

5.5 alldifferent

	DESCRIPTION	LINKS	GRAPH	AUTOMATON
Origin	[229]			
Constraint	<code>alldifferent(VARIABLES)</code>			
Synonyms	<code>alldiff</code> , <code>alldistinct</code> , <code>distinct</code> , <code>bound_alldifferent</code> , <code>bound_alldiff</code> , <code>bound_distinct</code> , <code>rel</code> .			
Argument	VARIABLES : <code>collection(var-dvar)</code>			
Restriction	<code>required(VARIABLES, var)</code>			
Purpose	Enforce all variables of the collection VARIABLES to take distinct values.			
Example	<code>((5, 1, 9, 3))</code>			
	The <code>alldifferent</code> constraint holds since all the values 5, 1, 9 and 3 are distinct.			
Typical	$ \text{VARIABLES} > 2$			
Symmetries	<ul style="list-style-type: none"> Items of VARIABLES are permutable. Two distinct values of VARIABLES.var can be swapped; a value of VARIABLES.var can be renamed to any unused value. 			
Usage	<p>The <code>alldifferent</code> constraint occurs in most practical problems directly or indirectly. A classical example is the n-queen chess puzzle problem: Place n queens on a n by n chessboard in such a way that no queen attacks another. Two queens attack each other if they are located on the same column, on the same row or on the same diagonal. This can be modelled as the conjunction of three <code>alldifferent</code> constraints. We associate to the i^{th} column of the chessboard a domain variable X_i that gives the row number where the corresponding queen is located. The three <code>alldifferent</code> constraints are:</p> <ul style="list-style-type: none"> <code>alldifferent($X_1, X_2 + 1, \dots, X_n + n - 1$)</code> for the upper-left to lower-right diagonals, <code>alldifferent(X_1, X_2, \dots, X_n)</code> for the rows, <code>alldifferent($X_1 + n - 1, X_2 + n - 2, \dots, X_n$)</code> for the lower right to upper-left diagonals. <p>They are respectively depicted by parts (A), (C) and (D) of Figure 5.4.</p> <p>A second example taken from [13] when the bipartite graph associated with the <code>alldifferent</code> constraint is convex is a <i>ski assignment problem</i>: “a set of skiers have each specified the smallest and largest skis they will accept from a given set of skis”. The task is to find a ski for each skier.</p>			

Examples such as [Costas arrays](#) or [Golomb rulers](#) involve one or several `alldifferent` constraints on *differences* of variables.

Quite often, the `alldifferent` constraint is also used in conjunction with several `element` constraints, specially in the context of assignment problems [pages 372–374][190].

Other examples involving several `alldifferent` constraints sharing some variables can be found in the **Usage** slot of the `k_alldifferent` constraint.

Remark

Even if the `alldifferent` constraint had not this form, it was specified in ALICE [228, 229] by asking for an injective correspondence between variables and values: $x \neq y \Rightarrow f(x) \neq f(y)$. From an algorithmic point of view, the algorithm for computing the cardinality of the maximum matching of a bipartite graph was not used for checking the feasibility of the `alldifferent` constraint, even if the algorithm was already known in 1976. This stands from the fact that the goal of ALICE was to show that a general system could be as efficient as dedicated algorithms. For this reason the concluding part of [228] explicitly mentions the fact that specialized algorithms should be discarded. On the one hand, many people, specially from the OR community, have complained about such radical statement [330, page 28]. On the other hand, the motivation of such statement stands from the fact that a truly intelligent system should not rely on black box algorithms, but should rather be able to reconstruct them from some kind of first principle. How to achieve this is still an open question.

Some solvers use in a pre-processing phase before stating all constraints, an algorithm for automatically extracting large cliques [79] from a set of binary disequalities in order to replace them by `alldifferent` constraints.

W.-J. van Hoeve provides a survey about the `alldifferent` constraint in [382].

For possible relaxation of the `alldifferent` constraints see the `alldifferent_except_0`, the `k_alldifferent` (i.e., `some_different`), the `soft_alldifferent_ctr`, the `soft_alldifferent_var` and the `weighted_partial_alldiff` constraints.

Within the context of [linear programming](#), relaxations of the `alldifferent` constraint are described in [401] and in [190, pages 362–367].

Within the context of *constraint-centered search heuristics*, G. Pesant and A. Zanarini [407] have proposed several estimators for evaluating the number of solutions of an `alldifferent` constraint (since counting the total number of maximum matchings of the corresponding variable-value graph is $\#P$ -complete [375]). Faster, but less accurate estimators, based on upper bounds of the number of solutions were proposed three years later by the same authors [408].

Given n variables taking their values within the interval $[1, n]$, the total number of solutions of the corresponding `alldifferent` constraint corresponds to the sequence [A000142](#) of the On-Line Encyclopedia of Integer Sequences [357].

Algorithm

The first complete filtering algorithm was independently found by M.-C. Costa [111] and J.-C. Régin [307]. This algorithm is based on a corollary of C. Berge that characterises the edges of a graph that belong to a maximum matching but not to all [51, page 120].¹ A short time after, assuming that all variables have no holes in their domain, M. Leconte came up with a filtering algorithm [231] based on edge finding. A first [bound-consistency](#) algorithm

¹A similar result is in fact given in [278].

was proposed by Bleuzen-Guernalec *et al.* [71]. Later on, two different approaches were used to design [bound-consistency](#) algorithms. Both approaches model the constraint as a bipartite graph. The first identifies [Hall intervals](#) in this graph [291, 237] and the second applies the same algorithm that is used to compute [arc-consistency](#), but achieves a speedup by exploiting the simpler structure [172] of the graph [252]. Ian P. Gent *et al.* discuss in [167] implementation issues behind the complete filtering algorithm and in particular the computation of the strongly connected components of the residual graph (i.e., a graph constructed from a maximum variable-value matching and from the possible values of the variables of the `alldifferent` constraint), which appears to be the main bottleneck in practice.

From a worst case complexity point of view, assuming that n is the number of variables and m the sum of the domains sizes, we have the following complexity results:

- Complete filtering is achieved in $O(m\sqrt{n})$ by Régin’s algorithm [307].
- Range consistency is done in $O(n^2)$ by Leconte’s algorithm [231].
- [Bound-consistency](#) is performed in $O(n \log n)$ in [291, 252, 237]. If sort can be achieved in linear time, typically when the `alldifferent` constraint encodes a permutation,² the worst case complexity of the algorithms described in [252, 237] goes down to $O(n)$.

Within the context of *explanations* [202], the explanation of the filtering algorithm that achieves [arc-consistency](#) for the `alldifferent` constraint is described in [326, pages 60–61]. Given the residual graph (i.e., a graph constructed from a maximum variable-value matching and from the possible values of the variables of the `alldifferent` constraint), the removal of an arc starting from a vertex belonging to a strongly connected component \mathcal{C}_1 to a distinct strongly connected component \mathcal{C}_2 is explained by all missing arcs starting from a descendant component of \mathcal{C}_2 and ending in an ancestor component of \mathcal{C}_1 (i.e., since the addition of any of these missing arcs would merge the strongly connected components \mathcal{C}_1 and \mathcal{C}_2). Let us illustrate this on a concrete example. For this purpose assume we have the following variables and the values that can potentially be assigned to each of them, $A \in \{1, 2\}$, $B \in \{1, 2\}$, $C \in \{2, 3, 4, 6\}$, $D \in \{3, 4\}$, $E \in \{5, 6\}$, $F \in \{5, 6\}$, $G \in \{6, 7, 8\}$, $H \in \{6, 7, 8\}$. Figure 5.5 represents the residual graph associated with the maximum matching corresponding to the assignment $A = 1$, $B = 2$, $C = 3$, $D = 4$, $E = 5$, $F = 6$, $G = 7$, $H = 8$. It has four strongly connected components containing respectively vertices $\{A, B, 1, 2\}$, $\{C, D, 3, 4\}$, $\{E, F, 5, 6\}$ and $\{G, H, 7, 8\}$. Arcs that are between strongly connected components correspond to values that can be removed:

- The removal of value 2 from variable C is explained by the absence of the arcs corresponding to the assignments $A = 3$, $A = 4$, $B = 3$ and $B = 4$ (since adding any of these missing arcs would merge the blue and the pink strongly connected components containing the vertices corresponding to value 2 and variable C).
- The removal of value 6 from variable C is explained by the absence of the arcs corresponding to the assignments $E = 3$, $E = 4$, $F = 3$ and $F = 4$. Again adding the corresponding arcs would merge the two strongly connected components containing the vertices corresponding to value 6 and variable C .

²In this context the total number of values that can be assigned to the variables of the `alldifferent` constraint is equal to the number of variables. Under this assumption sorting the variables on their minimum or maximum values can be achieved in linear time.

- The removal of value 6 from variable G is explained by the absence of the arcs corresponding to the assignments $E = 7, E = 8, F = 7$ and $F = 8$.
- The removal of value 6 from variable H is explained by the absence of the arcs corresponding to the assignments $E = 7, E = 8, F = 7$ and $F = 8$.

After applying [bound-consistency](#) the following property holds for all variables of an `alldifferent` constraint. Given a [Hall interval](#) $[l, u]$, any variable V whose range $[\underline{V}, \overline{V}]$ intersects $[l, u]$ without being included in $[l, u]$ has its minimum value \underline{V} (respectively maximum value \overline{V}) that is located before (respectively after) the [Hall interval](#) (i.e., $\underline{V} < l \leq u < \overline{V}$).

The `alldifferent` constraint is [entailed](#) if and only if there is no value v that can be assigned two distinct variables of the `VARIABLES` collection (i.e., the intersection of the two sets of potential values of any pair of variables is empty).

Reformulation

The `alldifferent` constraint can be reformulated into a set of [disequalities](#) constraints. This model neither preserves [bound-consistency](#) nor [arc-consistency](#):

- On the one hand a model, involving linear constraints, preserving [bound-consistency](#) was introduced in [65]. For each potential interval $[l, u]$ of consecutive values this model uses $|\text{VARIABLES}|$ 0-1 variables $B_{1,l,u}, B_{2,l,u}, \dots, B_{|\text{VARIABLES}|,l,u}$ for modelling the fact that each variable of the collection `VARIABLES` is assigned a value within interval $[l, u]$ (i.e., $\forall i \in [1, |\text{VARIABLES}|] : B_{i,l,u} \Leftrightarrow \text{VARIABLES}[i].\text{var} \in [l, u]$),³ and an inequality constraint for enforcing the condition that the sum of the corresponding 0-1 variables is less than or equal to the size $u - l + 1$ of the corresponding interval (i.e. $B_{1,l,u} + B_{2,l,u} + \dots + B_{|\text{VARIABLES}|,l,u} \leq u - l + 1$).
- On the other hand, it was shown in [68] that there is no polynomial sized decomposition that preserves [arc-consistency](#).

Systems

`alldifferent` in **Choco**, `distinct` in **Gecode**, `rel` in **Gecode**, `alldifferent` in **JaCoP**, `alldiff` in **JaCoP**, `alldistinct` in **JaCoP**, `all_different` in **SICStus**, `all_distinct` in **SICStus**.

Used in

[alldifferent_consecutive_values](#), [circuit_cluster](#),
[correspondence](#), [cumulative_convex](#), [size_max_seq_alldifferent](#),
[size_max_starting_seq_alldifferent](#), [sort_permutation](#).

See also

common keyword: [circuit](#), [circuit_cluster](#), [cycle](#),
[derangement \(permutation\)](#), [golomb \(all different\)](#), [size_max_seq_alldifferent](#),
[size_max_starting_seq_alldifferent \(all different, disequality\)](#),
[symmetric_alldifferent \(permutation\)](#).

cost variant: [minimum_weight_alldifferent](#), [weighted_partial_alldiff](#).

generalisation: [all_min_dist](#) (variable replaced by line segment, all of the same size), [alldifferent_between_sets](#) (variable replaced by set variable), [alldifferent_cst](#) (variable replaced by variable + constant), [alldifferent_interval](#) (variable replaced by variable/constant), [alldifferent_modulo](#) (variable replaced by variable mod constant), [alldifferent_partition](#) (variable replaced by variable \in partition),

³How to encode the reified constraint $B_{i,l,u} \Leftrightarrow \text{VARIABLES}[i].\text{var} \in [l, u]$ with linear constraints is described in the **Reformulation slot** of the [in_interval_reified](#) constraint.

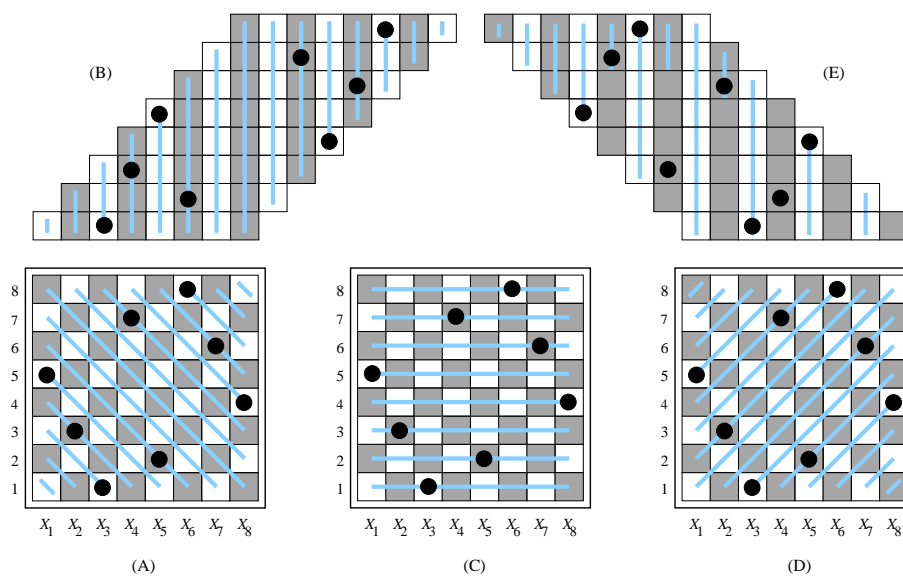


Figure 5.4: Upper-left to lower-right diagonals (A-B), rows (C) and lower-right to upper-left diagonals (D-E)

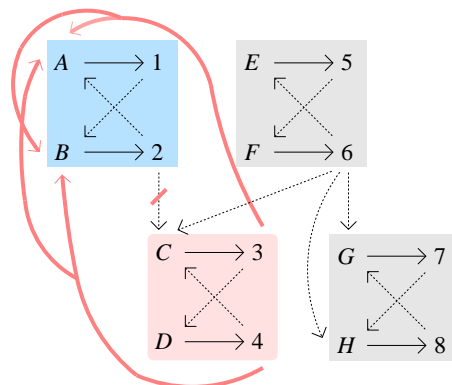


Figure 5.5: Strongly connected components of the residual graph illustrating the explanation of the removal of a value for the constraint $\text{alldifferent}(\langle A, B, C, D, E, F, G, H \rangle)$, $A \in \{1, 2\}$, $B \in \{1, 2\}$, $C \in \{2, 3, 4, 6\}$, $D \in \{3, 4\}$, $E \in \{5, 6\}$, $F \in \{5, 6\}$, $G \in \{6, 7, 8\}$, $H \in \{6, 7, 8\}$: the explanation why value 2 is removed from variable C corresponds to all missing arcs whose addition would merge the blue and the pink strongly connected components (i.e., the missing arcs corresponding to the assignments $A = 3$, $A = 4$, $B = 3$ and $B = 4$ that are depicted by thick pink lines)

diffn (variable replaced by *orthotope*), **disjunctive** (variable replaced by *task*), **global_cardinality** (control the number of occurrence of each value with a counter variable), **global_cardinality_low_up** (control the number of occurrence of each value with an interval), **lex_alldifferent** (variable replaced by *vector*), **nvalue** (count number of distinct values).

implied by: `alldifferent_consecutive_values`, `circuit`, `cycle`, `strictly_decreasing`, `strictly_increasing`.

implies: `alldifferent_except_0`, `not_all_equal`.

part of system of constraints: `neq`.

shift of concept: `alldifferent_on_intersection`, `alldifferent_same_value`.

soft variant: `alldifferent_except_0` (value 0 can be used several times), `open_alldifferent` (open constraint), `soft_alldifferent_ctr` (decomposition-based violation measure), `soft_alldifferent_var` (variable-based violation measure).

system of constraints: `k_alldifferent`.

used in reformulation: `in_interval_reified` (bound-consistency preserving reformulation).

uses in its reformulation: `elements_alldifferent`.

Keywords

characteristic of a constraint: `core`, `all different`, `disequality`, `automaton`, `automaton with array of counters`.

combinatorial object: `permutation`.

constraint type: `system of constraints`, `value constraint`.

filtering: `bipartite matching`, `bipartite matching in convex bipartite graphs`, `convex bipartite graph`, `flow`, `Hall interval`, `arc-consistency`, `bound-consistency`, `SAT`, `DFS-bottleneck`, `entailment`.

final graph structure: `one_succ`.

modelling exercises: `n-Amazon`, `zebra puzzle`.

problems: `maximum clique`, `graph colouring`.

puzzles: `n-Amazon`, `n-queen`, `Costas arrays`, `Euler knight`, `Golomb ruler`, `magic hexagon`, `magic square`, `zebra puzzle`, `Sudoku`.

Arc input(s)	VARIABLES
Arc generator	<code>CLIQUE</code> \mapsto <code>collection</code> (variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	<code>MAX_NSCC</code> \leq 1
Graph class	<code>ONE_SUCC</code>

Graph model

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.

Parts (A) and (B) of Figure 5.6 respectively show the initial and final graph associated with the **Example** slot. Since we use the `MAX_NSCC` graph property we show one of the largest strongly connected component of the final graph. The `alldifferent` holds since all the strongly connected components have at most one vertex: a value is used at most once.

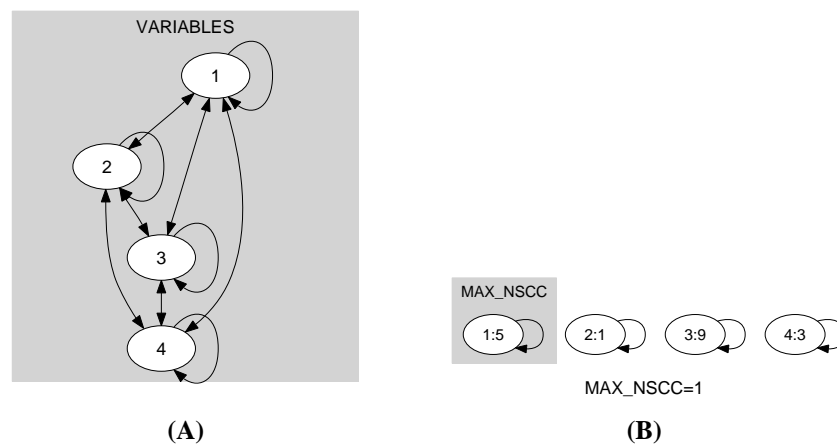


Figure 5.6: Initial and final graph of the `alldifferent` constraint

Automaton

Figure 5.7 depicts the **automaton** associated with the **alldifferent** constraint. To each item of the collection **VARIABLES** corresponds a signature variable S_i that is equal to 1. The **automaton** counts the number of occurrences of each value and finally imposes that each value is taken at most one time.

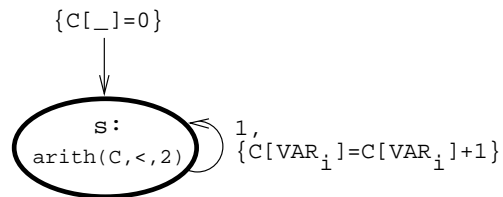


Figure 5.7: Automaton of the **alldifferent** constraint