5.403 tree

DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint tree(NTREES, NODES)

Arguments

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

Restrictions

NTREES ≥ 1
NTREES $\leq |$NODES$|$
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq |$NODES$|$
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq |$NODES$|$

Purpose Given a digraph G described by the NODES collection, cover G by a set of NTREES trees in such a way that each vertex of G belongs to one distinct tree. The edges of the trees are directed from their leaves to their respective roots.

Example

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

The first `tree` constraint holds since the graph associated with the items of the `NODES` collection corresponds to two trees (i.e., `NTREES` $= 2$): each tree respectively involves the vertices $\{1, 2, 3, 5, 6, 8\}$ and $\{4, 7\}$. They are depicted by Figure 5.763.

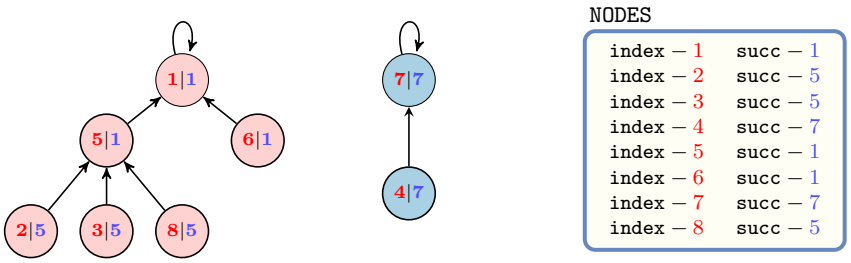


Figure 5.763: The two trees corresponding to the first example of the **Example** slot; each vertex contains the information `index`|`succ` where `succ` is the index of its father in the tree (by convention the father of the root is the root itself).

All solutions

Figure 5.764 gives all solutions to the following non ground instance of the `tree` constraint: `NTREES` $\in [3, 4]$, $S_1 \in [1, 2]$, $S_2 \in [1, 3]$, $S_3 \in [1, 4]$, $S_4 \in [2, 4]$, `tree`(`NTREES`, $\langle 1\ S_1, 2\ S_2, 3\ S_3, 4\ S_4 \rangle$).

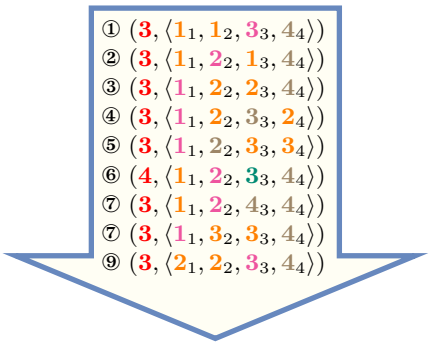


Figure 5.764: All solutions corresponding to the non ground example of the `tree` constraint of the **All solutions** slot (the `index` attribute is displayed as indices of the `succ` attribute)

Typical

`NTREES` $<$ |`NODES`|
|`NODES`| > 2

Symmetry

Items of `NODES` are permutable.

Arg. properties

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

Remark

Given a complete digraph of n vertices as well as an unrestricted number of trees `NTREES`, the total number of solutions to the corresponding `tree` constraint corresponds to the sequence A000272 of the On-Line Encyclopaedia of Integer Sequences [392].

Extension of the tree constraint to the *minimum spanning tree* constraint is described in [143, 349, 352].

Algorithm

An arc-consistency filtering algorithm for the tree constraint is described in [42]. This algorithm is based on a necessary and sufficient condition that we now depict.

To any tree constraint we associate the digraph $G = (V, E)$, where:

- To each item NODES$[i]$ of the NODES collection corresponds a vertex v_i of G.
- For every pair of items (NODES$[i]$, NODES$[j]$) of the NODES collection, where i and j are not necessarily distinct, there is an arc from v_i to v_j in E if and only if j is a potential value of NODES$[i]$.succ.

A strongly connected component C of G is called a *sink component* if all the successors of all vertices of C belong to C. Let MINTREES and MAXTREES respectively denote the number of sink components of G and the number of vertices of G with a loop.

The tree constraint has a solution if and only if:

- Each sink component of G contains at least one vertex with a loop,
- The domain of NTREES has at least one value within interval [MINTREES, MAXTREES].

Inspired by the idea of using dominators used in [223] for getting a linear time algorithm for computing strong articulation points of a digraph G, the worst case complexity of the algorithm proposed in [42] was also enhanced in a similar way by J.-G. Fages and X. Lorca [157].

Reformulation

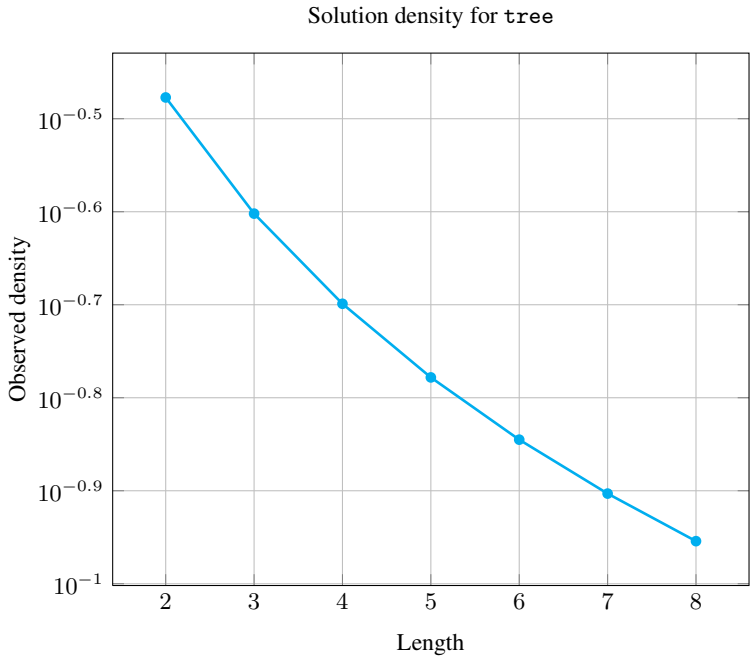
The tree constraint can be expressed in term of (1) a set of $|\text{NODES}|^2$ reified constraints for avoiding circuit between more than one node and of (2) $|\text{NODES}|$ reified constraints and of one sum constraint for counting the trees:

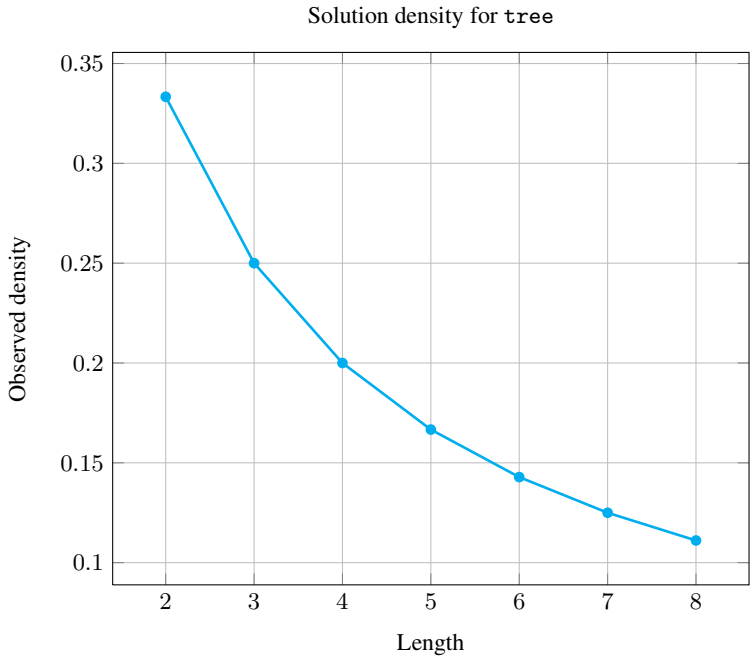
1. For each vertex NODES$[i]$ ($i \in [1, |\text{NODES}|]$) of the NODES collection we create a variable R_i that takes its value within interval $[1, |\text{NODES}|]$. This variable represents the *rank* of vertex NODES$[i]$ within a solution. It is used to prevent the creation of circuit involving more than one vertex as explained now. For each pair of vertices NODES$[i]$, NODES$[j]$ ($i, j \in [1, |\text{NODES}|]$) of the NODES collection we create a reified constraint of the form NODES$[i]$.succ $=$ NODES$[j]$.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 NODES$[i]$ to another vertex NODES$[j]$, then R_i should be strictly less than R_j.
2. For each vertex NODES$[i]$ ($i \in [1, |\text{NODES}|]$) of the NODES collection we create a 0-1 variable B_i and state the following reified constraint NODES$[i]$.succ $=$ NODES$[i]$.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 NODES$[i]$. Finally we create a constraint NTREES $= B_1 + B_2 + \cdots + B_{|\text{NODES}|}$ for stating the fact that the number of trees is equal to the number of loops of the graph.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	3	16	125	1296	16807	262144	4782969

Number of solutions for tree: domains $0..n$

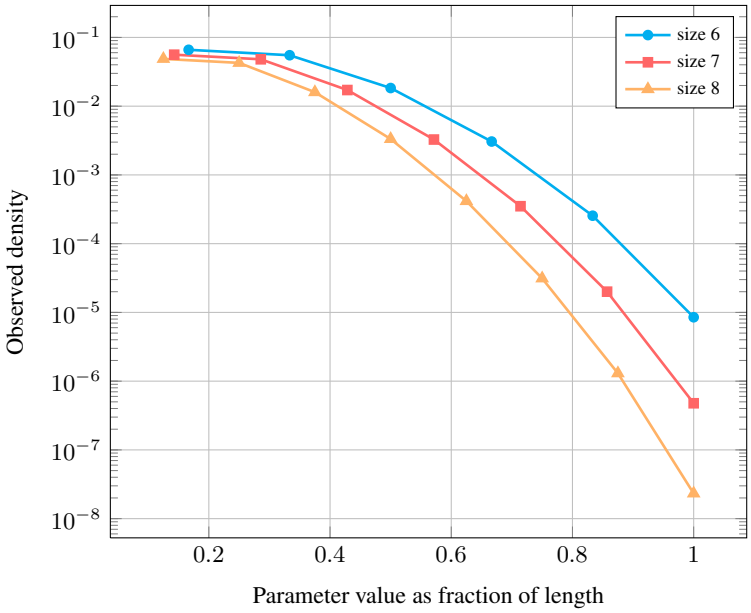

Solution density for tree
Observed density
$10^{-0.5}$
$10^{-0.6}$
$10^{-0.7}$
$10^{-0.8}$
$10^{-0.9}$
10^{-1}
2
3
4
5
6
7
8
Length

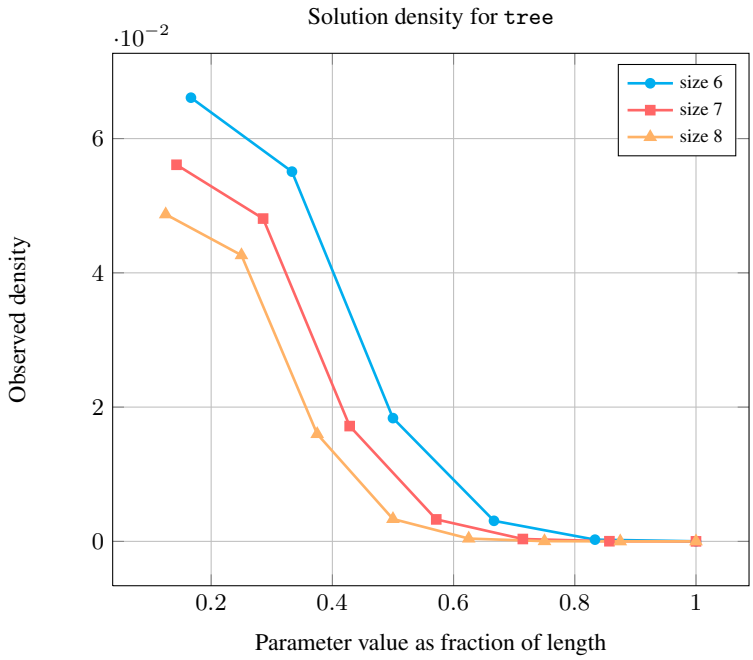

Solution density for tree
Observed density
0.35
0.3
0.25
0.2
0.15
0.1
2
3
4
5
6
7
8
Length

Length (n)		2	3	4	5	6	7	8
Total		3	16	125	1296	16807	262144	4782969
Parameter value	1	2	9	64	625	7776	117649	2097152
	2	1	6	48	500	6480	100842	1835008
	3	-	1	12	150	2160	36015	688128
	4	-	-	1	20	360	6860	143360
	5	-	-	-	1	30	735	17920
	6	-	-	-	-	1	42	1344
	7	-	-	-	-	-	1	56
	8	-	-	-	-	-	-	1

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

Solution density for `tree`





Systems tree in **Choco**.

See also **common keyword:** cycle, graph_crossing, map *(graph partitioning constraint)*, proper_forest *(connected component,tree)*.

implied by: binary_tree.

implies (items to collection): atleast_nvector.

related: balance_tree *(counting number of trees versus controlling how balanced the trees are)*, global_cardinality_low_up_no_loop, global_cardinality_no_loop *(can be used for restricting number of children since discard loops associated with tree roots)*.

shift of concept: stable_compatibility, tree_range, tree_resource.

specialisation: binary_tree *(no limit on the number of children replaced by at most two children)*, path *(no limit on the number of children replaced by at most one child)*.

uses in its reformulation: tree_range, tree_resource.

Keywords **constraint type:** graph constraint, graph partitioning constraint.

filtering: DFS-bottleneck, strong articulation point, arc-consistency.

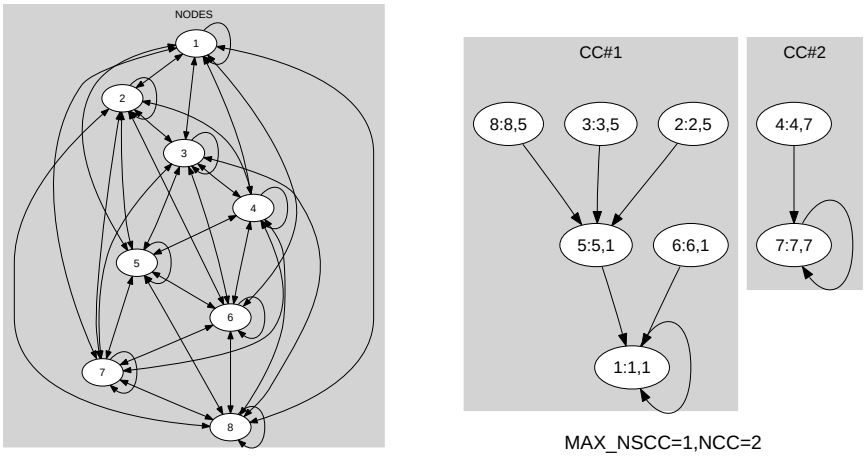
final graph structure: connected component, tree, one_succ.

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 = NTREES

Graph model We use the graph property $\mathbf{MAX\_NSCC} \leq 1$ in order to specify the fact that the size of the largest strongly connected component should not exceed one. In fact each root of a tree is a strongly connected component with a single vertex. The second graph property $\mathbf{NCC} = \texttt{NTREES}$ enforces the number of trees to be equal to the number of connected components.

Parts (A) and (B) of Figure 5.765 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the $\mathbf{NCC}$ graph property, we display the two connected components of the final graph. Each of them corresponds to a tree. The tree constraint holds since all strongly connected components of the final graph have no more than one vertex and since $\texttt{NTREES} = \mathbf{NCC} = 2$.



(A) **(B)**

Figure 5.765: Initial and final graph of the tree constraint