5.53 circuit

DESCRIPTION LINKS GRAPH

Origin [229]

Constraint circuit(NODES)

Synonyms atour, cycle.

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

Restrictions
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index ≤ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ ≤ |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.

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 [4] this constraint was introduced under the name atour. In the same context [190, 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. When the number of

vertices is odd (i.e., |NODES| is odd) a third necessary condition is to have a bipartite graph (see the **Algorithm** slot of the `bipartite` constraint).

Further necessary conditions (useful when the graph is sparse) combining the fact that we have a perfect matching and one single strongly connected component can be found in [347]. 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 5.116 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.

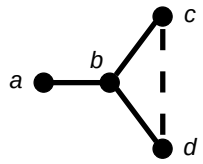

Figure 5.116: Reasoning about elementary chains and degrees: if we have an elementary chain between c and d and if b has 3 neighbours then the edge (a, b) is mandatory.

When the graph is planar [192][122] one can also use as a necessary condition discovered by Grinberg [177] for pruning.

Finally, another approach based an the notion of 1-toughness [105] was proposed in [209] and evaluated for small graphs (i.e., graphs with up to 15 vertices).

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)*, `tour` *(graph partitioning constraint, Hamiltonian)*.

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

implies: `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.

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= 1
Graph class	ONE_SUCC

Graph model The first graph property enforces to have one 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 5.117 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.

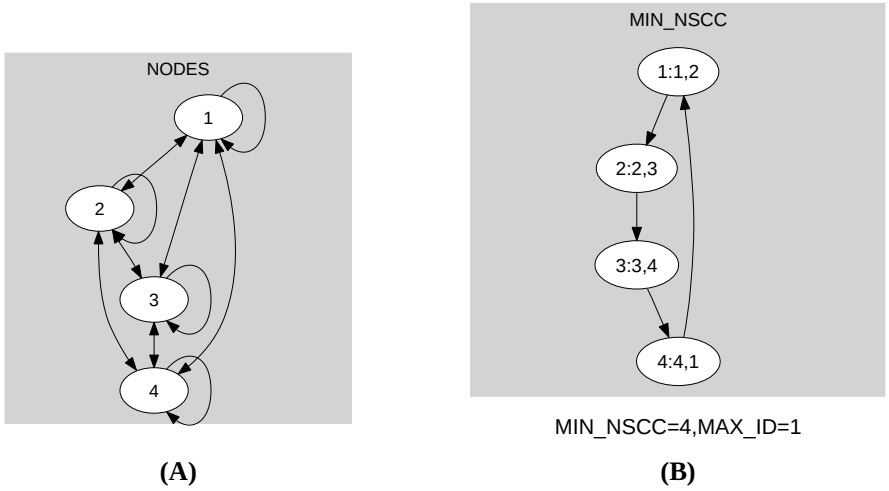

Figure 5.117: Initial and final graph of the `circuit` constraint

Signature Since the initial graph contains |NODES| vertices the final graph contains at most |NODES| vertices. Therefore we can rewrite the graph property $\mathbf{MIN\_NSCC} = |\text{NODES}|$ to $\mathbf{MIN\_NSCC} \geq |\text{NODES}|$. This leads to simplify $\overline{\text{MIN\_NSCC}}$ to $\overline{\mathbf{MIN\_NSCC}}$.

Because of the graph property $\mathbf{MIN\_NSCC} = |\text{NODES}|$ the final graph contains at least one vertex. Since a vertex v belongs to the final graph only if there is an arc that has v as one of its extremities the final graph contains at least one arc. Therefore $\mathbf{MAX\_ID}$ is greater than or equal to 1. So we can rewrite the graph property $\mathbf{MAX\_ID} = 1$ to $\mathbf{MAX\_ID} \leq 1$. This leads to simplify $\overline{\text{MAX\_ID}}$ to $\overline{\mathbf{MAX\_ID}}$.