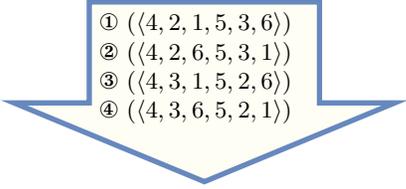


5.12 alldifferent

	DESCRIPTION	LINKS	GRAPH	AUTOMATON
Origin	[256]			
Constraint	alldifferent(VARIABLES)			
Synonyms	alldiff, alldistinct, distinct, bound_alldifferent, bound_alldiff, bound_distinct, rel.			
Argument	VARIABLES : collection(var-dvar)			
Restriction	required(VARIABLES, var)			
Purpose	Enforce all variables of the collection VARIABLES to take distinct values.			
Example	((5, 1, 9, 3))			
	The alldifferent constraint holds since all the values 5, 1, 9 and 3 are distinct.			
All solutions	<p>Figure 5.25 gives all solutions to the following non ground instance of the alldifferent constraint: $V_1 \in [2, 4]$, $V_2 \in [2, 3]$, $V_3 \in [1, 6]$, $V_4 \in [2, 5]$, $V_5 \in [2, 3]$, $V_6 \in [1, 6]$, $\text{alldifferent}(\langle V_1, V_2, V_3, V_4, V_5, V_6 \rangle)$.</p> 			
	<p>Figure 5.25: All solutions corresponding to the non ground example of the alldifferent constraint of the All solutions slot</p>			
Typical	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. 			
Arg. properties	Contractible wrt. VARIABLES.			

Usage

The `alldifferent` constraint occurs in most practical problems directly or indirectly. A classical example is the **n-queens** chess puzzle problem: Place n queens on an 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 `alldifferent` constraints. We associate to column i of the chessboard a domain variable X_i that gives the row number where the corresponding queen is located. The three `alldifferent` constraints are:

- `alldifferent`($X_1, X_2 + 1, \dots, X_n + n - 1$) for the descending diagonals,
- `alldifferent`(X_1, X_2, \dots, X_n) for the rows,
- `alldifferent`($X_1 + n - 1, X_2 + n - 2, \dots, X_n$) for the ascending diagonals.

They are respectively depicted by parts (A), (C) and (D) of Figure 5.26. Figure 5.27 makes explicit the link between the two families of diagonals and the corresponding `alldifferent` constraints. Note that this model matches the checker introduced by Gauss to test whether a permutation of row numbers is a solution or not to the 8 queens problem: first add the numbers 1 up to 8 to the permutation and check that the resulting numbers are distinct, second add the numbers 8 down to 1 and perform the same check [440, pages 165–166].

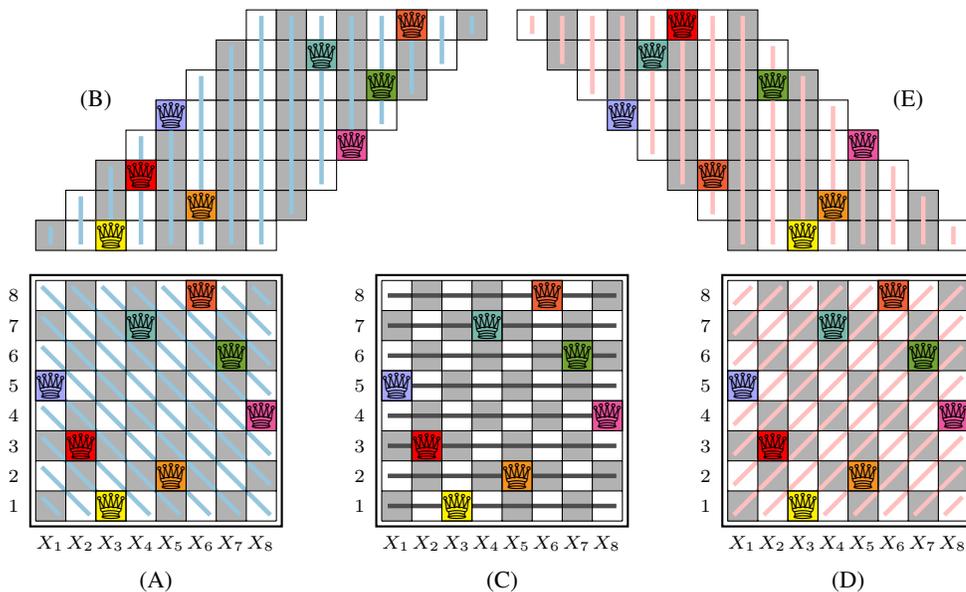


Figure 5.26: Descending diagonals (A-B), rows (C) and ascending diagonals (D-E)

A second example taken from [14], where the bipartite graph associated with the `alldifferent` constraint is *convex*, is a *ski assignment problem*: “a set of skiers have each specified the smallest and largest ski sizes they will accept from a given set of ski sizes”. The task is to find a ski size for each skier.

Examples such as **Costas arrays** and **Golomb rulers** involve one or several `alldifferent` constraints on *differences* of variables.

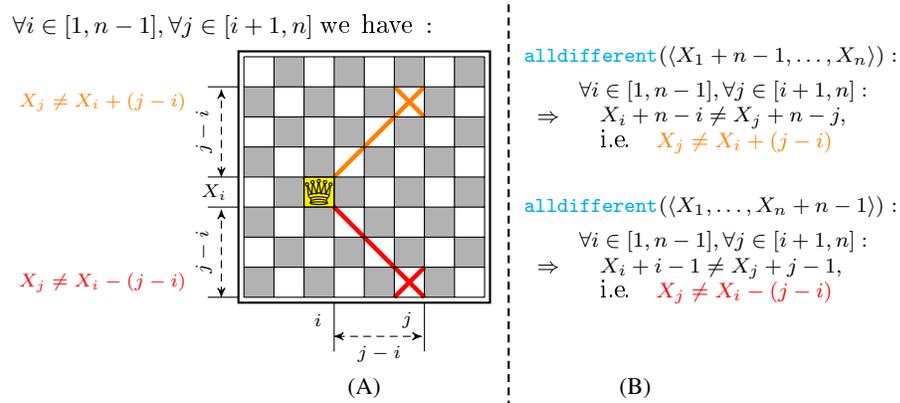


Figure 5.27: (A) For every pair of columns i, j ($i < j$), given the position X_i of the queen on column i , we respectively have from the ascending and descending diagonals that $X_j \neq X_i + (j - i)$ and $X_j \neq X_i - (j - i)$ (B) Equivalence of the two **alldifferent** constraints respectively associated with the ascending and descending diagonals with the two families of disequalities (i.e., the orange and the red one) depicted in Part (A)

Quite often, the **alldifferent** constraint is also used in conjunction with several **element** constraints, especially in the context of assignment problems [215, pages 372–374], or with several **precedence** constraints, especially in the context of symmetry breaking or scheduling problems [75].

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 did not have this form, it was specified in ALICE [255, 256] by asking for an injective correspondence f 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 in ALICE for checking the feasibility of the **alldifferent** constraint, even though the algorithm was already known in 1976. This is because 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 [255] explicitly mentions that specialised algorithms should be discarded. On the one hand, many people, especially from the OR community, have complained about such a radical statement [364, page 28]. On the other hand, the motivation of such a 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 principles. 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* [88, 153] 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 [421].

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, and Figure 2.4 of Section 2.1.5.

Within the context of [linear programming](#), relaxations of the `alldifferent` constraint are described in [441] and in [215, pages 362–367].

Within the context of *constraint-centered search heuristics*, G. Pesant and A. Zanarini [447] 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 [413]). Faster, but less accurate estimators, based on upper bounds of the number of solutions were proposed three years later by the same authors [448].

Given n variables taking their values within the interval $[1, n]$, the total number of solutions to the corresponding `alldifferent` constraint corresponds to the sequence [A000142](#) of the On-Line Encyclopaedia of Integer Sequences [392].

Algorithm

The first complete filtering algorithm was independently found by M.-C. Costa [123] and J.-C. Régin [340]. 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 [57, page 120].¹ Similarly, Dulmage-Mendelsohn decomposition [148] was also used recently by [129, 130] to characterise such edges and prune the corresponding variables both for the `alldifferent` constraint and for other constraints like `alldifferent_except_0`, `correspondence`, `inverse`, `same`, `used_by`, `global_cardinality_low_up`, `soft_alldifferent_var`, `soft_same_var`, `soft_used_by_var`. Assuming that all variables have no holes in their domain, M. Leconte came up with a filtering algorithm [259] based on edge finding. A first [bound-consistency](#) algorithm was proposed by Bleuzen-Guernalec *et al.* [78]. 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 [324, 266] and the second applies the same algorithm that is used to compute [arc-consistency](#), but achieves a speedup by exploiting the simpler structure [194] of the graph [281]. Ian P. Gent *et al.* discuss in [189] 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. Figures 2.1 and 2.2 of Section 2.1.3 illustrate the filtering of the `alldifferent` constraint with respect to [arc-consistency](#) and [bound-consistency](#). The leftmost part of Figure 3.29 illustrates a flow model for the `alldifferent` constraint where there is a one-to-one correspondence between feasible flows in the flow model and solutions of the `alldifferent` constraint.

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 [340].
- Range consistency is done in $O(n^2)$ by Leconte’s algorithm [259].
- [Bound-consistency](#) is performed in $O(n \log n)$ in [324, 281, 266]. If sort can be achieved in linear time, typically when the `alldifferent` constraint encodes a per-

¹A similar result is in fact given in [309].

mutation,² the worst case complexity of the algorithms described in [281, 266] goes down to $O(n)$.

Within the context of *explanations* [228], the explanation of the filtering algorithm that achieves *arc-consistency* for the `alldifferent` constraint is described in [359, 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 C_1 to a distinct strongly connected component C_2 is explained by all missing arcs starting from a descendant component of C_2 and ending in an ancestor component of C_1 (i.e., since the addition of any of these missing arcs would merge the strongly connected components C_1 and 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 5.28 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 .
- 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 $F = 8$.

An additional example for illustrating the generation of explanations for the `alldifferent` constraint when there are more values than variables is provided by Figure 2.3 of Section 2.1.4.

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).

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.

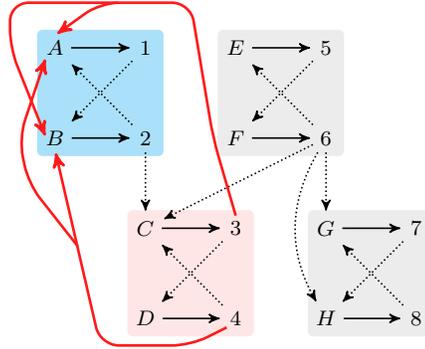


Figure 5.28: Strongly connected components of the residual graph illustrating the explanation of the removal of a value for the constraint `alldifferent`($\langle A, B, C, D, E, F, G, H \rangle$), $A \in \{1, 2\}$, $B \in \{1, 2\}$, $C \in \{2, 3, 4, 6\}$, $D \in \{3, 4\}$, $E \in \{5, 6\}$, $F \in \{5, 6\}$, $G \in \{6, 7, 8\}$, $H \in \{6, 7, 8\}$: 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)

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 [71]. 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 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 [74] that there is no polynomial sized decomposition that preserves `arc-consistency`.

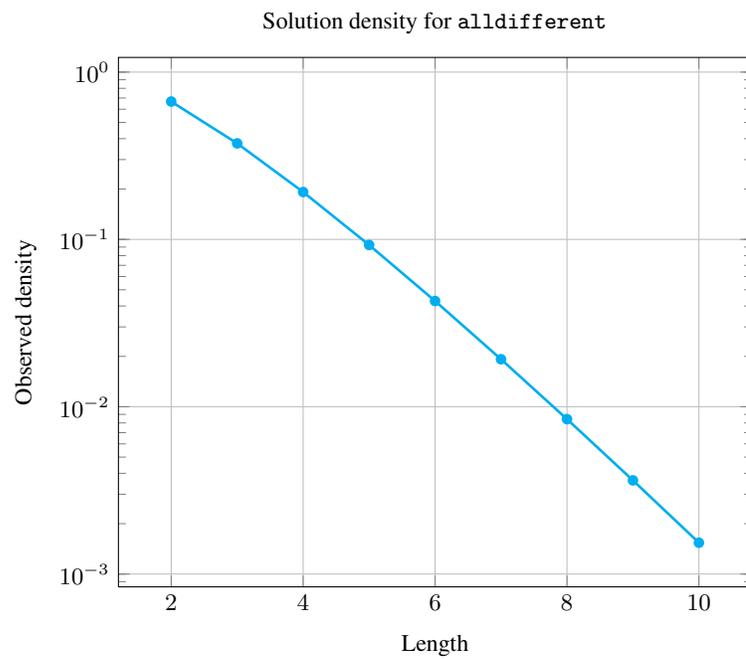
Finally the `alldifferent`(`VARIABLES`) constraint can also be reformulated as the conjunction `sort`(`VARIABLES`, `SORTED_VARIABLES`) \wedge `strictly-increasing`(`SORTED_VARIABLES`). Unlike the naive reformulation, i.e., a `disequality` constraint between each pair of variables, the `sort`-based reformulation is linear in space.

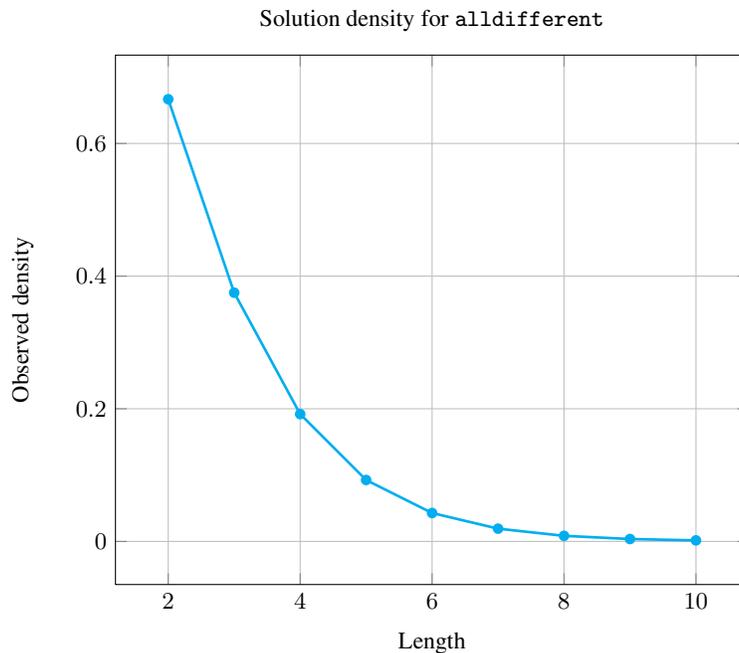
³How to encode the reified constraint $B_{i,l,u} \Leftrightarrow \text{VARIABLES}[i].\text{var} \in [l, u]$ with linear constraints is described in the `Reformulation slot` of the `in_interval_reified` constraint.

Counting

Length (n)	2	3	4	5	6	7	8	9	10
Solutions	6	24	120	720	5040	40320	362880	3628800	39916800

Number of solutions for alldifferent: domains 0.. n





Systems `allDifferent` in **Choco**, `linear` in **Gecode**, `alldifferent` in **JaCoP**, `alldiff` in **JaCoP**, `alldistinct` in **JaCoP**, `all_different` in **MiniZinc**, `all_different` in **SICStus**, `alldistinct` in **SICStus**.

Used in `alldifferent_consecutive_values`, `circuit_cluster`, `correspondence`, `cumulative_convex`, `max_occ_of_consecutive_tuples_of_values`, `max_occ_of_sorted_tuples_of_values`, `size_max_seq_alldifferent`, `size_max_starting_seq_alldifferent`, `sort_permutation`.

See also **common keyword:** `circuit`, `circuit_cluster`, `cycle`, `derangement` (*permutation*), `golomb` (*all different*), `proper_circuit` (*permutation*), `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), `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, multi_global_contiguity, not_all_equal.

negation: some_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), sort, strictly_increasing.

uses in its reformulation: cycle, elements_alldifferent, sort_permutation.

Keywords

characteristic of a constraint: core, all different, disequality, sort based reformulation, 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-Amazons, zebra puzzle.

problems: maximum clique, graph colouring.

puzzles: n-Amazons, n-queens, Costas arrays, Euler knight, Golomb ruler, magic hexagon, magic square, zebra puzzle, Sudoku.

Cond. implications

- alldifferent(VARIABLES)
implies lex_alldifferent(VECTORS : VARIABLES).
- alldifferent(VARIABLES)
implies soft_alldifferent_ctr(C, VARIABLES).
- alldifferent(VARIABLES)
implies balance(BALANCE, VARIABLES)
when BALANCE = 0.
- alldifferent(VARIABLES)
implies soft_all_equal_max_var(N, VARIABLES)
when $N < |\text{VARIABLES}|$.
- alldifferent(VARIABLES)
implies soft_all_equal_min_var(N, VARIABLES)
when $N > |\text{VARIABLES}|$.

- `alldifferent(VARIABLES)`
implies `change(NCHANGE, VARIABLES, CTR)`
when $NCHANGE = |VARIABLES| - 1$
and $CTR \in [\neq]$.
- `alldifferent(VARIABLES)`
implies `circular_change(NCHANGE, VARIABLES, CTR)`
when $NCHANGE = |VARIABLES|$
and $CTR \in [\neq]$.
- `alldifferent(VARIABLES)`
implies `longest_change(SIZE, VARIABLES, CTR)`
when $SIZE = |VARIABLES|$
and $CTR \in [\neq]$.
- `alldifferent(VARIABLES)`
with $|VARIABLES| > 0$
implies `length_first_sequence(LEN, VARIABLES)`
when $LEN = 1$.
- `alldifferent(VARIABLES)`
with $|VARIABLES| > 0$
implies `length_last_sequence(LEN, VARIABLES)`
when $LEN = 1$.
- `alldifferent(VARIABLES)`
with $|VARIABLES| > 0$
implies `min_nvalue(MIN, VARIABLES)`
when $MIN = 1$.

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

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 5.29 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.

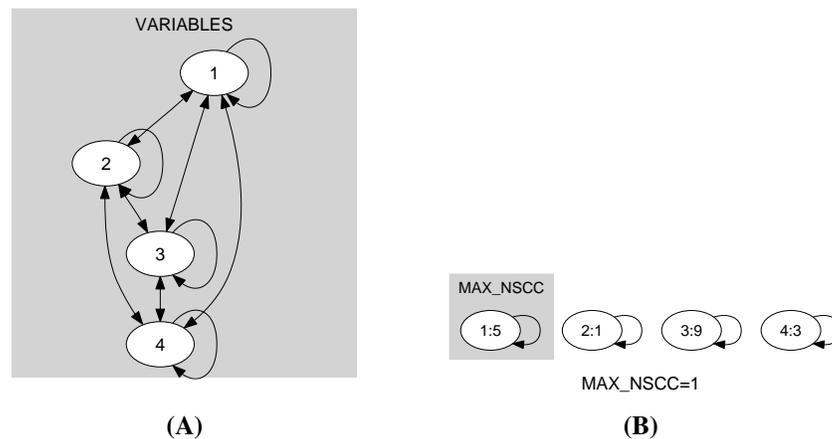


Figure 5.29: Initial and final graph of the `alldifferent` constraint

Automaton

Figure 5.30 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.

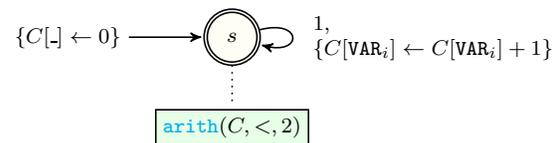


Figure 5.30: Automaton of the `alldifferent` constraint

Quiz**EXERCISE 1 (checking whether a ground instance holds or not)^a**

- A. Does the constraint `alldifferent`(⟨5, 1, 4, 8, 1⟩) hold?
- B. Does the constraint `alldifferent`(⟨8, 2, 4, 3⟩) hold?
- C. Does the constraint `alldifferent`(⟨0⟩) hold?

^aHint: go back to the definition of `alldifferent`.

EXERCISE 2 (finding all solutions)^a

Give all the solutions to the constraint:

$$\begin{cases} V_1 \in [3, 5], & V_2 \in [3, 4], & V_3 \in [2, 7], \\ V_4 \in [3, 4], & V_5 \in [2, 7], \\ \text{alldifferent}(\langle V_1, V_2, V_3, 6, V_4, V_5 \rangle). \end{cases}$$

^aHint: identify infeasible values, enumerate solutions in lexicographic order.

EXERCISE 3 (finding all solutions)^a

Give all the solutions to the constraint:

$$\begin{cases} V_1 \in [4, 6], & V_2 \in [1, 3], & V_3 \in [1, 4], & V_4 \in [1, 2], \\ V_5 \in [4, 7], & V_6 \in [4, 6], & V_7 \in [1, 2], \\ \text{alldifferent}(\langle V_1, V_2, V_3, V_4, V_5, V_6, V_7 \rangle). \end{cases}$$

^aHint: focus on variables with smallest domain first, identify Hall intervals for finding infeasible values, enumerate solutions in lexicographic order.

EXERCISE 4 (finding all solutions)^a

Give all the solutions to the constraint:

$$\left\{ \begin{array}{ll} V_1 \in \{1, 3\}, & V_2 \in \{1, 2, 3, 4\}, \\ V_3 \in \{1, 5\}, & V_4 \in \{1, 2, 3, 4, 5, 6\}, \\ V_5 \in \{3, 5\}, & \\ \text{alldifferent}(\langle V_1, V_2, V_3, V_4, V_5 \rangle). \end{array} \right.$$

^aHint: focus on variables with smallest domain first, identify Hall sets for finding infeasible values, enumerate solutions in lexicographic order.

EXERCISE 5 (identifying infeasible values)^a

Identify all variable-value pairs (V_i, val) ($1 \leq i \leq 6$), such that the following constraint has no solution when variable V_i is assigned value val :

$$\left\{ \begin{array}{lll} V_1 \in \{1, 2, 4\}, & V_2 \in \{1, 2, 3, 4, 6\}, & V_3 \in \{1, 2, 6\}, \\ V_4 \in [1, 6], & V_5 \in \{1, 4, 6\}, & V_6 \in \{2, 4, 6\}, \\ \text{alldifferent}(\langle V_1, V_2, V_3, V_4, V_5, V_6 \rangle). \end{array} \right.$$

^aHint: focus on variables with smallest domain first, identify Hall sets for finding infeasible values.

EXERCISE 6 (identifying infeasible values and counting)^a

- A. Identify six variable-value pairs (V_i, val) ($1 \leq i \leq 9$), such that the following conjunction of constraints has no solution when variable V_i is assigned value val .

$$\left\{ \begin{array}{l} V_1 \in [1, 7], V_2 \in [1, 7], V_3 \in [1, 7], \\ V_4 \in [1, 4], V_5 \in [1, 4], V_6 \in [1, 4], \\ V_7 \in [3, 6], V_8 \in [3, 6], V_9 \in [3, 6], \\ \text{alldifferent}(\langle V_1, V_2, V_3, V_4, V_5, V_6 \rangle), \\ \text{alldifferent}(\langle V_1, V_2, V_3, V_7, V_8, V_9 \rangle). \end{array} \right.$$

- B. Describe concisely the structure of the set of solutions and derive the total number of solutions.

^aHint: group together variables that belong to the same set of constraints and reason on the number of distinct values assigned to such groups.

EXERCISE 7 (variable-based degree of violation)^a

Compute the variable-based degree of violation^b of the following constraints:

- A. `alldifferent`($\langle 2, 2, 2, 2 \rangle$),
- B. `alldifferent`($\langle 3, 1, 5, 2, 7 \rangle$),
- C. `alldifferent`($\langle 5, 5, 0, 5, 5, 0, 7 \rangle$).

^aHint: focus on the groups of variables that are assigned the same value.

^bGiven a constraint for which all variables are fixed, the *variable-based degree of violation* is the minimum number of variables to assign differently in order to satisfy the constraint.

EXERCISE 8 (preventing conflict around the table^{ab})

Provide a concise and efficient model for the following problem. Given a set \mathcal{M} of n men, a set \mathcal{W} of n women, a set of pairs \mathcal{C} where each pair $(m, w) \in \mathcal{C}$ represents a conflict between the man m ($m \in \mathcal{M}$) and the woman w ($w \in \mathcal{W}$), a rectangular table, the goal is to place on one side of the table all the n men and on the opposite side all the n women in such a way that two persons that are in conflict do not sit face to face.

^aAdapted from the 2011 constraint programming exam at Polytechnique, C. Dürr.

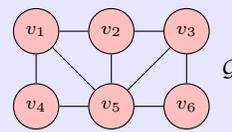
^bHint: break some symmetry of the problem.

EXERCISE 9 (identifying equalities from a clique of disequalities^a)

[CONTEXT] Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, colour each vertex $v \in \mathcal{V}$ in such a way that (1) two vertices that are linked by an edge of the set of edges \mathcal{E} are not assigned the same colour, and (2) no more than m distinct colours are used to colour all the vertices of \mathcal{G} . The goal of the exercise is to find out necessary conditions for this problem that go beyond the cardinality of a (maximum) clique.

A. [IDENTIFYING PAIRS OF VERTICES THAT SHOULD BE ASSIGNED THE SAME COLOUR]

- (a) Given the vertices of the following graph \mathcal{G} to colour with at most three distinct colours, explain why vertices v_1 and v_3 should be assigned the same colour.



- (b) For graph \mathcal{G} provide all pairs of vertices that should be assigned the same colour if no more than three distinct colours have to be used.

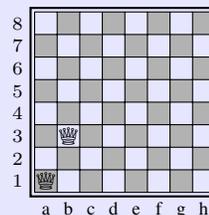
B. [GENERALISING THE NECESSARY CONDITION]

Given a clique of n vertices \mathcal{C} of the graph \mathcal{G} , let $\mathcal{V}_{\mathcal{C}}$ denotes the set of vertices that do not belong to \mathcal{C} and that are all connected to all vertices of \mathcal{C} . Assuming that one should use at most m distinct colours provide a necessary condition on the set $\mathcal{V}_{\mathcal{C}}$.

^aHint: restrict extra values wrt a clique of disequalities.

EXERCISE 10 (8-queens: unfeasibility of a partial solution^a)

Consider the 8-queens problem^b where we start filling the chessboard in a systematic way: we place a first queen in a1 and a second queen in b3. Prove that it is not possible to extend this partial assignment to a complete solution.



^aHint: consider each of the 4 remaining positions on column c; extract information from the conjunction of the three alldifferent constraints that allows the modelling of the n -queens problem.

^bPlace 8 queens on an 8 by 8 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.

SOLUTION TO EXERCISE 1

- A. No, since value 1 is used twice.
- B. Yes, since all values 8, 2, 4 and 3 are distinct.
- C. Yes, since value 0 is only used once.

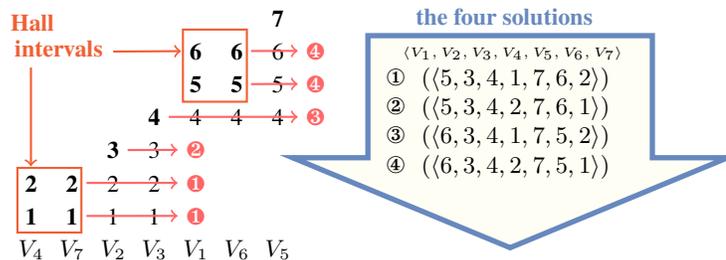
SOLUTION TO EXERCISE 2**the four solutions**

- $\langle V_1, V_2, V_3, V_4, V_5 \rangle$
- ① $\langle 5, 3, 2, 4, 7 \rangle$
 - ② $\langle 5, 3, 7, 4, 2 \rangle$
 - ③ $\langle 5, 4, 2, 3, 7 \rangle$
 - ④ $\langle 5, 4, 7, 3, 2 \rangle$

Values 3 and 4 have to be assigned to the two variables V_2 and V_4 . Consequently, V_1, V_3 and V_5 are different from 3 and 4.

Values 3, 4 and 5 have to be assigned to V_1, V_2 and V_4 . Value 6 is directly mentioned in the constraint. Consequently the two remaining variables V_3 and V_5 can only be assigned values 2 and 7.

SOLUTION TO EXERCISE 3

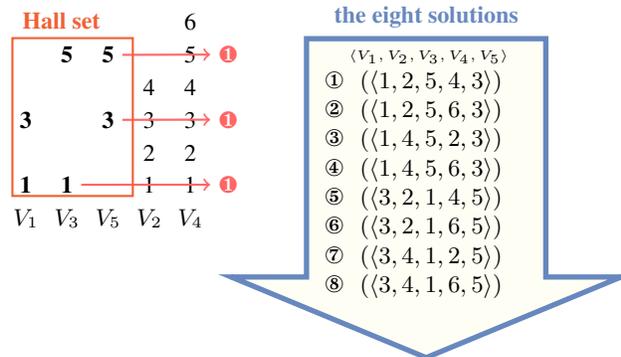


Let us reorder the variables by increasing minimum value, and by increasing maximum value in case of tie, for instance, $V_4, V_7, V_2, V_3, V_1, V_6, V_5$.

- ① Since values 1 and 2 have to be assigned to V_4 and V_7 (interval $[1, 2]$ is a Hall interval^a), they cannot be assigned to the other variables and consequently V_2 is fixed to 3.
- ② Since V_2 is fixed to 3, V_3 is fixed to 4.
- ③ Since V_3 is fixed to 4, V_1 and V_6 can only be assigned values 5 or 6 (interval $[5, 6]$ is a Hall interval).
- ④ Since values 5 and 6 cannot be assigned to V_5 , V_5 is fixed to 7.

^aGiven a set of domain variables, a *Hall interval* is an interval of values $[\ell, u]$ such that there are $u - \ell + 1$ variables whose domains are contained in $[\ell, u]$.

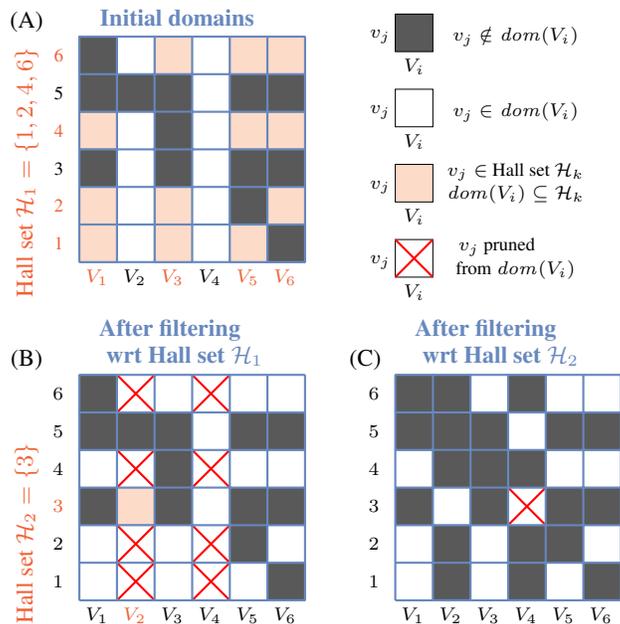
SOLUTION TO EXERCISE 4



Let us reorder the variables by increasing domain size, increasing minimum, and increasing maximum in case of tie, i.e., V_1, V_3, V_5, V_2, V_4 . Since values 1, 3 and 5 have to be assigned to V_1, V_3 and V_5 ($\{1, 3, 5\}$ is a Hall set^a), they cannot be assigned to the other variables and consequently values 1, 3 and 5 are removed from V_2 and V_4 (see ①).

^aGiven a set of domain variables, a *Hall set* is a set of values \mathcal{H} such that there are $|\mathcal{H}|$ variables whose domains are contained in \mathcal{H} .

SOLUTION TO EXERCISE 5



1. In part (A) we first identify the Hall set^a $\mathcal{H}_1 = \{1, 2, 4, 6\}$ which contains the domains of variables V_1, V_3, V_5 and V_6 .
2. In part (B) we remove values 1, 2, 4 and 6 from those variables for which the domain is not included within the Hall set \mathcal{H}_1 , namely V_2 and V_4 , see \times .
3. After the previous filtering we identify in part (B) a new Hall set $\mathcal{H}_2 = \{3\}$ which contains the domain of V_2 .
4. Finally in part (C) we remove value 3 from those variables for which the domain is not included within the Hall set \mathcal{H}_2 , namely V_4 , see \times .

^aGiven a set of domain variables, a Hall set is a set of values \mathcal{H} such that there are $|\mathcal{H}|$ variables whose domains are contained in \mathcal{H} .

SOLUTION TO EXERCISE 6

- A.** (i) The cardinality of the union of the domains of V_1, V_2, \dots, V_9 is equal to 7. Since V_1, V_2 and V_3 will be assigned 3 distinct values, the remaining variables V_4, V_5, \dots, V_9 should not be assigned more than $7 - 3 = 4$ distinct values.
- (ii) V_4, V_5, \dots, V_9 can be partitioned in two sets $\{V_4, V_5, V_6\}$ and $\{V_7, V_8, V_9\}$ which respectively correspond to the variables that only belong to the first and to the second all different. The first set will be assigned distinct values in interval $[1, 4]$, while the second set will be assigned distinct values in interval $[3, 6]$.
- (iii) Since V_4, V_5, \dots, V_9 should not be assigned more than 4 distinct values, the two values 3 and 4 that belong both to $[1, 4]$ and $[3, 6]$ should be both assigned to $\{V_4, V_5, V_6\}$ and to $\{V_7, V_8, V_9\}$. Consequently values 3 and 4 cannot be assigned to variables V_1, V_2 and V_3 .
- B.** As illustrated by the next figure, we have four families of solutions ①, ②, ③ and ④ where the three sets of variables $\{V_1, V_2, V_3\}$, $\{V_4, V_5, V_6\}$ and $\{V_7, V_8, V_9\}$ are assigned values from three distinct set of values. This leads to a total number of solutions of $4 \cdot 3! \cdot 3! = 864$.

$\{V_1, V_2, V_3\}$	$\{V_1, V_2, V_3\}$	$\{V_1, V_2, V_3\}$	$\{V_1, V_2, V_3\}$
$\{1, 5, 7\}$	$\{1, 6, 7\}$	$\{2, 5, 7\}$	$\{2, 6, 7\}$
$\{V_4, V_5, V_6\}$	$\{V_4, V_5, V_6\}$	$\{V_4, V_5, V_6\}$	$\{V_4, V_5, V_6\}$
$\{2, 3, 4\}$	$\{2, 3, 4\}$	$\{1, 3, 4\}$	$\{1, 3, 4\}$
$\{V_7, V_8, V_9\}$	$\{V_7, V_8, V_9\}$	$\{V_7, V_8, V_9\}$	$\{V_7, V_8, V_9\}$
$\{3, 4, 6\}$	$\{3, 4, 5\}$	$\{3, 4, 6\}$	$\{3, 4, 5\}$
①	②	③	④

SOLUTION TO EXERCISE 7

- A. The degree of violation is equal to 3 since at least three occurrences of value 2 (e.g. the three in red) out of the four occurrences of value 2 need to be assigned differently (e.g., 3, 4, 5 in blue) in order to obtain a solution.

$$\text{alldifferent}(\langle 2, \overset{3,4,5}{\color{red}2}, \color{red}2, \color{red}2 \rangle)$$

- B. The degree of violation is equal to 0 since the constraint holds, i.e. no value needs to be assigned differently.

- C. The degree of violation is equal to 4 since at least three occurrences of value 5 and one occurrence of value 0 (e.g. the three 5 and the 0 in red) need to be assigned differently (e.g., 1, 3, 6, 4 in blue) in order to obtain a solution.

$$\text{alldifferent}(\langle \overset{1,3,6,4}{\color{red}5}, \color{red}5, \color{red}0, \color{red}5, \color{red}5, 0, 7 \rangle)$$
SOLUTION TO EXERCISE 8

Without loss of generality let us assume that the sets \mathcal{M} and \mathcal{W} are both equal to $\{1, 2, \dots, n\}$. We associate to each woman w in \mathcal{W} a variable F_w providing the man which sits in front of w .^a

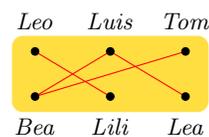
- ① To prevent any conflict, the initial domain of each variable F_w ($w \in \mathcal{W}$) is set to all the men of \mathcal{M} that are not in conflict with woman w , i.e. the men $m \in \mathcal{M}$ such that $(m, w) \notin \mathcal{C}$.
- ② To enforce the fact that each woman can only sit in front of a single man we enforce an $\text{alldifferent}(\langle F_1, F_2, \dots, F_n \rangle)$ constraint.

^aNote that this model does not introduce variables for the men, i.e. each man is a value and each woman a variable. The model also eliminates some symmetry, i.e., it does not care where a woman sits.

AN INSTANCE

$$\begin{aligned} \mathcal{W} &= \{Bea, Lea, Lili\} \\ \mathcal{M} &= \{Leo, Luis, Tom\} \\ \mathcal{C} &= \{(Luis, Bea), \\ &\quad (Tom, Bea), \\ &\quad (Luis, Lea), \\ &\quad (Leo, Lili)\} \end{aligned}$$
A SOLUTION

(red edges correspond to conflicts)

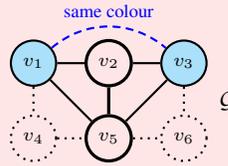


Bea Lili Lea

SOLUTION TO EXERCISE 9

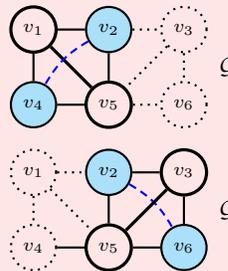
A. [IDENTIFYING PAIRS OF VERTICES THAT SHOULD BE ASSIGNED THE SAME COLOUR]

(a) Since we have to use at most 3 distinct colours, since v_2 and v_5 use two distinct colours, and since v_1 and v_3 are both linked to v_2 and v_5 , we infer that v_1 and v_3 both have to use the third and last available colour.



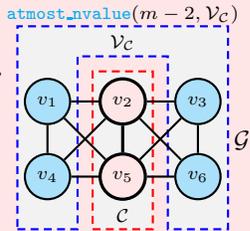
(b) For a similar reason:

- v_2 and v_4 use the same colour since they are both linked to v_1 and v_5 .
- v_2 and v_6 use the same colour since they are both linked to v_3 and v_5 .



B. [GENERALISING THE NECESSARY CONDITION]

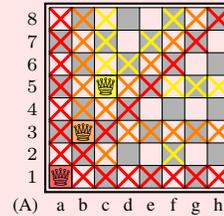
Since we should use at most m distinct colours and since all vertices of the set \mathcal{V}_C are linked to all vertices of the clique \mathcal{C} of n vertices, the set \mathcal{V}_C should use at most $m - n$ distinct colours. In the previous setting we had $m = 3$ and $n = 2$, i.e., one unique colour for all elements of \mathcal{V}_C . The figure on the right illustrates the constraint generated wrt the clique $\mathcal{C} = \{v_2, v_5\}$.



SOLUTION TO EXERCISE 10

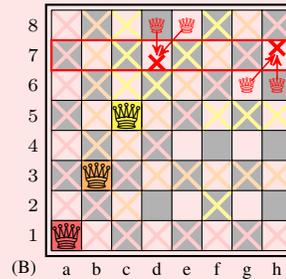
We do case reasoning depending on where we place the queen on the third column. After placing the third queen we mark all cells that are located on the same column, row, or diagonal of one of the three already placed queens. Then we focus on the rows or columns for which no more than three consecutive cells are still empty since it allows to prune the next row or column.

- [PLACING A QUEEN ON c5]
After marking all cells that are located on a same column, row, or diagonal than a1, b3, and c5 we get the chessboard shown on the right.



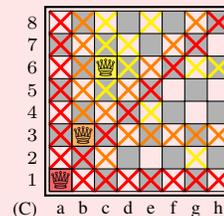
(A) a b c d e f g h

Then we focus on row 8, which contains only two consecutive free cells. Since the eighth row must contain one queen this queen will be located at position d8 or e8. In both cases the cell d7 will be forbidden. By performing the same reasoning on the sixth row we also find out that the cell h7 is forbidden. As a result no queen can be placed on row 7.



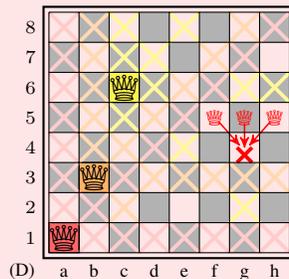
(B) a b c d e f g h

- [MOVING THE THIRD QUEEN FROM c5 TO c6]
After marking all cells that are located on a same column, row, or diagonal than a1, b3, and c6 we get the chessboard shown on the right.

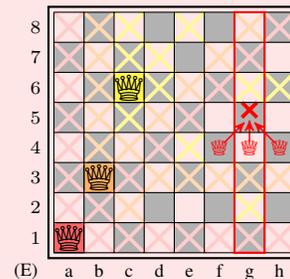


(C) a b c d e f g h

We focus on row 5, which contains only three consecutive free cells, see (D). Since row 5 must contain a queen, this queen will be located at position f5, g5, or h5. In all three cases the cell g4 will be forbidden. Similarly by considering the fourth row, see (E), we find out that no queen can be placed on g5. As a result no queen can be placed on column g.



(D) a b c d e f g h



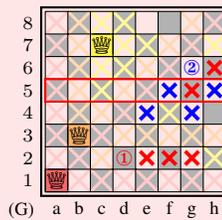
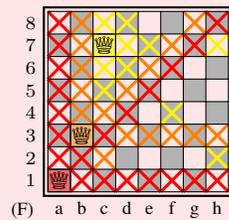
(E) a b c d e f g h

SOLUTION TO EXERCISE 10 (continued)

- [MOVING THE THIRD QUEEN FROM c6 TO c7]

After marking all cells that are located on a same column, row, or diagonal than a1, b3, and c7 we get the chessboard shown in (F).

Since on column d only position d2 is free a queen ① is placed on d2 and we mark with small red crosses the new forbidden positions, see (G). Then on row 6 only position g6 is free. Consequently a queen ② is placed on g6, which forbids all remaining free positions on row 5.



- [MOVING THE THIRD QUEEN FROM c7 TO c8]

After marking all cells that are located on a same column, row, or diagonal than a1, b3, and c8 we get the chessboard shown in (H).

Then we focus on row 5, see (I), which contains only two consecutive free cells. Since the fifth row must contain one queen this queen will be located at position g5 or h5. In both cases the cell g6 is forbidden.

By performing the same reasoning we also find out that cell h6 is forbidden. Since on row 6 only position d6 is free, a queen ① is placed on d6 and we mark with small blue crosses the new forbidden positions. Then on column f only position f2 is free. Consequently a queen ② is placed on f2 and we mark with small green crosses the new forbidden positions. Now row 7 and column e have only one free cell, namely h7 and e4 which are located on the same diagonal. Consequently, we cannot place the last two queens on e4 and h7. Hence the first two queens cannot be placed on a1 and b3.

