

5.49 change

| | DESCRIPTION | LINKS | GRAPH | AUTOMATON |
|---------------------|---|-------|-------|-----------|
| Origin | CHIP | | | |
| Constraint | change(NCHANGE, VARIABLES, CTR) | | | |
| Synonyms | nbchanges, similarity. | | | |
| Arguments | NCHANGE : dvar VARIABLES : collection(var-dvar) CTR : atom | | | |
| Restrictions | NCHANGE \geq 0 NCHANGE < VARIABLES required(VARIABLES, var) CTR \in [=, \neq , <, \geq , >, \leq] | | | |
| Purpose | NCHANGE is the number of times that constraint CTR holds on consecutive variables of the collection VARIABLES. | | | |
| Example | <div style="border: 1px solid black; padding: 5px; display: inline-block;"> $(3, \langle 4, 4, 3, 4, 1 \rangle, \neq)$ $(1, \langle 1, 2, 4, 3, 7 \rangle, >)$ </div> <p>In the first example the changes are located between values 4 and 3, 3 and 4, 4 and 1. Consequently, the corresponding change constraint holds since its first argument NCHANGE is fixed to value 3.</p> <p>In the second example the unique change occurs between values 4 and 3. Consequently, the corresponding change constraint holds since its first argument NCHANGE is fixed to 1.</p> | | | |
| Typical | NCHANGE > 0 VARIABLES > 1 range(VARIABLES.var) > 1 | | | |
| Symmetry | One and the same constant can be added to the var attribute of all items of VARIABLES. | | | |
| Usage | This constraint can be used in the context of timetabling problems in order to put an upper limit on the number of changes of job types during a given period. | | | |
| Remark | A similar constraint appears in [272, page 338] under the name of similarity constraint. The difference consists of replacing the arithmetic constraint CTR by a binary constraint. When CTR is equal to \neq this constraint is called nbchanges in [367]. | | | |
| Algorithm | A first incomplete algorithm is described in [29]. The sketch of a filtering algorithm for the conjunction of the change and the stretch constraints based on dynamic programming achieving arc-consistency is mentioned by Lars Hellsten in [184, page 56]. An arc-consistency algorithm in linear time of the sum of domain sizes is described in [47]. | | | |

- Used in** [pattern](#).
- See also** **common keyword:** [change_partition](#), [circular_change](#) (*number of changes in a sequence of variables with respect to a binary constraint*), [cyclic_change](#), [cyclic_change_joker](#) (*number of changes*), [smooth](#) (*number of changes in a sequence of variables with respect to a binary constraint*).
generalisation: [change_pair](#) (*variable replaced by pair of variables*).
shift of concept: [distance_change](#), [longest_change](#).
- Keywords** **characteristic of a constraint:** [automaton](#), [automaton with counters](#), [non-deterministic automaton](#).
constraint network structure: [sliding cyclic\(1\) constraint network\(2\)](#), [sliding cyclic\(1\) constraint network\(3\)](#), [Berge-acyclic constraint network](#).
constraint type: [timetabling constraint](#).
filtering: [dynamic programming](#), [arc-consistency](#).
final graph structure: [acyclic](#), [bipartite](#), [no loop](#).
modelling: [number of changes](#).

| | |
|----------------------------|---|
| Arc input(s) | VARIABLES |
| Arc generator | <i>PATH</i> \mapsto collection(variables1, variables2) |
| Arc arity | 2 |
| Arc constraint(s) | variables1.var CTR variables2.var |
| Graph property(ies) | NARC = NCHANGE |
| Graph class | <ul style="list-style-type: none"> • ACYCLIC • BIPARTITE • NO_LOOP |

Graph model

Since we are only interested by the constraints linking two consecutive items of the collection VARIABLES we use *PATH* to generate the arcs of the initial graph.

Parts (A) and (B) of Figure 5.91 respectively show the initial and final graph of the first example of the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

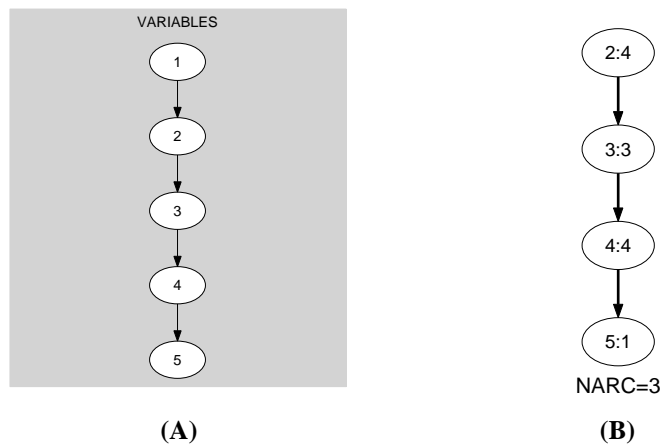


Figure 5.91: Initial and final graph of the change constraint

Automaton

Figure 5.92 depicts a first automaton that only accepts all the solutions of the change constraint. This automaton uses a counter in order to record the number of satisfied constraints of the form $\text{VAR}_i \text{ CTR } \text{VAR}_{i+1}$ already encountered. To each pair of consecutive variables $(\text{VAR}_i, \text{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable S_i . The following signature constraint links $\text{VAR}_i, \text{VAR}_{i+1}$ and S_i : $\text{VAR}_i \text{ CTR } \text{VAR}_{i+1} \Leftrightarrow S_i$.

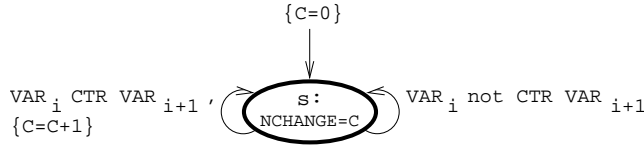


Figure 5.92: Automaton (with counter) of the change constraint

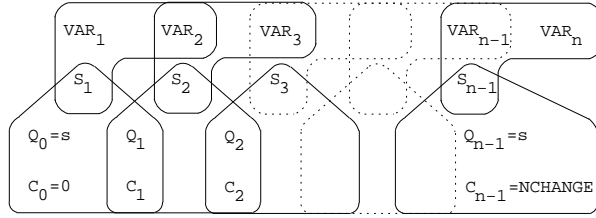


Figure 5.93: Hypergraph of the reformulation corresponding to the automaton (with counter) of the change constraint

Since the reformulation associated with the previous automaton is not **Berge-acyclic**, we now describe a second counter free automaton that also only accepts all the solutions of the change constraint. Without loss of generality, assume that the collection of variables VARIABLES contains at least two variables (i.e., $|\text{VARIABLES}| \geq 2$). Let n and \mathcal{D} respectively denote the number of variables of the collection VARIABLES, and the union of the domains of the variables of VARIABLES. Clearly, the maximum number of changes (i.e., the number of times the constraint $\text{VAR}_i \text{ CTR } \text{VAR}_{i+1}$ ($1 \leq i < n$) holds) cannot exceed the quantity $m = \min(n - 1, \overline{\text{NCHANGE}})$. The $(m + 1) \cdot |\mathcal{D}| + 2$ states of the automaton that only accepts all the solutions of the change constraint are defined in the following way:

- We have an initial state labelled by s_I .
- We have $m \cdot |\mathcal{D}|$ intermediate states labelled by s_{ij} ($i \in \mathcal{D}, j \in [0, m]$). The first subscript i of state s_{ij} corresponds to the value currently encountered. The second subscript j denotes the number of already encountered satisfied constraints of the form $\text{VAR}_i \text{ CTR } \text{VAR}_{i+1}$ from the initial state s_I to the state s_{ij} .
- We have a final state labelled by s_F .

Four classes of transitions are respectively defined in the following way:

1. There is a transition, labelled by i from the initial state s_I to the state s_{i0} , ($i \in \mathcal{D}$).
2. There is a transition, labelled by j , from every state s_{ij} , ($i \in \mathcal{D}, j \in [0, m]$), to the final state s_F .

3. $\forall i \in \mathcal{D}, \forall j \in [0, m], \forall k \in \mathcal{D} \cap \{k \mid i \neg \text{CTR } k\}$ there is a transition labelled by k from s_{ij} to s_{kj} (i.e., the counter j does not change for values k such that constraint i CTR k does not hold).
4. $\forall i \in \mathcal{D}, \forall j \in [0, m - 1], \forall k \in \mathcal{D} \setminus \{k \mid i \neg \text{CTR } k\}$ there is a transition labelled by k from s_{ij} to $s_{k,j+1}$ (i.e., the counter j is incremented by +1 for values k such that constraint i CTR k holds).

We have $|\mathcal{D}|$ transitions of type 1, $|\mathcal{D}| \cdot (m + 1)$ transitions of type 2, and at least $|\mathcal{D}|^2 \cdot m$ transitions of types 3 and 4. Since the maximum value of m is equal to $n - 1$, in the worst case we have at least $|\mathcal{D}|^2 \cdot (n - 1)$ transitions. This leads to a worst case time complexity of $O(|\mathcal{D}|^2 \cdot n^2)$ if we use Pesant's algorithm for filtering the `regular` constraint [275].

Figure 5.94 depicts the corresponding counter free non deterministic automaton associated with the change constraint under the hypothesis that (1) all variables of `VARIABLES` are assigned a value in $\{0, 1, 2, 3\}$, (2) $|\text{VARIABLES}|$ is equal to 4, and (3) CTR is equal to \neq .

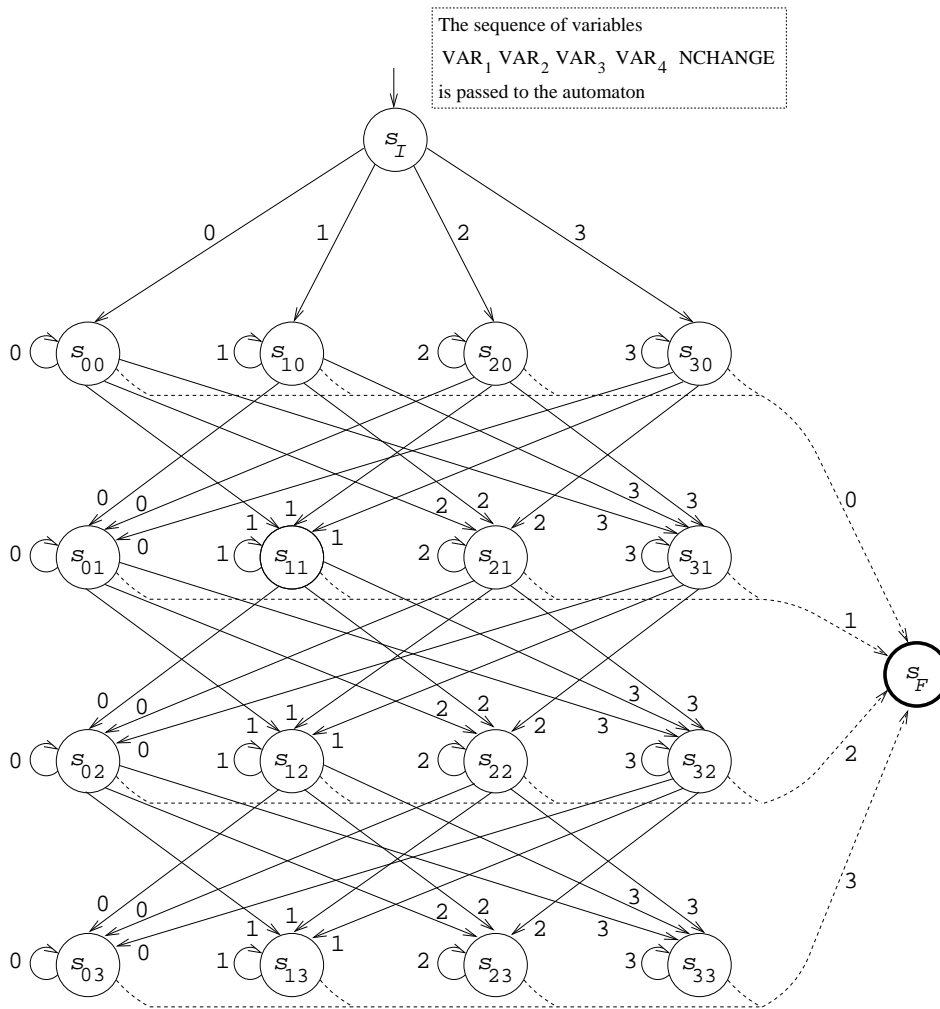


Figure 5.94: Counter free non deterministic automaton of the $\text{change}(\text{NCHANGE}, \langle \text{VAR}_1, \text{VAR}_2, \text{VAR}_3, \text{VAR}_4 \rangle, \neq)$ constraint assuming $\text{VAR}_i \in [0, 3]$ ($1 \leq i \leq 3$), with initial state s_I and final state s_F