5.397 symmetric_alldifferent_loop

DESCRIPTION LINKS GRAPH

Origin Derived from symmetric_alldifferent

Constraint symmetric_alldifferent_loop(NODES)

Synonyms symmetric_alldiff_loop, symmetric_alldistinct_loop, symm_alldifferent_loop, symm_alldiff_loop, symm_alldistinct_loop.

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

Restrictions
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq$ |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 is assigned value j then variable NODES$[j]$.succ is assigned value i. Note that i and j are not necessarily distinct. This can be interpreted as a graph-covering problem where one has to cover a digraph G with circuits of length two or one in such a way that each vertex of G belongs to a single circuit.

Example

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

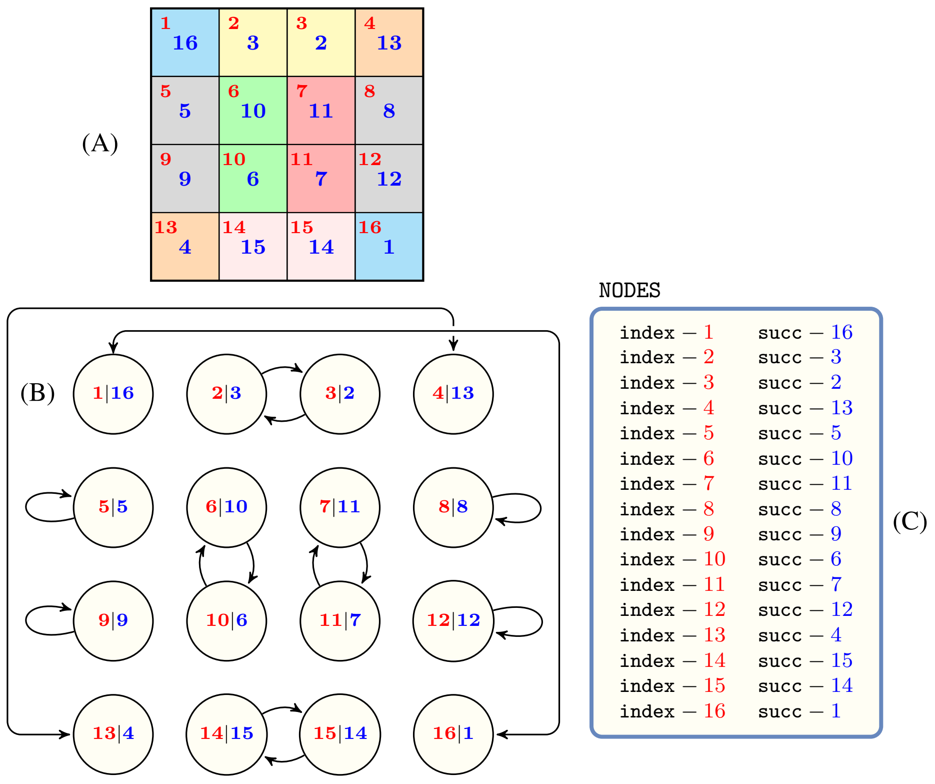
The symmetric_alldifferent_loop constraint holds since:

- We have two loops respectively corresponding to NODES[1].succ $=1$ and NODES[3].succ $=3$.
- We have one circuit of length 2 corresponding to NODES[2].succ $=4 \Leftrightarrow$ NODES[4].succ $=2$.

Figure 5.753 provides a second example involving a symmetric_alldifferent_loop constraint.

All solutions Figure 5.755 gives all solutions to the following non ground instance of the symmetric_alldifferent_loop constraint: $S_1 \in [2,5]$, $S_2 \in [1,3]$, $S_3 \in [1,4]$, $S_4 \in [2,4]$, $S_5 \in [1,5]$, symmetric_alldifferent_loop($\langle 1\ S_1, 2\ S_2, 3\ S_3, 4\ S_4, 5\ S_5\rangle$).

Typical |NODES| ≥ 4



NODES

index − 1	succ − 16
index − 2	succ − 3
index − 3	succ − 2
index − 4	succ − 13
index − 5	succ − 5
index − 6	succ − 10
index − 7	succ − 11
index − 8	succ − 8
index − 9	succ − 9
index − 10	succ − 6
index − 11	succ − 7
index − 12	succ − 12
index − 13	succ − 4
index − 14	succ − 15
index − 15	succ − 14
index − 16	succ − 1

Figure 5.753: (A) Magic square Duerer where cells that belong to a same cycle are coloured identically by a colour different from grey; each cell has an *index* in its upper left corner (in red) and a *value* (in blue). (B) Corresponding graph where there is an arc from node i to node j if and only if the value of cell i is equal to the index of cell j. (C) Collection of nodes passed to the `symmetric_alldifferent_loop` constraint: the four self-loops of the graph correspond to the four grey cells of the magic square such that the value of the cell (in blue) is equal to the index of the cell (in red).

Symmetry Items of `NODES` are permutable.

Algorithm An arc-consistency filtering algorithm for the `symmetric_alldifferent_loop` constraint is described in [131, 130]. The algorithm is based on the following ideas:

- First, one can map solutions of the `symmetric_alldifferent_loop` constraint to perfect (g, f)-matchings in a non-bipartite graph derived from the domain of the variables of the constraint where $g(x) = 0$, $f(x) = 1$ for vertices x which have a self-loop, and $g(x) = f(x) = 1$ for all the remaining vertices. A *perfect* (g, f)-*matching* $\mathcal{M}$ of a graph is a subset of edges such that every vertex x is incident with the number of edges in $\mathcal{M}$ between $g(x)$ and $f(x)$.
- Second, Gallai-Edmonds decomposition [179, 150] allows to find out all edges that do not belong any perfect (g, f)-matchings, and therefore prune the corresponding

① $(\langle 2_1, 1_2, 3_3, 4_4, 5_5 \rangle)$
② $(\langle 2_1, 1_2, 4_3, 3_4, 5_5 \rangle)$
③ $(\langle 3_1, 2_2, 1_3, 4_4, 5_5 \rangle)$
④ $(\langle 5_1, 2_2, 3_3, 4_4, 1_5 \rangle)$
⑤ $(\langle 5_1, 2_2, 4_3, 3_4, 1_5 \rangle)$
⑥ $(\langle 5_1, 3_2, 2_3, 4_4, 1_5 \rangle)$

Figure 5.754: All solutions corresponding to the non ground example of the `symmetric_alldifferent_loop` constraint of the **All solutions** slot; the `index` attribute is displayed as indices of the `succ` attribute and self loops are coloured in red.

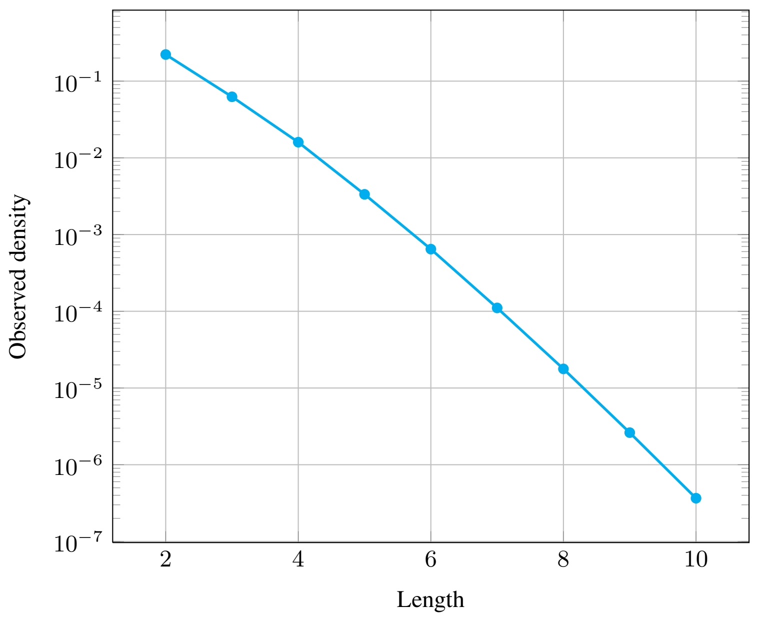
variables.

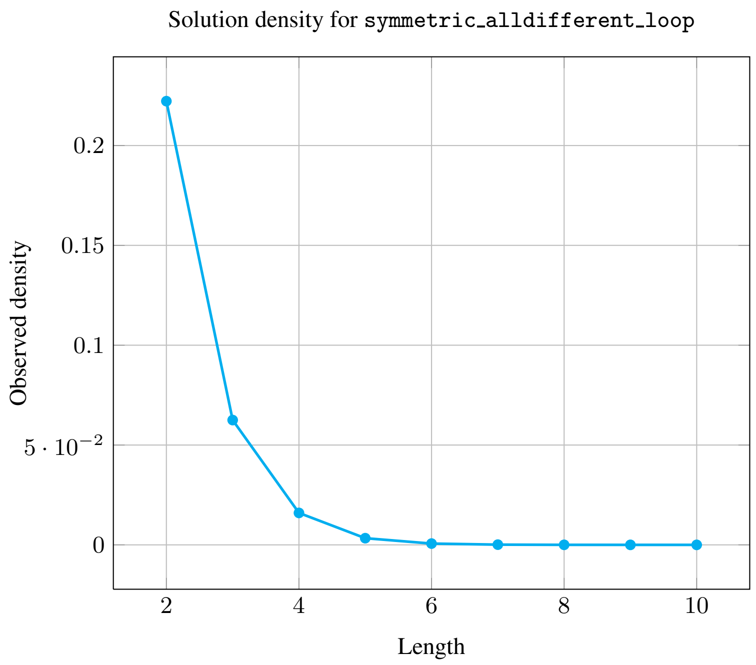
Counting

Length (n)	2	3	4	5	6	7	8	9	10
Solutions	2	4	10	26	76	232	764	2620	9496

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

Solution density for `symmetric_alldifferent_loop`





See also

implied by: `symmetric_alldifferent`.

implies: `twin`.

implies (items to collection): `lex_alldifferent`.

Keywords

characteristic of a constraint: all different, disequality.

combinatorial object: permutation, involution, matching.

constraint type: graph constraint, graph partitioning constraint.

final graph structure: circuit.

modelling: cycle.

Cond. implications

`symmetric_alldifferent_loop`(NODES)
implies `permutation`(VARIABLES : NODES).

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

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 5.755 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.

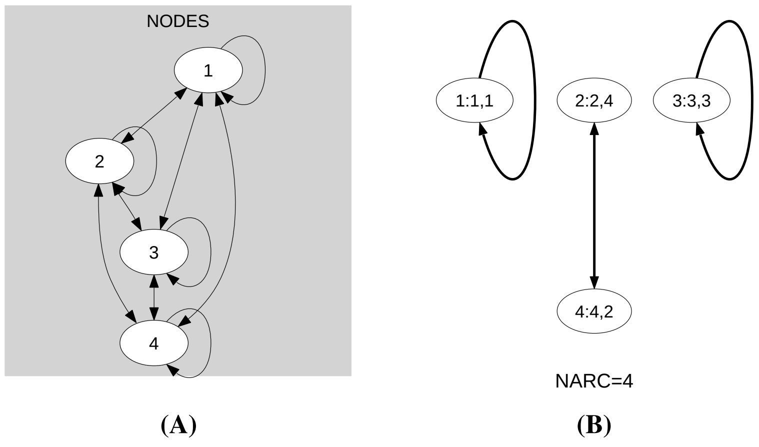


Figure 5.755: Initial and final graph of the `symmetric_alldifferent_loop` constraint

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 $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.