Optimization Framework for DFG-based Automated Process Discovery Approaches

Adriano Augusto[a,b], Marlon Dumas[b], Marcello La Rosa[a], Sander J.J. Leemans[c], Seppe K.L.M. vanden Broucke[d,e]

[a] *University of Melbourne, Australia*
[b] *University of Tartu, Estonia*
[c] *Queensland University of Technology, Australia*
[d] *UGent, Belgium*
[e] *KU Leuven, Belgium*

Abstract

The problem of automatically discovering business process models from event logs has been intensely investigated in the past two decades, leading to a wide 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 means of metaheuristic optimization techniques. However, these studies have remained at the level of proposals without validation on real-life datasets or they have only considered one metaheuristic in isolation. This article presents a metaheuristic optimization framework for automated process discovery. The key idea of the framework is to construct a Directly-Follows Graph (DFG) from the event log, to perturb this DFG so as to generate new candidate solutions, and to apply a DFG-based automated process discovery approach in order to derive a process model from each DFG. The framework can be instantiated by linking it to an automated process discovery approach, an optimization metaheuristic, and the quality measure to be optimized (e.g. fitness, precision, F-score). The article considers several instantiations of the framework corresponding to four optimization metaheuristics, three automated process discovery approaches (Inductive Miner – directly follows, Fodina, and Split Miner), and one accuracy measure (Markovian F-score). These framework instances are compared using a set of 20 real-life event logs. The evaluation shows that metaheuristic optimization consistently yields visible improvements in F-score for all the three automated process discovery approaches, at the cost of execution times in the order of minutes, versus seconds for the baseline approaches.

Keywords: Automated process discovery, metaheuristic optimization, process mining

Email address: a.augusto@unimelb.edu.au (Adriano Augusto)

1. Introduction

Modern information systems such as Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP) systems record transactions corresponding to activities executed within the business processes that these systems support. For example, a CRM system typically records transactions corresponding to the creation of a customer lead, a request for quote, and various other activities related to customer leads, quotes, and purchase orders. These transactional records can be extracted via SQL queries or via dedicated Application Programming Interfaces (APIs) and used to analyze the execution of the business processes supported by the CRM system, such as the lead-to-quote or the quote-to-order process.

Process mining [33] is a family of techniques to analyze transactional records associated to a given business process, also known as an event log, in order to extract insights about the performance of the process. Among other things, process mining techniques allow us to discover a process model from an event log, an operation known as *automated process discovery*. Automatically discovered process models allow analysts to understand how the process is executed in reality and to uncover unexpected behaviour. When enhanced with performance information (e.g. average activity durations or waiting times), such models are also used for performance analysis, e.g. to detect bottlenecks.

The problem of automated process discovery has been intensely studied in the past two decades [8]. Research in this field has led to a wide range of Automated Process Discovery Approaches (APDAs) that strike various trade-offs between accuracy[1], model complexity, and execution time. Existing approaches in this field rely on parameters (with certain default values) to strike this tradeoff. Analysts need to fine-tune these parameters to find a model with the best possible trade-off between different model quality metrics. This article addresses the question of how to automate the fine-tuning of automated process discovery techniques.

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 (P-metaheuristics), chiefly genetic algorithms [16, 13]. However, these heuristics are computationally heavy, requiring execution times in the order of hours to converge when applied to real-life logs [8]. Such high executions times make these techniques inapplicable in the context of exploratory and interactive process discovery, where an analyst may need to discover process models corresponding to several variants of a process (e.g. one process model per type of product, per type of customer, or per region or country) in order to compare the behavior of the process in different settings. Accordingly, other studies have considered the use of single-solution-based metaheuristics (S-metaheuristics) such as simulated annealing [29, 18], which are less computationally demanding. However, these latter studies remain at the level of proposals without validation on real-life logs and comparison of trade-offs between alternative metaheuristics.

In this setting, this article studies the following question: *to what extent can the accuracy of APDAs be improved by applying single-solution-based metaheuristics?* To

[1]Here, the term accuracy is used in its informal sense to refer to how well a given process model reflects the event log from which it was discovered. Later in the article, we introduce specific measures of accuracy such as fitness and precision.

address this question, the article outlines a framework to enhance APDAs by applying optimization metaheuristics. The core idea is to perturb the intermediate representation of event logs used by several 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.

The proposed framework can be instantiated by linking it to three components: (i) an automated process discovery approach; (ii) an optimization metaheuristic; and (iii) the quality measure to be optimized, such as fitness, precision, or F-score. The article considers instantiations of the framework corresponding to three APDAs (Inductive Miner [24],[2] Fodina [34], and Split Miner [9]), four optimization metaheuristics (iterative local search, repeated local search, tabu search, simulated annealing), and one accuracy measure (Markovian F-score).

Using a benchmark of 20 real-life logs, the article compares the accuracy gains yielded by the above optimization metaheuristics relative to each other, and relative to the baseline (unoptimized) APDAs upon which they rely. The experimental evaluation also considers the impact of metaheuristic optimization on model complexity measures as well as on execution times.

This article is an extended and revised version of a conference paper [10]. In the conference paper, we presented an approach to optimize the accuracy of one automated process discovery approach, namely Split Miner, by applying S-metaheuristics, and we reported a comparison between the benefits of applying single-solution-based metaheuristics against the benefits of applying P-metaheuristics (using Evolutionary Tree Miner [13] as representative APDA of this category). Our former comparison [10] showed that S-metaheuristics outperform P-metaheuristics not only in terms of execution time efficiency, but also in terms of accuracy of the discovered process models; such a result also supported the findings of the latest literature review of automated process discovery approaches [8]. This article extends our previous approach [10] into a modular framework that can be used to optimize other APDAs, specifically those APDAs that construct a DFG from the event log and use it as an intermediate artifact to discover a process model. This article also extends the conference paper by considering not only Split Miner, but also two other APDAs, namely Fodina and Inductive Miner. Finally, the article reports an empirical evaluation covering all three approaches (Split Miner, Fodina, and Inductive Miner). The evaluation not only proves the applicability and relevance of S-metaheuristics to the problem of automated process discovery, but it also highlights the benefits yielded by S-metaheuristics.

The rest of the paper is structured as follows. The next section gives an overview of APDAs and optimization metaheuristics, where we discuss the background and the related work. Section 3 presents the proposed metaheuristic optimization framework and its instantiations. Section 4 reports on the empirical evaluation and Section 5 draws conclusions and future work directions.

[2]We consider a specific version of Inductive Miner known as "Inductive Miner – directly-follows".

2. Background and Related Work

In this section, we give an overview of existing approaches for 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 [33]: 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 [8] 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 [24], Structured Heuristics Miner [7], Fodina [34], and Split Miner [9]. These algorithms have a hyperparameter to tune the amount of filtering applied when constructing the DFG. Optimizing this and other hyperparameters via greedy search [8], local search strategies [14], or sensitivity analysis techniques [27], 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.

The problem of discovering accurate process models from event logs is inevitably related to that of ensuring event log quality. There is a rich collection of methods for detecting and handling data quality issues in event logs [31]. However, this latter body of work is largely orthogonal to the contribution of this article, as this article focuses on discovering process models assuming that data quality issues have been addressed. This having been said, the methods presented in this paper partially address one type of data quality issue, namely the presence of noise (infrequent behaviour) in an event log [31]. To mitigate the impact of noise on the discovered process model, automated process discovery approaches, including those extended in this paper, apply a dependency filtering step. The optimization techniques proposed in this article fine-tune the level of filtering in order to maximize the accuracy of the discovered process model.

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. Metaheuris-

tics are usually classified into two broad categories [12]: (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 explore a population of solutions generated by mutating, combining, and/or improving previously identified solutions. S-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 computationally heavier than S-metaheuristics, but they are more likely to escape local optima. Providing an exhaustive discussion on all the available metaheuristics is beyond the scope of this paper, in the following, we focus on the four S-metaheuristics that we integrated in our optimization framework and on the P-metaheuristics that have been previously adapted to address the problem of automated process discovery.

Iterated Local Search [30] starts from a (random) solution and explores the neighbouring solutions (i.e. solutions obtained by applying a change to the current solution) 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. The exploration of the solution-space ends when a given termination criterion is met (e.g. maximum iterations, timeout).

Tabu Search [19] is a memory-driven local search. Its initialization includes a (random) solution and three memories: short, intermediate, and long term. 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. The solution-space exploration is repeated until a termination criterion is met.

Simulated Annealing [22] is based on the concepts of Temperature (T, a parameter chosen 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 termination criterion is met, which often relates to the energy or the temperature (e.g. energy below a threshold or $T = 0$).

Evolutionary (Genetic) Algorithms [20, 11] are inspired by Darwin's theory of evolution. Starting from a set of (random) solutions, a new solution is generated by mixing characteristics of two parents selected from the set of the current solutions, such an operation is known as *crossover*. Subsequently, *mutations* are applied to the new solutions to introduce randomness and avoid local optimal solution. Finally, the solutions obtained are assessed and a subset is retained for the next iteration. The algorithm continues until a stop criterion is met.

Swarm Particle Optimization [21] starts from a set of (random) solutions, referred to as particles. Each particle is identified using the concepts of *position* and *velocity*. The position is a proxy for the particle qualities and it embeds the characteristics of the solution, while the velocity is used to alter the position of the particles at each iteration.

Furthermore, each particle has memory of its best position encountered during the roaming of the search space, as well as the best position encountered by any other particle. At each iteration, the algorithm updates the particles positions according to their velocities and updates the best positions found. When a termination condition is met, the algorithm returns the particle having the absolute best position among the whole swarm.

Imperialist Competitive Algorithm [4] is inspired by the historical colonial period. It starts from a (random) set of solutions, called countries. Each country is assessed via an objective function, and a subset is selected as imperialistic countries (the selection is based on their objective function scores). All the countries left (i.e. those having low objective function scores) are considered colonies of the closest (by characteristics) imperialistic country. Then, each colony is altered to resemble its imperialistic country, the objective function scores are re-computed, and the colonies that became better than their imperialistic country are promoted to imperialistic countries and vice-versa. When a termination condition is met, the country with the highest objective function score is selected as the best solution.

2.3. Optimization Metaheuristics in Automated Process Discovery

Optimization metaheuristics have been considered in a few previous studies on automated process discovery. An early attempt to apply P-metaheuristics to automated process discovery was the Genetic Miner proposed by De Medeiros [16], subsequently overtaken by the Evolutionary Tree Miner [13]. Other applications of P-metaheuristics include the contribution of Alizadeh et al. [3] who showed to improve fitness and precision of the discovered process models by using the imperialist competitive algorithm, outperforming some state-of-the-art APDAs (including $\alpha++$ [36], Flexible Heuristics Miner [35], and Fodina [34]), however, the implementation of the method designed by Alizadeh et al. is not publicly available, and the benchmark they used differ from the one suggested in the latest literature review [8]. Some research studies adapted the particle swarm optimization metaheuristic to solve the problem of automated process discovery from event logs [15, 17], but these studies are seminal and they lack of a solid evaluation on real-life logs. One of the most recent studies tried to combine evolutionary computation with particle swarm optimization [25] by extending the work of Buijs et al [13], but also in this case the authors did not provide a working implementation of their method, and they did not assess it on public datasets, so that it is difficult to estimate the real benefits of their proposed improvements. In our context, the main limitation of P-metaheuristics is that they are computationally heavy due to the cost of constructing a solution (i.e. a process model) and evaluating its accuracy. This leads to execution times in the order of hours, to converge to a solution that in the end is comparable to those obtained by state-of-the-art APDAs that do not rely on optimization metaheuristics [8].

Finally, a handful of studies have considered the use of S-metaheuristics to automatically discover optimal process models, specifically simulated annealing [29, 18], but these proposals are preliminary and have not been compared against state-of-the-art approaches on real-life logs.

3. Metaheuristic Optimization Framework

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

3.1. Preliminaries

In order to discover a process model, an APDA takes as input an event log and transforms it into an intermediate representation from which a process model is derived. Below, we define one of the most popular intermediate representations, that is the Directly-Follows Graph (DFG). Although other intermediate representations are available in the literature (e.g behavioral profiles [28]), our framework focuses only on DFGs for two main reasons: first, because they are adopted by many state-of-the-art automated process discovery approaches [35, 24, 7, 34, 9]; second, because they allow us to leverage the Markovian accuracy [5] to facilitate the application of metaheuristics and the navigation of the solution space as we show later in this section.

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 $E = \{(x,y) \in N \times N \mid \exists t = \langle a_1, a_2, \ldots, a_n \rangle, t \in \mathscr{L} \wedge a_i = x \wedge a_{i+1} = y [1 \leq i \leq n-1]\}$.*

By definition, each node of the DFG represents an activity recorded in at least one trace of the event log, whilst each edge of a DFG represents a directly-follows relation between two activities (represented by the node source and the node target of the edge). An APDA is said to be *DFG-based* if it first generates the DFG of the event log, then applies an algorithm to manipulate the DFG (e.g. removing edges), and finally converts the processed DFG into a process model. Such a processed DFG will not adhere any more to Definition 2, therefore, we redefine it as *Refined DFG*.

Definition 3. ***[Refined DFG]*** *Given an event log $\mathscr{L}$ and its DFG $\mathscr{G}_{\mathscr{L}} = (N, E)$, a* Refined DFG *is a directed graph $\mathscr{G} = (N', E')$, where: $N' \subseteq N$ and $E' \subseteq E$. If $N' = N$ and $E' = E$, the refined DFG is equivalent to the event log DFG.*

Examples of DFG-based APDAs are Inductive Miner [24], Heuristics Miner [35, 7], Fodina [34], and Split Miner [9]. Different DFG-based APDAs may extract different *Refined DFGs* from the same log. Also, a DFG-based APDA may discover different *Refined DFGs* from the same log depending on its hyperparameter settings (e.g. a filtering threshold). The algorithm(s) used by a DFG-based APDA to discover the *Refined DFG* from the event log and convert it into a process model may greatly affect the accuracy of an APDA. Accordingly, our framework focuses on optimizing the discovery of the *Refined DFG* rather than its conversion into a process model.

Given that a *Refined DFG* is a binary graph, it is possible to represent it in the form of a matrix as follows.

Definition 4. ***[DFG-Matrix]*** *Given a* Refined DFG $\mathscr{G} = (N, E)$ *and a function* $\theta : N \rightarrow [1, |N|]$,[3] *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* $x_{i,j} = 0$.

In the remaining of this paper, we refer to the *Refined DFG* as DFG for simplicity reason.

3.2. Framework Overview

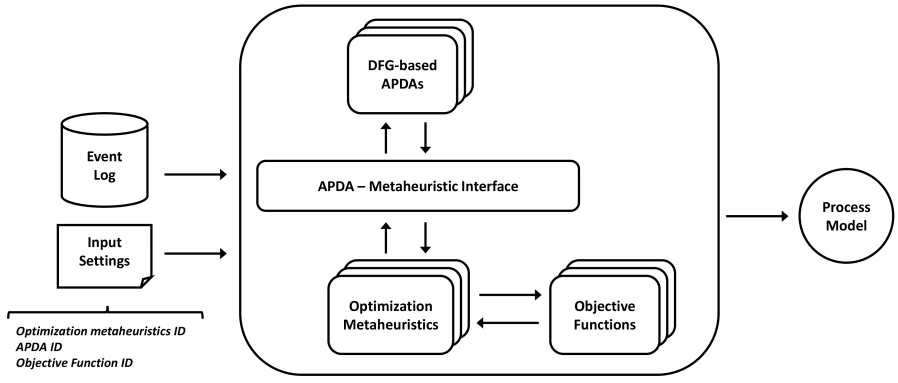

Figure 1: Overview of our optimization framework.

As shown in Figure 1, our framework 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.

Algorithm 1 describes how our framework operates, while Figure 2 captures the control flow representation of the Algorithm 1. First, the input event log is given to the APDA, which returns the discovered (refined) DFG and its corresponding process model (lines 1 and 2). This (refined) DFG becomes the *current DFG*, whilst the process model becomes the *best process model* (so far). This process model's objective function score (e.g. the 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 the function *GenerateNeighbours*, which applies changes to the current DFG to generate a set of neighbouring DFGs (line 6). The latter ones are given as input to the APDA, which returns the corresponding 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 its 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 the latter case, the framework outputs the best process model identified, i.e. the process model scoring the highest value for the objective function.

[3] θ maps each node of the *Refined DFG* to a natural number.

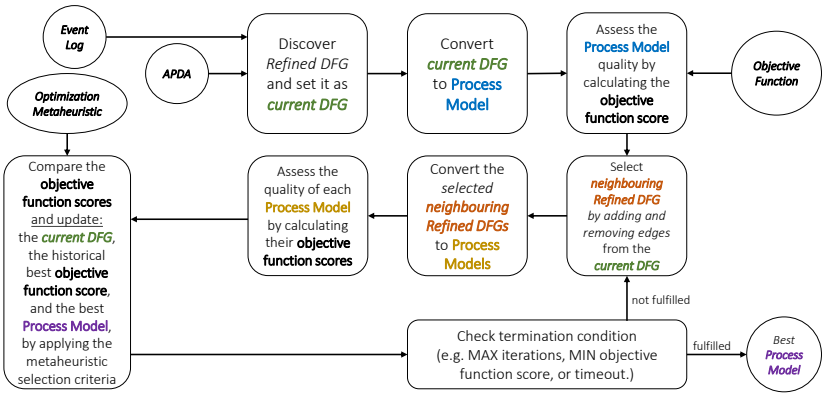

Figure 2: Algorithm 1 – control flow sketch.

Algorithm 1: Optimization Framework

input : Event Log $\mathscr{L}$, Metaheuristic ω, Objective Function $\mathscr{F}$, DFG-based APDA α

CurrentDFG $\mathscr{G}_c \leftarrow$ DiscoverRefinedDFG(α, $\mathscr{L}$);
BestModel $\hat{m} \leftarrow$ ConvertDFGtoProcessModel(α, $\mathscr{G}_c$);
CurrentScore $s_c \leftarrow$ AssessQuality($\mathscr{F}$, $\mathscr{L}$, $\hat{m}$);
BestScore $\hat{s} \leftarrow s_c$;
while *CheckTerminationCriteria() = FALSE* **do**
    Set $V \leftarrow$ GenerateNeighbours($\mathscr{G}_c$, s_c);
    Map $S \leftarrow \varnothing$;
    Map $M \leftarrow \varnothing$;
    for $\mathscr{G} \in V$ **do**
        ProcessModel $m \leftarrow$ ConvertDFGtoProcessModel(α, $\mathscr{G}$);
        Score $s \leftarrow$ AssessQuality($\mathscr{F}$, $\mathscr{L}$, m);
        add ($\mathscr{G}$, s) to S;
        add ($\mathscr{G}$, m) to M;
    $\mathscr{G}_c \leftarrow$ UpdateDFG(ω, S, $\mathscr{G}_c$, s_c, α, $\mathscr{L}$);
    $s_c \leftarrow$ GetMapElement(S, $\mathscr{G}_c$);
    if $\hat{s} < s_c$ **then**
        $\hat{s} \leftarrow s_c$;
        $\hat{m} \leftarrow$ GetMapElement(M, $\mathscr{G}_c$);
return $\hat{m}$;

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 influence 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. Our goal being 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 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 a 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 our optimization framework instantiations we refer to the objective function as the F-score of fitness and precision. Furthermore, we remark that our framework could operate also with objective functions that take into account multiple quality dimensions striving for a trade-off, e.g. F-score and model complexity, provided the multiple quality dimensions can be combined into a unique objective function.

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 that is better than the current one. 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 (captured in Algorithm 2) is the core of our optimization framework, and it implements the metaheuristic itself. The UDF receives in input the selected metaheuristic (ω), the neighbouring DFGs and their corresponding objective function scores (S), the current DFG ($\mathscr{G}_c$), the current score (s_c), the APDA (α), and the event log ($\mathscr{L}$). Then, 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 of the selected metaheuristic (see Algorithm 2, lines 4, 11, and 33 – respectively for ILS, TABU, and SIMA)). In the second case, the current DFG may be a local optimum, and each metaheuristic escapes it with a different strategy. Figures 3, 4, and 5 show the high-level control flow of how ILS, TABU, and SIMA update the current DFG (that is, the UDF – Algorithm 2). *Iterative Local Search* applies the simplest strategy, it perturbs the current DFG (Algorithm 2, line 7). 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 (Algorithm 2,

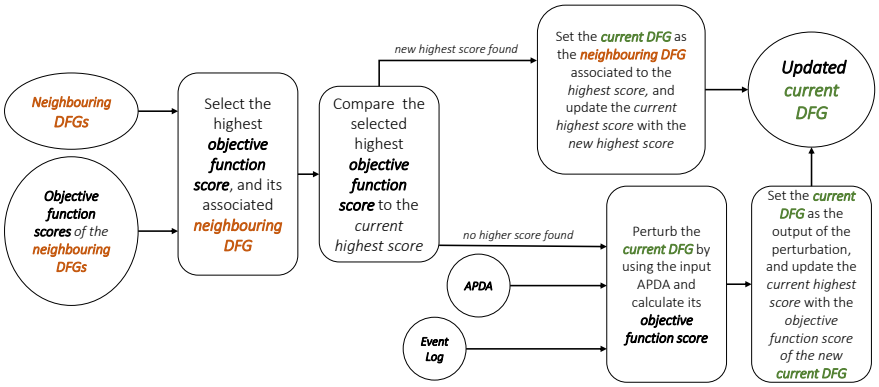

Figure 3: UDF when selecting ILS as optimization metaheuristic.

line 25 to 30). The short-term memory (a.k.a. Tabu-list), which contains DFG that must not be explored further. The intermediate-term memory, which contains DFGs that should lead to better results and, therefore, should be explored in the near future. The long-term memory, which contains 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, it being 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

Algorithm 2: Update DFG Function (UDF)

input : Metaheuristic ω, Neighboring-DFG Scores Map S, Current DFG $\mathscr{G}_c$, Current best score s_c, DFG-based APDA α, Event Log $\mathscr{L}$

$\mathscr{G}_b \leftarrow \mathscr{G}_c$;
if $\omega = ILS$ **then**
  for $\mathscr{G} \in GetMapKeys(S)$ **do**
    if $GetMapElement(S, \mathscr{G}) > s_c$ **then**
      $\mathscr{G}_b \leftarrow \mathscr{G}$;
      $s_c \leftarrow$ GetMapElement($S, \mathscr{G}$);
  if $\mathscr{G}_b = \mathscr{G}_c$ **then** $\mathscr{G}_b \leftarrow$ PerturbDFG($\alpha, \mathscr{G}_c$);
if $\omega = TABU$ **then**
  $s_{tmp} \leftarrow s_c$;
  for $\mathscr{G} \in GetMapKeys(S)$ **do**
    if $GetMapElement(S, \mathscr{G}) > s_c$ **then**
      $\mathscr{G}_b \leftarrow \mathscr{G}$;
      $s_c \leftarrow$ GetMapElement($S, \mathscr{G}$);
  remove $\mathscr{G}_b$ from S;
  $s_c \leftarrow s_{tmp}$;
  for $\mathscr{G} \in GetMapKeys(S)$ **do**
    if $GetMapElement(S, \mathscr{G}) \geq s_c$ **then**
      if $GetMapElement(S, \mathscr{G}) > s_c$ **then**
        add $\mathscr{G}$ to IntermediateTermMemory(ω);
      else
        add $\mathscr{G}$ to LongTermMemory(ω);
    else
      add $\mathscr{G}$ to TabuList(ω);
  add $\mathscr{G}_c$ to TabuList(ω);
  if $\mathscr{G}_b = \mathscr{G}_c$ **then**
    if $IntermediateTermMemory(\omega) \neq \varnothing$ **then**
      $\mathscr{G}_b \leftarrow$ GetLastIntermediateTermMemoryElement(ω);
    else
      if $LongTermMemory(\omega) \neq \varnothing$ **then** $\mathscr{G}_b \leftarrow$ GetLastLongTermMemoryElement(ω);
      $\mathscr{G}_b \leftarrow$ DiscoverDFG($\alpha, \mathscr{L}$);
if $\omega = SIMA$ **then**
  for $\mathscr{G} \in GetMapKeys(S)$ **do**
    if $GetMapElement(S, \mathscr{G}) > s_c$ **then**
      $\mathscr{G}_b \leftarrow \mathscr{G}$;
      $s_c \leftarrow$ GetMapElement($S, \mathscr{G}$);
  if $\mathscr{G}_b = \mathscr{G}_c$ **then**
    for $\mathscr{G} \in GetMapKeys(S)$ **do**
      $s_n \leftarrow$ GetMapElement($S, \mathscr{G}$);
      if $e^{-\frac{|s_n - s_c|}{T}} > Random(0,1)$ **then**
        $\mathscr{G}_b \leftarrow \mathscr{G}$;
return $\mathscr{G}_b$;

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 (Algorithm 2, line 36 to 40). 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 neighbour 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).

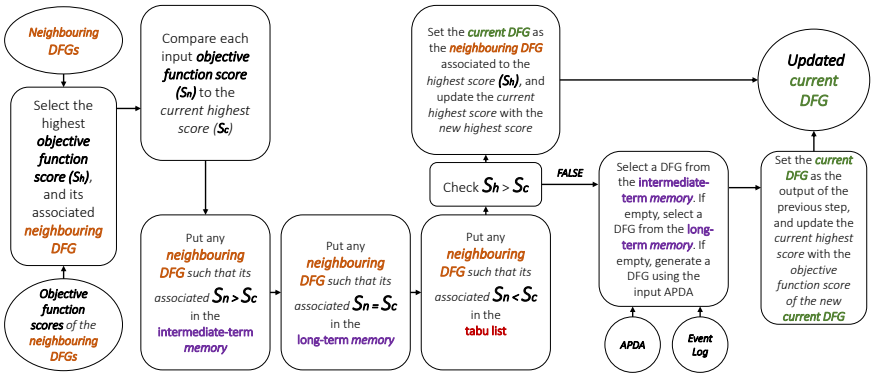

Figure 4: UDF when selecting TABU as optimization metaheuristic.

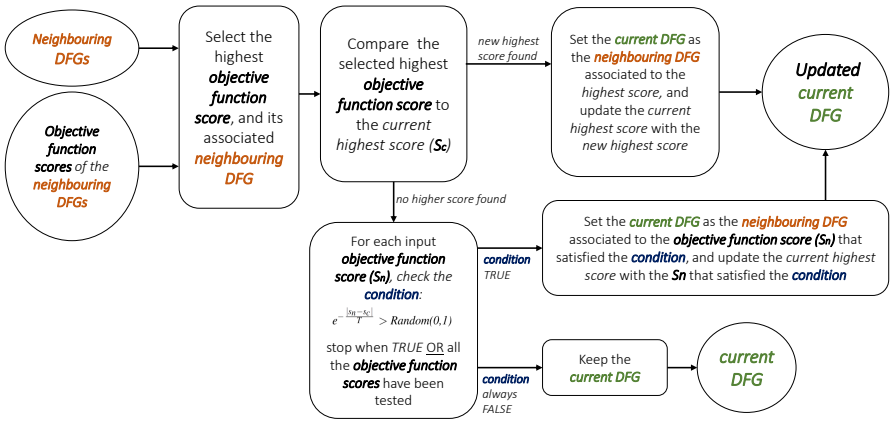

Figure 5: UDF when selecting SIMA as optimization metaheuristic.

3.4. Framework Instantiation

To assess our framework, we instantiated it for three APDAs: Split Miner [9], Fodina [34], and Inductive Miner [24]. These three APDAs are all DFG-based, and

they are representatives of the state of the art. In fact, the latest APDAs literature review and benchmark [8] showed that Fodina, Split Miner, and Inductive Miner outperformed other APDAs when their hyperparameters were optimized via a brute-force approach. Therefore, we decided to focus on those DFG-based APDAs that would benefit the most from the application of our optimization framework.

To complete the instantiation of our framework for any concrete DFG-based APDA, it is necessary to implement an interface that allows the metaheuristics to interact with the APDA (as discussed above). Such an 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, *DiscoverDFG* and *ConvertDFGtoProcessModel*, are inherited from the DFG-based APDA, in our case Split Miner, Fodina, and Inductive Miner. We note that Split Miner and Fodina receive as input parameter settings that can vary the output of the *DiscoverDFG* function. Precisely, *Split Miner* has two parameters: 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. Whilst, *Fodina* has three parameters: the noise filtering threshold, similar to the one of Split Miner, and two threshold to detect respectively self-loops and short-loops in the DFG. Instead, the DFG-based variant of Inductive Miner [24] that we integrated in our optimization framework does not receive any input parameters.

To discover the initial DFG (Algorithm 1, line 1) with Split Miner, default parameters are used. [4] 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 [9], and starting the solution space exploration from this latter can reduce the total exploration time.

Similarly, if Fodina is the selected APDA, the initial DFG (Algorithm 1, line 1) is discovered using the default parameters of Fodina, [5] even though there is no guarantee that the default parameters allow Fodina to discover a good starting solution [8]. Yet, this design choice is less risky than randomly choosing the values of the input parameters in order to discover the initial DFG, because it is likely Fodina would discover unsound models when randomly tuned, given that it does not guarantee soundness.

On the other hand, Inductive Miner [24] does not apply any manipulation to the discovered initial DFG. In this case, we pseudorandomly generate an initial DFG starting from a given seed, to ensure determinism. Differently than the case of Fodina, this is a suitable design choice for Inductive Miner, because it always guarantees block-structured sound process models, regardless of the DFG.

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* function of Split Miner and Fodina to output the DFG discovered with default parameters the first time it is executed, and a DFG discovered with pseudorandom parameters for the following executions. The case of Inductive Miner is simpler, because the *DiscoverDFG* function always returns a pseudorandom DFG. Consequently, we mapped RF to the *DiscoverDFG* function.

[4] The default parameters of Split Miner are the most likely to yield the best results [9].

[5] The default parameters of Fodina are the most likely to yield the best results [34].

Finally, function PF can be provided either by the APDA (through 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 chose a different PF for each of the three APDAs.

- **Split Miner PF.** 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 that is not present in the current solution, two edges are removed from the DFG, whils, for each deprecated parallel relation, two edges are added to the DFG.

- **Fodina PF.** Given the current DFG, we analyse its self-loops and short-loops relations using random loop thresholds. As a result, a new DFG is generated where a different set of edges is retained as self-loops and short-loops.

- **Inductive Miner PF.** Since Inductive Miner does not perform any manipulation on the DFG, we could not determine an efficient way to perturb the DFG. Thus, we set PF = RF, so that instead of perturbing the current DFG, a new random DFG is generated. This variant of the ILS is called Repeated Local Search (RLS). In the evaluation reported in Section 4, we use only RLS for Inductive Miner, and both ILS and RLS for Fodina and Split Miner.

To complete the instantiation of our framework, we need to set an objective function. With the goal of optimizing the accuracy of the APDAs, we chose as objective function the F-score of fitness and precision. Among the existing measures of fitness and precision, we selected the Markovian fitness and precision presented in [6, 5].[6] The rationale for this choice is that these measures of fitness and precision are the fastest to compute among state-of-the-art measures [6, 5]. Furthermore, these measures indicate what edges could be added to or removed from the DFG to improve the fitness or precision of the model. This feedback allows us to design an effective GNF.

In the instantiation of our framework, 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 edges that could be added to improve fitness or removed to improve precision. Algorithm 3 illustrates how we build this data structure, its high-level control flow sketch is captured in Figure 6.

Given an event log and a process model, we generate their respective Markovian abstractions by applying the method described in [5] (lines 1 and 2). We recall that the Markovian abstraction of the log/model is a graph, where each edge represents a subtrace[7] observed in the log/model. Next, we collect all the edges of the Markovian abstraction of the log and of the model into two different sets: E_l and E_m (lines 3 and 4). These two sets are used to determine the Markovian fitness and precision of the process model [5], by applying the formula in lines 5 and respectively 10. We note that the edges in E_l that cannot be found in E_m (set E_{df}, line 6) represent subtraces of the

[6] We used the Boolean function variant with order $k = 5$.

[7] The length of the subtrace is determined by the order of the Markovian abstraction, in our case $k = 5$.

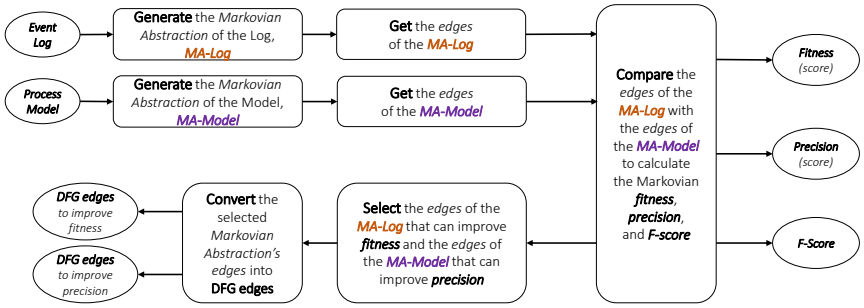

Figure 6: Algotihm 3 – control flow sketch.

Algorithm 3: Assess Quality with Markovian Accuracy

input : Event Log $\mathscr{L}$, Model m

1 MarkovianAbstraction $L \leftarrow$ GenerateMarkovianAbstraction($\mathscr{L}$);
2 MarkovianAbstraction $P \leftarrow$ GenerateMarkovianAbstraction(m);
3 Set $E_l \leftarrow$ GetEdges(L);
4 Set $E_m \leftarrow$ GetEdges(P);
5 $f \leftarrow 1 - \frac{|E_l \setminus E_m|}{|E_l|}$;
6 Set $E_{df} \leftarrow E_l \setminus E_m$;
7 Set $E_f \leftarrow \varnothing$;
8 **for** $e \in E_{df}$ **do**
9 $\quad E_f \leftarrow E_f \cup$ ExtractDFGEdges(e);
10 $p \leftarrow 1 - \frac{|E_m \setminus E_l|}{|E_m|}$;
11 $E_{dp} \leftarrow E_m \setminus E_l$;
12 Set $E_p \leftarrow \varnothing$;
13 **for** $e \in E_{dp}$ **do**
14 $\quad E_p \leftarrow E_p \cup$ ExtractDFGEdges(e);
15 $s \leftarrow \frac{2 \cdot f \cdot p}{f+p}$;
16 **return** (s, f, p, E_f, E_p);

log that cannot be found in the process model. Vice-versa, the edges in E_m that cannot be found in E_l (set E_{dp}, line 11) represent subtraces of the process model that cannot be found in the log. We analyse these subtraces to detect directly-follows relations, i.e. DFG edges (lines 9 and 14), that can be added or removed from the DFG that generated the process model in order to either improve fitness or precision. Precisely, the DFG edges that can be added to improve fitness are those captured by the directly-follows relations that we can find in the Markovian abstraction edges in set E_{df}. On the other hand, the edges that can be removed to improve precision are those captured by the directly-follows relations that we can find in the Markovian abstraction edges in set E_{dp}. Once these edges to be added or removed are identified (sets E_f and E_p), we can output the final data structure, which comprises the Markovian fitness and precision, their F-score, and the two sets E_f and E_p.

Given the above objective function's output, our GNF is described in Algorithm 4, while Figure 7 captures its high-level control flow sketch. This function receives as input the current DFG ($\mathscr{G}_c$), its objective function score (the data structure s_c), and the number of neighbours to generate ($size_n$). If fitness is greater than precision, we retrieve

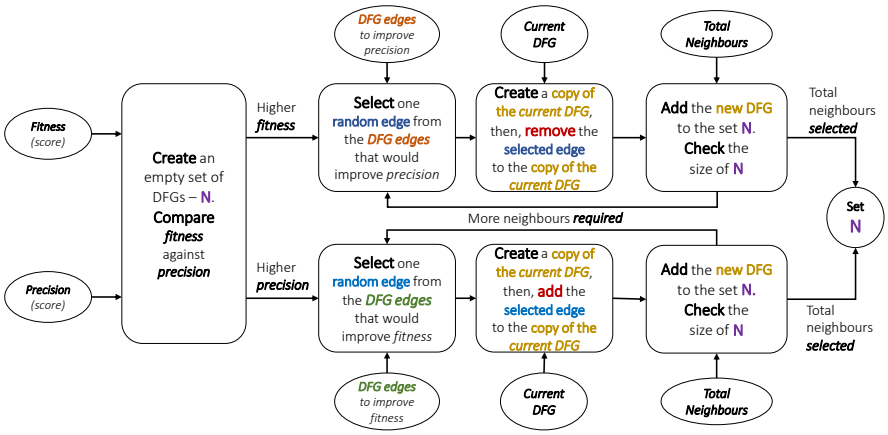

Figure 7: Algotihm 4 – control flow sketch.

Algorithm 4: Generate Neighbours Function (GNF)

input : CurrentDFG $\mathscr{G}_c$, 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 $\mathscr{G}_n \leftarrow$ copyDFG($\mathscr{G}_c$);
9 **if** $getFitnessScore(s_c) > getPrecisionScore(s_c)$ **then**
10 **if** $canRemoveEdge(\mathscr{G}_n, e)$ **then** add $\mathscr{G}_n$ to N;
11 **else**
12 addEdge($\mathscr{G}_n$, e);
13 add $\mathscr{G}_n$ to N;
14 **return** N;

from s_c the set of edges (E_m) that could be removed from $\mathscr{G}_c$ 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 $\mathscr{G}_c$ 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 lowest of the two measures (precision or fitness). That is, if the fitness is lower, we increase fitness, and conversely if the precision is lower we increase precision. Once we have E_m, we randomly select one edge from it, generate a copy of the current DFG ($\mathscr{G}_n$), and 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 $\mathscr{G}_n$, 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 maximum size (i.e. $size_n$). We then return N. The algorithm ends when the maximum execution time or the maximum number of iterations is reached.

4. Evaluation

We implemented the proposed optimization framework as a Java command-line application.[8] This tool uses Split Miner, Fodina, and Inductive Miner as the underlying APDAs, and the Markovian accuracy F-score as the objective function (cf. Section 3.4). Using this implementation, we undertook to empirically evaluate the magnitude of improvements in accuracy delivered by different instantiations of the framework.

4.1. Dataset, Quality Measures, and Experimental Setup

For our evaluation we used the dataset of the benchmark of automated process discovery approaches in [8], 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), the *Road Traffic Fines Management Process* (RTFMP) log and the *SEPSIS* log. These logs record executions of business processes from a variety of domains, e.g. healthcare, finance, government and IT service management. 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. The dataset comprises simple logs (e.g. $\text{BPIC13}_{\text{cp}}$) and very complex ones (e.g. SEPSIS, PRT2) in terms of percentage of distinct traces, and both small logs (e.g. $\text{BPIC13}_{\text{cp}}$ and SEPSIS) and large ones (e.g. BPIC17 and PRT9) in terms of total number of events.

Log		**BPIC12**	**$\text{BPIC13}_{\text{cp}}$**	**$\text{BPIC13}_{\text{inc}}$**	**BPIC14_{f}**	**$\text{BPIC15}_{\text{1f}}$**	**$\text{BPIC15}_{\text{2f}}$**	**$\text{BPIC15}_{\text{3f}}$**	**$\text{BPIC15}_{\text{4f}}$**	**$\text{BPIC15}_{\text{5f}}$**
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).

From each of these logs, we discovered 16 process models by applying the following techniques:

- Split Miner with default parameters (SM);
- Split Miner with hyper-parameter optimization[9](HPO_{sm});
- Split Miner optimized with our framework using the following optimization metaheuristics: RLS_{sm}, ILS_{sm}, TABU_{sm}, SIMA_{sm};

[8] Available under the label "Optimization Framework for Automated Process Discovery" at `http://apromore.org/platform/tools`.

- Fodina with default parameters (FO);
- Fodina with hyper-parameter optimization[9] (HPO_{fo});
- Fodina optimized with our framework using the following optimization metaheuristics: RLS_{fo}, ILS_{fo}, $TABU_{fo}$, $SIMA_{fo}$;
- Inductive Miner IM_d;
- Inductive Miner optimized with our framework using the following optimization metaheuristics: RLS_{imd}, $TABU_{imd}$, $SIMA_{imd}$.

For each of the above metaheuristics, we set the maximum execution time to five minutes and the maximum number of iterations to 50. The same timeout was also applied to the hyper-parameter optimizations.

For each of the discovered models we measured fitness, precision, complexity and execution time. For measuring fitness and precision, we adopted two different sets of measures. The first set of measures is based on *alignments*, computing fitness and precision with the approaches proposed by Adriansyah et al. [2, 1] (alignment-based accuracy). *Alignment-based fitness* selects for each trace in the log, the closest trace recognized by the process model, and measures the minimal number of error-corrections required to align these two traces (a.k.a. *minimal alignment cost*). The final fitness score is equal to one minus the normalized sum of the minimal alignment cost between each trace in the log and the closest corresponding trace recognized by the model. *Alignment-based precision* builds a *prefix automaton* from the event log, then it replays the process model behavior on top of the log prefix automation (with the aid of alignments) and counts the number of times that the model can perform a move that the prefix automaton cannot. Each of these mismatching moves is called an *escaping edge*. The final value of precision is function of the number of detected escaping edges. For more details regarding the Alignment-based fitness and precision, we refer to the corresponding studies [2, 1].

The second set of measures is based on *Markovian abstractions*, computing fitness and precision with the approaches in [5]. The *Markovian fitness* generates a Markovian abstraction from the behavior recorded in the event log and a Markovian abstraction from the behavior allowed by the process model. As mentioned in the previous section, a Markovian abstraction is a graph where each node represents a subtrace of a fixed length. The *Markovian fitness* relies on a graph comparison algorithm [23] to identify the edges of the Markovian abstraction generated from the log that do not appear in the Markovian abstraction generated from the process model. Similarly, the *Markovian precision* is calculated by identifying (via the same graph comparison algorithm [23]) the edges of the Markovian abstraction of the process model that do not appear in the Markovian abstraction of the log. For more details regarding the Alignment-based fitness and precision, we refer to the corresponding study [5].

For assessing the complexity of the models we relied on *size*, Control-Flow Complexity (CFC), and Structuredness. *Size* is the total number of nodes of a process model;

[9] Using the Markovian accuracy F-score as objective function.

Control flow complexity (CFC) is the amount of branching induced by the split gateways in a process model; *Structuredness* is the percentage of nodes located inside a single-entry single-exit fragment of a process model.

Note that we did not measure the generalization of the discovered process models because available generalization measures assess the capability of an APDA to generalise the behavior recorded in the event log during the discovery of a process model, and they do not assess the generalisation of the process model itself [32]. However, this should not be seen as a limitation of this study, since our objective is to analyse the benefits yielded by our optimization framework in terms of F-score (through fitness and precision).

We used the results of these measurements to compare the quality of the models discovered by each baseline APDA (SM, FO, IM_d) against the quality of the models discovered by the respective optimized approaches.

All 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 framework implementation, the batch tests, the results, and all the (public) models discovered during the experiments are available for reproducibility purposes at `https://doi.org/10.6084/m9.figshare.11413794`.

4.2. Split Miner

Tables 2 and 3 show the results of our comparative evaluation for Split Miner. Each row reports the quality of each discovered process model in terms of accuracy (both alignment-based and Markovian), complexity, and discovery time. We held out from the tables four logs: $BPIC13_{cp}$, $BPIC13_{inc}$, 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 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 consistently improved the Markovian F-score over that achieved by 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 $SIMA_{sm}$ (eight times), RLS_{sm} and $TABU_{sm}$ (six times each). We note however that the F-score difference between the four metaheuristics is small (in the order of one to two percentage points).

Despite the fact that the objective function of the metaheuristics was the Markovian F-score, all four metaheuristics also 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. This does not hold for precision. The result highlights the partial correlation between alignment-based and Markovian measures, already discussed in the previous section.

By close inspection to the complexity of the models, we note that most of the times (nine cases 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 [9]. To improve the F-score, new behavior is added to the

Event Log	Discovery Approach	Align. Acc.			Markov. Acc. ($k = 5$)			Complexity			Exec. Time(s)
		Fitness	Prec.	F-score	Fitness	Prec.	F-score	Size	CFC	Struct.	
BPIC12	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
$BPIC14_f$	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
$BPIC15_{1f}$	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
$BPIC15_{2f}$	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
$BPIC15_{3f}$	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
$BPIC15_{4f}$	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
$BPIC15_{5f}$	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	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	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 - Split Miner.

Event Log	Discovery Method	Align. Acc.			Markov. Acc. ($k = 5$)			Complexity			Exec. Time(s)
		Fitness	**Prec.**	**F-score**	**Fitness**	**Prec.**	**F-score**	**Size**	**CFC**	**Struct.**	
PRT1	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
	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
PRT2	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
	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
PRT3	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
	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
PRT4	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
	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
PRT6	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
	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
PRT7	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
	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
PRT10	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
	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 - Split Miner.

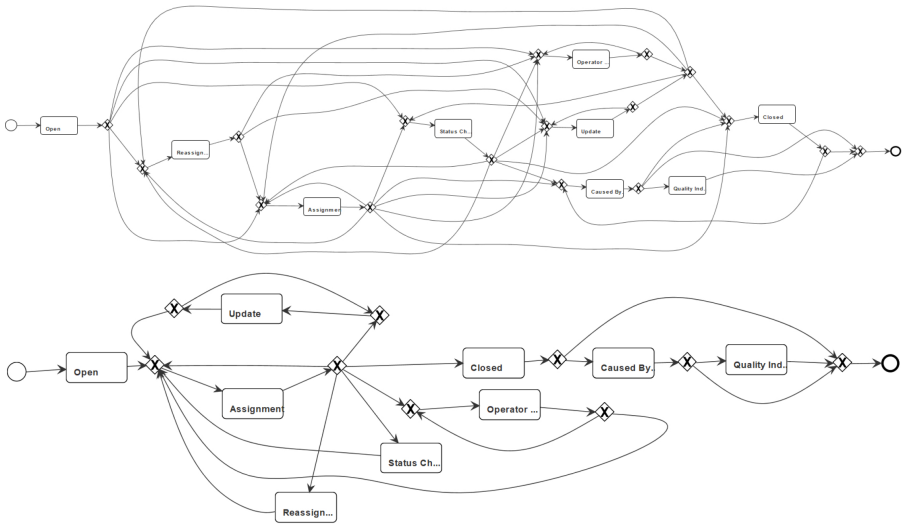

Figure 8: BPIC14$_{\text{f}}$ model discovered with SIMA$_{\text{sm}}$ (above) and with SM (below).

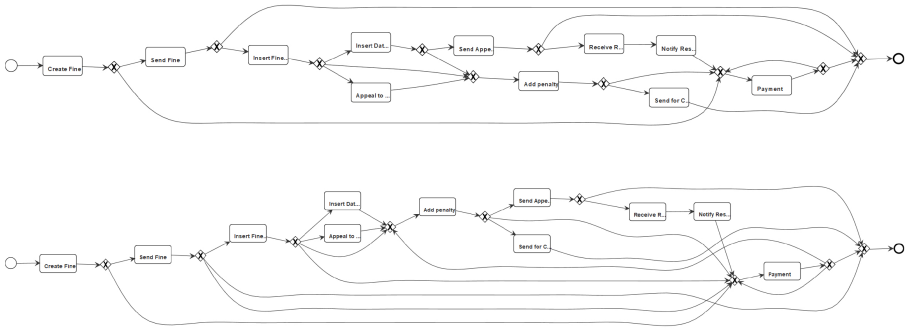

Figure 9: RTFMP model discovered with SIMA$_{\text{sm}}$ (above) and with SM (below).

model in the form of new edges (note that new nodes are never added); this leads to new gateways and consequently to higher size and CFC. On the other hand, when precision is lower than fitness, and thus the metaheuristic aims to increase the value of precision 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. Supporting examples of these two possible scenarios are Figure 8 and Figure 9. Figure 8 shows the models discovered by SIMA$_{\text{sm}}$ and SM from the BPIC14$_{\text{f}}$ log, where the model discovered by SIMA$_{\text{sm}}$ is more complex than that obtained with SM because it was necessary to improve its fitness (adding edges). While Figure 9 shows the models discovered by SIMA$_{\text{sm}}$ and SM from the RTFMP log, where the model discovered by SIMA$_{\text{sm}}$ is simpler than that obtained with SM because it was necessary to improve the precision (removing edges). Comparing the results obtained by the metaheuristics with HPO$_{\text{sm}}$, we can see that our approach allows us to discover models that cannot be

discovered simply by tuning the parameters of SM. This relates to the solution space exploration. Indeed, HPO_{sm} can only explore a limited number of solutions (DFGs), i.e. those that can be generated by underlying APDA (SM in this case) by varying its parameters. In contrast, the metaheuristics go beyond the solution space of HPO_{sm} by exploring new DFGs in a pseudorandom 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}, while consistently achieving higher accuracy.

4.3. Fodina

Tables 4 and 5 report the results of our comparative evaluation for Fodina. In these tables, we used "-" to report that a given accuracy measurement could not be reliably obtained due to the unsoundness of the discovered process model. We held out from the tables two logs: BPIC12 and SEPSIS, because none of the six approaches (base APDA, hyper-parameter optimized and the four metaheuristics) was able to discover a sound process model. This is due to Fodina's design which does not guarantee soundness.

Considering the remaining 18 logs, eleven times *all* the metaheuristics improved the Markovian F-score w.r.t. HPO_{fo} (and consequently FO), whilst 16 times at least one metaheuristic outperformed both FO and HPO_{fo}. The only two cases where none of the metaheuristics was able to discover a more accurate process model than HPO_{fo} were PRT2 and $BPIC14_f$. In the former log, this is because none of the metaheuristics discovered a sound process model within the given timeout of five minutes. However, we note that HPO_{fo} took almost four hours to discover a sound process model from the PRT2 log. In the latter log, this is because all the metaheuristics discovered the same model of HPO_{fo}.

Among the optimization metaheuristics, $TABU_{fo}$ performed the best. This metaheuristic achieved 14 times out of 18 the highest Markovian F-score, followed by ILS (ten times). However, like for Split Miner, the differences in the achieved F-score between the four metaheuristics are small. There is a difference of only 1-2 percentage points between the metaheuristics with highest F-score and the one with lowest F-score.

In the case of Fodina, the results achieved by the metaheuristics on the alignment-based F-score are more remarkable than the case of Split Miner, and in-line with the results obtained on the Markovian F-score. Indeed, 50% of the times, *all* the metaheuristics were able to outperform both FO and HPO_{fo} on the alignment-based F-score, and more than 80% of the times, at least one metaheuristic scored a higher alignment-based F-score than FO and HPO_{fo}. Such a result is remarkable considering that the objective function of the metaheuristics was the Markovian F-score.

Regarding the complexity of the models discovered by the metaheuristics, more than 50% of the times, it is lower than the complexity of the models discovered by FO and HPO_{fo}, and in the remaining cases in-line with the two baselines. Such a difference with the results we obtained for SM relates to the following two factors: (i) Split Miner discovers much simpler models than Fodina, and any further improvement is difficult to achieve; (ii) Fodina natively discovers more fitting models than Split Miner and hence, the metaheuristics aim at improving precision, ultimately removing model edges, and so reducing its complexity.

In terms of execution times, the four metaheuristics perform similarly, with an execution time between 150 and 300 seconds, slightly higher than the case of Split Miner.

Event Log	Discovery Approach	Align. Acc.			Markov. Acc. ($k=5$)			Complexity			Exec.
		Fitness	**Prec.**	**F-score**	**Fitness**	**Prec.**	**F-score**	**Size**	**CFC**	**Struct.**	**Time(s)**
$BPIC13_{cp}$	FO	**0.999**	0.879	0.935	**0.997**	0.647	0.784	13	10	0.77	**0.1**
	HPO_{fo}	**0.999**	0.879	0.935	**0.997**	0.647	0.784	13	10	0.77	17.7
	RLS_{fo}	0.994	**0.963**	**0.978**	0.947	**0.864**	**0.904**	**12**	9	0.67	290.6
	ILS_{fo}	0.994	0.880	0.934	0.935	0.758	0.837	**12**	**8**	**0.92**	151.2
	$TABU_{fo}$	0.994	**0.963**	**0.978**	0.947	**0.864**	**0.904**	**12**	9	0.67	95.2
	$SIMA_{fo}$	0.994	0.880	0.934	0.935	0.758	0.837	**12**	**8**	**0.92**	130.0
$BPIC13_{inc}$	FO	0.994	**0.877**	**0.932**	0.950	0.576	0.717	**13**	**10**	0.85	**0.291**
	HPO_{fo}	0.994	**0.877**	**0.932**	0.950	0.576	0.717	**13**	**10**	0.85	112.0
	RLS_{fo}	0.994	**0.877**	**0.932**	0.950	0.576	0.717	**13**	**10**	0.85	304.7
	ILS_{fo}	0.994	**0.877**	**0.932**	0.950	0.576	0.717	**13**	**10**	0.85	180.1
	$TABU_{fo}$	**0.998**	0.743	0.852	**0.987**	**0.604**	**0.749**	14	15	**1.00**	129.0
	$SIMA_{fo}$	0.994	**0.877**	**0.932**	0.950	0.576	0.717	**13**	**10**	0.85	146.1
$BPIC14_{f}$	FO	-	-	-	-	-	-	37	46	0.41	**36.8**
	HPO_{fo}	**1.000**	**0.757**	**0.862**	**1.000**	**0.985**	**0.992**	**27**	**36**	**0.56**	8612.7
	RLS_{fo}	**1.000**	**0.757**	**0.862**	**1.000**	**0.985**	**0.992**	**27**	**36**	**0.56**	370.7
	ILS_{fo}	**1.000**	**0.757**	**0.862**	**1.000**	**0.985**	**0.992**	**27**	**36**	**0.56**	365.5
	$TABU_{fo}$	**1.000**	**0.757**	**0.862**	**1.000**	**0.985**	**0.992**	**27**	**36**	**0.56**	358.5
	$SIMA_{fo}$	**1.000**	**0.757**	**0.862**	**1.000**	**0.985**	**0.992**	**27**	**36**	**0.56**	300.2
$BPIC15_{1f}$	FO	**1.000**	0.760	0.860	**1.000**	0.480	0.650	146	91	0.26	**0.3**
	HPO_{fo}	**1.000**	0.756	0.861	**1.000**	0.479	0.648	146	91	0.26	130.5
	RLS_{fo}	0.916	0.829	0.870	0.804	0.772	0.788	131	69	0.24	301.9
	ILS_{fo}	0.916	0.829	0.870	0.804	0.772	0.788	131	69	0.24	198.4
	$TABU_{fo}$	0.916	0.830	0.871	0.802	0.778	**0.790**	129	**67**	0.33	177.5
	$SIMA_{fo}$	0.918	**0.833**	**0.873**	0.777	**0.799**	0.788	**127**	**67**	**0.34**	174.4
$BPIC15_{2f}$	FO	-	-	-	-	-	-	195	159	0.09	**48.5**
	HPO_{fo}	-	-	-	-	-	-	187	145	0.11	118.7
	RLS_{fo}	-	-	-	-	-	-	181	131	0.09	306.0
	ILS_{fo}	-	-	-	-	-	-	**175**	**120**	0.11	276.1
	$TABU_{fo}$	**0.876**	**0.754**	**0.810**	**0.653**	**0.608**	**0.630**	177	**120**	**0.13**	262.3
	$SIMA_{fo}$	-	-	-	-	-	-	**175**	121	0.12	284.1
$BPIC15_{3f}$	FO	-	-	-	-	-	-	174	164	0.06	**4.3**
	HPO_{fo}	**0.983**	0.601	0.746	**0.925**	0.208	0.339	163	161	0.07	402.9
	RLS_{fo}	-	-	-	-	-	-	166	141	0.07	303.5
	ILS_{fo}	0.924	**0.713**	**0.805**	0.701	**0.444**	**0.543**	**158**	**131**	**0.10**	247.1
	$TABU_{fo}$	-	-	-	-	-	-	163	**131**	0.09	235.5
	$SIMA_{fo}$	-	-	-	-	-	-	163	**131**	0.09	241.8
$BPIC15_{4f}$	FO	-	-	-	-	-	-	157	127	0.15	**1.3**
	HPO_{fo}	**0.995**	0.660	0.793	**0.973**	0.302	0.461	153	126	0.14	443.0
	RLS_{fo}	0.887	0.790	0.836	0.708	0.610	0.655	**127**	77	**0.17**	308.3
	ILS_{fo}	0.882	0.801	**0.839**	0.697	0.628	0.661	**127**	75	**0.17**	300.5
	$TABU_{fo}$	0.864	**0.806**	0.834	0.675	**0.652**	**0.663**	**127**	**74**	**0.17**	274.5
	$SIMA_{fo}$	0.882	0.801	**0.839**	0.697	0.628	0.661	**127**	75	**0.17**	252.2
$BPIC15_{5f}$	FO	**1.000**	0.698	0.822	**1.000**	0.362	0.532	166	125	**0.15**	**2.4**
	HPO_{fo}	**1.000**	0.698	0.822	**1.000**	0.362	0.532	166	125	**0.15**	238.1
	RLS_{fo}	0.886	0.810	0.846	0.727	0.703	0.715	150	94	0.11	303.4
	ILS_{fo}	0.884	**0.819**	**0.850**	0.719	0.724	0.722	147	90	0.13	268.1
	$TABU_{fo}$	0.886	0.814	0.849	0.723	0.730	0.727	149	92	0.11	217.1
	$SIMA_{fo}$	0.884	0.808	0.844	0.721	**0.743**	**0.732**	**141**	**83**	0.14	208.5
$BPIC17_{f}$	FO	**1.000**	**0.675**	**0.806**	**1.000**	0.330	0.496	35	22	0.69	**22.4**
	HPO_{fo}	**1.000**	**0.675**	**0.806**	**1.000**	0.330	0.496	35	22	**0.71**	9755.7
	RLS_{fo}	0.999	**0.675**	**0.806**	0.997	**0.331**	**0.497**	**33**	**20**	0.70	309.9
	ILS_{fo}	0.999	**0.675**	**0.806**	0.997	**0.331**	**0.497**	**33**	**20**	0.70	313.5
	$TABU_{fo}$	0.999	**0.675**	**0.806**	0.997	**0.331**	**0.497**	**33**	**20**	0.70	305.7
	$SIMA_{fo}$	0.999	**0.675**	**0.806**	0.997	**0.331**	**0.497**	**33**	**20**	0.70	319.2
RTFMP	FO	**0.996**	0.933	0.964	**0.937**	0.148	0.256	31	32	0.19	**0.4**
	HPO_{fo}	0.884	**1.000**	0.939	0.646	0.857	0.737	**18**	**7**	**0.56**	2666.2
	RLS_{fo}	0.987	**1.000**	**0.994**	0.848	**0.938**	**0.890**	26	25	0.12	268.7
	ILS_{fo}	0.987	**1.000**	**0.994**	0.848	**0.938**	**0.890**	26	25	0.12	134.1
	$TABU_{fo}$	0.987	**1.000**	**0.994**	0.848	**0.938**	**0.890**	26	25	0.12	131.2
	$SIMA_{fo}$	0.987	**1.000**	0.993	0.847	0.923	0.883	28	27	0.11	133.9

Table 4: Comparative evaluation results for the public logs - Fodina.

Event Log	Discovery Method	Align. Acc.			Markov. Acc. ($k = 5$)			Complexity			Exec. Time(s)
		Fitness	Prec.	F-score	Fitness	Prec.	F-score	Size	CFC	Struct.	
PRT1	FO	-	-	-	-	-	-	30	28	0.53	**0.2**
	HPO_{fo}	0.998	0.925	0.960	**0.988**	0.739	0.845	**21**	17	0.81	402.6
	RLS_{fo}	0.988	**0.964**	**0.976**	0.888	0.827	0.857	**21**	**16**	**0.86**	302.7
	ILS_{fo}	0.988	**0.964**	**0.976**	0.888	0.827	0.857	**21**	**16**	**0.86**	183.1
	$TABU_{fo}$	**0.994**	0.957	**0.976**	0.981	**0.844**	**0.907**	**21**	17	**0.86**	149.3
	$SIMA_{fo}$	0.988	**0.964**	**0.976**	0.888	0.827	0.857	**21**	**16**	**0.86**	154.0
PRT2	FO	-	-	-	-	-	-	38	**45**	0.76	**92.7**
	HPO_{fo}	**1.000**	**0.276**	**0.432**	**0.998**	**0.148**	**0.258**	**29**	78	**1.00**	12937.1
	RLS_{fo}	-	-	-	-	-	-	48	56	0.08	301.0
	ILS_{fo}	-	-	-	-	-	-	48	56	0.08	308.1
	$TABU_{fo}$	-	-	-	-	-	-	53	70	0.08	313.0
	$SIMA_{fo}$	-	-	-	-	-	-	-	-	-	854.9
PRT3	FO	**0.999**	0.847	0.917	**0.993**	0.269	0.423	**34**	**37**	0.32	**0.2**
	HPO_{fo}	-	-	-	-	-	-	73	93	0.18	756.5
	RLS_{fo}	0.963	**0.902**	**0.932**	0.679	**0.446**	**0.539**	37	38	**0.35**	306.6
	ILS_{fo}	0.963	**0.902**	**0.932**	0.679	**0.446**	**0.539**	37	38	**0.35**	157.3
	$TABU_{fo}$	0.963	**0.902**	**0.932**	0.679	**0.446**	**0.539**	37	38	**0.35**	138.1
	$SIMA_{fo}$	0.963	**0.902**	**0.932**	0.679	**0.446**	**0.539**	37	38	**0.35**	143.0
PRT4	FO	-	-	-	-	-	-	37	40	0.54	**46.0**
	HPO_{fo}	**1.000**	0.858	0.924	**1.000**	0.965	0.982	32	41	0.50	10914.5
	RLS_{fo}	0.997	0.859	0.923	0.993	0.990	0.991	31	37	0.52	317.4
	ILS_{fo}	0.997	**0.903**	**0.948**	0.993	**0.993**	**0.993**	**27**	**32**	**0.74**	314.4
	$TABU_{fo}$	0.997	**0.903**	**0.948**	0.993	**0.993**	**0.993**	**27**	**32**	**0.74**	300.1
	$SIMA_{fo}$	0.977	0.887	0.930	0.793	0.963	0.870	32	38	0.50	309.1
PRT6	FO	**1.000**	0.908	0.952	**1.000**	0.632	0.775	**22**	17	**0.41**	**0.1**
	HPO_{fo}	**1.000**	0.908	0.952	**1.000**	0.632	0.775	**22**	17	**0.41**	25.0
	RLS_{fo}	0.984	**0.928**	**0.955**	0.840	**0.818**	**0.829**	**22**	**14**	**0.41**	278.8
	ILS_{fo}	0.984	**0.928**	**0.955**	0.840	**0.818**	**0.829**	**22**	**14**	**0.41**	140.0
	$TABU_{fo}$	0.984	**0.928**	**0.955**	0.840	**0.818**	**0.829**	**22**	**14**	**0.41**	129.3
	$SIMA_{fo}$	0.984	**0.928**	**0.955**	0.840	**0.818**	**0.829**	**22**	**14**	**0.41**	131.2
PRT7	FO	0.990	**1.000**	0.995	0.906	**1.000**	0.951	**26**	**16**	**0.39**	**0.3**
	HPO_{fo}	0.990	**1.000**	0.995	0.906	**1.000**	0.951	**26**	**16**	**0.39**	50.2
	RLS_{fo}	**0.993**	**1.000**	**0.997**	**0.933**	**1.000**	**0.966**	28	22	0.36	287.6
	ILS_{fo}	**0.993**	**1.000**	**0.997**	**0.933**	**1.000**	**0.966**	28	22	0.36	140.3
	$TABU_{fo}$	**0.993**	**1.000**	**0.997**	**0.933**	**1.000**	**0.966**	28	22	0.36	129.7
	$SIMA_{fo}$	**0.993**	**1.000**	**0.997**	**0.933**	**1.000**	**0.966**	28	22	0.36	132.1
PRT9	FO	-	-	-	-	-	-	32	45	0.72	**53.1**
	HPO_{fo}	-	-	-	-	-	-	24	18	0.54	2799.5
	RLS_{fo}	**0.969**	0.999	**0.984**	**0.894**	0.893	0.893	23	21	**0.91**	301.5
	ILS_{fo}	-	-	-	-	-	-	34	26	0.15	263.2
	$TABU_{fo}$	-	-	-	-	-	-	36	30	0.14	185.8
	$SIMA_{fo}$	0.968	**1.000**	**0.984**	0.887	**0.956**	**0.920**	**20**	**17**	0.80	278.4
PRT10	FO	**0.990**	0.922	0.955	**0.961**	0.087	0.159	52	85	**0.64**	**0.2**
	HPO_{fo}	0.872	0.958	0.913	0.659	0.786	0.717	**35**	**28**	0.60	750.8
	RLS_{fo}	0.964	**0.965**	**0.965**	0.870	**0.813**	**0.840**	44	46	0.25	301.1
	ILS_{fo}	0.964	**0.965**	**0.965**	0.870	**0.813**	**0.840**	44	46	0.25	195.0
	$TABU_{fo}$	0.964	**0.965**	**0.965**	0.870	**0.813**	**0.840**	44	46	0.25	165.0
	$SIMA_{fo}$	0.965	0.963	0.964	0.874	0.809	**0.840**	44	47	0.25	161.2

Table 5: Comparative evaluation results for the proprietary logs - Fodina.

4.4. Inductive Miner

Table 6 displays the results of our comparative evaluation for Inductive Miner. We held out from the table the five BPIC15 logs, because none of the three metaheuristics could discover a model within the five minutes timeout. This was due to scalability issues experienced by the Markovian accuracy, already known for the case of IM [6].

In the remaining 15 logs, 13 times *all* the metaheuristics improved the Markovian F-score w.r.t. IM_d, and only for the $BPIC17_f$ log none of the metaheuristics could outperform IM_d. The best performing metaheuristic was $SIMA_{imd}$, eight times achieving the highest Markovian F-score, followed by $TABU_{imd}$ and RLS_{imd}, which scored seven, and respectively, six times the highest Markovian F-score. Again, we note that the differences in the achieved F-score across the four metaheuristics is small. There are several cases in which multiple metaheuristics achieve the same F-score, and a difference of only 1-2 percentage point between the best-performing and the worst-performing metaheuristics.

The results of the metaheuristics on the alignment-based F-score are similar to the case of Fodina, and they are broadly in-line with the results achieved on the Markovian F-score. Indeed, 80% of the times, *all* the metaheuristics were able to outperform IM_d, failing only in two logs out of 15.

Regarding the complexity of the models discovered by the metaheuristics, we recorded little variation w.r.t. the complexity of the models discovered by IM_d. Size and CFC did not notably improve nor worsen, except for the PRT9 and the $BPIC14_f$ logs, where both size and CFC were reduced by about 30%.

In terms of execution times, the three metaheuristics perform similarly, with an average execution time close to 300 seconds, meaning that the majority of the times the solution-space exploration was interrupted by the timeout.

4.5. Discussion

The results of the evaluation show that the use of metaheuristics optimization brings consistent improvements in accuracy with respect to the baseline discovery approaches, in 80% of the cases. Furthermore, it produces consistently higher alignment-based F-score, even though this measure was not used as an objective function, due to the low scalability of alignment-based precision.

The drawback of using metaheuristics optimization is the longer execution time – several minutes versus less than a few seconds for the baselines.

In a small number of cases the optimization framework did not yield any F-score improvement with respect to the corresponding unoptimized approach, due to: (i) a small solution-space (i.e. the baseline already discovers the best process model); or (ii) scalability issues (i.e. the Markovian accuracy could be computed within the timeout). While the former scenario is beyond our control and strictly relates to the complexity of the input event log, the latter reminds us of the limitations of the state-of-the-art accuracy measures (and especially precision) in the context of automated process discovery, and justifies our design choice of a modular optimization framework, that allows the use of new accuracy measures as objective functions in the future, which may be able to overcome such scalability issues.

Another remarkable finding is that the metaheuristically optimized versions of Split Miner and Fodina consistently outperform their hyper-parameter optimized counterparts. This means that the space of possible process models that can explored by

Event Log	Discovery Approach	Align. Acc.			Markov. Acc. ($k=5$)			Complexity			Exec.
		Fitness	**Prec.**	**F-score**	**Fitness**	**Prec.**	**F-score**	**Size**	**CFC**	**Struct.**	**Time(s)**
BPIC12	IM_d	**1.000**	0.168	0.287	**1.000**	<0.001	<0.001	**30**	28	1.00	**0.7**
	RLS_{imd}	0.661	0.763	0.708	0.220	0.163	0.187	40	21	1.00	300.0
	$TABU_{imd}$	0.661	0.763	0.708	0.220	0.163	0.187	40	21	1.00	309.4
	$SIMA_{imd}$	0.660	**0.805**	**0.725**	0.204	**0.223**	**0.213**	39	**19**	1.00	308.7
$BPIC13_{cp}$	IM_d	**1.000**	0.862	0.926	**0.999**	0.161	0.277	15	11	1.00	**0.4**
	RLS_{imd}	0.984	**0.889**	0.934	0.882	**0.424**	0.573	**9**	**5**	1.00	301.4
	$TABU_{imd}$	0.990	0.888	**0.936**	0.942	0.414	**0.575**	10	7	1.00	101.8
	$SIMA_{imd}$	0.984	**0.889**	0.934	0.882	**0.424**	0.573	**9**	**5**	1.00	300.5
$BPIC13_{inc}$	IM_d	**1.000**	0.673	0.805	**1.000**	0.109	0.197	**10**	9	1.00	**0.5**
	RLS_{imd}	0.895	**0.921**	**0.908**	0.679	**0.517**	**0.587**	**10**	**6**	1.00	301.5
	$TABU_{imd}$	0.895	**0.921**	**0.908**	0.679	**0.517**	**0.587**	**10**	**6**	1.00	71.9
	$SIMA_{imd}$	0.895	**0.921**	**0.908**	0.679	**0.517**	**0.587**	**10**	**6**	1.00	300.7
$BPIC14_f$	IM_d	0.861	**0.782**	**0.820**	0.507	**0.814**	0.625	27	16	1.00	**0.8**
	RLS_{imd}	**0.977**	0.676	0.799	**0.918**	0.447	0.601	**16**	**11**	1.00	302.5
	$TABU_{imd}$	0.949	0.673	0.788	0.859	0.505	**0.636**	17	13	1.00	303.3
	$SIMA_{imd}$	**0.977**	0.676	0.799	**0.918**	0.447	0.601	**16**	**11**	1.00	300.8
$BPIC17_f$	IM_d	**1.000**	0.679	**0.808**	**1.000**	**0.284**	**0.442**	34	23	1.00	**1.5**
	RLS_{imd}	0.674	0.815	0.738	0.241	0.214	0.227	**27**	**11**	1.00	302.7
	$TABU_{imd}$	0.693	**0.817**	0.750	0.262	0.204	0.230	28	13	1.00	83.8
	$SIMA_{imd}$	0.674	0.815	0.738	0.241	0.214	0.227	**27**	**11**	1.00	301.0
RTFMP	IM_d	**1.000**	0.543	0.704	**1.000**	0.003	0.005	**15**	12	1.00	**0.8**
	RLS_{imd}	0.938	0.886	0.911	0.784	0.379	0.511	21	14	1.00	321.1
	$TABU_{imd}$	0.938	0.886	0.911	0.784	0.379	0.511	21	14	1.00	52.1
	$SIMA_{imd}$	0.917	**0.907**	**0.912**	0.780	**0.625**	**0.694**	19	**9**	1.00	300.8
SEPSIS	IM_d	**1.000**	0.291	0.451	**0.918**	0.006	0.012	**24**	23	1.00	**0.4**
	RLS_{imd}	0.796	**0.684**	**0.736**	0.367	**0.363**	0.365	27	18	1.00	305.5
	$TABU_{imd}$	0.796	**0.684**	**0.736**	0.367	**0.363**	0.365	27	18	1.00	306.9
	$SIMA_{imd}$	0.813	0.581	0.678	0.482	0.310	**0.377**	25	**16**	1.00	301.6
PRT1	IM_d	**1.000**	0.748	0.856	**1.000**	0.025	0.048	**14**	11	1.00	**0.5**
	RLS_{imd}	0.974	**0.946**	**0.960**	0.692	**0.707**	**0.699**	16	**10**	1.00	304.4
	$TABU_{imd}$	0.971	**0.946**	0.958	0.692	**0.707**	**0.699**	17	**10**	1.00	304.0
	$SIMA_{imd}$	0.974	**0.946**	**0.960**	0.692	**0.707**	**0.699**	16	**10**	1.00	300.7
PRT2	IM_d	**1.000**	0.243	0.390	**1.000**	0.109	0.196	**13**	**11**	1.00	**0.9**
	RLS_{imd}	0.811	**0.464**	**0.591**	0.588	0.601	**0.594**	18	13	1.00	305.5
	$TABU_{imd}$	0.788	0.461	0.581	0.542	0.566	0.554	16	**11**	1.00	303.0
	$SIMA_{imd}$	0.792	0.413	0.543	0.524	**0.674**	0.590	18	13	1.00	307.3
PRT3	IM_d	0.827	0.890	0.857	0.328	0.253	0.286	**26**	**10**	1.00	**0.4**
	RLS_{imd}	0.914	0.896	0.905	0.501	**0.593**	0.543	**26**	14	1.00	305.1
	$TABU_{imd}$	**0.933**	**0.900**	**0.917**	**0.626**	0.592	**0.608**	28	15	1.00	302.6
	$SIMA_{imd}$	0.930	0.898	0.914	0.562	0.539	0.550	29	17	1.00	300.8
PRT4	IM_d	0.880	0.811	0.844	0.876	**0.967**	0.919	27	**13**	1.00	**0.5**
	RLS_{imd}	**0.962**	**0.879**	**0.919**	**1.000**	0.956	**0.977**	**19**	**13**	1.00	301.0
	$TABU_{imd}$	**0.962**	**0.879**	**0.919**	**1.000**	0.956	**0.977**	**19**	**13**	1.00	307.2
	$SIMA_{imd}$	**0.962**	**0.879**	**0.919**	**1.000**	0.956	**0.977**	**19**	**13**	1.00	300.8
PRT6	IM_d	0.917	**0.988**	0.951	0.524	0.350	0.420	18	**6**	1.00	**0.4**
	RLS_{imd}	**0.953**	0.987	**0.969**	**0.674**	**0.941**	**0.785**	**17**	7	1.00	304.2
	$TABU_{imd}$	0.905	0.915	0.910	0.488	0.903	0.634	18	10	1.00	643.0
	$SIMA_{imd}$	**0.953**	0.987	**0.969**	**0.674**	**0.941**	**0.785**	**17**	7	1.00	300.6
PRT7	IM_d	0.852	0.997	0.919	0.618	0.407	0.491	21	**5**	1.00	**0.4**
	RLS_{imd}	0.917	**1.000**	0.957	**0.700**	**1.000**	**0.824**	**20**	6	1.00	305.5
	$TABU_{imd}$	0.917	**1.000**	0.957	**0.700**	**1.000**	**0.824**	**20**	6	1.00	1013.9
	$SIMA_{imd}$	**0.960**	0.988	**0.974**	0.664	0.752	0.705	23	13	1.00	309.6
PRT9	IM_d	0.586	0.461	0.516	0.078	0.014	0.024	22	15	1.00	**2.6**
	RLS_{imd}	0.945	**1.000**	0.972	0.851	**1.000**	**0.919**	16	9	1.00	304.2
	$TABU_{imd}$	0.946	**1.000**	0.972	0.856	0.947	0.899	18	10	1.00	306.6
	$SIMA_{imd}$	**0.954**	**1.000**	**0.976**	**0.890**	0.909	0.899	**15**	**8**	1.00	300.0
PRT10	IM_d	0.530	0.656	0.586	0.386	0.000	0.001	36	28	1.00	**0.5**
	RLS_{imd}	0.859	**0.961**	0.907	0.664	0.691	0.677	**30**	**24**	1.00	300.3
	$TABU_{imd}$	**0.912**	0.907	**0.909**	**0.790**	0.484	0.600	**30**	**24**	1.00	33.1
	$SIMA_{imd}$	0.862	0.941	0.900	0.671	**0.719**	**0.694**	32	28	1.00	307.6

Table 6: Comparative evaluation results for the public and proprietary logs - Inductive Miner.

tweaking the parameters in input (e.g. the noise filter threshold) is not as rich as the space of process models that can be generated by repeatedly perturbing the DFG.

Finally, we found that all four metaheuristics considered in the evaluation led to similar F-scores. The differences in F-score between the best-performing and the worst-performing metaheuristics are generally negligible, in the order of 1-2 percentage points. For Inductive Miner, all four metaheuristics end up exploring the search space in a similar manner, leading to the same results in several cases. This may be explained by the fact that the set of possible models that Inductive Miner can generate is narrower than that generated by Fodina or Split Miner, because Inductive Miner can only generate block-structured models.

5. Conclusion

This paper showed that the use of S-metaheuristics is a promising approach to enhance the accuracy of DFG-based automated process discovery approaches. 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 metaheuristically optimized approaches consistently achieve higher accuracy than the corresponding unoptimized baselines (Split Miner, Fodina, and Inductive Miner - directly follows). This observation holds both when measuring accuracy via Markovian F-score and via alignment-based F-score. As expected, these accuracy gains come at the expense of higher execution times. This is natural given that the metaheustics needs to execute the baseline approach several hundred times and it needs to measure the accuracy of each of the resulting process models. The evaluation also showed that the choice of optimization metaheuristic (among those considered in the paper) does not have a substantial effect on the accuracy of the resulting process models, nor on their size or complexity.

In its current form, the framework focuses on improving F-score. In principle, the framework could also be used to optimize other objective functions, such as model complexity, measured by number of edges or control-flow complexity measures, for example. Related to the above, the framework could be extended to optimize multiple dimensions simultaneously, using multi-objective (Pareto-front) optimization techniques instead of single-objective ones. Along the same lines, it may also be possible to adapt the framework in order to optimize one measure (e.g. F-score), subject to one or more constraints on other measures (e.g. that the number of edges in the discovered model must be below a given threshold). Lastly, another opportunity for future work may be the automation of the best metaheuristic selection, without compromising the time performance of the framework.

Another limitation of the proposed approach is that the DFG perturbations employed in the optimization phase do not use the frequencies of the directly-follows relations (i.e. arc frequencies in the DFG are not used). In other words, the proposed approach makes the following two design choices: (i) the perturbations either add an arc or remove an arc but they do not alter the frequency of an arc; and (ii) the decision as to which arcs to add or remove is not taken based on arc frequencies. The rationale for the first of these design choices is that modifying the arc frequencies would require us to have a criterion for deciding by how much should be frequencies be altered. This criterion would have to be dependent on the way the underlying automated process

discovery algorithm uses the arc frequencies. We opted not to do in order ensure the optimization method is independent of the underlying base algorithm. The rationale for the second choice is that the perturbations should have an element of randomness in order to allow the metaheuristics to explore a wider subset of the search space. Poor perturbations are likely to lead to solutions with lower F-scores, which are eventually discarded by the metaheuristics, but a transformation that leads to solutions with lower F-scores may later give rise to other solutions with higher F-score as the search unfolds. This having been said, it is possible that perturbations that remove arcs based on arc frequency might help the heuristics to focus on areas of the search space with higher F-score. A direction for future work is to explore other perturbation heuristics including frequency-aware ones.

A third limitation of the framework is that it only considers four S-metaheuristics. There is room for investigating further metaheuristics such as variants of simulated annealing, e.g. using different cooling schedules. Along a similar direction, this study could be extended to investigate the trade-offs between S-metaheuristics and P-metaheuristics in this setting.

Finally, the evaluation put into evidence scalability limitations of the Markovian precision measure for some datasets. These limitations are not specific to this precision measure – they also apply, and sometimes to a larger extent, to other precision measures including ETC precision and entropy-based precision [26]. There is a need for more scalable measures of precision in order to make metaheuristic optimization more broadly applicable in the context of automated process discovery.

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 European Research Council (PIX Project).

References

[1] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. van Dongen, and W. van der Aalst. Measuring precision of modeled behavior. *Information Systems and e-Business*, 13(1), 2015.

[2] A. Adriansyah, B. van Dongen, and W. van der Aalst. Conformance checking using cost-based fitness analysis. In *International Conference on Enterprise Computing (EDOC)*. IEEE, 2011.

[3] S. Alizadeh and A. Norani. Icma: a new efficient algorithm for process model discovery. *Applied Intelligence*, 48(11), 2018.

[4] Esmaeil Atashpaz-Gargari and Caro Lucas. Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 4661–4667. IEEE, 2007.

[5] A. Augusto, A. Armas Cervantes, R. Conforti, M. Dumas, and M. La Rosa. Measuring fitness and precision of automatically discovered process models: A principled and scalable approach. *IEEE Trans. Knowl. Data Eng.*, 2020. To appear, `https://doi.org/10.1109/TKDE.2020.3003258`.

[6] 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 *International Conference on Business Process Management (BPM)*. Springer, 2018.

[7] 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. *Data Knowl. Eng.*, 2017.

[8] 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 Trans. Knowl. Data Eng.*, 31(4), 2019.

[9] 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. *Know. Inf. Syst.*, 2018.

[10] Adriano Augusto, Marlon Dumas, and Marcello La Rosa. Metaheuristic optimization for automated business process discovery. In *International Conference on Business Process Management (BPM)*. Springer, 2019.

[11] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

[12] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Inf. Sci.*, 237:82–117, 2013.

[13] J. Buijs, B. van Dongen, and W. van der Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In *International Conference on Cooperative Information Systems (CoopIS)*. Springer, 2012.

[14] A. Burattin and A. Sperduti. Automatic determination of parameters' values for heuristics miner++. In *IEEE Congress on Evolutionary Computation*, 2010.

[15] 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.

[16] A. K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, 2006.

[17] Yutika Amelia Effendi and Riyanarto Sarno. Discovering optimized process model using rule discovery hybrid particle swarm optimization. In *2017 3rd International Conference on Science in Information Technology (ICSITech)*, pages 97–103. IEEE, 2017.

[18] D. Gao and Q. Liu. An improved simulated annealing algorithm for process mining. In *International Conference on Computer Supported Collaborative Work in Design (CSCWD)*. IEEE, 2009.

[19] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5), 1986.

[20] John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[21] James Kennedy. Particle swarm optimization. In *Encyclopedia of machine learning*, pages 760–766. Springer, 2011.

[22] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[23] H.W. Kuhn. The hungarian method for the assignment problem. *NRL*, 2(1-2), 1955.

[24] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Scalable process discovery and conformance checking. *Software and Systems Modeling*, 17(2):599–631, 2018.

[25] Afina Lina Nurlaili and Riyanarto Sarno. A combination of the evolutionary tree miner and simulated annealing. In *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pages 1–5. IEEE, 2017.

[26] Artem Polyvyanyy, Andreas Solti, Matthias Weidlich, Claudio Di Ciccio, and Jan Mendling. Monotone precision and recall measures for comparing executions and specifications of dynamic systems. *ACM Trans. Softw. Eng. Methodol.*, 29(3):17:1–17:41, 2020.

[27] Joel Ribeiro and Josep Carmona. A method for assessing parameter impact on control-flow discovery algorithms. *Trans. Petri Nets Other Model. Concurr.*, 11:181–202, 2016.

[28] Sergey Smirnov, Matthias Weidlich, and Jan Mendling. Business process model abstraction based on behavioral profiles. In *International Conference on Service-Oriented Computing*, pages 1–16. Springer, 2010.

[29] W. Song, S. Liu, and Q. Liu. Business process mining based on simulated annealing. In *International Conference for Young Computer Scientists (ICYCS)*. IEEE, 2008.

[30] T. Stützle. *Local search algorithms for combinatorial problems*. PhD thesis, Darmstadt University of Technology, 1998.

[31] Suriadi Suriadi, Robert Andrews, Arthur H. M. ter Hofstede, and Moe Thandar Wynn. Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. *Inf. Syst.*, 64:132–150, 2017.

[32] A.F. Syring, N. Tax, and W.M.P. van der Aalst. Evaluating conformance measures in process mining using conformance propositions. In *Transactions on Petri Nets and Other Models of Concurrency XIV*, pages 192–221. Springer, 2019.

[33] W. van der Aalst. *Process Mining - Data Science in Action*. Springer, 2016.

[34] Seppe K. L. M. vanden Broucke and Jochen De Weerdt. Fodina: A robust and flexible heuristic process discovery technique. *Decis. Support Syst.*, 100:109–118, 2017.

[35] A. Weijters and J. Ribeiro. Flexible heuristics miner (FHM). In *International Conference on Computational Intelligence and Data Mining (CIDM)*. IEEE, 2011.

[36] Lijie Wen, Wil MP Van Der Aalst, Jianmin Wang, and Jiaguang Sun. Mining process models with non-free-choice constructs. *Data Mining and Knowledge Discovery*, 15(2):145–180, 2007.