5.66 circuit

DESCRIPTION **LINKS** **GRAPH**

Origin [256]

Constraint `circuit(NODES)`

Synonyms `atour`, `cycle`.

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 Enforce to cover a digraph G described by the `NODES` collection with one circuit visiting once all vertices of G.

Example

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

The `circuit` constraint holds since its `NODES` argument depicts the following Hamiltonian circuit visiting successively the vertices 1, 2, 3, 4 and 1.

All solutions Figure 5.179 gives all solutions to the following non ground instance of the `circuit` constraint: $\mathtt{S}_1 \in [3,4]$, $\mathtt{S}_2 \in [1,2]$, $\mathtt{S}_3 \in [1,4]$, $\mathtt{S}_4 \in [2,4]$, `circuit`$(\langle 1\ \mathtt{S}_1, 2\ \mathtt{S}_2, 3\ \mathtt{S}_3, 4\ \mathtt{S}_4\rangle)$.

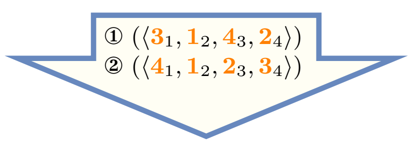


Figure 5.179: All solutions corresponding to the non ground example of the `circuit` constraint of the **All solutions** slot (the `index` attribute is displayed as indices of the `succ` attribute)

Typical `|NODES|` > 2

Symmetry Items of `NODES` are permutable.

Remark

In the original `circuit` constraint of **CHIP** the `index` attribute was not explicitly present. It was implicitly defined as the position of a variable in a list.

Within the context of linear programming [5] this constraint was introduced under the name `atour`. In the same context [215, page 380] provides continuous relaxations of the `circuit` constraint.

Within the KOALOG constraint system this constraint is called `cycle`.

Algorithm

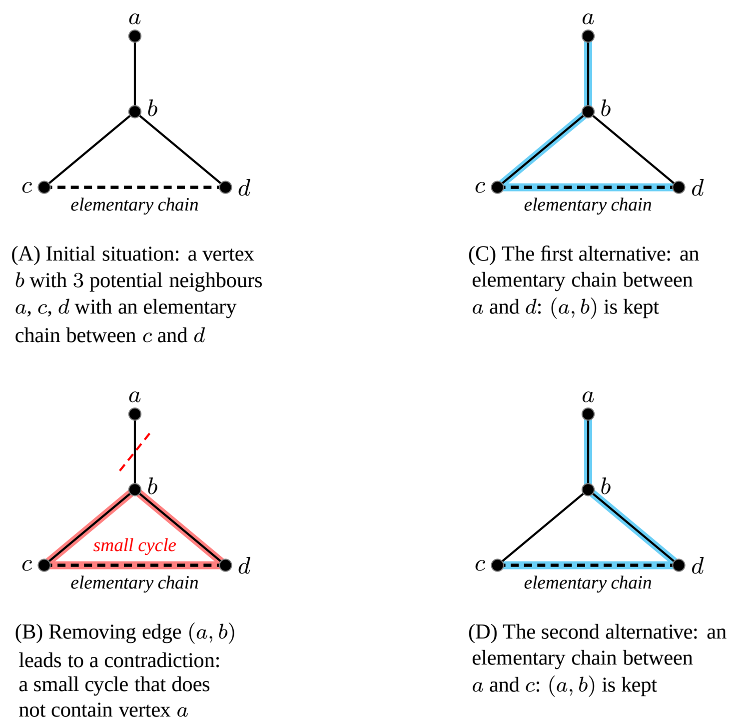
Since all `succ` variables of the `NODES` collection have to take distinct values one can reuse the algorithms associated with the `alldifferent` constraint. A second necessary condition is to have no more than one strongly connected component. Pruning for enforcing this condition can be done by forcing all strong bridges to belong to the final solution, since otherwise the strongly connected component would be broken apart. A third necessary condition is that, if the graph is bipartite then the number of vertices of each class should be identical. Consequently if the number of vertices is odd (i.e., $|\texttt{NODES}|$ is odd) the graph should not be bipartite. Further necessary conditions (useful when the graph is sparse) combining the fact that we have a perfect matching and a single strongly connected component can be found in [381]. These conditions forget about the orientation of the arcs of the graph and characterise new required elementary chains. A typical pattern involving four vertices is depicted by Figure 5.180 where we assume that:

- There is an elementary chain between c and d (depicted by a dashed edge),
- b has exactly 3 neighbours.

In this context the edge between a and b is mandatory in any covering (i.e., the arc from a to b or the arc from b to a) since otherwise a small circuit involving b, c and d would be created.

When the graph is planar [217][138] one can also use as a necessary condition discovered by Grinberg [199] for pruning.

Finally, another approach based an the notion of 1-toughness [116] was proposed in [236] and evaluated for small graphs (i.e., graphs with up to 15 vertices).



(A) Initial situation: a vertex b with 3 potential neighbours a, c, d with an elementary chain between c and d

(C) The first alternative: an elementary chain between a and d: (a, b) is kept

(B) Removing edge (a, b) leads to a contradiction: a small cycle that does not contain vertex a

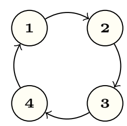
(D) The second alternative: an elementary chain between a and c: (a, b) is kept

Figure 5.180: Reasoning about elementary chains and degrees: if we have an elementary chain between c and d and if b has 3 neighbours then the edge (a, b) is mandatory.

Reformulation Let n and $s_1, s_2, \ldots, s_n$ respectively denotes the number of vertices (i.e., $|\texttt{NODES}|$) and the successor variables associated with vertices $1, 2, \ldots, n$. The `circuit` constraint can be reformulated as a conjunction of one `domain` constraint, two `alldifferent` constraints, and n `element` constraints.

- First, we state an `alldifferent`$(\langle s_1, s_2, \ldots, s_n\rangle)$ constraint for enforcing distinct values to be assigned to the successor variables.
- Second, the key idea is, starting from vertex 1, to successively extract the vertices $t_1, t_2, \ldots, t_{n-1}$ of the circuit until we come back on vertex 1, where t_i (with $i \in [2, n-1]$) denotes the successor of t_{i-1} and t_1 the successor of vertex 1. Since we have one single circuit all the $t_1, t_2, \ldots, t_{n-1}$ should be different from 1. Consequently we state a `domain`$(\langle t_1, t_2, \ldots, t_{n_1}\rangle, 2, n)$ constraint for declaring their initial domains. To express the link between consecutive t_i we also state a conjunction of n `element` constraints of the form:
  `element`$(1, \langle s_1, s_2, \ldots, s_n\rangle, t_1)$,
  `element`$(t_1, \langle s_1, s_2, \ldots, s_n\rangle, t_2)$,
  ....
  `element`$(t_{n-1}, \langle s_1, s_2, \ldots, s_n\rangle, 1)$.
- Finally we add a redundant constraint for stating that all t_i (with $i \in [1, n-1]$) are distinct, i.e. `alldifferent`$(\langle t_1, t_2, \ldots, t_{n-1}\rangle)$.

Illustration of the reformulation of `circuit`$(\langle 1\ 2,\ 2\ 3,\ 3\ 4,\ 4\ 1\rangle)$



`alldifferent`$(\langle 2, 3, 4, 1\rangle)$
`domain`$(\langle 2, 3, 4\rangle, 2, 4)$

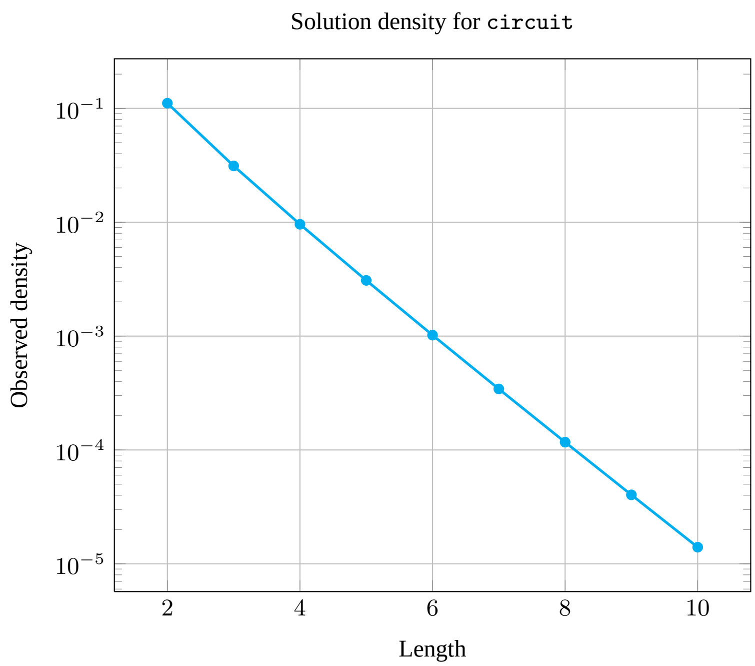
`element`$(1, \langle 2, 3, 4, 1\rangle, 2)$
`element`$(2, \langle 2, 3, 4, 1\rangle, 3)$
`element`$(3, \langle 2, 3, 4, 1\rangle, 4)$
`element`$(4, \langle 2, 3, 4, 1\rangle, 1)$

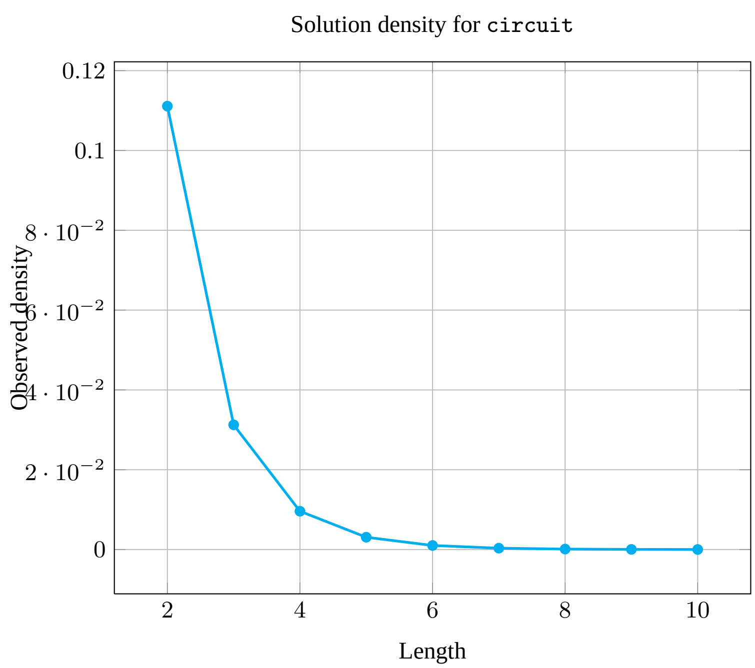
`alldifferent`$(\langle 2, 3, 4\rangle)$

Counting

Length (n)	2	3	4	5	6	7	8	9	10
Solutions	1	2	6	24	120	720	5040	40320	362880

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





Systems `circuit` in **Gecode**, `circuit` in **JaCoP**, `circuit` in **SICStus**.

See also **common keyword:** `alldifferent` (*permutation*), `circuit_cluster` (*graph constraint*, *one_succ*), `path` (*graph partitioning constraint*, *one_succ*), `proper_circuit` (*permutation*, *one_succ*), `tour` (*graph partitioning constraint*, *Hamiltonian*).

generalisation: `cycle` (*introduce a variable for the number of circuits*).

implies: `alldifferent`, `proper_circuit`, `twin`.

implies (items to collection): `lex_alldifferent`.

related: `strongly_connected`.

Keywords

combinatorial object: permutation.

constraint type: graph constraint, graph partitioning constraint.

filtering: linear programming, planarity test, strong bridge, DFS-bottleneck.

final graph structure: circuit, one_succ.

problems: Hamiltonian.

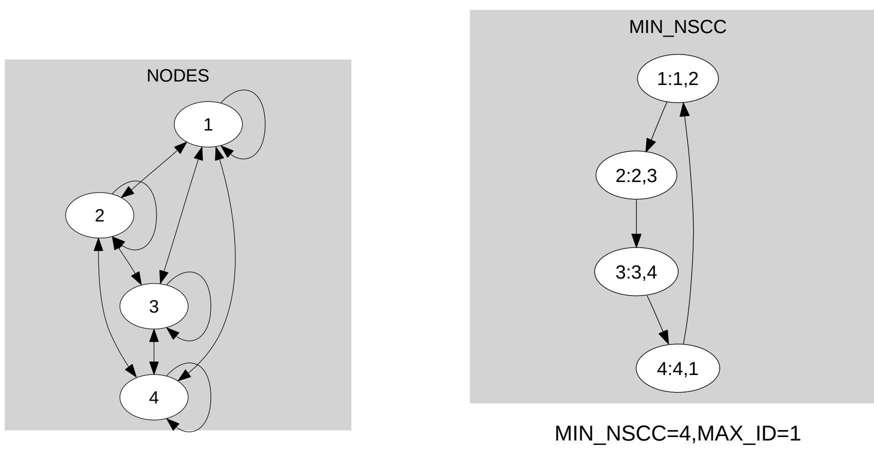
Cond. implications

- `circuit`(NODES)
  implies `cycle`(NCYCLE, NODES)
  when NCYCLE $= 1$.
- `circuit`(NODES)
  with $|$NODES$| > 1$
  implies `derangement`(NODES).
- `circuit`(NODES)
  with $|$NODES$| > 1$
  implies `k_alldifferent`(VARS : NODES).
- `circuit`(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`
Graph property(ies)	• **MIN_NSCC**= |`NODES`| • **MAX_ID**$\leq$ 1
Graph class	`ONE_SUCC`

Graph model The first graph property enforces to have a single strongly connected component containing |`NODES`| vertices. The second graph property imposes to only have circuits. Since each vertex of the final graph has only one successor we do not need to use set variables for representing the successors of a vertex.

Parts (A) and (B) of Figure 5.181 respectively show the initial and final graph associated with the **Example** slot. The `circuit` constraint holds since the final graph consists of one circuit mentioning once every vertex of the initial graph.



(A) **(B)**

Figure 5.181: Initial and final graph of the `circuit` constraint