an example of a real-life task assignment problem. *Assignment of service professionals* [13] is yet another industrial example where professionals have to be assigned positions in such a way that positions assigned to a given professional do not overlap in time.
  Part (A) of Figure 5.476 gives an example of task assignment problem. For each task we indicate the set of resources where it can potentially be assigned (i.e., the domain

of its assignment variable). For instance, task t_1 can be assigned to resources 1 or 2. Part (B) of Figure 5.476 gives the corresponding interval graph: We have one vertex for each task and an edge between two tasks that overlap in time. We have a system of `alldifferent` constraints corresponding to the maximum cliques of the interval graph (i.e., $\{t_1, t_5, t_8\}$, $\{t_2, t_5, t_8\}$, $\{t_2, t_6\}$, $\{t_3, t_6, t_9\}$, $\{t_3, t_7, t_9\}$, $\{t_4, t_7, t_9\}$). Finally, part (C) of Figure 5.476 provides a possible solution to the task assignment problem where tasks t_1, t_2, t_9 are assigned to resource 1, tasks t_3, t_4, t_8 are assigned to resource 2, and tasks t_5, t_6, t_7 are assigned to resource 3.

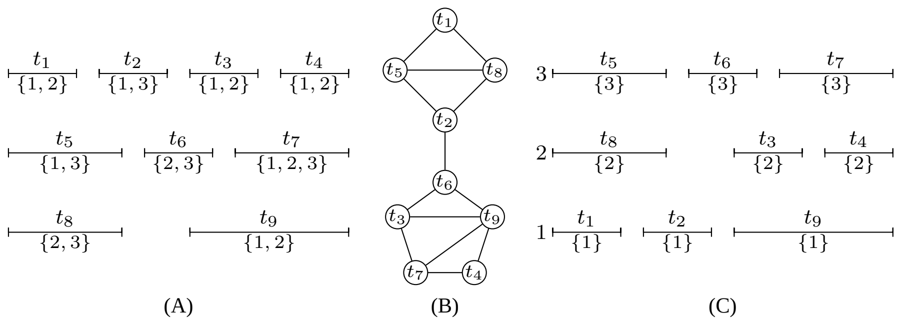


Figure 5.476: (A) Tasks $t_1, t_2, \ldots, t_9$ with their potential assignment 1, 2 or 3 (B) Interval graph where to each task of corresponds a vertex, and to each pair of overlapping tasks corresponds an edge (C) A valid assignment where tasks assigned to a same machine do not overlap

- The *tree partitioning with precedences* problem is to compute a vertex-partitioning of a given digraph $\mathcal{G}$ in disjoint trees (i.e., a forest), so that a given set of precedences holds. The problem can be modelled with a `tree_precedence`(`NTREE`, `VERTICES`) constraint, where `NTREE` is a domain variable specifying the numbers of trees in the forest and `VERTICES` is a collection of the digraph's n vertices. Each item $v \in$ `VERTICES` has the following attributes, which complete the description of the digraph:

  - `index` is an integer in $[1, n]$ that can be interpreted as the *label* of v.
  - `father` is a domain variable whose domain consists of elements (vertex label) of $[1, n]$. It can be interpreted as the *unique successor* of v.
  - `preds` is a possibly empty set of integers, its elements (vertex label) being in $[1, n]$. It can be interpreted as the *mandatory ancestors* of v.

  We model the `tree_precedence` constraint by the digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in which the vertices represent the elements of `VERTICES` and the arcs represent the successors relations between them. Formally, $\mathcal{G}$ is defined as follows:

  - To the i^{th} vertex $(1 \leq i \leq n)$, `VERTICES`$[i]$, of the `VERTICES` collection corresponds a vertex of $\mathcal{V}$ denoted by v_i.
  - For every pair of vertices (`VERTICES`$[i]$,`VERTICES`$[j]$), where i and j are not necessarily distinct, there is an arc from v_i to v_j in $\mathcal{E}$.

  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. $\forall i \in [1, n]$: VERTICES$[i]$.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 $j \in$ VERTICES$[i]$.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.

For the set of precedences depicted by part (A) of Figure 5.477[12], 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:

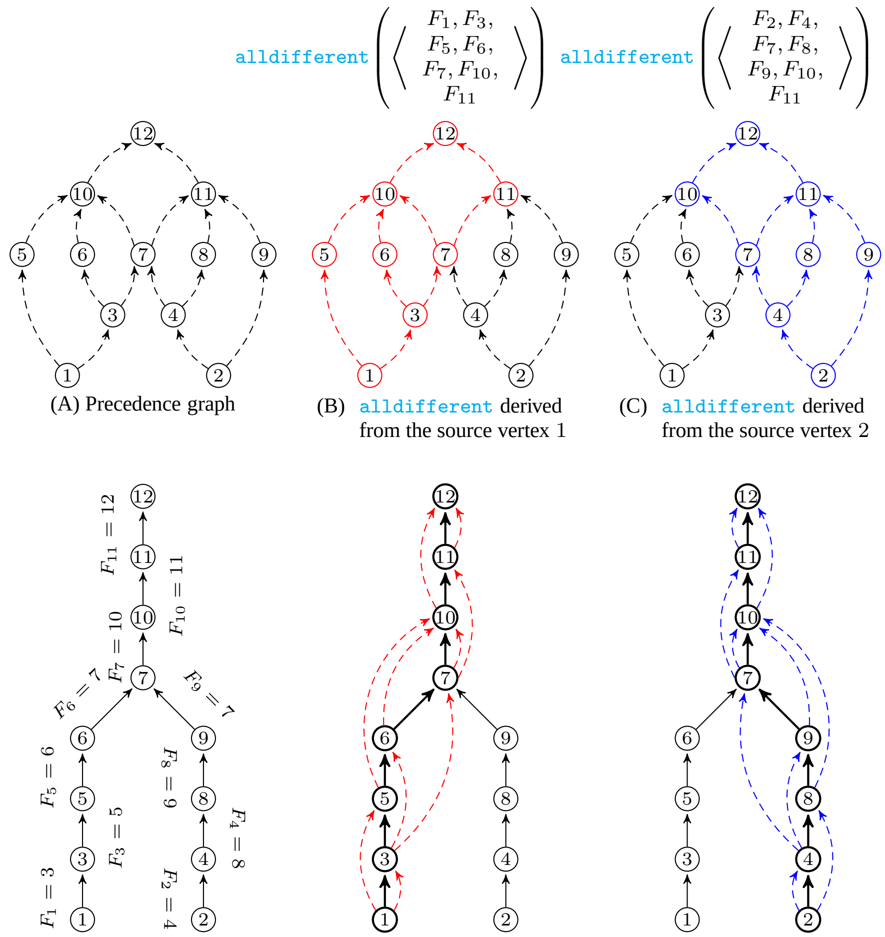
- alldifferent$(\langle F_1, F_3, F_5, F_6, F_7, F_{10}, F_{11}\rangle)$,
- alldifferent$(\langle F_2, F_4, F_7, F_8, F_9, F_{10}, F_{11}\rangle)$.

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 parts (B) and (C) of Figure 5.477. On part (B) and (C) of Figure 5.477 the descendants of F_1 and F_2 are respectively depicted in red and blue. Their intersection, $\{F_7, F_{10}, F_{11}, F_{12}\}$, from which we remove 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 (D) of Figure 5.477 gives a possible tree satisfying all the precedences constraints expressed by part (A). It corresponds to the following ground solution:

tree_precedence($\langle$	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 $-\{10,11\}$ $\rangle$)

Parts (E) and (F) of Figure 5.477 illustrate how the precedence constraints are satisfied by the solution depicted by part (D): each precedence, represented by a dashed arc, links two vertices that belong to a same path of the tree that is directed toward the root of the tree.

[12]The number in a vertex gives the value of the index attribute of the corresponding item.



(A) Precedence graph

(B) alldifferent derived from the source vertex 1

(C) alldifferent derived from the source vertex 2

(D) A tree satisfying the precedence graph

(E) Checking the red precedences on the path from the source 1 to the sink 12 of the tree depicted in (D)

(F) Checking the blue precedences on the path from the source 2 to the sink 12 of the tree depicted in (D)

Figure 5.477: (A) A set of precedences and (D) a corresponding feasible tree where F_i stands for the father of the i^{th} vertex; (B) the `alldifferent` constraint associated with the source vertex 1 and (E) the satisfied precedences in red along the paths of the tree of (D); (C) the `alldifferent` constraint associated with the source vertex 2 and (F) the satisfied precedences in blue along the paths of the tree of (D);

Remark It was shown in [152] 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 [357] 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 [8]. The special case where $k = 2$ is discussed in [9].

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`$(\langle U_1, \ldots, U_n, V_1, \ldots, V_m\rangle)$ and `alldifferent`$(\langle U_1, \ldots, U_n, W_1, \ldots, W_m\rangle)$ where the domain of all variables $U_1, \ldots, U_n$, $V_1, \ldots, V_m$, $W_1, \ldots, W_m$ is included in the interval $[1, n + m]$, one can state a `same_and_global_cardinality` constraint stating that the variables $V_1, \ldots, V_m$ should correspond to a permutation of the variables $W_1, \ldots, W_m$ and that the variables $V_1, \ldots, V_m$ should be assigned to distinct values.
- In the general case of two `alldifferent` constraints `alldifferent`$(\langle U_1, \ldots, U_n, V_1, \ldots, V_m\rangle)$ and `alldifferent`$(\langle U_1, \ldots, U_n, W_1, \ldots, W_o\rangle)$, one can state an `nvalue` constraint involving the variables $V_1, \ldots, V_m$ and $W_1, \ldots, W_o$ 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 $U_1, \ldots, U_n$, $V_1, \ldots, V_m$, $W_1, \ldots, W_o$.

Several propagation rules for the `k_alldifferent` constraint are also described in [253].

Reformulation Given two `alldifferent` constraints that share some variables, a reformulation preserving bound-consistency was introduced in [73]. 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:

Arc input(s)	VARS.vars
Arc generator	$CLIQUE \mapsto$ collection(x1, x2)
Arc arity	2
Arc constraint(s)	x1.x $=$ x2.x
Graph property(ies)	**MAX_NSCC** ≤ 1

Graph model For each collection of variables depicted by an item of VARS we generate a *clique* with an *equality* 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.