5.89 cycle

DESCRIPTION LINKS GRAPH

Origin [37]

Constraint cycle(NCYCLE, NODES)

Arguments

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

Restrictions

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

Purpose Consider a digraph G described by the NODES collection. NCYCLE is equal to the number of circuits for covering G in such a way that each vertex of G belongs to one single circuit. NCYCLE can also be interpreted as the number of cycles of the permutation associated with the successor variables of the NODES collection.

Example

$$\left\langle \begin{array}{c} 2, \left\langle \begin{array}{cc} \mathtt{index}-1 & \mathtt{succ}-2, \\ \mathtt{index}-2 & \mathtt{succ}-1, \\ \mathtt{index}-3 & \mathtt{succ}-5, \\ \mathtt{index}-4 & \mathtt{succ}-3, \\ \mathtt{index}-5 & \mathtt{succ}-4 \end{array} \right\rangle \end{array} \right\rangle$$

In this example we have the following 2 (NCYCLE = 2) cycles: $1 \rightarrow 2 \rightarrow 1$ and $3 \rightarrow 5 \rightarrow 4 \rightarrow 3$. Consequently, the cycle constraint holds.

Typical

|NODES| > 2
NCYCLE < |NODES|

Symmetry Items of NODES are permutable.

Usage The PhD thesis of Éric Bourreau [76] mentions the following applications of extensions of the cycle constraint:

- The balanced Euler knight problem where one tries to cover a rectangular chessboard of size $N \cdot M$ by C knights that all have to visit between $2 \cdot \lfloor \lfloor (N \cdot M)/C \rfloor /2 \rfloor$ and $2 \cdot \lceil \lceil (N \cdot M)/C \rceil /2 \rceil$ distinct locations. For some values of N, M and C there does not exist any solution to the previous problem. This is for instance the case when $N = M = C = 6$. Figure 5.179 depicts the graph associated with the 6×6 chessboard as well as examples of balanced solutions with respectively 1, 2, 3, 4 and 5 knights.

- Some pick-up delivery problems where a fleet of vehicles has to transport a set of orders. Each order is characterised by its initial location, its final destination and its weight. In addition one also has to take into account the capacity of the different vehicles.

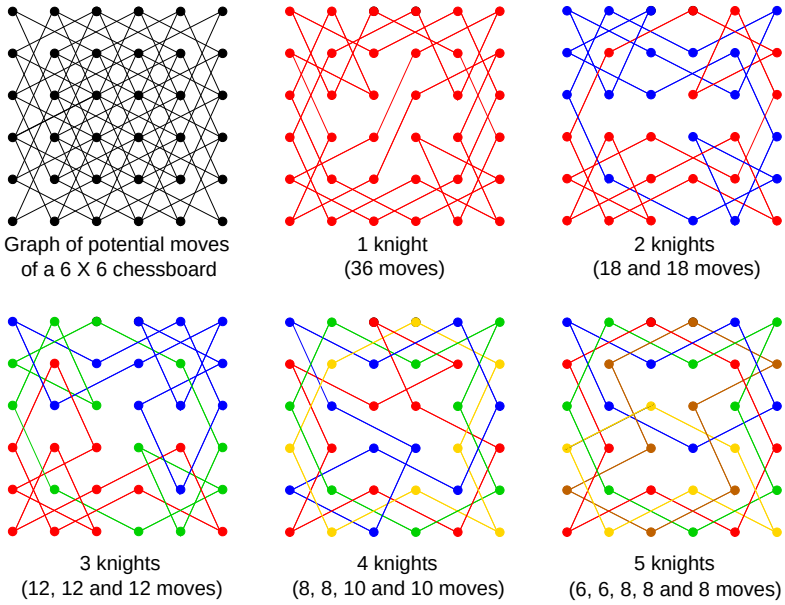

Figure 5.179: Graph of potential moves of the 6×6 chessboard and corresponding balanced tours

Remark

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

In an early version of the **CHIP** there was a constraint named `circuit` that, from a declarative point of view, was equivalent to `cycle`$(1, \texttt{NODES})$. In ALICE [229] the `circuit` constraint was also present.

Given a complete digraph of n vertices as well as an unrestricted number of circuits `NCYCLE`, the total number of solutions of the corresponding `cycle` constraint corresponds to the sequence A000142 of the On-Line Encyclopedia of Integer Sequences [357]. Given a complete digraph of n vertices as well as a fixed number of circuits `NCYCLE` between 1 and n, the total number of solutions of the corresponding `cycle` constraint corresponds to the so called *Stirling number of first kind*.

Algorithm

Since all `succ` variables 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 $\overline{\texttt{NCYCLE}}$ strongly connected components. Pruning for enforcing this condition, as soon as we have $\overline{\texttt{NCYCLE}}$ strongly connected components, can be done by forcing all strong bridges to belong to the final solution, since otherwise we would have more than $\overline{\texttt{NCYCLE}}$ strongly connected components. Since all the vertices of a circuit belong to the same strongly

connected component an arc going from one strongly connected component to another strongly connected component has to be removed.

See also

common keyword: `alldifferent` *(permutation)*,
`circuit_cluster` *(graph constraint, one_succ)*,
`cycle_card_on_path` *(permutation,graph partitioning constraint)*,
`cycle_or_accessibility` *(graph constraint)*,
`cycle_resource` *(graph partitioning constraint)*,
`derangement` *(permutation)*,
`graph_crossing` *(graph constraint,graph partitioning constraint)*,
`inverse` *(permutation)*,
`map` *(graph partitioning constraint)*,
`symmetric_alldifferent` *(permutation)*,
`tour` *(graph constraint)*,
`tree` *(graph partitioning constraint)*.

implies: `alldifferent`.

specialisation: `circuit` *(*`NCYCLE` *set to* 1*)*.

Keywords

characteristic of a constraint: core.

combinatorial object: permutation.

constraint arguments: business rules.

constraint type: graph constraint, graph partitioning constraint.

filtering: strong bridge, DFS-bottleneck.

final graph structure: circuit, connected component, strongly connected component, one_succ.

modelling: cycle.

problems: pick-up delivery.

puzzles: Euler knight.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	• **NTREE** = 0 • **NCC** = NCYCLE
Graph class	ONE_SUCC

Graph model From the restrictions and from the arc constraint, we deduce that we have a bijection from the successor variables to the values of interval $[1, |\texttt{NODES}|]$. With no explicit restrictions it would have been impossible to derive this property.

In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the `cycle` constraint considers objects that have two attributes:

- One fixed attribute `index` that is the identifier of the vertex,
- One variable attribute `succ` that is the successor of the vertex.

The graph property **NTREE** $= 0$ is used in order to avoid having vertices that both do not belong to a circuit and have at least one successor located on a circuit. This concretely means that all vertices of the final graph should belong to a circuit.

Parts (A) and (B) of Figure 5.180 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NCC** graph property, we show the two connected components of the final graph. The constraint holds since all the vertices belong to a circuit (i.e., **NTREE** $= 0$) and since NCYCLE $=$ **NCC** $= 2$.

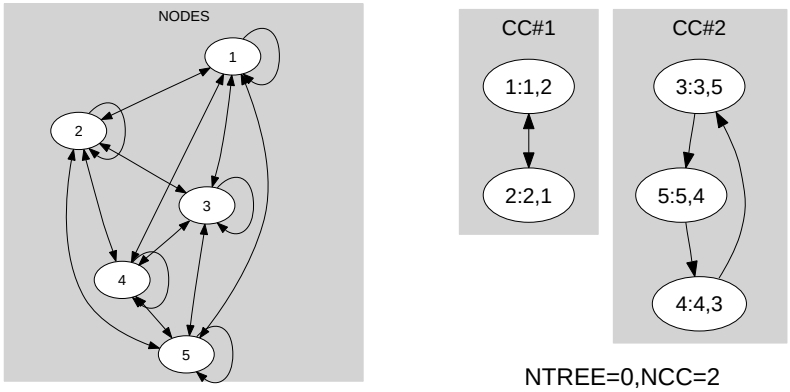

Figure 5.180: Initial and final graph of the `cycle` constraint