Metaheuristic Optimization for Automated Business Process Discovery

Adriano Augusto[1,2], Marlon Dumas[2], and Marcello La Rosa[1]

[1] University of Melbourne, Australia
{a.augusto, marcello.larosa}@unimelb.edu.au
[2] University of Tartu, Estonia
marlon.dumas@ut.ee

Abstract. The problem of automated discovery of process models from event logs has been intensely investigated in the past two decades, leading to a range of approaches that strike various trade-offs between accuracy, model complexity, and execution time. A few studies have suggested that the accuracy of automated process discovery approaches can be enhanced by using metaheuristic optimization. However, these studies have remained at the level of proposals without validation on real-life logs or they have only considered one metaheuristics in isolation. In this setting, this paper studies the following question: To what extent can the accuracy of automated process discovery approaches be improved by applying different optimization metaheuristics? To address this question, the paper proposes an approach to enhance automated process discovery approaches with metaheuristic optimization. The approach is instantiated to define an extension of a state-of-the-art automated process discovery approach, namely Split Miner. The paper compares the accuracy gains yielded by four optimization metaheuristics relative to each other and relative to state-of-the-art baselines, on a benchmark comprising 20 real-life logs. The results show that metaheuristic optimization improves the accuracy of Split Miner in a majority of cases, at the cost of execution times in the order of minutes, versus seconds for the base algorithm.

1 Introduction

The problem of automatically discovering business process models from event logs has been intensely studied in the past two decades. Research in this field has led to a wide range of Automated Process Discovery Approaches (APDAs) that strike various trade-offs between accuracy, model complexity, and execution time [7].

A few studies have suggested that the accuracy of APDAs can be enhanced by applying optimization metaheuristics. Early studies in this direction considered population-based metaheuristics, chiefly genetic algorithms [14, 10]. These heuristics are computationally heavy, requiring execution times in the order of hours to converge when applied to real-life logs [7]. Another work has considered single-solution-based metaheuristics such as simulated annealing [21, 15], which are less computationally demanding. However, these latter studies have remained at the level of proposals without validation on real-life logs and comparison of trade-offs between alternative heuristics.

In this setting, this paper studies the following question: *to what extent can the accuracy of APDAs be improved by applying single-solution-based metaheuristics?* To address this question, we propose a flexible approach to enhance APDAs by applying different optimization metaheuristics. The core idea is to perturb the intermediate representation of event logs used by the majority of the available APDAs, namely the Directly-follows Graph (DFG). The paper specifically considers perturbations that add or remove edges with the aim of improving fitness or precision, and in a way that allows the underlying APDA to discover a process model from the perturbed DFG. An instantiation of our approach is defined for a state-of-the-art APDA, namely Split Miner.

Using a benchmark of 20 real-life logs, the paper compares the accuracy gains yielded by four optimization metaheuristics relative to each other and relative to state-of-the-art APDAs. The experimental evaluation also considers the impact of metaheuristic optimization on model complexity measures as well as on execution times.

The next section gives an overview of APDAs and optimization metaheuristics. Section 3 presents the proposed metaheristic optimization approach. Section 4 reports on the empirical evaluation and Section 5 draws conclusions and future work directions.

2 Background and Related Work

In this section, we give an overview of existing approaches to automated process discovery, followed by an introduction to optimization metaheuristics in general, and their application to automated process discovery in particular.

2.1 Automated Process Discovery

The execution of business processes is often recorded in the form of *event logs*. An event log is a collection of event records produced by individual instances (i.e. cases) of the process. The goal of automated process discovery is to generate a process model that captures the behavior observed in or implied by an *event log*. To assess the goodness of a discovered process model, four quality dimensions are used [23]: fitness, precision, generalization, and complexity. *Fitness* (a.k.a. recall) measures the amount of behavior observed in the log that is captured by the model. A perfectly fitting process model is one that recognizes every trace in the log. *Precision* measures the amount of behavior captured in the process model that is observed in the log. A perfectly precise model is one that recognizes only traces that are observed in the log. *Generalization* measures to what extent the process model captures behavior that, despite not being observed in the log, is implied by it. Finally, *complexity* measures the understandability of a process model, and it is typically measured via size and structural measures. In this paper, we focus on fitness, precision, and F-score (the harmonic mean of fitness and precision).

A recent comparison of state-of-the-art APDAs [7] showed that an approach capable of consistently discovering models with the best fitness-precision trade-off is currently missing. The same study showed, however, that we can obtain consistently good trade-offs by hyperparameter-optimizing some of the existing APDAs based on DFGs – Inductive Miner [19], Structured Heuristics Miner [6], Fodina [24], and Split Miner [8]. These algorithms have a hyperparameter to tune the amount of filtering applied when constructing the DFG. Optimizing this and other hyperparameters via greedy search [7], local search strategies [11], or sensitivity analysis techniques [20], can greatly improve the accuracy of the discovered process models. Accordingly, in the evaluation reported later we use a hyperparameter-optimized version of Split Miner as one of the baselines.

2.2 Optimization Metaheuristics

The term *optimization metaheuristics* refers to a parameterized algorithm, which can be instantiated to address a wide range of optimization problems. Metaheuristics are usually classified into two broad categories [9]: i) *single-solution-based metaheuristics*, or S-metaheuristics, which explore the solution space one solution at a time starting from a single initial solution of the problem; and ii) *population-based metaheuristics*, or P-metaheuristics, which explored a population of solutions generated by mutating, combining, and/or improving previously computed solutions. Single-solution based metaheuristics tend to converge faster towards an optimal solution (either local or global) than P-metaheuristics, since the latter by dealing with a set of solutions require more time to assess and improve the quality of each single solution. P-metaheuristics are more computationally heavy but they are more likely to escape local optima. An exhaustive discussion on all available metaheuristics is beyond the scope of this paper, in the following we focus only on the S-metaheuristics that we explore in our approach: iterated local search, tabu search, and simulated annealing.

Iterated Local Search [22] starts from a (random) solution and explores the neighbouring solutions (i.e. solutions obtained by applying a perturbation) in search of a better one. When a better solution cannot be found, it perturbs the current solution and starts again. The perturbation is meant to avoid local optimal solutions. *Tabu Search* [16] is a memory-driven local search. Its initialization includes a (random) solution and three memories. The short-term memory keeps track of recent solutions and prohibits to revisit them. The intermediate-term memory contains criteria driving the search towards the best solutions. The long-term memory contains characteristics that have often been found in many visited solutions, to avoid revisiting similar solutions. Using these memories, the neighbourhood of the initial solution is explored and a new solution is selected accordingly. *Simulated Annealing* [17] is based on the concepts of Temperature (T, a parameter choose arbitrarily) and Energy (E, the objective function to minimize). At each iteration the algorithm explores (some of) the neighbouring solutions and compares their energies with the one of the current solution. This latter is updated if the energy of a neighbour is lower, or with a probability that is function of T and the energies of the current and candidate solutions (usually $e^{-\frac{|E_1-E_2|}{T}}$). The temperature drops over time, thus reducing the chance of updating the current solution with a higher-energy one. The algorithm ends when a criterion is met (e.g. energy below a threshold or $T = 0$).

2.3 Optimization Metaheuristics in Automated Process Discovery

Metaheuristic optimization has been considered in a few previous studies on automated process discovery. An early attempt to apply P-metaheuristics for automated process discovery was the Genetic Miner proposed by De Medeiros [14], subsequently overtaken by the Evolutionary Tree Miner [10]. In this latter approach, an evolutionary algorithm is used on top of process trees (i.e. a block-structured representation of a process model). Other applications of P-metaheuristics to automated process discovery are based on the imperialist competitive algorithms [3] and particle swam optimization [12]. The main limitation P-metaheuristics in this context is that they are computationally heavy due to the cost of constructing a solution (i.e. process model) and evaluating its accuracy. This leads to execution times in the order of hours to converge to a solution, which on the end is comparable to that obtained by state-of-the-art algorithms that do not rely on optimization metaheuristics [7].

Only a handful of studies have considered the use of S-metaheuristics in this setting, specifically simulated annealing [21, 15]. However, these latter proposals are preliminary and have not been compared against state-of-the-art approaches on real-life logs.

3 Approach

This section outlines our approach for extending APDAs by means of S-metaheuristics (cf. Section 2). First, we give an overview of the approach and its components. Next, we discuss the adaptation of the metaheuristics to the problem of process discovery. Finally, we describe an instantiation of the approach for Split Miner.

3.1 Preliminaries

An APDA takes as input an event log. This log is transformed into an intermediate representation from which a model is derived. In many APDAs, the intermediate representation is the DFG, which is represented as a numerical matrix as formalized below.

Definition 1. ***[Event Log]*** *Given a set of activities $\mathscr{A}$, an* event log *$\mathscr{L}$ is a multiset of traces where a trace $t \in \mathscr{L}$ is a sequence of activities $t = \langle a_1, a_2, \ldots, a_n \rangle$, with $a_i \in \mathscr{A}, 1 \leq i \leq n$.*

Definition 2. ***[Directly-Follows Graph (DFG)]*** *Given an event log $\mathscr{L}$, its* Directly-Follows Graph (DFG) *is a directed graph $\mathscr{G} = (N, E)$, where: N is the set of nodes, $N = \{a \in \mathscr{A} \mid \exists t \in \mathscr{L} \wedge a \in t\}$; and E is the set of edges $E = \{(x,y) \in N \times N \mid \exists t = \langle a_1, a_2, \ldots, a_n \rangle, t \in \mathscr{L} \wedge a_i = x \wedge a_{i+1} = y [1 \leq i \leq n-1]\}$.*

Definition 3. ***[DFG-Matrix]*** *Given a DFG $\mathscr{G} = (N, E)$ and a function $\theta : N \rightarrow [1, |N|]$,[1] the DFG-Matrix is a squared matrix $X_{\mathscr{G}} \in [0,1] \cap \mathbb{N}^{|N| \times |N|}$, where each cell $x_{i,j} = 1 \iff \exists (a_1, a_2) \in E \mid \theta(a_1) = i \wedge \theta(a_2) = j$, otherwise $x_{i,j} = 0$.*

An APDA is said to be *DFG-based* if it first generates the DFG of the event log, then applies an algorithm to filter (e.g. removing activities) from the DFG, and finally converts the processed DFG into a process model.[2] Examples of DFG-based APDAs are Inductive Miner [18], Heuristics Miner[25, 6], Fodina[24], and Split Miner[8].

Different DFG-based APDAs may extract different DFGs from the same log. Also, a DFG-based APDA may discover different DFGs from the same log depending on its hyperparameter settings (e.g. the filtering threshold). The algorithm(s) used by a DFG-based APDA to discover the DFG from the event log and convert it into a process model may greatly affect the accuracy of an APDA. Accordingly, our approach focuses on optimizing the discovery of the DFG rather than its conversion into a process model.

3.2 Approach Overview

As shown in Figure 1, our approach takes three inputs (in addition to the log): i) the optimization metaheuristics; ii) the objective function to be optimized (e.g. F-score); iii) and the DFG-based APDA to be used for discovering a process model.

[1] θ maps each node of the DFG to a natural number.

[2] Herein, when using the term DFG, we refer to the processed DFG (after filtering).

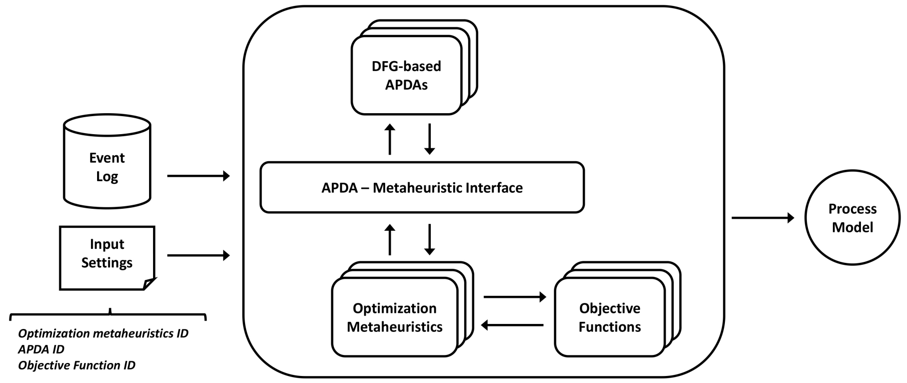

Fig. 1: Overview of our approach.

Algorithm 1 describes how our approach operates. First, the input event log is given to the APDA, which returns the discovered DFG and its corresponding process model (lines 1 and 2). This DFG becomes the current DFG and process model becomes the best process model (so far). The model's objective function score (e.g. F-score) is stored as the current score and the best score (lines 3 and 4). The current DFG is then given as input to function *GenerateNeighbours*, which applies changes to the current DFG to generate a set of neighbouring DFGs (line 6). These latter are given as input to the APDA, which returns the corresponding into process models. The process models are assessed by the objective function evaluators (line 9 to 13). When the metaheuristic receives the results from the evaluators (along with the current DFG and score), it chooses the new current DFG and updates the current score (lines 14 and 15). If the new current score is higher than the best score(line 16), it updates the best process model and the best score (lines 17 and 18). After the update, a new iteration starts, unless a termination criterion is met (e.g. a timeout, a maximum number of iterations, or a minimum threshold for the objective function). In this latter case, it outputs the best model found, i.e. the process model scoring the highest value for the objective function.

3.3 Adaptation of the Optimization Metaheuristics

To adapt Iterative Local Search (ILS), Tabu Search (TABU), and Simulated Annealing (SIMA) to the problem of automated process discovery, we need to define the following three concepts: i) the problem solution space; ii) a solution neighbourhood; iii) the objective function. These design choices determine how each of the metaheuristics navigates the solution space and escapes local minima, i.e. how to design the Algorithm 1 functions: *GenerateNeighbours* and *UpdateDFG*, resp. lines 6 and 14.

Solution Space. Being our goal the optimization of APDAs, we are forced to choose a solution space that fits well our context regardless the selected APDA. If we assume that the APDA is DFG-based (that is the case for the majority of the available APDAs), we can define the solution space as the set of all the DFG discoverable from the event log. Indeed, any DFG-based APDA can generate deterministically a process model from a DFG.

Solution Neighbourhood. Having defined the solution space as the set of all the DFG discoverable from the event log, we can refer to any element of this solution space

Algorithm 1: Optimization Approach

```
   input   : Event Log $\mathscr{L}$, Metaheuristic $\omega$, Objective Function $\mathscr{F}$, DFG-based APDA $\alpha$
1  CurrentDFG $\mathscr{G}_c \leftarrow$ DiscoverDFG($\alpha$, $\mathscr{L}$);
2  BestModel $\hat{m} \leftarrow$ ConvertDFGtoProcessModel($\alpha$, $\mathscr{G}_c$);
3  CurrentScore $s_c \leftarrow$ AssessQuality($\mathscr{F}$, $\hat{m}$);
4  BestScore $\hat{s} \leftarrow s_c$;
5  while CheckTerminationCriteria() = FALSE do
6      Set $V \leftarrow$ GenerateNeighbours($\mathscr{G}_c$);
7      Map $S \leftarrow \varnothing$;
8      Map $M \leftarrow \varnothing$;
9      for $\mathscr{G} \in V$ do
10         ProcessModel $m \leftarrow$ ConvertDFGtoProcessModel($\alpha$, $\mathscr{G}$);
11         Score $s \leftarrow$ AssessQuality($\mathscr{F}$, $m$);
12         add ($\mathscr{G}$, $s$) to $S$;
13         add ($\mathscr{G}$, $m$) to $M$;
14     $\mathscr{G}_c \leftarrow$ UpdateDFG($\omega$, $S$, $\mathscr{G}_c$, $s_c$);
15     $s_c \leftarrow$ GetMapElement($S$, $\mathscr{G}_c$);
16     if $\hat{s} < s_c$ then
17         $\hat{s} \leftarrow s_c$;
18         $\hat{m} \leftarrow$ GetMapElement($M$, $\mathscr{G}_c$);
19 return $\hat{m}$;
```

as a DFG-Matrix. Given a DFG-Matrix, we define its neighbourhood as the set of all the matrices having one different cell value (i.e. DFGs having one more/less edge). In the following, every time we refer to DFG we assume it is represented as a DFG-Matrix.

Objective Function. It is possible to define the objective function as any function assessing one of the four quality dimensions for discovered process models (introduced in Section 2). However, being interested in optimizing the APDAs to discover the most accurate process model, in the remaining of this paper, we refer to the objective function as the F-score of fitness and precision: $\frac{2 \cdot fit \cdot prec}{fit + prec}$. Nonetheless, we remark that our approach can operate also with objective functions that take into account multiple quality dimensions striving for a trade-off, e.g. F-score and model complexity.

Having defined the solution space, a solution neighbourhood, and the objective function, we can turn our attention on how ILS, TABU, and SIMA navigate the solution space. ILS, TABU, and SIMA share similar traits in solving an optimization problem, especially when it comes to the navigation of the solution space. Given a problem and its solution space, any of these three S-metaheuristics starts from a (random) solution, discovers one or more neighbouring solutions, and assesses them with the objective function to find a solution better than the current. If a better solution is found, it is chosen as the new current solution and the metaheuristic performs a new neighbourhood exploration. If a better solution is not found, e.g. the current solution is locally optimal, the three metaheuristics follow different approaches to escape the local optimum and continue the solution space exploration. Algorithm 1 orchestrates and facilitates the parts of this procedure shared by the three metaheuristics. However, we must define the functions *GenerateNeighbours* (GNF) and *UpdateDFG* (UDF).

The GNF receives in input a solution of the solution space, i.e. a DFG, and it generates a set of neighbouring DFGs. By definition, GNF is independent from the metaheuristic and it can be as simple or as elaborate as we demand. An example of a simple GNF is a function that randomly selects neighbouring DFGs turning one cell of the input DFG-Matrix to 0 or to 1. Whilst, an example of an elaborate GNF is a function that accurately selects neighbouring DFGs relying on the feedback received from the objective function assessing the input DFG, as we show in Section 3.4.

The UDF is at the core of our optimization, and it represents the metaheuristic itself. It receives in input the neighbouring DFGs, the current DFG, and the current score, and it selects among the neighbouring DFGs the one that should become the new current DFG. At this point, we can differentiate two cases: i) among the input neighbouring DFGs there is at least one having a higher objective function score than the current; ii) none of the input neighbouring DFGs has a higher objective function score than the current. In the first case, UDF always outputs the DFG having the highest score (regardless the selected metaheuristic). In the second case, the current DFG may be a local optimum, and each metaheuristic escapes it with a different strategy.

Iterative Local Search applies the simplest strategy, it perturbs the current DFG. The perturbation is meant to alter the DFG in such a way to escape the local optimum, e.g. randomly adding and removing multiple edges from the current DFG. The perturbed DFG is the output of the UDF.

Tabu Search relies on its three memories to escape a local optimum. The short-term memory (a.k.a. Tabu-list), containing DFG that must not be explored further. The intermediate-term memory, containing DFGs that should lead to better results and, therefore, should be explored in the near future. The long-term memory, containing DFGs (with characteristics) that have been seen multiple times and, therefore, not to explore in the near future. TABU updates the three memories each time the UDF is executed. Given the set of neighbouring DFGs and their respective objective function scores (see Algorithm 1, map S), TABU adds each DFG to a different memory. DFGs worsening the objective function score are added to the Tabu-list. DFGs improving the objective function score, yet less than another neighbouring DFG, are added to the intermediate-term memory. DFGs that do not improve the objective function score are added to the long-term memory. Also, the current DFG is added to the Tabu-list, being it already explored. When TABU does not find a better DFG in the neighbourhood of the current DFG, it returns the latest DFG added to the intermediate-term memory. If the intermediate-term memory is empty, TABU returns the latest DFG added to the long-term memory. If both these memories are empty, TABU requires a new (random) DFG from the APDA, and outputs its DFG.

Simulated Annealing avoids getting stuck in a local optimum by allowing the selection of DFGs worsening the objective function score. In doing so, SIMA explores areas of the solution space that other S-metaheuristics do not. When a better DFG is not found in the neighbourhood of the current DFG, SIMA analyses one neighbouring DFG at a time. If this latter does not worsen the objective function score, SIMA outputs it. Instead, if the neighbouring DFG worsens the objective function score, SIMA outputs it with a probability of $e^{-\frac{|s_n - s_c|}{T}}$, where s_n and s_c are the objective function scores of (respectively) the neighbouring DFG and the current DFG, and the temperature T is an integer that converges to zero as a linear function of the maximum number of iterations. The temperature is fundamental to avoid updating the current DFG with a worse one if there would be no time to recover from the worsening (i.e. too few iterations left for continuing the exploration of the solution space from the worse DFG).

3.4 Instantiation for Split Miner

To assess our approach, we define an instantiation of it for Split Miner – a DFG-based APDA that performs favourably relative to other state-of-the-art APDAs [7]. To instantiate our approach for a concrete APDA, we need to implement an interface that allows the metaheuristics to interact with the APDA (as discussed above). The interface should provide four functions: *DiscoverDFG* and *ConvertDFGtoProcessModel* (see Algorithm 1), the *Restart Function* (RF) for TABU, and the *Perturbation Function* (PF) for ILS. The first two functions come with the DFG-based APDA, in our case Split Miner. Note that, the output of *DiscoverDFG* of Split Miner varies according to the hyperparameters settings.[3] To discover the initial DFG (Algorithm 1, line 1), Split Miner uses its default parameters. We removed the randomness for discovering the initial DFG because most of the times, the DFG discovered by Split Miner with default parameters is already a good solution [8], and starting the solution space exploration from this latter can reduce the total exploration time.

Function RF is very similar to *DiscoverDFG*, since it requires the APDA to output a DFG, the only difference is that RF must output a different DFG every time it is executed. We adapted the *DiscoverDFG* of Split Miner to output the DFG discovered with default parameters the first time it is executed, and for the following executions a DFG discovered with random parameters.

Finally, function PF can be provided either by the APDA (via the interface) or by the metaheuristic. However, PF can be more effective when not generalised by the metaheuristic, allowing the APDA to apply different perturbations to the DFGs, taking into account how the APDA converts the DFG to a process model.

We invoke Split Miner's concurrency oracle to extract the possible parallelism relations in the log using a randomly chosen parallelism threshold. For each new parallel relation discovered (not present in the current solution), two edges are removed from the DFG, whilst, for each deprecated parallel relation, two edges are added to the DFG. Alternatively, it is possible to set PF = RF, so that instead of perturbing the current DFG, a new random DFG is generated. This variant of the ILS is called Repetitive Local Search (RLS). In the evaluation reported below, we use both ILS and its variant RLS.

We use the F-score as the objective function, which is computed from the fitness and precision. Among the existing measures of fitness and precision we selected the Markovian fitness and precision defined in [5] (boolean function variant, order $k = 5$). The rationale for this choice is that these measures of fitness and precision are the fastest to compute among state-of-the-art measures [4, 5]. Furthermore, the Markvovian fitness (precision) provides a feedback that tells us what edges could be added to (removed from) the DFG to improve the fitness (precision). This feedback allows us to design an effective GNF. In the instantiation of our approach for Split Miner, the objective function's output is a data structure composed of: the Markovian fitness and precision of the model, the F-score, and the mismatches between the model and the event log identified during the computation of the Markovian fitness and precision, i.e. the sets of the edges that could be added (removed) to improve the fitness (precision).

Given this objective function's output, our GNF is described in Algorithm 2. The function receives as input the current DFG ($\mathscr{G}_c$), its objective function score (the data

[3] Split Miner has two hyperparameters: the noise filtering threshold, used to drop infrequent edges in the DFG, and the parallelism threshold, used to determine which potential parallel relations between activities are used when discovering the process model from the DFG.

Algorithm 2: Generate Neighbours Function (GNF)

```
input    : CurrentDFG $\mathcal{G}_c$, CurrentMarkovianScore $s_c$, Integer $size_n$
1  if getFitnessScore($s_c$) > getPrecisionScore($s_c$) then
2      Set $E_m \leftarrow$ getEdgesForImprovingPrecision($s_c$);
3  else
4      Set $E_m \leftarrow$ getEdgesForImprovingFitness($s_c$);
5  Set $N \leftarrow \varnothing$;
6  while $E_m \neq \varnothing \wedge |N| \neq size_n$ do
7      Edge $e \leftarrow$ getRandomElement($E_m$);
8      NeighbouringDFG $\mathcal{G}_n \leftarrow$ copyDFG($\mathcal{G}_c$);
9      if getFitnessScore($s_c$) > getPrecisionScore($s_c$) then
10         if canRemoveEdge($\mathcal{G}_n$, $e$) then add $\mathcal{G}_n$ to $N$;
11     else
12         addEdge($\mathcal{G}_n$, $e$);
13         add $\mathcal{G}_n$ to $N$;
14 return $N$;
```

structure s_c), and the number of neighbours to generate ($size_n$). If fitness is greater than precision, we retrieve (from s_c) the set of edges (E_m) that could be removed from $\mathcal{G}_c$ to improve its precision (line 2). Conversely, if precision is greater than fitness, we retrieve (from s_c) the set of edges (E_m) that could be added to $\mathcal{G}_c$ to improve its fitness (line 4). The reasoning behind this design choice is that, given that our objective function is the F-score, it is preferable to increase the lower of the two measures (precision or fitness). i.e. if the fitness is lower, we increase fitness, and conversely if the precision is lower. Once we have E_m, we select randomly one edge from it, we generate a copy of the current DFG ($\mathcal{G}_n$), and we either remove or add the randomly selected edge according to the accuracy measure we want to improve (precision or fitness), see lines 7 to 13. If the removal of an edge generates a disconnected $\mathcal{G}_n$, we do not add this latter to the neighbours set (N), line 10. We keep iterating over E_m until the set is empty (i.e. no mismatching edges are left) or N reaches its max size (i.e. $size_n$). We then return N.

The algorithm ends when the maximum execution time is reached or and the maximum number of iterations it reached (in the experiments below, we set them by default to 5 minutes and 50 iterations).

4 Evaluation

We implemented our approach as a Java command-line application[4] using Split Miner as the underlying automated process discovery approach and Markovian accuracy F-score as the objective function (cf. Section 3.4). We compared the quality of the models discovered by applying each of the optimization metaheuristics mentioned against those discovered by four baselines: i) Split Miner; ii) Split Miner with hyper-parameter optimization; iii) Evolutionary Tree Miner; and iv) Inductive Miner.

The experiments were performed on an Intel Core i5-6200U@2.30GHz with 16GB RAM running Windows 10 Pro (64-bit) and JVM 8 with 14GB RAM (10GB Stack and 4GB Heap). The approach's implementation, the batch tests, the results, and all the

[4] Available under the label "Metaheuristically Optimized Split Miner" at `http://apromore.org/platform/tools`.

models discovered during the experiments are available for reproducibility purposes at `https://doi.org/10.6084/m9.figshare.7824671.v1`.

4.1 Dataset

For our evaluation we used the dataset of the benchmark of automated process discovery approaches in [7], which to the best of our knowledge is the most recent benchmark on this topic. This dataset includes twelve public logs and eight private logs. The public logs originate from the 4TU Centre for Research Data, and include the *BPI Challenge* (BPIC) logs (2012-17),[5] the *Road Traffic Fines Management Process* (RTFMP) log[6] and the *SEPSIS* log[7]. These logs record executions of business processes from a variety of domains, e.g. healthcare, finance, government and IT service management. In seven logs (BPIC14, the BPIC15 collection and BPIC17), the filtering technique in [13] was applied to remove infrequent behavior; this step was necessary to maintain consistency with the benchmark dataset. The eight proprietary logs are sourced from several companies in the education, insurance, IT service management and IP management domains.

Table 1 reports the characteristics of the logs. As seen in the table, the dataset comprises simple logs (e.g. BPIC13$_{cp}$) and very complex ones (e.g. SEPSIS, PRT2) in terms of percentage of distinct traces, and both small logs (e.g. BPIC13$_{cp}$ and SEPSIS) and large ones (e.g. BPIC17 and PRT9) in terms of total number of events.

Log		BPIC12	BPIC13$_{cp}$	BPIC13$_{inc}$	BPIC14$_{f}$	BPIC15$_{1f}$	BPIC15$_{2f}$	BPIC15$_{3f}$	BPIC15$_{4f}$	BPIC15$_{5f}$
Total Traces		13,087	1,487	7,554	41,353	902	681	1,369	860	975
Dist. Traces(%)		33.4	12.3	20	36.1	32.7	61.7	60.3	52.4	45.7
Total Events		262,200	6,660	65,533	369,485	21,656	24,678	43,786	29,403	30,030
Dist. Events		36	7	13	9	70	82	62	65	74
Tr. length	(min)	3	1	1	3	5	4	4	5	4
	(avg)	20	4	9	9	24	36	32	34	31
	(max)	175	35	123	167	50	63	54	54	61

Log		BPIC17$_{f}$	RTFMP	SEPSIS	PRT1	PRT2	PRT3	PRT4	PRT6	PRT7	PRT9	PRT10
Total Traces		21,861	150,370	1,050	12,720	1,182	1,600	20,000	744	2,000	787,657	43,514
Dist. Traces(%)		40.1	0.2	80.6	8.1	97.5	19.9	29.7	22.4	6.4	0.01	0.01
Total Events		714,198	561,470	15,214	75,353	46,282	13,720	166,282	6,011	16,353	1,808,706	78,864
Dist. Events		41	11	16	9	9	15	11	9	13	8	19
Tr. length	(min)	11	2	3	2	12	6	6	7	8	1	1
	(avg)	33	4	14	5	39	8	8	8	8	2	1
	(max)	113	2	185	64	276	9	36	21	11	58	15

Table 1: Descriptive statistics of the real-life logs (public and proprietary).

4.2 Experimental setup

For each log in our dataset, we discovered eight process models: four using the metaheuristics presented in Section 3 (RLS, ILS, TABU and SIMA) and four baselines. The

[5] `https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`
`https://doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07`
`https://doi.org/10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35`
`https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1`
`https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b`

[6] `https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5`

[7] `https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460`

baselines include the Evolutionary Tree Miner (ETM) [10], Inductive Miner infrequent variant (IM) [18], and Split Miner (SM) [8], all with default parameters settings. ETM was allowed to run with a 4-hours timeout. All comparisons with ETM are meant as comparisons of accuracy (fitness, precision, F-score) and not as execution time comparisons, as the computational heaviness of ETM has already been shown in previous work [10, 7]. The fourth baseline (HPO_{sm}) is a hyperparameter-optimized version of the SM algorithm, where we varied the two hyperparameters of SM (noise filtering and parallelism filtering threshold) across their full range with steps of 0.01 (from 0.01 to 1.00), and retaining the model with the highest Markovian F-score.

ETM, IM and SM were selected as baselines because they had the highest accuracy in a recent benchmark comparison of APDAs [7]. We also selected HPO_{sm} to compare the effects of optimization metaheuristics versus hyperparameter optimization.

For each of the discovered models we measured accuracy, complexity and discovery time. For the accuracy, we adopted two different sets of measures: one based on *alignments*, computing fitness and precision with the approaches proposed in [2, 1] (alignment-based accuracy); and one based on *Markovian abstractions*, computing fitness and precision with the approaches proposed in [4, 5] (Markovian accuracy). For assessing the complexity of the models we relied on *size* (number of nodes of the model), Control-Flow Complexity (CFC) (the amount of branching caused by split gateways in the model), and Structuredness (the percentage of nodes located directly inside a well-structured single-entry single-exit fragment).

4.3 Results

Tables 2 and 3 show the results of our comparative evaluation. Each row reports the quality of each discovered process model in terms of accuracy (both alignment-based and Markovian) and complexity, as well as the discovery time.

Due to space, we held out from the tables four logs: $\text{BPIC13}_{\text{cp}}$, $\text{BPIC13}_{\text{inc}}$, BPIC17, and PRT9. For these logs, none of the metaheuristics could improve the accuracy of the model already discovered by SM. This is due to the high fitness score achieved by SM in these logs. By design, our metaheuristics try to improve the precision by removing edges, but in these four cases, no edge could be removed without compromising the structure of the model (i.e. the model would become disconnected).

For the remaining 16 logs, all the metaheuristics improved consistently the Markovian F-score w.r.t. SM. Also, all the metaheuristics performed better than HPO_{sm}, except in two cases (BPIC12 and PRT1). Overall, the most effective optimization metaheuristic was ILS, which delivered the highest Markovian F-score nine times out of 16, followed by SIMA (eight times), RLS and TABU (six times each). Compared to ETM, the four metaheuristics achieved better Markovian F-scores in 15 out of 16 cases, and better alignment F-scores 14 times out of 16, while compared to IM, all the optimization metaheuristics achieved better Markovian F-scores in 15 cases out of 16, and better alignment F-scores across the whole dataset.

Despite the fact that the objective function of the metaheuristics was the Markovian F-score, all four metaheuristics optimized in half of the cases the alignment-based F-score. This is due to the fact that any improvement on the Markovian fitness translates into an improvement on the alignment-based fitness, though the same does not hold for the precision. This result highlights the (partial) correlation between the alignment-based and the Markovian accuracies, already reported in previous studies [4,

Event	Discovery	Align. Acc.			Markov. Acc. ($k = 5$)			Complexity			Exec.
Log	Approach	Fitness	Prec.	F-score	Fitness	Prec.	F-score	Size	CFC	Struct.	Time(s)
BPIC12	ETM	0.440	**0.820**	0.573	0.536	**0.462**	**0.496**	67	**16**	**1.00**	14,400
	IM	**0.990**	0.502	0.666	0.280	0.002	0.005	59	37	**1.00**	6.6
	SM	0.963	0.520	0.675	**0.818**	0.139	0.238	51	41	0.69	**3.2**
	HPO$_{sm}$	0.781	0.796	**0.788**	0.575	0.277	0.374	**40**	17	0.58	4295.8
	RLS$_{sm}$	0.921	0.671	0.776	0.586	0.247	0.348	49	31	0.90	159.3
	ILS$_{sm}$	0.921	0.671	0.776	0.586	0.247	0.348	49	31	0.90	159.4
	TABU$_{sm}$	0.921	0.671	0.776	0.586	0.247	0.348	49	31	0.90	140.7
	SIMA$_{sm}$	0.921	0.671	0.776	0.586	0.247	0.348	49	31	0.90	151.1
BPIC14$_{f}$	ETM	0.610	**1.000**	0.758	0.009	0.313	0.017	23	**9**	**1.00**	14,400
	IM	0.890	0.646	0.749	0.501	0.346	0.409	31	18	**1.00**	3.4
	SM	0.772	0.881	0.823	0.150	**1.000**	0.262	**20**	14	**1.00**	**0.8**
	HPO$_{sm}$	0.852	0.857	0.855	0.449	**1.000**	0.619	22	16	0.59	575.8
	RLS$_{sm}$	**1.000**	0.771	**0.871**	**1.000**	0.985	**0.992**	28	34	0.54	139.0
	ILS$_{sm}$	**1.000**	0.771	**0.871**	**1.000**	0.985	**0.992**	28	34	0.54	151.3
	TABU$_{sm}$	0.955	0.775	0.855	0.856	0.999	0.922	26	31	0.69	154.7
	SIMA$_{sm}$	**1.000**	0.771	**0.871**	**1.000**	0.985	**0.992**	28	34	0.54	140.3
BPIC15$_{1f}$	ETM	0.560	**0.940**	0.702	0.235	0.284	0.257	**67**	**19**	**1.00**	14,400
	IM	**0.970**	0.566	0.715	0.665	0.001	0.002	164	108	**1.00**	1.1
	SM	0.899	0.871	0.885	0.701	0.726	0.713	111	45	0.51	**0.7**
	HPO$_{sm}$	0.962	0.833	**0.893**	**0.804**	0.670	0.731	117	55	0.45	1242.3
	RLS$_{sm}$	0.925	0.839	0.880	0.774	0.803	0.788	124	63	0.39	163.6
	ILS$_{sm}$	0.925	0.839	0.880	0.774	0.803	0.788	124	63	0.39	166.8
	TABU$_{sm}$	0.948	0.843	0.892	0.774	0.805	**0.789**	125	64	0.33	187.2
	SIMA$_{sm}$	0.920	0.839	0.878	0.772	**0.807**	**0.789**	125	63	0.43	160.4
BPIC15$_{2f}$	ETM	0.620	**0.910**	0.738	0.301	0.389	0.339	**95**	**32**	**1.00**	14,400
	IM	**0.948**	0.556	0.701	0.523	0.002	0.004	193	123	**1.00**	1.7
	SM	0.783	0.877	0.828	0.514	0.596	0.552	129	49	0.36	**0.6**
	HPO$_{sm}$	0.808	0.851	0.829	0.561	0.582	0.572	133	56	0.30	1398.9
	RLS$_{sm}$	0.870	0.797	**0.832**	0.667	0.670	0.668	156	86	0.20	158.3
	ILS$_{sm}$	0.869	0.795	0.830	0.663	**0.680**	**0.671**	157	86	0.20	157.6
	TABU$_{sm}$	0.870	0.794	0.830	0.665	0.667	0.666	150	83	0.23	176.8
	SIMA$_{sm}$	0.871	0.775	0.820	**0.677**	0.662	0.669	159	93	0.26	167.4
BPIC15$_{3f}$	ETM	0.680	0.880	0.767	0.238	0.172	0.199	**84**	**29**	**1.00**	14,400
	IM	**0.950**	0.554	0.700	0.480	0.002	0.003	159	108	**1.00**	1.3
	SM	0.774	**0.925**	0.843	0.436	0.764	0.555	96	35	0.49	**0.5**
	HPO$_{sm}$	0.783	0.910	0.842	0.477	0.691	0.564	99	39	0.56	9230.4
	RLS$_{sm}$	0.812	0.903	**0.855**	0.504	**0.775**	0.611	110	53	0.35	151.5
	ILS$_{sm}$	0.833	0.868	0.850	0.533	**0.775**	**0.631**	120	66	0.23	153.8
	TABU$_{sm}$	0.832	0.852	0.842	0.558	0.690	0.617	121	64	0.23	173.4
	SIMA$_{sm}$	0.827	0.839	0.833	**0.565**	0.694	0.623	123	71	0.18	159.4
BPIC15$_{4f}$	ETM	0.650	**0.930**	0.765	0.351	0.292	0.319	**83**	**28**	**1.00**	14,400
	IM	**0.955**	0.585	0.726	0.567	0.001	0.002	162	111	**1.00**	2.4
	SM	0.762	0.886	0.820	0.516	0.615	0.562	101	37	0.27	**0.5**
	HPO$_{sm}$	0.785	0.860	0.821	0.558	0.578	0.568	103	40	0.27	736.4
	RLS$_{sm}$	0.825	0.854	**0.839**	0.634	**0.672**	0.652	114	57	0.21	146.9
	ILS$_{sm}$	0.853	0.807	0.829	**0.649**	0.657	**0.653**	117	64	0.27	147.8
	TABU$_{sm}$	0.811	0.794	0.803	0.642	0.661	0.651	115	61	0.24	161.7
	SIMA$_{sm}$	0.847	0.812	0.829	0.624	0.649	0.636	117	61	0.18	148.2
BPIC15$_{5f}$	ETM	0.570	0.940	0.710	0.365	0.504	0.423	**88**	**18**	**1.00**	14,400
	IM	**0.937**	0.179	0.301	0.242	0.000	0.000	134	95	**1.00**	2.5
	SM	0.806	0.915	0.857	0.555	0.598	0.576	110	38	0.34	**0.6**
	HPO$_{sm}$	0.789	**0.941**	**0.858**	0.529	0.655	0.585	102	30	0.33	972.3
	RLS$_{sm}$	0.868	0.813	0.840	0.737	0.731	0.734	137	78	0.14	159.3
	ILS$_{sm}$	0.868	0.813	0.840	0.737	0.731	0.734	137	78	0.14	153.8
	TABU$_{sm}$	0.885	0.818	0.850	**0.739**	**0.746**	**0.743**	137	79	0.14	173.3
	SIMA$_{sm}$	0.867	0.811	0.838	0.734	0.727	0.731	137	78	0.16	154.3
RTFMP	ETM	0.990	0.920	0.954	0.981	0.010	0.019	57	32	**1.00**	14,400
	IM	0.980	0.700	0.817	0.934	0.046	0.087	34	20	**1.00**	13.9
	SM	**0.996**	0.958	0.977	**0.959**	0.311	0.470	22	17	0.46	**2.9**
	HPO$_{sm}$	0.887	**1.000**	0.940	0.685	0.696	0.690	**20**	**9**	0.35	2452.7
	RLS$_{sm}$	0.988	**1.000**	**0.994**	0.899	0.794	0.843	22	14	0.46	142.8
	ILS$_{sm}$	0.988	**1.000**	**0.994**	0.899	0.794	0.843	22	14	0.46	143.8
	TABU$_{sm}$	0.988	**1.000**	**0.994**	0.899	0.794	0.843	22	14	0.46	114.8
	SIMA$_{sm}$	0.986	**1.000**	0.993	0.875	**0.893**	**0.884**	23	15	0.39	131.0
SEPSIS	ETM	0.830	0.660	**0.735**	0.696	0.096	0.169	108	101	**1.00**	14,400
	IM	**0.991**	0.445	0.614	0.741	0.012	0.024	50	32	**1.00**	1.3
	SM	0.764	**0.706**	0.734	0.349	**0.484**	0.406	**32**	**23**	0.94	**0.4**
	HPO$_{sm}$	0.925	0.588	0.719	**0.755**	0.293	0.423	33	34	0.39	28,846
	RLS$_{sm}$	0.839	0.630	0.720	0.508	0.430	**0.466**	35	29	0.77	145.4
	ILS$_{sm}$	0.812	0.625	0.706	0.455	0.436	0.445	35	28	0.86	157.1
	TABU$_{sm}$	0.839	0.630	0.720	0.508	0.430	**0.466**	35	29	0.77	137.0
	SIMA$_{sm}$	0.806	0.613	0.696	0.477	0.445	0.460	35	30	0.77	137.2

Table 2: Comparative evaluation results for the public logs.

Event	**Discovery**	**Align. Acc.**			**Markov. Acc. ($k = 5$)**			**Complexity**			**Exec.**
Log	**Method**	**Fitness**	**Prec.**	**F-score**	**Fitness**	**Prec.**	**F-score**	**Size**	**CFC**	**Struct.**	**Time(s)**
	ETM	0.990	0.811	0.892	0.977	0.213	0.350	23	12	**1.00**	14,400
	IM	0.902	0.673	0.771	0.232	0.051	0.084	20	**9**	**1.00**	3.8
	SM	0.976	**0.974**	**0.975**	0.730	0.669	0.698	20	14	**1.00**	**0.4**
	HPO$_{sm}$	**0.999**	0.948	0.972	**0.989**	0.620	0.762	**19**	14	0.53	298.3
PRT1	RLS$_{sm}$	0.976	**0.974**	**0.975**	0.730	0.669	0.698	20	14	**1.00**	155.3
	ILS$_{sm}$	0.976	**0.974**	**0.975**	0.730	0.669	0.698	20	14	**1.00**	153.2
	TABU$_{sm}$	0.976	**0.974**	**0.975**	0.730	0.669	0.698	20	14	**1.00**	10.3
	SIMA$_{sm}$	0.983	0.964	0.974	0.814	**0.722**	**0.765**	20	15	**1.00**	132.6
	ETM	0.572	**0.943**	0.712	0.105	0.788	0.186	86	21	**1.00**	14,400
	IM	ex	ex	ex	0.329	0.179	0.232	45	33	**1.00**	2.3
	SM	0.795	0.581	0.671	0.457	**0.913**	0.609	29	23	**1.00**	**0.3**
	HPO$_{sm}$	0.826	0.675	**0.743**	0.501	0.830	0.625	**21**	**13**	0.67	406.4
PRT2	RLS$_{sm}$	0.886	0.421	0.571	0.629	0.751	0.685	29	34	**1.00**	141.4
	ILS$_{sm}$	**0.890**	0.405	0.557	**0.645**	0.736	**0.688**	29	35	**1.00**	172.3
	TABU$_{sm}$	0.866	0.425	0.570	0.600	0.782	0.679	29	33	**1.00**	143.1
	SIMA$_{sm}$	0.886	0.424	0.574	0.629	0.751	0.685	29	34	**1.00**	139.7
	ETM	**0.979**	0.858	0.915	0.858	0.313	0.459	51	37	**1.00**	14,400
	IM	0.975	0.680	0.801	**0.874**	0.481	**0.621**	37	20	**1.00**	0.9
	SM	0.882	0.887	0.885	0.381	0.189	0.252	31	23	0.58	**0.4**
	HPO$_{sm}$	0.890	0.899	0.895	0.461	0.518	0.488	**26**	**14**	0.81	290.2
PRT3	RLS$_{sm}$	0.945	**0.902**	**0.923**	0.591	0.517	0.551	31	23	0.55	138.4
	ILS$_{sm}$	0.945	**0.902**	**0.923**	0.591	0.517	0.551	31	23	0.55	144.2
	TABU$_{sm}$	0.944	**0.902**	0.922	0.589	**0.519**	0.552	30	20	0.60	134.7
	SIMA$_{sm}$	0.945	**0.902**	**0.923**	0.591	0.517	0.551	31	23	0.55	133.7
	ETM	0.844	0.851	0.847	0.629	0.950	0.757	64	28	**1.00**	14,400
	IM	0.927	0.753	0.831	0.615	0.952	0.747	27	**13**	**1.00**	1.1
	SM	0.884	**1.000**	0.938	0.483	**1.000**	0.652	**25**	15	0.96	**0.5**
	HPO$_{sm}$	0.973	0.930	**0.951**	0.929	0.989	0.958	26	24	0.31	867.5
PRT4	RLS$_{sm}$	**0.997**	0.903	0.948	**0.993**	0.990	**0.992**	26	28	0.92	140.1
	ILS$_{sm}$	**0.997**	0.903	0.948	**0.993**	0.990	**0.992**	26	28	0.92	152.3
	TABU$_{sm}$	0.955	0.914	0.934	0.883	0.988	0.932	26	26	0.77	138.6
	SIMA$_{sm}$	**0.997**	0.903	0.948	**0.993**	0.990	**0.992**	26	28	0.92	136.9
	ETM	0.980	0.796	0.878	0.890	0.611	0.725	41	16	**1.00**	14,400
	IM	**0.989**	0.822	0.898	**0.946**	0.444	0.604	23	10	**1.00**	2.9
	SM	0.937	**1.000**	**0.967**	0.542	**1.000**	0.703	**15**	**4**	**1.00**	**0.3**
	HPO$_{sm}$	0.937	**1.000**	**0.967**	0.542	**1.000**	0.703	**15**	**4**	**1.00**	105.1
PRT6	RLS$_{sm}$	0.984	0.928	0.955	0.840	0.818	**0.829**	22	14	0.41	141.1
	ILS$_{sm}$	0.984	0.928	0.955	0.840	0.818	**0.829**	22	14	0.41	144.2
	TABU$_{sm}$	0.984	0.928	0.955	0.840	0.818	**0.829**	22	14	0.41	124.9
	SIMA$_{sm}$	0.984	0.928	0.955	0.840	0.818	**0.829**	22	14	0.41	131.2
	ETM	0.900	0.810	0.853	0.969	0.217	0.355	60	29	**1.00**	14,400
	IM	**1.000**	0.726	0.841	**1.000**	0.543	0.704	29	13	**1.00**	1.3
	SM	0.914	0.999	0.954	0.650	**1.000**	0.788	29	10	0.48	**0.6**
	HPO$_{sm}$	0.944	**1.000**	0.971	0.772	**1.000**	0.871	**22**	**9**	0.64	173.1
PRT7	RLS$_{sm}$	0.993	**1.000**	**0.996**	0.933	**1.000**	**0.965**	23	11	0.78	139.2
	ILS$_{sm}$	0.993	**1.000**	**0.996**	0.933	**1.000**	**0.965**	23	11	0.78	142.9
	TABU$_{sm}$	0.993	**1.000**	**0.996**	0.933	**1.000**	**0.965**	23	11	0.78	134.0
	SIMA$_{sm}$	0.993	**1.000**	**0.996**	0.933	**1.000**	**0.965**	23	11	0.78	131.9
	ETM	**1.000**	0.627	0.771	0.748	0.001	0.003	61	45	**1.00**	14,400
	IM	0.964	0.790	0.868	**0.945**	0.001	0.001	41	29	**1.00**	4.6
	SM	0.970	0.943	**0.956**	0.905	0.206	0.335	45	47	0.84	**0.5**
	HPO$_{sm}$	0.936	0.943	0.939	0.810	0.243	0.374	**30**	**22**	0.73	1214.3
PRT10	RLS$_{sm}$	0.917	**0.989**	0.952	0.741	**0.305**	**0.432**	44	41	0.86	153.0
	ILS$_{sm}$	0.917	**0.989**	0.952	0.741	**0.305**	**0.432**	44	41	0.86	155.4
	TABU$_{sm}$	0.917	**0.989**	0.952	0.741	**0.305**	**0.432**	44	41	0.86	117.6
	SIMA$_{sm}$	0.917	**0.989**	0.952	0.741	**0.305**	**0.432**	44	41	0.86	136.7

Table 3: Comparative evaluation results for the proprietary logs.

5]. Analysing the complexity of the models, we note that most of the times (nine cases

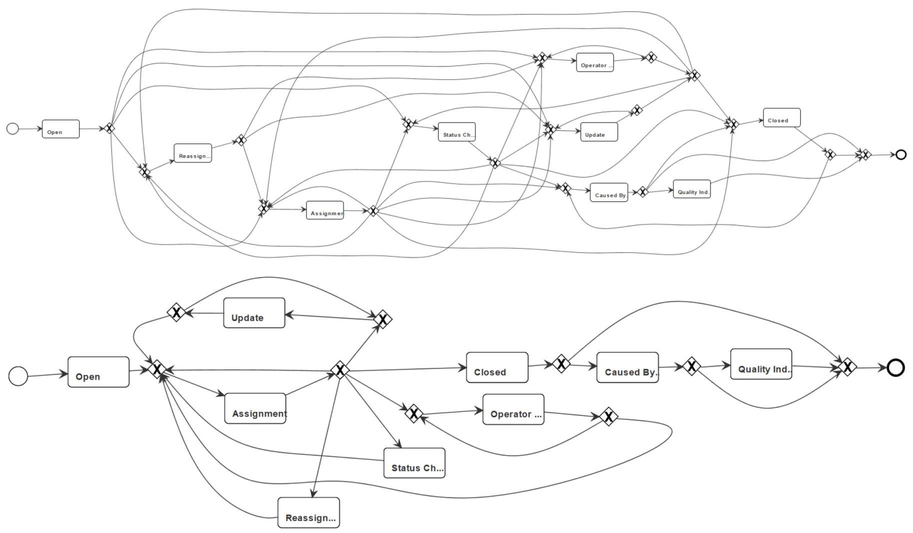

Fig. 2: BPIC14$_f$ models discovered with SIMA (above) and SM (below).

out of 16) the F-score improvement achieved by the metaheuristics comes at the cost of size and CFC. This is expected, since SM tends to discover models with higher precision than fitness [7]. What happens is that to improve the F-score, new behavior is added to the model in the form of new edges (note that new nodes are never added). Adding new edges leads to new gateways and consequently to increasing size and CFC. On the other hand, when the precision is lower than fitness and the metaheuristic aims to increase the value of this measure to improve the overall F-score, the result is the opposite: the model complexity reduces as edges are removed. This is the case of the RTFMP and PRT10 logs. As an example of the two possible scenarios, Figure 2 shows the models discovered by SIMA and SM from the BPIC14$_f$ log (where the model discovered by SIMA is more complex than that obtained with SM), while Figure 3 shows the models discovered by SIMA and SM from the RTFMP log (where the model discovered by SIMA is simpler). Comparing the results obtained by the metaheuristics with HPO$_{sm}$, we can see that our approach allows us to discover models that cannot be discovered simply by tuning the hyperparameters of SM. This relates to the solution space exploration. Indeed, while HPO$_{sm}$ can only explore a limited number of solutions (DFGs), i.e. those that can be generated by the underlying APDA, SM in this case, by varying its hyperparameters, the metaheuristics go beyond the solution space of HPO$_{sm}$ by exploring new DFGs in a pseudo-random manner.

In terms of execution times, the four metaheuristics perform similarly, having an average discovery time close to 150 seconds. While this is considerably higher than the execution time of SM ($\sim$ 1 second on average), it is much lower than HPO$_{sm}$ and ETM, while consistently achieving higher accuracy.

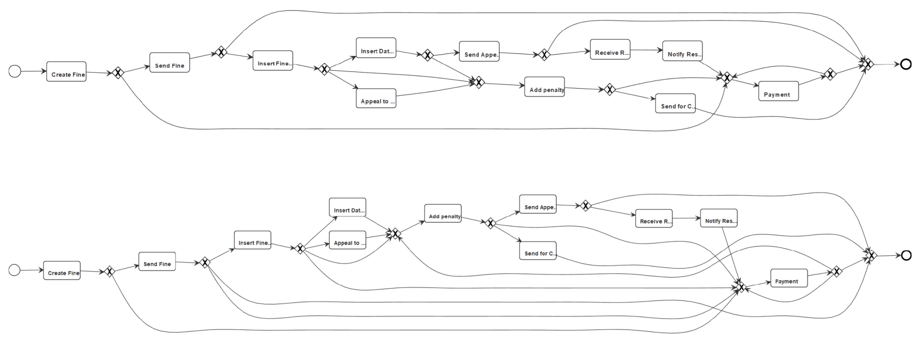

Fig. 3: BPIC14$_f$ models discovered with SIMA (above) and SM (below).

5 Conclusion

This paper showed that the use of S-metaheuristics is a promising approach to enhance the accuracy of DFG-based APDAs. The outlined approach takes advantage of the DFG's simplicity to define efficient perturbation functions that improve fitness or precision while preserving structural properties required to ensure model correctness.

The evaluation showed that the metaheuristic extensions of Split Miner achieve considerably higher accuracy for a clear majority of logs in the benchmark, particularly when using fine-grained measures of fitness and precision based on Markovian abstractions, but also when using measures based on trace alignment. These accuracy gains come at the expense of slightly higher model size and structural complexity. The results also show that the choice of S-metaheuristics (among the four considered in this paper) does not visibly affect accuracy nor model complexity. The metaheuristic extensions do come with a penalty in terms of execution times. The execution times of the metaheuristic-enhanced versions of Split Miner are $\sim$ 2-3 minutes versus $\sim$ 1 second for the base miner. Interestingly, the S-metaheuristics improve accuracy even with respect a hyperparameter-optimized version of Split Miner, while achieving considerably lower execution times. This means that the metaheuristic extensions of Split Miner explore solutions that cannot be constructed by varying the filtering and parallelism thresholds.

The study reported here is limited to one APDA (Split Miner). A possible direction for future work is to define and evaluate extensions of this approach for other DFG-based APDAs such as Fodina and Inductive Miner. Also, the approach focuses on improving F-score, while it could be applied to optimize other objective functions (e.g. combinations of F-score and model complexity) or to perform Pareto-front optimization, i.e. finding Pareto-optimal solutions with respect to multiple quality measures. Finally, this study only considered four S-metaheuristics. There is room for investigating other metaheuristics or other variants of simulated annealing, e.g. using different cooling schedules. Finally, the paper only considered one baseline approach that uses a P-metaheuristics (ETM). A more detailed comparison of tradeoffs between S-metaheuristics and P-metaheuristics in this setting is another avenue for future work.

Acknowledgements. We thank Raffaele Conforti for his input to an earlier version of this paper. This research is partly funded by the Australian Research Council (DP180102839) and the Estonian Research Council (IUT20-55).

References

1. A. Adriansyah, J. Munoz-Gama, J. Carmona, B. van Dongen, and W. van der Aalst. Measuring precision of modeled behavior. *ISeB*, 13(1), 2015.
2. A. Adriansyah, B. van Dongen, and W. van der Aalst. Conformance checking using cost-based fitness analysis. In *EDOC*. IEEE, 2011.
3. S. Alizadeh and A. Norani. Icma: a new efficient algorithm for process model discovery. *Applied Intelligence*, 48(11), 2018.
4. A. Augusto, A. Armas-Cervantes, R. Conforti, M. Dumas, M. La Rosa, and D. Reissner. Abstract-and-compare: A family of scalable precision measures for automated process discovery. In *BPM*. Springer, 2018.
5. A. Augusto, A. Armas Cervantes, R. Conforti, M. Dumas, M. La Rosa, and D. Reissner. Measuring fitness and precision of automatically discovered process models: A principled and scalable approach. Technical report, University of Melbourne, 2019.
6. A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and G. Bruno. Automated Discovery of Structured Process Models From Event Logs: The Discover-and-Structure Approach. *DKE*, 2017.
7. A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F.M. Maggi, A. Marrella, M. Mecella, and A. Soo. Automated discovery of process models from event logs: Review and benchmark. *IEEE TKDE*, 31(4), 2019.
8. A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and A. Polyvyanyy. Split miner: automated discovery of accurate and simple business process models from event logs. *KAIS*, 2018.
9. I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237, 2013.
10. J. Buijs, B. van Dongen, and W. van der Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In *CoopIS*. Springer, 2012.
11. A. Burattin and A. Sperduti. Automatic determination of parameters' values for heuristics miner++. In *IEEE Congress on Evolutionary Computation*, 2010.
12. V. R. Chifu, C. B. Pop, I. Salomie, I. Balla, and R. Paven. Hybrid particle swarm optimization method for process mining. In *ICCP*. IEEE, 2012.
13. R. Conforti, M. La Rosa, and A. ter Hofstede. Filtering out infrequent behavior from business process event logs. *IEEE TKDE*, 29(2), 2017.
14. A. K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, 2006.
15. D. Gao and Q. Liu. An improved simulated annealing algorithm for process mining. In *CSCWD*. IEEE, 2009.
16. F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5), 1986.
17. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598), 1983.
18. S. Leemans, D. Fahland, and W. van der Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. In *BPM Workshops*. Springer, 2014.
19. S. Leemans, D. Fahland, and W. van der Aalst. Scalable process discovery and conformance checking. *Software & Systems Modeling*, 2016.
20. J. Ribeiro and J. Carmona Vargas. A method for assessing parameter impact on control-flow discovery algorithms. In *Algorithms & Theories for the Analysis of Event Data*, 2015.
21. W. Song, S. Liu, and Q. Liu. Business process mining based on simulated annealing. In *ICYCS*. IEEE, 2008.
22. T. Stützle. *Local search algorithms for combinatorial problems*. PhD thesis, Darmstadt University of Technology, 1998.
23. W. van der Aalst. *Process Mining - Data Science in Action*. Springer, 2016.
24. S. vanden Broucke and J. De Weerdt. Fodina: a robust and flexible heuristic process discovery technique. *DSS*, 2017.
25. A. Weijters and J. Ribeiro. Flexible heuristics miner (FHM). In *CIDM*. IEEE, 2011.