5.395 symmetric_alldifferent

DESCRIPTION LINKS GRAPH

Origin [345]

Constraint `symmetric_alldifferent(NODES)`

Synonyms `symmetric_alldiff`, `symmetric_alldistinct`, `symm_alldifferent`, `symm_alldiff`, `symm_alldistinct`, `one_factor`, `two_cycle`.

Argument NODES : collection(index−int, succ−dvar)

Restrictions

|NODES| mod 2 = 0
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index ≤ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ ≤ |NODES|

Purpose All variables associated with the succ attribute of the NODES collection should be pairwise distinct. In addition enforce the following condition: if variable NODES[i].succ takes value j with $j \neq i$ then variable NODES[j].succ takes value i. This can be interpreted as a graph-covering problem where one has to cover a digraph G with circuits of length two in such a way that each vertex of G belongs to a single circuit.

Example

$$\left(\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-3,\\ \texttt{index}-2 & \texttt{succ}-4,\\ \texttt{index}-3 & \texttt{succ}-1,\\ \texttt{index}-4 & \texttt{succ}-2\end{array}\right\rangle\right)$$

The `symmetric_alldifferent` constraint holds since:

- NODES[1].succ = 3 ⇔ NODES[3].succ = 1,
- NODES[2].succ = 4 ⇔ NODES[4].succ = 2.

All solutions Figure 5.750 gives all solutions to the following non ground instance of the `symmetric_alldifferent` constraint: $S_1 \in [1,4]$, $S_2 \in [1,3]$, $S_3 \in [1,4]$, $S_4 \in [1,3]$, `symmetric_alldifferent`$(\langle 1\ S_1, 2\ S_2, 3\ S_3, 4\ S_4\rangle)$.

Typical |NODES| ≥ 4

Symmetry Items of NODES are permutable.

Usage As it was reported in [345, page 420], this constraint is useful to express matches between persons or between teams. The `symmetric_alldifferent`constraint also appears implicitly in the *cycle cover problem* and corresponds to the four conditions given in section 1 *Modeling the Cycle Cover Problem* of [308].

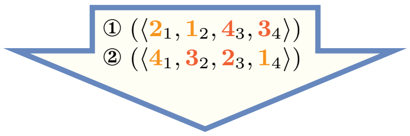


Figure 5.750: All solutions corresponding to the non ground example of the `symmetric_alldifferent` constraint of the **All solutions** slot (the `index` attribute is displayed as indices of the `succ` attribute)

Remark This constraint is referenced under the name `one_factor` in [211] as well as in [409]. From a modelling point of view this constraint can be expressed with the `cycle` constraint [41] where one imposes the additional condition that each cycle has only two nodes.

Algorithm A filtering algorithm for the `symmetric_alldifferent` constraint was proposed by J.-C. Régin in [345]. It achieves arc-consistency and its running time is dominated by the complexity of finding all edges that do not belong to any maximum cardinality matching in an undirected n-vertex, m-edge graph, i.e., $O(m \cdot n)$.

For the soft case of the `symmetric_alldifferent` constraint where the cost is the minimum number of variables to assign differently in order to get back to a solution, a filtering algorithm achieving arc-consistency is described in [131, 130]. It has a complexity of $O(p \cdot m)$, where p is the number of maximal extreme sets in the value graph associated with the constraint and m is the number of edges. It iterates over extreme sets and not over vertices as in the algorithm due to J.-C. Régin.

Reformulation The `symmetric_alldifferent`(NODES) constraint can be expressed in term of a conjunction of $|\texttt{NODES}|^2$ reified constraints of the form $\texttt{NODES}[i].\texttt{succ} = j \Leftrightarrow \texttt{NODES}[j].\texttt{succ} = i$ $(1 \leq i, j \leq |\texttt{NODES}|)$. The `symmetric_alldifferent` constraint can also be reformulated as an `inverse` constraint as shown below:

$$\texttt{symmetric\_alldifferent}\left(\left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-s_1, \\ \texttt{index}-2 & \texttt{succ}-s_2, \\ \vdots & \vdots \\ \texttt{index}-n & \texttt{succ}-s_n \end{array}\right\rangle\right)$$

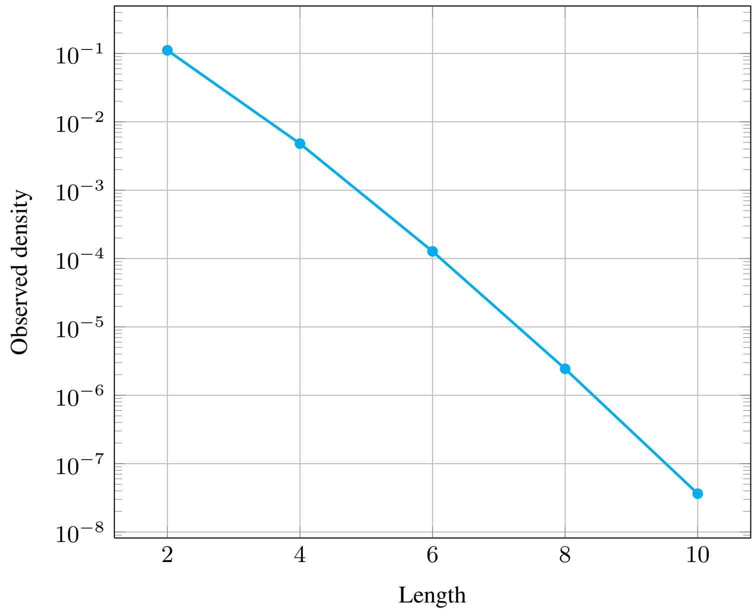
$$\texttt{inverse}\left(\left\langle\begin{array}{lll} \texttt{index}-1 & \texttt{succ}-s_1 & \texttt{pred}-s_1, \\ \texttt{index}-2 & \texttt{succ}-s_2 & \texttt{pred}-s_2, \\ \vdots & \vdots & \vdots \\ \texttt{index}-n & \texttt{succ}-s_n & \texttt{pred}-s_n \end{array}\right\rangle\right)$$

Counting

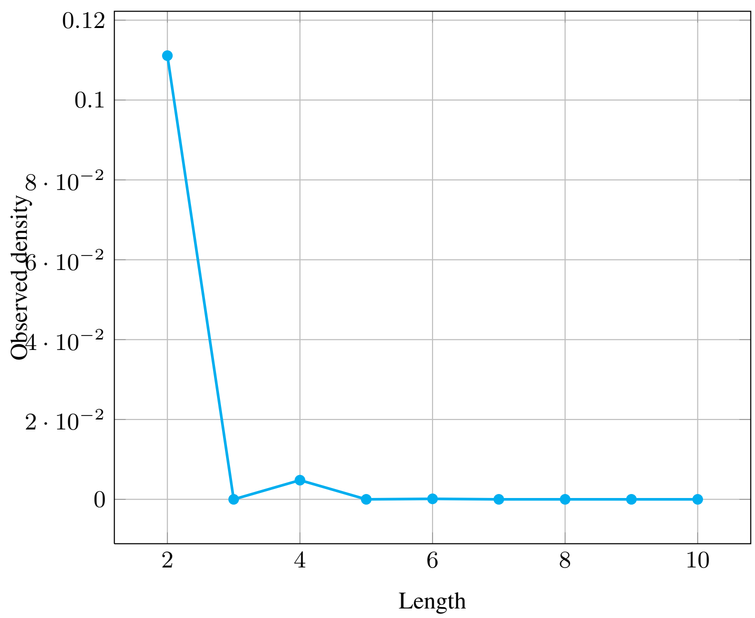
Length (n)	2	3	4	5	6	7	8	9	10
Solutions	1	0	3	0	15	0	105	0	945

Number of solutions for `symmetric_alldifferent`: domains $0..n$

Solution density for `symmetric_alldifferent`



Solution density for `symmetric_alldifferent`



See also **common keyword:** `alldifferent`, `cycle`, `inverse` *(permutation)*.

implies: `derangement`, `symmetric_alldifferent_except_0`, `symmetric_alldifferent_loop`.

implies (items to collection): `k_alldifferent`, `lex_alldifferent`.

related: `roots`.

Keywords

application area: sport timetabling.

characteristic of a constraint: all different, disequality.

combinatorial object: permutation, involution, matching.

constraint type: graph constraint, timetabling constraint, graph partitioning constraint.

filtering: arc-consistency.

final graph structure: circuit.

modelling: cycle.

Cond. implications

- `symmetric_alldifferent`(NODES)
  implies `balance_cycle`(BALANCE, NODES)
  when $\mathtt{BALANCE} = 0$.

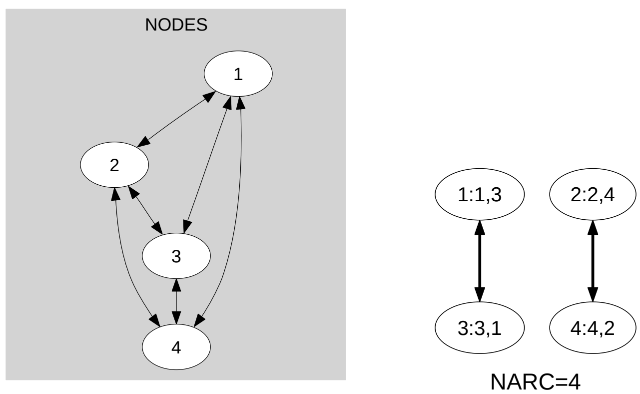
- `symmetric_alldifferent`(NODES)
  implies `cycle`(NCYCLE, NODES)
  when $2 * \mathtt{NCYCLE} = |\mathtt{NODES}|$.

- `symmetric_alldifferent`(NODES)
  implies `permutation`(VARIABLES : NODES).

Arc input(s)	`NODES`
Arc generator	$CLIQUE(\neq) \mapsto$ `collection(nodes1, nodes2)`
Arc arity	2
Arc constraint(s)	• `nodes1.succ` $=$ `nodes2.index` • `nodes2.succ` $=$ `nodes1.index`
Graph property(ies)	$\mathbf{NARC} = |$`NODES`$|$

Graph model In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices.

Parts (A) and (B) of Figure 5.751 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.751: Initial and final graph of the `symmetric_alldifferent` constraint

Signature Since all the `index` attributes of the `NODES` collection are distinct, and because of the first condition `nodes1.succ` $=$ `nodes2.index` of the arc constraint, each vertex of the final graph has at most one successor. Therefore the maximum number of arcs of the final graph is equal to the maximum number of vertices $|$`NODES`$|$ of the final graph. So we can rewrite $\mathbf{NARC} = |$`NODES`$|$ to $\mathbf{NARC} \geq |$`NODES`$|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.