Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence

A Generalized Arc-Consistency Algorithm for a Class of Counting Constraints

Thierry Petit and **Nicolas Beldiceanu** and **Xavier Lorca**
Mines-Nantes, LINA UMR CNRS 6241,
4, rue Alfred Kastler, FR-44307 Nantes, France.
{Thierry.Petit, Nicolas.Beldiceanu, Xavier.Lorca}@mines-nantes.fr

Abstract

This paper introduces the SEQ_BIN meta-constraint with a polytime algorithm achieving generalized arc-consistency. SEQ_BIN can be used for encoding counting constraints such as CHANGE, SMOOTH, or INCREASING_NVALUE. For all of them the time and space complexity is linear in the sum of domain sizes, which improves or equals the best known results of the literature.

1 Introduction

Many constraints are such that a *counting* variable is equal to the number of times a given property is satisfied in a sequence of variables. To represent some of these constraints in a generic way, we introduce the SEQ_BIN(N, X, C, B) meta-constraint, where N is an integer variable, X is a sequence of integer variables and C and B are two binary constraints.

Based on the notion C-stretch, a generalization of stretch [Pesant, 2001] where the equality constraint is made explicit and is replaced by C, SEQ_BIN holds if and only if two conditions are both satisfied: (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.

Among the constraints that can be expressed thanks to SEQ_BIN, many were introduced for solving real-world problems, *e.g.*, CHANGE [Cosytec, 1997] (time tabling problems), SMOOTH [Beldiceanu *et al.*, 2010a] (time tabling and scheduling), or INCREASING_NVALUE [Beldiceanu *et al.*, 2010b] (symmetry breaking for resource allocation problems).

The main contribution of this paper is a generic polytime filtering algorithm for SEQ_BIN, achieving generalized arc-consistency (GAC) when the constraint B is *monotonic* [Van Hentenryck *et al.*, 1992]. This algorithm can be seen as a generalization of the INCREASING_NVALUE filtering algorithm [Beldiceanu *et al.*, 2010b]. Given n the size of X, d the maximum domain size, and Σ_{Di} the sum of domain sizes, we characterize properties on C and B which lead to a time and space complexity in $O(\Sigma_{\mathrm{Di}})$. These properties are satisfied when SEQ_BIN represents CHANGE, SMOOTH and INCREASING_NVALUE. For all these constraints, our technique improves or equals the best known results.

Section 2 provides the definitions used in this paper. Section 3 defines SEQ_BIN and shows how to express well-known constraints with SEQ_BIN. Section 4 provides a necessary and sufficient condition for achieving GAC. Section 5 details the corresponding GAC filtering algorithm. Finally, Section 6 discusses about related works and Section 7 concludes.

2 Background

A *Constraint Network* is defined by a sequence of variables $X = [x_0, x_1, \ldots, x_{n-1}]$, a sequence of domains $\mathcal{D}$, where each $D(x_i) \in \mathcal{D}$ is the finite set of values that variable x_i can take, and a set of constraints $\mathcal{C}$ that specifies the allowed combinations of values for given subsets of variables. $\min(x)$ (resp. $\max(x)$) is the minimum (resp. maximum) value of $D(x)$. A sequence of variables $X' = [x_i, x_{i+1}, \ldots, x_j]$, $0 \le i \le j \le n-1$ (resp. $i > 0$ or $i < n-1$), is a *subsequence* (resp. a *strict subsequence*) of X and is denoted by $X' \subseteq X$ (resp. $X' \subset X$). $A[X]$ denotes an assignment of values to variables in X. Given $x \in X$, $A[x]$ is the value of x in $A[X]$. $A[X]$ is *valid* if and only if $\forall x_i \in X$, $A[x_i] \in D(x_i)$. An *instantiation* $I[X]$ is a valid assignment of X. Given $x \in X$, $I[x]$ is the value of x in $I[X]$. Given the sequence X and i, j two integers such that $0 \le i \le j \le n-1$, $I[x_i, \ldots, x_j]$ is the projection of $I[X]$ on $[x_i, x_{i+1}, \ldots, x_j]$. A *constraint* $C(X) \in \mathcal{C}$ specifies the allowed combinations of values for X. We also use the simple notation C. $C(X)$ defines a subset $\mathcal{R}_C(\mathcal{D})$ of the cartesian product of the domains $\Pi_{x_i \in X} D(x_i)$. If X is a pair of variables, then $C(X)$ is *binary*. We denote by vCw a pair of values (v, w) that satisfies a binary constraint C. $\neg C$ is the *opposite* of C, that is, $\neg C$ defines the relation $\mathcal{R}_{\neg C}(\mathcal{D}) = \Pi_{x_i \in X} D(x_i) \setminus \mathcal{R}_C(\mathcal{D})$. A *feasible instantiation* $I[X]$ of $C[X]$ is an instantiation which is in $\mathcal{R}_C(\mathcal{D})$. We say that $I[X]$ *satisfies* $C(X)$, or that $I[X]$ is a *support* on $C(X)$. Otherwise, $I[X]$ *violates* $C(X)$. If C is a binary constraint on $X = \{x_i, x_{i+1}\}$ and $v \in D(x_i)$ then the set of supports such that $x_i = v$ can be considered as a set of values (a subset of $D(x_{i+1})$). A *solution* of a constraint network is an instantiation of all the variables satisfying all the constraints.

Value $v \in D(x_i)$, $x_i \in X$, is (generalized) *arc-consistent* (GAC) with respect to $C(X)$ if and only if v belongs to a support of $C(X)$. A domain $D(x_i)$, $x_i \in X$, is GAC with respect to $C(X)$ if and only if $\forall v \in D(x_i)$, v is GAC with respect to $C(X)$. $C(X)$ is GAC if and only if $\forall x_i \in X$, $D(x_i)$ is GAC with respect to $C(X)$. A constraint network is GAC if and only if it is closed for *GAC* [Bessière, 2006]: $\forall x_i \in X$ all values in $D(x_i)$ that are not GAC with respect to a constraint in $\mathcal{C}$ have been removed.

3 The SEQ_BIN Meta-Constraint

We first generalize the notion of *stretches* [Pesant, 2001] to characterize a sequence of consecutive variables where the same binary constraint is satisfied.

Definition 1 (C-stretch). *Let $I[X]$ be an instantiation of the variable sequence $X = [x_0, x_1, \ldots, x_{n-1}]$ and C a binary constraint. The C-sequence constraint $\mathcal{C}(I[X], C)$ holds if and only if:*

- *Either $n = 1$,*
- *or $n > 1$ and $\forall k \in [0, n-2]$ $C(I[x_k], I[x_{k+1}])$ holds.*

A C-stretch of $I[X]$ is a subsequence $X' \subseteq X$ such that the two following conditions are both satisfied:

1. *The C-sequence $\mathcal{C}(I[X'], C)$ holds,*
2. *$\forall X$" such that $X' \subset X" \subseteq X$ the C-sequence $\mathcal{C}(I[X"], C)$ does not hold.*

The intuition behind Definition 1 is to consider the maximum length subsequences where the binary constraint C is satisfied between consecutive variables. Thanks to this generalized definition of stretches we can now introduce SEQ_BIN.

Definition 2. *The meta-constraint SEQ_BIN(N, X, C, B) is defined by a variable N, a sequence of n variables $X = [x_0, x_1, \ldots, x_{n-1}]$ and two binary constraints C and B. Given an instantiation $I[N, x_0, x_1, \ldots, x_{n-1}]$, SEQ_BIN$(N, X, C, B)$ is satisfied if and only if for any $i \in [0, n-2]$, $I[x_i]\, B\, I[x_{i+1}]$ holds, and $I[N]$ is equal to the number of C-stretches in $I[X]$.*

The constraint CHANGE was introduced in the context of timetabling problems [Cosytec, 1997], in order to put an upper limit on the number of changes of job types during a given period. The relation between classical stretches and CHANGE was initially stressed in [Hellsten, 2004, page 64]. CHANGE is defined on a variable N, a sequence of variables $X = [x_0, x_1, \ldots, x_{n-1}]$, and a binary constraint $C \in \{=, \neq, <, >, \leq, \geq\}$. It is satisfied if and only if N is equal to the number of times the constraint C holds on consecutive variables of X. Without hindering propagation (the constraint network is Berge-acyclic), CHANGE can be reformulated as SEQ_BIN$(N', X, \neg C, \texttt{true}) \wedge [N = N' - 1]$, where $\texttt{true}$ is the universal constraint.

SMOOTH(N, X) is a variant of CHANGE(N, X, C), where $x_i\, C\, x_{i+1}$ is defined by $|x_i - x_{i+1}| > cst$, $cst \in \mathbb{N}$. It is useful to limit the number of drastic variations on a cumulative profile [Beldiceanu *et al.*, 2010a; De Clercq, 2010].

As a last example, consider the INCREASING_NVALUE constraint, which is a specialized version of NVALUE [Pachet and Roy, 1999]. It was introduced for breaking variable symmetry in the context of resource allocation problems [Beldiceanu *et al.*, 2010b]. INCREASING_NVALUE is defined on a variable N and on a sequence of variables $X = [x_0, x_1, \ldots, x_{n-1}]$. Given an instantiation, INCREASING_NVALUE(N, X) is satisfied if and only if N is equal to the number of distinct values assigned to variables in X, and for any $i \in [0, n-2]$, $x_i \leq x_{i+1}$. We reformulate INCREASING_NVALUE(N, X) as SEQ_BIN$(N, X, =, \leq)$.

4 Consistency of SEQ_BIN

We first present how to compute, for any value in a given domain of a variable $x_i \in X$, the minimum and maximum number of C-stretches within the suffix of X starting at x_i (resp. the prefix of X ending at x_i) satisfying a chain of binary constraints of type B. Then, we introduce several properties useful to obtain a feasibility condition for SEQ_BIN, and a necessary and sufficient condition for filtering which leads to the GAC filtering algorithm presented in Section 5.

4.1 Computing of the Number of C-stretches

According to Definition 2, we have to ensure that the chain of B constraints are satisfied along the sequence of variables $X = [x_0, x_1, \ldots, x_{n-1}]$. An instantiation $I[X]$ is said *B-coherent* if and only if either $n = 1$ or for any $i \in [0, n-2]$, we have $I[x_i]\, B\, I[x_{i+1}]$. A value $v \in D(x_i)$ is said to be *B-coherent* with respect to x_i if and only if it can be part of at least one B-coherent instantiation. Then, given an integer $i \in [0, n-2]$, if $v \in D(x_i)$ is B-coherent *with respect to x_i* then there exists $w \in D(x_{i+1})$ such that $v\, B\, w$.

Consequently, within a given domain $D(x_i)$, values that are not B-coherent can be removed since they cannot be part of any solution of SEQ_BIN. Our aim is now to compute for each B-coherent value v in the domain of any variable x_i the minimum and maximum number of C-*stretches* on X.

Notation 1. *$\underline{s}(x_i, v)$ (resp. $\overline{s}(x_i, v)$) is the minimum (resp. maximum) number of C-stretches within the sequence of variables $[x_i, x_{i+1}, \ldots, x_{n-1}]$ under the hypothesis that $x_i = v$. $\underline{p}(x_i, v)$ (resp. $\overline{p}(x_i, v)$) is the minimum (resp. maximum) number of C-stretches within the sequence $[x_0, x_1, \ldots, x_i]$ under the hypothesis that $x_i = v$. Given $X = [x_0, x_1, \ldots, x_{n-1}]$, $\underline{s}(X)$ (resp. $\overline{s}(X)$) denotes the minimum (resp. maximum) value of $\underline{s}(x_0, v)$ (resp. $\overline{s}(x_0, v)$).*

Lemma 1. *Given SEQ_BIN(N, X, C, B) with $X = [x_0, x_1, \ldots, x_{n-1}]$, assume the domains in X contain only B-coherent values. Given $i \in [0, n-1]$ and $v \in D(x_i)$,*

- *If $i = n-1$: $\underline{s}(x_{n-1}, v) = 1$.*
- *Else:*

$$\underline{s}(x_i, v) = \min_{w \in D(x_{i+1})} \begin{pmatrix} \min_{[vBw] \wedge [vCw]}(\underline{s}(x_{i+1}, w)), \\ \min_{[vBw] \wedge [v \neg C w]}(\underline{s}(x_{i+1}, w)) + 1 \end{pmatrix}$$

Proof. By induction. From Definition 1, for any $v \in D(x_{n-1})$, we have $\underline{s}(x_{n-1}, v) = 1$ (i.e., a C-*stretch* of length 1). Consider now $x_i \in X$ with $i < n-1$, and a value $v \in D(x_i)$. Consider the set of instantiations $I[x_{i+1}, x_{i+2}, \ldots, x_{n-1}]$ that are B-coherent, and that minimize the number of C-*stretches* in $[x_{i+1}, x_{i+2}, \ldots, x_{n-1}]$. We denote this minimum number of C-*stretches* by $mins$. At least one B-coherent instantiation exists since all values in the domains of $[x_{i+1}, x_{i+2}, \ldots, x_{n-1}]$ are B-coherent. For each such instantiation, let us denote by w the value associated with $I[x_{i+1}]$. Either there exists such an instantiation with $mins$ C-*stretches* with the conjunction $B \wedge C$ satisfied by $(I[x_i], I[x_{i+1}])$. Then, $\underline{s}(x_i, v) = \underline{s}(x_{i+1}, w)$ since the first C-*stretch* of $I[x_{i+1}, x_{i+2}, \ldots, x_{n-1}]$ is extended when augmenting $I[x_{i+1}, x_{i+2}, \ldots, x_{n-1}]$ with value v for x_i. Or all instantiations $I[x_{i+1}, x_{i+2}, \ldots, x_{n-1}]$ with $mins$

C-*stretches* are such that C is violated by $(I[x_i], I[x_{i+1}])$: $(I[x_i], I[x_{i+1}])$ satisfies $B \wedge \neg C$. By construction, any instantiation $I[x_i, x_{i+1}, \ldots, x_{n-1}]$ with $I[x_i] = v$ has a number of C-*stretches* strictly greater than $mins$. Consequently, given $I[x_{i+1}, x_{i+2}, \ldots, x_{n-1}]$ with $mins$ C-*stretches*, the number of C-*stretches* obtained by augmenting this instantiation with value v for x_i is exactly $mins + 1$. □

Lemma 2. *Given* SEQ_BIN(N, X, C, B) *with* $X = [x_0, x_1, \ldots, x_{n-1}]$*, assume the domains in* X *contain only* B*-coherent values. Given* $i \in [0, n-1]$ *and* $v \in D(x_i)$*:*

- *If* $i = n - 1$*:* $\overline{s}(x_{n-1}, v) = 1$.
- *Else:*

$$\overline{s}(x_i, v) = \max_{w \in D(x_{i+1})} \begin{pmatrix} \max_{[vBw] \wedge [vCw]}(\overline{s}(x_{i+1}, w)), \\ \max_{[vBw] \wedge [v \neg Cw]}(\overline{s}(x_{i+1}, w)) + 1 \end{pmatrix}$$

Given a sequence of variables $[x_0, x_1, \ldots, x_{n-1}]$ such that their domains contain only B-coherent values, for any x_i in the sequence and any $v \in D(x_i)$, computing $\underline{p}(x_i, v)$ (resp. $\overline{p}(x_i, v)$) is symmetrical to $\underline{s}(x_i, v)$ (resp. $\overline{s}(x_i, v)$). We substitute min by max (resp. max by min), x_{i+1} by x_{i-1}, and vRw by wRv for any $R \in \{B, C, \neg C\}$.

4.2 Properties on the Number of C-stretches

This section provides the properties linking the values in a domain $D(x_i)$ with the minimum and maximum number of C-stretches in X. We consider only B-coherent values, which may be part of a feasible instantiation of SEQ_BIN. Next property is a direct consequence of Lemmas 1 and 2.

Property 1. *For any* B*-coherent value* v *in* $D(x_i)$*, with respect to* x_i*,* $\underline{s}(x_i, v) \leq \overline{s}(x_i, v)$*.*

Property 2. *Consider* SEQ_BIN(N, X, C, B)*, a variable* $x_i \in X$ *(*$0 \leq i \leq n-1$*), and two* B*-coherent values* $v_1, v_2 \in D(x_i)$*. If* $i = n-1$ *or if there exists a* B*-coherent* $w \in D(x_{i+1})$ *such that* $v_1 B w$ *and* $v_2 B w$*, then* $\overline{s}(x_i, v_1) + 1 \geq \underline{s}(x_i, v_2)$*.*

Proof. Obviously, if $i = n-1$. If $v_1 = v_2$, by Property 1 the property holds. Otherwise, assume there exist two values v_1 and v_2 such that $\exists w \in D(x_{i+1})$ for which $v_1 B w$ and $v_2 B w$, and $\overline{s}(x_i, v_1) + 1 < \underline{s}(x_i, v_2)$ (hypothesis H). By Lemma 2, $\overline{s}(x_i, v_1) \geq \overline{s}(x_{i+1}, w)$. By Lemma 1, $\underline{s}(x_i, v_2) \leq \underline{s}(x_{i+1}, w) + 1$. From hypothesis H, this entails $\overline{s}(x_{i+1}, w) + 1 < \underline{s}(x_{i+1}, w) + 1$, which leads to $\overline{s}(x_{i+1}, w) < \underline{s}(x_{i+1}, w)$, which is, by Property 1, not possible. □

Property 3. *Consider* SEQ_BIN(N, X, C, B)*, a variable* $x_i \in X$ *(*$0 \leq i \leq n-1$*), and two* B*-coherent values* $v_1, v_2 \in D(x_i)$*. If either* $i = n-1$ *or there exists* B*-coherent* $w \in D(x_{i+1})$ *such that* $v_1 \, B \, w$ *and* $v_2 \, B \, w$ *then, for any* $k \in [\min(\underline{s}(x_i, v_1), \underline{s}(x_i, v_2)), \max(\overline{s}(x_i, v_1), \overline{s}(x_i, v_2))]$*, either* $k \in [\underline{s}(x_i, v_1), \overline{s}(x_i, v_1)]$ *or* $k \in [\underline{s}(x_i, v_2), \overline{s}(x_i, v_2)]$*.*

Proof. Obviously, if $i = n-1$ or $v_1 = v_2$. If $[\underline{s}(x_i, v_1), \overline{s}(x_i, v_1)] \cap [\underline{s}(x_i, v_2), \overline{s}(x_i, v_2)]$ is not empty, then the property holds. Assume $[\underline{s}(x_i, v_1), \overline{s}(x_i, v_1)]$ and $[\underline{s}(x_i, v_2), \overline{s}(x_i, v_2)]$ are disjoint. W.l.o.g., assume $\overline{s}(x_i, v_1) < \underline{s}(x_i, v_2)$. By Property 2, $\overline{s}(x_i, v_1) + 1 \geq \underline{s}(x_i, v_2)$, thus $\overline{s}(x_i, v_1) = \underline{s}(x_i, v_2) - 1$. Either $k \in [\underline{s}(x_i, v_1), \overline{s}(x_i, v_1)]$ or $k \in [\underline{s}(x_i, v_2), \overline{s}(x_i, v_2)]$ (there is no hole in the range formed by the union of these intervals). □

4.3 Properties on Binary Constraints

Property 3 is central for providing a GAC filtering algorithm based on the count, for each B-coherent value in a domain, of the minimum and maximum number of C-*stretches* in complete instantiations. Given SEQ_BIN(N, X, C, B), we focus on binary constraints B which guarantee that Property 3 holds.

Definition 3. [Van Hentenryck *et al.*, 1992] *A binary constraint* F *is* monotonic *if and only if there exists a total ordering* $\prec$ *of values in domains such that: for any value* v *and any value* w*,* vFw *holds implies* $v'Fw'$ *holds for all valid tuple such that* $v' \prec v$ *and* $w \prec w'$*.*

Binary constraints $<$, $>$, $\leq$ and $\geq$ are monotonic, as well as the universal constraint `true`.

Property 4. *Consider* SEQ_BIN(N, X, C, B) *such that all non* B*-coherent values have been removed from domains of variables in* X*.* B *is monotonic if and only if for any variable* $x_i \in X$*,* $0 \leq i < n-1$*, for any values* $v_1, v_2 \in D(x_i)$*, there exists* $w \in D(x_{i+1})$ *such that* $v_1 B w$ *and* $v_2 B w$*.*

Proof. ($\Rightarrow$) From Definition 3 and since we consider only B-coherent values, each value has at least one support on B. Moreover, from Definition 3, $\{w \mid v_2 C w\} \subseteq \{w \mid v_1 C w\}$ or $\{w \mid v_1 C w\} \subseteq \{w \mid v_2 C w\}$. The property holds. ($\Leftarrow$) Suppose that the second proposition is true and B is not monotonic. From Definition 3, if B is not monotonic then $\exists v_1$ and v_2 in the domain of a variable $x_i \in X$ such that, by considering the constraint B on the pair of variables (x_i, x_{i+1}), neither $\{w \mid v_2 C w\} \subseteq \{w \mid v_1 C w\}$ nor $\{w \mid v_1 C w\} \subseteq \{w \mid v_2 C w\}$. Thus, there exists a support $v_1 B w$ such that (v_2, w) is not a support on B, and a support $v_2 B w'$ such that (v_1, w') is not a support on B. We can have $D(x_{i+1}) = \{w, w'\}$, which leads to a contradiction with the second proposition. The property holds. □

4.4 Feasibility

From Property 4, this section provides an equivalence relation between the existence of a solution for SEQ_BIN and the current variable domains of X and N. Without loss of generality, in this section we consider that all non B-coherent values have been removed from domains of variables in X. First, Definition 2 entails the following necessary condition for feasibility.

Proposition 1. *Given* SEQ_BIN(N, X, C, B)*, if* $\underline{s}(X) > \max(D(N))$ *or* $\overline{s}(X) < \min(D(N))$ *then* SEQ_BIN *fails.*

$D(N)$ can be restricted to $[\underline{s}(X), \overline{s}(X)]$, but $D(N)$ may have holes or may be strictly included in $[\underline{s}(X), \overline{s}(X)]$. We have the following proposition.

Proposition 2. *Consider* SEQ_BIN(N, X, C, B) *such that* B *is monotonic, with* $X = [x_0, x_1, \ldots, x_{n-1}]$*. For any integer* k *in* $[\underline{s}(X), \overline{s}(X)]$ *there exists* v *in* $D(x_0)$ *such that* $k \in [\underline{s}(x_0, v), \overline{s}(x_0, v)]$*.*

Proof. Let $v_1 \in D(x_0)$ a value such that $\underline{s}(x_0, v_1) = \underline{s}(X)$. Let $v_2 \in D(x_0)$ a value such that $\overline{s}(x_0, v_2) = \overline{s}(X)$. By

Property 4, either $n = 1$ or $\exists w \in D(x_1)$ such that $v_1 B w$ and $v_2 B w$. Thus, from Property 3, $\forall k \in [\underline{s}(X), \overline{s}(X)]$, either $k \in [\underline{s}(x_0, v_1), \overline{s}(x_0, v_1)]$ or $k \in [\underline{s}(x_0, v_2), \overline{s}(x_0, v_2)]$. □

Thus, any value for N in $D(N) \cap [\underline{s}(X), \overline{s}(X)]$ is generalized arc-consistent.

Proposition 3. *Given an instance of* SEQ_BIN(N, X, C, B) *such that B is monotonic,* SEQ_BIN(N, X, C, B) *has a solution if and only if* $[\underline{s}(X), \overline{s}(X)] \cap D(N) \neq \emptyset$.

Proof. ($\Rightarrow$) Assume SEQ_BIN(N, X, C, B) has a solution. Let $I[\{N\} \cup X]$ be such a solution. By Lemmas 1 and 2, the number of C-stretches $I[N]$ belongs to $[\underline{s}(X), \overline{s}(X)]$. ($\Leftarrow$) Let $k \in [\underline{s}(X), \overline{s}(X)] \cap D(N)$ (not empty). From Proposition 2, for any value k in $[\underline{s}(X), \overline{s}(X)]$, $\exists v \in D(x_0)$ such that $k \in [\underline{s}(x_0, v), \overline{s}(x_0, v)]$. By Definition 2 and since Lemmas 1 and 2 consider only B-coherent values, there is a solution of SEQ_BIN(N, X, C, B) with k C-stretches. □

4.5 Necessary and Sufficient Filtering Condition

Given SEQ_BIN(N, X, C, B), Proposition 3 can be used to filter the variable N from variables in X. Propositions 1 and 2 ensure that every remaining value in $[\underline{s}(X), \overline{s}(X)] \cap D(N)$ is involved in at least one solution satisfying SEQ_BIN. We consider now the filtering of variables in X.

Proposition 4. *Given* SEQ_BIN(N, X, C, B) *such that B is monotonic, let v be a value in* $D(x_i)$*,* $i \in [0, n-1]$*. The two following propositions are equivalent:*

1. *v is B-coherent and v is GAC with respect to* SEQ_BIN
2. $\begin{bmatrix} \underline{p}(x_i, v) + \underline{s}(x_i, v) - 1, \\ \overline{p}(x_i, v) + \overline{s}(x_i, v) - 1 \end{bmatrix} \cap D(N) \neq \emptyset$

Proof. If v is not B-coherent then, by Definition 2, v is not GAC. Otherwise, $\underline{p}(x_i, v)$ (resp. $\underline{s}(x_i, v)$) is the exact minimum number of *C-stretches* among B-coherent instantiations $I[x_0, x_1, \ldots, x_i]$ (resp. $I[x_i, x_{i+1}, \ldots, x_{n-1}]$) such that $I[x_i] = v$. Thus, by Lemma 1 (and its symmetrical for prefixes), the exact minimum number of C-stretches among B-coherent instantiations $I[x_0, x_1, \ldots, x_{n-1}]$ such that $I[x_i] = v$ is $\underline{p}(x_i, v) + \underline{s}(x_i, v) - 1$. Let $\mathcal{D}_{(i,v)} \subseteq \mathcal{D}$ such that all domains in $\mathcal{D}_{(i,v)}$ are equal to domains in $\mathcal{D}$ except $D(x_i)$ which is reduced to $\{v\}$. We call $X_{(i,v)}$ the sequence of variables associated with domains in $\mathcal{D}_{(i,v)}$. By construction $\underline{p}(x_i, v) + \underline{s}(x_i, v) - 1 = \underline{s}(X_{(i,v)})$. By a symmetrical reasoning, $\overline{p}(x_i, v) + \overline{s}(x_i, v) - 1 = \overline{s}(X_{(i,v)})$. By Proposition 3, the proposition holds. □

The "$-$ 1" in expressions $\underline{p}(x_i, v) + \underline{s}(x_i, v) - 1$ and $\overline{p}(x_i, v) + \overline{s}(x_i, v) - 1$ prevents us from counting twice a C-stretch at an extremity x_i of the two sequences $[x_0, x_1, \ldots, x_i]$ and $[x_i, x_{i+1}, \ldots, x_{n-1}]$.

5 GAC Filtering Algorithm

Based on the necessary and sufficient filtering condition of Proposition 4, this section provides an implementation of the GAC filtering algorithm for SEQ_BIN(N, X, C, B) with a monotonic constraint B.

If $B \notin \{\leq, \geq, <, >, \mathtt{true}\}$ then the total ordering $\prec$ entailing monotonicity of B is not the natural order of integers. In this case, if $\prec$ is not known, it is necessary to compute such an ordering with respect to all values in $\cup_{i \in [0,n-1]}(D(x_i))$, once before the first propagation of SEQ_BIN. Consider that the two variables of B can take any value in $\cup_{i \in [0,n-1]}(D(x_i))$: Due to the inclusion of sets of supports of values (see Definition 3), the order remains the same when the domains of the variables constrained by B do not contain all values in $\cup_{i \in [0,n-1]}(D(x_i))$.

To compute $\prec$, the following procedure can be used: Count the number of supports of each value, in $O(d^2)$ time (recall d is the maximum domain size of a variable in X), and sort values according to the number of supports, in $O(| \cup_{i \in [0,n-1]} (D(x_i))| log(| \cup_{i \in [0,n-1]} (D(x_i))|))$ time.

Then, given the sequence of variables X, the algorithm is decomposed into four phases:

① Remove all non B-coherent values in the domains of X.

② For all values in the domains of X, compute the minimum and maximum number of *C-stretches* of prefixes and suffixes.

③ Adjust the minimum and maximum value of N with respect to the minimum and maximum number of *C-stretches* of X.

④ Using the result phase ② and Proposition 4, prune the remaining B-coherent values.

With respect to phase ①, recall that B is monotonic: According to $\prec$, for any pair of variables (x_i, x_{i+1}), $\exists v_0$ in $D(x_i)$ such that $\forall v_j \in D(x_i)$, $v_j \neq v_0$, v_j has a set of supports on $B(x_i, x_{i+1})$ included in the supports of v_0 on $B(x_i, x_{i+1})$. By removing from $D(x_{i+1})$ non supports of v_0 on $B(x_i, x_{i+1})$ in $O(|D(x_{i+1})|)$, all non B-coherent values of $D(x_{i+1})$ with respect to $B(x_i, x_{i+1})$ are removed. By repeating such a process in the two directions (starting from the pair (x_{n-2}, x_{n-1}) and from the pair (x_0, x_1)), all non B--coherent values can be removed from domains in $O(\Sigma_{\mathrm{Di}})$ time complexity.

To achieve phase ② we use Lemmas 1 and 2 and their symmetrical formulations for prefixes. Without loss of generality, we focus on the minimum number of C-stretches of a value v_j in the domain of a variable x_i, $i < n-1$, thanks to Lemma 1. Assume that for all $w \in D(x_{i+1})$, $\underline{s}(x_{i+1}, w)$ has been computed. If there is no particular property on C, the supports $S_j \in D(x_{i+1})$ of v_j on $C(x_i, x_{i+1}) \wedge B(x_i, x_{i+1})$ and the subset $\neg S_j \in D(x_{i+1})$ of non-supports of v_j on $C(x_i, x_{i+1})$ which satisfy B have to be scanned, in order to determine for each set a value $w \in S_j$ minimizing $\underline{s}(x_{i+1}, w)$ and a value $w' \in \neg S_j$ minimizing $\underline{s}(x_{i+1}, w') + 1$. This process takes $O(|D(x_{i+1})|)$ for each value, leading to $O(d^2)$ for the whole domain. Since all the variables need to be scanned and for all the values in domains the quantities are stored, phase ② takes $O(nd^2)$ in time, and $O(\Sigma_{\mathrm{Di}})$ in space.

Phases ③ and ④ take $O(\Sigma_{\mathrm{Di}})$ time each since all the domains have to be scanned. By Proposition 4, all the non-GAC values have been removed after this last phase.

If $B \in \{\leq, \geq, <, >, \texttt{true}\}$, $\prec$ is known. The worst-case time and space results come from Phase ②. The bottleneck stems from the fact that, when a domain $D(x_i)$ is scanned, the minimum and maximum number of C-stretches of each value are computed from scratch, while an incremental computation would avoid to scan $D(x_{i+1})$ for each value in $D(x_i)$. This observation leads to Property 5. Again, we focus on the minimum number of C-stretches on suffixes. Other cases are symmetrical.

Notation 2. *Given* SEQ_BIN(N, X, C, B)*,* $x_i \in X$*,* $0 \leq i < n$ *and a value* $v_j \in D(x_i)$*, if* $i < n-1$*, let* V_j *denote the set of integer values such that a value* $s(v_j, w) \in V_j$ *corresponds to each* $w \in D(x_{i+1})$ *and is equal to:*

- $\underline{s}(x_{i+1}, w)$ *if and only if* $w \in S_j$
- $\underline{s}(x_{i+1}, w) + 1$ *if and only if* $w \in \neg S_j$

Within notation 2, the set V_j corresponds to the minimum number of stretches of values in $D(x_{i+1})$ increased by one if they are non supports of value v_j with respect to C.

Property 5. *Given* SEQ_BIN(N, X, C, B) *such that* $B \in \{\leq, \geq, <, >, \texttt{true}\}$ *and* $x_i \in X$*,* $0 \leq i < n-1$*, if the computation of* $\min_{w \in D(x_{i+1})}(s(v_j, w))$ *for all* $v_j \in D(x_i)$ *can be performed in* $O(|D(x_{i+1})|)$ *time then GAC can be achieved on* SEQ_BIN *in* $O(\Sigma_{\mathrm{Di}})$ *time and space complexity.*

Proof. Applying Lemma 1 to the whole domain $D(x_i)$ takes $O(|D(x_{i+1})|)$ time. Storing the minimum number of stretches for each value in $D(x_i)$ requires $O(|D(x_i)|)$ space. Phase ② takes $O(\Sigma_{\mathrm{Di}})$ space and $O(\Sigma_{\mathrm{Di}})$ time. □

When they are represented by SEQ_BIN, all the practical constraints mentioned in the introduction satisfy a condition that entails Property 5: Given x_i, it is possible to compute in $O(|D(x_{i+1})|)$ the quantity $\min_{w \in D(x_{i+1})}(s(v_0, w))$ for a first value $v_0 \in D(x_i)$ and then, following the natural order of integers, to derive with a constant or amortized time complexity the quantity for the next value v_1, and then the quantity for the next value v_2, and so on. Thus, to obtain GAC in $O(\Sigma_{\mathrm{Di}})$ for all these constraints, we specialize Phase ② in order to exploit such a property. We now detail how to proceed.

When SEQ_BIN represents CHANGE, SMOOTH or INCREASING_NVALUE, computing $\min_{w \in D(x_{i+1})}(s(v_0, w))$ for the minimum value $v_0 = \min(D(x_i))$ (respectively the maximum value) can be performed by scanning the minimum number of C-stretches of values in $D(x_{i+1})$.

We now study for CHANGE, SMOOTH and INCREASING_NVALUE how to efficiently compute the value $\min_{w \in D(x_{i+1})}(s(v_k, w))$ of $v_k \in D(x_i)$, either directly or from the previous value $\min_{w \in D(x_{i+1})}(s(v_{k-1}, w))$, in order to compute $\min_{w \in D(x_{i+1})}(s(v_j, w))$ for all $v_j \in D(x_i)$ in $O(|D(x_i)|)$ time and therefore achieve Phase ② in $O(\Sigma_{\mathrm{Di}})$.

The CHANGE constraint

Section 3 showed a reformulation of CHANGE(N, X, CTR) as SEQ_BIN$(N', X, C, \texttt{true}) \wedge [N = N' - 1]$, where C is the opposite of CTR.

– If C is '$=$' then for each $v_j \in D(x_i)$ there is a unique potential support for C on x_{i+1}, the value v_j. Therefore, by memorizing once the value $vmin_1$ in $D(x_{i+1})$ which corresponds to the smallest minimum numbers of C-stretches on the suffix starting at x_{i+1}: $\forall v_j$, $\min_{w \in D(x_{i+1})}(s(v_j, w)) = \min(\underline{s}(x_{i+1}, v_j), \underline{s}(x_{i+1}, vmin_1) + 1)$, assuming $\underline{s}(x_{i+1}, v_j) = +\infty$ when $v_j \notin D(x_{i+1})$.

– If C is '$\neq$' then for each $v_j \in D(x_i)$ there is a single non support. By memorizing the two values $vmin_1$ and $vmin_2$ which minimize the minimum numbers of C-stretches on the suffix starting at x_{i+1}, for any value v_j $\min_{w \in D(x_{i+1})}(s(v_j, w))$ is equal to: $\min(\underline{s}(x_{i+1}, vmin_1) + 1, \underline{s}(x_{i+1}, vmin_2))$ when $vmin_1 = v_j$, and $\underline{s}(x_{i+1}, vmin_1)$ otherwise.

– If C is '$>$' (the principle is similar for '$\leq$','$\geq$' and '$<$'), we introduce two quantities $lt(v_j, x_{i+1})$ and $geq(v_j, x_{i+1})$ respectively equal to $\min_{w \in [\min(D(x_i)), v_j[}(\underline{s}(x_{i+1}, w))$ and $\min_{w \in [v_j, \max(D(x_i))]}(\underline{s}(x_{i+1}, w))$. The computation is performed in three steps:

1. Starting from $v_0 = \min(D(x_i))$, that is, the value having the smallest number of supports for C on x_{i+1}, compute $lt(v_j, x_{i+1})$ in increasing order of v_j. Taking advantage that, given a value $v_{j-1} \in D(x_i)$ and the next value $v_j \in D(x_i)$, $[\min(D(x_i)), v_{j-1}[$ is included in $[\min(D(x_i)), v_j[$. Therefore, the computation of all $\min_{w \in [\min(D(x_i)), v_j[}(\underline{s}(x_{i+1}, w))$ can be amortized over $D(x_{i+1})$. The time complexity for computing $lt(v_j, x_{i+1})$ for all $v_j \in D(x_i)$ is in $O(|D(x_i)| + |D(x_{i+1})|)$.
2. Similarly starting from $v_0 = \max(D(x_i))$, compute incrementally $geq(v_j, x_{i+1})$ in decreasing order of v_j, in $O(|D(x_i)| + |D(x_{i+1})|)$.
3. Finally, for each $v_j \in D(x_i)$, $\min_{w \in D(x_{i+1})}(s(v_j, w))$ is equal to $\min(lt(v_j, x_{i+1}), geq(v_j, x_{i+1}) + 1)$.

Since step 3. takes $O(D(x_i))$, we get an overall time complexity for Phase ② in $O(\Sigma_{\mathrm{Di}})$.

The SMOOTH constraint

It is a variant of CHANGE(N, X, CTR), where $x_i \, CTR \, x_{i+1}$ is $|x_i - x_{i+1}| > cst$, $cst \in \mathbb{N}$ that can be reformulated as SEQ_BIN$(N', X, C, \texttt{true}) \wedge [N = N' - 1]$, where C is $|x_i - x_{i+1}| \leq cst$. Assume $v_0 = \min(D(x_i))$ and we scan values in increasing order. Supports of values in $D(x_i)$ for $|x_i - x_{i+1}| \leq cst$ define a set of sliding windows for which both the starts and the ends are increasing sequences (not necessarily strictly). Thus, $\min_{w \in S_j}(s(v_j, w))$ can be computed for all v_j in $D(x_i)$ in $O(|D(x_i)|)$ thanks to the *ascending minima algorithm.*[1] Given a value $v_j \in D(x_i)$ the set $\neg S_j$ of non supports of v_j on $|x_i - x_{i+1}| \leq cst$ is partitioned in two sequences of values: a first sequence before the smallest support and a second sequence after the largest support. While scanning values in $D(x_i)$ these two sequences correspond also to sliding windows on which the ascending minima algorithm can also be used.

The INCREASING_NVALUE constraint

It is represented by SEQ_BIN$(N, X, =, \leq)$. Since B is not $\texttt{true}$, we have to take into account B when evaluating

[1] See http://home.tiac.net/~cri/2001/slidingmin.html

$\min_{w \in D(x_{i+1})}(s(j, w))$ for each $v_j \in D(x_i)$. Fortunately, we can start from $v_0 = \max(D(x_i))$ and consider the decreasing order since B is '$\leq$'. In this case the set of supports on B can only increase as we scan $D(x_i)$. We follow the same idea used for CHANGE$(N, X, =)$, except that the quantity $vmin_1$ now represents the values in $D(x_{i+1})$ which corresponds to the smallest minimum numbers of C-stretches *only* on supports of the current value $v_j \in D(x_i)$ on B. Since the set of supports on B only increases, $vmin_1$ can be updated for each new value in $D(x_i)$ in $O(1)$.

6 Related Work

Using automata, CHANGE and SMOOTH can be represented either by REGULAR [Pesant, 2004] or by COST-REGULAR [Demassey *et al.*, 2006]. In the first case this leads to a GAC algorithm in $O(n^2d^2)$ time [Beldiceanu *et al.*, 2010a, pages 584–585, 1544–1545] (where d denotes the maximum domain size). In the second case the filtering algorithm of COST-REGULAR does not achieve GAC.

Bessière *et al.* [Bessière *et al.*, 2008] presented an encoding of the CARDPATH constraint with SLIDE_2. A similar reformulation can be used for encoding SEQ_BIN(N, X, C, B). Recall that $\text{SLIDE}_j(C, [x_0, x_1, \ldots, x_{n-1}])$ holds if and only if $C(x_{ij}, \ldots, x_{ij+k-1})$ holds for $0 \leq i \leq \frac{n-k}{j}$. Following a schema similar to the one proposed in Section 4 of Bessière *et al.* paper, SEQ_BIN(N, X, C, B) can be represented by adding a variable N' and n variables $[M_0, \ldots, M_{n-1}]$, with $M_0 = 0$ and $M_{n-1} = N'$. SEQ_BIN(N, X, C, B) is then reformlated by $\text{SLIDE}_2(C', [M_0, x_0, M_1, x_1, \ldots, M_{n-1}, x_{n-1}]) \wedge [N' = N - 1]$, where $C' = [\neg C(x_i, x_{i+1}) \wedge B(x_i, x_{i+1}) \wedge M_{i+1} = M_i + 1] \vee [C(x_i, x_{i+1}) \wedge B(x_i, x_{i+1}) \wedge M_{i+1} = M_i]$. According to Section 6 of Bessière *et al.* paper, GAC can be achieved thanks to a reformulation of SLIDE_2, provided a complete propagation is performed on C', which is the case because $B(x_i, x_{i+1})$ and $C(x_i, x_{i+1})$ involve the same variables. The reformulation requires n additional intersection variables (one by sub-sequence $[M_i, x_i]$), on which $O(n)$ compatibility constraints between pairs of intersection variables and $O(n)$ functional channelling constraints should hold. Arity of C' is $k = 4$ and $j = 2$: the domain of an intersection variable contains $O(d^{k-j}) = O(d^2)$ values (corresponding to binary tuples), where d is the maximum size of a domain. Enforcing GAC on a compatibility constraint takes $O(d^3)$ time, while functional channelling constraint take $O(d^2)$, leading to an overall time complexity $O(nd^3)$ for enforcing arc-consistancy on the reformulation, corresponding to GAC for SEQ_BIN. To compare such a time complexity $O(nd^3)$ with our algorithm, note that $O(\Sigma_{\text{Di}})$ is upper-bounded by $O(nd)$.

At last, some *ad hoc* techniques can be compared to our generic GAC algorithm, *e.g.*, a GAC algorithm in $O(n^3m)$ for CHANGE [Hellsten, 2004, page 57], where m is the total number of values in the domains of X. Moreover, the GAC algorithm for SEQ_BIN generalizes to a class of counting constraints the ad-hoc GAC algorithm for INCREASING_NVALUE [Beldiceanu *et al.*, 2010b] without degrading time and space complexity in the case where SEQ_BIN represents INCREASING_NVALUE.

7 Conclusion

Our contribution is a structural characterization of a class of counting constraints for which we come up with a general polytime GAC filtering algorithm, and a characterization of the property which makes such an algorithm linear in the sum of domain sizes. A still open question is whether it would be possible or not to extend this class (*e.g.*, considering n-ary constraints for B and C) without degrading complexity or giving up on GAC, in order to capture more constraints.

References

[Beldiceanu *et al.*, 2010a] N. Beldiceanu, M. Carlsson, and J.-X. Rampon. Global Constraint Catalog, 2nd Ed. Technical Report T2010-07, SICS, 2010.

[Beldiceanu *et al.*, 2010b] N. Beldiceanu, F. Hermenier, X. Lorca, and T. Petit. The *increasing nvalue* Constraint. In *CPAIOR*, volume 6140 of *LNCS*, pages 25–39, 2010.

[Bessière *et al.*, 2008] C. Bessière, E. Hebrard, B. Hnich, Z. Kızıltan, and T. Walsh. SLIDE: A useful special case of the CARDPATH constraint. In *ECAI*, 2008.

[Bessière, 2006] C. Bessière. Constraint Propagation. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 3. Elsevier, 2006.

[Cosytec, 1997] Cosytec. *CHIP Reference Manual*, release 5.1 edition, 1997.

[De Clercq, 2010] A. De Clercq. A soft constraint for cumulative problems with over-loads of resource. In *CP Doctoral Programme*, pages 49–55, 2010.

[Demassey *et al.*, 2006] S. Demassey, G. Pesant, and L.-M. Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006.

[Hellsten, 2004] L. Hellsten. Consistency propagation for *stretch* constraints. Master's thesis, Waterloo, 2004.

[Pachet and Roy, 1999] F. Pachet and P. Roy. Automatic generation of music programs. In *CP*, volume 1713 of *LNCS*, pages 331–345, 1999.

[Pesant, 2001] G. Pesant. A Filtering Algorithm for the *Stretch* Constraint. In *CP*, volume 2239 of *LNCS*, pages 183–195, 2001.

[Pesant, 2004] G. Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. In *CP*, volume 3258 of *LNCS*, pages 482–495, 2004.

[Van Hentenryck *et al.*, 1992] P. Van Hentenryck, Y. Deville, and C.-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57(2-3):291–321, 1992.