5.316 path

DESCRIPTION LINKS GRAPH

Origin

Derived from `binary_tree`.

Constraint

`path`(`NPATH`, `NODES`)

Arguments

`NPATH` : `dvar`
`NODES` : `collection`(`index`−`int`, `succ`−`dvar`)

Restrictions

`NPATH` ≥ 1
`NPATH` $\leq |$`NODES`$|$
`required`(`NODES`, [`index`, `succ`])
$|$`NODES`$| > 0$
`NODES.index` ≥ 1
`NODES.index` $\leq |$`NODES`$|$
`distinct`(`NODES`, `index`)
`NODES.succ` ≥ 1
`NODES.succ` $\leq |$`NODES`$|$

Purpose

Cover the digraph G described by the `NODES` collection with `NPATH` paths in such a way that each vertex of G belongs to exactly one path.

Example

$$\left(3, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-1, \\ \texttt{index}-2 & \texttt{succ}-3, \\ \texttt{index}-3 & \texttt{succ}-5, \\ \texttt{index}-4 & \texttt{succ}-7, \\ \texttt{index}-5 & \texttt{succ}-1, \\ \texttt{index}-6 & \texttt{succ}-6, \\ \texttt{index}-7 & \texttt{succ}-7, \\ \texttt{index}-8 & \texttt{succ}-6 \end{array} \right\rangle \right)$$

$$\left(1, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-8, \\ \texttt{index}-2 & \texttt{succ}-7, \\ \texttt{index}-3 & \texttt{succ}-6, \\ \texttt{index}-4 & \texttt{succ}-5, \\ \texttt{index}-5 & \texttt{succ}-5, \\ \texttt{index}-6 & \texttt{succ}-4, \\ \texttt{index}-7 & \texttt{succ}-3, \\ \texttt{index}-8 & \texttt{succ}-2 \end{array} \right\rangle \right)$$

$$\left(8, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-1, \\ \texttt{index}-2 & \texttt{succ}-2, \\ \texttt{index}-3 & \texttt{succ}-3, \\ \texttt{index}-4 & \texttt{succ}-4, \\ \texttt{index}-5 & \texttt{succ}-5, \\ \texttt{index}-6 & \texttt{succ}-6, \\ \texttt{index}-7 & \texttt{succ}-7, \\ \texttt{index}-8 & \texttt{succ}-8 \end{array} \right\rangle \right)$$

The first `path` constraint holds since its second argument corresponds to the 3 (i.e., the first argument of the `path` constraint) paths depicted by Figure 5.657.

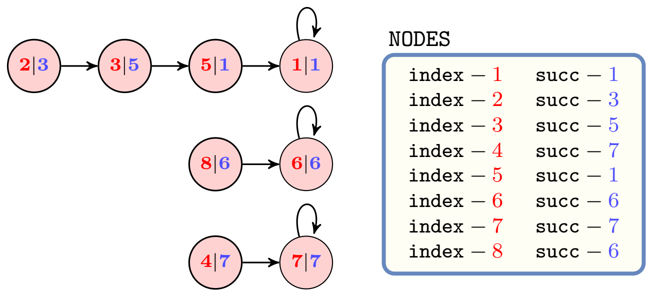


Figure 5.657: The three paths corresponding to the first example of the **Example** slot; each vertex contains the information `index`|`succ` where `succ` is the index of its successor in the path (by convention one of the extremities of a path points to itself).

Typical

$\texttt{NPATH} < |\texttt{NODES}|$
$|\texttt{NODES}| > 1$

Symmetry

Items of `NODES` are permutable.

Arg. properties

Functional dependency: `NPATH` determined by `NODES`.

Reformulation

The `path` constraint can be expressed in term of (1) a set of $|\texttt{NODES}|^2$ reified constraints for avoiding circuit between more than one node and of (2) $|\texttt{NODES}|$ reified constraints and of one sum constraint for counting the paths and of (3) a set of $|\texttt{NODES}|^2$ reified constraints and of $|\texttt{NODES}|$ inequalities constraints for enforcing the fact that each vertex has at most two children.

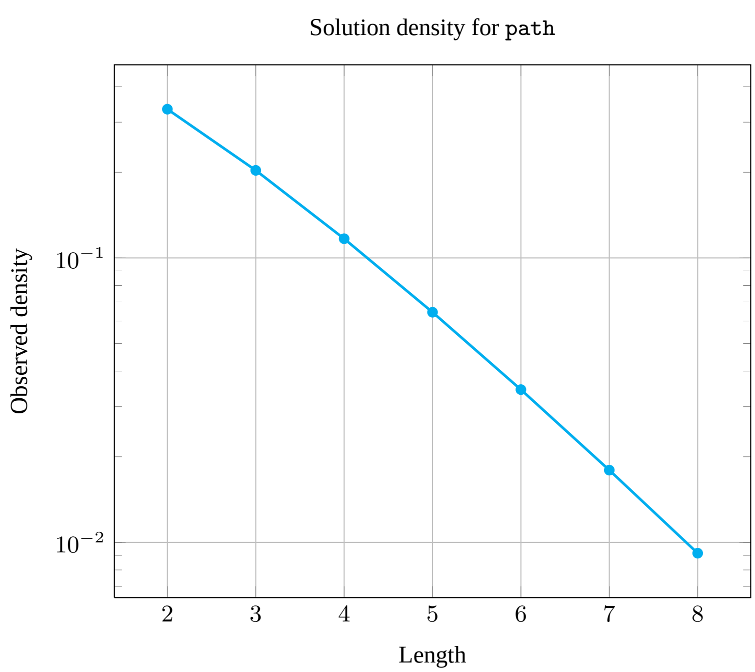
1. For each vertex $\texttt{NODES}[i]$ ($i \in [1, |\texttt{NODES}|]$) of the `NODES` collection we create a variable R_i that takes its value within interval $[1, |\texttt{NODES}|]$. This variable represents the *rank* of vertex $\texttt{NODES}[i]$ within a solution. It is used to prevent the creation of circuit involving more than one vertex as explained now. For each pair of vertices $\texttt{NODES}[i], \texttt{NODES}[j]$ ($i, j \in [1, |\texttt{NODES}|]$) of the `NODES` collection we create a reified constraint of the form $\texttt{NODES}[i].\texttt{succ} = \texttt{NODES}[j].\texttt{index} \wedge i \neq j \Rightarrow R_i < R_j$. The purpose of this constraint is to express the fact that, if there is an arc from vertex $\texttt{NODES}[i]$ to another vertex $\texttt{NODES}[j]$, then R_i should be strictly less than R_j.
2. For each vertex $\texttt{NODES}[i]$ ($i \in [1, |\texttt{NODES}|]$) of the `NODES` collection we create a 0-1 variable B_i and state the following reified constraint $\texttt{NODES}[i].\texttt{succ} = \texttt{NODES}[i].\texttt{index} \Leftrightarrow B_i$ in order to force variable B_i to be set to value 1 if and only if there is a loop on vertex $\texttt{NODES}[i]$. Finally we create a constraint $\texttt{NPATH} = B_1 + B_2 + \cdots + B_{|\texttt{NODES}|}$ for stating the fact that the number of paths is equal to the number of loops of the graph.
3. For each pair of vertices $\texttt{NODES}[i], \texttt{NODES}[j]$ ($i, j \in [1, |\texttt{NODES}|]$) of the `NODES` collection we create a 0-1 variable B_{ij} and state the following reified constraint

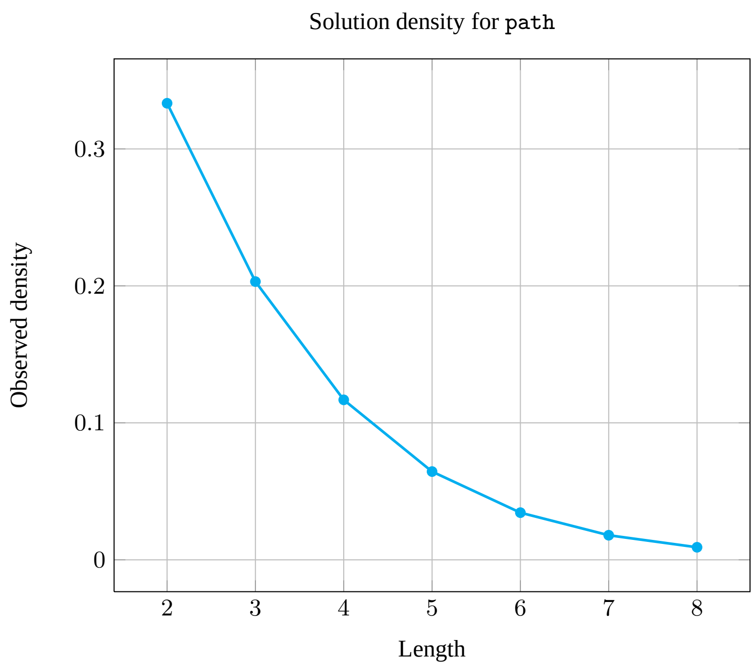
$\texttt{NODES}[i].\texttt{succ} = \texttt{NODES}[j].\texttt{index} \wedge i \neq j \Leftrightarrow B_{ij}$. Variable B_{ij} is set to value 1 if and only if there is an arc from $\texttt{NODES}[i]$ to $\texttt{NODES}[j]$. Then for each vertex $\texttt{NODES}[j]$ ($j \in [1, |\texttt{NODES}|]$) we create a constraint of the form $B_{1j} + B_{2j} + \cdots + B_{|\texttt{NODES}|j} \leq 1$.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	3	13	73	501	4051	37633	394353

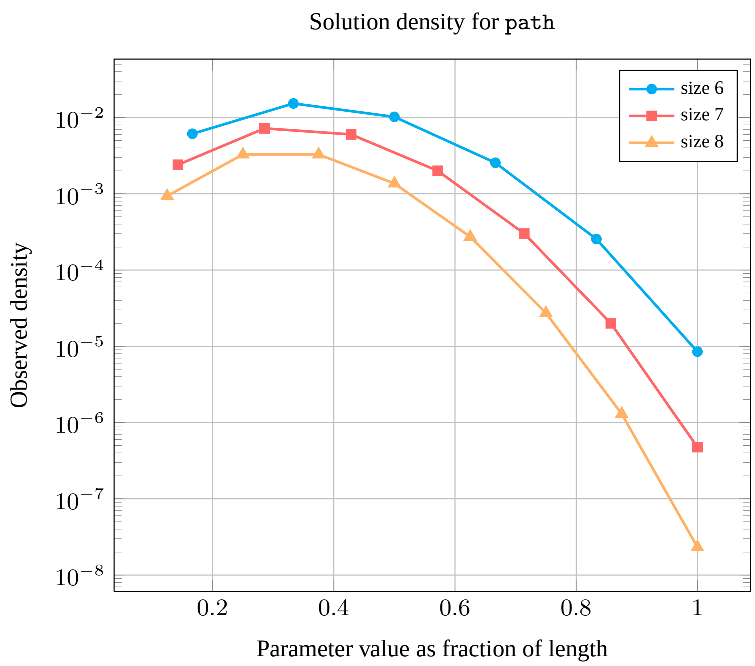
Number of solutions for `path`: domains $0..n$

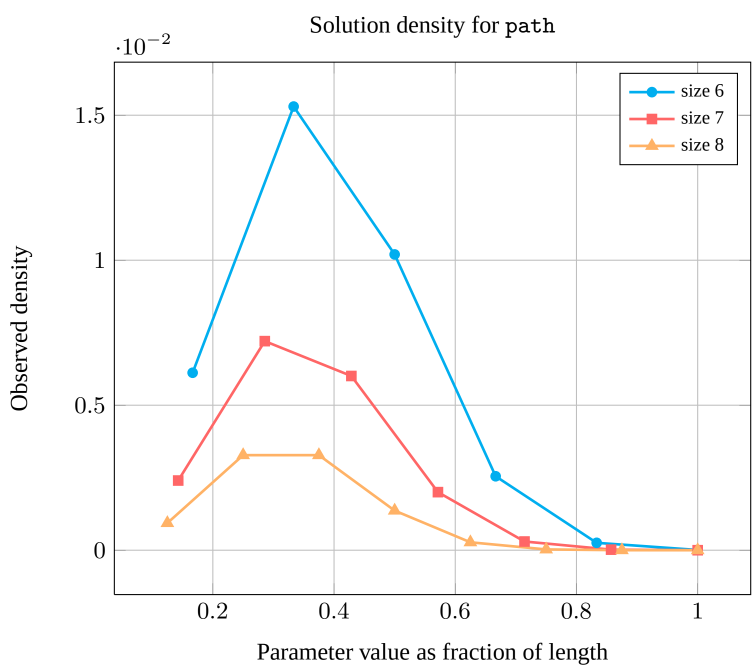




Length (n)		2	3	4	5	6	7	8
Total		3	13	73	501	4051	37633	394353
Parameter value	1	2	6	24	120	720	5040	40320
	2	1	6	36	240	1800	15120	141120
	3	-	1	12	120	1200	12600	141120
	4	-	-	1	20	300	4200	58800
	5	-	-	-	1	30	630	11760
	6	-	-	-	-	1	42	1176
	7	-	-	-	-	-	1	56
	8	-	-	-	-	-	-	1

Solution count for `path`: domains $0..n$





See also **common keyword:** `circuit` *(graph partitioning constraint, one_succ)*, `dom_reachability` *(path)*, `path_from_to` *(path, select an induced subgraph so that there is a path from a given vertex to an other given vertex)*, proper_circuit *(graph partitioning constraint, one_succ).*

generalisation: binary_tree *(at most one child replaced by at most two children)*, temporal_path *(vertices are located in time, and to each arc corresponds a precedence constraint)*, tree *(at most one child replaced by no limit on the number of children)*.

implies: binary_tree.

related: balance_path *(counting number of paths versus controlling how balanced the paths are)*.

Keywords

combinatorial object: path.

constraint type: graph constraint, graph partitioning constraint.

filtering: DFS-bottleneck.

final graph structure: connected component, tree, one_succ.

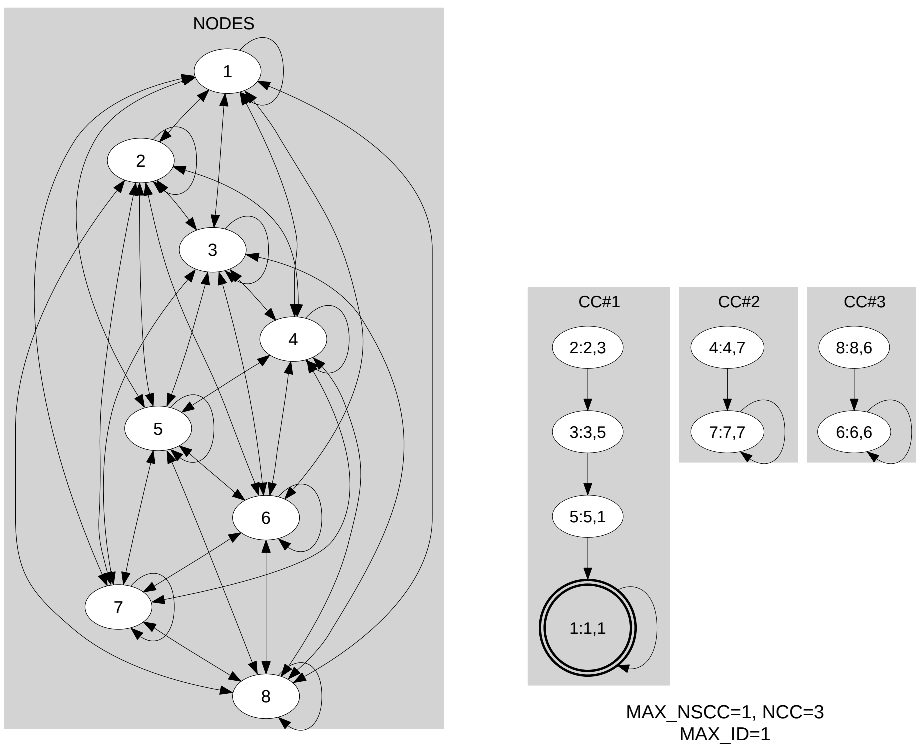
modelling: functional dependency.

Arc input(s)	NODES
Arc generator	*CLIQUE* $\mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	• **MAX_NSCC** ≤ 1 • **NCC** = NPATH • **MAX_ID** ≤ 1
Graph class	ONE_SUCC

Graph model We use the same graph constraint as for the binary_tree constraint, except that we replace the graph property **MAX_ID** ≤ 2, which constraints the maximum in-degree of the final graph to not exceed 2 by **MAX_ID** ≤ 1. **MAX_ID** does not consider loops: This is why we do not have any problem with the final node of each path.

Parts (A) and (B) of Figure 5.658 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NCC** graph property, we display the three connected components of the final graph. Each of them corresponds to a path. Since we use the **MAX_ID** graph property, we also show with a double circle a vertex that has a maximum number of predecessors.

The path constraint holds since all strongly connected components of the final graph have no more than one vertex, since NPATH = **NCC** $= 3$ and since **MAX_ID** $= 1$.



(A) **(B)**

Figure 5.658: Initial and final graph of the `path` constraint