Computing Alignments with Constraint Programming: The Acyclic Case

María Teresa Gómez-López[1], Diana Borrego[1], Josep Carmona[2], Rafael M. Gasca[1]

[1] Universidad de Sevilla, Seville, Spain,
{maytegomez,dianabn,gasca}@us.es
[2] Universitat Politècnica de Catalunya, Barcelona, Spain,
jcarmona@cs.upc.edu

Abstract. Conformance checking confronts process models with real process executions to detect and measure deviations between modelled and observed behaviour. The core technique for conformance checking is the computation of an alignment. Current approaches for alignment computation rely on a shortest-path technique over the product of the state-space of a model and the observed trace, thus suffering from the well-known state explosion problem. This paper presents a fresh alternative for alignment computation of acyclic process models, that encodes the alignment problem as a Constraint Satisfaction Problem. Since modern solvers for this framework are capable of dealing with large instances, this contribution has a clear potential. Remarkably, our prototype implementation can handle instances that represent a real challenge for current techniques. Main advantages of using Constraint Programming paradigm lie in the possibility to adapt parameters such as the maximum search time, or the maximum misalignment allowed. Moreover, using search and propagation algorithms incorporated in Constraint Programming Solvers permits to find solutions for problems unsolvable with other techniques.

Keywords: Conformance Checking, Constraint Programming

1 Introduction

Nowadays organizations analyze and use the huge amount of data that their information systems generate. This data represents an important source of information, since it contains many of the evidences an organization may need to know in order to reach its (business) goals. Among others perspectives, the focus on the process dimension is of paramount importance.

Process mining has evolved in the last decade to act as a meeting point between data and process science. Techniques in process mining enable the discovery of evidence-based process models, the conformance analysis and the enhancement of process models. Conformance analysis, which is the topic considered in this paper, studies the adequacy of a process model in describing the real behavior observed as a collection of traces denoting the footprints of the execution of

a process. While there exist several techniques for discovery and enhancement of process models, the current few techniques available for conformance analysis are not yet satisfactory.

In this paper we tackle a central problem in conformance analysis: the computation of an *alignment* between a process model and an *event log*. Informally, an alignment is a two-row matrix where the first row denotes the steps in the observed trace, while the second row describes the steps performed by the model in order to fit as much as possible the trace. Alignments are crucial to evaluate the important metrics in conformance, i.e., *fitness* and *generalization* [2] and *precision* [3].

We deviate from the current approaches for alignment computation, which are based on state-space explorations of models. Instead, we encode the problem of computing alignments as a *Constraint Satisfaction Problem* (CSP), and use a CSP solver to compute alignments. The CSP framework brings many advantages when compared to the state-of-the-art approaches for conformance analysis: a portfolio of available search techniques, natural encoding of certain model constructs, capability of handling large instances, ability to interact with the solver to obtain valid solutions, etc.

In this paper we consider the computation of alignments for acyclic process models. In spite of this model restriction, current techniques may still have problems to handle certain instances, as it was demonstrated in [14]. In our prototype implementation, we show how the approach presented in this paper may be a solid alternative when current approaches fail at deriving an alignment.

This paper is organized as follows: in Section 2 a brief introduction to Constraint Programming is provided, since it is the basis of the encoding presented in the rest of the paper. Then in Section 3 the encoding is shown, together with further extensions to optimize the computation of alignments. Then in Section 4 some the results on some instances from the literature are reported. Finally, Section 6 provides the current context for conformance analysis and Section 7 concludes and discusses current research directions.

2 Constraint Programming

A CSP represents a reasoning framework consisting of variables, domains and constraints, where the model is described declaratively. Formally, it is defined as a tuple $\langle X, D, C \rangle$, where $X = \{x_1, \ldots, x_n\}$ is a finite set of variables, $D = \{d(x_1), \ldots, d(x_n)\}$ is a set of domains of the values of the variables, and $C = \{C_1, \ldots, C_m\}$ is a set of constraints. Each constraint C_i is defined as a relation R on a subset of variables $V = \{x_i, x_j, \ldots, x_l\}$, called the *constraint scope*. The relation R may be represented as a subset of the Cartesian product $d(x_i) \times d(x_j) \times \ldots \times d(x_l)$. A constraint $C_i = (V_i, R_i)$ simultaneously specifies the possible values of the variables in V that satisfy R. Let $V_k = \{x_{k_1}, \ldots, x_{k_l}\}$ be a subset of X, and an l-tuple $(x_{k_1}, \ldots, x_{k_l})$ from $d(x_{k_1}), \ldots, d(x_{k_l})$ can therefore be called an

instantiation of the variables in V_k. An instantiation is a solution if and only if it satisfies the constraints C.

In order to solve a CSP, a combination of search and consistency techniques is commonly used [8][4]. The consistency techniques remove inconsistent values from the domains of the variables during or before the search. During the search, a propagation process is executed which analyses the combination of values of variables where the constraints are satisfiable. Several local consistency and optimization techniques have been proposed as ways of improving the efficiency of search algorithms.

When it is not only necessary to ascertain if a solution can be found, and it is important to find the best solution, a Constraint Optimization Problem (COP) can be created and solved. A COP is a CSP with an optimization function where only the tuple of possible values that optimize this function is determined as the solution of the COP. Constraint Programming has already been used to compare expected and observed behaviour to diagnose models according observations, and it has also been applied to business process models [9, 11, 5].

A simple example to illustrate the usage of a CSP can be found to represent the possible execution order of the activities of a model. Imagine a model where activity A must be executed first, and activities B or C must be executed after, but not both. Variables mod_A, mod_B, mod_C can be used to obtain the possible execution moments. And the constraints should represent that (1) A must be executed, (2) B or C must be executed (but only one), and (3) if B or C are executed, this will happen after the execution of A.

mod_A, mod_B, mod_C in the domain {0..n} //{0..model.size()}
mod_A>0 AND (mod_B>0 XOR mod_C>0) AND
if (mod_B≠0) then (mod_B>mod_A)
if (mod_C≠0) then (mod_C>mod_A)

With this CSP, some solutions provided by a constraint solver would be:

sol1: mod_A=1, mod_B=2, mod_C=0
sol2: mod_A=1, mod_B=0, mod_C=2
sol3: mod_A=1, mod_B=3, mod_C=0
...

An example of optimization function can be to $minimize(mod_A + mod_B + mod_C)$. In this case only *sol1* is obtained.

3 Alignment Computation with Constraint Programming

In this paper we propose to encode by means of a CSP the constraints that describe the possible execution order of the transitions in a Petri net (the expected behaviour), and the order of the transition in the logs (observed behaviour) following model-based diagnosis paradigm [10]. The COP will find the minimum misalignment between the observed and the expected transitions. The encoding consists in the creation of two sets of variables that represent, respectively, the

Petri net model (set called *Var-Model*), and the real observed behaviour registered in each case of the event log (set called *Var-Log*). These two sets have the same number of variables, since they are composed of all activities in the model, plus all activities appearing in the event log but not in the model. For the alignment computation, the constraints that represent the model are determined once, while the constraints that represent the event log depend on each case.

Considering all these activities, these two sets of variables represent the step order where each activity (transition in the Petri net) can be executed following the model (*Var-Model*) or in accordance to the event log (*Var-Log*). If it is possible to assign the same value to every variable in *Var-Model* and *Var-Log*, it implies that there is a total alignment between the model and the reality. This way, in order to model both sequences of activities (i.e. modelled and observed behaviour), each variable is modelled as an integer that is evaluated in accordance with the position that it takes in the execution order. Then, the positions assigned to each activity in the modelled (*Var-Model*) and expected (*Var-Log*) behaviours are compared to determine whether some event within a case in the event log is misaligned.

3.1 Modelling the Variables to represent the Petri Net

As mentioned, the expected and observed occurrences of activities should be modelled within the CSP, so that the modelled and observed sequences of execution of activities can be compared. Therefore, certain sets of variables should be part of the CSP, with the following meanings:

- *Var-Model*: Set of decision variables $\{mod_a, mod_b, \ldots, mod_n\}$ representing the position that all activities $a, b, \ldots, n$ take in the expected execution order, whose domains are Integers in $0..n$, being n the number of transitions plus the log size -i.e. the worst possible value of alignment-.
- *Var-Log*: Set of decision variables $\{log_a, log_b, \ldots, log_n\}$, representing the step order of the transitions in the observed trace, and whose domains are equal to the variables in *Var-Model*.
- *Var-Difference*: Set of n integer variables $\{dif_a, dif_b, \ldots, dif_n\}$, one for each transition, whose domains are {0, 1, 2}, to represent that: there is alignment between the observed and expected behaviour of the transition ($mod_x == log_x \rightarrow dif_x = 0$); the transition is in the modelled trace but not in the real trace or viceversa ($mod_x == 0$ XOR $log_x == 0 \rightarrow dif_x = 1$); or the transition is in both traces but in different positions in the execution order ($else \rightarrow dif_x = 2$). It holds whether there is alignment between the n-th values of *Var-Model* and *Var-Log*.
- *Var-Alignment*: Integer that represents the sum of all values in *Var-Difference*, representing the worst possible value of alignment. This value is used in the optimization function, since if this value can be set to 0, it means that the model and the event log are totally aligned.

In order to facilitate a clear understanding of the created COP, we use the example in Figure 1 to show the model and solutions obtained.

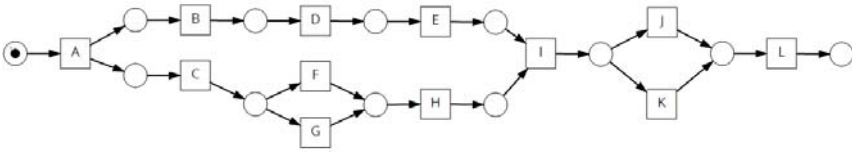

Fig. 1. Simple Petri Net

3.2 Modelling the Constraints to represent the Petri Net

The COP must include the five necessary parts: definition of variables, constraints to relate the order of the transitions in the model, constraints to describe the order of the log, constraints to determine the misalignment for each activity, and the objective function.

The modelling of the constraints in the COP is based on the transformation of the Petri net model into numerical constraints. For this reason, every place (and hence, the structure of the flow surrounding it) is analysed, and the following constraints are included into the COP to represent the control flow between the transitions. To differentiate the constraints that form the created COP, from the programming structures used to cover the Petri net to obtain the relations between the transitions, *italic* letters are used to distinguish constraints.

– **Start place (i.e. place with no input arcs):** Being $ot_1 \ldots ot_m$ the output transitions (as shown in Figure 2), the following constraint is part of the COP:

$$(mod_{ot_1} \neq 0 + \ldots + mod_{ot_m} \neq 0) = 1$$

For the example:

$$(mod_A \neq 0) = 1$$

– **Intermediate place (i.e. place with some input and output arcs):** Being $it_1 \ldots it_n$ the input transitions, and $ot_1 \ldots ot_m$ the output transitions (as shown in Figure 3), the following constraints are part of the COP:

FOR EACH pair it_i, ot_j
if $(mod_{ot_j} \neq 0)$ *then* $(mod_{ot_j} > mod_{it_i})$
END FOR
$(mod_{it_1} \neq 0 + \ldots + mod_{it_n} \neq 0) \leq 1$ *AND* $(mod_{ot_1} \neq 0 + \ldots + mod_{ot_m} \neq 0) \leq 1$
$(mod_{it_1} \neq 0 + \ldots + mod_{it_n} \neq 0) = (mod_{ot_1} \neq 0 + \ldots + mod_{ot_m} \neq 0)$

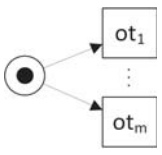

Fig. 2. Start place

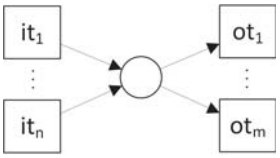

Fig. 3. Intermediate place

Meaning that:

- for each output transition ot_j, either it is not part of the execution, or it should be executed after the executed input transition ($mod_{ot_j} > mod_{it_i}$);
- and, if an input transition is executed, one and only one of the output transitions can be executed. Otherwise, none of them is executed.

Applied to the example:

//**A** $\rightarrow$ **B**//intermediate places
if($mod_B \neq 0$) then ($mod_B > mod_A$)
($mod_A \neq 0$)$\leq$1 AND ($mod_B \neq 0$)$\leq$1 AND ($mod_A \neq 0$)=($mod_B \neq 0$)
//the modelling of **B** $\rightarrow$ **D**, **D** $\rightarrow$ **E**, **E** $\rightarrow$ **I**, **A** $\rightarrow$ **C**, **H** $\rightarrow$ **I** is equivalent
//**C** $\rightarrow$ **(F xor G)**
if($mod_F \neq 0$) then ($mod_F > mod_C$)
if($mod_G \neq 0$) then ($mod_G > mod_C$)
($mod_C \neq 0$)$\leq$1 AND ($mod_F \neq 0$ + $mod_G \neq 0$)$\leq$1
($mod_C \neq 0$)=($mod_F \neq 0$ + $mod_G \neq 0$)
//the modelling of **I** $\rightarrow$ **J xor K** is equivalent
//**(F xor G)** $\rightarrow$ **H**
if($mod_H \neq 0$) then ($mod_H > mod_F$)
if($mod_H \neq 0$) then ($mod_H > mod_G$)
($mod_F \neq 0$ + $mod_G \neq 0$)$\leq$1 AND ($mod_H \neq 0$)$\leq$1
($mod_F \neq 0$ + $mod_G \neq 0$)=($mod_H \neq 0$)
//the modelling of **J xor K** $\rightarrow$ **L** is equivalent

– **End place (i.e. place with no output arcs):** Being $it_1 \ldots it_n$ the input transitions (as shown in Figure 4), the following constraint is part of the COP:

$(mod_{it_1} \neq 0 + \ldots + mod_{it_n} \neq 0) = 1$

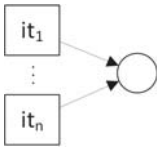

Fig. 4. End place

Applied to the example:

$(mod_L \neq 0)=1$

– Every transition a_i appearing in the case (from the event log) to check, but not in the model, is included as a variable mod_{a_i} in the set *Var-Model*, with the constraint:

$mod_{a_i}=0$

3.3 Modelling the Constraints to represent the Event Log

As it was aforementioned, the variables in the set *Var-Log* are created to study the positions in the execution order of both the elements appearing in a certain case and in the model. Therefore, different sets are created for each case in the event log, and then a different CSP is created for each case.

For every case in the event log, composed of activities presented as an ordered list a_1, a_2, ..., a_q, the constrains that should be created and included in the COP are:

$log_{a_1} > 0 \ AND \ log_{a_2} > log_{a_1} \ AND \ \ldots \ AND \ log_{a_q} > log_{a_{q-1}}$

Meaning that, since all events in the log were executed, they should have a value greater than 0, keeping the execution order recorded in the case.

Likewise, for each activity a_i appearing in the model but not in the log, the following constraint is included:

$log_{a_i}=0$

3.4 Modelling a COP to find the alignment between model and event log

The alignment can be described by the distance between the observed and the expected behaviour. The observed activity executions are represented by the variables in the set *Var-Log* while the expected behaviour is modelled by the set *Var-Model*. The minimization of the difference between them is the aim of the alignment. In our solution, it is modelled using the variables in the set *Var-Difference*, where each variable dif_{a_i} represents the difference between the expected and the observed behaviour for activity a_i (0, 1 or 2 as explained before). The sum of all variables in the set *Var-Different* is stored in the variable *Var-Alignment*, which is the value to minimize, objective of the optimization function.

FOR EVERY activity a_i DO:
if(log_{a_i}==mod_{a_i}) *then* (dif_{a_i}=0)
else if(log_{a_i}==0 $\vee$ mod_{a_i}==0) *then* (dif_{a_i}=1)
else (dif_{a_i}=2)
END

Following the theory of alignment to prohibit that two different activities can be executed in the same instant of time, the following constraints must be included:

FOR EACH pair of variables mod_i and log_j in Var-Model DO:
if ($mod_i \neq 0$) *then* ($mod_i \neq log_j$)
END

Finally, to include the objective function that minimize the summation of differences, the following constraints are included:

Var-Alignment = $\sum_{dif_i \in Var-Difference} dif_i$

minimize(Var-Alignment)

3.5 Some Evaluations of the example

Likewise, and depending on the case to check, the rest of the COP is defined. To illustrate this, three cases in the event log, and their resulting constraints, are shown as examples in the following:

- A fitting case: {A, C, B, F, D, E, H, I, J, L}

  //Activities in the case
  log_A>0 AND log_C>log_A AND
  log_B>log_C AND log_F>log_B AND
  log_D>log_F AND log_E>log_D AND
  log_H>log_E AND log_I>log_H AND
  log_J>log_I AND log_L>log_J
  //Activities in the model but not in the case
  log_G=0 AND log_K=0

- Unfitting case 1, since there is an activity in the model that should appear in the case (activity l): {A, C, B, F, D, E, H, I, J}

  //Activities in the case
  log_A>0 AND log_C>log_A AND
  log_B>log_C AND log_F>log_B AND
  log_D>log_F AND log_E>log_D AND
  log_H>log_E AND log_I>log_H AND
  log_J>log_I
  //Activities in the model but not in the case
  log_L=0 AND log_G=0 AND log_K=0

– Unfitting case 2, since there is an activity in the log that does not appear in a correct trace of the model although it is in the model (order of D and E): {A, C, B, E, D, F, H, I, J, L}

//Activities in the case
log_A>0 AND log_C>log_A AND
log_B>log_C AND log_F>log_B AND
log_G>log_F AND log_D>log_G AND
log_E>log_D AND log_H>log_E AND
log_I>log_H AND log_J>log_I AND
log_L>log_J
 	
//Activities in the model but not in the case
log_K=0

The automatic computation of these three examples obtains the resulting sets *Var-Model*, *Var-Log*, *Var-Difference* and the value of *Var-Aligment* shown in Figure 5.

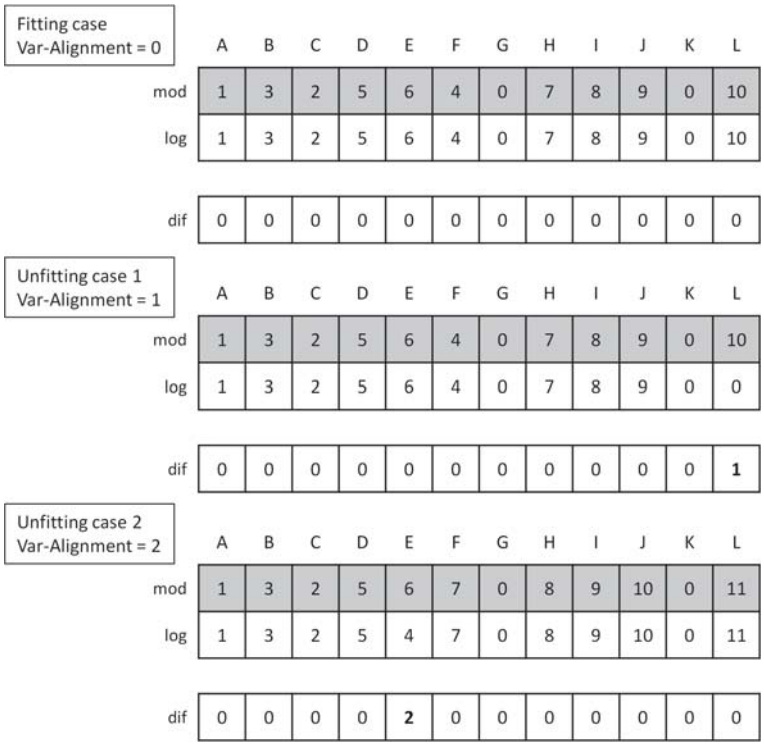

Fig. 5. Results of three case examples

3.6 Improvements to Reduce the Size of Search Space

The addressed problem presents a high computational complexity. However, thanks to the aforementioned characteristics of Constraint Programming, it can be improved by reducing the search space. In order to do it, some analyses of the Petri net graph have been performed to reduce the domain of the variables and the possible combinations, including more constraints into the problem and reducing the search:

- **Minimum and maximum number of zeros in the set *Var-Model*:** In order to reduce the search space by avoiding the exploration of non-valid solutions, the valuation of the variables in *Var-Model* (regarding the amount of zeros) is limited. Therefore, and since a zero in a variable of *Var-Model* means that the corresponding transition may not be executed, we can deduce from the model the minimum and maximum number of zeros that can appear in *Var-Model*. To ascertain the number of zeros, a COP is created with the constraints related to the model minimizing or maximize the summation of zeros to obtain *maxNumZeros* and *minNumZeros* respectively. The obtained values and the following constraints are included in the explained COP.

```
countNumZeros[Var-Model.size()] in the domain model.size()
int sumCountZeros in the domain
    [model.size()−maxNumZeros..model.size()−minNumZeros]

FOR EACH mod_i in Var-Model
    if(mod_i == 0)
      then (countNumZeros[i] = 1)
      else (countNumZeros[i] = 0)
END FOR

sumNumZeros = countNumZeros[1] + ... + countNumZeros[size]
```

- **Minimum distance between transitions:** In order to reduce the domain of the variables composing *Var-Model*, we execute the Floyd algorithm to obtain a Matrix that includes every minimum transition distance. With this information, the domain of the variables in *Var-Model* can be reduced, since the possible value assigned to a mod_i must take into account the values of the other variables to reach an correct trace in the model.

```
FOR EACH mod_i in Var-Model
    mod_i in the domain [0..size−floyd[i][closestEndTransition]]
END FOR
```

- **Maximum value in *Var-Log*:** A reduction in the domain of each variable composing *Var-Log* is also carried out following a similar idea. The possible order when a transition can be executed in the log, can take into account the events that appear after it.

FOR EACH log_{a_i} in Var-Log
  $log_{a_i} \leq size - case.size() + i$
END FOR

– **Mandatory transition relation:** A mandatory matrix is built, where each position [i][j] of the matrix can take two values {0, 1}, being 0 if every correct case that includes the transition i must include the transition j, and 1 otherwise. Thus, for every 1 found in a position [i][j], a constraint is created:

FOR EACH pair a_i, a_j of transitions
  IF (a_i is a start transition)
    THEN *($mod_{a_j} >$ floyd[i][j])*
    ELSE *(if ($mod_{a_i} \neq 0$) then ((floyd[i][j]+mod_{a_i}) $\leq mod_{a_j}$)*
  END IF
END FOR

– **Define Goal to Propagate Solutions:** Since the CSPs analyse all the promising possible values of the variables, in order to reduce this analysis we propose to include in the COP a goal over the *Var-Log* variables to generate only the possible values of this array and propagate the solution to the rest of variables. This enables the search to stop the instantiation in the branches where no new values of decision variables can be found, thereby bounding the unnecessary combinations of values for the rest of the variables but ensuring that there exists a solution inside.

4 Evaluation

This proposal has been implemented using Jsolver^{TM} as COP solver, *XES* library and *PNML* framework[3]. We used the benchmarks from [14], which are hard conformance instances, most of them representing a challenge for the state-of-the-art alignment technique from [2].

The COPs have been tested using the example whose characteristics are shown in Table 1. All *XES* for each Petri net in *PNML* are formed by several cases, 500 for C_petri and 1200 for the rest. Each case is evaluated in an independent way, it implies that the files of *PNML* and *XES* are load from the beginning, and the mentioned algorithms to analyse the Petri net graph are repeated for each case and included in the evaluation times of the table.

Since some of the examples are very hard, Constraint Programming offers the possibility to know the best solution found so far in the exploration. With this capacity we can wait or abort the search keeping a possible solution, although sometimes it cannot be ensured that this is the best solution. The information about the tests is:

– **Test:** Name of the Petri net analysed in the test.

[3] The test cases are measured using a Windows 10 machine, with an Intel Core I7 processor, 3.4 GHz and 32.0 GB RAM.

– **Num. Trans.:** Number of transitions in the Petri net.
– **Num. Places:** Number of places in the Petri net.
– **Width:** Minimum distance from a start place to an end place, equal to the position {startTransition, endTransition} in the Floyd Matrix.
– **Min. Num of Trans.:** The minimum number of transitions that can appear in a correct case for the model (model.size()−maxNumZeros).
– **Max. Num of Trans.:** The maximum number of transitions that can appear in a correct case for the model (model.size()−minNumZeros).
– $\sum$ **time:** The summation of the evaluation time for every cases (1200). This value is included only if we can ensure that the minimum is found in a bounded time (one minute in our case).
– **Minimal found?:** Field to indicate if we can ensure that the minimum is found or we have abort the execution. The possibilities are *Yes* or *Not*, to describe if the best solution in a bounded time is found for every case.
– **Best solution in a Bounded Time:** When the minimum cannot be ensured to be found, we need to decide the time that we are going to wait until stopping the process. We have included some examples of the minimum misalignment found and the spending time to obtain this solution.

Table 1. Tests of the proposal

Test	Num. Trans.	Num. Places	Width	Min. Num of Trans.	Max. Num of Trans.	$\sum$ time	Minimal found?	Best sol. in a Bounded Time
prAm6	363	347	16	19	42	1703.829s	All	
prBm6	317	317	29	14	59	1564.647s	All	
prCm6	317	317	30	14	59	22754.087	All	
prDm6	429	529	17	235	271	-	None	19 in 18.426s 26 in 15.590s 26 in 16.201s 24 in 17.922s 25 in 18.524s
prEm6	275	277	26	80	117	2976.404s	All	
prFm6	299	362	28	234	245	57390.782s	All	
prGm6	335	337	32	123	160	-	None	6 in 55.778 6 in 36.988 8 in 61.567 7 in 71.363 9 in 34.586

Analysing Table 1, it is possible to ascertain that the evaluation time is not totally related to the size of the Petri net, nor with the number of variables. Actually there exists a dependency between the evaluation time and the size of the log. This size is not included in the table since they are 1200 different for each case, but the *min.* and *max. number of transitions* determine the bound of the log size in each case. We can observe that two COPs for the same model, but for

different log (then with the same number of variables), can have very different evaluation times. Our proposal found solutions for the examples *prDm6*, *prEm6*, *prFm6*, *prGm6*, whereas they cannot be found in previous works [14].

5 Benefits and Limitations of the proposal

This proposal presents three main benefits: 1) the solutions found during the search can be known, then we can abort the process when considering that the alignment is good enough, or even to bound the waiting time; 2) the capacity to interact with the model, by allowing the parametrization of values for each particular case and model; and 3) Constraint solver tools are in constant development, therefore every improvement in constraint programming research can be easily included in our implementation. Some example of the variables that can be parametrized are:

- Limitation of the maximum search time.
- A lower bound on the degree of alignment.
- Maximum or minimum allowed misalignment.
- Types of supported cases, described by means of: 1) those containing a minimum or maximum number of transitions; 2) those containing a set of transitions in a mandatory way.

Among the limitations of the current state of our proposal are that:

- Only Petri nets with one token is considered, and different transitions with the same name are not allowed.
- It is restricted to acyclic process models.
- Only safe nets are considered and with exactly one input and output place.
- For some cases, we cannot always ensure that the found solution is minimal.

6 Related Works

The seminal work by [15] represents the first attempt to relate observed and modelled behaviour. Given an observed trace, it is based on heuristically replaying it on the process model. Although in practice this approach may be very useful for large models, for indeterministic models it cannot guarantee the existence of a solution even though it exists. Evolutions of this replay technique can also be find in recent work [6, 7], which also inherit the aforementioned fundamental problem of replaying techniques.

The work in [2] proposed the notion of alignment for the first time, and developed a technique to compute optimal alignments for a particular class of process models. The approach is implemented in ProM, and can be considered as the state-of-the-art technique for computing alignments. Unfortunately, the alignment techniques in [2] cannot handle large inputs.

Decompositional techniques have been recently presented [1, 14] that instead of computing optimal alignments, they focus on the *decisional problem* of deciding whereas a given trace fits or not a process model. The underlying idea is to split the model into a particular set of transition-bordered fragments which satisfy certain conditions, and local alignments can be computed for each one of the fragments, thus providing a upper bound on the cost of an alignment.

Finally, the work in [13, 12] considers the problem of dealing with partially ordered event data, a common situation in certain context like healthcare. The notion of *partially ordered alignment* is introduced, and a variation of the techniques presented in [2] is used for its computation.

7 Conclusions and Future work

The Constraint Programming paradigm has been used in various and complex scenarios where different combinations and possibilities need to be analysed, obtaining promising solutions. For this reason, we have considered interesting to carry out the alignment computation in conformance checking by using Constraint Programming. The creation and solution of Constraint Optimization Problems allows to incorporate in the computation of alignments propagation algorithms incorporated in Constraint Programming Solvers. It has enabled to find solutions for complex Petri nets and logs that represent a challenge for other techniques. Obviously there exist computational complex problems, very hard to optimize also using Constraint Programming. For this reason, and since the solutions found during the search can be known, the user has the choice to stop the search when a satisfactory (perhaps not minimal) solution is found in the exploration.

In addition, the use a declarative model as Constraint Programming facilitates the incorporation of parameters in the problems, with no extra implementation, such as the maximum time of search of solutions, or the maximum and minimum misalignment allowed.

For the future, several research directions will be taken. First, extending the theory to deal with models containing cyclic behaviour is a necessary step. Second, we plan to explore the correlation between the characteristics of the model and the log, and the evaluation time. Also we consider very interesting to combine various cases to find the minimal diagnosis of the model or the log, to ascertain the activities responsible of a misalignment.

Acknowledgments. This work has been partially funded by the Spanish Ministry for Economy and Competitiveness (TIN2015-63502-C3-2-R, TIN2013-46181-C2-1-R) and the European Regional Development Fund (ERDF/FEDER).

References

1. van der Aalst, W.M.P.: Decomposing Petri nets for process mining: A generic approach. Distributed and Parallel Databases 31(4), 471–507 (2013), `http://dx.doi.org/10.1007/s10619-013-7127-5`

2. Adriansyah, A.: Aligning observed and modeled behavior. Ph.D. thesis, Technische Universiteit Eindhoven (2014)
3. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Measuring precision of modeled behavior. Inf. Syst. E-Business Management 13(1), 37–67 (2015), http://dx.doi.org/10.1007/s10257-014-0234-7
4. Apt, K.: Principles of Constraint Programming. Cambridge University Press, New York, NY, USA (2003)
5. Borrego, D., Eshuis, R., Gómez-López, M.T., Gasca, R.M.: Diagnosing correctness of semantic workflow models. Data Knowl. Eng. 87, 167–184 (2013)
6. vanden Broucke, S.K.L.M., Munoz-Gama, J., Carmona, J., Baesens, B., Vanthienen, J.: Event-based real-time decomposed conformance analysis. In: On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings. pp. 345–363 (2014)
7. vanden Broucke, S.K.L.M., Weerdt, J.D., Vanthienen, J., Baesens, B.: Determining process model precision and generalization with weighted artificial negative events. IEEE Trans. Knowl. Data Eng. 26(8), 1877–1889 (2014), http://dx.doi.org/10.1109/TKDE.2013.130
8. Dechter, R.: Constraint Processing (The Morgan Kaufmann Series in Artificial Intelligence). Morgan Kaufmann (May 2003)
9. Gómez-López, M.T., Gasca, R.M., Pérez-Álvarez, J.M.: Compliance validation and diagnosis of business data constraints in business processes at runtime. Inf. Syst. 48, 26–43 (2015)
10. Gómez-López, M.T., Gasca, R.M., Rinderle-Ma, S.: Explaining the incorrect temporal events during business process monitoring by means of compliance rules and model-based diagnosis. In: 17th IEEE International Enterprise Distributed Object Computing Conference Workshops, EDOC Workshops, Vancouver, BC, Canada, September 9-13, 2013. pp. 163–172 (2013)
11. Gómez-López, M.T., Parody, L., Gasca, R.M., Rinderle-Ma, S.: Prognosing the compliance of declarative business processes using event trace robustness. In: On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings. pp. 327–344 (2014)
12. Lu, X., Fahland, D., van der Aalst, W.M.P.: Conformance checking based on partially ordered event data. In: Business Process Management Workshops - BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers. pp. 75–88 (2014)
13. Lu, X., Mans, R., Fahland, D., van der Aalst, W.M.P.: Conformance checking in healthcare based on partially ordered event data. In: Proceedings of the 2014 IEEE Emerging Technology and Factory Automation, ETFA 2014, Barcelona, Spain, September 16-19, 2014. pp. 1–8 (2014), http://dx.doi.org/10.1109/ETFA.2014.7005060
14. Munoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Single-entry single-exit decomposed conformance checking. Inf. Syst. 46, 102–122 (2014), http://dx.doi.org/10.1016/j.is.2014.04.003
15. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64–95 (2008)