5.106 cycle_resource

DESCRIPTION LINKS GRAPH

Origin CHIP

Constraint cycle_resource(RESOURCE, TASK)

Arguments

RESOURCE : collection(id−int, first_task−dvar, nb_task−dvar)
TASK : collection(id−int, next_task−dvar, resource−dvar)

Restrictions

required(RESOURCE, [id, first_task, nb_task])
RESOURCE.id ≥ 1
RESOURCE.id $\leq$ |RESOURCE|
distinct(RESOURCE, id)
RESOURCE.first_task ≥ 1
RESOURCE.first_task $\leq$ |RESOURCE| + |TASK|
RESOURCE.nb_task ≥ 0
RESOURCE.nb_task $\leq$ |TASK|
required(TASK, [id, next_task, resource])
TASK.id $>$ |RESOURCE|
TASK.id $\leq$ |RESOURCE| + |TASK|
distinct(TASK, id)
TASK.next_task ≥ 1
TASK.next_task $\leq$ |RESOURCE| + |TASK|
TASK.resource ≥ 1
TASK.resource $\leq$ |RESOURCE|

Purpose

Consider a digraph G defined as follows:

- To each item of the RESOURCE and TASK collections corresponds one vertex of G. A vertex that was generated from an item of the RESOURCE (respectively TASK) collection is called a *resource* vertex (respectively *task* vertex).
- There is an arc from a resource vertex r to a task vertex t if $t \in$ RESOURCE[r].first_task.
- There is an arc from a task vertex t to a resource vertex r if $r \in$ TASK[t].next_task.
- There is an arc from a task vertex t_1 to a task vertex t_2 if $t_2 \in$ TASK[t_1].next_task.
- There is no arc between two resource vertices.

Enforce to cover G in such a way that each vertex belongs to a single circuit. Each circuit is made up from a single *resource* vertex and zero, one or more *task* vertices. For each resource-vertex a domain variable indicates how many task-vertices belong to the corresponding circuit. For each task a domain variable provides the identifier of the resource that can effectively handle that task.

Example

$$\left(\begin{array}{l}\left\langle\begin{array}{lll}\texttt{id}-1 & \texttt{first\_task}-5 & \texttt{nb\_task}-3,\\ \texttt{id}-2 & \texttt{first\_task}-2 & \texttt{nb\_task}-0,\\ \texttt{id}-3 & \texttt{first\_task}-8 & \texttt{nb\_task}-2\end{array}\right\rangle,\\ \left\langle\begin{array}{lll}\texttt{id}-4 & \texttt{next\_task}-7 & \texttt{resource}-1,\\ \texttt{id}-5 & \texttt{next\_task}-4 & \texttt{resource}-1,\\ \texttt{id}-6 & \texttt{next\_task}-3 & \texttt{resource}-3,\\ \texttt{id}-7 & \texttt{next\_task}-1 & \texttt{resource}-1,\\ \texttt{id}-8 & \texttt{next\_task}-6 & \texttt{resource}-3\end{array}\right\rangle\end{array}\right)$$

The `cycle_resource` constraint holds since the graph corresponding to the vertices described by its arguments consists of the following 3 disjoint circuits:

- The first circuit involves the *resource* vertex 1 as well as the *task* vertices 5, 4 and 7.
- The second circuit is limited to the *resource* vertex 2.
- Finally the third circuit is made up from the remaining vertices, namely the *resource* vertex 3 and the *task* vertices 8 and 6.

Typical

$|\texttt{RESOURCE}| > 1$
$|\texttt{TASK}| > 1$
$|\texttt{TASK}| > |\texttt{RESOURCE}|$

Symmetries

- Items of `RESOURCE` are permutable.
- Items of `TASK` are permutable.
- All occurrences of two distinct values in `RESOURCE.id` or `TASK.resource` can be swapped.

Usage

This constraint is useful for some vehicles routing problem where the number of locations to visit depends of the vehicle type that is actually used. The resource attribute allows expressing various constraints such as:

- The compatibility or incompatibility between tasks and vehicles,
- The fact that certain tasks should be performed by the same vehicle,
- The pre-assignment of certain tasks to a given vehicle.

Remark

This constraint could be expressed with the `cycle` constraint of **CHIP** by using the following optional parameters:

- The *resource node* parameter [84, page 97],
- The *circuit weight* parameter [84, page 101],
- The *name* parameter [84, page 104].

See also

common keyword: `cycle` *(graph partitioning constraint)*.

Keywords

characteristic of a constraint: derived collection.

constraint type: graph constraint, resource constraint, graph partitioning constraint.

final graph structure: connected component, strongly connected component.

Derived Collection

$$\texttt{col}\left(\begin{array}{l}\texttt{RESOURCE\_TASK}-\texttt{collection}\left(\begin{array}{l}\texttt{index}-\texttt{int},\\ \texttt{succ}-\texttt{dvar},\\ \texttt{name}-\texttt{dvar}\end{array}\right),\\ \left[\begin{array}{l}\texttt{item}\left(\begin{array}{l}\texttt{index}-\texttt{RESOURCE.id},\\ \texttt{succ}-\texttt{RESOURCE.first\_task},\\ \texttt{name}-\texttt{RESOURCE.id}\end{array}\right),\\ \texttt{item}\left(\begin{array}{l}\texttt{index}-\texttt{TASK.id},\\ \texttt{succ}-\texttt{TASK.next\_task},\\ \texttt{name}-\texttt{TASK.resource}\end{array}\right)\end{array}\right]\end{array}\right)$$

Arc input(s) RESOURCE_TASK

Arc generator $CLIQUE \mapsto$ collection(resource_task1, resource_task2)

Arc arity 2

Arc constraint(s)
- resource_task1.succ = resource_task2.index
- resource_task1.name = resource_task2.name

Graph property(ies)
- **NTREE** $= 0$
- **NCC** $=$ |RESOURCE|
- **NVERTEX** $=$ |RESOURCE| + |TASK|

Graph class ONE_SUCC

For all items of RESOURCE:

Arc input(s) RESOURCE_TASK

Arc generator $CLIQUE \mapsto$ collection(resource_task1, resource_task2)

Arc arity 2

Arc constraint(s)
- resource_task1.succ = resource_task2.index
- resource_task1.name = resource_task2.name
- resource_task1.name = RESOURCE.id

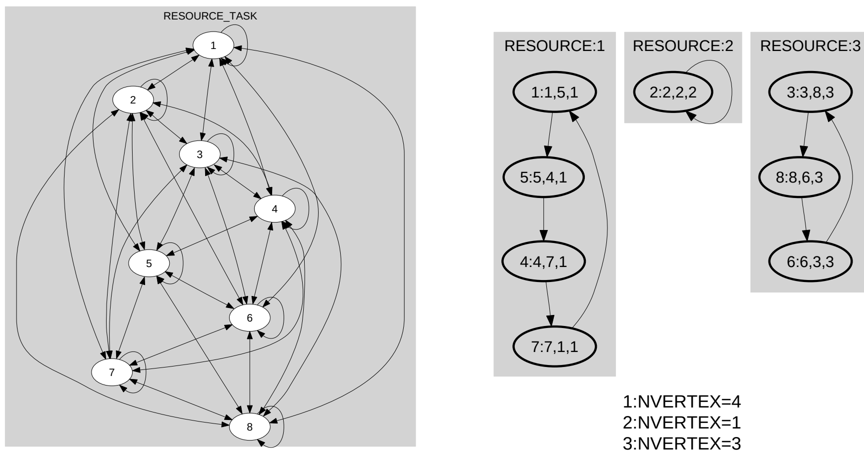
Graph property(ies) **NVERTEX** $=$ RESOURCE.nb_task $+ 1$

Graph model The graph model of the `cycle_resource` constraint illustrates the following points:

- How to differentiate the constraint on the length of a circuit according to a resource that is assigned to a circuit? This is achieved by introducing a collection of resources and by asking a different graph property for each item of that collection.
- How to introduce the concept of name that corresponds to the resource that handles a given task? This is done by adding to the arc constraint associated with the `cycle` constraint the condition that the name variables of two consecutive vertices should be equal.

Part (A) of Figure 5.234 shows the initial graphs (of the second graph constraint) associated with resources 1, 2 and 3 of the **Example** slot. Part (B) of Figure 5.234 shows the corresponding final graphs (of the second graph constraint) associated with resources 1, 2 and 3.

Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold. To each resource corresponds a circuit of respectively 3, 0 and 2 task-vertices.



(A) **(B)**

Figure 5.234: Initial and final graph of the `cycle_resource` constraint

Signature Since the initial graph of the first graph constraint contains $|\mathtt{RESOURCE}| + |\mathtt{TASK}|$ vertices, the corresponding final graph cannot have more than $|\mathtt{RESOURCE}| + |\mathtt{TASK}|$ vertices. Therefore we can rewrite the graph property $\mathbf{NVERTEX} = |\mathtt{RESOURCE}| + |\mathtt{TASK}|$ to $\mathbf{NVERTEX} \geq |\mathtt{RESOURCE}| + |\mathtt{TASK}|$ and simplify $\underline{\overline{\mathbf{NVERTEX}}}$ to $\overline{\mathbf{NVERTEX}}$.