# Generating Linear Invariants for a Conjunction of Automata Constraints

Ekaterina Arafailova, Nicolas Beldiceanu, Helmut Simonis

# Generating Linear Invariants for a Conjunction of Automata Constraints *

Ekaterina Arafailova[1], Nicolas Beldiceanu[1], and Helmut Simonis[2]

[1] TASC (LS2N), IMT Atlantique, FR – 44307 Nantes, France
{Ekaterina.Arafailova,Nicolas.Beldiceanu}@imt-atlantique.fr
[2] Insight Centre for Data Analytics, University College Cork, Ireland
Helmut.Simonis@insight-centre.org

**Abstract.** We propose a systematic approach for generating linear implied constraints that link the values returned by several automata with accumulators after consuming the same input sequence. The method handles automata whose accumulators are increased by (or reset to) some non-negative integer value on each transition. We evaluate the impact of the generated linear invariants on conjunctions of two families of time-series constraints.

## 1 Introduction

We present a compositional method for deriving linear invariants for a conjunction of global constraints that are each represented by an automaton with accumulators [10]. Since they do not encode explicitly all potential values of accumulators as states, automata with accumulators allow a constant size representation of many counting constraints imposed on a sequence of integer variables. Moreover their compositional nature permits representing a conjunction of constraints on a same sequence as the intersection of the corresponding automata [22,21], i.e. the intersection of the languages accepted by all automata, without representing explicitly the Cartesian product of all accumulator values. As a consequence, the size of such an intersection automaton is often quite compact, even if maintaining domain consistency for such constraints is in general NP-hard [8]; for instance the intersection of the 22 automata that restrict the number of occurrences of patterns of the Vol. II of the time-series catalogue [3] in a sequence has only 16 states. The contributions of this paper are twofold:

– First, Sections 3 and 4 provide the basis of a simple, systematic and uniform preprocessing technique to compute necessary conditions for a conjunction of automata with accumulator constraints on the same sequence. Each necessary condition is a linear inequality involving the result variables of the different automata, representing the fact that the result variables cannot vary

* E. Arafailova is supported by the EU H2020 programme under grant 640954 for the GRACeFUL project. N. Beldiceanu is partially supported by GRACeFUL and by the Gaspard-Monge programme. H. Simonis is supported by Science Foundation Ireland (SFI) under grant numbers SFI/12/RC/2289 and SFI/10/IN.1/I3032.

independently. These inequalities are parametrised by the sequence size and are independent of the domains of the sequence variables. The method may be extended if the scopes of the automata constraints overlap, but not if the scopes contain the same variables ordered differently.
– Second, within the context of the time-series catalogue, Section 5 shows that the method allows to precompute in less than five minutes a data base of 7755 invariants that significantly speed up the search for time series satisfying multiple time-series constraints.

Adding implied constraints to a constraint model has been recognized from the very beginning of Constraint Programming as a major source of improvement [13]. Attempts to generate such implied constraints in a systematic way were limited (1) by the difficulty to manually prove a large number of conjectures [17,6], (2) by the limitations of automatic proof systems [15,12], or (3) to specific constraints like `alldifferent`, `gcc`, `element` or `circuit` [19,1,18]. Within the context of automata with accumulators, linear invariants relating consecutive accumulators values of a same constraint were obtained [14] using Farkas' lemma [11] in a resource intensive procedure.

## 2 Background

Consider a sequence of integer variables $X = \langle X_1, X_2, \ldots, X_n \rangle$, and a function $S \colon \mathbb{Z}^p \to \Sigma$, where $\Sigma$ is a finite set denoting an alphabet. Then, the *signature* of $X$ is a sequence $\langle S_1, S_2, \ldots, S_{n-p+1} \rangle$, where every $S_i$ equals $S(X_i, X_{i+1}, \ldots, X_{i+p-1})$. Intuitively, the signature of a sequence is a mapping of $p$ consecutive elements of this sequence to an alphabet $\Sigma$, where $p$ is called the *arity* of the signature.

An *automaton* $\mathcal{M}$ with a memory of $m \geq 0$ integer accumulators [10] is a tuple $\langle Q, \Sigma, \delta, q_0, I, A, \alpha \rangle$, where $Q$ is the set of *states*, $\Sigma$ 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 *initial 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* – the identity in this paper –, transforming the memory of an accepting state into an integer. If the left-to-right consumption of the symbols of a word $w$ in $\Sigma^*$ transits from $q_0$ to some accepting state and the $m$-tuple $C$ of final accumulator values, then the automaton *returns* the value $\alpha(C)$, otherwise it *fails*. An integer sequence $\langle X_1, X_2, \ldots, X_n \rangle$ is an *accepting sequence* wrt an automaton $\mathcal{M}$ if its signature is accepted by $\mathcal{M}$. The *intersection* [21] of $k$ automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ is denoted by $\mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$.

An automaton with accumulators can be seen as a checker for a constraint that has two arguments, namely (1) a sequence of integer variables $X$; (2) an integer variable $R$. Then, for a ground sequence $X$ and an integer number $R$, the constraint holds iff after consuming the *signature* of $X$, the corresponding automaton returns $R$. In Example 1, we introduce two constraints with their automata that will be further used as a running example.

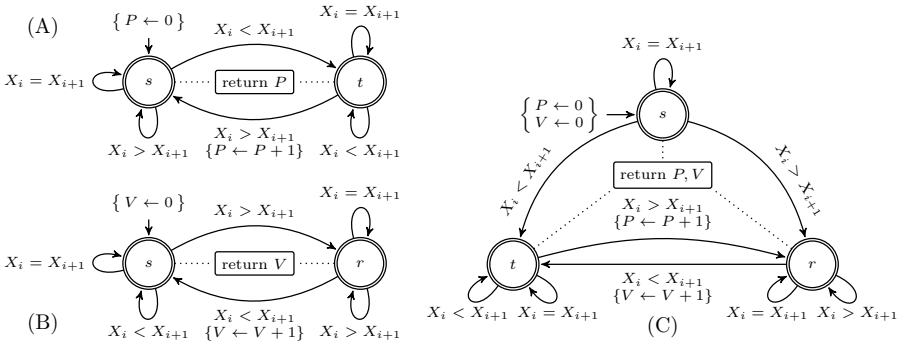



Fig. 1: Automata for (A) `peak`, (B) `valley`, and (C) their intersection; within each automaton accepting states are shown by double circles.

*Example 1.* Consider a sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ of integer variables. A *peak* (resp. *valley*) is a variable $X_k$ of $X$ (with $k \in [2, n-1]$) such that there exists an $i$ where $X_{i-1} < X_i$ (resp. $X_{i-1} > X_i$) and $X_i = X_{i+1} = \cdots = X_k$ and $X_k > X_{k+1}$ (resp. $X_k < X_{k+1}$). For example, the sequence $\langle 1, 2, 6, 6, 7, 0, 4, 2 \rangle$ has two peaks, namely 7 and 4, and one valley, namely 0. Then, the $\texttt{peak}(X, P)$ (resp. $\texttt{valley}(X, V)$) constraint restricts $P$ (resp. $V$) to be the number of peaks (resp. valleys) in the sequence $X$.

Both constraints can be represented by automata with one accumulator, which consume the signature of $X$, defined by the following conjunction of constraints: $S(X_i, X_{i+1}) = \text{`<'} \Leftrightarrow X_i < X_{i+1} \wedge S(X_i, X_{i+1}) = \text{`='} \Leftrightarrow X_i = X_{i+1} \wedge S(X_i, X_{i+1}) = \text{`>'} \Leftrightarrow X_i > X_{i+1}$. Fig. 1 gives the automata for `peak`, `valley`, and their intersection. For a ground sequence $X$, and for an integer value $P$ (resp. $V$), the constraint $\texttt{peak}(X, P)$ (resp. $\texttt{valley}(X, V)$) holds iff, after consuming the signature of $X$, the automaton in Part (A) (resp. Part(B)) of Fig. 1 returns $P$ (resp. $V$). Given the sequence $X = \langle 1, 2, 6, 6, 7, 0, 4, 2 \rangle$, the constraints $\texttt{peak}(X, 2)$ and $\texttt{valley}(X, 1)$ hold since, after consuming $X = \langle 1, 2, 6, 6, 7, 0, 4, 2 \rangle$, the peak and valley automata of Fig. 1 return 2 and 1, respectively. △

## 3 Generating Linear Invariants

Consider $k$ automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ over a same alphabet $\Sigma$. Let $r_i$ denote the number of accumulators of $\mathcal{M}_i$, and let $V_i$ designate its returned value. In this section we show how to systematically generate linear invariants of the form

$$e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i \geq 0 \quad \text{with} \quad e, e_0, e_1, \ldots, e_k \in \mathbb{Z}, \tag{1}$$

which hold after the signature of a same input sequence $\langle X_1, X_2, \ldots, X_n \rangle$ is completely consumed by the $k$ automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$. We call such invariant *general* since it holds regardless of any conditions on the result variables

$V_1, V_2, \ldots, V_k$. Stronger, but less general, invariants may be obtained when the result variables cannot be assigned the initial values of the accumulators.

Our method for generating invariants is applicable for a restricted class of automata with accumulators that we now introduce.

*Property 1.* An automaton $\mathcal{M}$ with $r$ accumulators have the *incremental-automaton* property if the following conditions are all satisfied:

1. For every accumulator $A_j$ of $\mathcal{M}$, its initial value $\alpha_j^0$ is a natural number.
2. For every accumulator $A_j$ of $\mathcal{M}$ and for every transition $t$ of $\mathcal{M}$, the update of $A_j$ upon triggering transition $t$ is of the form $A_j \leftarrow \alpha_{j,0}^t + \sum_{i=1}^{r} \alpha_{j,i}^t \cdot A_i$, with $\alpha_{j,0}^t \in \mathbb{N}$ and $\alpha_{j,1}^t, \alpha_{j,2}^t, \ldots, \alpha_{j,r}^t \in \{0, 1\}$.
3. The accumulator $A_r$ is called the *main accumulator* and verifies the following three conditions:
   (a) the value returned by automaton $\mathcal{M}$ is the last value of its main accumulator $A_r$,
   (b) for every transition $t$ of $\mathcal{M}$, $\alpha_{r,r}^t = 1$,
   (c) for a non-empty subset $T$ of transitions of $\mathcal{M}$, $\sum_{i=1}^{r-1} \alpha_{r,i}^t > 0,\ \forall t \in T$.
4. For all other accumulators $A_j$ with $j < r$, on every transition $t$ of $\mathcal{M}$, we have $\sum_{i=1, i\neq j}^{r} \alpha_{j,i}^t = 0$ and if $\alpha_{r,j}^t > 0$, then $\alpha_{j,j}^t$ is 0.

The intuition behind the incremental-automaton property is that there is one accumulator that we name *main accumulator*, whose last value is the final value, returned by the automaton, (see 3a). At some transitions, the update of the main accumulator is a linear combination of the other accumulators, while on the other transitions its value either does not change or incremented by a non-negative constant, (see 3b and 3c). All other accumulators may only be incremented by a non-negative constant or assigned to some non-negative integer value, and they *may* contribute to the final value, (see 4). These accumulators are called *potential accumulators*. Both automata in Fig. 1 have the incremental-automaton property, and their single accumulators are main accumulators. Volumes I and II of the global constraint catalogue contain more than 50 such automata. In the rest of this paper we assume that all automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ have the incremental-automaton property.

Our approach for systematically generating linear invariants of type $e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i \geq 0$ considers each combination of signs of the coefficients $e_i$ (with $i \in [0, k]$). It consists of three main steps:

1. Construct a non-negative function $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i$, which represents the left-hand side of the sought invariant (see Section 3.1).

2. Select the coefficients $e_0, e_1, \ldots, e_k$, called the *relative coefficients* of the linear invariant, so that there exists a constant $C$ such that $e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i \geq C$ (see Section 3.2).
3. Compute $C$ and set the coefficient $e$, called the *constant term* of the linear invariant, to $-C$ (see Section 3.3).

The three previous steps are performed as follows:

1. First, we assume a sign for each coefficient $e_i$ (with $i \in [0, k]$), which tells whether we have to consider or not the contribution of the potential accumulators; note that each combination of signs of the coefficients $e_i$ (with $i \in [1, k]$) will lead to a different linear invariant. Then, from the intersection automaton $\mathcal{I}$ of $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$, we construct a digraph called the *invariant digraph*, where each transition $t$ of $\mathcal{I}$ is replaced by an arc whose weight represents the lower bound of the variation of the term $e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i$ while triggering $t$.
2. Second, we find the coefficients $e_i$ (with $i \in [0, k]$) so that the invariant digraph does not contain any negative cycles.
3. Third, to obtain $C$ we compute the shortest path in the invariant digraph from the node of the invariant digraph corresponding to the initial state of $\mathcal{I}$ to all nodes corresponding to accepting states of $\mathcal{I}$.

### 3.1 Constructing the Invariant Digraph for a Conjunction of Automaton Constraints wrt a Linear Function

First, Definition 1 introduces the notion of *invariant digraph* $G_{\mathcal{I}}^{v}$ of the automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$ wrt a linear function $v$ involving the values returned by these automata. Second, Definition 2 introduces the notion of *weight of an accepting sequence* $X$ wrt $\mathcal{I}$ in $G_{\mathcal{I}}^{v}$, which makes the link between a path in $G_{\mathcal{I}}^{v}$ and the vector of values returned by $\mathcal{I}$ after consuming the signature of $X$. Finally, Theorem 1 shows that the weight of $X$ in $G_{\mathcal{I}}^{v}$ is a lower bound on the linear function $v$.

**Definition 1.** *Consider an accepting sequence* $X = \langle X_1, X_2, \ldots, X_n \rangle$ *wrt the automaton* $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$*, and a linear function* $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i$*, where* $(V_1, V_2, \ldots, V_k)$ *is the vector of values returned by* $\mathcal{I}$ *after consuming the signature of* $X$*. The* invariant digraph *of* $\mathcal{I}$ *wrt* $v$*, denoted by* $G_{\mathcal{I}}^{v}$ *is a weighted digraph defined in the following way:*

- *The set of nodes of* $G_{\mathcal{I}}^{v}$ *is the set of states of* $\mathcal{I}$*.*
- *The set of arcs of* $G_{\mathcal{I}}^{v}$ *is the set of transitions of* $\mathcal{I}$*, where for every transition* $t$ *the corresponding symbol of the alphabet is replaced by an integer weight,*

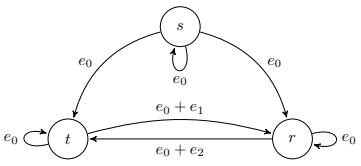
*which is* $e_0 + \sum_{i=1}^{k} e_i \cdot \beta_i^t$*, where* $\beta_i^t$ *is defined as follows, and where* $r_i$ *denotes the number of accumulators of* $\mathcal{M}_i$*:*

$$\beta_i^t = \begin{cases} \alpha_{r_i,0}^t \text{ of } \mathcal{M}_i, & \text{if } e_i \geq 0 \quad (1) \\ \sum_{j=1}^{r_i} \alpha_{j,0}^t \text{ of } \mathcal{M}_i, & \text{if } e_i < 0 \quad (2) \end{cases}$$

**Definition 2.** *Consider an accepting sequence* $X = \langle X_1, X_2, \ldots, X_n \rangle$ *wrt the automaton* $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$*, and a linear function* $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i$*, where* $(V_1, V_2, \ldots, V_k)$ *is the vector of values returned by* $\mathcal{I}$ *after consuming the signature of* $X$*. The* walk of $X$ in $G_{\mathcal{I}}^v$ *is a path* $\omega$ *in* $G_{\mathcal{I}}^v$ *whose sequence of arcs is the sequence of the corresponding transitions of* $\mathcal{I}$ *triggered upon consuming the signature of* $X$*. The* weight of $X$ in $G_{\mathcal{I}}^v$ *is the weight of its path in* $G_{\mathcal{I}}^v$ *plus a constant value, which is a lower bound on* $v$ *corresponding to the initial values of the accumulators. It equals* $e + e_0 \cdot (p-1) + \sum_{i=1}^{k} e_i \cdot \beta_i^0$*, where* $p$ *is the arity of the signature, and where* $\beta_i^0$ *is defined as follows, and where* $r_i$ *denotes the number of accumulators of* $\mathcal{M}_i$*:*

$$\beta_i^0 = \begin{cases} \alpha_{r_i}^0 \text{ of } \mathcal{M}_i, & \text{if } e_i \geq 0 \quad (1) \\ \sum_{j=1}^{r_i} \alpha_j^0 \text{ of } \mathcal{M}_i, & \text{if } e_i < 0 \quad (2) \end{cases}$$

*Example 2.* Consider $\texttt{peak}(\langle X_1, X_2, \ldots, X_n \rangle, P)$ and $\texttt{valley}(\langle X_1, X_2, \ldots, X_n \rangle, V)$ introduced in Example 1. Fig. 1 gives the automata for `peak`, `valley`, and their intersection $\mathcal{I}$. We aim to find inequalities of the form $e + e_0 \cdot n + e_1 \cdot P + e_2 \cdot V \geq 0$ that hold for every integer sequence $X$. After consuming the signature of $X = \langle X_1, X_2, \ldots, X_n \rangle$, $\mathcal{I}$ returns a pair of values $(P, V)$, which are the number of peaks (resp. valleys) in $X$. The invariant digraph of $\mathcal{I}$ wrt $v = e + e_0 \cdot n + e_1 \cdot P + e_2 \cdot V$ is given in the figure on the right. Since both automata do not have any potential accumulators, the weights of the arcs of $G_{\mathcal{I}}^v$ do not depend on the signs of $e_1$ and $e_2$. Hence, for every integer sequence $X$, its weight in $G_{\mathcal{I}}^v$ equals $e + e_0 \cdot n + e_1 \cdot P + e_2 \cdot V$. △




**Theorem 1.** *Consider an accepting sequence* $X = \langle X_1, X_2, \ldots, X_n \rangle$ *wrt the automaton* $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$*, and a linear function* $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i$*, where* $(V_1, V_2, \ldots, V_k)$ *is the vector of values return by* $\mathcal{I}$*. Then, the weight of* $X$ *in* $G_{\mathcal{I}}^v$ *is less than or equal to* $e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i$*.*

*Proof.* Since, when doing the intersection of automata we do not merge accumulators, the accumulators of $\mathcal{I}$ that come from different automata $\mathcal{M}_i$ and $\mathcal{M}_j$

do not interact, hence the returned values of $\mathcal{M}_i$ and $\mathcal{M}_j$ are independent. By definition of the invariant digraph, the weight of any of its arc is $e_0 + \sum_{i=1}^{k} e_i \cdot \beta_i^t$, where $\beta_i^t$ depends on the sign of $e_i$, and where $t$ is the corresponding transition in $\mathcal{I}$. Then, the weight of $X$ in $G_{\mathcal{I}}^v$ is the constant $e+e_0\cdot(p-1)+\sum_{i=1}^{k} e_i\cdot\beta_i^0$ (see Definition 2) plus the weight of the walk of $X$, which is in total $e+e_0\cdot(p-1)+\sum_{i=1}^{k} e_i\cdot\beta_i^0+e_0\cdot(n-p+1)+\sum_{j=1}^{n-p+1}\sum_{i=1}^{k} e_i\cdot\beta_i^{t_j} = e+e_0\cdot n+\sum_{i=1}^{k} e_i\cdot\left(\beta_i^0+\sum_{j=1}^{n-p+1}\beta_i^{t_j}\right)$, where $p$ is the arity of the considered signature, and $t_1, t_2, \ldots t_{n-p+1}$ is the sequence of transitions of $\mathcal{I}$ triggered upon consuming the signature of $X$. We now show that the value $e_i \cdot \left(\beta_i^0 + \sum_{j=1}^{n-p+1}\beta_i^{t_j}\right)$ is not greater than $e_i \cdot V_i$. This will imply that the weight of the walk of $X$ in $G_{\mathcal{I}}^v$ is less than or equal to $v = e+e_0\cdot n+\sum_{i=1}^{k} e_i\cdot V_i$.

Consider the $v_i = e_i\cdot V_i$ linear function. We show that the weight of $X$ in $G_{\mathcal{I}}^{v_i}$, which equals $e_i \cdot \left(\beta_i^0 + \sum_{j=1}^{n-p+1}\beta_i^{t_j}\right)$, is less than or equal to $e_i \cdot V_i$. Depending on the sign of $e_i$ we consider two cases.

**Case 1: $\mathbf{e_i \geq 0}$**. In this case, the weight of every arc of $G_{\mathcal{I}}^{v_i}$ is $e_i$ multiplied by $\alpha_{r_i,0}^t$, where $t$ is the corresponding transition in $\mathcal{I}$, and $r_i$ is the main accumulator of $\mathcal{M}_i$ (see Case 1 of Definition 1). If, on transition $t$, some potential accumulators of $\mathcal{M}_i$ are incremented by a positive constant, the real contribution of the accumulator updates on this transition to $V_i$ is at least $\alpha_{r_i,0}^t$ since $e_i \geq 0$. The same reasoning applies to the contribution of the initial values of the potential accumulators to the final value $V_i$. Since this contribution is non-negative, it is ignored, and $\beta_i^0 = \alpha_{r_i}^0$ (see Case 1 of Definition 2). Hence, $e_i \cdot (\beta_i^0 + \sum_{j=1}^{n-p+1}\beta_i^{t_j}) = e_i \cdot (\alpha_{r_i}^0 + \sum_{j=1}^{n-p+1}\alpha_{r_i,0}^t) \leq e_i \cdot V_i$.

**Case 2: $\mathbf{e_i < 0}$**. In this case, the weight of every arc of $G_{\mathcal{I}}^{v_i}$ is $e_i$ multiplied by the sum of the non-negative constants, which come from the updates of *every* accumulator of $\mathcal{M}_i$ (see Case 2 of Definition 1). The contribution of the potential accumulators is always taken into account, and since $e_i < 0$, it is always negative. The same reasoning applies to the contribution of the initial values of the potential accumulators to the returned value $V_i$. Since the initial values of the potential accumulators are non-negative, and $e_i < 0$, in order to obtain a lower bound on $v$ we assume that the initial values of the potential accumulators always contribute to $V_i$ (see Case 2 of Definition 2). Hence, $e_i \cdot (\beta_i^0 + \sum_{j=1}^{n-p+1}\beta_i^{t_j}) \leq e_i \cdot V_i$.

□

Note that if all the considered automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ do not have potential accumulators, then for every accepting sequence $X = \langle X_1, X_2, \ldots, X_n\rangle$

wrt $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$ and for any linear function $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i$, the weight of $X$ in $G^v_{\mathcal{I}}$ is *equal* to $v$. If there is at least one potential accumulator for at least one automaton $\mathcal{M}_i$, then there may exist an integer sequence whose weight in $G^v_{\mathcal{I}}$ is strictly less than $v$.

### 3.2 Finding the Relative Coefficients of the Linear Invariant

We now focus on finding the relative coefficients $e_0, e_1, \ldots, e_k$ of the linear invariant $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i \geq 0$ such that, after consuming the signature of any accepting sequence by the automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$, the value of $v$ is non-negative.

For any accepting sequence $X$ wrt $\mathcal{I}$, by Theorem 1, we have that the weight $w$ of $X$ in $G^v_{\mathcal{I}}$ is less than or equal to $v$. Recall that $w$ consists of a constant part, and of a part that depends on $X$, which involves the coefficients $e_0, e_1, \ldots, e_k$; thus, these coefficients must be chosen in a way that there exists a constant $C$ such that $w \geq C$, and $C$ does not depend on $X$. This is only possible when $G^v_{\mathcal{I}}$ does not contain *any* negative cycles. Let $\mathcal{C}$ denote the set of all simple circuits of $G^v_{\mathcal{I}}$, and let $w_e$ denote the weight of an arc $e$ of $G^v_{\mathcal{I}}$. In order to prevent negative cycles in $G^v_{\mathcal{I}}$, we solve the following minimisation problem, parameterised by $(s_0, s_1, \ldots s_k)$, the signs of $e_0, e_1, \ldots, e_k$:

$$\text{minimise} \quad \sum_{c \in \mathcal{C}} W_c + \sum_{i=1}^{k} |e_i| \tag{3}$$

$$\text{subject to} \quad W_c = \sum_{e \in c} w_e \qquad \forall c \in \mathcal{C} \tag{4}$$

$$W_c \geq 0 \qquad \forall c \in \mathcal{C} \tag{5}$$

$$s_i = \text{`}-\text{'} \Rightarrow e_i \leq 0, \quad s_i = \text{`}+\text{'} \Rightarrow e_i \geq 0 \qquad \forall i \in [0, k] \tag{6}$$

$$e_i \neq 0 \qquad \forall i \in [1, k] \tag{7}$$

In order to obtain the coefficients $e_0, e_1, \ldots, e_k$ so that $G^v_{\mathcal{I}}$ does not contain any negative cycles, it is enough to find a solution to the satisfaction problem (4)-(7). Minimisation is required to obtaining invariants that eliminate as many infeasible values of $(V_1, V_2, \ldots, V_k)$ as possible. Within the objective function (3), the term $\sum_{c \in \mathcal{C}} W_c$ is for minimising the weight of every simple circuit, while the term $\sum_{i=1}^{k} |e_i|$ is for obtaining the coefficients with the smallest absolute value. By changing the sign vector $(s_0, s_1, \ldots s_k)$ we obtain different invariants.

*Example 3.* Consider `peak`$(\langle X_1, X_2, \ldots, X_n \rangle, P)$ and `valley`$(\langle X_1, X_2, \ldots, X_n \rangle, V)$ from Example 2. The invariant digraph of the intersection of the automata for the `Peak` and `Valley` constraints wrt $v = e + e_0 \cdot n + e_1 \cdot P + e_2 \cdot V$ was given in Example 2. This digraph has four simple circuits, namely $s - s$, $t - t$,

$r - r$, and $r - t - r$, which are labeled by 1, 2, 3 and 4, respectively. Then, the minimisation problem for finding the relative coefficients of the invariant $v \geq 0$, parameterised by $(s_0, s_1, s_2)$, the signs of $e_0$, $e_1$ and $e_2$, is the following:

$$\begin{array}{lll} \text{minimise} & \sum_{j=1}^{4} W_j + \sum_{i=0}^{2} |e_i| & \\ \text{subject to} & W_j = e_0, & \forall j \in [1,3] \\ & W_4 = e_0 + e_1 + e_2 & \\ & W_j \geq 0 & \forall j \in [1,4] \\ & s_i = \text{`}-\text{'} \Rightarrow e_i \leq 0, \quad s_i = \text{`}+\text{'} \Rightarrow e_i \geq 0 & \forall i \in [0,2] \\ & e_i \neq 0 & \forall i \in [1,2] \end{array} \tag{8}$$

Note that the value of $e_0$ must be non-negative otherwise (8) cannot be satisfied for $j \in \{1,2,3\}$. Hence, we consider only the combinations of signs of the form $(\text{`}+\text{'}, s_1, s_2)$ with $s_1$ ans $s_2$ being either '$-$' or '$+$'. The following table gives the optimal solution of the minimisation problem for the considered combinations of signs:

| $(s_0, s_1, s_2)$ | $(+,-,-)$ | $(+,-,+)$ | $(+,+,-)$ | $(+,+,+)$ |
|---|---|---|---|---|
| $(e_0, e_1, e_2)$ | $(1,-1,-1)$ | $(0,-1,1)$ | $(0,1,-1)$ | $(0,1,1)$ |

△

### 3.3 Finding the Constant Term of the Linear Invariant

Finally, we focus on finding the constant term $e$ of the linear invariant $v = e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i \geq 0$, when the coefficients $e_0, e_1, \ldots, e_k$ are known, and when the digraph of the automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$ wrt $v$ does not contain any negative cycles. By Theorem 1, the weight of any accepting sequence $X$ wrt $\mathcal{I}$ in $G^v_{\mathcal{I}}$ is smaller or equal to $v$, then if the weight of $X$ is non-negative, it implies that $v$ is also non-negative. Since the digraph $G^v_{\mathcal{I}}$ does not contain any negative cycles, then the weight of $X$ cannot be smaller than some constant $C$. Hence, it suffices to find this constant and set the constant term $e$ to $-C$. The value of $C$ is computed as the constant $e_0 \cdot (p-1) - \sum_{i=1}^{k} \beta_i^0$ (see Definition 2) plus the shortest path length from the node of $G^v_{\mathcal{I}}$ corresponding to the initial state of $\mathcal{I}$ to all the nodes of $G^v_{\mathcal{I}}$ corresponding to the accepting states of $\mathcal{I}$.

*Example 4.* Consider `peak`$(\langle X_1, X_2, \ldots, X_n \rangle, P)$ and `valley`$(\langle X_1, X_2, \ldots, X_n \rangle, V)$ from Example 2 with $n \geq 2$, i.e. the signature of $\langle X_1, X_2, \ldots, X_n \rangle$ is not empty. In Example 3, we found four vectors for the relative coefficients $e_0$, $e_1$, $e_2$ of the invariant $e + e_0 \cdot n + e_1 \cdot P + e_2 \cdot V \geq 0$. For every found vector for the relative coefficients $(e_0, e_1, e_2)$, we obtain a weighted digraph, whose weights now are integer numbers. For example, for the vector $(e_0, e_1, e_2) = (0, -1, 1)$, the obtained digraph is given in Part (A) of Fig. 2. We compute the length of

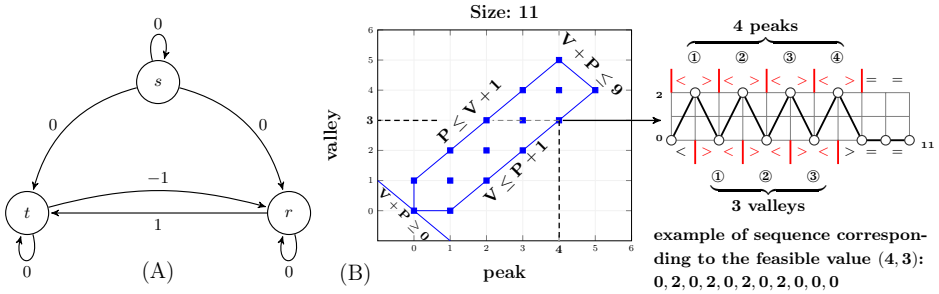



Fig. 2: (A) The invariant digraph of the automata for the `Peak` and the `Valley` constraints. (B) The set of feasible values of the result variables $P$ and $V$ of the `Peak` and the `Valley` constraints, respectively, for sequences of size 11.

a shortest path from the node $s$, which corresponds to the initial state of the automaton in Part (C) of Fig. 1 to every node corresponding to the accepting state of the automaton in Part (C) of Fig. 1. The length of the shortest path from $s$ to $s$ is 0, from $s$ to $t$ is 0, and from $s$ to $r$ is $-1$. The minimum of this values is $-1$, hence the constant term $e$ equals $-(0 + (-1)) = 1$. The obtained invariant is $P \leq V + 1$.

In a similar way, we find the constant terms for the other found vectors of the relative coefficients $(e_0, e_1, e_2)$, and obtain the following invariants: $V \leq P + 1$, $V + P \leq n - 2$, $V + P \geq 0$.

Part (B) of Fig. 2 gives the polytope of feasible points $(P, V)$ when $n$ is 11. Observe that three of the four found linear invariants are facets of the convex hull of this polytope, which implies that these invariants are sharp. △

*Example 5.* We illustrate how the method presented in this section can also be used for generating linear invariants for non time-series constraints. Consider a sequence of integer variables $X = \langle X_1, X_2, \ldots, X_n \rangle$ with every $X_i$ ranging over $[0, 3]$, four `among` [7] constraints that restrict the variables $V_0$, $V_1$, $V_2$, $V_3$ to be the number of occurrences of values $0, 1, 2, 3$, respectively, in $X$, as well as the four corresponding `stretch` [23] constraints restricting the stretch length in $X$ to be respectively in $[1, 4]$, $[2, 5]$, $[3, 5]$, and $[1, 2]$. In addition assume that value 2 (resp. 1) cannot immediately follow a 3 (resp. 2). The intersection of the corresponding automata has 17 states and allows to generate 16 linear invariants, one of them being $2 + n + V_0 + V_1 - V_2 - 2 \cdot V_3 \geq 0$. Since the sum of all $V_i$ is $n$, this invariant can be simplified to $2 + 2 \cdot n - 2 \cdot V_2 - 3 \cdot V_3 \geq 0$, which is equivalent to $2 \cdot (V_2 + V_3 - n) \leq 2 - V_3$. This inequality means that if $X$ consists only of the values 2 and 3, i.e. $V_2 + V_3 - n = 0$, then $V_3 \leq 2$, which represents the conjunction of the conditions that the stretch length of $V_3 \in [1, 2]$ and $(X_i = 3) \Rightarrow (X_{i+1} \neq 2)$. △

## 4 Conditional Linear Invariants

In Section 3, we presented a method for generating linear invariants linking the values returned by an automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$ after consuming the signature of a same accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt $\mathcal{I}$. In this section, we present several cases where the same method can be used for generating conditional linear invariants.

Quite often an automaton $\mathcal{M}_i$ (with $i$ in $[1, k]$) returns its initial value only when the signature of $X$ does not contain any occurrence of some regular expression $\sigma_i$. This may lead to a convex hull of points of coordinates $(V_1, V_2, \ldots, V_k)$ returned by $\mathcal{I}$ containing infeasible points, e.g. see Part (A) of Fig. 3. Some of these infeasible points can be eliminated by stronger invariants subject to the condition, called the *non-default value* condition, that no variable of the returned vector is assigned to the initial value of the corresponding accumulator. Section 4.1 shows to generate such invariants. Section 4.2 introduces the notion of *guard* of a transition $t$ of $\mathcal{I}$, a linear inequality of the form $e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i \geq 0$, which is a *necessary condition* on the vector of values returned by $\mathcal{I}$ after consuming $X$ for triggering the transition $t$ upon consuming $X$.

### 4.1 Linear Invariants with the Non-Default Value Condition

We first illustrate the motivation for such invariants.

*Example 6.* Consider the `nb_decreasing_terrace`$(\langle X_1, X_2, \ldots, X_n \rangle, V_1)$ and the `sum_width_increasing_terrace`$(\langle X_1, X_2, \ldots, X_n \rangle, V_2)$ constraints, where $V_1$ is restricted to be the number of maximal occurrences of `DecreasingTerrace` = ' $>=^{+}>$ ' in the signature of $X = \langle X_1, X_2, \ldots, X_n \rangle$, and $V_2$ is restricted to be the sum of the number of elements in subseries of $X$ whose signatures correspond to words of the language of `IncreasingTerrace` = ' $<=^{+}<$ '. In Fig. 3, for $n = 12$, the squared points represent feasible pairs $(V_1, V_2)$, while the circled points stand for infeasible pairs $(V_1, V_2)$ inside the convex hull. The linear invariant $2 \cdot V_1 + V_2 \leq n - 2$ is a facet of the polytope, which does not eliminate the points $(1, 8)$, $(2, 6)$, $(3, 4)$, $(4, 2)$. However, if we assume that both $V_1 > 0$ and $V_2 > 0$, then we can add a linear invariant eliminating these four infeasible points, namely $2 \cdot V_1 + V_2 \leq n - 3$, shown in Part (B) of Fig. 3. In addition, if we assume that $V_1 > 0$ and $V_2 > 0$, the infeasible points on the straight line $V_2 = 1$ will also be eliminated by the restriction $V_2 = 0 \vee V_2 \geq 2$ given in [3, p. 2598]. $\triangle$

Consider that each automaton $\mathcal{M}_i$ (with $i$ in $[1, k]$) returns its initial value after consuming the signature of an accepting sequence $X$ wrt $\mathcal{M}_i$ iff the signature of $X$ does not contain any occurrence of some regular expression $\sigma_i$ over the alphabet $\Sigma$. Let $\mathcal{M}'_i$ denote the automaton which accepts the words of the language $\Sigma^* \sigma_i \Sigma^*$, where $\Sigma^*$ denotes any word over $\Sigma$. Then, using the method of Section 3 we generate the invariants for $\mathcal{M}'_1 \cap \mathcal{M}'_2 \cap \cdots \cap \mathcal{M}'_k$. These invariants hold when the non-default value condition is satisfied.

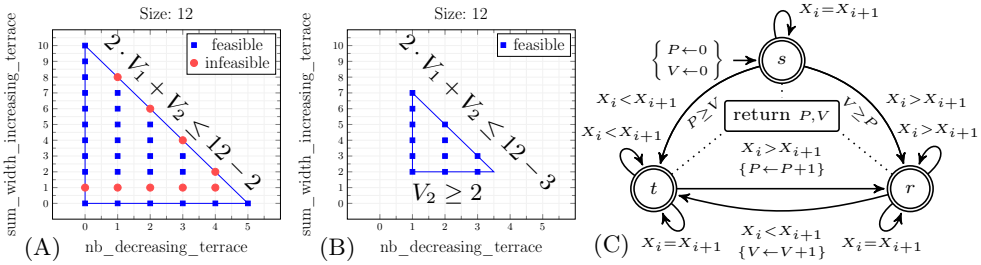



Fig. 3: Invariants on the result values $V_1$ and $V_2$ of `nb_decreasing_terrace` and `sum_width_increasing_terrace` for a sequence size of 12 (A) with the general invariants, and (B) with the Non-Default Value condition. (C) Intersection automaton for `Peak` and `Valley` with the guards $P \geq V$ and $V \geq P$ on transitions $s \to t$ and $s \to r$ (as for the return statement, the $P$ and $V$ accumulators in the guards refer to the final values of the corresponding accumulators).

### 4.2 Generating Guards for Transitions of the Intersection of Several Automata

Consider $k$ automata $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ and let $V_i$ (with $i \in [1, k]$) designate the value returned by $\mathcal{M}_i$. We focus on generating necessary conditions, called *guards*, introduced in Definition 3, for enabling transitions of the automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$. Further, we give a three-step procedure for generating guards for transitions of $\mathcal{I}$.

**Definition 3.** *Consider a transition $t$ of the automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$. A* guard *of $t$ is a linear inequality of the form $e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i \geq 0$ such that there does not exist any accepting sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$ wrt $\mathcal{I}$ such that (1) after consuming the signature of $X$, the vector $(V_1, V_2, \ldots, V_k)$ returned by $\mathcal{I}$ satisfies the inequality $e + e_0 \cdot n + \sum_{i=1}^{k} e_i \cdot V_i < 0$, (2) and the transition $t$ was triggered upon consuming the signature of $X$.*

The following example illustrates Definition 3.

*Example 7.* Given a sequence $X = \langle X_1, X_2, \ldots, X_n \rangle$, consider the `peak`$(X, P)$ and `valley`$(X, V)$ constraints. The intersection $\mathcal{I}$ of the automata for `peak` and `valley` was given in Part (C) of Fig. 1. Observe that, if at the initial state $s$ the automaton consumes '$<$' (resp. '$>$'), then the number of peaks (resp. valleys) in $X$ is greater than or equal to the number of valleys (resp. peaks). Hence, we can impose the guard $P \geq V$ (resp. $V \geq P$) on the transition from $s$ to $t$ (resp. to $r$). Part (C) of Fig. 3 gives the automaton $\mathcal{I}$ with the obtained guards. △

Guards for the transitions of an automaton $\mathcal{I} = \mathcal{M}_1 \cap \mathcal{M}_2 \cap \cdots \cap \mathcal{M}_k$ can be generated in three steps:

1. First, we identify the subset $\mathcal{T}$ of transitions of $\mathcal{I}$ such that, for any transition $t$ in $\mathcal{T}$, upon consuming any sequence, $t$ can be triggered at most once.
2. Second, for every transition $t$ in $\mathcal{T}$, we obtain a new automaton $\mathcal{I}_t$ by removing from $\mathcal{I}$ all transitions of $\mathcal{T}$ different from $t$ that start at the same state as $t$.
3. Third, using the technique of Section 3 on the invariant digraph $G^v_{\mathcal{I}_t}$, we obtain linear invariants that are guards of transition $t$.

## 5 Evaluation

To test the effectiveness of the generated invariants, we first try systematic tests on the conjunction of pairs of the 35 time-series constraints [5] of the *nb* and *sum_ width* families for which the glue matrix constraints exist [2]. The *nb* constraints count the number of occurrences of some pattern in a time series, while the *sum_ width* family constrains the sum of the width of pattern occurrences. Our intended use case is similar to [9], where constraints and parameter ranges of the problem are learned from real-world data, and are used to produce solutions that are similar to the previously observed data. It is important both to remove infeasible parameter combinations quickly, as well as helping to find solutions for feasible problems. Real world datasets often will only show a tiny subset of all possible parameter combinations, but as we don't know the data a priori, a systematic evaluation seems the most conservative approach.

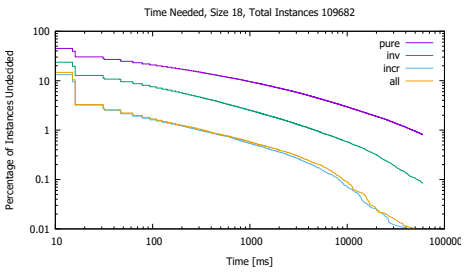
For the experiments we use a database of generated invariants in a format compatible with the Global Constraint Catalogue [3]. Invariants are generated as Prolog facts, from which executable code, and other formats are then produced automatically. The time required to produce the invariants (5 min) is insignificant compared to the overall runtime of the experiments. For the 595 combinations of the 35 constraints we produce over 4100 unconditional invariants, over 3500 conditional invariants, and 86 guard invariants. In the test, we try each pair of constraints and try to find solutions for all possible pairs of parameter values. We compare four different versions of our methods: The *pure* baseline version uses the automata that were described in [4], the bounds on the parameter values for each prefix and suffix, and the glue matrix constraints as described in [2]. This version represents the state of the art before the current work. In the *invariant* version we add the generated invariants for the parameters of the complete time series. In the *incremental* version, we not only state the invariants for the complete time series, but also apply them for each suffix. The required variables are already available as part of the glue matrix setup, we only need to add the linear inequalities for each suffix length. In the *all* version, we add the product automaton of the conjunction of the two constraints, if it contains guard constraints, and also state some additional, manually derived invariants.

The test program uses a labeling routine that first assigns the signature variables, and only afterwards assigns values for the $X_i$ decision variables. The variables in each case are assigned from left to right. For each pair of parameters

values, defined by the product of the bounds from [2], we try to find a first solution with a timeout of 60s.

We have tested the results for different time series length, Figure 4 shows the result for length 18 and domain size 0..18, the largest problem size where we find solutions for each case within the timeout. All experiments were run on a laptop with Intel i7 CPU (2.9GHz), 64Gb main memory and Windows 10 64bit OS using SICStus Prolog 4.3.5 utilizing a single core. For our four problem variants, we plot the percentage of undecided problem instances as a function of computation time. The plot uses log-log scales to more clearly show the values for short runtimes and for low number of undecided problems. The baseline *pure* variant solves around 55% of the instances immediately, and leaves just under one percent unsolved within the timeout. The *invariants* version improves on this by pruning more infeasible problems immediately. On the other hand, stating the invariants on the full series has no effect on feasible instances. When using the *incremental* version of the constraints, this has very little additional impact on infeasible problems, but improves the solution time for the feasible instances significantly. Adding (variant *all*) additional constraints further reduces the number of backtracks required, but these savings are largely balanced with the additional processing time, and therefore have no major impact on the overall results. After one second, around 9.5% of all instances are unsolved in the baseline, but only 0.5% in the *incremental* or *all* variant.

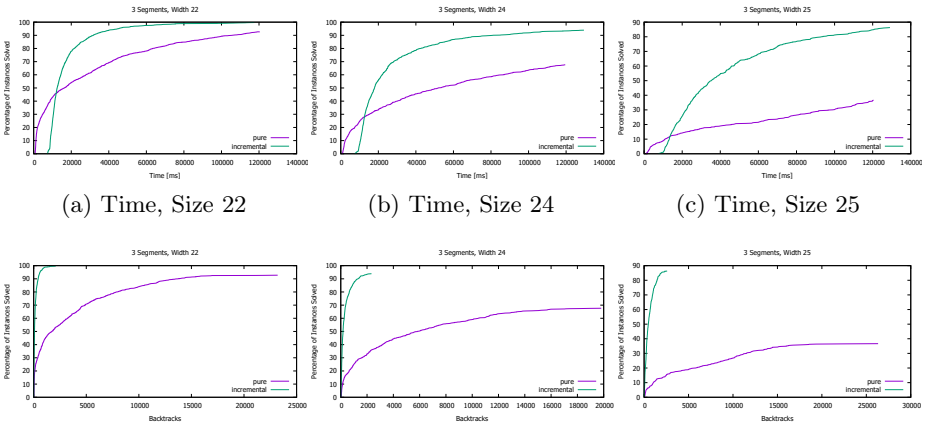
Fig. 4: Comparing Constraint Variants, Undecided Instances Percentage for Size 18 as a Function of Time, Timeout=60s




To test the method in a more realistic setting, we consider the conjunction of all 35 considered time-series constraints on electricity demand data provided by an industrial partner. The time-series describes daily demand levels in half-hour intervals, giving 48 data points. To capture the shape of the time-series more accurately, we split the series into overlapping segments from 00-12, 06-18, and 12-24 hours, each segment containing 24 data points, overlapping in 12 data points with the previous segment. We then setup the conjunction of the 35 time-series constraints for each segment, using the *pure* and *incremental* variants described above. This leads to $3 \times 35 \times 2 = 210$ automata constraints with shared signature and decision variables. The invariants are created for every pair of constraints, and every suffix, leading to a large number of inequalities. The search routine assigns all signature variables from left to right, and then assigns the decision variables, with a timeout of 120s.

In order to understand the scalability of the method, we also consider time series of 44 resp. 50 data points (three segments of length 22 and 25), extracted from the daily data stream covering a four year period (1448 samples). In Fig-

Fig. 5: Percentage of Problems Solved for 3 Overlapping Segments of Lengths 22, 24, and 25; Execution time in top row, backtracks required in bottom row




(a) Time, Size 22 (b) Time, Size 24 (c) Time, Size 25

(d) Backtracks, Size 22 (e) Backtracks, Size 24 (f) Backtracks, Size 25

ure 5 we show the time and backtrack profiles for finding a first solution. The top row shows the percentage of instances solved within a given time budget, the bottom row shows the percentage of problems solved within a backtrack budget. For easy problems, the *pure* variant finds solutions more quickly, but the *incremental* version pays off for more complex problems, as it reduces the number of backtracks required sufficiently to account for the large overhead of stating and pruning all invariants. The problems for segment length 20 (not shown) can be solved without timeout for both variants, as the segment length increases, the number of time outs increases much more rapidly for the *pure* variant. Adding the invariants drastically reduces the search space in all cases, future work should consider if we can identify those invariants that actively contribute to the search by cutting off infeasible branches early on. Restricting the invariants to such an active subset should lead to a further improvement in execution time.

## 6 Conclusion

Future work may look how to extend the current approach to handle automata with accumulators that also allow the min and max aggregators for accumulator updates. It also should investigate the use of such invariants within the context of MIP. While MIP has been using linear cuts for a long time [16,20], no off-the-shelf data base of cuts in some computer readable format is currently available. Cuts are typically defined in papers and are then directly embedded within MIP solvers.

## References


1. Appa, G., Magos, D., Mourtos, I.: LP relaxations of multiple `all_different` predicates. In: Régin, J.C., Rueher, M. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, First International Conference, CPAIOR 2004. LNCS, vol. 3011, pp. 364–369. Springer (2004)
2. Arafailova, E., Beldiceanu, N., Carlsson, M., Flener, P., Francisco Rodríguez, M.A., Pearson, J., Simonis, H.: Systematic derivation of bounds and glue constraints for time-series constraints. In: Rueher, M. (ed.) CP 2016. LNCS, vol. 9892, pp. 13–29. Springer (2016)
3. Arafailova, E., Beldiceanu, N., Douence, R., Carlsson, M., Flener, P., Rodríguez, M.A.F., Pearson, J., Simonis, H.: Global constraint catalog, volume ii, time-series constraints. CoRR abs/1609.08925 (2016), http://arxiv.org/abs/1609.08925
4. Arafailova, E., Beldiceanu, N., Douence, R., Flener, P., Francisco Rodríguez, M.A., Pearson, J., Simonis, H.: Time-series constraints: Improvements and application in CP and MIP contexts. In: Quimper, C.G. (ed.) CP-AI-OR 2016. LNCS, vol. 9676, pp. 18–34. Springer (2016)
5. Beldiceanu, N., Carlsson, M., Douence, R., Simonis, H.: Using finite transducers for describing and synthesising structural time-series constraints. Constraints 21(1), 22–40 (January 2016), journal fast track of CP 2015: summary on p. 723 of LNCS 9255, Springer, 2015
6. Beldiceanu, N., Carlsson, M., Rampon, J.X., Truchet, C.: Graph invariants as necessary conditions for global constraints. In: van Beek, P. (ed.) Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005. LNCS, vol. 3709, pp. 92–106. Springer (2005)
7. Beldiceanu, N., Contejean, E.: Introducing global constraints in CHIP. Mathl. Comput. Modelling 20(12), 97–123 (1994)
8. Beldiceanu, N., Flener, P., Pearson, J., Van Hentenryck, P.: Propagating regular counting constraints. In: Brodley, C.E., Stone, P. (eds.) AAAI 2014. pp. 2616–2622. AAAI Press (2014)
9. Beldiceanu, N., Ifrim, G., Lenoir, A., Simonis, H.: Describing and generating solutions for the EDF unit commitment problem with the ModelSeeker. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 733–748. Springer (2013)
10. Beldiceanu, N., Mats, C., Petit, T.: Deriving filtering algorithms from constraint checkers. In: Wallace, M. (ed.) Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004. LNCS, vol. 3258, pp. 107–122. Springer (2004)
11. Boyd, S.P., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)
12. Charnley, J.W., Colton, S., Miguel, I.: Automatic generation of implied constraints. In: ECAI 2006. Frontiers in AI and Applications, vol. 141, pp. 73–77. IOS Press (2006)
13. Dincbas, M., Simonis, H., Hentenryck, P.V.: Solving the car-sequencing problem in constraint logic programming. In: ECAI. pp. 290–295 (1988)
14. Francisco Rodríguez, M.A., Flener, P., Pearson, J.: Implied constraints for Automaton constraints. In: Gottlob, G., Sutcliffe, G., Voronkov, A. (eds.) GCAI 2015. EasyChair Proceedings in Computing, vol. 36, pp. 113–126 (2015)
15. Frisch, A., Miguel, I., Walsh, T.: Extensions to proof planning for generating implied constraints. In: 9th Symp. on the Integration of Symbolic Computation and Mechanized Reasoning (2001)

16. Gomory, R.: Outline of an algorithm for integer solutions to linear programs. Bulletin of the American Mathematical Society 64, 275–278 (1958)
17. Hansen, P., Caporossi, G.: Autographix: An automated system for finding conjectures in graph theory. Electronic Notes in Discrete Mathematics 5, 158–161 (2000)
18. Hooker, J.N.: Integrated Methods for Optimization. Springer Publishing Company, Incorporated, 2nd edn. (2011)
19. Lee, J.: All-different polytopes. J. Comb. Optim. 6(3), 335–352 (2002)
20. Marchand, H., Martin, A., Weismantel, R., Wolsey, L.A.: Cutting planes in integer and mixed integer programming. Discrete Applied Mathematics 123(1-3), 397–446 (2002)
21. Menana, J.: Automata and Constraint Programming for Personnel Scheduling Problems. Theses, Université de Nantes (Oct 2011), https://tel.archives-ouvertes.fr/tel-00785838
22. Menana, J., Demassey, S.: Sequencing and counting with the `multicost-regular` constraint. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings. pp. 178–192 (2009)
23. Pesant, G.: A filtering algorithm for the `stretch` constraint. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 183–195. Springer (2001)