Propagating Regular Counting Constraints

Nicolas Beldiceanu
TASC team (CNRS/INRIA)
Mines de Nantes
44307 Nantes, France

Pierre Flener and **Justin Pearson**
Department of Information Technology
Uppsala University
751 05 Uppsala, Sweden

Pascal Van Hentenryck
Optimization Research Group, NICTA
and Australian National University
Canberra, Australia

Abstract

Constraints over finite sequences of variables are ubiquitous in sequencing and timetabling. This led to general modelling techniques and generic propagators, often based on deterministic finite automata (DFA) and their extensions. We consider counter-DFAs (cDFA), which provide concise models for regular counting constraints, that is constraints over the number of times a regular-language pattern occurs in a sequence. We show how to enforce domain consistency in polynomial time for *at-most* and *at-least* regular counting constraints based on the frequent case of a cDFA with only accepting states and a single counter that can be increased by transitions. We also show that the satisfaction of *exact* regular counting constraints is NP-hard and that an incomplete propagator for *exact* regular counting constraints is faster and provides more pruning than the existing propagator from (Beldiceanu, Carlsson, and Petit 2004). Finally, by avoiding the unrolling of the cDFA used by COSTREGULAR, the space complexity reduces from $O(n \cdot |\Sigma| \cdot |Q|)$ to $O(n \cdot (|\Sigma| + |Q|))$, where Σ is the alphabet and Q the state set of the cDFA.

1 Introduction

Constraints over finite sequences of variables arise in many sequencing and timetabling applications. The last decade has witnessed significant research on how to model and propagate, in a generic way, idiosyncratic constraints that are often featured in these applications. The resulting modelling techniques are often based on formal languages and, in particular, deterministic finite automata (DFA). Indeed, DFAs are a convenient tool to model a wide variety of constraints, and their associated propagators can enforce domain consistency in polynomial time (Beldiceanu, Carlsson, and Petit 2004; Pesant 2004).

This paper is concerned with the concept of counter-DFA (cDFA), an extension of DFAs initially proposed in (Beldiceanu, Carlsson, and Petit 2004), and uses it to model regular counting constraints, that is constraints on the number of regular-language patterns occurring in a sequence of variables. cDFAs typically result in more concise and natural encodings of regular counting constraints compared to DFAs, but, to our knowledge, there is no published proof that they do not admit efficient propagators enforcing domain consistency. This paper originated as an attempt to clarify this issue and to overcome the practical limitation of current filtering algorithms due to a large memory consumption stemming from the explicit unrolling of the automaton (Pesant 2004; Demassey, Pesant, and Rousseau 2006). We consider the subset of cDFAs satisfying two conditions: (1) all their states are accepting, and (2) they manipulate a single counter that can be increased by transitions. These conditions are satisfied for many regular counting constraints and offer a good compromise between expressiveness and efficiency.

Our first contribution is to show that for such a cDFA $\mathcal{A}$ it is possible to enforce domain consistency efficiently on *at-most* and *at-least* regular counting constraints. The constraint REGCOUNTATMOST$(N, X, \mathcal{A})$ holds if the counter of $\mathcal{A}$ is at most N after $\mathcal{A}$ has consumed sequence X. The constraint REGCOUNTATLEAST$(N, X, \mathcal{A})$ is defined similarly. We also show the NP-hardness of feasibility testing for the constraint REGCOUNT$(N, X, \mathcal{A})$, which holds if the counter of $\mathcal{A}$ is exactly N after $\mathcal{A}$ has consumed X. Compared to the constraint COSTREGULAR (Demassey, Pesant, and Rousseau 2006), as generalised for the *Choco* solver (Choco 2012), our second contribution is a propagator for exact regular counting that uses asymptotically less space (for its internal data structures) and yet propagates more on the variables of X. Furthermore, our propagators for *at-most* and *at-least* regular counting achieve domain consistency on the counter variable N (and X) in the same asymptotic time as the COSTREGULAR propagator achieves only bounds consistency on N (but also domain consistency on X).

The rest of the paper is organised as follows. Section 2 defines cDFAs and regular counting constraints. Section 3 gives the propagator, its complexity, and its evaluation. Section 4 discusses related work and Section 5 concludes.

2 Background

Deterministic Finite Counter Automata

Recall that a *deterministic finite automaton* (DFA) is a tuple $\langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is the set of states, Σ is the alphabet, $\delta \colon Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of accepting states.

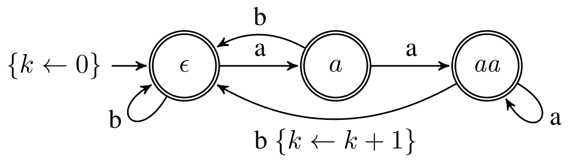

Figure 1: Counter-DFA $\mathcal{AAB}$ for the constraint NUMBERWORD(N, X, "aab")

This paper considers a subclass of counter-DFAs in which all states are accepting and only one counter is used. The counter is initialised to 0 and increases by a given natural number at every transition. Such an automaton accepts every string and assigns a value to its counter. More formally, a *counter-DFA* (cDFA) is here specified as a tuple $\langle Q, \Sigma, \delta, q_0, F\rangle$, where Q, Σ, q_0, and F are as in a DFA except that $F = Q$ and the DFA transition function δ is extended to the signature $Q \times \Sigma \rightarrow Q \times \mathbb{N}$, so that $\delta(q, \ell) = \langle r, inc\rangle$ indicates that r is *the* successor state of state q upon reading alphabet symbol ℓ and that the counter is increased by inc. (All our results generalise to $inc \in \mathbb{R}^+$, that is *weighted* regular counting.) We also define two projections of this extended transition function: if $\delta(q, \ell) = \langle r, inc\rangle$, then $\delta_{\mathrm{Q}}(q, \ell) = r$ and $\delta_{\mathbb{N}}(q, \ell) = inc$. Given $\delta(q, \ell) = \langle r, inc\rangle$, we denote by $\mathcal{C}(q \xrightarrow{\ell} r)$ the counter increase inc of transition $q \xrightarrow{\ell} r$ from state q to state r upon consuming symbol ℓ. Similarly, we denote by $\mathcal{C}(q \overset{\sigma}{\rightsquigarrow} r)$ the counter increase of a path $q \overset{\sigma}{\rightsquigarrow} r$ from state q to state r upon consuming a (possibly empty) string σ.

Example 1. Consider the automaton $\mathcal{AAB}$ in Figure 1. It represents a cDFA with state set $Q = \{\epsilon, a, aa\}$ and alphabet $\Sigma = \{\mathrm{a}, \mathrm{b}\}$. The transition function δ is given by the labelled arcs between states, and the start state is $q_0 = \epsilon$ (indicated by an arc coming from no state; we often denote the start state by ϵ, because it can be reached by consuming the empty string ϵ). Since the final states F are all the states in Q, this automaton recognises every string over $\{\mathrm{a}, \mathrm{b}\}$ and is thus by itself not very interesting. However, the cDFA features a counter k that is initialised to 0 at the start state, increased by 1 on the transition from state aa to state ϵ upon reading symbol 'b', and increased by 0 on all other transitions. As a result, the final value of k is the number of occurrences of the word "aab" within the string.

Regular Counting Constraints

A *regular counting constraint* is defined as a constraint that can be modelled by a cDFA. The REGCOUNT($N, X, \mathcal{A}$) constraint holds if the value of variable N, called the *counter variable*, is equal to the final value of the counter after cDFA $\mathcal{A}$ has consumed the values of the entire sequence X of variables. Consider the constraint NUMBERWORD(N, X, w), which holds if N is the number of occurrences of the non-empty word w in the sequence X of variables. The constraint NUMBERWORD(N, X, "aab") can be modelled by the constraint REGCOUNT($N, X, \mathcal{AAB}$) with the automaton $\mathcal{AAB}$ specified in Figure 1.

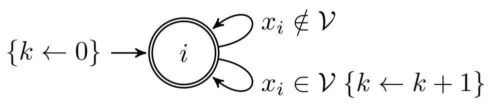

Figure 2: Counter-DFA $\mathcal{AMONG}$ for the constraint AMONG($N, X, \mathcal{V}$)

Signature Constraints

A constraint on a sequence X of variables can sometimes be modelled with the help of a DFA or cDFA that operates not on X, but on a sequence of *signature variables* that functionally depend via *signature constraints* on a sliding window of variables within X (Beldiceanu, Carlsson, and Petit 2004).

For example, the AMONG($N, X, \mathcal{V}$) constraint (Beldiceanu and Contejean 1994) requires N to be the number of variables in the sequence X that are assigned a value from the given set $\mathcal{V}$. With signature constraints $x_i \in \mathcal{V} \Leftrightarrow s_i = 1$ and $x_i \notin \mathcal{V} \Leftrightarrow s_i = 0$ (with $x_i \in X$), we obtain a sequence of $|X|$ signature variables s_i that can be used in a cDFA that counts the number of occurrences of value 1 in that sequence. (The choice of AMONG is purely pedagogical: we do not claim this is the best way to model and propagate this constraint.) Rather than labelling the transitions of such a cDFA with *values* of the domain of the signature variables (the set $\{0, 1\}$ here), we label them with the corresponding *conditions* of the signature constraints, giving the cDFA in Figure 2.

If each signature variable depends on a sliding window of size 1 within X (as for AMONG), then the signature constraints are *unary*. Our results also apply to cDFAs with unary signature constraints because a network of a REGCOUNTATMOST constraint and unary signature constraints is Berge-acyclic.

3 The Propagator

Feasibility Test and Domain Consistency Filtering

Our propagator is defined in terms of the following concepts, which assume a sequence $x_1, \ldots, x_n$ of variables with domains denoted by $\mathrm{dom}(x_i)$:

- Define $\underline{\mathrm{P}}(i)$ (respectively $\overline{\mathrm{P}}(i)$) to be the set of pairs $\langle q, c\rangle$ where c is the minimum (respectively maximum) counter increase (or value) after the automaton consumes any *prefix* string $\sigma = \sigma_1 \cdots \sigma_i$ from state q_0 to reach state q, with $i \in [0, n]$ and $\sigma_j \in \mathrm{dom}(x_j)$ for each $j \in [1, i]$.
- Define $\underline{\mathrm{S}}(i)$ (respectively $\overline{\mathrm{S}}(i)$) to be the set of pairs $\langle q, c\rangle$ where c is the minimum (respectively maximum) counter increase after the automaton consumes any *suffix* string $\sigma = \sigma_i \cdots \sigma_n$ from state q to reach a state appearing in $\underline{\mathrm{P}}(n)$ (respectively $\overline{\mathrm{P}}(n)$), with $i \in [1, n+1]$ and $\sigma_j \in \mathrm{dom}(x_j)$ for each $j \in [i, n]$.

Example 2. By illustrating one representative of these four quantities, we show that we have to maintain the maximum

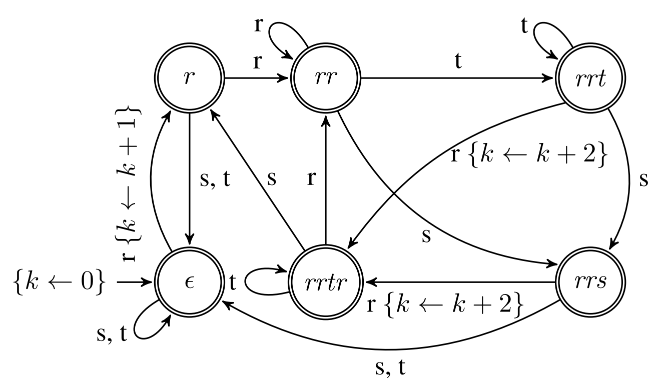

Figure 3: Counter-DFA $\mathcal{RST}$

counter value *for every state* reachable from q_0 in i steps, rather than just maintaining the overall maximum counter value and the set of states reachable from q_0 in i steps. Consider the automaton $\mathcal{RST}$ in Figure 3, where q_0 is ϵ.

In a sequence of $n = 6$ variables $x_1, \ldots, x_6$ that must be assigned value 'r' or 't', we have:

$$\begin{aligned}
\overline{\mathrm{P}}(0) &= \{\langle \epsilon, 0\rangle\}\\
\overline{\mathrm{P}}(1) &= \{\langle \epsilon, 0\rangle, \langle r, 1\rangle\}\\
\overline{\mathrm{P}}(2) &= \{\langle \epsilon, 1\rangle, \langle r, 1\rangle, \langle rr, 1\rangle\}\\
\overline{\mathrm{P}}(3) &= \{\langle \epsilon, 1\rangle, \langle r, 2\rangle, \langle rr, 1\rangle, \langle rrt, 1\rangle\}\\
\overline{\mathrm{P}}(4) &= \{\langle \epsilon, 2\rangle, \langle r, 2\rangle, \langle rr, 2\rangle, \langle rrt, 1\rangle, \langle rrtr, 3\rangle\}\\
\overline{\mathrm{P}}(5) &= \{\langle \epsilon, 2\rangle, \langle r, 3\rangle, \langle rr, 3\rangle, \langle rrt, 2\rangle, \langle rrtr, 3\rangle\}\\
\overline{\mathrm{P}}(6) &= \{\langle \epsilon, 3\rangle, \langle r, 3\rangle, \langle rr, 3\rangle, \langle rrt, 3\rangle, \langle rrtr, 4\rangle\}
\end{aligned}$$

Indeed, $\langle rrtr, 4\rangle \in \overline{\mathrm{P}}(6)$ because $\langle rrt, 2\rangle \in \overline{\mathrm{P}}(5)$ and there is a transition in $\mathcal{A}$ from rrt to $rrtr$ on symbol 'r' with a counter increase of 2, even though three states have a higher counter value (namely 3) than rrt in $\overline{\mathrm{P}}(5)$.

A decomposition using sliding transition constraints where the state and counter variables are not explicitly linked, such as in (Beldiceanu, Carlsson, and Petit 2004), has no link between the reached state at level i and the corresponding counter value, and thus hinders propagation.

To compute $\underline{\mathrm{P}}(i)$ and $\underline{\mathrm{S}}(i)$, we need an operation that takes a set of state-and-integer pairs and keeps only the pairs $\langle q, c\rangle$ where there is no pair $\langle q, c'\rangle$ with $c' < c$. Formally, $\mathrm{trimMin}(S) = \{\langle q, c\rangle \in S \mid \nexists \langle q, c'\rangle \in S : c' < c\}$. For brevity, we use $\underset{\phi(q,c)}{\mathrm{trimMin}}(\langle q, c\rangle)$ to denote $\mathrm{trimMin}(\{\langle q, c\rangle \mid \phi(q, c)\})$, for any condition ϕ. We inductively define $\underline{\mathrm{P}}(i)$ and $\underline{\mathrm{S}}(i)$ as follows:

$$\underline{\mathrm{P}}(i) = \begin{cases} \{\langle q_0, 0\rangle\} & \text{if } i = 0\\ \underset{\substack{\langle q,c\rangle \in \underline{\mathrm{P}}(i-1)\\ \ell \in \mathrm{dom}(x_i)}}{\mathrm{trimMin}} (\langle \delta_{\mathrm{Q}}(q,\ell),\ c + \delta_{\mathbb{N}}(q,\ell)\rangle) & \text{if } i \in [1, n] \end{cases}$$

$$\underline{\mathrm{S}}(i) = \begin{cases} \{\langle q, 0\rangle \mid \exists c \in \mathbb{N} : \langle q, c\rangle \in \underline{\mathrm{P}}(n)\} & \text{if } i = n+1\\ \underset{\substack{\langle q',c'\rangle \in \underline{\mathrm{S}}(i+1)\\ \ell \in \mathrm{dom}(x_i)\\ \delta(q,\ell) = \langle q', inc\rangle}}{\mathrm{trimMin}} (\langle q,\ c' + inc\rangle) & \text{if } i \in [1, n] \end{cases}$$

We show that the inductively computed quantities correspond to the definitions of $\underline{\mathrm{P}}(i)$ and $\underline{\mathrm{S}}(i)$. First consider $\underline{\mathrm{P}}(i)$. The base case $\underline{\mathrm{P}}(0)$ follows from the initialisation to zero of the counter. By induction, suppose the set $\underline{\mathrm{P}}(i-1)$ is correct. Before applying trimMin, the set contains all pairs obtained upon reading the symbol ℓ starting from some pair $\langle q, c\rangle$ in $\underline{\mathrm{P}}(i-1)$, where c is the minimum counter value for q over sequences of length $i-1$. The trimMin operation then filters out all the pairs $\langle q', c'\rangle$ with non-minimum counter value for q'. The correctness proof for $\underline{\mathrm{S}}(i)$ is similar.

We define the REGCOUNTATMOST$(N, X, \mathcal{A})$ propagator. The propagator for REGCOUNTATLEAST is similar. The following theorem gives a feasibility test.

Theorem 1. *A* REGCOUNTATMOST$(N, [x_1, \ldots, x_n], \mathcal{A})$ *constraint has a solution iff the minimum value of the counter of $\mathcal{A}$ after consuming the entire sequence is at most the maximum of the domain of N:*

$$\min_{\langle q,c\rangle \in \underline{\mathrm{P}}(n)} c \le \max(\mathrm{dom}(N))$$

Proof. Suppose $\underline{c}$ is the minimum counter value such that $\langle q, \underline{c}\rangle \in \underline{\mathrm{P}}(n)$ for some state q. By the definition of $\underline{\mathrm{P}}(n)$, there is some sequence $\sigma = \sigma_1 \cdots \sigma_n$ where for all $1 \le j \le n$ the symbol σ_j belongs to $\mathrm{dom}(x_j)$ such that $\mathcal{C}(q_0 \overset{\sigma}{\rightsquigarrow} q) = \underline{c}$. Because each σ_j belongs to the domain of the corresponding variable, we have that σ is a solution to REGCOUNTATMOST iff $\underline{c} \le \max(\mathrm{dom}(N))$. ☐

We now show how to achieve domain consistency (DC).

Theorem 2. *For a* REGCOUNTATMOST$(N, [x_1, \ldots, x_n], \mathcal{A})$ *constraint, define the minimum value of the counter of $\mathcal{A}$ for variable x_i to take value ℓ:*

$$\underline{\mathrm{m}}(i, \ell) = \min_{\substack{\langle q,c\rangle \in \underline{\mathrm{P}}(i-1),\\ q' = \delta_{\mathrm{Q}}(q,\ell),\\ \langle q',c'\rangle \in \underline{\mathrm{S}}(i+1)}} (c + \delta_{\mathbb{N}}(q, \ell) + c')$$

- *A value ℓ in* $\mathrm{dom}(x_i)$ (*with* $i \in [1, n]$) *appears in a solution iff the minimum value of the counter is at most the maximum of the domain of N:* $\underline{\mathrm{m}}(i, \ell) \le \max(\mathrm{dom}(N))$.
- *A value in* $\mathrm{dom}(N)$ *appears in a solution iff it is at least the minimum counter value given in Theorem 1.*

Proof. We start with the first claim. (If) We show that any $\ell \in \mathrm{dom}(x_i)$ with $\underline{\mathrm{m}}(i, \ell) \le \max(\mathrm{dom}(N))$ participates in a solution. Suppose $\underline{\mathrm{m}}(i, \ell)$ equals $\underline{c} + \delta_{\mathbb{N}}(q, \ell) + \underline{c}'$ for some $\langle q, \underline{c}\rangle \in \underline{\mathrm{P}}(i-1)$ and some $\langle q', \underline{c}'\rangle \in \underline{\mathrm{S}}(i+1)$, with $q' = \delta_{\mathrm{Q}}(q, \ell)$. Then there exist two strings $\sigma = \sigma_1 \cdots \sigma_{i-1}$ and $\tau = \sigma_{i+1} \cdots \sigma_n$ and some state q_n such that

$$\mathcal{C}(q_0 \overset{\sigma}{\rightsquigarrow} q) = \underline{c} \text{ and } \mathcal{C}(q' \overset{\tau}{\rightsquigarrow} q_n) = \underline{c}'$$

with $\sigma_j \in \mathrm{dom}(x_j)$ for all $j \in [1, n]$. Note that the length of $\sigma\ell\tau$ is n. We have:

$$\begin{aligned}
\mathcal{C}(q_0 \overset{\sigma\ell\tau}{\rightsquigarrow} q_n) &= \mathcal{C}(q_0 \overset{\sigma}{\rightsquigarrow} q) + \delta_{\mathbb{N}}(q, \ell) + \mathcal{C}(q' \overset{\tau}{\rightsquigarrow} q_n)\\
&= \underline{c} + \delta_{\mathbb{N}}(q, \ell) + \underline{c}'\\
&= \underline{\mathrm{m}}(i, \ell) \le \max(\mathrm{dom}(N)).
\end{aligned}$$

Hence the assignment corresponding to $\sigma\ell\tau$ satisfies the domains and the constraint, so $\ell \in \mathrm{dom}(x_i)$ participates in a solution. (Only if) If $\ell \in \mathrm{dom}(x_i)$ participates in a solution, then the counter of that solution is at least $\underline{\mathrm{m}}(i,\ell)$ and at most $\max(\mathrm{dom}(N))$, hence $\underline{\mathrm{m}}(i,\ell) \le \max(\mathrm{dom}(N))$.

The second claim follows from Theorem 1. Indeed, let $\underline{c} = \min_{\langle q,c\rangle \in \underline{\mathrm{P}}(n)} c$ So there exists a sequence $\sigma = \sigma_1 \cdots \sigma_n$ with each $\sigma_j \in \mathrm{dom}(x_j)$ such that $\mathcal{C}(q_0 \overset{\sigma}{\leadsto} q) = \underline{c}$. Further, for any $\sigma' = \sigma'_1 \cdots \sigma'_n$ with each $\sigma'_j \in \mathrm{dom}(x_j)$ for all $j \in [1,n]$, we have $\underline{c} \le \mathcal{C}(q_0 \overset{\sigma'}{\leadsto} q'_n)$ for some state q'_n. So, by Theorem 1, we need to prove that $v \in \mathrm{dom}(N)$ participates in a solution iff $\underline{c} \le v$. (Only if) If $v \in \mathrm{dom}(N)$ participates in a solution, then there exists a sequence $\sigma' = \sigma'_1 \cdots \sigma'_n$ such that each $\sigma'_j \in \mathrm{dom}(x_j)$ and $\mathcal{C}(q_0 \overset{\sigma'}{\leadsto} q'_n) \le v$. Since $\underline{c} \le \mathcal{C}(q_0 \overset{\sigma'}{\leadsto} q'_n)$, we have $\underline{c} \le v$. (If) If $\underline{c} \le v$, then the sequence σ above necessarily also forms a solution with $N = v$. ☐

The algorithm of an idempotent propagator consists of the feasibility test of Theorem 1 and the pruning of the values from $\mathrm{dom}(N)$ and all $\mathrm{dom}(x_i)$ that do not satisfy the conditions of Theorem 2, upon first computing its $\underline{\mathrm{m}}(i,\ell)$ values.

Complexity

The complexity of a non-incremental implementation of the propagator is established as follows. Recall that we consider sequences of n variables x_i, each with at most the automaton alphabet Σ as domain. Let the automaton have $|Q|$ states. Each set $\underline{\mathrm{P}}(i)$ has $O(|Q|)$ elements and takes $O(|\Sigma| \cdot |Q|)$ time to construct and trim (assuming it is implemented as a counter-value array indexed by Q, with all cells initialised to $+\infty$). There are $n+1$ such sets, hence the entire $\underline{\mathrm{P}}(\cdot)$ vector takes $O(n \cdot |\Sigma| \cdot |Q|)$ time and $\Theta(n \cdot |Q|)$ space. Similarly, the entire $\underline{\mathrm{S}}(\cdot)$ vector takes $O(n \cdot |\Sigma| \cdot |Q|)$ time and $\Theta(n \cdot |Q|)$ space. Each value $\underline{\mathrm{m}}(i,\ell)$ takes $O(|Q|)$ time to construct, since at most $|Q|$ pairs $\langle q,c\rangle$ of $\underline{\mathrm{P}}(i-1)$ are iterated over and the corresponding pair $\langle q',c'\rangle$ is unique and can be retrieved in constant time (under the assumed data structure). There are $n \cdot |\Sigma|$ such values, hence the entire $\underline{\mathrm{m}}(\cdot,\cdot)$ matrix takes $O(n \cdot |\Sigma| \cdot |Q|)$ time and $\Theta(n \cdot |\Sigma|)$ space. Each test of a domain value takes constant time, hence $\Theta(n+1)$ time in total for the n variables x_i and the counter variable N. In total, such an implementation takes $O(n \cdot |\Sigma| \cdot |Q|)$ time, and $\Theta(n \cdot (|Q| + |\Sigma|))$ space.

The Exact Regular Counting Constraint

Not surprisingly, decomposing RegCount$(N, X, \mathcal{A})$ into the conjunction of RegCountAtMost$(N, X, \mathcal{A})$ and RegCountAtLeast$(N, X, \mathcal{A})$ does not yield DC at the fixpoint of their propagators: for the cDFA $\mathcal{B}$ in Figure 4 and the constraint RegCount$(N, [2, x, 2], \mathcal{B})$, with $N \in \{0, 1, 2\}$ and $x \in \{1, 2\}$, it misses the inference of $N \neq 1$. Worse, achieving DC on exact regular counting is NP-hard:

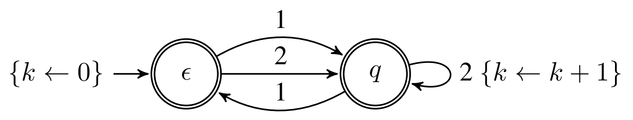

Figure 4: Counter-DFA $\mathcal{B}$, where domain consistency is not achieved for exact regular counting

Theorem 3. *The feasibility of* RegCount *is NP-hard.*

Proof. By reduction from Subset-Sum. Consider an instance $\langle \{a_1, \ldots, a_k\}, s\rangle$ of Subset-Sum, which holds if there is a subset $A \subseteq \{a_1, \ldots, a_k\}$ such that $\sum_{v \in A} v = s$. Construct a cDFA $\mathcal{A}$ with one state and alphabet $\Sigma = \{a_1, \ldots, a_k, 0\}$. A transition labelled by a_i increases the counter by a_i, and the transition labelled by 0 does not increase the counter. Build a sequence of variables $X = \langle x_1, \ldots, x_k\rangle$ such that $\mathrm{dom}(x_i) = \{0, a_i\}$ and a variable N such that $\mathrm{dom}(N) = \{s\}$. Such a reduction can be done in polynomial time. Subset-Sum holds iff RegCount$(N, X, \mathcal{A})$ holds. ☐

The propagator for RegCountAtMost can be generalised into an incomplete propagator for RegCount. A value ℓ is removed from the domain of variable x_i if the following condition holds for all $\langle q, \underline{c}\rangle \in \underline{\mathrm{P}}(i-1)$:

$$\bigwedge_{\substack{\langle q,\overline{c}\rangle \in \overline{\mathrm{P}}(i-1)\\ q' = \delta_{\mathrm{Q}}(q,\ell)\\ \langle q',\underline{c'}\rangle \in \underline{\mathrm{S}}(i+1)\\ \langle q',\overline{c'}\rangle \in \overline{\mathrm{S}}(i+1)}} \begin{array}{c}\left[\, \underline{c} + \delta_{\mathbb{N}}(q,\ell) + \underline{c'},\ \overline{c} + \delta_{\mathbb{N}}(q,\ell) + \overline{c'} \,\right] \\ \cap\, \mathrm{dom}(N) = \emptyset\end{array}$$

This propagator has the same space complexity as RegCountAtMost, but it may need more than one run to achieve idempotency. Indeed, it differs from the previous propagator in that lower and upper bounds have to be calculated for *each* state in $\underline{\mathrm{P}}(i-1)$, and it is possible that some states will give different bounds. Hence the first run of the propagator might not reach idempotency. The propagator is strictly stronger than computing the fixpoint of RegCountAtMost and RegCountAtLeast, because the intersection test with respect to $\mathrm{dom}(N)$ is strictly stronger than the conjunction of the two comparisons on the *at-most* and *at-least* sides: for the cDFA in Figure 4 and the constraint RegCount$(1, [2, x, 1, y, z], \mathcal{B})$, with $x, y, z \in \{1, 2\}$, the RegCount propagator infers $z \neq 2$, whereas the decomposition misses this inference. The RegCount propagator is also incomplete: the counter-example before Theorem 3 for the decomposition also applies to it.

Evaluation

We implemented in *SICStus Prolog* version 4.2.1 (Carlsson, Ottosson, and Carlson 1997) the described propagators for RegCountAtMost, RegCountAtLeast, and RegCount. The full source code is in (Beldiceanu et al. 2013). We evaluated their merits as follows.

We generated random cDFAs of up to five states (it is very important to note that all 34 counter automata of the *Global Constraint Catalogue* (Beldiceanu et al. 2007) have at most five states, since counters are a very powerful device that allows a drastic reduction from the number of states needed by using a conventional DFA) using the random DFA generator (Almeida, Moreira, and Reis 2007) of *FAdo* (version 0.9.6) and doing a counter increase by 1 on each arc with a probability of 20%. For each random cDFA, we generated random instances, with random lengths (up to $n = 10$) of $X = [x_1, \ldots, x_n]$ and random initial domains of the counter variable N (one value, two values, and intervals of length 2 or 3) and the signature variables s_i (intervals of any length, and sets with holes).

The results, upon many millions of random instances, are that our REGCOUNT propagator never propagates less but often more, to the point of detecting more failures, than the built-in AUTOMATON (Beldiceanu, Carlsson, and Petit 2004) of SICStus Prolog. Further, it is already up to 3 times faster than the latter, even though it is naïvely implemented in Prolog, while the built-in works by decomposition into a conjunction of TABLE constraints (with tuples according to the transition function δ), which is very carefully implemented in *C*. There is thus strong reason to believe that our propagators will do better on *any* benchmark. Also, no counterexample to the domain consistency of REGCOUNTATMOST has been generated and no pruning by the propagators of actually supported values was observed, giving a sanity check while proving Theorems 1 and 2.

Table 1 gives the cumulative runtimes (under Mac OS X 10.7.5 on a 2.8 GHz Intel Core 2 Duo with a 4 GB RAM), the numbers of detected failures, and (when both propagators succeed) the numbers of pruned values for random instances of some constraints, the four-state cDFA for the NUMBERWORD$(N, X,$ "toto") constraint being unnecessary to reproduce here.

To demonstrate the power of our propagators, we have also tested them on constraints whose counter-DFAs have *binary* signature constraints, so that our *at-least* and *at-most* regular counting propagators may not achieve domain consistency, because they were designed for unary signature constraints. For example, the INFLEXION(N, X) constraint holds if there are N inflexions (local optima) in the integer sequence X; a cDFA is given in (Beldiceanu et al. 2007), with signature constraints using the predicates $x_i\{<, =, >\}x_{i+1}$ on the sliding window $[x_i, x_{i+1}]$ of size 2. Our exact regular counting propagator outperforms the built-in AUTOMATON (Beldiceanu, Carlsson, and Petit 2004) of SICStus Prolog, as shown in the last line of Table 1. Further, our instance generator has not yet constructed any counterexample to domain consistency on *at-most* regular counting.

4 Related Work

Our regular counting constraints are related to COSTREGULAR$(X, \mathcal{A}, N, C)$ (Demassey, Pesant, and Rousseau 2006), an extension of the REGULAR$(X, \mathcal{A})$ constraint (Pesant 2004): a ground instance holds if the sum of the variable-value assignment costs is exactly N after DFA $\mathcal{A}$ has accepted X, where the two-dimensional cost matrix C, indexed by Σ and X, gives the costs of assigning each value of the alphabet Σ of $\mathcal{A}$ to each variable of the sequence X. Indeed, both the abstractions and the underlying propagators of regular counting constraints and COSTREGULAR are closely related. However, we now argue that regular counting constraints sometimes provide both a more natural abstraction and some computational benefits, namely more propagation and asymptotically less space, within the same asymptotic time.

At the *conceptual level*, the regular counting and COSTREGULAR constraints differ in how costs are expressed. In the COSTREGULAR constraint the costs are associated with variable-value assignments, while in regular counting constraints the costs (seen as counter increases) are associated with the transitions of the counter automaton. This is an important conceptual distinction, as counter automata provide a more natural and compact abstraction for a variety of constraints, where the focus is on counting rather than costing. Footnote 1 of (Demassey, Pesant, and Rousseau 2006, page 318) points out that the cost matrix C can be made three-dimensional, indexed also by the states Q of $\mathcal{A}$, but this is not discussed further in (Demassey, Pesant, and Rousseau 2006). This allows the expression of costs on transitions, and it seems that this has no impact on the time complexity of their propagator. This generalisation is implemented in the *Choco* solver (Choco 2012). It is only with such a three-dimensional cost matrix that it is possible for the modeller to post a regular counting constraint by using the COSTREGULAR constraint: first unroll the counter automaton for the length $|X|$ into a directed acyclic weighted graph G (as described in (Pesant 2004), and the counter increases become the weights) and then post COSTREGULAR(X, N, G); the *Choco* implementation (Choco 2012, page 95) of COSTREGULAR features this option. The alternative is to read the three-dimensional cost matrix C off $\mathcal{A}$ only (since X is not needed) and to post COSTREGULAR$(X, \mathcal{A}', N, C)$, where DFA $\mathcal{A}'$ is counter-DFA $\mathcal{A}$ stripped of its counter increases. Either way, this encoding is not particularly convenient and it seems natural to adopt counter automata as an abstraction. Also note that the cost matrix C has to be computed for every different value of $|X|$ that occurs in the problem model, while this is not the case with counter automata. Essentially, regular counting is a specialisation of the generalised COSTREGULAR constraint (with a three-dimensional cost matrix), obtained by projecting the generalised cost matrix onto two different dimensions than in the original COSTREGULAR constraint, namely Q and Σ, and using it to extend the transition function of the DFA to the signature $Q \times \Sigma \rightarrow Q \times \mathbb{N}$ and calling the extended DFA a cDFA.

At the *efficiency level*, regular counting constraints are propagated using dynamic programming, like COSTREGULAR. This is not surprising. The *time* complexity is the same as for the encoding using COSTREGULAR, namely $O(n \cdot |\Sigma| \cdot |Q|)$, where $n = |X|$, as the unrolling of the cDFA takes the same time as the propagator itself. It is interesting however to note that the structure of regular counting constraints enables a better *space* complexity thanks to the compactness of a counter automaton as the input data structure, as well as fundamentally different internal data structures: we do *not* store the unrolled automaton. Indeed, we have shown that regular counting constraints have a space complexity of $\Theta(n \cdot (|Q| + |\Sigma|))$, while the encoding by COSTREGULAR constraint takes

		seconds		failures		prunings	
Constraint	#instances	REGCOUNT	AUTO	REGCOUNT	AUTO	REGCOUNT	AUTO
AMONG$(N, X, \mathcal{V})$	4,400	0.8	1.8	2,241	2,241	3,749	3,749
NUMBERWORD$(N, X,$ “aab”$)$	13,200	0.9	2.3	4,060	4,020	1,294	943
NUMBERWORD$(N, X,$ “toto”$)$	17,600	0.9	2.7	4,446	4,435	1,149	663
REGCOUNT$(N, X, \mathcal{RST})$	13,200	3.9	7.3	5,669	4,333	13,213	2,275
INFLEXION(N, X)	13,200	3.7	7.5	4,447	4,066	9,279	4,531

Table 1: Comparison between REGCOUNT and the AUTO(MATON) constraint of SICStus Prolog

$\Theta(n \cdot |\Sigma| \cdot |Q|)$ space, to store either the three-dimensional cost matrix C or the unrolled graph G (*Choco* allows both ways of parametrising COSTREGULAR). Note that adding a counter to a DFA bears no asymptotic space overhead on the representation of the DFA.

At the *consistency level*, note that our *at-most* and *at-least* regular counting propagators achieve domain consistency on the counter variable N in the same asymptotic time as the COSTREGULAR propagator (Demassey, Pesant, and Rousseau 2006; Choco 2012) achieves only bounds consistency on N. The claim by (Demassey, Pesant, and Rousseau 2006; Choco 2012) that their propagator achieves domain consistency on X is invalidated by Theorem 3 (stressing the need for propagator sanity checks like ours, even with random instances), hence this would only hold for *at-most* and *at-least* variants of COSTREGULAR: for the cDFA $\mathcal{B}$ in Figure 4 and the constraint REGCOUNT$(N, [2, 2, x, 2, y], \mathcal{B})$, with $N \in \{1, 3\}$ and $x, y \in \{1, 2\}$, their propagator misses the inference of $y \neq 2$, and so does our propagator for exact regular counting. Our data structures are more compact (see above), and yet enable more propagation on X for exact regular counting.

To summarise, although regular counting constraints and the COSTREGULAR constraint are closely related, we believe that our results contribute both to our understanding of these constraints and to the practice in the field.

The SEQBIN constraint (Petit, Beldiceanu, and Lorca 2011; Katsirelos, Narodytska, and Walsh 2012) can be represented by a regular counting constraint, but it would require non-unary signature constraints.

The METER constraint (Roy and Pachet 2013) does not take an automaton but constraints on two consecutive variables. It has only variables: the cost of a variable is its value, while counter increases or costs are distinguished from the variables for both REGCOUNT and COSTREGULAR. It does not provide a counter variable but allows partial sum constraints. It achieves domain consistency in pseudo-polynomial time.

5 Conclusion

We consider regular counting constraints over finite variable sequences, which are ubiquitous and very diverse in sequencing and timetabling (e.g., restricting the number of monthly working weekends or two-day-periods where a nurse works during a night followed by an afternoon). We study a class of deterministic finite automata with counters (cDFA) that allows more concise models for regular counting constraints than representations using standard DFAs.

Our first contribution is to show how to enforce domain consistency in polynomial time for *at-most* and *at-least* regular counting constraints, even for the counter variable, based on the frequent case of a cDFA with only accepting states and a single counter that can be increased by transitions. We also show that deciding the feasibility of *exact* regular counting constraints is NP-hard. Our second contribution is to reduce the space complexity from $O(n \cdot |\Sigma| \cdot |Q|)$ to $O(n \cdot (|\Sigma| + |Q|))$ by not explicitly representing the unrolled cDFA.

It is possible to lift our restriction to counter automata where all states are accepting, even though we are then technically outside the realm of regular counting. For instance, this would allow us to constrain the number N of occurrences of some pattern, recognised by cDFA $\mathcal{A}_1$, in a sequence X of variables, while X is not allowed to contain any occurrence of another pattern, recognised by cDFA $\mathcal{A}_2$. Rather than decomposing this constraint into the conjunction of REGCOUNT$(N, X, \mathcal{A}_1)$ and REGCOUNT$(0, X, \mathcal{A}_2)$, with poor propagation through the shared variables, we can design a cDFA $\mathcal{A}_{12}$ that counts the number of occurrences of the first pattern and *fails* at any occurrence of the second pattern (instead of *counting* them) and post the unique constraint REGCOUNT$(N, X, \mathcal{A}_{12})$, after using the following recipe. Add an accepting state, say q, and an alphabet symbol, say \$, whose meaning is end-of-string. Add transitions on \$ from all existing accepting states to q, with counter increase by zero. Add transitions on \$ from all non-accepting states to q, with counter increase by a suitably large number, such as $\max(\text{dom}(N)) + 1$. Make the non-accepting states accepting. Append the symbol \$ to the sequence X when posting the constraint, so that the extended automaton never actually stops in a state different from q, thereby making it irrelevant whether the original states are accepting or not.

Open issues include the following questions. Can we implement our propagators to run in $O(n \cdot |\Sigma|)$ time? Which cDFAs admit a propagator achieving domain consistency for *exact* regular counting? Can we generalise our domain-consistency result to non-unary signature constraints? We conjecture our results extend to non-deterministic counter automata: our notation was merely simplified by requiring that the transition function δ_{Q} return a single state.

Other constraints, such as the one of (Barták 2002), can be used to encode counters: a comparison is future work.

Since our propagator essentially uses linear constraints, we strongly believe it is also relevant in the context of linear programming, as in (Côté, Gendron, and Rousseau 2007).

Acknowledgements

The first author is supported by INRIA's associate-team programme TASCMELB. The second and third authors are supported by grants 2011-6133 and 2012-4908 of the Swedish Research Council (VR). NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. We wish to thank Mats Carlsson for help with SICStus Prolog, Joseph Scott for help with FAdo, and Arnaud Letort for help with performing the test with COSTREGULAR in *Choco*. We also thank the reviewers for their constructive comments.

References

Almeida, M.; Moreira, N.; and Reis, R. 2007. Enumeration and generation with a string automata representation. *Theoretical Computer Science* 387(2):93–102. The *FAdo* tool is available at `http://fado.dcc.fc.up.pt/`.

Barták, R. 2002. Modelling resource transitions in constraint-based scheduling. In Grosky, W. I., and Plášil, F., eds., *SOFSEM 2002*, volume 2540 of *LNCS*, 186–194. Springer.

Beldiceanu, N., and Contejean, E. 1994. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling* 20(12):97–123.

Beldiceanu, N.; Carlsson, M.; Demassey, S.; and Petit, T. 2007. Global constraint catalogue: Past, present, and future. *Constraints* 12(1):21–62. The current working version of the catalogue is at `http://www.emn.fr/z-info/sdemasse/aux/doc/catalog1.pdf`.

Beldiceanu, N.; Flener, P.; Pearson, J.; and Van Hentenryck, P. 2013. Propagating regular counting constraints. Technical Report 1309.7145, Computing Research Repository. Available at `http://arxiv.org/abs/1309.7145`.

Beldiceanu, N.; Carlsson, M.; and Petit, T. 2004. Deriving filtering algorithms from constraint checkers. In Wallace, M., ed., *CP 2004*, volume 3258 of *LNCS*, 107–122. Springer.

Carlsson, M.; Ottosson, G.; and Carlson, B. 1997. An open-ended finite domain constraint solver. In Glaser, H.; Hartel, P.; and Kuchen, H., eds., *PLILP 1997*, volume 1292 of *LNCS*, 191–206. Springer. The solver is at `http://sicstus.sics.se`.

Choco. 2012. Choco solver: Documentation, version 2.1.5. Available at `http://choco.emn.fr/`.

Côté, M.-C.; Gendron, B.; and Rousseau, L.-M. 2007. Modeling the Regular constraint with integer programming. In Van Hentenryck, P., and Wolsey, L. A., eds., *CPAIOR 2007*, volume 4510 of *LNCS*, 29–43. Springer.

Demassey, S.; Pesant, G.; and Rousseau, L.-M. 2006. A `Cost-Regular` based hybrid column generation approach. *Constraints* 11(4):315–333.

Katsirelos, G.; Narodytska, N.; and Walsh, T. 2012. The SEQBIN constraint revisited. In Milano, M., ed., *CP 2012*, volume 7514 of *LNCS*, 332–347. Springer.

Pesant, G. 2004. A regular language membership constraint for finite sequences of variables. In Wallace, M., ed., *CP 2004*, volume 3258 of *LNCS*, 482–495. Springer.

Petit, T.; Beldiceanu, N.; and Lorca, X. 2011. A generalized arc-consistency algorithm for a class of counting constraints. In *IJCAI 2011*, 643–648. IJCAI/AAAI. Corrected version at `http://arxiv.org/abs/1110.4719`.

Roy, P., and Pachet, F. 2013. Enforcing meter in finite-length Markov sequences. In des Jardins, M., and Littman, M. L., eds., *AAAI 2013*, 854–861. AAAI Press.