5.187 increasing_nvalue

DESCRIPTION LINKS GRAPH AUTOMATON

Origin Conjoin nvalue and increasing.

Constraint increasing_nvalue(NVAL, VARIABLES)

Arguments

NVAL	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

NVAL $\geq$ min(1, |VARIABLES|)
NVAL $\leq$ |VARIABLES|
required(VARIABLES, var)
increasing(VARIABLES)

Purpose The variables of the collection VARIABLES are increasing. In addition, NVAL is the number of distinct values taken by the variables of the collection VARIABLES.

Example

$(2, \langle 6, 6, 8, 8, 8\rangle)$
$(1, \langle 6, 6, 6, 6, 6\rangle)$
$(5, \langle 0, 2, 3, 6, 7\rangle)$

The first increasing_nvalue constraint (see Figure 5.415 for a graphical representation) holds since:

- The values of the collection $\langle 6, 6, 8, 8, 8\rangle$ are sorted in increasing order.
- NVAL $= 2$ is set to the number of distinct values occurring within the collection $\langle 6, 6, 8, 8, 8\rangle$.

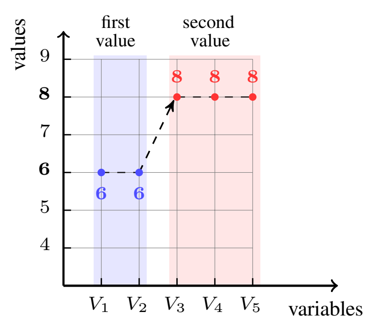


Figure 5.415: Illustration of the first example of the **Example** slot: five variables V_1, V_2, V_3, V_4, V_5 respectively fixed to values 6, 6, 8, 8 and 8, and the corresponding number of distinct values NVAL $= 2$

Typical

$|\texttt{VARIABLES}| > 1$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$

Symmetry

One and the same constant can be added to the `var` attribute of all items of `VARIABLES`.

Arg. properties

Functional dependency: `NVAL` determined by `VARIABLES`.

Algorithm

A complete filtering algorithm in a linear time complexity over the sum of the domain sizes is described in [45].

Reformulation

The `increasing_nvalue` constraint can be expressed in term of a conjunction of a `nvalue` and an `increasing` constraints (i.e., a chain of non strict inequality constraints on adjacent variables of the collection `VARIABLES`). But as shown by the following example, $V_1 \in [1,2]$, $V_2 \in [1,2]$, $V_1 \leq V_2$, $\texttt{nvalue}(2, \langle V_1, V_2 \rangle)$, this hinders propagation (i.e., the unique solution $V_1 = 1$, $V_2 = 2$ is not directly obtained after stating all the previous constraints).

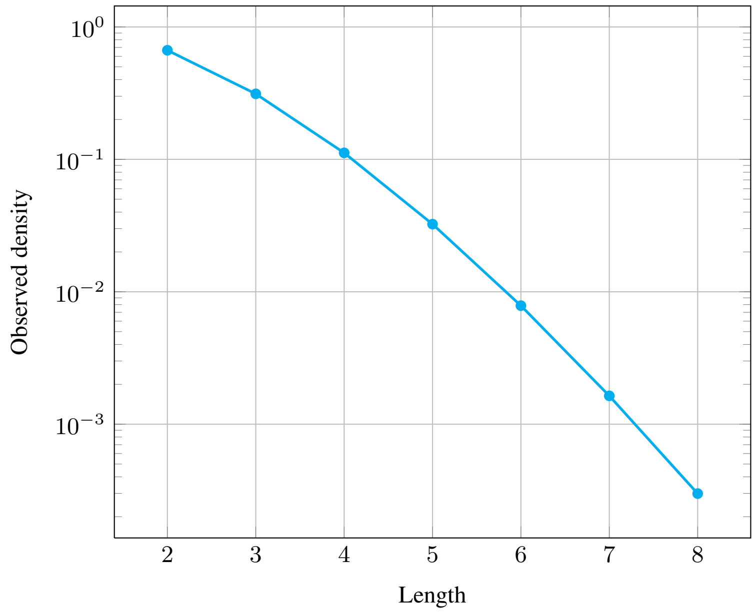
A better reformulation achieving arc-consistency uses the `seq_bin` constraint [310] that we now introduce. Given `N` a domain variable, `X` a sequence of domain variables, and `C` and `B` two binary constraints, $\texttt{seq\_bin}(\texttt{N}, \texttt{X}, \texttt{C}, \texttt{B})$ holds if (1) `N` is equal to the number of `C`-stretches in the sequence `X`, and (2) `B` holds on any pair of consecutive variables in `X`. A `C`-stretch is a generalisation of the notion of stretch introduced by G. Pesant [305], where the equality constraint is made explicit by replacing it by a binary constraint `C`, i.e., a `C`-stretch is a maximal length subsequence of `X` for which the binary constraint `C` is satisfied on consecutive variables. $\texttt{increasing\_nvalue}(\texttt{NVAL}, \texttt{VARIABLES})$ can be reformulated as $\texttt{seq\_bin}(\texttt{NVAL}, \texttt{VARIABLES}, =, \leq)$.

Counting

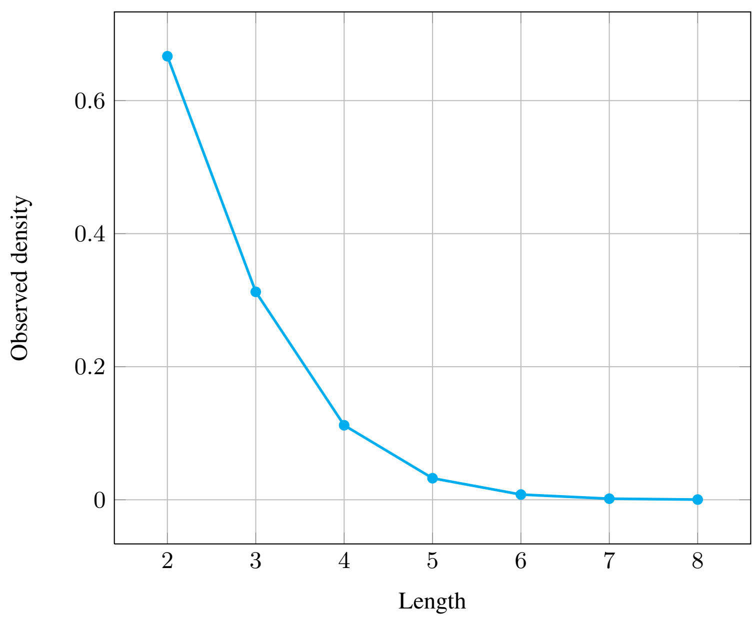
Length (n)	2	3	4	5	6	7	8
Solutions	6	20	70	252	924	3432	12870

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

Solution density for `increasing_nvalue`



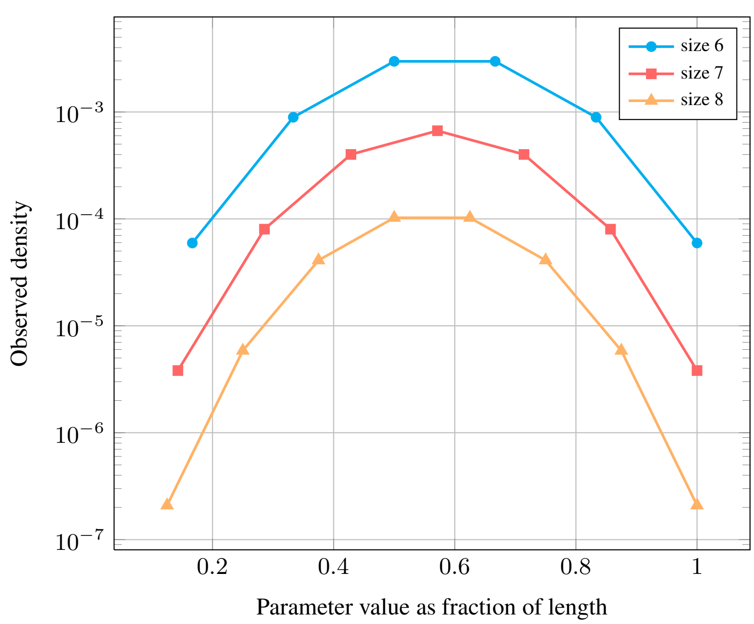
Solution density for `increasing_nvalue`

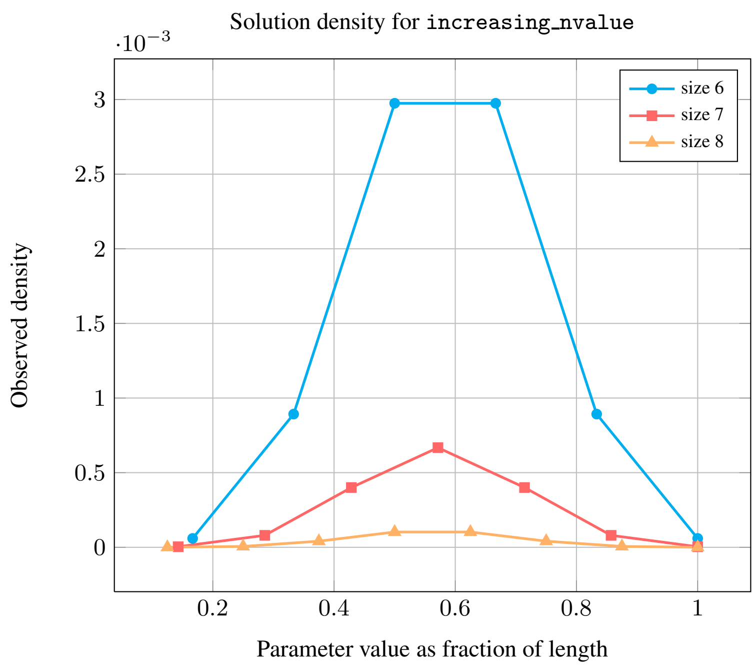


Length (n)		2	3	4	5	6	7	8
Total		6	20	70	252	924	3432	12870
Parameter value	1	3	4	5	6	7	8	9
	2	3	12	30	60	105	168	252
	3	-	4	30	120	350	840	1764
	4	-	-	5	60	350	1400	4410
	5	-	-	-	6	105	840	4410
	6	-	-	-	-	7	168	1764
	7	-	-	-	-	-	8	252
	8	-	-	-	-	-	-	9

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

Solution density for `increasing_nvalue`





Systems increasingNValue in **Choco**.

See also **implies:** `increasing` (*remove* `NVAL` *parameter from* `increasing_nvalue`), `nvalue`, `nvisible_from_start`.

related: `increasing_nvalue_chain`.

shift of concept: `ordered_nvector` (`variable` *replaced by* `vector` *and* $\leq$ *replaced by* `lex_lesseq`).

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: counting constraint, value partitioning constraint, order constraint.

filtering: arc-consistency.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes, number of distinct values, functional dependency.

symmetry: symmetry.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NSCC** = NVAL
Graph class	EQUIVALENCE

Graph model Parts (A) and (B) of Figure 5.416 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a value that is assigned to some variables of the VARIABLES collection. The 2 following values 6 and 8 are used by the variables of the VARIABLES collection.

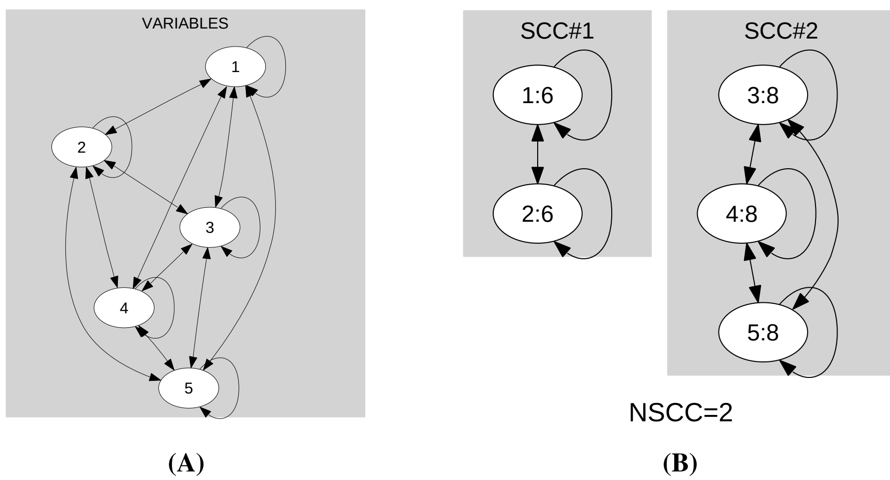


Figure 5.416: Initial and final graph of the increasing_nvalue constraint

Automaton

A first systematic approach for creating an automaton that only recognises the solutions to the `increasing_nvalue` constraint could be to:

- First, create an automaton that recognises the solutions to the `increasing` constraint.
- Second, create an automaton that recognises the solutions to the `nvalue` constraint.
- Third, make the product of the two previous automata and minimise the resulting automaton.

However this approach is not going to scale well in practice since the automaton associated with the `nvalue` constraint has a too big size. Therefore we propose an approach where we directly construct in a single step the automaton that only recognises the solutions to the `increasing_nvalue` constraint. Note that we do not have any formal proof that the resulting automaton is always minimum.

Without loss of generality, assume that the collection of variables `VARIABLES` contains at least one variable (i.e., $|\texttt{VARIABLES}| \geq 1$). Let l, m, n, min and max respectively denote the minimum and maximum possible value of variable `NVAL`, the number of variables of the collection `VARIABLES`, the smallest value that can be assigned to the variables of `VARIABLES`, and the largest value that can be assigned to the variables of `VARIABLES`. Let $s = max - min + 1$ denote the total number of potential values. Clearly, the maximum number of distinct values that can be assigned to the variables of the collection `VARIABLES` cannot exceed the quantity $d = \min(m, n, s)$. The $\frac{s\cdot(s+1)}{2} - \frac{(s-d)\cdot(s-d+1)}{2} + 1$ states of the automaton that only accepts solutions to the `increasing_nvalue` constraint can be defined in the following way:

- We have an initial state labelled by s_{00}.
- We have $\frac{s\cdot(s+1)}{2} - \frac{(s-d)\cdot(s-d+1)}{2}$ states labelled by s_{ij} $(1 \leq i \leq d, i \leq j \leq s)$. The first index i of a state s_{ij} corresponds to the number of distinct values already encountered, while the second index j denotes the the current value (i.e., more precisely the index of the current value, where the minimum value has index 1).

Terminal states depend on the possible values of variable `NVAL` and correspond to those states s_{ij} such that i is a possible value for variable `NVAL`. Note that we assume no further restriction on the domain of `NVAL` (otherwise the set of accepting states needs to be reduced in order to reflect the current set of possible values of `NVAL`). Three classes of transitions are respectively defined in the following way:

1. There is a transition, labelled by $min + j - 1$, from the initial state s_{00} to the state s_{1j} $(1 \leq j \leq s)$.
2. There is a loop, labelled by $min + j - 1$ for every state s_{ij} $(1 \leq i \leq d, i \leq j \leq s)$.
3. $\forall i \in [1, d-1], \forall j \in [i, s], \forall k \in [j+1, s]$ there is a transition labelled by $min+k-1$ from s_{ij} to s_{i+1k}.

We respectively have s transitions of class 1, $\frac{s\cdot(s+1)}{2} - \frac{(s-d)\cdot(s-d+1)}{2}$ transitions of class 2, and $\frac{(s-1)\cdot s\cdot(s+1)}{6} - \frac{(s-d)\cdot(s-d+1)\cdot(s-d+2)}{6}$ transitions of class 3.

Note that all states s_{ij} such that $i + s - j < l$ can be discarded since they do not allow to reach the minimum number of distinct values required l.

Part (A) of Figure 5.417 depicts the automaton associated with the increasing_nvalue constraint of the first example of the **Example** slot. For this purpose, we assume that variable NVAL is fixed to value 2 and that variables of the collection VARIABLES take their values within interval $[6, 8]$. Part (B) of Figure 5.417 represents the simplified automaton where all states that do not allow to reach an accepting state were removed. The corresponding increasing_global_cardinality constraint holds since the corresponding sequence of visited states, s_{00} s_{11} s_{11} s_{23} s_{23} s_{23}, ends up in an accepting state (i.e., accepting states are denoted graphically by a double circle).

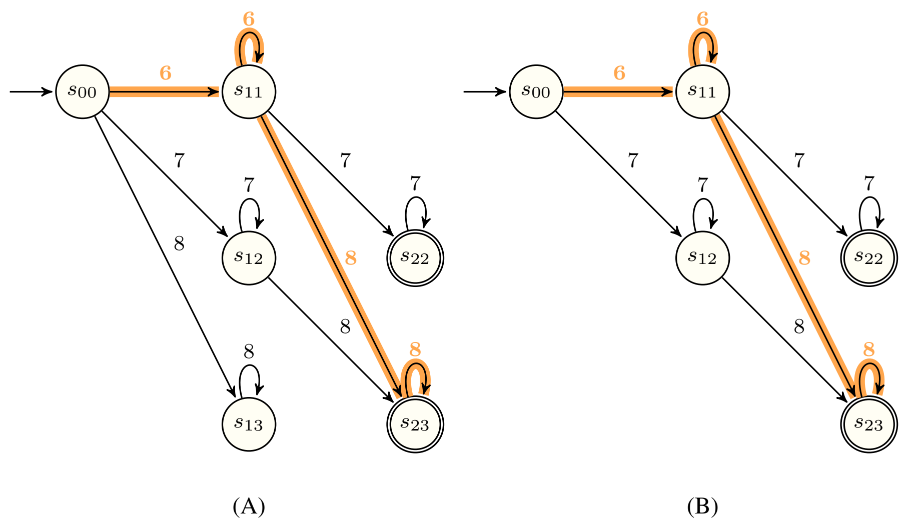


Figure 5.417: Automaton – Part (A) – and simplified automaton – Part (B) – of the increasing_nvalue$(2, \langle 6, 6, 8, 8, 8\rangle)$ constraint of the first example of the **Example** slot: the path corresponding to the second argument $\langle 6, 6, 8, 8, 8\rangle$ is depicted by thick orange arcs, where the self-loop on state s_{23} is applied twice

Figure 5.418 depicts a second deterministic automaton with one counter associated with the increasing_nvalue constraint, where the argument NVAL is unified to the final value of the counter.

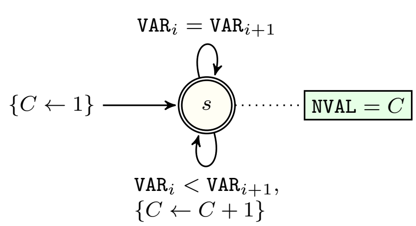


Figure 5.418: Automaton (with one counter) of the increasing_value constraint for a non-empty collection of variables