Multi-Objective Metaheuristics for Effective and Efficient Stochastic Process Discovery

Hootan Zhian[1], Rajkumar Buyya[1], and Artem Polyvyanyy[1]

[1]The University of Melbourne, Victoria 3010, Australia
hzhian@student.unimelb.edu.au
{rbuyya;artem.polyvyanyy}@unimelb.edu.au

Abstract. Process discovery studies algorithms that, given an event log of a system captured as a collection of recorded sequences of actions executed by the system, construct process models that describe the system. A process discovery problem is a multi-objective optimization problem that aims to simultaneously optimize the simplicity of the constructed models and their quality, ensuring models accurately represent both the input event log and the underlying system. Multi-objective metaheuristics provide effective strategies for navigating these trade-offs, offering practical approaches to exploring Pareto-optimal solutions. Recent research has demonstrated that genetic strategies based on grammatical inference can produce superior models compared to state-of-the-art discovery algorithms. In this paper, we conduct a comprehensive evaluation of existing multi-objective metaheuristics for process discovery using grammatical inference, refining them where necessary to enhance efficiency and control the number of the discovered Pareto-optimal models. This evaluation, based on multiple real-life event logs and our open-source implementation of various metaheuristic algorithms, confirms the feasibility of efficiently discovering significantly better models. Specifically, the *Differential Evolution*-based optimization approach, which we refer to as *ADESPD*, consistently produces high-quality models with diverse characteristics within practical time constraints, which are deterministic and sound.

Keywords: Stochastic process discovery, directly-follows graphs, metaheuristics

1 Introduction

Process discovery is a core problem in process mining that focuses on constructing process models from event logs recorded by an information system. These models aim to accurately and concisely represent the business process that generated the event log [5]. An event log consists of multiple traces, where each trace is a sequence of actions executed in a specific case, or instance, of the business process. The discovered models help in understanding the current process and provide a ground for its improvement.

Numerous process discovery algorithms have been proposed [2]. These algorithms construct process models using various modeling languages, such as Directly-Follows Graphs (DFGs), Petri nets, and Business Process Model and Notation. DFGs are widely used in commercial tools due to their intuitive interpretation and efficient construction [3, 25]. In a DFG, nodes represent actions, and edges indicate that the target action can directly follow the source action in some execution of the process. Nodes and

edges are annotated with numerical values that reflect, respectively, the frequency of corresponding actions and subsequent, non-interrupted by other actions, occurrences of actions in the traces of the input event log. A DFG has two special nodes: an input node, which has no incoming edges, and an output node, which has no outgoing edges. A sequence of actions on a walk from the input node to the output node represents a possible execution (a trace) of the process [21]. Figures 2(a) and 2(c) show two DFGs.

We demonstrated that process discovery based on the genetic optimization of the *Alergia* grammatical inference algorithm—called *GASPD*—can construct interesting process models in terms of size and accuracy of reflecting the likelihood of traces generated by the system. Moreover, we showed that models allowing multiple nodes to represent the same action can outperform those restricting each action to a single node. As DFGs are grounded in the "can follow" relation over actions in the event log, that is, pairs of actions that can appear one after another in some trace of the event log, in a DFG, each action is represented by exactly one node; the pairs of actions specify the edges in the DFGs. To distinguish from DFGs, graphs that allow multiple nodes to refer to the same action are called Stochastic Directed Action Graphs (SDAGs). Unlike DFGs, SDAG edges are annotated with probabilities rather than frequencies, indicating the likelihood of performing the target action after completing the source action. Given the total number of traces, these edge annotations are sufficient to reconstruct the observed traces and the corresponding action counts from which the probabilities were estimated [6]. Figure 2(b) shows an example SDAG.

Due to its genetic nature, *GASPD* has significant runtime requirements. Moreover, the interesting models discovered by *GASPD* tend to be confined to a limited range of model sizes [6]. In this paper, we conduct a comprehensive review of multi-objective metaheuristics to identify those that efficiently support the discovery of models—based on the *Alergia* grammatical inference algorithm—that are simple (in terms of *size*), accurate (in terms of *entropic relevance*), and diverse. Entropic relevance is chosen as the accuracy measure because it assesses how well a model reflects the likelihood of target traces, balances precision and recall, and is computationally efficient, that is, is computable in time linear to the size of the input event log [7]. To ensure the discovered models exhibit a wide range of quality characteristics, we incorporate a niching technique, which allows metaheuristics to explore multiple promising search subspaces simultaneously, reducing the risk of premature convergence to local optima [14, 26].

Specifically, this paper makes the following contributions:

- A classification of multi-objective metaheuristics suitable for process discovery using the *Alergia* grammatical inference algorithm, based on their strategies for selecting candidate solutions.
- Integration of a niching technique to maintain solution diversity within the proposed classes of multi-objective optimization algorithms; for some classes—and consequently, some metaheuristics—this niching technique is applied for the first time.
- An empirical evaluation using industrial event logs, demonstrating that *Differential Evolution*-based optimization of *Alergia* grammatical inference discovers smaller, more accurate models with diverse quality characteristics, outperforming existing methods within practical time limits; we refer to this methodology as *Alergia* and

Differential Evolution for Stochastic Process Discovery (*ADESPD*). In addition, all the models discovered by *ADESPD* are *deterministic* and *sound* by construction.

The next section presents a motivating example and highlights the challenges of process discovery. Section 3 introduces fundamental concepts essential for understanding the subsequent discussions. Section 4 reviews existing metaheuristics, proposes their taxonomy, and discusses integration of niching. Section 5 details the evaluation setup and analyzes the results. Finally, Section 7 presents concluding remarks.

2 Motivating Example

Figures 2(a) and 2(c) show two DFGs discovered using Directly Follows visual Miner (*DFvM*) [21] from the Sepsis Cases event log[1], each generated with a different trace filtering threshold. In these DFGs, nodes labeled "▶" represent inputs and nodes labeled "■" represent outputs. Both DFGs describe, among other walks, a path that starts at the input node, proceeds to node A, then visits node B, and subsequently traverses node F before reaching the output node. This corresponds to the trace ABF, which represents an execution that begins with action A, continues with B, and concludes with action F.

In the DFG in Fig. 2(a), for instance, the edge from the input node to node A is annotated with a frequency of 1,050, indicating that all the 1,050 traces this DFG describes begin with action A. The edges from A to B and from B to F indicate that 1,050 occurrences of action A are followed by occurrences of action B and 1,050 occurrences of action B are followed by occurrences of action F. Finally, 454 occurrences of action F are not followed by any actions, refer to the edge from node F to the output node, while the remaining 596 occurrences of action F are followed by occurrences of D or E. Note that node frequencies are not annotated, as they can be trivially derived by summing the frequencies of all incoming or outgoing edges.

Figure 2(b) presents an SDAG discovered from the same Sepsis Cases event log using *ADESPD*, the approach introduced in this paper. The input and output nodes of the SDAG share the same annotations as those of the DFGs. The edge frequencies in the SDAG are derived from the annotated probabilities based on a total of 1,050 traces. We use 1,050 as the target frequency to account for all traces in the input event log. Notably, we do not specify node frequencies, as again, they can be trivially inferred from the edge frequencies. The SDAG indicates that the probability of the trace ABF is the product of probabilities along the corresponding path: $1.0 \times 1.0 \times 1.0 \times 0.43 = 0.43$. A DFG can be interpreted as an SDAG by estimating edge probabilities based on frequencies, where probability of an edge is equal to the ratio of its frequency to the sum of frequencies of all outgoing edges from the same source node. The inverse is not necessarily true.

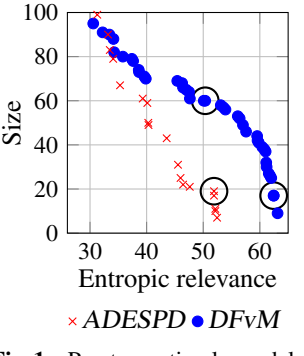

Fig. 1. Pareto optimal models discovered from the Sepsis Cases event log.

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

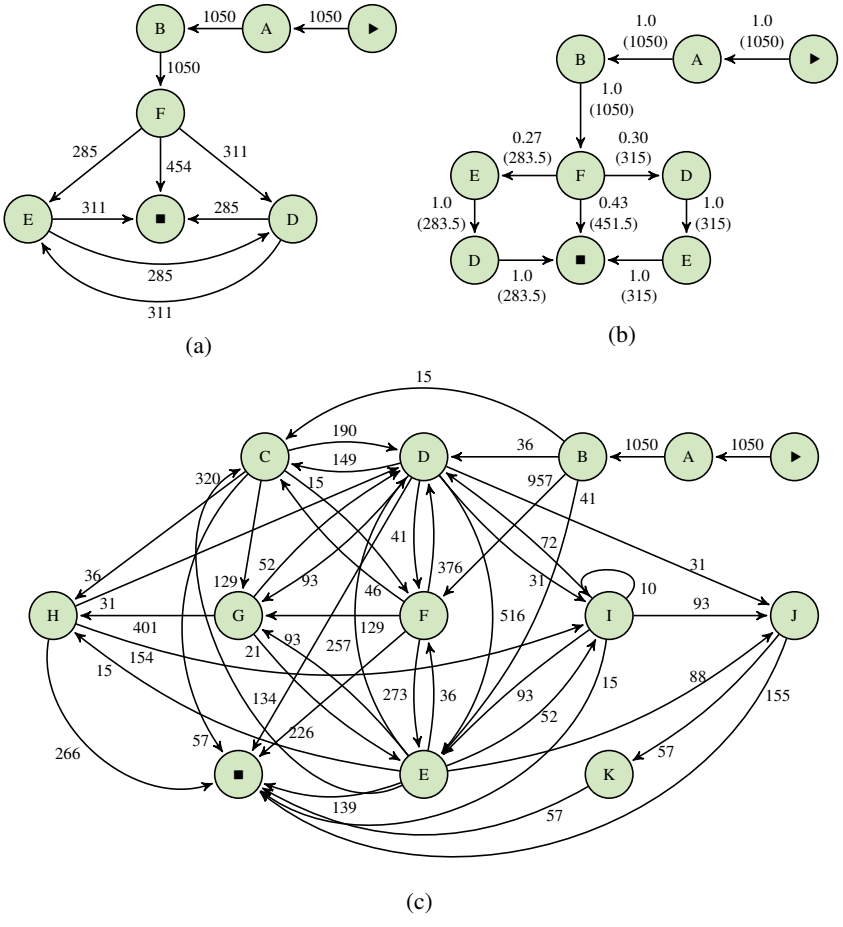

Fig. 2. Two DFGs (a) and (c) and SDAG (b) discovered from the Sepsis Cases event log; node labels represent the following actions: A = "ER Registration", B = "ER Triage", C = "LacticAcid", D = "Leucocytes", E = "CRP", F = "ER Sepsis Triage", G = "IV Liquid", H = "IV Antibiotics", I = "Admission NC", J = "Release A", and K = "Return ER".

The DFG in Fig. 2(a) and the SDAG in Fig. 2(b) are similar in size; 17 and 19, respectively. The SDAG exhibits better entropic relevance of 51.9, compared to 62.4 for the DFG; a lower entropic relevance value indicates a more faithful representation of the relevant traces and their likelihood. Both models belong to the set of Pareto-optimal solutions discovered by *DFvM* and *ADESPD*. The Pareto-optimal model discovered by *DFvM* with entropic relevance compared to the SDAG has a significantly larger size of 60 and is shown in Fig. 2(c). These three models are highlighted with circles in Fig. 1, where they are presented alongside other optimal models of practical sizes identified by the two techniques. To obtain the optimal *DFvM* models, we generated 100 DFGs by varying the trace filtering threshold from zero to one in increments of 0.01.

3 Preliminaries

This section introduces basic concepts used in this paper. An SDAG represents a collection of traces along with their likelihood of occurrence during model execution.

Definition 3.1 (SDAGs [6]) A *stochastic directed action graph* (SDAG) is a tuple $(N, \Lambda, \beta, \gamma, q, \blacktriangleright, \blacksquare)$, where N is a finite set of *nodes*, Λ is a finite set of *actions*, $\beta : N \to \Lambda$ is a *labeling function*, $\gamma \subseteq (N \times N) \cup (\{\blacktriangleright\} \times N) \cup (N \times \{\blacksquare\})$ is the *flow relation*, $q : \gamma \to [0, 1]$ is a *flow probability function*, $\blacktriangleright \notin N$ is the *input* node, and $\blacksquare \notin N$ is the *output* node, such that $\forall\, n \in N \cup \{\blacktriangleright\} : (\sum_{m \in \{k \in N \cup \{\blacksquare\} \mid (n,k) \in \gamma\}} q(n, m) = 1)$. ⌟

An *execution* of an SDAG is a finite sequence of its nodes, beginning with $\blacktriangleright$ and ending with $\blacksquare$, such that for every pair of consecutive nodes u and v in the sequence, it holds that $(u, v) \in \gamma$ For example, the sequence $\langle \blacktriangleright, \text{A}, \text{B}, \text{F}, \text{D}, \text{E}, \blacksquare \rangle$ is an execution of the SDAG shown in Fig. 2(b). A *trace* of an SDAG is a sequence of actions such that there exists an execution that *confirms* the trace, i.e., the execution visits nodes (excluding $\blacktriangleright$ and $\blacksquare$) in the same order as they appear in the trace. For instance, the sequence $\langle$A, B, F, D, E$\rangle$ (or ABFDE, for short) is a trace confirmed by the example execution above. The probability of observing a trace corresponds to the product of the probabilities on the edges traversed by the confirming execution. According to the SDAG in Fig. 2(b), the probability of observing the trace ABFDE is 0.3

An SDAG is *sound* if every of its node is on a directed walk from the input node to the output node. It is *deterministic* if, for each node, the outgoing edges lead to distinct actions. Every trace of a deterministic SDAG is confirmed by exactly one execution.

Definition 3.2 (Deterministic SDAGs [6]) An SDAG $(N, \Lambda, \beta, \gamma, q, \blacktriangleright, \blacksquare)$ is *deterministic* if and only if for every pair of edges originating from the same node and leading to distinct target nodes it holds that the labels of those target nodes are different; that is: $\forall\, n \in N \cup \{\blacktriangleright\}\ \forall\, n_1, n_2 \in N : (((n, n_1) \in \gamma \wedge (n, n_2) \in \gamma \wedge n_1 \neq n_2) \Rightarrow \beta(n_1) \neq \beta(n_2))$. ⌟

Entropic relevance[7] applies the minimum description length principle to quantify the number of bits required to encode a trace from an event log using a process model that specifies the likelihood of traces, such as an SDAG. Models yielding lower entropic relevance values for a given event log are preferred, as they represent the traces and their likelihoods more effectively. Entropic relevance incorporates a background encoding mechanism to account for traces in the event log that are not captured by the model. In this work, we adopt the *zero-order* background coding mechanism, which strikes a balance between a strict and lenient punishment of non-modeled traces [7].

GASPD is a genetic algorithm for constructing SDAGs from an event log [6]. It evolves an initial population of randomly generated models through iterative selection, crossover, and mutation operations. Selection favors models that achieve a good balance between structural simplicity and quality in terms of entropic relevance. Each model is characterized by three parameters: a filtering threshold that specifies the proportion of frequent traces retained from the event log, and two parameters associated with the *Alergia* [13] stochastic grammar inference algorithm. *Alergia* learns a probabilistic grammar from the filtered traces, which is then translated into an SDAG. The translation process guarantees that the resulting SDAG is both deterministic and sound.

4 Stochastic Process Discovery as Multi-Objective Optimization

A *multi-objective optimization problem* is defined as follows: Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a vector-valued objective function, where $f(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x}))$ and each $f_i : \mathbb{R}^n \to \mathbb{R}$, $i \in [1 \,..\, m]$, represents an objective to be minimized. The goal is to find a solution $\mathbf{X}^* \in \mathbb{R}^n$ such that: (i) $\mathbf{X}^*$ is feasible, meaning it satisfies all the constraints of the problem, and (ii) $\mathbf{X}^*$ is Pareto optimal, meaning there is no other feasible solution $\mathbf{y} \in \mathbb{R}^n$ such that $f_i(\mathbf{y}) \leq f_i(\mathbf{X}^*)$ for all $i \in [1 \,..\, m]$, and $f_j(\mathbf{y}) < f_j(\mathbf{X}^*)$ for at least one $j \in [1 \,..\, m]$. Thus, $\mathbf{X}^*$ is a feasible solution for which no other feasible solution improves at least one objective without worsening at least one other objective.

To construct SDAGs, *GASPD* formulates and solves a multi-objective optimization problem that seeks parameter triplets—comprising the event log filtering threshold and two *Alergia* parameters—that yield Pareto optimal models with respect to size and entropic relevance. *GASPD* employs a genetic strategy to explore the space of parameter configurations, where each triplet induces an SDAG. The goal is to identify configurations that produce compact models that accurately reflect the behavior of the system that generated the input event log. Given the demonstrated success of this approach [6], it is worth exploring whether alternative metaheuristics could further improve the effectiveness and efficiency of convergence toward high-quality SDAGs inferred using the *Alergia* grammatical inference algorithm.

In Section 4.1, we review existing multi-objective metaheuristics. Then, in Section 4.2, we propose a classification of these metaheuristics and integrate the resulting classes with a niching technique to support the discovery of diverse models. We also outline an algorithm for metaheuristic-driven SDAG discovery.

4.1 Metaheuristics

Metaheuristic algorithms are widely recognized for their effectiveness in solving complex optimization problems across diverse domains. We examine nine prominent metaheuristic strategies [16], with particular emphasis on their solution update mechanisms used to guide the search. Notably, comparative studies indicate that Differential Evolution, Particle Swarm Optimization, Genetic Algorithms, and Symbiotic Organisms Search consistently outperform other strategies in terms of success rate [16].

Particle Swarm Optimization (PSO) [20] PSO is inspired by the collective movement observed in bird flocking and fish schooling. In PSO, individual particles traverse the search space by updating their positions based on their personal best-known position and the swarm's global best-known position. The position update is defined as:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1)\,, \tag{1}$$

where $\mathbf{x}_i(t)$ is the current position of particle i and $\mathbf{v}_i(t+1)$ is its velocity at the next time step influenced by the personal best position $\mathbf{p}_i$ and the global best position $\mathbf{X}^*$:

$$\mathbf{v}_i(t+1) = w \cdot \mathbf{v}_i(t) + c_1 \cdot \mathrm{rand}_1() \cdot (\mathbf{p}_i - \mathbf{x}_i(t)) + c_2 \cdot \mathrm{rand}_2() \cdot (\mathbf{X}^* - \mathbf{x}_i(t))\,. \tag{2}$$

Here, w is the inertia weight controlling momentum, c_1 and c_2 are acceleration coefficients, and $\text{rand}_1()$ and $\text{rand}_2()$ are random numbers in $[0, 1]$ responsible for stochasticity that encourages exploration. This formulation balances exploration and exploitation, enabling particles to search the solution space effectively.

Whale Optimization (WO) [23] WO simulates the hunting behavior of humpback whales, specifically their bubble-net feeding strategy, through a spiral-based position update mechanism. The algorithm balances exploration and exploitation by alternating between two strategies: encircling the prey and spiral movement toward it. The position update of search agent (representing a whale) i is defined as:

$$\mathbf{x}_i(t+1) = \begin{cases} \mathbf{X}^* - \mathbf{A} \cdot |\mathbf{C} \cdot \mathbf{X}^* - \mathbf{x}_i(t)| & \text{if } p < 0.5 \\ \mathbf{D} \cdot e^{bl} \cdot \cos(2\pi l) + \mathbf{X}^* & \text{if } p \geq 0.5 \end{cases}, \tag{3}$$

where $\mathbf{X}^*$ denotes the best solution found so far, and p is a random number in $[0, 1]$ that determines the movement strategy. The coefficient vectors $\mathbf{A}$ and $\mathbf{C}$ control the balance between exploration and exploitation, while $\mathbf{D}$ is the distance between the current position and the best solution. The constant b controls the spiral's shape, and l is a random number that introduces stochastic variation into the spiral motion.

Cuckoo Search (CS) [31] CS is inspired by the brood parasitism behavior of cuckoo species. The algorithm employs *Lévy flights* to generate new candidate solutions, mimicking the unpredictable and long-range movements observed in nature. By combining local exploitation and global exploration, CS effectively navigates the search space. The position update of search agent (representing a cuckoo) i is defined as:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \alpha \oplus \text{Lévy}(\lambda), \tag{4}$$

where $\alpha > 0$ is a scaling factor that controls the step size, $\oplus$ denotes element-wise multiplication, and $\text{Lévy}(\lambda)$ represents a Lévy flight—a random walk whose steps follow a heavy-tailed distribution characterized by the exponent λ, which governs the magnitude and frequency of jumps.

Genetic Algorithm (GA) [29] Genetic Algorithm (GA) is inspired by the principles of natural selection and genetics. It evolves a population of candidate solutions by applying three genetic operators: selection, crossover, and mutation. Through these operations, GA iteratively improves solution quality across generations. The update of a solution (individual) i at generation $t + 1$ can be expressed as:

$$\mathbf{x}_i(t+1) = \text{Crossover}(\mathbf{x}_i(t), \mathbf{x}_j(t)) + \text{Mutation}(\mathbf{x}_i(t)), \tag{5}$$

where $\mathbf{x}_j(t)$ is a selected mate from the current population, Crossover combines genetic material from two parents to generate offspring, and Mutation introduces small random changes to the input individual to maintain diversity and avoid premature convergence.

Differential Evolution (DE) [27] DE is a population-based optimization algorithm that evolves candidate solutions through vector-based recombination. It uses differences between vectors to create variations in the population. This approach is particularly effective for continuous optimization problems. The mutation step in DE is defined as:

$$\mathbf{u}_i(t+1) = \mathbf{x}_{r1}(t) + F \cdot (\mathbf{x}_{r2}(t) - \mathbf{x}_{r3}(t)), \tag{6}$$

where $\mathbf{u}_i$ is the donor vector for individual i, $\mathbf{x}_{r1}$, $\mathbf{x}_{r2}$, and $\mathbf{x}_{r3}$ are distinct individuals randomly selected from the current population (but not $\mathbf{x}_i$), and F is a scaling factor.

Firefly Algorithm (FA) [30] FA is inspired by the flashing behavior of fireflies. In FA, each firefly represents a candidate solution, and the objective function value determines its brightness. The algorithm assumes that fireflies are attracted to brighter individuals, and this attraction decreases with distance due to light absorption. The position update of search agent i (a firefly), when attracted to a brighter agent j, is defined as:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \beta_0 \cdot e^{-\gamma r_{ij}^2} \cdot (\mathbf{x}_j(t) - \mathbf{x}_i(t)) + \alpha \cdot (\text{rand} - \tfrac{1}{2}), \tag{7}$$

where β_0 is the attractiveness at zero distance, γ is the light absorption coefficient controlling how attractiveness decreases with distance, r_{ij} is the Euclidean distance between fireflies i and j, α is a randomization parameter, and rand is a random number introducing stochasticity.

Gravitational Search Algorithm (GSA) [1] GSA is inspired by the law of gravity and mass interactions. It simulates a system in which search agents are treated as objects whose performance determines their masses. Heavier masses represent better solutions and exert stronger attractive forces on others. The movement of agents is governed by gravitational attraction, guiding the search toward optimal solutions. The position update rule in GSA is defined as:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1), \tag{8}$$

where the velocity is updated as $\mathbf{v}_i(t+1) = \text{rand}_i \cdot \mathbf{v}_i(t) + \mathbf{a}_i(t)$, with rand_i being a random number (fresh for each iteration), and $\mathbf{a}_i(t)$ denoting the acceleration of object i, computed based on the cumulative gravitational forces exerted by the other objects.

Ant Colony Optimization (ACO) [15] ACO is inspired by the foraging behavior of ants, which communicate via pheromone trails to find efficient paths between their colony and food sources. In ACO, artificial ants construct solutions by probabilistically selecting paths, where the probability is influenced by pheromone intensity and heuristic information. The probability that search agent i (representing an ant) moves from position j to position k, given as nodes in a search graph, is defined as:

$$p_{jk}^i = \frac{(\tau_{jk})^\alpha \cdot (\eta_{jk})^\beta}{\sum_{l \in N_j^i} (\tau_{jl})^\alpha \cdot (\eta_{jl})^\beta}, \tag{9}$$

where τ_{jk} is the pheromone level on edge (j,k), η_{jk} is the heuristic desirability of edge (j,k), e.g., inverse of distance, α controls the influence of pheromone, β controls the influence of the heuristic, and N_j^i is the set of allowed next nodes. After all ants construct their solutions, the pheromone levels τ_{jk} are updated to $(1-\rho) \cdot \tau_{jk} + \Delta\tau_{jk}$, where ρ is the evaporation rate of the pheromone and $\Delta\tau_{jk}$ is the amount of pheromone deposited.

Symbiotic Organisms Search (SOS) [17] SOS mimics symbiotic relationships observed in nature through three distinct phases: mutualism (both organisms benefit), commensalism (one benefits, the other is unaffected), and parasitism (one benefits at the expense of another). It updates search agents (organisms) by simulating these interactions, promoting exploration of the search space and reducing the risk of premature convergence. It is a parameter-free algorithm, which simplifies its implementation

across a range of optimization problems. The position update rules for each interaction phase are as follows:

$$\text{Mutualism:} \quad \mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \text{rand}(0,1) \cdot (\mathbf{X}^* - \mathbf{x}_i(t)) + \text{BF}_1 \cdot (\mathbf{x}_j(t) - \mathbf{x}_i(t)), \tag{10}$$

$$\text{Commensalism:} \quad \mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \text{rand}(-1,1) \cdot (\mathbf{X}^* - \mathbf{x}_j(t)), \tag{11}$$

$$\text{Parasitism:} \quad \text{if } f(\mathbf{x}_i(t)) > f(\mathbf{x}_j(t)) \text{ then } \mathbf{x}_j(t+1) = \mathbf{x}_i(t), \tag{12}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are two randomly selected organisms, $\mathbf{X}^*$ is the current best solution, BF_1 is a randomly chosen benefit factor, $\text{rand}(a,b)$ generates a random number uniformly in the range $[a,b]$, and $f(\cdot)$ denotes the fitness function.

4.2 Niching, Classification, and Discovery

A key challenge in multi-objective optimization tasks is maintaining diversity of solutions to avoid premature convergence to suboptimal solutions. Niching techniques address this by promoting the exploration of multiple optima, supporting identification of multiple high-quality solutions in a single run. One fundamental niching technique is based on sharing functions, which modify fitness values to reduce the dominance of similar solutions. This prevents overcrowding in highly competitive regions of the search space and encourages the discovery of alternative solutions.

The *sharing function* $S(d)$ modifies fitness to maintain population diversity:

$$S(d) = \begin{cases} 1 - \left(\frac{d}{\sigma_{share}}\right)^{\alpha} & \text{if } d < \sigma_{share} \\ 0 & \text{otherwise,} \end{cases} \tag{13}$$

where d is the distance between two solutions in the objective space, σ_{share} is the sharing radius, and α controls the rate of modification. By reducing the fitness of similar individuals, this mechanism discourages clustering and promotes broader exploration. The distance d_{ij} between solutions i and j is computed as $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$. Then, the shared fitness $\hat{f}_i$ of a solution i is computed as follows:

$$\hat{f}_i = \frac{f_i}{\sum_{i \neq j} S(d_{ij})}, \tag{14}$$

where f_i is the raw fitness of solution i. This adjustment ensures that solutions in densely populated regions receive lower fitness values, thereby maintaining population diversity and allowing the algorithm to explore multiple regions of the objective space effectively.

To enable the seamless integration of niching techniques with multi-objective metaheuristics, we propose a classification based on the solution update strategies. Specifically, we identify four classes of such strategies, as detailed below.

Leader-Based Update. In this class of metaheuristics, all solutions are updated based on the leader, the current best option. Specifically, the update of a solution s is influenced by the leader L selected from the Pareto front of solutions F:

$$L = \arg\max_{s \in F} \hat{f}_s. \tag{15}$$

Algorithm 1: Stochastic Process Discovery

Data: Event log L, population size n, maximum number of iterations *iter*, and metaheuristic algorithm *meta*
Result: A Pareto frontier F of optimal discovered SDAGs

```
1  F ← ∅;
2  population ← InitializePopulation(n);
3  for i = 1; i ≤ iter; iter = iter + 1 do
4      for p in population do
5          p.model ← ConstructSDAG(p, L);
6          p.sharedfitness ← ComputeSharedFitness(p, F);
7          F ← UpdateParetoFront(p, F, L);
8          if F has changed then
9              for x ∈ F do
10                 x.sharedfitness ← EvaluateSharedFitness(x, F);
11     population ← UpdatePopulation(population, F, meta);
12 return F
```

The class of metaheuristics with leader-based updates includes PSO and WO.

Selection-Based Update. In this class of optimization strategies, the next generation of solutions is created from solutions from the current solutions [10]. The probability P_s of selecting solution s to generate new solutions is given by:

$$P_s = \frac{\hat{f}_s}{\sum_{k=1}^{N} \hat{f}_k}, \tag{16}$$

where N is the total number of current solutions.

GA, DE, and CS are members of this class of metaheuristics.

Collaborative Influence Update. The solution update rule of metaheuristics from this class is influenced by all solutions in the Pareto front of solutions.

ACO, GAS, and FA are members of this class of techniques.

Hybrid Update. Some algorithms combine multiple update strategies, leveraging the strengths of different approaches to enhance performance. The SOS strategy falls into this class, as it incorporates a leader-based update during the commensalism phase and a collaborative influence update during the mutualism phase.

Algorithm 1 summarizes the high-level strategy for discovering optimal SDAGs using a metaheuristic with niching based on the concept of shared fitness. The algorithm starts by initializing an empty set of optimal solutions F (line 1). Next, the initial population of n solutions, each given as a triplet of parameters sufficient to induce an SDAG (event log filtering and two parameters of the *Alergia* grammatical inference algorithm), is generated at line 2. Subsequently, the `for` loop of line 3 executes *iter* number of times. In each iteration, for each parameter triplet in the current population of solutions, we construct the corresponding SDAG *p.model* (line 5), compute the shared fitness of *p.model* using Eq. (14) (line 6), and update the Pareto front of solutions using model p (line 7). Moreover, if model p changes the Pareto front (line 8), all the shared fitness values of the models on the Pareto front get updated in the `for` loop of line 9. Finally, the current population of models gets updated using the model on the Pareto front F, the current population of models, and the metaheuristic rules *meta* (line 11). After the population

of models and, thus, the Pareto front of optimal models gets updated *iter* times, the resulting optimal models are returned on line 12.

5 Evaluation

We implemented all nine metaheuristics reviewed in Section 4.1, along with the *GASPD* and *DFvM* discovery algorithms.[2] These algorithms were evaluated using twelve real-world event logs from the IEEE Task Force on Process Mining (https://www.tf-pm.org/resources/logs). The selected event logs exhibit diverse characteristics, such as varying numbers of actions and traces, different trace lengths, and differing maximum model sizes discovered by *DFvM* [21]. This diversity enables a comprehensive assessment of the algorithms' robustness across various scenarios. Table 1 summarizes the characteristics of the twelve event logs. All experiments were conducted on a system equipped with a 13th Gen Intel® Core™ i7-1355U processor (1.70GHz) and 16.0GB of RAM.

Table 1. Characteristics of the event logs used in our evaluation.

Event log	Actions	Traces				Max. *DFvM* size
		Total	Distinct	Max. len.	Avg. len.	
Sepsis Cases	16	1,050	846	185	14.48	155
BPIC 2012[3]	24	13,087	4,366	175	20.03	176
BPIC 2013 Incidents[4]	4	7,554	1,511	123	8.67	22
Hospital Billing[5]	18	100,000	1,020	217	4.51	177
Request For Payment[6]	19	6,886	89	20	5.34	77
Prepaid Travel Costs[7]	29	18,246	1,085	21	5.13	226
International Declarations[8]	34	6,449	753	27	11.18	245
PDC 2022 Base Logs[9]	12	1,000	293	27	5.39	83
PDC 2022 Ground Truth Logs[10]	10	1,000	229	32	5.52	62
PDC 2024 Base Logs[11]	16	1,000	892	35	12.21	128
PDC 2024 Ground Truth Logs[12]	17	1,000	885	42	12.24	118
PDC 2024 Test Logs[13]	20	1,000	902	39	12.29	171

We explored two scenarios: an unconstrained environment allowing free exploration and a constrained environment with a 60-second time limit and a reduced Pareto front of up to 20 models, simulating real-world computational constraints. Models exceeding a size of 150 were discarded during construction, as most models constructed from the event logs listed in Table 1 using the baseline *DFvM* algorithm are smaller than this size; refer to the last column in the table for the maximum sizes of models constructed

[2] https://github.com/HzhianUnimelb/BPM_ProcessDiscovery_Optimization
[3] https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
[4] https://doi.org/10.4121/uuid:500573e6-accc-4b0c-9576-aa5468b10cee
[5] https://doi.org/10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741
[6] https://doi.org/10.4121/uuid:895b26fb-6f25-46eb-9e48-0dca26fcd030
[7] https://doi.org/10.4121/uuid:5d2fe5e1-f91f-4a3b-ad9b-9e4126870165
[8] https://doi.org/10.4121/uuid:2bbf8f6a-fc50-48eb-aa9e-c4ea5ef7e8c5
[9] https://doi.org/10.4121/21261402.v1 (pdc2022_020111.xes)
[10] https://doi.org/10.4121/21261402.v1 (pdc2022_020111.xes)
[11] https://doi.org/10.4121/3cfcdbb7-c909-4f60-8bec-62c780598047.v1 (pdc2024_100111.xes)
[12] https://doi.org/10.4121/3cfcdbb7-c909-4f60-8bec-62c780598047.v1 (pdc2024_020111.xes)
[13] https://doi.org/10.4121/3cfcdbb7-c909-4f60-8bec-62c780598047.v1 (pdc2024_000110.xes)

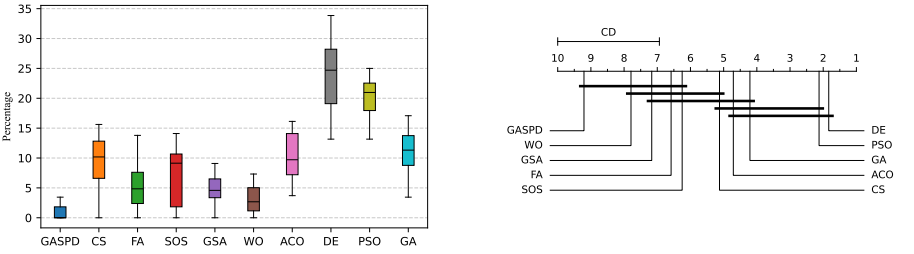

Fig. 3. Pareto front dominance count for fixed iterations optimization (50 iterations).

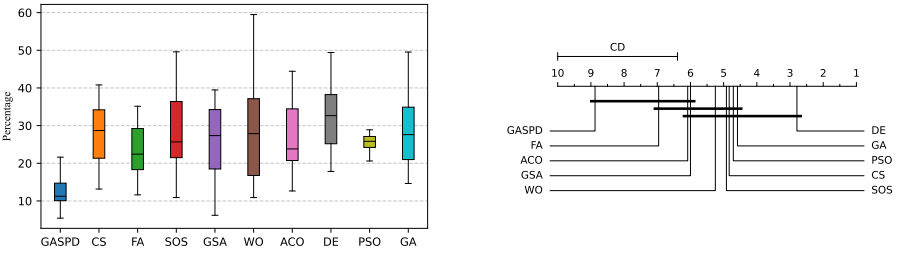

Fig. 4. Pareto front diversity (DCI) for fixed iterations optimization (50 iterations).

using *DFvM*. Both scenarios used consistent parameters—an initial population size of 50 and a maximum of 50 iterations—to ensure fair comparisons. We then compare the best-performing metaheuristic-based approach with *DFvM*.

Performance is evaluated using two metrics: dominance count and Diversity Comparison Indicator (DCI) [22]. Dominance count measures how many solutions from one algorithm dominate solutions from the other algorithms in the global Pareto front of all solutions from all algorithms, while DCI quantifies solution diversity by analyzing their distribution in the objective space. To compare the performance of the discovery techniques, we use the Friedman test [19], a non-parametric statistical test suitable for ranking-based comparisons. We report the mean ranks of the algorithms and apply the critical difference to determine statistically significant differences, enabling a robust evaluation of algorithm effectiveness.

Table 2. Runtime comparison (in seconds) across datasets for fixed iterations optimization (50); the best runtime per dataset is shown in **bold**, and the second-best is underlined.

Event log	GASPD	CS	FA	SOS	GSA	WO	ACO	DE	PSO	GA
Sepsis Cases	110	**25**	61	40	112	36	27	31	53	52
Hospital Billing	125	34	50	38	118	43	35	**23**	46	75
Request For Payment	**33**	104	54	93	102	137	53	34	344	102
BPIC 2012	384	313	351	742	435	365	**254**	341	323	384
BPIC 2013 Incidents	84	47	70	154	144	**33**	45	70	84	96
Prepaid Travel Costs	21	**12**	16	16	13	20	17	13	16	25
International Declarations	84	**64**	88	96	119	82	72	81	88	96
PDC 2022 Base Logs	73	27	95	101	128	45	**25**	95	49	74
PDC 2022 Ground Truth Logs	80	**53**	73	90	77	69	61	82	69	82
PDC 2024 Base Logs	75	**43**	83	140	154	74	56	94	77	65
PDC 2024 Ground Truth Logs	70	**43**	53	80	57	49	51	82	59	62
PDC 2024 Test Logs	89	63	36	104	103	61	**34**	75	57	75

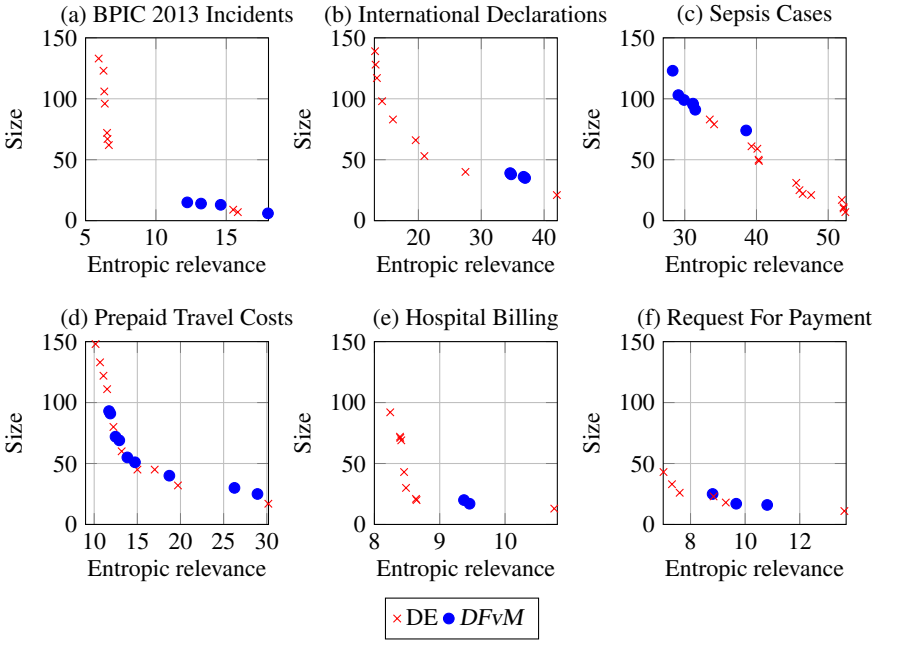

Fig. 5. Size and entropic relevance of SDAGs discovered using the DE metaheuristics and DFGs discovered using *DFvM* for fixed iterations optimization (50 iterations); smaller sizes and entropic relevance values signify better models.

Figures 3 and 4 report the results of the Friedman tests for dominance counts and diversity analysis, respectively. DE outperforms all other metaheuristics, achieving the highest median and lowest rank[14], both for dominance count and DCI. In contrast, *GASPD* is the least effective, with the lowest median and the highest mean rank across all the event logs. Similarly, DE demonstrates superior performance for the diversity of the discovered SDAGs, while *GASPD* constructs the least diverse models.

Table 2 reports the runtimes of discovering optimal SDAGs using different metaheuristics. It is interesting to see that the DE-based discovery is not characterized by prolonged runtimes, often constructing optimal models in a time that is comparable to the runtimes of the other metaheuristics.

To compare the models constructed using the DE metaheuristics and *DFvM*, in Fig. 5, we show global Pareto fronts of the models for six event logs. To obtain a Pareto front of the *DFvM* models for a given event log, we constructed 100 DFGs by varying the trace filtering threshold from 0.01 to 1 in increments of 0.01. On average, constructing a *DFvM* Pareto front took 29.42 seconds, with the fastest case being 9 seconds for the Request For Payment log and the slowest 78 seconds for BPIC 2012. From the visual analysis of the Pareto fronts, one can clearly conclude that there are more diverse, optimal DE models than *DFvM* models. On average, across the twelve event logs, the share of models on the global Pareto front (dominance count) of the DE and *DFvM*

[14] Note that lower rank is better, signifying superior performance.

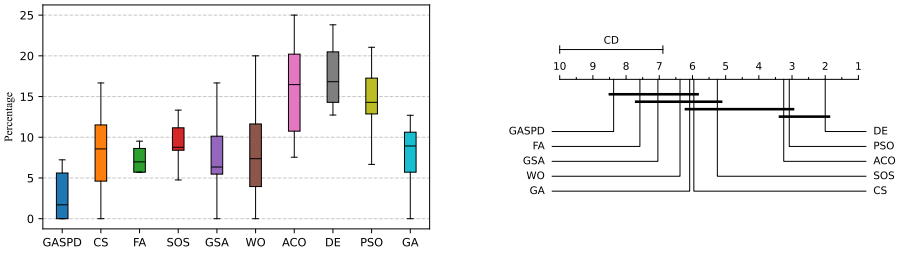

Fig. 6. Pareto front dominance count for limited execution time (60 seconds) and Pareto front size (at most 20 models).

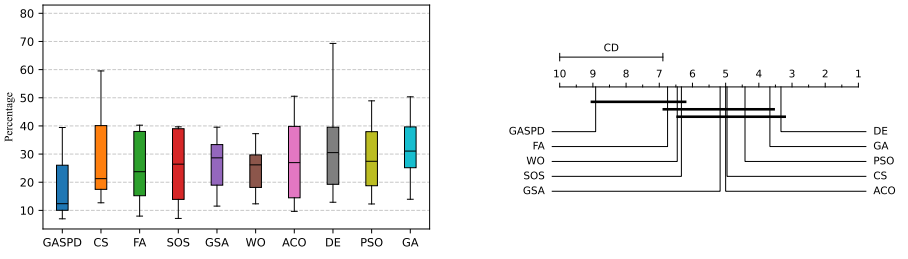

Fig. 7. Pareto front diversity (DCI) for limited execution time (60 seconds) and Pareto front size (at most 20 models).

models is 66.67% and 33.33%, respectively, confirming that there are more optimal DE models. Furthermore, for all the evaluated event logs, the diversity of the DE models, measured as DCI, is 44.58%, while the diversity of the *DFvM* models is 25.20%, suggesting the DE models are substantially more diverse in their quality characteristics.

Figures 6 and 7 summarize the analysis of Pareto front dominance counts and diversity for the limited 60-second execution time scenario, respectively, where at most 20 of the fittest models are retained in the population of each generation. Again, the DE metaheuristic demonstrates superior performance compared to other techniques, producing a greater number of models on the global Pareto front while maintaining diversity.

Figure 8 shows global Pareto fronts of the DE and *DFvM* models for six event logs, when the DE-based optimization was constrained to run for 60 seconds, preserving at most 20 Pareto optimal models in each generation. For this constrained scenario, across the twelve event logs, the share of models on the global Pareto front (dominance count) of the DE and *DFvM* models is 68.17% and 31.83%, respectively. Again, this suggests that more optimal DE models have been discovered. Furthermore, the diversity of the DE models, measured as DCI, is 42.29%, while the diversity of the *DFvM* models is 23.98%. Again, the DE models are substantially more diverse than the *DFvM* models.

6 Related Work

Evolutionary approaches to process discovery were initiated by the work of van der Aalst et al. [4], who applied genetic algorithms to extract process models while addressing the challenges of noise in event logs. This line of research was further advanced by techniques such as Evolutionary Tree Miner [11, 12] and Evolutionary Miner [24].

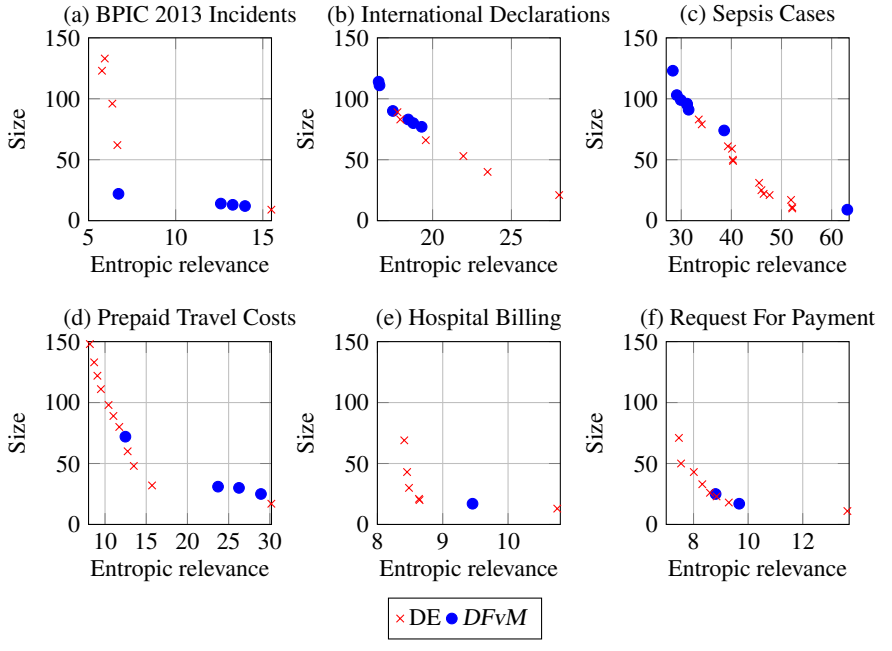

Fig. 8. Size and entropic relevance of SDAGs discovered using the DE metaheuristics and DFGs discovered using *DFvM* for limited execution time (60 seconds) and Pareto front size (at most 20 models); smaller sizes and entropic relevance values signify better models.

Vázquez-Barreiros et al. [28] introduced ProDiGen, a genetic algorithm-based method that integrates three key quality dimensions—precision, recall, and simplicity—into the selection of optimal process models. Unlike earlier approaches that rely on weighted fitness functions, ProDiGen adopts a multi-objective perspective, directly prioritizing the competing objectives in the model evaluation process.

Fantinato et al. [18] introduced the X-Processes method for process discovery, which is also based on genetic algorithms. Their approach employs a fitness function that integrates four key metrics commonly used to assess process model quality in process mining: recall, precision, generalization, and simplicity. Building on this foundation, Alkhammash et al. [6] demonstrate the effectiveness of genetic algorithms in optimizing grammatical inference for discovering high-quality stochastic process models. Motivated by these findings, the present study conducts a comprehensive evaluation of established metaheuristics to identify the approach best suited for the effective and efficient discovery of diverse, high-quality stochastic process models.

Genetic process discovery techniques typically begin with an initial population of process models, which are iteratively modified through crossover and random mutations. At each iteration, models are evaluated based on predefined quality measures. While these evolutionary approaches are robust against noise and incomplete data, they often suffer from slow convergence and may fail to consistently produce high-quality models due to limitations in the evaluation criteria. In contrast, Augusto et al. [9] introduce an alternative strategy by integrating single-solution-based metaheuristics, such

as iterative local search, tabu search, and simulated annealing, with Split Miner [8] to enhance model accuracy. However, their approach is tailored to non-stochastic settings, where the goal is to reproduce observed traces rather than estimate the likelihood of trace generation by the system.

7 Conclusion

In this study, we implemented and evaluated nine metaheuristics as alternatives to the Genetic Algorithm used in the *GASPD* stochastic process discovery algorithms, enhanced by niching techniques to ensure the diversity of the discovered models. These metaheuristics support the exploration of multiple promising regions in the search space, avoiding premature convergence and yielding a diverse set of high-quality process models. Our empirical evaluation over twelve real-world event logs demonstrates that Differential Evolution (DE) consistently outperforms other metaheuristics, including the *GASPD* algorithm. We refer to this DE-based discovery approach as *ADESPD*. Notably, *ADESPD* often surpasses *DFvM*, a state-of-the-art algorithm for discovering DFGs, constructing smaller, more accurate, and diverse process models, thus highlighting DE's effectiveness in stochastic process discovery. Our evaluation also confirms that *ADESPD* constructs superior models within practical timeframes, comparable with the runtimes of *DFvM*. We also proposed a classification of metaheuristics based on various solution update strategies to enhance the application of niching techniques to ensure solution diversity in multi-objective optimization. Future research could focus on refining niching techniques for process discovery and exploring further enhancements to DE and other metaheuristics for more robust process discovery algorithms.

References

[1] A multi-objective gravitational search algorithm. In: CICN, pp. 7–12 (2010)
[2] van der Aalst, W.M.P.: Process Mining—Data Science in Action. Springer, 2nd edn. (2016)
[3] van der Aalst, W.M.P.: A practitioner's guide to process mining: Limitations of the directly-follows graph. Procedia Computer Science **164**, 321–328 (2019)
[4] van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Genetic process mining. In: ICATPN, LNCS, vol. 3536, pp. 48–69, Springer (2005)
[5] van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
[6] Alkhammash, H., Polyvyanyy, A., Moffat, A.: Stochastic directly-follows process discovery using grammatical inference. In: CAiSE, LNCS, vol. 14663, pp. 87–103 (2024)
[7] Alkhammash, H., Polyvyanyy, A., Moffat, A., García-Bañuelos, L.: Entropic relevance: A mechanism for measuring stochastic process models discovered from event data. Inf. Syst. **107**, 101922 (2022)
[8] Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Polyvyanyy, A.: Split miner: Automated discovery of accurate and simple business process models from event logs. Knowl. Inf. Syst. **59**(2), 251–284 (2018)
[9] Augusto, A., Dumas, M., La Rosa, M.: Metaheuristic optimization for automated business process discovery. In: BPM, LNCS, vol. 11675, pp. 268–285 (2019)
[10] Biswas, S., Kundu, S., Das, S.: Inducing niching behavior in differential evolution through local information sharing. IEEE Trans. Evol. Comput. **19**(2), 246–263 (2015)

[11] Buijs, J.C.A.M.: Flexible Evolutionary Algorithms for Mining Structured Process Models. Ph.D. thesis, Eindhoven University of Technology (2014)
[12] Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A genetic algorithm for discovering process trees. In: IEEE Congress on Evolutionary Computation, pp. 1–8 (2012)
[13] Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: ICGI, pp. 139–152 (1994)
[14] Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
[15] Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization. IEEE Comput. Intell. Mag. **1**(4), 28–39 (2006)
[16] Ezugwu, A.E., Adeleke, O.J., Akinyelu, A.A., Viriri, S.: A conceptual comparison of several metaheuristic algorithms on continuous optimisation problems. Neural Comput. Appl. **32**(10), 6207–6251 (2020)
[17] Ezugwu, A.E., Prayogo, D.: Symbiotic organisms search algorithm: Theory, recent advances and applications. Expert Syst. Appl. **119**, 184–209 (2019)
[18] Fantinato, M., Peres, S.M., Reijers, H.A.: X-Processes: Process model discovery with the best balance among fitness, precision, simplicity, and generalization through a genetic algorithm. Inf. Syst. **119**, 102247 (2023)
[19] Friedman, M.: A comparison of alternative tests of significance for the problem of *m* rankings. The Annals of Mathematical Statistics **11**(1), 86–92 (1940)
[20] Kennedy, J., Eberhart, R.: Particle swarm optimization. In: ICNN (1995)
[21] Leemans, S.J.J., Poppe, E., Wynn, M.T.: Directly follows-based process mining: Exploration & a case study. In: ICPM, pp. 25–32 (2019)
[22] Li, M., Yang, S., Liu, X.: Diversity comparison of Pareto front approximations in many-objective optimization. IEEE Trans. Cybern. **44**(12), 2568–2584 (2014)
[23] Mirjalili, S., Lewis, A.: The whale optimization algorithm. Advances in Engineering Software **95**, 51–67 (2016)
[24] Molka, T., Redlich, D., Gilani, W., Zeng, X., Drobek, M.: Evolutionary computation based discovery of hierarchical business process models. In: BIS, LNBIP, vol. 208, pp. 191–204 (2015)
[25] Polyvyanyy, A., Moffat, A., García-Bañuelos, L.: An entropic relevance measure for stochastic conformance checking in process mining. In: ICPM, pp. 97–104 (2020)
[26] Sareni, B., Krähenbühl, L.: Fitness sharing and niching methods revisited. IEEE Trans. Evol. Comput. **2**(3), 97–106 (1998)
[27] Storn, R., Price, K.: Differential evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces. Tech. Rep. TR-95-012, International Computer Science Institute, 1947 Center Street, Berkeley (1995)
[28] Vázquez-Barreiros, B., Mucientes, M., Lama, M.: A genetic algorithm for process discovery guided by completeness, precision and simplicity. In: BPM, LNCS, vol. 8659, pp. 118–133 (2014)
[29] Winter, G., Periaux, J., Galan, M., Cuesta, P.: Genetic Algorithms in Engineering and Computer Science (1996)
[30] Yang, X.S.: Nature-Inspired Metaheuristic Algorithms (2008)
[31] Yang, X.S., Deb, S.: Cuckoo search via Lévy flights. In: World Congress on Nature & Biologically Inspired Computing (NaBIC), pp. 210–214 (2009)