Time-Series Constraints: Improvements and Application in CP and MIP Contexts

Ekaterina Arafailova, Nicolas Beldiceanu, Rémi Douence, Pierre Flener, María Andreína Francisco Rodríguez, Justin Pearson, Helmut Simonis

Time-Series Constraints: Improvements and Application in CP and MIP Contexts

Ekaterina Arafailova[1], Nicolas Beldiceanu[1], Rémi Douence[1], Pierre Flener[2], María Andreína Francisco Rodríguez[2], Justin Pearson[2], and Helmut Simonis[3]

[1] TASC/ASCOLA (CNRS/INRIA), Mines Nantes, FR – 44307 Nantes, France
{Ekaterina.Arafailova,Nicolas.Beldiceanu,Remi.Douence}@mines-nantes.fr
[2] Uppsala University, Dept of Information Technology, SE – 751 05 Uppsala, Sweden
{Pierre.Flener,Maria.Andreina.Francisco,Justin.Pearson}@it.uu.se
[3] Insight Centre for Data Analytics, University College Cork, Ireland
Helmut.Simonis@insight-centre.org

Abstract. A checker for a constraint on a variable sequence can often be compactly specified by an automaton, possibly with accumulators, that consumes the sequence of values taken by the variables; such an automaton can also be used to decompose its specified constraint into a conjunction of logical constraints. The inference achieved by this decomposition in a CP solver can be boosted by automatically generated implied constraints on the accumulators, provided the latter are updated in the automaton transitions by linear expressions. Automata with non-linear accumulator updates can be automatically synthesised for a large family of time-series constraints. In this paper, we describe and evaluate extensions to those techniques. First, we improve the automaton synthesis to generate automata with fewer accumulators. Second, we decompose a constraint specified by an automaton with accumulators into a conjunction of linear inequalities, for use by a MIP solver. Third, we generalise the implied constraint generation to cover the entire family of time-series constraints. The newly synthesised automata for time-series constraints outperform the old ones, for both the CP and MIP decompositions, and the generated implied constraints boost the inference, again for both the CP and MIP decompositions. We evaluate CP and MIP solvers on a prototypical application modelled using time-series constraints.

1 Context and Motivation

Frameworks are given in [4,14] for specifying a constraint on a sequence of variables in a high-level way by means of a finite automaton, possibly augmented with accumulators in the framework of [4]. An automaton can be seen as a checker for ground instances of the specified constraint. For example, in a nonogram puzzle, a row constrained to contain two stretches of black cells, of lengths 4 and 3 in this order, separated by at least one white cell but preceded and followed by any amounts of white cells, can be checked by an automaton equivalent to the regular expression $w^*b^4w^+b^3w^*$, where the row is represented by a sequence of variables whose domain value 'w' stands for white and 'b' for black.

Accumulators enable the specification of a constraint γ on a variable sequence X by an automaton whose size does not depend on the length of X: accumulators are initialised at the start state and are updated through the transitions; upon acceptance, the accumulators are linked to another variable of γ via an arithmetic constraint. For example, one could constrain the number of white cells between the two black stretches in the nonogram constraint above to be at most half the length of the row.

The framework of [14] lifts an automaton without accumulators into a propagator for the specified constraint; it maintains domain consistency in polynomial time. The more general framework of [4] lifts an automaton, possibly with accumulators, into a decomposition of the specified constraint in terms of constraints with existing propagators; in the presence of accumulators, this decomposition does not maintain domain consistency in general [2]. Encoding the potential accumulator values in the states of the automaton may lead to an exponentially large automaton. In this paper, we focus on automata with accumulators.

The propagation achieved by the automaton decomposition of [4] in a CP solver can be boosted by invariants, seen as implied constraints, on the accumulators. If the latter are updated in the automaton transitions by linear expressions on the accumulators — such as increments and decrements by constant amounts (as in $c \coloneqq c + 1$) or by other accumulators (as in $c \coloneqq c + r$), or resets (as in $c \coloneqq 0$) — then such implied constraints can be automatically generated [11].

Automata with non-linear accumulator updates can be automatically synthesised for a large family of structural time-series constraints [3]. A time series is here a sequence of integers, corresponding to measurements taken over a time interval. Time series are common in many application areas, such as the power output of electric power stations over multiple days, or environmental data (temperature, humidity, CO_2 level) in buildings. Time series are constrained by physical or organisational limits, which restrict the evolution of the series.

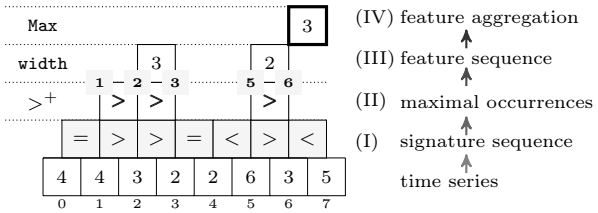
After a summary of the background material in Section 2, the **contributions** and **impact** of this paper are as follows:

- We improve the automated automaton synthesis of [3] so as to synthesise automata with fewer accumulators and simpler accumulator updates, using fewer 'min' and 'max' operators, say (Section 3).
- We decompose a constraint specified by an automaton *with* accumulators into a linear-sized conjunction of linear inequalities, for use by a mixed-integer programming (MIP) solver (Section 4).
- We generalise the implied constraint generation of [11] so as to cover the entire family of time-series constraints of [3] and to rank the generated implied constraints by decreasing propagation strength, thereby easing the human selection of which implied constraints actually to use (Section 5).
- We show that the newly synthesised automata for time-series constraints outperform the automata of [3], for both the CP and MIP decompositions, and that the newly generated implied constraints boost the inference, again for both the CP and MIP decompositions (Section 6).
- We evaluate CP and MIP solvers on a prototypical application modelled with the help of time-series constraints (Section 7).

2 Specifying (Time-Series) Constraints using Automata

We showed in [3] that many constraints $\gamma(N, \langle X_0, \ldots, X_{n-1}\rangle)$ on an unknown time series $\langle X_0, \ldots, X_{n-1}\rangle$ of given length n can be specified as a triple $\langle p, f, g\rangle$, where p is a regular expression over the alphabet $\{<, =, >\}$ and is called the *pattern*; $f \in \{\texttt{max}, \texttt{min}, \texttt{one}, \texttt{range}, \texttt{surface}, \texttt{width}\}$ is called the *feature*; and $g \in \{\texttt{Max}, \texttt{Min}, \texttt{Sum}\}$ is called the *aggregator*. The semantics is that integer variable N is required to be the aggregation, computed using g, of the list of features f of all maximal words matching p within the sequence $\langle S_0, \ldots, S_{n-2}\rangle$ of variables, called the *signature sequence*, which is linked to the time series via the *signature constraints* $(X_i < X_{i+1} \Leftrightarrow S_i = \text{`}<\text{'}) \wedge (X_i = X_{i+1} \Leftrightarrow S_i = \text{`}=\text{'}) \wedge (X_i > X_{i+1} \Leftrightarrow S_i = \text{`}>\text{'})$ for all $i \in [0, n-2]$. A list of 23 patterns was identified, giving 266 constraints. We now introduce our running example.

Example 1. The MaxWidthStrictlyDecreasingSequence(N, X) constraint, requiring N to be the maximum width of the maximal strictly decreasing sequences within the time series X, is specified by the pattern $>^+$, the feature `width`, and the aggregator `Max`. The time series $\langle 4, 4, 3, 2, 2, 6, 3, 5\rangle$ contains two maximal strictly decreasing sequences, namely $4 > 3 > 2$ and $6 > 3$, of widths 3 and 2, so their maximum width is $N = 3$. The following figure shows how to check MaxWidthStrictlyDecreasingSequence$(3, \langle 4, 4, 3, 2, 2, 6, 3, 5\rangle)$ by (I) building the signature sequence by comparing adjacent time-series values; (II) finding all maximal words matching the regular expression $>^+$; (III) computing the feature `width` of each such strictly decreasing sequence; and (IV) aggregating the feature values using the `Max` aggregator:



An *automaton* with a memory of $m \geq 0$ integer accumulators [4] is a tuple $\langle Q, \Sigma, \delta, q_0, I, A, \alpha\rangle$, where Q is the set of *states*, Σ the *alphabet*, $\delta\colon (Q \times \mathbb{Z}^m) \times \Sigma \to Q \times \mathbb{Z}^m$ the *transition function*, $q_0 \in Q$ the *start state*, I the m-tuple of *initial values* of the accumulators, $A \subseteq Q$ the set of *accepting states*, and $\alpha\colon \mathbb{Z}^m \to \mathbb{Z}$ the *acceptance function*, transforming the memory of an accepting state into an integer. If the left-to-right consumption of the symbols of a word w in Σ^* transits from q_0 to some accepting state and the m-tuple C of current accumulator values, then the automaton *returns* the value $\alpha(C)$, else it *fails*.

Example 2. A ground instance of the constraint of Example 1 holds if and only if its value of N is returned by the automaton in Figure 1 after consuming the signature sequence linked to its time series X. The automaton uses $m = 2$ accumulators: at any moment, accumulator c has the length of the *current* strictly decreasing sequence, while r has the length of the *longest* strictly decreasing

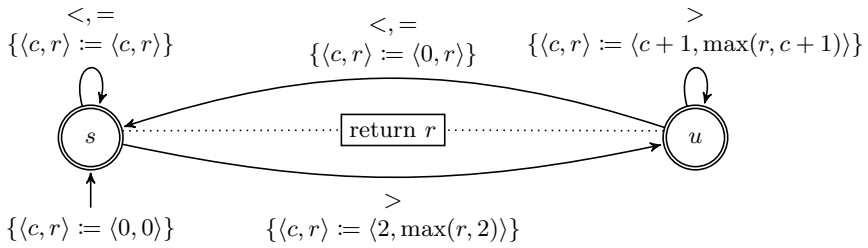


Fig. 1: Automaton for MaxWidthStrictlyDecreasingSequence

sequence seen so far. The state set Q is $\{s, u\}$: at s the current sequence is *not* strictly decreasing, and at u the current sequence *is* strictly decreasing. The start state $q_0 = s$ is indicated by an arc coming from nowhere, annotated within braces by the initialisation to zero of both c and r, hence $I = \langle 0, 0\rangle$. The alphabet Σ is $\{<, =, >\}$. The arc from s to u depicts the transition of δ from s to u upon consuming symbol $>$, and is annotated within braces by accumulator updates: r is updated to its maximum with 2, and c is set to 2. All states are accepting, hence $A = Q$. The acceptance function α transforms a memory $\langle c, r\rangle$ into r at both states, and is given in a box linked to s and u by dotted lines. □

An automaton can be seen as a constraint *checker*. The framework of [14] lifts an automaton with $m = 0$ accumulators into a CP *propagator* for the specified constraint; it maintains domain consistency in time polynomial in the automaton size and sequence length. The more general framework of [4] lifts an automaton with $m \geq 0$ accumulators into a CP *decomposition* of the specified constraint in terms of constraints with existing CP propagators; when $m \geq 1$, this decomposition does not maintain domain consistency in general [2]. Encoding the potential accumulator values in the states of the automaton, so as to get an automaton with $m = 0$ accumulators, may lead to a large automaton.

In this paper, we focus on automata with $m \geq 1$ accumulators, motivated [4] by the wish to specify a constraint on a sequence X by an automaton whose size does not depend on the length of X; this is the case for the automaton in Figure 1. In Section 3, we improve our synthesiser [3] of automata from $\langle p, f, g\rangle$ specifications of time-series constraints, so that it automatically synthesises automata with fewer accumulators and simpler accumulator updates, namely linear accumulator updates rather than updates involving the min and max operators. In Section 4, we lift an automaton with $m \geq 1$ accumulators into a MIP decomposition of linear inequalities. In Section 5, we boost the inference achieved for the CP and MIP decompositions by generalising our generator [11] of constraints implied by an automaton, so that it covers the entire family of time-series constraints of this section and [3]. Those sections are orthogonal and any subset thereof can be read in any sequence.

3 Simplification of Synthesised Time-Series Automata

In [3] we synthesise automatically an automaton from a triple $\langle p, f, g\rangle$ specifying a time-series constraint. The synthesis relies on a declarative encoding of procedural knowledge into what we call *decoration tables* [3]. Each pattern is specified by a transducer [6,15] obeying wellformedness conditions. The decoration tables are parametrised by features and aggregators, and define substitution rules on the transducers that allow an automaton with $m = 3$ accumulators to be synthesised. The future work in [3] included simplifying the synthesised automata, as they often have more accumulators and more complex accumulator updates than manually designed ones: this may slow down the checker and weaken CP or MIP decompositions of the constraint specified by the synthesised automaton.

In this paper, we largely overcome this bottleneck. Rather than designing a procedural minimisation algorithm for automata with accumulators, we have again opted for capturing such procedural knowledge in a *declarative* and thus more easily reusable way: it suffices to specialise the decoration tables of [3] for some combinations of algebraic properties of pattern-feature-aggregator triples.

First, we recall the concept of pattern e-occurrence from [3], capturing where a feature value is extracted from the time series.

Definition 1. *Given a pattern p; a sequence $X_0, \ldots, X_{n-1}$; its signature sequence $S_0, \ldots, S_{n-2}$; and a non-empty subsequence $S_i, S_{i+1}, \ldots, S_j$ forming a maximal word that matches p, with $0 \leq i \leq j \leq n-2$; the* e-occurrence *of that maximal word is the interval $[\ell, u]$ of corresponding indices within $X_0, \ldots, X_{n-1}$.*

In Example 1, the sequence $X = \langle 4, 4, 3, 2, 2, 6, 3, 5\rangle$ gives the signature sequence $S = \langle =, >, >, =, <, >, <\rangle$, which contains two maximal words matching the pattern $>^+$ of strictly decreasing sequences, namely $\langle S_1, S_2\rangle = \langle >, >\rangle$ and $\langle S_5\rangle = \langle >\rangle$, corresponding to the strictly decreasing sequences $\langle X_1, X_2, X_3\rangle = \langle 4, 3, 2\rangle$ and $\langle X_5, X_6\rangle = \langle 6, 3\rangle$, hence the e-occurrences are $[1, 3]$ and $[5, 6]$. A pattern occurrence $\langle S_i, \ldots, S_j\rangle$ within the signature sequence has the e-occurrence $[i, j + 1]$ for this constraint, but it could be $[i + 1, j]$ for other constraints [3].

All synthesised automata in [3] have the accumulators c, d, and r, which respectively denote the feature value of the *current* pattern e-occurrence (such as accumulator c in Figure 1); the feature value of a *potential* part of a pattern e-occurrence (no such accumulator is needed in Figure 1, and achieving this is the purpose of this section); and the aggregated *result* value for the feature values of the pattern e-occurrences already encountered (such as accumulator r in Figure 1). Figure 2B&C gives the functions used to compute the feature and aggregation values. If the pattern, feature, and aggregator satisfy some properties, then either it is enough to perform the accumulator update only on one specific transition of the automaton, as in Definition 3, or it is possible to start aggregating immediately upon finding an e-occurrence, as in Definition 4. To state these properties, we need another concept.

Definition 2. *A transition from state q to state q' in an automaton is called a* 'found' transition *if it is the only transition on some path from the initial state q_0 to q' that modifies the accumulator c.*

Simplification	Percentage
aggregate once	28.9 %
immediate aggreg.	45.9 %
other properties	11.6 %
unchanged automata [3]	13.6 %

(A)

Aggregator g	$\text{default}_{g,f}$
`Max`	$\min_f$
`Min`	$\max_f$
`Sum`	0

(B)

Feature f	id_f	$\min_f$	$\max_f$	ϕ_f	δ_f^i
`one`	1	1	1	1	1
`width`	0	0	n	$+$	1
`surface`	0	$-\infty$	$+\infty$	$+$	X_i
`max`	$-\infty$	$-\infty$	$+\infty$	max	X_i
`min`	$+\infty$	$-\infty$	$+\infty$	min	X_i
`range`	0	0	$+\infty$	n/a	X_i

(C)

Fig. 2: (A) Percentage, among the 266 time-series constraints, of automata that can be simplified using the discovered properties. (C) Features: their identity, minimum, and maximum values; the functions ϕ_f and δ_f^i are used to compute recursively the feature value v_u of a sequence $\langle X_\ell, \ldots, X_u \rangle$ by $v_\ell = \phi_f(\text{id}_f, \delta_f^\ell)$ and $v_i = \phi_f(v_{i-1}, \delta_f^i)$ for $i > \ell$; note that δ_f^i provides the contribution of X_i to the value of feature f; (B) Aggregators and their default values.

For example, the transition from the start state s to state u in Figure 1 is a 'found' transition, as it sets c to 2.

Definition 3. *Given a time-series constraint γ on feature f, an e-occurrence $[\ell, u]$ of its pattern such that X_s triggers a 'found' transition of its automaton, with $s \in [\ell, u]$, we say that γ is an* aggregate-once constraint *if δ_f^s equals $\phi_f(\phi_f(\ldots \phi_f(\text{id}_f, \delta_f^\ell), \ldots, \delta_f^{u-1}), \delta_f^u)$, where ϕ_f and δ_f^i are as in Figure 2B.*

For aggregate-once constraints the feature value of an e-occurrence depends only on the value of δ_f^s, hence we need only one counter for aggregating.

For example, any constraint with feature $f =$ `one`, i.e., any constraint counting the number of occurrences of a pattern, is an aggregate-once constraint, because for any e-occurrence $[\ell, u]$ and any $i, i+1 \in [\ell, u]$ we have $\phi_f(\delta_f^i, \delta_f^{i+1}) = \delta_f^\ell = \delta_f^{\ell+1} = \cdots = \delta_f^u = 1$. Also, consider any constraint with feature $f =$ `max` and pattern '<(<|=)*(>|=)*>', which means there is a strict increase followed by a non-strictly increasing subsequence, possibly a plateau, and then a non-strictly decreasing subsequence, followed by a strict decrease. The maximal value δ_f^s of an e-occurrence $[\ell, u]$ of that pattern is found already when we traverse the 'found' transition for $s \in [\ell, u]$, which is the first transition on signature symbol '>': there is no need then to consider other elements of the e-occurrence because the rest of the pattern is a non-strictly decreasing sequence, so we can aggregate once we know δ_f^s. Formally, such a constraint is an aggregate-once constraint, because for any e-occurrence $[\ell, u]$ we have that $\phi_f(\phi_f(\ldots \phi_f(\text{id}_f, \delta_f^\ell), \ldots, \delta_f^{u-1}), \delta_f^u) =$

$\max(\mathrm{id}_f, \delta_f^\ell, \ldots, \delta_f^u) = \max(\mathrm{id}_f, X_f^\ell, \ldots, X_f^u) = X_s = \delta_f^s$, where X_s triggers a 'found' transition of the automaton, with $s \in [\ell, u]$.

The second kind of time-series constraints, in Definition 4 below, is characterised by a combination of feature and pattern properties for which we can start aggregating a current feature value into the result accumulator r as soon as when we find out that we are within a pattern e-occurrence, i.e., without waiting for the end of that pattern e-occurrence. To understand how a synthesised automaton works, we define the following functions, parametrised by entries from Figure 2B&C, representing the updates of the accumulators c and r:

- $F_f \;: \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \times \mathbb{Z} \qquad (c_i, r_i) \mapsto (\phi_f(c_i, \delta_f^i),\ r_i)$
- $G'_{f,g}\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \times \mathbb{Z} \qquad (c_i, r_i) \mapsto (\mathrm{id}_f,\ g(r_i, \phi_f(c_i, \delta_f^i)))$
- $G''_{f,g}\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \times \mathbb{Z} \qquad (c_i, r_i) \mapsto (\phi_f(c_i, \delta_f^i),\ g(r_i,\ \phi_f(c_i, \delta_f^i)))$

When a synthesised automaton from [3] computes the value of feature f for an e-occurrence $[\ell, u]$ and aggregates it into the result accumulator r, the new value of r is computed by first applying $u - \ell$ times the function F_f and then applying the function $G'_{f,g}$. However it is often possible to aggregate this feature value into r without waiting for the end of the e-occurrence. There are two such situations: either (a) before aggregating, we must evolve the feature value of the e-occurrence in accumulator c; or (b) we need not evolve this feature value in c, but after each aggregation c is reset to the id_f value from Figure 2B. We apply $u - \ell$ times the function $G''_{f,g}$ or $G'_{f,g}$ for the situations (a) and (b) respectively. Finally $G'_{f,g}$ is applied once for both (a) and (b), since we do not have to keep in accumulator c the feature value when we are at the end of the e-occurrence. The old [3] order of accumulator updates corresponds to $G'_{f,g} \circ F_f \circ \cdots \circ F_f$, called order (1), while the new order of updates corresponds to either $G'_{f,g} \circ G'_{f,g} \circ \cdots \circ G'_{f,g}$, called order (2), or $G'_{f,g} \circ G''_{f,g} \circ \cdots \circ G''_{f,g}$, called order (3).

Definition 4. *A time-series constraint is an* immediate-aggregation constraint *if for any e-occurrence the use of order* (1) *has the same result as using either order* (2) *or order* (3).

Due to the immediate-aggregation property, we do not have to distinguish the potential and current parts anymore. In [3], updating r is done after the end of an e-occurrence, taking into account the current feature value in c. However, we need not aggregate after the end of an e-occurrence, as the update of r should happen when we are sure that the current element X_i belongs to the e-occurrence, so we can use c for keeping both the potential and current parts.

For example, the MaxWidthStrictlyDecreasingSequence constraint is an immediate-aggregation constraint. This is illustrated in Figure 3, where c_i and r_i respectively denote the values of accumulators c and r after consuming X_i: we consider an e-occurrence $[\ell, u]$ and apply the two orders (1) and (3); after the last update, the value of the accumulator r coincides for both orders. The column 'before' contains the value of the accumulators just before the e-occurrence $[\ell, u]$. The simplified automaton for this constraint is given in Figure 1.

The percentage of constraints for which we can simplify the automata using the different types of simplifications is given in Figure 2A.

	before	update 1	$\cdots$ update $u-\ell$	update $u-\ell+1$
order (1)				
c update		$c_\ell = c_{\ell-1}+1$	$\cdots$ $c_{u-1} = c_{u-2}+1$	$c_u = 0$
r update		$r_\ell = r_{\ell-1}$	$\cdots$ $r_{u-1} = r_{u-2}$	$r_u = \max(r_{u-1}, c_{u-1}+1)$
(c,r)	$(0, r_{\ell-1})$	$(1, r_{\ell-1})$	$\cdots$ $(u-\ell, r_{\ell-1})$	$(0, \max(r_{\ell-1}, u-\ell+1))$
order (3)				
c update		$c_\ell = c_{\ell-1}+1$	$\cdots$ $c_{u-1} = c_{u-2}+1$	$c_u = 0$
r update		$r_\ell = \max(r_{\ell-1}, c_{\ell-1}+1)$	$\cdots$ $r_{u-1} = \max(r_{u-2}, c_{u-2}+1)$	$r_u = \max(r_{u-1}, c_{u-1}+1)$
(c,r)	$(0, r_{\ell-1})$	$(1, \max(r_{\ell-1}, 1))$	$\cdots$ $(u-\ell, \max(r_{\ell-1}, u-\ell))$	$(0, \max(r_{\ell-1}, u-\ell+1))$

Fig. 3: MaxWidthStrictlyDecreasingSequence immediately aggregates

4 MIP Decomposition of Automaton-Based Constraints

Consider a constraint $\gamma(N, \langle X_0, \ldots, X_{n-1}\rangle)$ and signature constraints linking its n variables X_j to $n+1-w$ signature variables S_i, each S_i being functionally determined by a linear relation on w consecutive X_j variables. For ease of notation, we here assume $w = 2$: each S_i is linked to X_i and X_{i+1}, as for the time-series constraints in Section 2. (Other frequent scenarios are $w = 1$, where each S_i is linked to X_i only, and the absence of signature constraints, in which case one would assume $S_i = X_i$ are the signature constraints, also with $w = 1$.)

Assume a ground instance of $\gamma(N, \langle X_0, \ldots, X_{n-1}\rangle)$ holds iff an automaton $\mathcal{A}$ with $m \geq 1$ accumulators a_j that are updated by linear expressions ϕ, possibly using the 'max' and 'min' operators, returns the value of its variable N, called the *result variable*, after consuming the values of its signature variables $S_0, \ldots, S_{n-2}$.

Following [1], we decompose γ for a MIP solver by formulating *logical* constraints that model the triggering of transitions in $\mathcal{A}$ (Section 4.1) and *linearising* those constraints (Section 4.2). For $m = 0$, there is the flow-based MIP decomposition of [8]. For $m = 1$ accumulator that is only updated through increments by positive integers, there is the column-generation approach of [9].

4.1 Logical Constraints

Beside the integer variables $X_0, \ldots, X_{n-1}$ and N of γ, to model the behaviour of $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, I, A, \alpha\rangle$ on the signature variables $S_0, \ldots, S_{n-2}$ over Σ, the key idea is to represent the states visited by $\mathcal{A}$ using *state variables* $Q_0, \ldots, Q_{n-1}$ over Q: each Q_i denotes the state reached *after* consuming S_{i-1}, with $Q_0 = q_0$.

Also, we need *transition variables* $T_0, \ldots, T_{n-2}$ over the set $T = Q \times \Sigma$ of constants denoting all the transitions of the total function δ: each T_i denotes the $(i+1)^{\text{st}}$ triggered transition of $\mathcal{A}$, that is *while* consuming S_i.

Last, we need *accumulator variables* $A_{i,j}$ for $i \in [0, n-1]$ and $j \in [1, m]$: each integer $A_{i,j}$ denotes the value of accumulator a_j *after* the i^{th} transition of $\mathcal{A}$, that is *after* consuming S_{i-1}; each $A_{0,j}$ is given in the tuple I of initial values.

The *signature constraints* functionally determine each signature variable S_i from a linear relation on X_i and X_{i+1}. For example, the signature constraints for time-series constraints are given at the beginning of Section 2.

The *transition constraints* encode the transitions of δ as follows:

$$Q_0 = q_0$$
$$Q_i = q \wedge S_i = \sigma \Rightarrow Q_{i+1} = \delta(q,\sigma) \wedge T_i = \langle q, \sigma \rangle,\ \forall i \in [0, n-2],\ \forall q \in Q,\ \forall \sigma \in \Sigma$$

For example, a representative transition constraint for the automaton of Figure 1 is: $Q_i = s \wedge S_i = \text{`}<\text{'} \Rightarrow Q_{i+1} = s \wedge T_i = \langle s, < \rangle,\ \forall i \in [0, n-2]$.

The *accumulator constraints* are of three kinds: the values of the accumulator variables $A_{0,j}$ before any transitions are found in the m-tuple I of initial values; there is an implication constraint for each transition of δ with its accumulator updates; and the values of the accumulator variables $A_{n-1,j}$ after all transitions are linked to the result variable N according to the acceptance function α. If $A \subsetneq Q$, then we have to pose the additional constraint $Q_{n-1} \in A$.

For example, the accumulator constraints for the automaton in Figure 1 are as follows, using the accumulator variables C_i and L_i for denoting the successive values of the accumulators c and ℓ respectively: the constraints $L_0 = 0$ and $C_0 = 0$ correspond to the pair $I = \langle 0, 0 \rangle$ of initial values; the constraint $N = L_{n-1}$ stems from the acceptance function; further:

$$\begin{array}{lll}
T_i = t & \Rightarrow C_{i+1} = C_i, & \forall t \in \{\langle s, < \rangle, \langle s, = \rangle\},\ \forall i \in [0, n-2] \\
T_i = \langle s, > \rangle & \Rightarrow C_{i+1} = 2, & \forall i \in [0, n-2] \\
T_i = t & \Rightarrow C_{i+1} = 0, & \forall t \in \{\langle u, < \rangle, \langle u, = \rangle\},\ \forall i \in [0, n-2] \\
T_i = \langle u, > \rangle & \Rightarrow C_{i+1} = C_i + 1, & \forall i \in [0, n-2] \\
T_i = t & \Rightarrow L_{i+1} = L_i, & \forall t \in \{\langle s, < \rangle, \langle s, = \rangle, \langle u, < \rangle, \langle u, = \rangle\},\ \forall i \in [0, n-2] \\
T_i = \langle s, > \rangle & \Rightarrow L_{i+1} = \max(L_i, 2), & \forall i \in [0, n-2] \\
T_i = \langle u, > \rangle & \Rightarrow L_{i+1} = \max(L_i, C_i + 1), & \forall i \in [0, n-2]
\end{array}$$

For n variables X_i and m accumulators, there are $n-1$ signature variables, n state variables, $n-1$ transition variables, and mn accumulator variables, hence $\Theta(n)$ variables in total, since m is a constant. Since $\mathcal{A}$ has a constant size, each variable occurs in a constant number of constraints, so there are $\Theta(n)$ constraints.

4.2 Linearising the Logical Constraints

To obtain a linear model, we linearise each group of logical constraints.

For each variable S_i over Σ, we introduce 0-1 variables S_i^σ, with 1 denoting truth and 0 denoting falsity, hence the semantics $S_i^\sigma = 1 \ \Leftrightarrow \ S_i = \sigma$ for all $i \in [0, n-2]$ and $\sigma \in \Sigma$. This requires that exactly one of the S_i^σ takes value 1:

$$\sum_{\sigma \in \Sigma} S_i^\sigma = 1,\ \forall i \in [0, n-2] \tag{1}$$

We replace each atom $S_i = \sigma$ by the Boolean S_i^σ in each logical constraint.

We perform the same operation for the Q_i and T_i variables with respect to their domains, getting variables Q_i^q and T_i^t for all $q \in Q$ and $t \in T$. If $A \subsetneq Q$, then we additionally require $Q_{n-1}^q = 0$ for all $q \in Q \setminus A$.

To linearise the transition constraints, which are now implications where both sides are conjunctions of Boolean variables, we use the technique of [17, pages 172–177].

The accumulator constraints have the general logical form

$$T_i = t \Rightarrow A_{i+1,j} = \phi, \text{ with } i \in [0, n-2],\ j \in [1, m], \text{ and } t \in T$$

where ϕ is here a linear expression, possibly using the 'max' and min' operators, that mentions variables $A_{i,j}$ denoting accumulator values *before* the considered i^{th} transition. We linearise such an implication as follows:

$$A_{i+1,j} - \phi \leq M_j \cdot (1 - T_i^t), \text{ with } i \in [0, n-2],\ j \in [1, m], \text{ and } t \in T$$
$$A_{i+1,j} - \phi \geq M_j \cdot (T_i^t - 1), \text{ with } i \in [0, n-2],\ j \in [1, m], \text{ and } t \in T$$

where constant M_j, chosen with respect to the function ϕ, is such that the constraints above always hold. Computation of M_j may also require calculation of the values serving as plus and minus infinities. For example, for a time-series constraint specified by a triple $\langle p, f, g \rangle$, we have that each M_j depends on the extrema of feature f. If ϕ uses the 'max' and min' operators, then we first linearise it using the technique of [10, pages 4–5], introducing a constant number of new variables.

We linearise the signature constraints by using the following technique, explained on the example of time-series constraints, where the minimum difference between two consecutive integer variables X_i is 1. We rewrite the signature constraint $X_i < X_{i+1} \Leftrightarrow S_i = \text{'<'}$ as two linear inequalities enforcing $S_i^{<} = 1$ if $X_i < X_{i+1}$, and $S_i^{<} = 0$ otherwise:

$$\frac{X_{i+1} - X_i}{M_i'} \leq S_i^{<} \leq \frac{X_{i+1} - X_i}{M_i'} + \frac{2M_i' - 1}{2M_i'},\ \forall i \in [0, n-2]$$

where constant M_i' is $\max\limits_{v \in \text{dom}(X_i),\ w \in \text{dom}(X_{i+1})} |w - v| + 1$, for all $i \in [0, n-2]$, assuming $\text{dom}(Y)$ denotes the domain of variable Y. The linearisation of $X_i > X_{i+1} \Leftrightarrow S_i = \text{'>'}$ is symmetric. The linearisation of $X_i = X_{i+1} \Leftrightarrow S_i = \text{'='}$ is $S_i^{<} = 0 \wedge S_i^{>} = 0$, since the instance $S_i^{<} + S_i^{=} + S_i^{>} = 1$ of (1) implies $S_i^{=} = 1$.

For n variables X_i and m accumulators, there are $(n-1) \cdot |\Sigma|$ signature variables, $n \cdot |Q|$ state variables, $(n-1) \cdot |Q| \cdot |\Sigma|$ transition variables, and mn accumulator variables. Linearising any of the $(n-1) \cdot |Q| \cdot |\Sigma|$ accumulator constraints requires a constant number of new variables, if any. So we still have $\Theta(n)$ variables in total, since m, $|Q|$, and $|\Sigma|$ are constants; for the time-series constraints, we have $|Q| \leq 4$ for 240 of the 266 automata and $|Q| \leq 13$ otherwise, $m \leq 3$ upon the improvements in Section 3, and $|\Sigma| = 3$. Since each variable occurs in a constant number of constraints, there still are $\Theta(n)$ constraints.

5 Improved Generation of Implied Constraints

Given an automaton $\mathcal{A}$ with $m \geq 1$ accumulators a_j, our tool ImpGen [11] generates *invariants* of the form $\alpha_1 a_1 + \cdots + \alpha_m a_m + \gamma \geq 0$: these inequalities

hold at every state of $\mathcal{A}$ for any symbols consumed so far. Let variable $A_{i,j}$ denote the value of accumulator a_j after $\mathcal{A}$ has consumed the first i symbols of a sequence of n symbols: these variables appear in the CP decomposition [4], for a sequence of n variables S_i, of the constraint specified by $\mathcal{A}$. This decomposition in general does not achieve domain consistency when $m \geq 1$ [2]: achieving it is NP-hard for such a constraint in general [5]. Each invariant translates into $n+1$ constraints of the form $\alpha_1 A_{i,1} + \cdots + \alpha_m A_{i,m} + \gamma \geq 0$, for all $0 \leq i \leq n$. We showed in [11] that these constraints are implied by the mentioned CP decomposition, and that the implied constraints translating a suitable selection of invariants improve the propagation strength and speed of that decomposition. The generation of implied constraints is specific to an automaton, but neither to a constrained sequence of variables S_i nor to its length n, and can thus be done offline.

ImpGen handles automata where each accumulator update is a linear expression on accumulators. This includes increments and decrements by constant amounts (as in $c \coloneqq c + 1$) or other accumulators (as in $c \coloneqq c + \ell$), resets (as in $c \coloneqq 0$), etc. This excludes updates via the 'max' and 'min' operators, for instance: ImpGen handles only 64 of the 266 time-series constraints in Section 2.

Towards handling all the time-series constraints, we need to extend ImpGen to handle also conditional accumulator updates of the form $c \coloneqq \text{if } \rho \text{ then } \phi \text{ else } \psi$, where ρ is a linear (in)equality and ϕ, ψ are linear expressions on accumulators: following an idea in [16], we extend the encoding of automaton transitions by allowing preconditions to be expressed. ImpGen now automatically first rewrites accumulator updates containing the binary 'min', 'max', or 'abs' operators into conditional updates. For example, the accumulator update on the arc from s to t in Figure 1 is rewritten as $\langle c, \ell \rangle \coloneqq \langle 2, \text{if } \ell > 2 \text{ then } \ell \text{ else } 2 \rangle$.

Finally, we extend ImpGen to *rank* the implied constraints by decreasing propagation strength when added to the CP decomposition: this is done based on a series of random instances. This enables *automated* selection via a top-k rule for a user-chosen parameter k, as opposed to the previous *manual* selection among a *set* of implied constraints. For example, the top three implied constraints generated from the automaton in Figure 1 are $L_i \geq L_{i-1}$, $L_i \geq L_{i-2}$, and $L_i + L_{i-1} \geq 2 \cdot L_{i-2}$, where L_i denotes the value of accumulator ℓ after consuming the first i symbols. The new tool is available online.[1]

Intuitively, the implied constraints generated by ImpGen can improve inference also for the MIP decomposition of Section 4 because they are generated directly from an automaton and are not necessarily linear combinations of the linear inequalities in that decomposition [13]. Our experiments in the next section confirm that implied constraints that improve the propagation of the CP decomposition can also improve the inference of the MIP decomposition.

[1] `http://www.it.uu.se/research/group/astra/software/impGen.zip`.

(A) Runtime (B) Backtracks

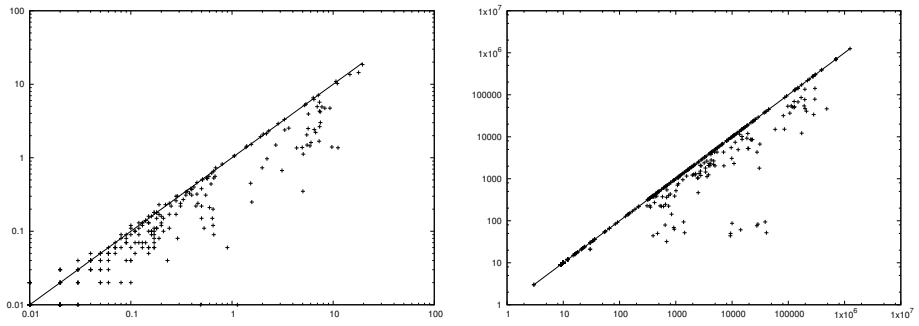


Fig. 4: Time in seconds (left) and backtracks (right) to maximise the result variable for random instances under SICStus Prolog 4.3.2 on a 2011 MacBook Pro 2.2 GHz quad-core Intel Core i7-950 machine with 6MB cache and 16 GB memory. The x-axis is for the new automata and the y-axis is for the old automata: points below the diagonal represent good results for the new automata.

6 Benchmark on CP and MIP Solvers

To evaluate the CP and MIP decompositions of the time-series constraints, we compared their old automata [3] against the new automata of Section 3, and the new automata with and without implied constraints generated as in Section 5.

To compare the old automata against the new automata for CP, we generated instances for all the 266 time-series constraints over time series of length 15 over the domain $\{1, 2, 3\}$. Note that a domain of size 3 is large enough to allow all patterns to occur and to focus the propagation effort on the transition constraints and accumulator constraints but not on the signature constraints. We maximised the result variable, and used a timeout of 100 seconds. As can be seen in Figure 4, the decompositions of the new automata are almost always faster (actually 1.6 times faster on average) and always have fewer backtracks (actually 25% fewer backtracks on average) than those of the old automata.

To compare the new automata with and without implied constraints both for CP and MIP, we generated 40 instances for each constraint used in Section 7 below over time series of length 100 and random sub-intervals of $[0, 1000]$ as domains. We maximised the result variable, and used a timeout of 300 seconds.

Using SICStus Prolog [7], we chose a static search strategy, assigning the variables X_i by increasing index and trying values from smallest to largest. This means that the first solution found is the same with and without implied constraints, and that the times and backtrack counts are directly comparable. The decompositions of the new automata are always faster in the presence of the top two implied constraints, namely 3.33 times faster on average, and always have fewer backtracks, by up to 5 orders of magnitude. In particular, all instances of half the constraints are now solved in less than 1 second instead of timing out.

Using the Gurobi 6.5 [12] MIP solver, the decompositions of the new automata are almost always faster in the presence of the top two implied constraints, namely also 3.33 times faster on average, and can solve to optimality 14% more instances. For the considered constraints, the decompositions of the new automata are always faster than those of the old automata, namely 1.63 times faster on average.

7 Evaluation on a Staff Scheduling Application

For a more realistic evaluation, we introduce a prototypical staff scheduling application that uses a number of time-series constraints. We consider the case of a service company, where demand varies over time, and has to be met at each time point. In order to provide the service level required, we have to define a manpower resource profile over time. Resource cost may vary over time, i.e., employees may be paid different rates at different times. If we could hire and fire personnel arbitrarily, we could follow the demand curve exactly, but this is not allowed, as business processes, employment rules, and union contracts limit how quickly we can change the number of persons employed. We are therefore required to sometimes employ more people than strictly necessary. Note that we are not dealing with a shift rostering problem, where the demand must be covered by people working different shift patterns. In the current problem we are only interested in the total manpower curve, over a long-term horizon.

The overall problem is to cover the given resource demand over time, while minimising overall resource cost, and at the same time satisfying the given time-series constraints.

7.1 Notation, Constants and Variables

In our benchmark, we use a time resolution of one week over a one year horizon, i.e. we consider $n = 52$ time points. The integer variables X_i describe the scheduled resource level at time i. These variables form a single time-series $X_1, ..., X_n$, all constraints are expressed over this time-series or over one of its sub-sequences. The symbols d_i define the given, fixed demand at each time point i. The symbols c_i define the cost of a resource unit at time point i. For each constraint we also introduce an integer variable which represents the aggregated feature value for the constraint. The lower or upper domain bound of these variables will be constrained.

7.2 Objective Function

The objective is to minimise the total cost of the schedule, i.e.

$$obj^* = \min \sum_{i=1}^{n} X_i c_i$$

The overhead $obj^* - \sum_{i=1}^{n} d_i c_i$ is the increase in cost due to the working rules. We can use the overhead also to evaluate the potential cost/savings due to adding/removing a specific working rule. Another lower bound is the sum of the lower domain bounds after initial propagation: we use this to compute the finite-domain optimality gap in our evaluation.

7.3 Constraints

There are two types of constraints, one concerning the demand profile, and the other a set of time-series constraints. At each timepoint, the resources provided must exceed the required demand $X_i \geq d_i$.

The constraints on the time series are given in natural language form below, we also note the constraints used, following the naming scheme in [3].

1. The manpower profile can have at most two peaks. This is expressed with a NbPeak constraint with a parameter variable with an upper bound of two.
2. The manpower profile can have at most two valleys. This is handled by the NbValley constraint.
3. The maximal manpower level at any peak of employment is 250. The numbers employed at the start or end of the planning period can be higher. The MaxMaxPeak constraint handles this condition.
4. We can hire at most 5 persons in one week. This limit is caused by the induction training required. The induction covers safety training, where spaces in each course are limited. We use the MaxRangeIncreasing constraint to model this condition.
5. We can fire at most 7 persons in one week (expressed with a MaxRangeDecreasing constraint).
6. We can only have at most four consecutive increases of personnel in the planning period. This is expressed by the MaxWidthStrictlyIncreasingSequence constraint, considering that four consecutive increases lead to a pattern of width five.
7. We can only have at most six consecutive decreases of personnel numbers in the planning period (using MaxWidthStrictlyDecreasingSequence from Example 1).
8. If we reach a peak in the employment, the profile has to stay constant for at least 10 weeks. Otherwise, we will be violating a "hire and fire" union rule. This is handled by a MinWidthPlateau constraint.
9. If we fire a person, we can not hire another person for four weeks. Instead, we should keep on employing the person (MinWidthPlain).
10. We are not allowed to fire persons in the two weeks before Christmas (expressed with a NbDecreasing constraint on a sub-sequence).
11. In every month, we can have at most 20 new hires. This is due to limitations of the human resources department. For this we use one SumRangeIncreasing constraint for each month.
12. The difference between the highest and lowest peak should not be more than 30. We already have a MaxMaxPeak constraint to constrain the level of

the highest peak. A MinMaxPeak constrains the height of the lowest peak, an inequality between the parameters limits the difference to at most 30.

Manually generated redundant constraints In order to find solutions more easily, we initially manually defined some redundant constraints controlling the domain envelope. Constraint (4) can be approximated by inequalities $X_{i+1} \leq X_i + c$ with a constant c equal to five (this is also generated by ImpGen), while constraints (4) and (6) imply inequalities of the form $X_{i+p+1} \leq X_i + pc$, as any sequence of $p + 1$ intervals can contain at most $p = 4$ increases. These constraints are currently out of the scope of ImpGen because they are linear only at the instance level.

7.4 Search Routine and Experimental Setup

In order to evaluate the impact of different implementations of the constraints, we choose a static search strategy, assigning the X_i variables by increasing index, and enumerating values from smallest to largest. This means that the first solution found is the same for all CP models used, and the times and backtrack counts are directly comparable.

We create random sample problem instances that follow a common structure. There are demand peaks in Spring and Autumn, and reduced demand during Summer and Winter. The minimal difference between peaks and valleys is controlled by a parameter P, which we vary from 10 to 40 in steps of 5. For each parameter value, we generate 100 instances.

We compare different implementations of the time-series constraints, together with manually or automatically generated implied constraints, using the solvers described in Section 6, on the hardware introduced in Figure 4. On their own, the time-series constraints perform quite poorly. Both the old and the new automata definitions only solve instances for the easiest instance set (P=10), finding solutions for 12, respectively 16, of the 100 problems. Adding either manually defined constraints or the top two implied constraints as described in Section 5 to the new automata allow us to find solutions for all problem instances for all parameter values. Using the old automata with the manually defined constraints solves 90, 70, 45, 36, 31, 35, and 32 out of 100 instances for parameter values 10 to 40.

For the combinations of automata and implied constraints that solve all instances we compare backtracks and solution times for the CP model in Table 1, which also shows the average and maximal optimality gap for both the CP and MIP models. Note that the finite-domain solver typically only finds a first solution, and cannot prove optimality within the timeout period. We report results for finding that first solution. At the moment, the MIP solver, even when using the implied constraints and with a timeout of 300 seconds, only finds optimal solutions for some of the problem instances (column Opt), and performs worse than the CP model for some instances.

We can see that both automatically and manually generated implied constraints are important, and that their combination significantly reduces the

Table 1: Backtracks, Execution Times, Solution Quality

	new+implied				new+manual				new+impl.+man.				optimality gap				
	back		time		back		time		back		time		cp		mip		
p	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max	opt
10	20	55	0.08	0.10	478	2168	0.37	1.41	12	35	0.09	0.12	2.86	8.45	1.75	7.97	14
15	80	730	0.11	0.34	548	2144	0.47	1.59	18	42	0.09	0.12	3.27	11.25	1.82	7.22	13
20	200	990	0.17	0.63	496	3921	0.49	4.07	18	43	0.09	0.12	3.42	9.67	2.28	18.77	27
25	1034	17719	0.60	9.30	766	6119	0.73	5.30	35	448	0.10	0.33	3.20	10.54	2.15	17.25	24
30	1001	17726	0.68	13.01	789	6452	0.80	5.85	34	452	0.10	0.35	3.20	8.02	2.04	6.34	26
35	1247	17726	0.86	15.17	824	6621	0.85	6.96	36	460	0.10	0.40	3.38	8.25	2.03	6.21	28
40	1992	25986	1.23	15.44	962	7369	1.02	5.80	37	468	0.10	0.39	3.51	17.32	1.97	10.47	18

search space explored. On average, the best CP solutions found are within 4% of the lower bound, but for some instances the gap is as large as 17%. The average MIP optimality gap is smaller, but the worst cases are even higher, and do not occur for the same instances as for the CP model.

8 Conclusion

Within the context of automaton-specified constraints in general, and time-series constraints in particular, the theoretical contributions of this paper have been shown to improve significantly both CP and MIP models. We hope our work motivates the quest for other general results that have a positive impact on different solving technologies, such as CP, MIP, local search, and SAT.

Acknowledgements. We thank Michel Minoux for his feedback on the integer linear programming decomposition in Section 4. We thank Mats Carlsson for his useful input during the early discussions of this paper. We also thank the anonymous referees for their helpful comments. The first and second authors are partially supported by the Gaspard-Monge programme. The authors at Mines Nantes are supported by project GRACeFUL, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement № 640954. The authors at Uppsala University are supported by grants 2011-6133 and 2012-4908 of the Swedish Research Council (VR). The last author was supported by Science Foundation Ireland under Grant Number SFI/10/IN.1/I3032. The INSIGHT Centre for Data Analytics is supported by Science Foundation Ireland under Grant Number SFI/12/RC/2289.

References

1. Arafailova, E.: Reformulation of automata for time series constraints as linear programs. Master's thesis, Université de Nantes, France (2015)
2. Beldiceanu, N., Carlsson, M., Debruyne, R., Petit, T.: Reformulation of global constraints based on constraints checkers. Constraints 10(4), 339–362 (2005)

3. Beldiceanu, N., Carlsson, M., Douence, R., Simonis, H.: Using finite transducers for describing and synthesising structural time-series constraints. Constraints 21(1), 22–40 (2016), http://dx.doi.org/10.1007/s10601-015-9200-3
4. Beldiceanu, N., Carlsson, M., Petit, T.: Deriving filtering algorithms from constraint checkers. In: CP 2004. LNCS, vol. 3258, pp. 107–122. Springer (2004)
5. Beldiceanu, N., Flener, P., Pearson, J., Van Hentenryck, P.: Propagating regular counting constraints. In: AAAI 2014. pp. 2616–2622. AAAI Press (2014)
6. Berstel, J.: Transductions and Context-Free Languages. Teubner (1979)
7. Carlsson, M., Ottosson, G., Carlson, B.: An open-ended finite domain constraint solver. In: Glaser, H., Hartel, P., Kuchen, H. (eds.) PLILP 1997. LNCS, vol. 1292, pp. 191–206. Springer (1997), the solver is at http://sicstus.sics.se
8. Côté, M.C., Gendron, B., Rousseau, L.M.: Modeling the regular constraint with integer programming. In: CPAIOR 2007. LNCS, vol. 4510, pp. 29–43. Springer (2007)
9. Demassey, S., Pesant, G., Rousseau, L.M.: A Cost-Regular based hybrid column generation approach. Constraints 11(4), 315–333 (2006)
10. FICO: MIP formulations and linearizations. Fair Isaac Corporation (June 2009), http://www.fico.com/en/node/8140?file=5125
11. Francisco Rodríguez, M.A., Flener, P., Pearson, J.: Implied constraints for automaton constraints. In: GCAI 2015. EasyChair Epic Series in Computing (forthcoming), preprint at http://www.it.uu.se/research/group/astra/publications
12. Gurobi Optimization, Inc.: Gurobi optimizer reference manual (2015), http://www.gurobi.com
13. Minoux, M.: Personal communication (July 2015)
14. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: CP 2004. LNCS, vol. 3258, pp. 482–495. Springer (2004)
15. Sakarovitch, J.: Elements of Language Theory. Cambridge University Press (2009)
16. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Constraint-based linear-relations analysis. In: SAS 2004. LNCS, vol. 3148, pp. 53–68. Springer (2004)
17. Williams, H.P.: Model Building in Mathematical Programming. Wiley (2015)