Federated Stochastic Process Discovery Using Grammatical Inference

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 for constructing process models that describe control flow of systems that generated given event logs, where an event log is a recording of executed traces of a system, with each trace captured as a sequence of executed actions. Traditional process discovery relies on an event log recorded and stored in a centralized repository. However, in distributed environments, such as cross-organizational process discovery, this centralization raises concerns about data availability, privacy, and high communication and bandwidth demands. To address these challenges, this paper introduces a novel Federated Stochastic Process Discovery (FSPD) approach. FSPD avoids centralized event logs by retaining them in decentralized silos, at organizations where they were originally recorded. Process discovery is then performed locally within each organization on its event log, and the resulting local models are shared with a central server for aggregation into a global model. Our evaluations on industrial event logs demonstrate that FSPD effectively constructs global process models while preserving organizational autonomy and privacy, providing a scalable and robust solution for process discovery in distributed settings.

Keywords: Stochastic process mining, federated process discovery, cross-silos process discovery, grammatical inference

1 Introduction

Process mining studies ways to use data generated by information systems when executing business processes to understand historical and to improve future processes [2]. Process discovery is one of the core problems in process mining. Given an event log, a collection of recorded traces, each captured as a sequence of executed actions, a process discovery algorithm aims to construct a process model that faithfully describes the traces the system that generated the event log can support [5]. A good discovered process model should describe as many traces from the input event log as possible (good recall), not describe many traces that are not in the event log (good precision), describe traces the system is likely to execute in the future (good generalization), and be as simple as possible (good simplicity) [7].

Most commercial process discovery tools construct Directly-Follows Graphs (DFGs) from event logs [3]. A DFG is a directed graph with nodes representing actions, edges capturing the "can follow" ordering constraints between the actions, and numeric weights attached to nodes and edges specifying indicative frequencies of observing the corresponding actions and their adjacent executions in the event log. The level of detail in the constructed DFG is typically controlled by a threshold, which specifies how much information from the event log can be disregarded in the model [3]. Business analysts study the system by exploring models constructed at various thresholds and, thus, different levels of detail [15]. To support interactive adjustments of the threshold, most DFG discovery algorithms aim to meet tight runtime requirements, often operating in linear time relative to the size of the input event log.

Alkhammash et al. [6] present a genetic algorithm for stochastic process discovery (*GASPD*) that can construct DFGs that are Pareto-superior in interesting qualities over DFGs constructed by the state-of-the-art techniques proposed by academia, which construct models of similar qualities as process discovery tools from major process mining vendors [14]. *GASPD* is based on *ALERGIA*, a grammatical inference technique with linear empirical and cubic worst-case runtime complexity relative to the size of the event log [9]. However, *GASPD* searches the parameter space of the grammatical inference using a genetic strategy, which substantially increases the overall runtime requirements.

In this paper, we present an extension of *GASPD* capable of federated process discovery. *GASPD*, like other traditional process discovery algorithms, operates over a centralized event log. Federated *GASPD* works over a distributed event log in two phases. First, *GASPD* discovers models of various quality characteristics on each part of a distributed event log, where the distributed event log can be seen as a collection of local event logs, with each local event log stored on a dedicated device. Then, superior DFGs discovered from each local event log are sent to the server, where they are aggregated into a (collection of) sound [6], that is, correct, DFG(s) that describe the overall system. This distributed divide-and-conquer process discovery approach supports cross-organizational process discovery, which aims to analyze processes that span multiple organizations that may be reluctant to share raw data and instead exchange event log abstractions [4, 12]. Furthermore, federated process discovery in the cross-silos setting improves data privacy and security [12, 16], reduces data transfer costs [12, 17], and supports scalability and efficiency [1, 8, 17].

Concretely, this paper contributes:

1. An efficient algorithm for merging a deterministic frequency finite automaton (DFFA) into a target DFFA, where a DFFA is an abstract model of an event log constructed by *ALERGIA* that can be translated to a DFG by a technique used by *GASPD*.
2. A discussion of formal properties of merged DFFAs (and the practical consequences of these properties), including consistency, similarity of the DFFAs involved in merging, and morphism between the merged DFFA and the input target DFFA.
3. The *FedGASPD* and *FedCGASPD* algorithms (federated versions of *GASPD*) that adapt *GASPD* to discover DFFAs using *ALERGIA* from local event logs and then merge superior discovered DFFAs into, respectively, a single DFFA and a collection of DFFAs that can be translated to sound DFGs, thus implementing FSPD.
4. An evaluation over industrial event logs confirming that *FedGASPD* and *FedCGASPD* can discover process models from distributed event logs that have similar quality characteristics as process models constructed by *GASPD* from the corresponding centralized event logs and work substantially faster than *GASPD*, with *FedCGASPD* constructing interesting models of smaller sizes.

The remainder of the paper proceeds as follows. The next section discusses related work. Then, Section 3 presents preliminaries necessary to understand the subsequent discussions. Section 4 introduces *FedGASPD* and *FedCGASPD*, while Section 5 presents results of an evaluation of these algorithms based on their open-source implementation.[1] Finally, Section 6 closes the paper with a discussion of the limitations of our approach and directions for future work, and states concluding remarks.

2 Related Work

This section discusses existing works in federated process mining, divided-and-conquer strategies for process discovery, and cross-organizational process mining.

Van der Aalst [4] introduces a federated process mining approach to create a unified view of cross-organizational processes. This is achieved by mapping multiple organization-specific event logs into a single federated event log. The paper distinguishes between two types of federated process mining: organizations share filtered event data, and organizations share abstractions, such as DFGs, to maintain higher levels of confidentiality. However, it does not propose a solution for merging these abstractions, leaving this as an open problem for future research, which is addressed in this work.

Rojo et al. [17] explore federated process mining techniques that leverage distributed devices, such as smartphones, to analyze human actions and interactions (including those with other individuals). These techniques conceptualize human behavior as a process and focus on event logs collected from individual devices. The authors propose two approaches: discovering process models directly from the data of individuals and integrating event logs from multiple devices into a centralized dataset to construct process models. While the latter approach involves aggregating traces from multiple devices to a central repository, it often incurs substantial data transfer costs. In contrast, our approach discovers models locally on each device and merges them into a final model, significantly reducing communication overhead.

Khan et al. [12] present a federated approach for cross-silo process mining, utilizing a dependency graph to analyze distributed process logs while maintaining data privacy collaboratively. The authors design a protocol for federated process discovery tailored to the Heuristic Miner process discovery algorithm that relies on a centralized trusted server to orchestrate communication between the parties that own parts of the entire event log. Our approach can be embedded into similar protocols for federate process discovery. Rafiei and van der Aalst [16] propose an abstraction-based approach for privacy-aware federated process discovery in inter-organizational settings, utilizing directly-follows relations as abstractions of event logs. The approach highlights the importance of enabling organizations to collaborate while safeguarding sensitive information through mechanisms such as handover relations and risk-aware reveal methods. A handover occurs when the execution of a trace transitions from one organization to another. Their method constructs event log abstractions from trace fragments visible to each organization, which are then aggregated into a unified abstraction representing the entire event log. By contrast, we partition the event log into groups of complete

[1] https://github.com/HzhianUnimelb/FederatedProcessDiscovery

traces—for example, those managed by subsidiaries or business units within an organization—and derive abstractions for each group prior to merging them.

Carmona et al. [8] presents two techniques for decomposing process discovery using the theory of regions. The first technique performs a local search for regions rather than globally, and constructs model components from these local regions. The second technique involves selecting groups of related events from the event log, projecting the log onto these groups, and then discovering model components from each of these projections. In both cases, the identified components are then combined into the final process model using parallel composition. Van der Aalst [1] introduces a divide-and-conquer framework for process discovery based on partitioning of actions. This framework also proceeds by splitting the actions into (possibly overlapping) groups, projecting the event log onto these groups, discovering model components from the projections, and composing the components into the resulting process model. Several strategies for decomposing event logs are discussed. For example, one strategy is based on single-entry-single-exit (SESE) fragments, which split the actions based on well-defined sub-processes with unique entry and exit points. Yan et al. [19] present a five-step framework for decomposing process discovery. First, an action relationship graph is derived from the event log, representing the dependencies between actions. This graph is then decomposed into smaller subgraphs, each representing a subset of the process. These subgraphs are used to create corresponding sublogs, which serve as input for constructing submodels. Finally, the submodels are composed to form the complete process model. An instantiation of the framework based on SESE decomposition and heuristic miner is proposed. Different from the existing techniques, we work with multiple collections of complete traces rather than their projections, discover models from these collections, each capturing aspects of the overall process based on a data subset, and, finally, merging, rather than assembling, the intermediate models.

3 Preliminaries

This section presents concepts necessary to understand the contributions discussed in the subsequent sections. A Deterministic Frequency Finite Automaton (DFFA) is a mathematical model that describes traces and their frequencies.

Definition 3.1 (Deterministic Frequency Finite Automata)
A *Deterministic Frequency Finite Automaton* (DFFA) is a tuple $(Q, \Lambda, q_0, \mathbb{I}, \mathbb{F}, \delta)$, where:
- Q is a finite nonempty set of *states*;
- Λ is a finite set of *actions*, called the *alphabet*;
- $\mathbb{I} : Q \to \mathbb{N}$ is the *initial state frequency function*, with one *initial state* $q_0 \in Q$ for which it holds that $\mathbb{I}(q_0) > 0$ and for all $q \in Q$, $q \neq q_0$, it holds that $\mathbb{I}(q) = 0$;
- $\mathbb{F} : Q \to \mathbb{N}$ is the *final state frequency function*; and
- $\delta : Q \times \Lambda \times Q \to \mathbb{N}$ is the *transition frequency function*, such that $\forall q \in Q\, \forall a \in \Lambda\, \forall q' \in Q\, \forall q'' \in Q : ((\delta(q, a, q') > 0 \wedge \delta(q, a, q'') > 0) \Rightarrow (q' = q''))$. ⌟

The notation $\delta(q, a, q') = n$ indicates that there is a transition from state q to state q' labeled with a, occurring n times. Figure 1 shows a DFFA with states $\{p_0, \ldots, p_4\}$, initial state p_0, $\mathbb{I}(p_0) = 10$, $\mathbb{F} = \{(p_0, 1), (p_1, 4), (p_2, 1), (p_3, 2), (p_4, 2)\}$, and $\delta(p_0, a, p_1) = 4$, $\delta(p_0, b, p_1) = 5$, $\delta(p_1, a, p_2) = 3$, $\delta(p_1, b, p_3) = 2$, $\delta(p_2, a, p_4) = 2$, and $\delta(p_4, a, p_4) = 2$.

A DFFA is consistent if for each of its states the sum of frequencies of entering and leaving (or terminating at) that state is identical.

Definition 3.2 (Consistency [11])
A DFFA $(Q, \Lambda, q_0, \mathbb{I}, \mathbb{F}, \delta)$ is *consistent* if and only if it holds that:

$$\forall q \in Q : \left(\mathbb{I}(q) + \sum_{q' \in Q,\, a \in \Lambda} \delta(q', a, q) \right) = \left(\mathbb{F}(q) + \sum_{q' \in Q,\, a \in \Lambda} \delta(q, a, q') \right).$$

⌟

The DFFA in Fig. 1 is consistent. The *simulation* relation associates automata that behave in a similar way, that is, one automaton can "mimic" the actions of the other.

Definition 3.3 (Similarity [18])
DFFA $A_1 = \left(Q^1, \Lambda^1, q_0^1, \mathbb{I}^1, \mathbb{F}^1, \delta^1\right)$ *simulates* DFFA $A_2 = \left(Q^2, \Lambda, q_0^2, \mathbb{I}^2, \mathbb{F}^2, \delta^2\right)$, denoted by $A_2 \preceq A_1$, if and only if there exists a relation $\mathbb{S} \subseteq Q^1 \times Q^2$ such that:

– $\left(q_0^1, q_0^2\right) \in \mathbb{S}$, and
– for every $(p, q) \in \mathbb{S}$, if there exists $a \in \Lambda^1$ and $p' \in Q^1$ such that $\delta^1(p, a, p') > 0$, then it holds that there exists $q' \in Q^2$ such that $\delta^2(q, a, q') > 0$ and $(p', q') \in \mathbb{S}$. ⌟

Our approach to federated process discovery is based on *GASPD*, an evolutionary stochastic process discovery algorithm grounded in grammatical inference [6]. *GASPD* uses *ALERGIA* to extract accurate stochastic language models of traces recorded in an input event log. *ALERGIA* operates by first constructing the prefix automaton of the event log and then iteratively reduces it through recursive merging of states that spawn similar sequences of actions into a consistent DFFA [9].

Algorithm 1 summarizes *GASPD* in pseudocode.

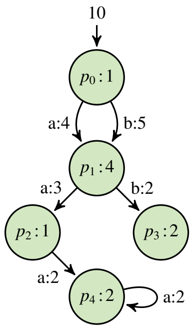

Fig. 1: DFFA A_1

Algorithm 1: *GASPD* [6]

```
Data: Initial population size $p$, number of generations $m$, number
      of parents $k$ to generate offspring, and event log $L$
Result: Pareto-optimal DFGs defines by parameter triples
1 $g \leftarrow 0$;
2 $P \leftarrow$ POPULATION($p$);
3 while $g < m$ do
4     $F \leftarrow$ SELECT($P, L$);
5     $U \leftarrow$ CROSSOVER-MUTATION($F, k$);
6     $P \leftarrow$ REPLACE-ELITE($U, F, L$);
7     $g \leftarrow g + 1$;
8 return SELECT($P, L$);
```

GASPD uses a multi-objective genetic search to discover interesting DFGs from the input event log. It begins by generating a population P of p parameter triplets (Line 2).

Each triplet defines a DFG, with one parameter determining which infrequent traces to exclude from the event log and two parameters guiding *ALERGIA* in constructing a DFFA from the remaining frequent traces. Using its native translation mechanism, *GASPD* converts these DFFAs into sound DFGs. In the selection phase (Line 4), the most "fit" population members F are chosen. Fitness is defined by Pareto optimality concerning model size (smaller models are preferred) and entropic relevance of discovered models (lower relevance values are preferred). Relevance serves as a quality criterion because it enables rapid model scoring, balances precision and recall relative to the event log, and quantifies how well the model captures the likelihood of observed traces [6]. Next, in the crossover phase (Line 5), offspring U are generated by applying crossover and mutation operations to k randomly selected parents from F. These operations aim to ensure that offspring inherit the best features of prior generations. Finally, in the replacement phase (Line 6), *GASPD* updates the population by preserving Pareto-optimal members in F and incorporating new, superior members from U.

4 Federated Process Discovery

This section presents the problem of federated (stochastic) process discovery, a technique for merging DFFAs, the *FedGASPD* algorithm that discovers one DFFA from each input event log and then merges them into the final discovered process model, and the *FedCGASPD* algorithm that refines *FedGASPD* to merge similar models to obtain a range of models of different characteristics.

4.1 Definition

Let X be a set. By $\mathcal{B}(X)$, we denote the set of all finite multisets over X, while, by X^*, we denote the set of all finite sequences over X. Let $\mathbb{X} \in X^*$ be a sequence over X. Then, by $\mathbb{X}(i)$, $i \in [1 .. |\mathbb{X}|]$, we denote the element at position i in $\mathbb{X}$.

Let $\mathcal{A}$ be the universe of *actions*. Then, $\mathcal{L} = \mathcal{B}(\mathcal{A}^*)$ is the universe of *event logs*. Let $\mathcal{M}$ be the universe of *process models*. Given an event log $L \in \mathcal{L}$, the conventional *process discovery* problem studies ways to construct a process model that describes the system that generated L. A solution to the process discovery problem can be captured as a function $d : \mathcal{L} \rightarrow \mathcal{M}$. The *federated process discovery* problem studies ways to construct a process model from a list of event logs, where each event log from the list can be stored on a dedicated device at a particular organization. That is, a solution to the federated process discovery problem can be given as a function $f : \mathcal{L}^* \rightarrow \mathcal{M}$.

Let $q : \mathcal{L} \times \mathcal{M} \rightarrow [0, 1]$ be a function that measures *quality* $q(L, M)$, $L \in \mathcal{L}$, $M \in \mathcal{M}$ of model M discovered from event log L. For instance, measure q can be precision, recall, simplicity, or generalization [7]. Such a measure usually associates larger values with models that describe the system better. A good federated process discovery approach f should construct a model from multiple event logs that is comparable in quality to the model constructed from the corresponding centralized event log:

$$\forall \mathbb{L} \in \mathcal{L}^* : q(L, f(\mathbb{L})) \approx q(L, d(L)), \text{ where } L = \uplus_{i=1}^{|\mathbb{L}|} \mathbb{L}(i) .$$

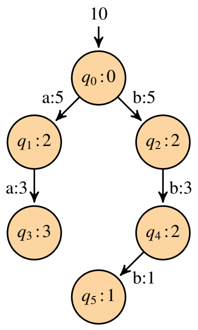

Fig. 2: DFFA A_2

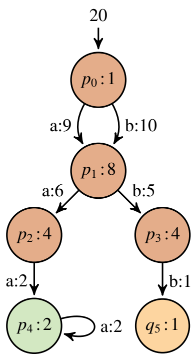

Fig. 3: DFFA A

In this paper, we solve the federated process discovery problem by first constructing process models from individual event logs using the conventional approach and then merging the obtained models into the final process model. Let $m : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ be a function that merges models. Specifically, given a non-empty, finite list of event logs $\langle L_1, \ldots, L_n \rangle \in \mathcal{L}^*$, $n \in \mathbb{N}$, we first discover the sequence of models $\langle M_1, \ldots, M_n \rangle$, $M_i = d(L_i)$, $i \in [1 \mathinner{..} n]$, using the conventional approach on each event log. Each such model can be constructed on a dedicated device at the organization that owns the data and is an abstract, and thus more confidential [4], representation of the corresponding event log. We then merge the models incrementally using function $F_m : \mathcal{M}^* \to \mathcal{M}$:

$$F_m(\langle M_1 \rangle) = M_1 \text{ and } F_m(\langle M_1, M_2, \ldots, M_n \rangle) = F(\langle m(M_1, M_2), \ldots, M_n \rangle) .$$

In general, different orders and approaches to model merging can lead to different resulting models. Next, we discuss an approach to merging DFFAs.

4.2 Model Merging

We refer to the procedure for merging a DFFA A_2 into a DFFA A_1 as *MergeDFFA*. In addition to the two automata, the procedure takes as input a parameter $d \in \mathbb{N}$. The first step is to construct the (prefix of the) unfolding of A_2 truncated at the $d + 1$ occurrences of its states [10]. In this prefix automaton, state and transition frequencies are derived from those in A_2 by propagating frequencies along the unfolding branches, using probabilities estimated based on the original frequencies in A_2. *MergeDFFA* then "embeds" this prefix automaton into A_1 by reusing the structure of A_1 and extending it as necessary to incorporate any remaining parts of the prefix. During this embedding process, the frequencies from A_1 and the constructed prefix of A_2 are aggregated. We denote the result of merging A_2 into A_1 using parameter d as $A_1 \triangleleft_d A_2$.

DFFA A in Fig. 3 is the result of merging A_2 in Fig. 2 into A_1 in Fig. 1 using $d = 1$, that is, $A = A_1 \triangleleft_1 A_2$. In the figure, the khaki-colored states represent states from A_1 that were reused during the embedding of the prefix unfolding of A_2 into A_1. The green state p_4 corresponds to a state of A_1 that was not visited during the embedding, while the orange state q_5 originates from A_2 and was used to extend the structure of A_1.

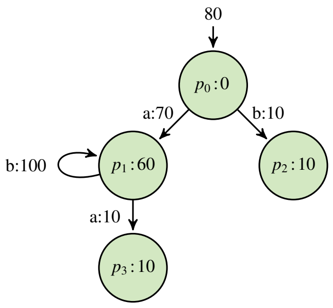

Fig. 4: DFFA A_3

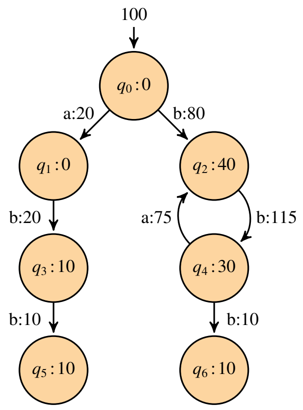

Fig. 5: DFFA A_4

Since A_2 is acyclic and has no states with multiple incoming transitions, the prefix of its unfolding for any $k \in \mathbb{N}$ is structurally identical to A_2. The merged automaton A incorporates all transitions from both A_1 and A_2, and it permits termination in the same states as these two automata. Consequently, A simulates both A_1 and A_2. When embedding A_2 into A_1, the transitions from A_2 "follow" those of A_1 as closely as possible accumulating transition and state termination frequencies from both automata, effectively merging their behaviors. After executing b twice, both A_1 and A_2 reach a common merged state p_3. At this point, A_1 terminates, while A_2 is capable of performing an additional b transition. This leads to the introduction of a new state q_5, extending the behavior of A beyond A_1 to simulate the behavior of A_2.

Figure 6 shows the prefix of the unfolding of the DFFA A_4 from Fig. 5 computed for $d = 2$. The construction begins at the initial state of the input automaton and proceeds iteratively: at each step, a state in the current prefix is selected and extended using one of its outgoing transitions from the input automaton. Each extension introduces a fresh state to represent the target of the chosen transition, ensuring that the unfolding forms a tree. To guide the construction, only states with unexplored transitions are selected for expansion. If the target state of a transition has already appeared d times among the ancestors of the current node in the prefix, the expansion is halted; otherwise, if the input DFFA has a cycle, the prefix construction will continue forever. In the example prefix shown in Fig. 6, the state q'_4 corresponds to state q_4 in A_4. However, it is not expanded via transition a because doing so would introduce a third occurrence of state q_2 along the path from q'_4 to the root (state q_0). To ensure the consistency of the prefix automaton, the frequency of a pruned state is accumulated in the parent state (state q'_4 in the example). The frequencies of states and transitions in the prefix automaton are computed by multiplying the frequency of each concept in the input automaton by the probability of reaching its corresponding occurrence during the execution of the input automaton. This probability is estimated based on the state and transition frequencies in the input automaton.

DFFA B in Fig. 7 is the result of merging A_4 from Fig. 5 into A_3 from Fig. 4 using $d = 2$, that is, $B = A_3 \triangleleft_2 A_4$. The khaki-colored states indicate states from A_3 that were

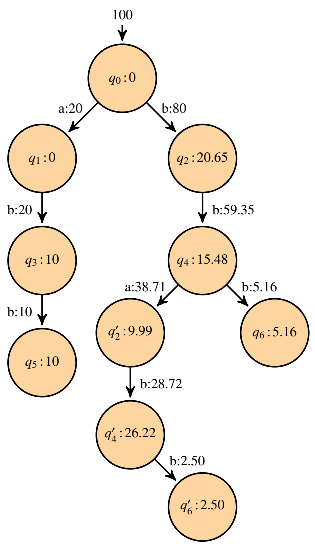

Fig. 6: Prefix of DFFA A_4 ($d = 2$)

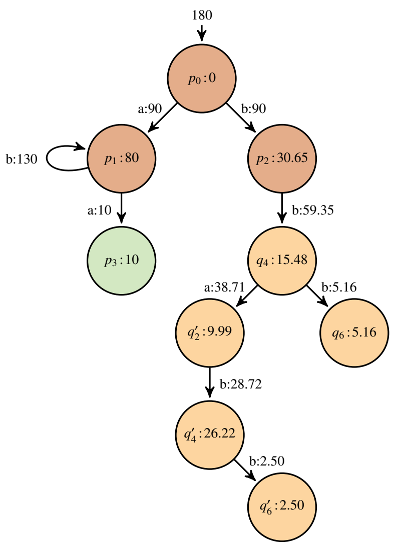

Fig. 7: DFFA B

reused during the embedding of the prefix of A_4 from Fig. 6 into A_3. The green states correspond to the original states of A_3 not visited during the embedding of the prefix, while the orange states are the states of the prefix that do not "fit" into A_3.

MergeDFFA terminates after it explores the necessary pairs of states from the two input automata starting from the pair of the initial states. The procedure always terminates because the number of all possible such state pairs is finite. The runtime of the merging process is primarily determined by the size of the prefix of the unfolding of DFFA A_2, which, in the worst case, can be exponential in the size of A_2. When the parameter d is set to a small value, the size of the prefix unfolding is typically only a few times larger than that of A_2. However, increasing d allows the behavior of A_2 to be captured more accurately in the resulting merged automaton.

The automaton that results from merging two consistent automata is consistent.

Lemma 4.1 (Consistency)
If A_1 and A_2 are consistent DFFAs, then DFFA A, such that $A = A_1 \lhd_d A_2$, $d \in \mathbb{N}$, is consistent. ⌟

Lemma 4.1 holds because the prefix automaton that *MergeDFFA* embeds into A_1 is, by construction, consistent. Furthermore, each time a state from the prefix of the unfolding is embedded into the resulting automaton, the *MergeDFFA* procedure updates both the termination frequency of the merged state and the frequencies of its outgoing transitions to preserve consistency.

The automaton produced by *MergeDFFA* is in the following simulation relations with the input automata.

Lemma 4.2 (Similarity)
If A_1 and A_2 are consistent DFFAs and $A = A_1 \lhd_d A_2$, $d \in \mathbb{N}$, it holds that:

– *A simulates A_1, that is, $A_1 \preceq A$; and*
– *if A_2 is acyclic, then A simulates A_2, that is, $A_2 \preceq A$.* ⌟

Lemma 4.2 holds based on the following observations. Each automaton produced by *MergeDFFA* is constructed by extending the structure of DFFA A_1, thus preserving all of its transitions. Also, *MergeDFFA* ensures that all transitions from the prefix of the unfolding of A_2 are retained in the resulting DFFA. If A_2 is acyclic, the unfolding of A_2 simulates A_2. Finally, the unfolding of an acyclic automaton is isomorphic to the prefix of the unfolding of this automaton for any truncation depth $d > 0$.

Notably, given DFFAs A_1 and A_2 and $d \in \mathbb{N}$, it is not always the case that $A_1 \lhd_d A_2 = A_2 \lhd_d A_1$; that is, the merging operation is not commutative. This fact suggests that if *MergeDFFA* is used to merge models when solving the federated process discovery problem, different orders of mergings will indeed lead to different discovered models.

4.3 Discovering Single Model

Given parameters to configure the *GASPD* algorithm and a collection of event logs, each stemming from the same process but possibly recorded by different organizations or different departments within a single organization, the *FedGASPD* algorithm solves the federated process discovery problem by constructing one model from each event log and then merging them into the final process model; refer to Algorithm 2.

Algorithm 2: *FedGASPD*

```
Input: Initial population size $p$, number of generations $m$, number of parents $k$ to
       generate offspring, and a non-empty list of event logs $L = \langle L_1, \dots, L_n \rangle$
Output: A DFFA discovered from $L$
1 $M \leftarrow \langle\rangle$;
2 parallelfor $i \in [1 \mathinner{..} n]$ do
3     $M \leftarrow M \circ \langle SELECTDFFA(GASPD(p, m, k, L_i))\rangle$;
4 return $F_{MergeDFFA}(M)$;
```

To discover a model from an input event log, we first use the *GASPD* algorithm (Algorithm 1) to identify Pareto-optimal models based on their size and entropic relevance. From the set of Pareto-optimal models, one is selected using the *SELECTDFFA* function. The selected model is then added to the list of constructed DFFAs, M, using the sequence concatenation operator "$\circ$" (Line 3). Since the above-outlined procedure for obtaining a model can be performed independently for each input event log, we execute all such procedures for all input event logs in parallel (Line 2). Various model selection policies can be implemented by different implementations of the *SELECTDFFA* function. Finally, once all models are selected, they are merged pairwise using the *MergeDFFA* procedure (Line 4); cf. Section 4.2 for details on the merging procedure.

The discovery of Pareto-optimal models and the selection of one optimal model for each input event log can be performed locally on devices at the respective organizations where the input event logs are stored. Once the models are selected, they can be transmitted to a central server, where the merging process takes place.

Note that *GASPD* identifies parameter triplets that trigger *ALERGIA* to construct automata that can be translated into interesting DFGs. The *SELECTDFFA* function, thus, selects one parameter triplet identified by *GASPD* and constructs the corresponding DFFA using *ALERGIA*. After merging all DFFAs into a final model, this model can be trivially translated into a probabilistic automaton using Algorithm 16.1 by de la Higuera [11], which, in turn, can be trivially converted into a DFG model following the procedure outlined in Definition 4.2 by Alkhammash et al. [6]. To ensure brevity of presentation and because the quality of DFFAs directly impacts the quality of the resulting DFGs, here, we design the core techniques to operate with DFFAs.

4.4 Discovering Multiple Models

FedGASPD solves federated process discovery in a straightforward manner: it discovers one model from each input event log and merges them. This approach gives a basic baseline for federated process discovery. However, when the discovered models exhibit diverse characteristics, merging them may result in a loss of distinctiveness and dilute their strong individual features due to the averaging effect. To address this limitation, we propose a variation of *FedGASPD* that avoids merging all models into one. Instead, it groups similar models into clusters and merges the models within each cluster, yielding multiple models that retain similar, strong characteristics of the merged models. We refer to this variant as *FedCGASPD*, which is summarized in Algorithm 3.

```
Algorithm 3: FedCGASPD
Input: Initial population size $p$, number of generations $m$, number of parents $k$ to
       generate offspring, and a non-empty list of event logs $L = \langle L_1, \ldots, L_n \rangle$
Output: A list of DFFAs discovered from $L$
1 $M \leftarrow \langle\rangle; \mathbb{M} \leftarrow \langle\rangle$;
2 parallelfor $i \in [1 \mathinner{..} n]$ do
3     $M \leftarrow M \circ \langle SELECTDFFA(GASPD(p, m, k, L_i))\rangle$;
4 $C \leftarrow KMeansCluster(M, n)$;
5 for $i \in [1 \mathinner{..} n]$ do
6     if $|C(i)| \geq 0$ then $\mathbb{M} \leftarrow \mathbb{M} \circ \langle F_{MergeDFFA}(C(i))\rangle$ ;
7 return $\mathbb{M}$;
```

After discovering models from input event logs (Lines 2 and 3 in Algorithm 3), they are clustered using the k-means algorithm [13] via the *KMeansCluster* function (Line 4). The *KMeansCluster* function takes as input the list of models, M, and the desired number of clusters, n, and outputs a list of clusters, C, where each cluster is stored as a list of models. We specify n as the number of desired clusters and initialize n random centroids (initial cluster centers), each represented as a point in the space $[0, 1] \times [0, 1]$. If there are fewer than n meaningful clusters, the k-means algorithm identifies these clusters by iteratively refining the centroids while allowing some centroids to remain unused. For clustering, each DFFA is represented as a point in a two-dimensional space, defined by its normalized size and entropic relevance. Normalization is achieved by dividing the size and entropic relevance of each model by the respective maximum values across all models in M. Then, the *MergeDFFA* procedure is applied to each non-empty cluster. All models within a cluster are merged, and the resulting merged model is appended to the list of discovered models (Line 6).

5 Evaluation

We implemented the *GASPD*, *FedGASPD*, and *FedCGASPD* algorithms and made them publicly available.[2] Using this implementation, we evaluated the performance of these algorithms using nine real-world event logs, which we sourced from the IEEE Task Force on Process Mining (https://www.tf-pm.org/resources/logs). For experiments, we selected event logs that exhibit a wide range of characteristics, such as the number of (distinct) actions, the number of (distinct) traces, and trace length. This diversity is essential for evaluating the robustness of our algorithms in various real-world scenarios. Table 1 summarizes the characteristics of the nine event logs.

Table 1: Characteristics of the event logs used in our experiments

Event log	#Actions	#Traces	#Distinct traces	Max. trace length	Avg. trace length
BPI-2012	24	13,087	4,366	175	20.03
BPI-2013	4	7,554	1,511	123	8.67
BPI-2017	26	31,509	15,930	180	38.15
BPI-2018	34	2,861	2,498	680	61.58
BPI-2019	42	251,734	11,973	990	06.33
BPI-2020-1	34	6,449	753	27	11.18
BPI-2020-2	19	6,886	89	20	05.34
Sepsis Cases	16	1,050	846	185	14.48
nasa-cev	47	2,566	2,513	50	28.69

All experiments were executed on a computer featuring a 13th Gen Intel® Core$^{\text{TM}}$ i7-1355U processor running at 1.70GHz using 16.0GB of RAM (15.7GB of effective memory). The aim of the experiments is twofold. Firstly, we study the feasibility of using federated process discovery in industrial settings. Secondly, we compare the quality of the models generated by *FedGASPD* and *FedCGASPD* to those produced using the centralized *GASPD* method. For each event log, *GASPD*, *FedGASPD*, and *FedCGASPD* were initialized the initial population size, p, and the number of generations, m, to 50. To construct human-readable models, we controlled the parameter responsible for the exclusion of infrequent traces. Consequently, we constructed models of at most 1,000 nodes and arcs, with many models substantially smaller than this target size.

To simulate cross-organizational environments, each event log was split randomly among a requested number of computational nodes, ensuring that each node has approximately the same portion of the event log traces. We implemented the *SELECTDFFA* function from Algorithms 2 and 3, by selecting one DFFA from a Pareto-frontier of DFFAs that minimizes this objective function: $(1 - size^*) + (1 - relevance^*)/2$, where $size^*$ and $relevance^*$ are the normalized size and entropic relevance of the model, with

[2] https://github.com/HzhianUnimelb/FederatedProcessDiscovery

Table 2: Execution times of *GASPD*, *FedGASPD*, and *FedCGASPD* algorithms

Event log	Execution time (in seconds)						
	GASPD	*FedGASPD*			*FedCGASPD*		
		2 nodes	4 nodes	8 nodes	2 nodes	4 nodes	8 nodes
BPI-2012	416	178	83	61	166	81	58
BPI-2013	84	35	26	11	43	22	17
BPI-2017	3,120	1,459	514	342	1,401	499	367
BPI-2018	2,898	1,325	165	76	1,317	173	78
BPI-2019	2,288	2,216	471	251	2,224	480	260
BPI-2020-1	1402	685	331	229	676	342	236
BPI-2020-2	175	94	57	46	92	53	42
Sepsis Cases	421	234	64	31	237	60	36
nasa-cev	3,820	575	329	136	568	332	142

normalization performed as described in Section 4.4. This way, we select a model that balances the two competing objectives equally.

Table 2 presents the execution times of *GASPD* and its federated alternatives. Note that *FedGASPD* and *FedCGASPD* are substantially faster than *GASPD*. For example, for the nasa-cev event log, the federated discovery techniques are up to 26 times faster than *GASPD* and, at a minimum, 5 times faster, refer to BPI-2013 event log. This observation can be explained by the fact that *FedGASPD* and *FedCGASPD* perform computationally demanding *GASPD* genetics on many smaller event logs in parallel and then apply a computationally efficient merging of the resulting models.

Figure 8 shows the characteristics of the Pareto-optimal models discovered from the nine event logs by the three algorithms for the different numbers of computation nodes, while Tables 3 and 4 summarize the ranges of the sizes and entropic relevance values of the discovered models. These results further support our contention that federated approaches yield useful new models compared to the centralized solution. When comparing the two federated approaches, *FedCGASPD* results more often in smaller models than *FedGASPD*, as it tends to merge models of similar characteristics and, hence, preserve the distinctive sizes of the various groups of similar models. *FedGASPD*, however, tends to discover models of lower entropic relevance, hence describing the traces and their frequencies from the input event logs more accurately. This observation confirms the trade-off between the size and quality of the discovered models [14] and suggests that the two proposed methods prioritize different aspects of this trade-off.

We conclude that *FedGASPD* and *FedCGASPD* discover models of comparable characteristics as *GASPD*. Table 5 presents Pareto dominance counts for the various techniques. Note that the centralized solution, *GASPD*, does not consistently dominate across different datasets. Specifically, it delivers a good share of Pareto-optimal solutions for BPI-2017, BPI-2020-2, Sepsis Cases, and nasa-cev event logs, but not for other event logs, where *FedCGASPD* discovers the lion's share of Pareto-optimal models.

6 Conclusions

This paper presents the *FedGASPD* and *FedCGASPD* algorithms for stochastic process discovery in distributed environments. These algorithms are built upon the robust foundation of *GASPD*, a stochastic process discovery algorithm that operates over centralized event logs. The new algorithms discover process models from several local event

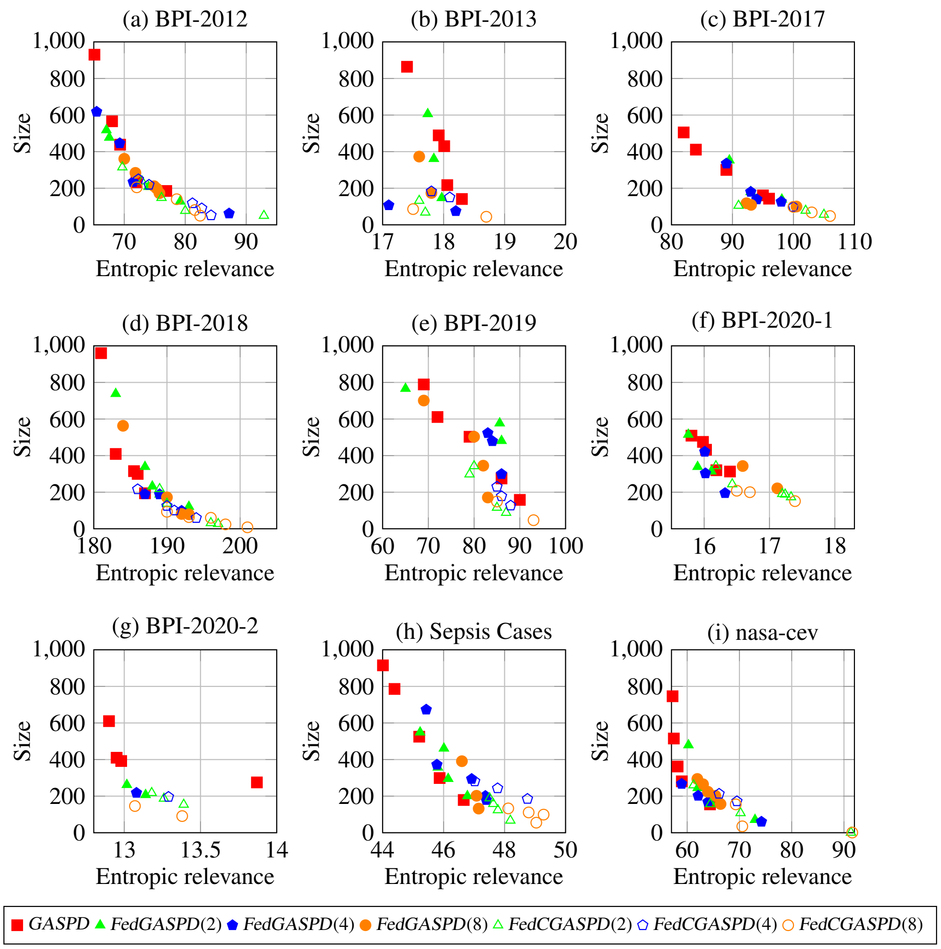

Fig. 8: Size and entropic relevance of DFFAs discovered by *GASPD*, *FedGASPD*, and *FedCGASPD*; smaller sizes and entropic relevance values signify better models. The numbers in the parenthesis signify the numbers of computation nodes.

logs, possibly scattered across different organizations, and aim to preserve the autonomy and privacy of each party and to decrease data communication requirements and the overall model discovery time. Our experiments with real-world event logs demonstrate the effectiveness of *FedGASPD* and *FedCGASPD*, providing scalable alternatives for process discovery in distributed environments. The algorithms discover DFFAs that can be translated, using the native translation of *GASPD*, into sound DFGs.

Although the presented results suggest that the models constructed by *FedGASPD* and *FedCGASPD* are comparable in quality to those discovered by *GASPD*, several avenues exist to further enhance the findings. First, a systematic exploration of the impact of different orders of merging DFFAs discovered from local event logs on the quality of the constructed global DFFAs and, consequently, DFGs, could provide valuable

Table 3: Size of DFFAs discovered by *GASPD*, *FedGASPD*, and *FedCGASPD*

Event Log	Size													
	GASPD		*FedGASPD*						*FedCGASPD*					
			2 nodes		4 nodes		8 nodes		2 nodes		4 nodes		8 nodes	
	min	max	min	max	min	max	min	max	min	max	min	max	min	max
BPI-2012	185	930	128	856	173	361	52	249	62	446	50	315	11	205
BPI-2013	486	760	143	715	174	373	130	291	107	311	67	218	47	190
BPI-2017	143	460	166	507	100	184	98	121	125	7500	56	154	48	121
BPI-2018	196	403	120	339	81	282	59	125	98	318	27	218	8	66
BPI-2019	158	808	481	766	171	503	127	230	229	632	116	572	47	218
BPI-2020-1	313	509	314	515	221	465	187	369	195	421	173	321	151	235
BPI-2020-2	275	629	234	459	247	362	196	283	213	345	153	266	90	241
Sepsis Cases	179	786	201	549	179	673	132	391	132	189	149	314	43	133
nasa-cev	155	746	71	514	60	576	157	294	1	260	172	241	1	155

Table 4: Relevance of DFFAs discovered by *GASPD*, *FedGASPD*, and *FedCGASPD*

Event Log	Entropic relevance													
	GASPD		*FedGASPD*						*FedCGASPD*					
			2 nodes		4 nodes		8 nodes		2 nodes		4 nodes		8 nodes	
	min	max	min	max	min	max	min	max	min	max	min	max	min	max
BPI-2012	65.11	76.89	64.84	79.13	70.00	75.17	72.32	84.23	65.47	87.19	72.91	92.90	72.05	98.00
BPI-2013	17.40	18.10	17.72	18.25	17.65	18.46	17.82	18.71	16.33	18.29	17.70	18.83	17.51	20.77
BPI-2017	82.01	93.34	89.79	95.42	92.63	103.04	100.88	105.45	89.69	100.18	93.88	106.57	101.31	110.33
BPI-2018	185.26	187.42	187.77	193.08	191.87	194.16	190.44	197.13	189.25	193.34	192.33	198.58	193.01	205.07
BPI-2019	61.01	87.27	64.77	87.39	69.51	88.59	70.65	89.66	76.40	88.69	81.56	92.22	83.39	94.73
BPI-2020-1	15.80	16.39	15.75	16.30	16.59	17.13	16.30	17.61	16.00	16.75	16.18	17.33	16.70	18.35
BPI-2020-2	12.95	13.78	13.07	13.47	13.09	13.85	13.40	13.53	13.06	13.25	13.17	13.44	13.40	13.75
Sepsis Cases	44.01	46.66	45.23	46.67	45.24	47.45	46.6	47.51	47.45	48.32	47.01	48.85	48.12	50.03
nasa-cev	57.21	64.37	60.25	72.98	58.91	74.21	62.95	66.39	61.23	91.95	66.11	69.52	67.91	91.59

Table 5: Pareto dominance counts

Event log	*GASPD*	*FedGASPD*			*FedCGASPD*			Total score
		2 nodes	4 nodes	8 nodes	2 nodes	4 nodes	8 nodes	
BPI-2012	2 (10.52%)	**4 (21.05%)**	2 (10.52%)	0 (00.00%)	3 (15.78%)	0 (00.00%)	**4 (21.05%)**	19 (100.00%)
BPI-2013	0 (00.00%)	0 (00.00%)	1 (25.00%)	0 (00.00%)	1 (25.00%)	0 (00.00%)	**2 (50.00%)**	04 (100.00%)
BPI-2017	**3 (30.00%)**	0 (00.00%)	0 (00.00%)	0 (00.00%)	**3 (30.00%)**	1 (10.00%)	**3 (30.00%)**	10 (100.00%)
BPI-2018	3 (15.00%)	1 (05.00%)	3 (15.00%)	3 (15.00%)	3 (15.00%)	3 (15.00%)	**4 (20.00%)**	20 (100.00%)
BPI-2019	2 (20.00%)	1 (10.00%)	0 (00.00%)	2 (20.00%)	**3 (30.00%)**	0 (00.00%)	2 (20.00%)	10 (100.00%)
BPI-2020-1	1 (10.00%)	2 (20.00%)	2 (20.00%)	0 (00.00%)	**3 (30.00%)**	1 (10.00%)	1 (10.00%)	10 (100.00%)
BPI-2020-2	**4 (57.14%)**	1 (14.28%)	0 (00.00%)	0 (00.00%)	0 (00.00%)	0 (00.00%)	2 (28.58%)	07 (100.00%)
Sepsis Cases	**5 (38.46%)**	2 (15.38%)	1 (07.69%)	1 (07.69%)	2 (15.38%)	0 (00.00%)	2 (15.38%)	13 (100.00%)
nasa-cev	**4 (28.57%)**	1 (07.14%)	3 (21.42%)	0 (00.00%)	3 (21.42%)	0 (00.00%)	3 (21.42%)	14 (100.00%)

insights. Similarly, investigating clustering techniques that improve the performance of *FedCGASPD* may lead to significant advancements. Additionally, optimizing the discovered models for other quality criteria than size and relevance could further refine their applicability. Finally, exploring alternative policies for selecting superior local models represents an interesting direction for future research.

References

[1] van der Aalst, W.M.P.: Decomposing Petri nets for process mining: A generic approach. Distributed Parallel Databases **31**(4), 471–507 (2013)

[2] van der Aalst, W.M.P.: Process Mining—Data Science in Action. Springer Berlin Heidelberg, 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.: Federated process mining: Exploiting event data across organizational boundaries. In: SMDS, pp. 1–7, IEEE (2021)
[5] van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. Transactions on Knowledge and Data Engineering **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, Springer (2024)
[7] Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. International Journal of Cooperative Information Systems **23**(01), 1440001 (2014)
[8] Carmona, J., Cortadella, J., Kishinevsky, M.: Divide-and-conquer strategies for process mining. In: BPM, LNCS, vol. 5701, pp. 327–343, Springer (2009)
[9] Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: Grammatical Inference and Applications, LNCS, vol. 862, pp. 139–152, Springer (1994)
[10] Esparza, J., Heljanko, K.: Unfoldings: A Partial-Order Approach to Model Checking. EATCS Monographs in Theoretical Computer Science, Springer (2008)
[11] de la Higuera, C.: Grammatical Inference. Cambridge University Press, Cambridge (2010)
[12] Khan, A., Ghose, A., Dam, H.K.: Cross-silo process mining with federated learning. In: ICSOC, LNCS, vol. 13121, pp. 612–626, Springer (2021)
[13] MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281–297, University of California Press (1967)
[14] Polyvyanyy, A., Moffat, A., Garcia-Banuelos, L.: An entropic relevance measure for stochastic conformance checking in process mining. In: ICPM, pp. 97–104, IEEE (2020)
[15] Polyvyanyy, A., Smirnov, S., Weske, M.: Process model abstraction: A slider approach. In: EDOC, pp. 325–331, IEEE Computer Society (2008)
[16] Rafiei, M., van der Aalst, W.M.P.: An abstraction-based approach for privacy-aware federated process mining. IEEE Access **11**, 33697–33714 (2023), ISSN 2169-3536
[17] Rojo, J., Garcia-Alonso, J., Berrocal, J., Hernández, J., Murillo, J.M., Canal, C.: SOWCompact: A federated process mining method for social workflows. Information Sciences **595**, 18–37 (2022)
[18] Sangiorgi, D.: On the origins of bisimulation and coinduction. ACM Trans. Program. Lang. Syst. **31**(4), 15:1–15:41 (2009)
[19] Yan, Z., Sun, B., Chen, Y., Wen, L., Hu, L., Wang, J., Yang, M., Wang, L.: Decomposed and parallel process discovery: A framework and application. Future Gener. Comput. Syst. **98**, 392–405 (2019)