An Evolutionary Technique to Approximate Multiple Optimal Alignments

Farbod Taymouri and Josep Carmona

Universitat Politècnica de Catalunya, Barcelona (Spain)
{taymouri,jcarmona}@cs.upc.edu

Abstract. The alignment of observed and modeled behavior is an essential aid for organizations, since it opens the door for root-cause analysis and enhancement of processes. The state-of-the-art technique for computing alignments has exponential time and space complexity, hindering its applicability for medium and large instances. Moreover, the fact that there may be multiple optimal alignments is perceived as a negative situation, while in reality it may provide a more comprehensive picture of the model's explanation of observed behavior, from which other techniques may benefit. This paper presents a novel evolutionary technique for approximating multiple optimal alignments. Remarkably, the memory footprint of the proposed technique is bounded, representing an unprecedented guarantee with respect to the state-of-the-art methods for the same task. The technique is implemented into a tool, and experiments on several benchmarks are provided.

1 Introduction

Current conformance checking techniques strongly rely on *alignments*: given an observed trace representing a process instance, to find the best model trace that resembles it [1]. This way, the best model explanation of the reality is reported, so that one sees the reality through the model' perspective. This opens the door for further techniques, including root-cause analysis, model enhancement and predictive monitoring.

Since the reality can be explained in many ways, costs need to be defined on the deviations so that certain explanations are rendered less interesting, coining the notion of *optimal alignment*. In spite of this, many different optimal explanations may exist for a given trace, a concept denoted *all-optimal alignments* in [1]. The derivation of more than one explanation may provide a better, more global, analysis: for instance, to estimate the *precision* of a process model in describing an event log, different metrics are defined in [2] when considering all or just one optimal alignment.

Due to the existence of concurrency and iteration, the behaviour of underlying process models can be exponential, a fact that hampers the application of the state-of-the-art technique for computing alignments, which is based on exploring the model state space using A^* search [1]. The situation becomes significantly worse in case multiple optimal alignments need to be computed, since none of the heuristics to speed-up the search is applicable, and therefore the full exploration of the model' search space is then unavoidable. It is well-known (e.g., [3]) that the memory requirements of the A^*-based alignment technique are the most limiting factor to apply it on the large.

Taymouri, F., Carmona, J. An evolutionary technique to approximate multiple optimal alignments. A: International Conference on Business Process Management. "Business Process Management,16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018: proceedings". Berlín: Springer, 2018, p. 215-232.
The final authenticated version is available online at https://doi.org/10.1007/978-3-319-98648-7_13

In this paper we propose an evolutionary technique to approximate multiple optimal alignments. We trade-off computation time for memory, i.e., assume that in some contexts, it is acceptable to spend more time in the computation, provided that the memory footprint is guaranteed to not exceed a given bound. To accomplish this, we encode the computation of alignments as a *genetic algorithm* (GA), where tailored *crossover* and *mutation* operators are applied to an initial population of candidate model explanations. This way, the derivation of a set of alignments is the result of genetic evolution.

The technique proposed has some weakness that should be reported: first, it can only provide optimal alignments when certain conditions are satisfied (variability in the population and genetic convergence). In practice, however, the number of iterations may be decided a priori, which may be insufficient for genetic convergence, and the initial population may not contribute to reach optimal solutions. Second, the number of optimal alignments obtained is in practice inferior to the real number of all optimal alignments, due to the dependence to the initial population and genetic convergence. Hence, the proposed technique only approximates several optimal alignments.

In spite of the approximation nature of the technique proposed, we still see a clear value for several reasons: first, to obtain more than one model explanation of an observed trace may open the door to apply a posteriori root-cause analysis to identify the most likely explanation, as has been described in [4]. Second, the technique proposed represents the first algorithmic alternative to search for multiple optimal alignments, which can be applied on large instances under bounded memory. In the same way as GA provided an interesting perspective for process discovery [5,6,7], this work contributes to open a research direction for computing alignments on the large. Third, in contrast to the A^*-based alignment technique, our technique is non-deterministic in providing alignments, so that two runs of the method may obtain different result. This may be very useful in *multi-perspective alignments* [8]: since control-flow is aligned before other perspectives, randomness in the generation of the control-flow alignment will enable the exploration of a broader solution space in the rest of perspectives.

The paper is organized as follows: in Sec. 2 we provide related work. In Sec. 3, the necessary ingredients to understand the contents of this paper are presented. Then in Sec. 4 we describe the encoding as a GA of the problem of searching several best model explanations. The general framework for approximating multiple optimal alignments is described in 5. Tool support and experiments with various benchmarks are reported in Sec. 6. Finally, conclusions and pointers to future work are reported in Sec. 7.

2 Related Work

The work in [1] proposed the notion of alignment, and developed a technique to compute optimal alignments for a particular class of process models. For each trace σ, the approach consists on exploring the synchronous product of model's state space and σ. In the exploration, the shortest path is computed using the A^* algorithm, once costs for model and log moves are defined. The approach represents the state-of-the-art technique for computing alignments, and can be adapted (at the expense of increasing significantly the memory footprint) to provide all optimal alignments.

Alternatives to the A^* have appeared very recently: in the approach presented in [9], the alignment problem is mapped as an *automated planning* instance. Unlike the A^*, the aforementioned work is only able to produce one optimal alignment (not all optimal), but it is expected to consume considerably less memory. Automata-based techniques have also appeared [10,11]. In particular, the technique in [10] can compute all optimal alignments. The technique in [10] relies on state space exploration and determinization of automata, whilst the technique in [11] is based on computing several subsets of activities and projecting the alignment instances accordingly.

The work in [12], presented the notion of *approximate* alignment to alleviate the computational demands of the current challenge by proposing a recursive paradigm on the basis of structural theory of Petri nets. In spite of resource efficiency, the solution is not guaranteed to be executable. A follow-up work of [12] is presented in [13], which proposes a trade-off between complexity and optimality of solutions, and guarantees executable properties of results. The technique in [3], presents a framework to reduce a process model and the event log accordingly, with the goal to alleviate the computation of alignments. The obtained alignment, which is called *macro-alignment* since some of the positions are high-level elements, is expanded based on the gathered information during the initial reduction. Decompositional techniques have been presented [14], [15] that instead of computing optimal alignments, they focus on the *decisional problem* of whereas a given trace fits or not a process model.

3 Preliminaries

3.1 Petri Nets, Event Logs and Parikh vector representations of Traces

A *Petri Net* [16] is a 3-tuple $N = \langle P, T, \mathcal{F} \rangle$, where P is the set of places, T is the set of transitions, $P \cap T = \emptyset$, $\mathcal{F} : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$ is the flow relation. A *labeled Petri net (LPN)* is a 3-tuple $\langle N, \Sigma, \ell \rangle$, where N is a Petri net, Σ is an alphabet (a set of labels) and $\ell : T \rightarrow \Sigma \cup \{\tau\}$ is a *labeling function* that assigns to each transition $t \in T$ either a symbol from Σ or the empty symbol τ.

Given an alphabet of events $\Sigma = \{a_1, \ldots, a_n\}$, a trace is a word $\sigma \in \Sigma^*$ that represents a finite sequence of events. An *event log* $L \in \mathcal{B}(\Sigma^*)$ is a multiset of traces[1]. $|\sigma|_a$ represents the number of occurrences of a in σ. The *Parikh vector* of a sequence of events σ is a function $\widehat{}: \Sigma^* \rightarrow \mathbb{N}^n$ defined as $\widehat{\sigma} = (|\sigma|_{a_1}, \ldots, |\sigma|_{a_n})$.

Workflow processes can be represented in a simple way by using Workflow Nets (WF-nets). A WF-net is a Petri net where there is a place $start$ (denoting the initial state of the system) with no incoming arcs and a place end (denoting the final state of the system) with no outgoing arcs, and every other node is within a path between $start$ and end. For the sake of simplicity, in this paper we assume WF-nets.

3.2 Alignment of Observed Behavior

[1] $\mathcal{B}(\Sigma)$ denotes the set of all multisets of the set Σ.

The notion of aligning event log and process model was introduced by [1]. To achieve an alignment between a process model and a observed trace, we need to relate *moves* in the trace to *moves* in the model. When some of the moves in the observed trace can not be mimicked by the model or vice versa, there is an *asynchronous move*. When both model and trace agree in the performed label a *synchronous move* arises. For instance, consider the model M_1 in Fig. 1, with the following labels, $\ell(t_1) = a_1, \ell(t_2) = a_2, \ell(t_3) = a_3$ and $\ell(t_4) = a_4$, and trace $\sigma = a_1a_1a_4a_2$; two possible alignments between M_1 and σ are:

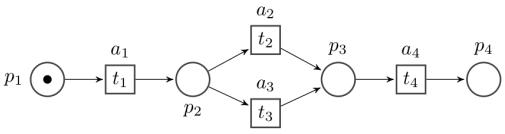

Fig. 1: Process model M_1.

$$\alpha_1 = \begin{array}{|c|c|c|c|c|} \hline a_1 & a_1 & \bot & a_4 & a_2 \\ \hline t_1 & \bot & t_3 & t_4 & \bot \\ \hline \end{array} \qquad \alpha_2 = \begin{array}{|c|c|c|c|c|} \hline a_1 & a_1 & \bot & a_4 & a_2 \\ \hline \bot & t_1 & t_2 & t_4 & \bot \\ \hline \end{array}$$

The moves are represented in tabular form, where moves in the observed trace are at the top and moves by model are at the bottom. For example the first move in α_2 is asynchronous: $(a_1, \bot)$, and it means that the observed trace performs a_1 while the model does not make any move, i.e., a_1 is an *inserted* transition. In contrast, the fourth move in α_2, (a_4, t_4), is a synchronous move. Cost can be associated to alignments, with asynchronous moves having greater cost than synchronous ones [1]. Given assigned cost values, an alignment with optimal cost is preferred. Alignments open the door to compute metrics, report diagnosis, enhance the model, among others.

4 GA for Computing Several Explanations of Observed Behavior

GA starts by creating an initial population, and then combining the best solutions through operators, to create a new generation of solutions which should be better than the previous generation. As it will be noticed bellow, in some cases the evaluation of solutions will be adapted depending on the operator applied, so that the search for solutions can be better guided. A GA approach to a problem usually starts by encoding a solution which is called a *chromosome*, and define functions to evaluate how good it is. Next, generating the initial population of chromosomes and defining corresponding operators, i.e., *crossover* and *mutation*. In our setting, chromosomes will be potential model traces, which are combined through tailored crossover and mutation operators.

Given an observed trace σ and WF-net N, a random population of chromosomes is first generated (Sect. 4.1). Then, it evaluates each chromosome based on a specific fitness function[2], which considers both the initial model (for measuring replayability), and the observed trace (for measuring similarity) (see Sect. 4.2). It then applies traditional crossover and mutation operators, as well as novel ones defined for this problem, to speed up the process of evolving chromosomes and convergence (see Sect. 4.3). This process continues until reaching satisfactory results, or will be stopped by a predefined number of iterations. The detailed descriptions will be presented in the next sections.

[2] As the reader will soon realize, we refer to the term fitness in the genetic algorithms context.

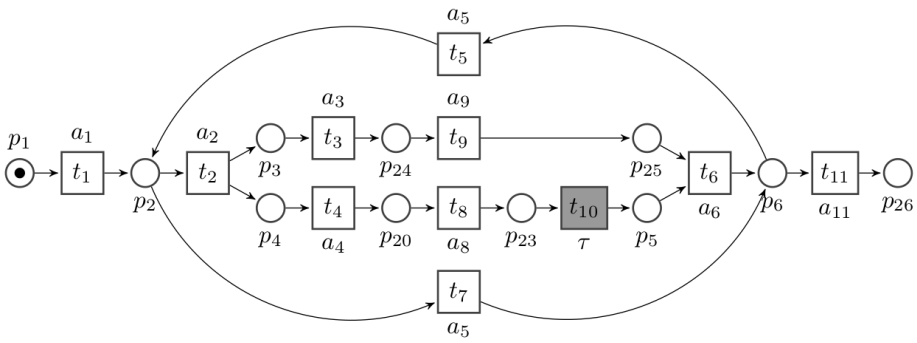

Fig. 2: Process model M_2.

4.1 Generation of the Initial Population

Given an observed trace σ, and WF-net N, the objective of this part is to generate an initial population. The population size is an important decision, which often affects the final solution in terms of accuracy and convergence [17]. Also, diversity in the population will help reaching different parts of the solution space. We rely on previous work for obtaining different model explanations [3,12]: these methods are based on finding maximal sets of transitions (called *Parikh* vectors [12]) that the model can reproduce to mimic the observed trace. The model traces arising from these sets, together with random traces, are used as seeds for generating new chromosomes.

For example consider the model in Fig. 2, and the observed trace $\sigma = a_1a_3a_8a_4a_2$ $a_9a_5a_6$. Some chromosomes with respect to σ could be $\chi_1 = t_1t_7t_{11}$, $\chi_2 = t_1t_2t_3t_9$ $t_4t_8t_{10}t_6t_{11}$, $\chi_3 = t_2t_1t_3t_8t_4t_9t_{10}t_6t_{11}$ and $\chi_4 = t_1t_7t_5t_{11}$. It is worth mentioning that some chromosomes may not be replayable at this stage (e.g., χ_4 above).

4.2 Evaluation Criteria

In GA's jargon a fitness function is a particular type of objective function that prescribes the optimality of a solution (that is, a chromosome) in the corresponding population. Elevated chromosomes, which are the best ones at the corresponding time are allowed to breed and mix their datasets by any of several techniques, producing a new generation that will (hopefully) be even better. An ideal fitness function correlates closely with the algorithm's goal, and yet may be computed quickly. Speed of execution is very important, as a typical GA must be iterated many times in order to produce a usable result for a non-trivial problem.

In this paper a chromosome χ is evaluated based on two metrics. The fitness value of a chromosome χ is summed over the following terms:

$$f^t(\chi) = \lambda_1 \cdot f^m(\chi) + \lambda_2 \cdot f^{ed}(\chi) \tag{1}$$

where $f^m(\chi)$ denotes the ratio of *missed tokens*[3] to the total tokens while χ is being replayed in the model, and $f^{ed}(\chi)$ shows the normalized *Edit Distance* between χ, and observed trace σ. Both λ_1 and λ_2 denote the penalization terms which will be adjusted individually for each genetic operator, as will be discussed in the next sections. It is clear that the lower value of $f^t(\chi)$ represents a better chromosome, i.e., modeled trace, by which the observed trace is mimicked. It should be pointed out that always having a chromosome χ with small $f^t(\chi)$ does not represent a desired or good solution if it is not replayable (i.e., $f^m(\chi) \neq 0$).

To get the idea of evaluation criteria consider chromosome $\chi = t_1 t_2 t_9 t_3 t_8$ $t_4 t_{10} t_{11} t_6$, the model in Fig. 2 and observed trace $\sigma = a_2 a_1 a_3 a_9 a_8 a_4 a_{26} a_6$; the number of missed and total tokens while χ is replayed equals to 3 and 23 respectively, thus $f^m(\chi) = \frac{3}{23}$. Additionally, unreplayable transitions t_9, t_8 and t_{11}, are likely to be considered through genetic operators, in the next step of the proposed approach. Also, the corresponding edit distance, i.e., $f^{ed}(\chi)$, between χ ans σ[4] is 5. Thereby by selecting $\lambda_1, \lambda_2 = 1$, the corresponding fitness value is $f^t(\chi) = 1 * \frac{3}{23} + 1 * 5 \approx 5.130$.

4.3 Genetic Operators

Genetic operators used in GA are analogous to those which occur in the natural world: survival of the fittest, or selection; reproduction (crossover); and mutation. When GA proceeds, both the search direction to optimal solution and the search speed should be considered as important factors, in order to keep a balance between exploration and exploitation in search space. In general, the exploitation of the accumulated information resulting from GA search is done by the selection mechanism, while the exploration to new regions of the search space is accounted for by genetic operators.

In the remainder of this section, several genetic operators will be proposed. Some of them are inspired from ones found in analogous problems, whilst new ones are proposed that tend to improve the evaluation criteria described in the previous section.

Crossover operators Crossover is the main genetic operator. It operates on two chromosomes at a time and generates two new chromosomes by combining both chromosomes' features. A standard way to achieve crossover is to choose a random segment at both chromosomes, and generate two new chromosomes as the result of interchanging the two segments among the original two chromosomes. We apply an adaptation of this standard crossover operator, denoted *Modified Partially-Mapped Crossover (MPMX)*, for chromosomes having the same Parikh vector representation (see 3.1)[5]. The intuitive idea for operating over chromosomes with identical Parikh vector is due to the fact that the search space is reduced, and in particular the generation of the initial population is oriented towards satisfying this property. In order to keep Parikh vector representation of the original chromosomes, some modifications are done after the segments are interchanged. Let us look at the example in Fig. 3 to illustrate the MPMX operator;

[3] In Petri net terms, missed tokens represent tokens that hamper the firing of a transition.

[4] Note that indeed the edit distance is computed between σ and $\ell(\chi)$.

[5] The restriction on having the same Parikh vector is for the sake of simplicity of application.

the initial chromosomes χ_1 and χ_2 are mixed with this operator, generating the new chromosomes χ_3 and χ_4, choosing a segment between positions 4 and 6.

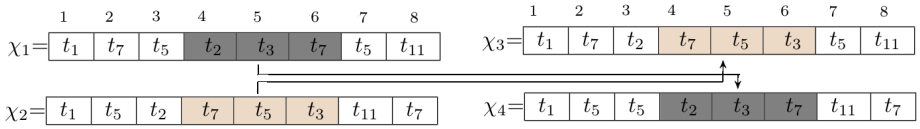

Fig. 3: The MPMX operator.

To keep the Parikh vector representation of the original chromosomes, some modifications are performed circularly starting from the first position after the segment (in the example, position 7). For instance, in χ_3 (that arised from χ_2 inserting the segment from χ_1), in the third position t_5 is removed since $|\chi_1|_{t_5} = 2$ and when we reach this position we already have 2 occurrences of t_5.

The next crossover operator, denoted *Cross-Insert Crossover (CIX)*, tries to guide the search towards chromosomes that are replayable in the model. Still, the CIX operator works under the assumption of both initial chromosomes have the same Parikh vector representation. To induce replayability, it focuses on the parts of a chromosome that are not replayable, and uses the other chromosome in order to find candidate positions where it may be possible to reply the set of unreplayable transitions. This is done for each unreplayable transition in each one of the chromosomes that are merged. For each candidate position, the transition is moved to that position and the chromosome is shifted accordingly to fill the space left. For instance, let us look at the two chromosomes χ_1 and χ_2 in Fig.4, and model M_2.

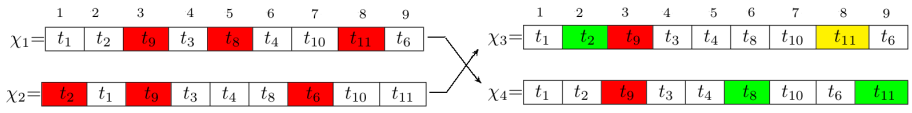

Fig. 4: The CIX operator: in the figure, red background means unreplayable positions of the trace, while green denotes positions in the new chromosomes where unreplayable transitions have been fixed. Yellow denotes transitions that, in spite of being initially replayable, due to other moves, they became unreplayable.

In χ_1 the transitions in the third, fifth and the eighth position cannot be replayed, namely t_9, t_8 and t_{11}. Transition t_9 cannot be moved, since in both chromosomes it is unreplayable (so there is no candidate position in this case). However, for transition t_8 in χ_1 (which is at position 5) there is a candidate position (position 6, extracted from χ_2) to move. Moving t_8 to position 6 and shifting once from position 6 will leave the space in position 6 to put t_8 in χ_4. A similar situation happens with t_{11} position from χ_1 to the new position in χ_4. Notice that, as denoted in χ_3 in yellow, shifting may introduce new unreplayable transitions: see t_{11}.

We stress that, since this operator tries to generate more executable offspring regardless of the corresponding edit distance, in our experiments we assign small values to λ_2 of Eq. 1, to retain the new generated chromosomes in next generations even if the edit distance has been degraded at the expense of improving replayability.

Mutation operators Mutation applies to a single chromosome, generating a new chromosome as a modification of the initial one. It is viewed as a background operator to maintain genetic diversity in the population. Mutation helps escaping from local minima's trap and maintains diversity in the population. This part presents both generic and specific mutation operators related to the problem considered in this paper.

As with the crossover operators, we start by adapting a generic one. The *Scramble Mutation (SM)* operator simply chooses a segment in the chromosome, and randomly shuffles it. For instance, in Fig. 5 we show how the operator works.

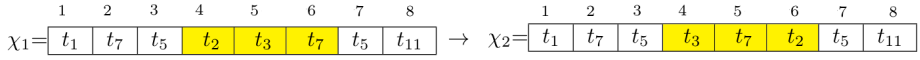

Fig. 5: Scramble mutation operator (SM).

In contrast, the *Mimic Mutation (MM)* operator is a specific operator proposed exclusively for the problem at hand. It tries to mimic the observed trace, by repositioning a transition t as close as possible into the position that $\ell(t)$ was observed in σ. Hence, this operator tends to reduce the edit distance to σ for the mutated chromosome. To implement this idea, we need to reflect on the observation that, due to the Central Limit Theorem, in the limit the position of labels in observed traces follows a Normal distribution (with the corresponding parameters)[6]. Since we are assuming a considerable amount of observed traces in the event log, we can compute these distributions for each one of the distinct labels, using statistical methods like *Regression Splines* [18][7]. This is done only once, before of applying the genetic algorithm.

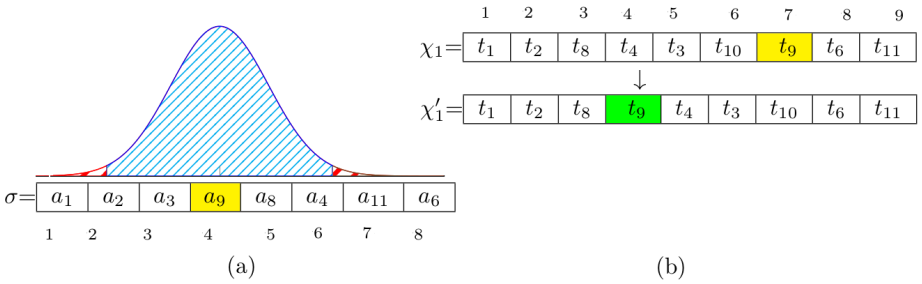

Fig. 6: (a) A probability distribution of locations of a_9 in σ, (b) Mimic mutation.

Fig. 6 shows an example for the observed trace (Fig. 6(a)). In there, the probability of the position of a_9 follows a Normal distribution $N(4, 4)$. It implies that in practice a_9 should be in positions near by position 4.

Once the density function of a certain label is known, a random number following the function will be generated (in the previous example, it should be around position 4).

[6] In case we have a limited number of observations, the real distribution can be estimated by traditional methods, like Kernel Smoothing [18].

[7] In case of duplicate labels in the traces, the normality assumption may be violated and therefore the estimation may be less accurate. In spite of this, the distribution used is only an oracle for generating new locations and does not limit the applicability or our approach.

Once this is obtained, the current position of the event in the chromosome and the new position are swapped. This explains the mutation from χ_1 to χ'_1 in Fig. 6(b).

For a chromosome χ, and its offspring χ', since the goal is to mimic the observed trace, if $f^m(\chi) < f^m(\chi')$ and $f^{ed}(\chi) > f^{ed}(\chi')$ then λ_1 and λ_2 in Eq. 1 are adjusted so that sometimes χ' survives in the next iteration. In other words replayability is overshadowed for this operator. Playing with these parameters would decrease the risk of getting stuck in a local optimum.

Launch Mutation (LM). Up to this point, none of the mutation operators above try to improve the replayable property of chromosomes. The intuitive idea is, for a given chromosome with an unreplayable position i, the transition in i will be relocated forward (i.e., in a position $j > i$) by this operator. The rationale behind this policy comes from the idea that in some situations, by delaying the firing of a transition to a future Petri net marking, enough tokens will be placed by the transitions occupying positions between i and $j-1$. Since the overall goal of this operator is to improve replayability, we set $\lambda_1 < \lambda_2$ so that replayability has more importance to decide survival for the next iteration.

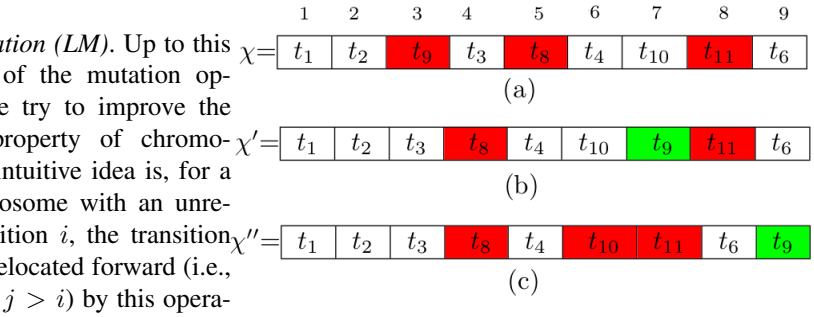

Fig. 7: (a) χ, (b) χ' after LM on χ to position 7, (c) χ'' after LM on χ to position 9.

To give a concrete example consider the chromosome in Fig. 7 (a), and the model M_2. Unreplayable transitions are highlighted (positions 3, 5 and 8). Assume that t_9 is selected to be mutated with is operator. Fig. 7 (b) shows one possible launch mutation from position 3 to position 7. One can see that transition t_9 can now be replayable, as highlighted in green. Unfortunately, this operator can sometimes introduce new unreplayable transitions, as demonstrated in Fig. 7 (c) for t_{10}.

5 General Framework for Obtaining Multiple Alignments

Given a process model represented as a WF-net, N, and a trace σ, the schema of the proposed framework is depicted in Fig. 8. Explanations of each part are provided below:

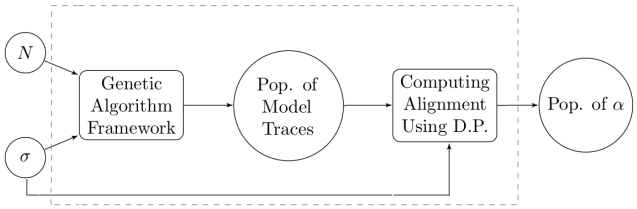

Fig. 8: Overall description of the general approach to compute alignments.

– *Genetic Algorithm Framework*: In the initial stage, the genetic approach described in the previous section is performed. Once finished it generates a final population of model traces. Among them, we choose those chromosomes χ having both $f^m(\chi) = 0$ (so, replayable), and minimal $f^{ed}(\chi)$.
– *Computing Alignment Using Dynamic Programming*: This part concerns the computation of alignments between the chromosomes of final population and σ. The adopted method in this section is a dynamic programming approach inspired from aligning two sequence of genes [19], [20]. The alignments computed are called *best alignments*, which are not necessarily optimal: this is due to the lack of guarantees that the model explanations provided in the previous stage correspond to the optimal model explanation for σ.

5.1 Computing an Alignment using Dynamic Programming

To compute an alignment between a chromosome like χ and observed trace σ, the technique presented in this paper is inspired from [20]. This technique was already applied in [3] for the same task, so we informally describe it here. Consider an oversimplified example, $\chi = t_3t_{11}t_{17}$ with $\ell(\chi) = a_3a_{11}a_{17}$ and $\sigma = a_3a_{11}$. To obtain an alignment α between these two sequences, a two-dimensional table is created, where the first row and first column are filled with the observed trace and chromosome, respectively, as depicted in Fig. 9 (a). The second row and second column are initialized with numbers starting from 0,-1,-2,..., they are depicted in yellow color. The task then is to fill the remaining cells with the recurrence Eq. (2), in which δ represents the *gap penalty*[8] and $s(t_i, a_j)$ represents both the match and mismatch cost between two elements t_i and a_j which are modeled and observed trace elements, respectively.

$$SIM(t_i, a_j) = MAX \begin{cases} SIM(t_{i-1}, a_{j-1}) + s(t_i, a_j) \\ SIM(t_{i-1}, a_j) - \delta \\ SIM(t_i, a_{j-1}) - \delta \end{cases} \qquad s(t_i, a_j) = \begin{cases} \beta & \text{If} \quad \ell(t_i) = a_j \\ -\beta & \text{If} \quad \ell(t_i) \neq a_j \end{cases} \tag{2}$$

$SIM(t_i, a_j)$ represents the similarity score between t_i and a_j.

		a_3	a_{11}
	0	-1	-2
a_3	-1	0	-1
a_{11}	-2	-1	0
a_{17}	-3	-2	-1

(a)

$\alpha =$

a_3	a_{11}	$\perp$
a_3	a_{11}	a_{17}

(b)

Fig. 9: (a) Computing alignment using dynamic programming (b) Obtained alignment.

After filling the matrix, to compute the alignment we start from the bottom right entry, and compare the value with three possible sources, i.e., top, left and diagonal to identify from which one of them it came from. If it was fed by a diagonal entry, it represents a synchronous move between corresponding elements and if it was fed by a top or left entries then it represents an asynchronous move or a gap. For the mentioned chromosome and observed trace the computed alignment is shown in Fig. 9 (b).

[8] The gap penalty represents asynchronous move in our setting.

6 Experiments

The approach of this paper has been incorporated into the tool ALI [21]. This section evaluates the method proposed over the following perspectives:

- What is the sensitivity of the approach on the number of evolutionary iterations ?
- How does the approach compares to [1,10] for the memory and execution time ?
- How does the approach compares to [1,10] for the quality and quantity of alignments obtained ?
- What is the impact on the fitness calculation ?

The tool has been evaluated over different family of examples from artificial to realistic, containing transitions with duplicate labels and from well-structured to completely Spaghetti[9]. The number of transitions varies between models, i.e., minimum 15 and maximum 429. The specification of benchmark datasets can be found in [3], [14], [12]. We also included a real-life benchmark from [9], where a model was discovered using the Inductive Miner by sampling 10000 observed traces of a *Road Traffic* process dataset[10], and using the rest of the log for alignment computation. Also, to examine the proposed approach in dealing with models that contain duplicate labels, a set of models consisting of duplicate transitions, with the corresponding logs obtained by injecting different degree of noise (25%, 35%, 50% and 75%), were generated by PLG2 [22][11]. The details of the models with duplicate labels are presented in Table 1. The results on these benchmarks are compared with the state-of-the-art technique for computing one optimal alignments [1], since the version for computing all optimal alignments ran out of memory for all models considered in this paper. We also compare ALI with a recent technique to compute all-optimal alignments [10].

Table 1: Models with duplicate labels.

Model	$\|P\|$	$\|T\|$	$\|Arc\|$	Cases	Fitting	$\|\sigma\|_{avg}$	Duplicate Transitions
ML_1	27	35	74	500	No	28	2
ML_2	165	177	404	500	No	87	12
ML_3	45	45	106	500	No	26	2
ML_4	36	33	80	500	No	28	6
ML_5	159	172	390	500	No	42	14

Configuration of the Genetic Algorithm. For each observed trace a population of 700 chromosomes were generated and the quality and quantity of them were compared with state of the art approach at iterations 10,20,30 and 100. In the implementation, the application crossover had a high probability, whilst mutation operators were applied with a low probability. As we commented in the previous section, best alignments in

[9] The experiments have been done on Intel Core i7-2.20GHz computer with 8GB of RAM.

[10] https://data.4tu.nl/repository/uuid:270fd440-1057-4fb9-89a9-b699b47990f5

[11] At the time of generating models for the experiments, PLG2 in fact was unable to produce models containing duplicate labels from scratch, therefore the generated models and logs were modified in order to have transitions with duplicate labels.

our setting correspond to replayable chromosomes with minimal edit distance to the observed trace among the final population.

Execution Times. Fig. 10 shows violin plots of execution time (in seconds) for each model per iteration given an observed trace. Obviously the required execution time varies from different observed traces and this is why the corresponding distributions via violin plots are presented. One can see that for big models with large traces (*prDm6, prEm6, prFm6*), models with many deviations in observed traces (*prCm6*) and models with many duplicate transitions (ML_5), the corresponding distributions are wider due to more operations made by the proposed operators at any iteration. An important point should be done: although the computation time per trace (corresponding to multiplying the execution time per iteration shown in the plot by the number of iterations performed) is significantly higher with respect to [1], our evaluation is done with a simple, unoptimized implementation of the technique of this paper.

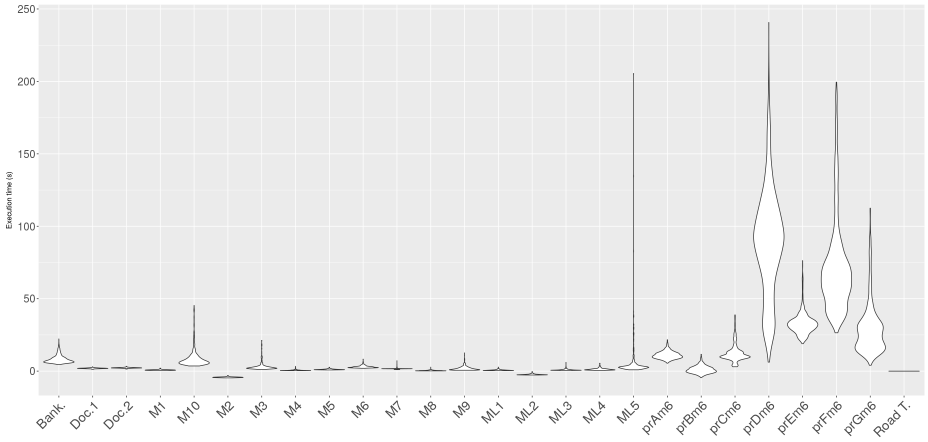

Fig. 10: Distribution of execution time for an observed trace per iteration.

Fitness Comparison. Table 2 represents the *mean square error* (MSE) of fitness values based on metric presented in [1], [12], between the best alignments provided by our technique and the approach in [1] as optimal solutions, respectively. One can see that the quality of alignments is improved from 10 iterations to 100 iterations for all models. For models $prAm6$, $prBm6$ and $prEm6$ optimal alignments were found (for some of them, at iterations 2 and 3 respectively). Comparisons were done only for those benchmark datasets whenever the approach in [1] could provide solutions. Overall, one can see that the approach of this paper is very close to the optimal solutions computed by [1], in spite of several factors like the size of the model and observed traces, presence of loops, silent transitions and duplicate labels in the model.

Table 2: MSE comparison of fitness values of chromosomes at different iterations and A^* as the optimal one

Model	It.(10)	It.(20)	It.(30)	It.(100)
prAm6	0.0106	0.0055	0.0035	0
prBm6	0.0009	0	0	0
prCm6	0.0776	0.0044	0.0031	0.0012
prEm6	0.0436	0.0353	0	0
M_1	0.0571	0.0089	0.0084	0.0046
M_2	0.0475	0.0116	0.0094	0.0070
M_3	0.0351	0.0341	0.0327	0.0219
M_4	0.1980	0.0512	0.0508	0.0398
M_5	0.0958	0.0289	0.0226	0.0155
M_8	0.0836	0.0384	0.0379	0.0357
M_9	0.0725	0.0220	0.0214	0.0203
ML_1	0.0775	0.0400	0.0146	0.0091
ML_3	0.1864	0.0991	0.0159	0.0142
ML_4	0.3113	0.1740	0.0330	0.0251
Bank.	0.0013	0.0008	0.0005	0.0002
Doc.1	0.2912	0.1971	0.0702	0.0698
Doc.2	0.2581	0.1701	0.0579	0.0570
Road T.	0.0036	0.0022	0.0018	0.0014

Table 3: Number of different alignments for the best solution found in average

Model	It.(10)	It.(20)	It.(30)	It.(100)	ATM. $A*$
prAm6	1.11	1.21	1.25	1.45	NA
prBm6	1.00	1.15	1.15	1.34	NA
prCm6	1.16	1.52	1.79	3.46	NA
prDm6	1.11	1.33	1.57	1.71	NA
prEm6	1.16	1.43	1.52	1.56	NA
prFm6	1.03	1.17	1.36	1.46	NA
prGm6	1.08	1.30	1.49	1.78	NA
M_1	1.94	2.91	3.32	4.32	62.12(92%)
M_2	2.98	4.97	5.89	7.13	320.1(53%)
M_3	1.30	1.98	2.41	2.79	NA
M_4	1.00	1.01	1.21	1.62	7.40 (39%)
M_5	1.77	2.62	3.44	6.01	114.78(10%)
M_6	1.68	2.34	2.87	4.37	NA
M_7	2.05	3.38	4.27	7.12	NA
M_8	1.36	1.55	1.71	2.14	7.81(69%)
M_9	1.01	1.02	1.31	1.46	8.32(30%)
M_{10}	1.02	2.56	3.54	5.23	NA
ML_1	1.04	1.15	1.27	1.29	11.78(34%)
ML_2	1.85	2.49	3.38	4.85	NA
ML_3	1.75	2.71	3.25	3.60	7.94(21%)
ML_4	1.72	2.98	3.39	5.80	NA
ML_5	1.05	1.84	2.42	3.42	NA
Bank.	1.08	1.44	1.83	2.66	NA
Doc1.	1.00	1.08	2.21	2.70	I/O Error
Doc2.	1.00	1.01	1.68	1.98	I/O Error
Road T.	1.00	1.01	1.03	1.21	1.41 (100%)

Quantity of Best Alignments. Table 3 shows the average number of best alignment obtained per each observed trace and each model at different number of iterations. In the last column, we report this number for [10]: NA denotes that the tool was unable to provide the result due to memory problems. When it can, we also provide in parenthesis the percentage of the log traces where [10] can find solutions; for instance, for M_4, only 39% of the traces have a solution. One sees that these average numbers are improved from 10 to 100 iterations and this improvements are usually more tangible in models containing loops, i.e., M_1, M_2, M_3, M_7. The approach from [10] usually obtains more alignments than our method, but that only holds for small or medium instances.

Also, Fig. 11 and 12 show for each model, the violin plots or distribution of number of best alignment for 30 and 100 iterations, respectively (the corresponding average values are shown in the fourth and fifth column of Table 3, respectively). When focusing in the experiment for 100 iterations (Fig. 12), It can be seen from the plot that, for some models like M_1, M_2 M_5, M_7 and ML_2, the number of distinct best solutions are close to 30 for some cases and for *Documentflow2* the best solutions are unique.

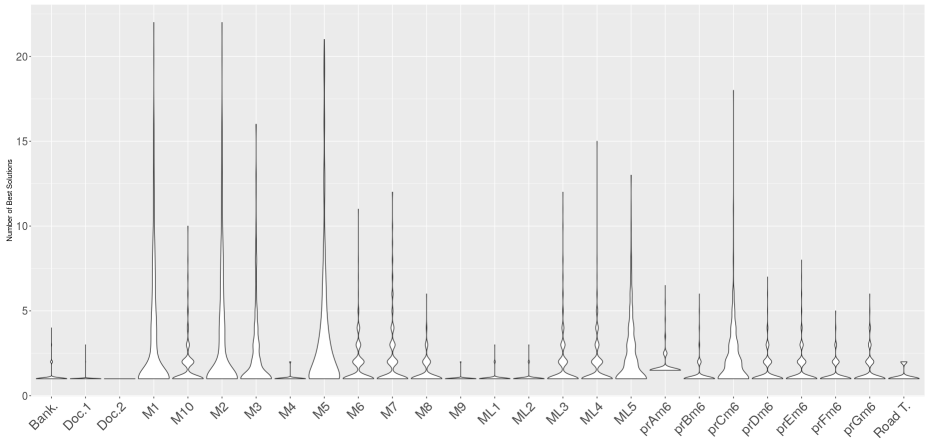

Fig. 11: Distribution of number of best solutions for 30 iterations.

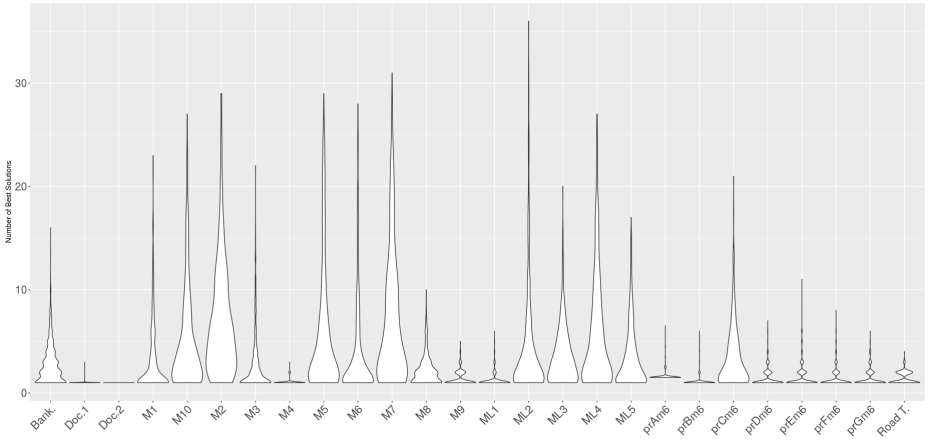

Fig. 12: Distribution of number of best solutions for 100 iterations.

Memory Consumption. The memory footprint of the proposed technique and the approaches in [1] and [10], for all the benchmarks of this paper are represented in Fig.13, using black, gray and brown colors, respectively. It must be stressed that the comparison reported in Fig.13 provides just an indication of the huge difference in terms of memory footprint between the technique of this paper and the other techniques: for [1], experiments were only done for computing one optimal alignment inevitably, since the implementation for all optimal alignments ran out of memory. In contrast, in Fig. 13 we provide the results of our technique and the technique in [10] to compute multiple alignments.

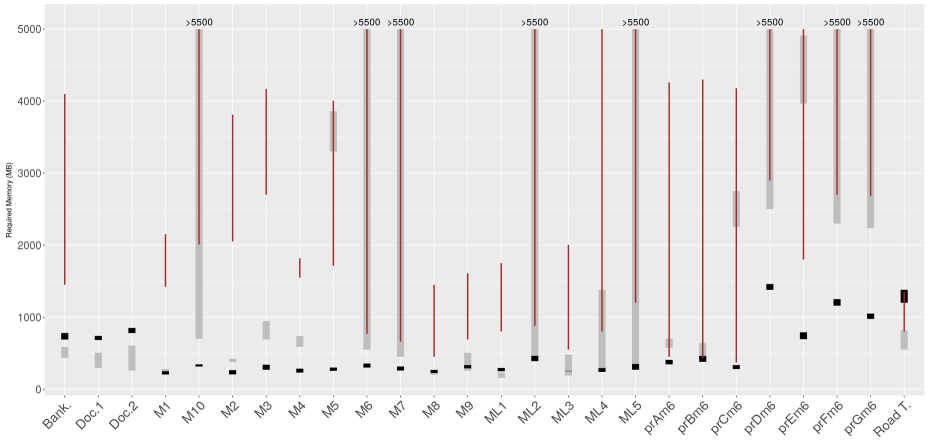

Fig. 13: Mem. footprint of our approach (black), [1] (gray), and [10] (brown, thin line).

One can see that the proposed technique requires considerable less memory than the other two techniques. Obviously for small and medium models, the memory footprints are similar. For large models the tendency is inversed: as an example, for *prDm6* the proposed method required around 1.5GB whereas [1] and [10] need more than 5.5GB Notice that the memory footprint of the proposed approach for computing best alignments is bounded through iterations, and is not sensitive to size of the model and length of the observed trace. Also, the required memory for the proposed approach is not sensitive to the labels of transitions i.e., silent or duplicate labels, see $ML_1, \ldots, ML_5$. The other two approaches are more sensitive to the aforementioned factors.

7 Conclusion and Future Work

Conformance checking is a crucial aid for diagnosing deviations between modeled and observed behavior. The best way to detect such deviations is the use of aligments, as they open the door for further analyses. This paper presents a novel approach to compute several approximation of an optimal alignment. It is based on an evolutionary algorithm, where the memory footprint is guaranteed to be bounded. Tailored genetic operators have been proposed, which help guiding the algorithm through the search space of solutions, and speed up convergence accordingly. The experiments performed on the tool developed witness the quality of obtained alignments, deriving solutions that are close to optimal ones, and which can be improved iteratively. Moreover, the quantity of alignments improves considerably as more genetic iterations are performed. In spite of not having theoretical guarantees on optimality or replayability, the results show that in practice it is always the case that replayable, quasi-optimal or optimal solutions are produced. For the future work there are many possibilities to explore, like introducing more efficient operators, or devising mechanism to alleviate the mentioned drawbacks.

Acknowledgments We would like to thank B. van Dongen for interesting discussions. This work has been supported by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R.

References

1. A. Adriansyah, Aligning observed and modeled behavior, Ph.D. thesis, Technische Universiteit Eindhoven (2014).
2. A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. van Dongen, W. M. P. van der Aalst, Measuring precision of modeled behavior, Inf. Syst. E-Business Man. 13 (1) (2015) 37–67.
3. F. Taymouri, J. Carmona, Model and event log reductions to boost the computation of alignments, in: Data-Driven Process Discovery and Analysis - 6th IFIP WG 2.6 International Symposium, SIMPDA, Graz, Austria, 2016, pp. 1–21.
4. M. Koorneef, A. Solti, H. Leopold, H. A. Reijers, Automatic root cause identification using most probable alignments, in: BPM 2017 Workshops, Barcelona, Spain, 2017, pp. 204–215.
5. W. M. P. van der Aalst, A. K. A. de Medeiros, A. J. M. M. Weijters, Genetic process mining, in: Applications and Theory of Petri Nets (ICATPN), Miami, USA, 2005, pp. 48–69.
6. J. C. A. M. Buijs, B. F. van Dongen, W. M. P. van der Aalst, A genetic algorithm for discovering process trees, in: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, 2012, 2012, pp. 1–8.
7. B. Vázquez-Barreiros, M. Mucientes, M. Lama, Prodigen: Mining complete, precise and minimal structure process models with a genetic algorithm, Inf. Sci. 294 (2015) 315–333.
8. F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. P. van der Aalst, Balanced multi-perspective checking of process conformance, Computing 98 (4) (2016) 407–437.
9. M. de Leoni, A. Marrella, Aligning real process executions and prescriptive process models through automated planning, Expert Syst. Appl. 82 (2017) 162–183.
10. D. Reißner, R. Conforti, M. Dumas, M. L. Rosa, A. Armas-Cervantes, Scalable conformance checking of business processes, in: OTM CoopIS, , Rhodes, Greece, 2017, pp. 607–627.
11. S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Scalable process discovery and conformance checking, Software and System Modeling 17 (2) (2018) 599–631.
12. F. Taymouri, J. Carmona, A recursive paradigm for aligning observed behavior of large structured process models, in: 14th International Conference of Business Process Management (BPM), Rio de Janeiro, Brazil, 2016.
13. B. F. van Dongen, J. Carmona, T. Chatain, F. Taymouri, Aligning modeled and observed behavior: A compromise between computation complexity and quality, in: Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings, 2017, pp. 94–109.
14. J. Munoz-Gama, J. Carmona, W. M. P. Van Der Aalst, Single-entry single-exit decomposed conformance checking, Inf. Syst. 46 (2014) 102–122.
15. W. M. P. van der Aalst, Decomposing Petri nets for process mining: A generic approach, Distributed and Parallel Databases 31 (4) (2013) 471–507.
16. T. Murata, Petri nets: Properties, analysis and applications, Proc. of the IEEE 77 (4) (1989) 541–574.
17. A. Piszcz, T. Soule, Genetic programming: Optimal population sizes for varying complexity problems, in: Conference on Genetic and Evolutionary Computation, 2006, pp. 953–954.
18. D. Ruppert, M. P. Wand, R. J. Carroll, Scatterplot Smoothing, Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press, 2003, p. 57–90.
19. R. Neapolitan, Foundations Of Algorithms, 5th Edition, Jones and Bartlett Publishers, Inc., USA, 2014, pp. 138–146.
20. S. B. Needleman, C. D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, J. of Molecular Biology 48 (3) (1970) 443 – 453.
21. F. Taymouri, ALI: Alignment for Large Instances (2017).
URL `https://www.cs.upc.edu/~taymouri/tool.html`
22. A. Burattin, PLG2: multiperspective process randomization with online and offline simulations, in: BPM Demo Track 2016, 2016, pp. 1–6.