# 5.262 minimum

DESCRIPTION LINKS GRAPH AUTOMATON

**Origin** CHIP

**Constraint** minimum(MIN, VARIABLES)

**Synonym** min.

**Arguments**

MIN : dvar
VARIABLES : collection(var−dvar)

**Restrictions**

|VARIABLES| > 0
required(VARIABLES, var)

**Purpose** MIN is the minimum value of the collection of domain variables VARIABLES.

**Example**

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

The first minimum constraint holds since its first argument MIN $=$ 2 is set to the minimum value of the collection $\langle 3, 2, 7, 2, 6\rangle$.

**Typical**

|VARIABLES| > 1
range(VARIABLES.var) > 1

**Symmetries**

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped.
- One and the same constant can be added to MIN as well as to the var attribute of all items of VARIABLES.

**Arg. properties**

- Functional dependency: MIN determined by VARIABLES.
- Aggregate: MIN(min), VARIABLES(union).

**Usage** In some project scheduling problems one has to introduce dummy activities that correspond for instance to the starting time of a given set of activities. In this context one can use the minimum constraint to get the minimum starting time of a set of tasks.

**Remark** Note that minimum is a constraint and not just a function that computes the minimum value of a collection of variables: potential values of MIN influence the variables of VARIABLES, and reciprocally potential values that can be assigned to variables of VARIABLES influence MIN.

The minimum constraint is called min in **JaCoP** (http://www.jacop.eu/).

**Algorithm**

A filtering algorithm for the `minimum` constraint is described in [27].

The `minimum` constraint is entailed if all the following conditions hold:

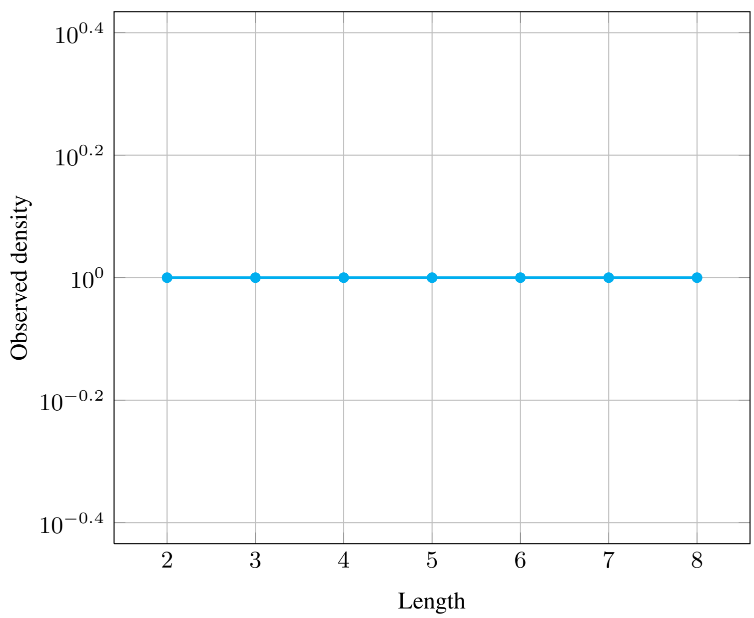
1. `MIN` is fixed.
2. At least one variable of `VARIABLES` is assigned value `MIN`.
3. All variables of `VARIABLES` have their minimum value greater than or equal to value `MIN`.

**Counting**

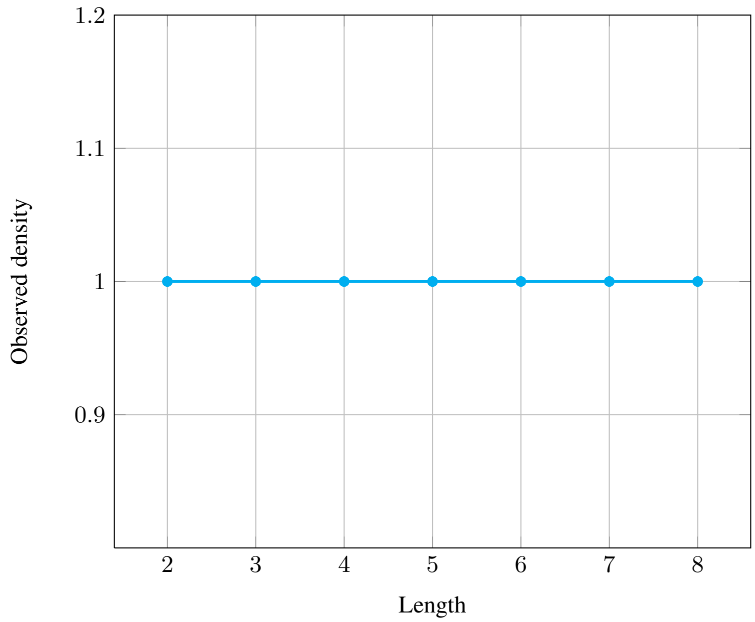
| Length ($n$) | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Solutions | 9 | 64 | 625 | 7776 | 117649 | 2097152 | 43046721 |

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

Solution density for `minimum`

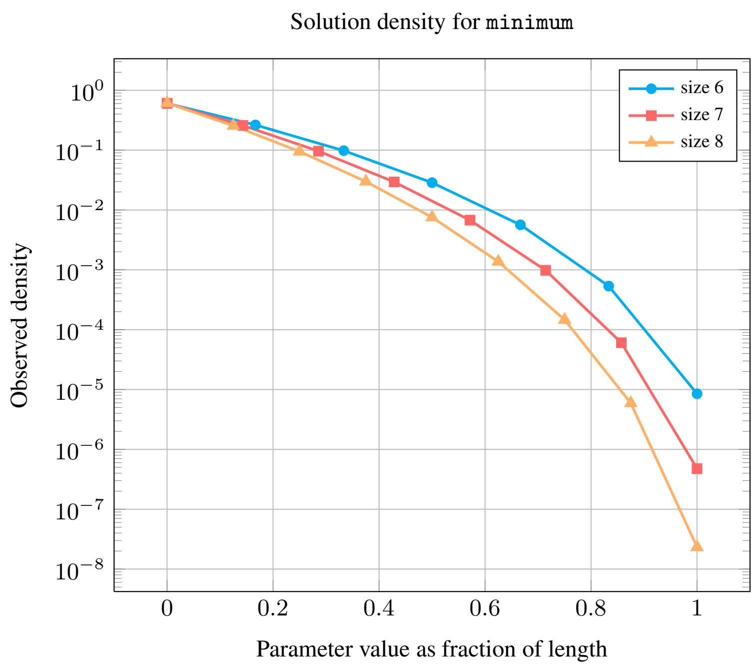


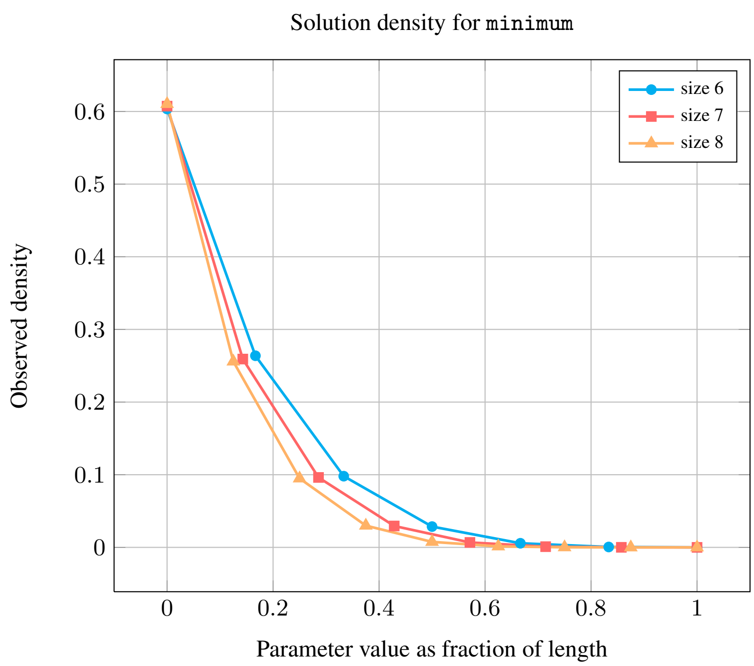
Solution density for `minimum`




| Length ($n$) | | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Total | | 9 | 64 | 625 | 7776 | 117649 | 2097152 | 43046721 |
| Parameter value | 0 | 5 | 37 | 369 | 4651 | 70993 | 1273609 | 26269505 |
| | 1 | 3 | 19 | 175 | 2101 | 31031 | 543607 | 11012415 |
| | 2 | 1 | 7 | 65 | 781 | 11529 | 201811 | 4085185 |
| | 3 | - | 1 | 15 | 211 | 3367 | 61741 | 1288991 |
| | 4 | - | - | 1 | 31 | 665 | 14197 | 325089 |
| | 5 | - | - | - | 1 | 63 | 2059 | 58975 |
| | 6 | - | - | - | - | 1 | 127 | 6305 |
| | 7 | - | - | - | - | - | 1 | 255 |
| | 8 | - | - | - | - | - | - | 1 |

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




**Systems** min in **Choco**, min in **Gecode**, min in **JaCoP**, minimum in **MiniZinc**, minimum in **SICStus**.

**Used in** minimum_greater_than, next_element, next_greater_element.

**See also**

**common keyword:** maximum *(order constraint)*.

**comparison swapped:** maximum.

**generalisation:** minimum_modulo *(*variable *replaced by* variable mod constant*)*.

**implied by:** and.

**implies:** between_min_max, in.

**soft variant:** minimum_except_0 *(value* 0 *is ignored)*, open_minimum *(open constraint)*.

**specialisation:** min_n *(minimum or order* n *replaced by absolute minimum)*.

**uses in its reformulation:** cycle.

**Keywords**

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

**constraint arguments:** reverse of a constraint, pure functional dependency.

**constraint network structure:** centered cyclic(1) constraint network(1).

**constraint type:** order constraint.

**filtering:** glue matrix, arc-consistency, entailment.

**modelling:** functional dependency.

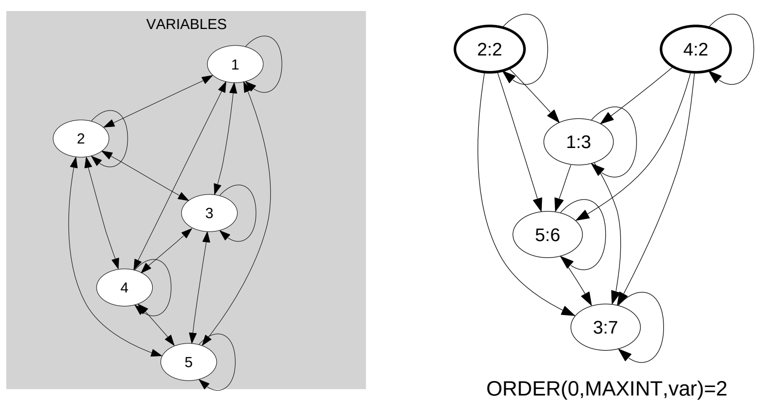
**Cond. implications**

minimum(MIN, VARIABLES)
with first(VARIABLES.var) $>$ MIN
and last(VARIABLES.var) $>$ MIN
**implies** deepest_valley(DEPTH, VARIABLES).

| | |
|---|---|
| **Arc input(s)** | `VARIABLES` |
| **Arc generator** | $\mathit{CLIQUE} \mapsto$ `collection`(`variables1`, `variables2`) |
| **Arc arity** | 2 |
| **Arc constraint(s)** | $\bigvee\left(\begin{array}{l}\texttt{variables1.key} = \texttt{variables2.key},\\ \texttt{variables1.var} < \texttt{variables2.var}\end{array}\right)$ |
| **Graph property(ies)** | **ORDER**(0, `MAXINT`, `var`) = `MIN` |

**Graph model** The condition `variables1.key` = `variables2.key` holds if and only if `variables1` and `variables2` corresponds to the same vertex. It is used in order to enforce to keep all the vertices of the initial graph. **ORDER**(0, `MAXINT`, `var`) refers to the source vertices of the graph, i.e., those vertices that do not have any predecessor.

Parts (A) and (B) of Figure 5.555 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **ORDER** graph property, the vertices of rank 0 (without considering the loops) of the final graph are outlined with a thick circle.




**(A)** **(B)**

Figure 5.555: Initial and final graph of the `minimum` constraint

**Automaton** Figure 5.556 depicts a first counter free deterministic automaton associated with the `minimum` constraint. Let $\texttt{VAR}_i$ be the $i^{th}$ variable of the `VARIABLES` collection. To each pair $(\texttt{MIN}, \texttt{VAR}_i)$ corresponds a signature variable $S_i$ as well as the following signature constraint: $(\texttt{MIN} < \texttt{VAR}_i \Leftrightarrow S_i = 0) \wedge (\texttt{MIN} = \texttt{VAR}_i \Leftrightarrow S_i = 1) \wedge (\texttt{MIN} > \texttt{VAR}_i \Leftrightarrow S_i = 2)$.

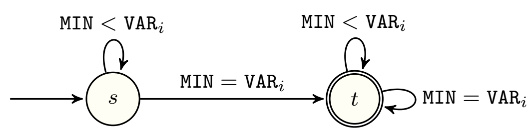



Figure 5.556: Counter free automaton of the `minimum` constraint

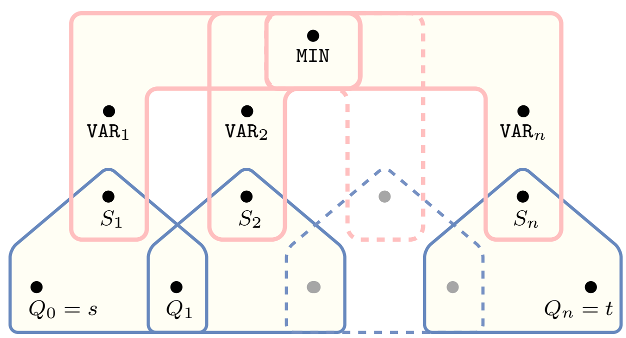



Figure 5.557: Hypergraph of the reformulation corresponding to the automaton of the `minimum` constraint

Figure 5.557 depicts a second counter free non deterministic automaton associated with the `minimum` constraint, where the argument `MIN` is also part of the sequence passed to the automaton.

Figure 5.560 depicts a third deterministic automaton with one counter associated with the `minimum` constraint, where the argument `MIN` is unified to the final value of the counter.

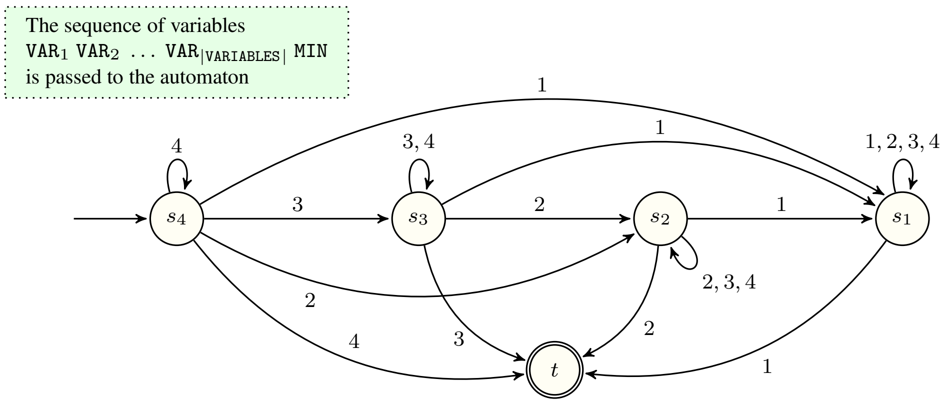



Figure 5.558: Counter free non deterministic automaton of the `minimum`(`MIN`, `VARIABLES`) constraint assuming that the union of the domain of the variables is the set $\{1, 2, 3, 4\}$ and that the elements of `VARIABLES` are first passed to the automaton followed by `MIN` (state $s_i$ means that no value strictly less than value $i$ was found and that value $i$ was already encountered at least once)

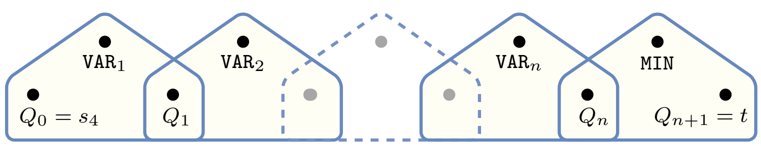



Figure 5.559: Hypergraph of the reformulation corresponding to the counter free non deterministic automaton of the `minimum` constraint

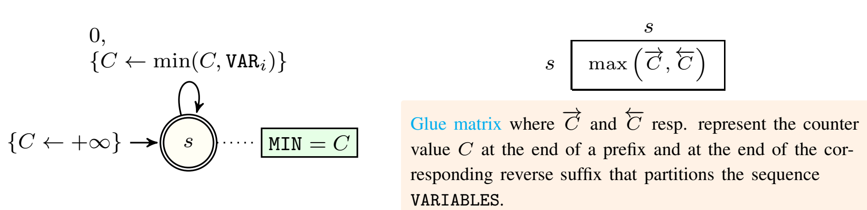



Figure 5.560: Automaton (with one counter) of the `minimum` constraint and its glue matrix

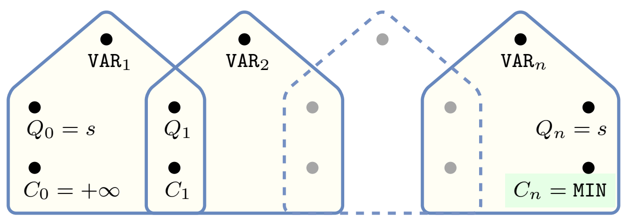



Figure 5.561: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the `minimum` constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state $s$ of the automaton, the transitions constraints share only the counter variable $C$ and the constraint network is Berge-acyclic