Agent Miner: An Algorithm for Discovering Agent Systems from Event Data

Andrei Tour[1] , Artem Polyvyanyy[1] , Anna Kalenkova[2] , and
Arik Senderovich[3]

[1] The University of Melbourne, VIC, 3010, Australia
atour@student.unimelb.edu.au; artem.polyvyanyy@unimelb.edu.au
[2] The University of Adelaide, Adelaide, Australia
anna.kalenkova@adelaide.edu.au
[3] York University, Toronto, Ontario, Canada
sariks@yorku.ca

Abstract. Process discovery studies ways to use event data generated by business processes and recorded by IT systems to construct models that describe the processes. Existing discovery algorithms are predominantly concerned with constructing process models that represent the control flow of the processes. Agent system mining argues that business processes often emerge from interactions of autonomous agents and uses event data to construct models of the agents and their interactions. This paper presents and evaluates Agent Miner, an algorithm for discovering models of agents and their interactions from event data composing the system that has executed the processes which generated the input data. The conducted evaluation using our open-source implementation of Agent Miner and publicly available industrial datasets confirms that our algorithm can provide insights into the process participants and their interaction patterns and often discovers models that describe the business processes more faithfully than process models discovered using conventional process discovery algorithms.

1 Introduction

Process discovery is a subarea of process mining that studies ways to construct models that faithfully describe processes of a system based on event data the system has generated [26]. Constructed models assist analysts in understanding the system and, consecutively, deciding how to improve it. The state-of-the-art process discovery algorithms build models that describe the control flow of the processes. This focus on control flow has at least two limitations. Firstly, the resulting models are not well suited for analyzing the behavior of individual process participants and their interactions, as activities and interactions performed by a specific actor are often scattered across a discovered control flow model. Secondly, control flow models discovered from large data arrays are often too complex, the phenomenon known as *spaghetti models* [26]. Models of interacting agents, or *agent systems*, where agents are the process participants, address the former limitation by their very definition and, importantly, do *not* necessarily grow in complexity with the growth of the amount of data they represent [29].

Agent system mining is a type of process mining that studies ways to derive and use knowledge about systems composed of interacting agents based on the events these agents generate [24]. This paper presents and evaluates an agent system discovery algorithm. Concretely, this paper makes these contributions:

- It presents Agent Miner, a divide-and-conquer algorithm for discovering models of agents and their interactions from event data. The algorithm "divides" the input collection of events into several special subsets and "conquers" these subsets using conventional discovery algorithms, like Inductive Miner [11] and Split Miner [2], to construct an agent system that has generated the data.
- It presents the results of an evaluation of Agent Miner based on our open-source implementation of the algorithm that shows that Agent Miner can discover agent and interaction models from publicly available industrial event data that i) provide an additional perspective on the system that has generated the data and ii) often describe the processes more faithfully than the corresponding process models constructed using conventional control flow discovery algorithms; thus, addressing the two stated limitations.
- It demonstrates the value of agent system mining and invites the community to more intensive explorations of the agent-based paradigm in process mining.

The remainder of this paper proceeds as follows. The next section presents a motivating example. Then, Section 3 gives basic notions required for understanding the subsequent discussions. Section 4 presents our new discovery algorithm, while Section 5 discusses the results of its evaluation. Finally, Section 6 surveys related work before Section 7 gives concluding remarks on this work.

2   Motivating Example

As a motivating example, we propose a hypothetical health surveillance process sketched in Fig. 1. The process starts with a check (see the *check* activity in the figure) of the patient by doctor *d1*. If the analysis (*analyze*) of the results of the check indicates a risk of developing ill-health, the doctor prescribes (*prescribe*) medical tests and preventive therapy. The possible tests are blood test (*B-test*), ultrasound (*U-sound*), and *X-ray*. The therapy includes *yoga*, *physio*, *gym*, and swimming (*swim*). The tests and the therapy are performed independently; any subset of tests and exercises can be prescribed. Doctors *d2* and *d4* perform the tests, while the therapy is conducted by doctors *d3* and *d5*. Once the tests and the therapy are completed, doctor *d1* rechecks (*check*) the patient. If the new
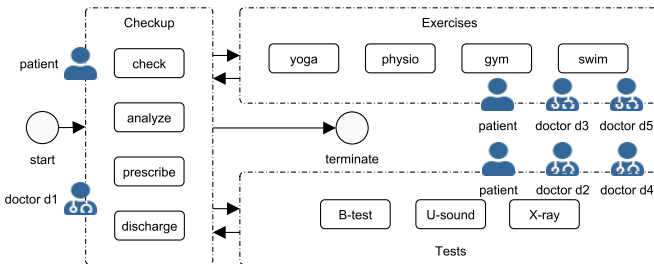

Fig. 1: A schematic visualization of the health surveillance process.

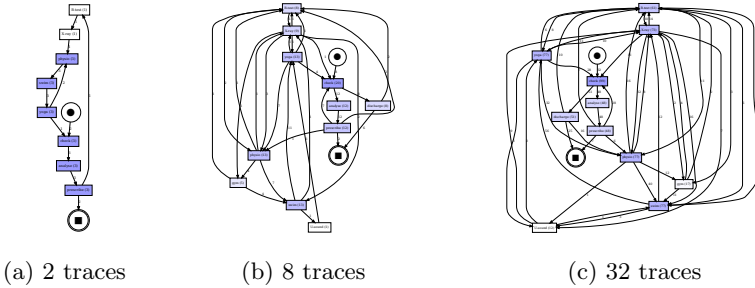(a) 2 traces   (b) 8 traces   (c) 32 traces

Fig. 2: DFGs that describe the health surveillance process (readability not intended).

check shows good results, the patient is discharged (*discharge*), and the process terminates; otherwise, further tests and therapy are prescribed. The information system that supports the process recorded a *log* of events that stem from managing 1 024 patients. Each recorded event has four attributes that specify the *timestamp* of the event occurrence, *activity* that triggered the event, the patient *case* the event relates to, and the doctor, or *agent*, that performed the activity.

Figure 2 shows three directly-follows graphs (DFGs) [26] that describe control flow dependencies between the activities of the health surveillance process discovered from 2, 8, and 32 traces from the log, where a *trace* is a sequence of all events with the same case attribute ordered by their timestamps. The complexity of the DFGs, defined as the number of nodes and arcs, grows as they represent more data, where the model in Fig. 2c is an example spaghetti model.

Figure 3a and Fig. 3b show the interaction net and one agent net that describes one of the three *agent types* discovered by Agent Miner from the events of the 32 traces used to construct the DFG in Fig. 2c captured as Petri nets [19]. Besides providing an alternative, modular perspective on the process, that is, an explicit representation of process participants and their interactions, these models, similar to the DFGs in Fig. 2, describe control flow dependencies between the activities. These dependencies can be captured explicitly in an integrated Petri net, called the Multi-Agent System (MAS) net, obtained by refining labeled transitions of the interaction net with the corresponding agent nets. Interestingly, this MAS net is smaller and represents the traces encoded in the event data more faithfully than the Petri net constructed from the same 32 traces using Inductive Miner (IM) [11]—a conventional process discovery algorithm. The MAS
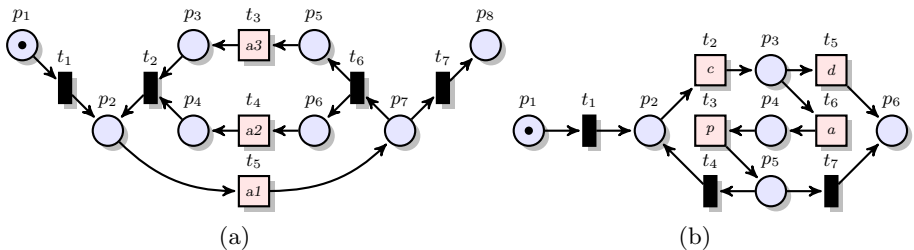

(a)   (b)

Fig. 3: Petri nets that describe: (a) interactions of the three agent types *a1* (doctor *d1*), *a2* (*d2* and *d4*), and *a3* (*d3* and *d5*) and (b) agent type *a1* from the health surveillance process with transition labels check (*c*), analyze (*a*), prescribe (*p*), and discharge (*d*).
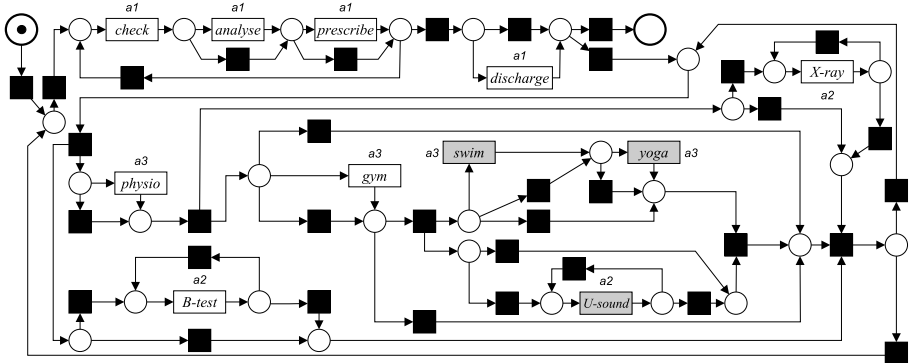
Fig. 4: The IM net discovered from 32 traces of the example health surveillance process.

net has 155 nodes and arcs, while the IM net shown in Fig. 4 has 175 nodes and arcs. The precision of the MAS net and the IM net is 0.37 and 0.16, respectively. To support model comparison, both algorithms were configured to discover perfectly fitting models (recall of 1.0). A model has good precision if it does not replay traces not recorded in the log and a good recall if it replays many traces from the log, where the values closer to 1.0 indicate better models. Precision and recall were measured for the 32 traces used to construct the models using the entropy-based approach [16,17], the only existing measures that guarantee that better discovered models result in better measurements [23].

A MAS net often does not require an increase in size to represent more data well due to an ability of an agent system to simulate the non-decreasing complexity of the behavior of a system [24,29]. For instance, the same MAS net we discovered from the 32 traces has precision of 0.43 and recall of 1.0 if measured for all 1 024 traces. This example confirms that Agent Miner can construct modular, fitting, and precise models of process participants and their interactions. Finally, while the interaction and agent nets enable dedicated analyses of the respective artifacts, like identification of repetitive work handovers and verification of safeness and liveness of individual agent nets, the information about agents is scattered in the IM net, see the *U-sound* activity by agent *a2* amidst activities *swim* and *yoga* by agent *a3* in Fig. 4, which hinders the analysis of individual agents and their interactions. The input log, models, and measurements discussed in this section are publicly available [25].

3   Preliminaries

This section introduces Petri nets (Section 3.1) and event logs (Section 3.2).

3.1   Petri Nets and Workflow Nets

Petri nets formalism suits well for describing models of distributed systems [19].

Definition 3.1 (Petri nets)

A *(labeled) Petri net*, or a *net*, N is a quintuple $(P, T, F, \Lambda, \lambda)$, where P is a finite set of *places*, T is a finite set of *transitions*, $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*, Λ is a set of *labels*, such that $\tau \in \Lambda$ is the *silent label* and sets P, T, and Λ are pairwise disjoint, and $\lambda : T \to \Lambda$ is the *labeling function*.   ⌐

If $\lambda(t) = \tau$, $t \in T$, t is *silent*; otherwise t is *observable*. Observable transitions represent activities from the problem domain, and silent transitions encode internal actions of the system. A *marking M* of a net encodes its state and is a multiset over its places. Figure 3a shows a Petri net $N = (P, T, F, \Lambda, \lambda)$, with eight places ($P = \{p_1, \dots, p_8\}$) and seven transitions ($T = \{t_1, \dots, t_7\}$). In the graphical notation, places are drawn as circles, while transitions as squares or rectangles. Transitions t_1, t_2, t_6, and t_7 are silent, shown as black rectangles. The labeling function assigns labels *a1*, *a2*, and *a3* to transitions t_5, t_4, and t_3, respectively. The flow relation is shown as directed arcs. A marking is denoted as an arrangement of black dots, called *tokens*, inside of the corresponding places. Marking M shown in Fig. 3a is the multiset $[p_1]$; see one black dot in place p_1.

Let $n \in P \cup T$ be a place or transition, then by $\bullet n = \{x \in (P \cup T) \mid (x, n) \in F\}$ we denote its *preset* and by $n\bullet = \{x \in (P \cup T) \mid (n, x) \in F\}$ we denote its *postset*. Let $N = (P, T, F, \Lambda, \lambda)$ be a net. A transition $t \in T$ is *enabled* in a marking M of N, denoted by $(N, M)[t\rangle$, if every *input place* of t contains at least one token, i.e., $\forall p \in \bullet t : M(p) > 0$; by $M(p)$ we denote the multiplicity of p in M. An enabled transition $t \in T$ can occur. An *occurrence* of t leads to a fresh marking $M' = (M \setminus \bullet t) \uplus t\bullet$ of N, denoted by $(N, M)[t\rangle(N, M')$.

Workflow nets are special nets used for modeling workflows [26]. A *workflow net* is a Petri net $(P, T, F, \Lambda, \lambda)$ with a dedicated *initial place $i \in P$*, $\bullet i = \emptyset$, a dedicated *final place $f \in P$*, $f\bullet = \emptyset$, and every place and transition on the vertex sequence of some directed walk from i to f in graph $(P \cup T, F)$. Marking $[i]$ is the initial marking of a workflow net. A workflow net is *safe* if every marking reachable from $[i]$ via a sequence of transition occurrences is a set. It is *sound* if every transition of the net can occur in some sequence of transition occurrences that starts in $[i]$, every sequence of transition occurrences that starts in $[i]$ can be extended to "put" a token in the final place, and once there is a token in the final place no other places hold tokens [26]. Figure 3a and Fig. 3b show safe and sound workflow nets with final markings $[p_8]$ and $[p_6]$, respectively.

3.2 Events, Event Logs, and Traces

An *event log*, or *log*, represents real-world processes recorded by an information system. In process mining, events are often organized into time-ordered sequences, called *traces*. As explained in Section 2, in our work, each event has at least four attributes: *timestamp*, *activity*, *case*, and *agent* (either instance or type). Let $\mathcal{A}$ be the universe of attribute names, with $\{timestamp, activity, case, agent\} \subseteq \mathcal{A}$. Let $\mathcal{V}$ be the universe of attribute values. Then, an *event* is an attribute function $e : \mathcal{A} \to \mathcal{V}$ that maps attribute names to attribute values. By $\mathcal{E}$ we denote the universe of events. An *event se-*

Table 1: An event selection.

event	timestamp	case	activity	agent
e^a	30 Mar 16:34	*case1*	*check*	*d1/a1*
e^b	30 Mar 16:35	*case1*	*analyze*	*d1/a1*
e^c	31 Mar 16:35	*case2*	*check*	*d1/a1*
e^d	01 Apr 08:22	*case2*	*analyze*	*d1/a1*
e^e	01 Apr 16:05	*case1*	*prescribe*	*d1/a1*
e^f	03 Apr 11:55	*case1*	*B-test*	*d4/a2*
e^g	03 Apr 16:59	*case1*	*X-ray*	*d4/a2*
e^h	06 Apr 10:02	*case1*	*physio*	*d3/a3*
e^i	06 Apr 11:01	*case1*	*swim*	*d3/a3*
e^j	07 Apr 15:55	*case1*	*yoga*	*d3/a3*
e^k	07 Apr 11:11	*case1*	*physio*	*d3/a3*
e^l	10 Apr 13:13	*case1*	*swim*	*d3/a3*
e^m	10 Apr 15:05	*case1*	*yoga*	*d3/a3*
e^n	11 Apr 09:12	*case1*	*physio*	*d3/a3*
e^o	11 Apr 10:05	*case1*	*swim*	*d3/a3*
e^p	13 Apr 11:03	*case1*	*yoga*	*d3/a3*
e^q	13 Apr 14:57	*case1*	*check*	*d1/a1*
e^r	16 Apr 12:11	*case1*	*analyze*	*d1/a1*
e^s	17 Apr 10:03	*case1*	*prescribe*	*d1/a1*
e^t	17 Apr 16:36	*case2*	*prescribe*	*d1/a1*

lection $S \subseteq \mathcal{E}$ is a finite set of events. Without loss of generality, we assume that events have unique timestamps. Table 1 defines event selection $S = \{e^a, \ldots, e^t\}$. Each row of the table describes one event. For example, $e^t = \{(\mathit{timestamp}, 17 \text{ Apr } 16:36), (\mathit{activity}, \mathit{prescribe}), (\mathit{case}, \mathit{case2}), (\mathit{agent}, \mathit{a1})\}$ is the event from the last row of Table 1; the agent attribute specifies agent type.

Let X be a finite non-empty set. A *partition* of X is a set Π of disjoint subsets of X such that the union of the subsets equals X; the subsets are *parts* of Π. Partition Π can be defined by an equivalence relation $\sim \subseteq X \times X$ such that if two events $e_i, e_j \in X$ are equivalent under $\sim$, that is, it holds that $(e_i, e_j) \in \sim$, then e_i and e_j belong to the same part of Π. We use notation $\Pi = X/\sim$ to denote that partition Π is defined by the equivalence relation $\sim$ over X. Against this background, we define a trace of an event selection as follows.

Definition 3.2 (Traces induced by partitions)
The *trace* σ of event selection S induced by part π of partition $\Pi = S/\sim$ is the ordered by timestamps sequence of all and only events in π.   ⌟

We refer to Π and $\sim$ as the *trace partition* and the *trace relation* that induce σ. The *trace set* of S induced by $\Pi = S/\sim$ is the set of traces Σ that for each part π of Π contains the trace of S induced by π, and contains no other traces.

Definition 3.3 (Event logs)
An *event log*, or *log*, of event selection S induced by partition $\Pi = S/\sim$ is a triple (S, Σ, ν), where $\Sigma \subset S^*$ is the trace set induced by Π, and $\nu : S \to \mathcal{V}$ is a naming function that assigns names to events in S.   ⌟

Multiple event logs induced by different partitions and naming functions can be defined. For example, in classical process discovery, traces are induced by case attributes. Let $\mathcal{C}$ be a universe of cases. The *case trace set* of event selection S is the trace set of S defined by relation $\sim^{\mathbf{c}} = \{(e_i, e_j) \in S \times S \mid e_i(\mathit{case}) = e_j(\mathit{case})\}$, denoted by $\Sigma^{\mathbf{c}}$; we refer to traces in $\Sigma^{\mathbf{c}}$ as *case traces*. For instance, the case trace set of the event selection in Table 1 consists of two case traces, namely $\langle e^a, e^b, e^e, e^f, e^g, e^h, e^i, e^j, e^k, e^l, e^m, e^n, e^o, e^p, e^q, e^r, e^s \rangle$ and $\langle e^c, e^d, e^t \rangle$.

The naming function of an event log specifies the names of the events used by the discovery algorithms to identify activity names in the constructed process models. In process discovery, events are often identified by their activity attributes, that is, $\nu = \{(e, e(\mathit{activity})) \mid e \in S\}$. In general, other naming functions can be used. We will use this feature in the subsequent sections.

4   Agent Miner

In this section, we introduce the core notions required to define the Agent Miner algorithm (Section 4.1) and present the algorithm (Section 4.2).

4.1   Agent Logs and Nets

A trace of an *agent trace set* is a sequence of events that refer to the same case, are performed by the same agent (identified by the agent attribute), and are not interrupted by an event from the same case performed by a different agent.

Definition 4.1 (Agent trace sets)
The *agent trace set* of event selection S is the trace set of S induced by the partition of S defined by relation $\sim = \big\{ (e_i, e_j) \in S \times S \mid (e_i(agent) = e_j(agent)) \wedge (e_i(case) = e_j(case)) \wedge (\neg\exists\, e_k \in S : (((e_i(timestamp) < e_k(timestamp) < e_j(timestamp)) \vee (e_j(timestamp) < e_k(timestamp) < e_i(timestamp))) \wedge (e_i(agent) \neq e_k(agent) \wedge e_i(case) = e_k(case)))) \big\}$. ⌟

The set $\{\langle e^a, e^b, e^e \rangle, \langle e^c, e^d, e^t \rangle, \langle e^f, e^g \rangle, \langle e^h, e^i, e^j, e^k, e^l, e^m, e^n, e^o, e^p \rangle, \langle e^q, e^r, e^s \rangle\}$ is the agent trace set of the event selection in Table 1. For instance, in trace $\langle e^a, e^b, e^e \rangle$, all events are from *case1*, performed by agent *a1*, and, though interrupted by events e^c and e^d, the latter events are from a different case. Note that, by definition, relation $\sim$ from Definition 4.1 is an equivalence relation.

Next, we define several useful logs. Traces of an interaction log are composed of events that allow identifying all handovers of work between agents.

Definition 4.2 (Interaction logs)
The *interaction log* of event selection S is the triple $\big(\bar{S}, \Sigma, \nu\big)$, where $\bar{S}$ is the *agent event selection* composed of the first events of all the traces in the agent trace set Δ of S, that is, $\bar{S} = \{\delta(1) \mid \delta \in \Delta\}$, Σ is the case trace set of $\bar{S}$, and $\nu = \big\{(e, e(agent)) \mid e \in \bar{S}\big\}$ names each event by the corresponding agent. ⌟

The interaction log of the event selection from Table 1 is, therefore, the event log $\big(\bar{S}, \Sigma, \nu\big)$, where $\bar{S} = \{e^a, e^c, e^f, e^h, e^q\}$, $\Sigma = \{\langle e^a, e^f, e^h, e^q \rangle, \langle e^c \rangle\}$, and $\nu = \big\{(e^a, a1), (e^c, a1), (e^f, a2), (e^h, a3), (e^q, a1)\big\}$. For instance, trace $\langle e^a,\ e^f, e^h, e^q \rangle$ in Σ suggests that agent *a1* starts *case1*; note that $e^a(agent) = a1$ and $e^a(case) = case1$. Then, agent *a2* takes over the work on *case1*. Agent *a2* then passes the work on *case1* to agent *a3*, who later hands work back to agent *a1*.

Traces in the agent trace set done by the same agent compose its agent log.

Definition 4.3 (Agent logs)
The *agent log* of agent a and event selection S is the triple (S^a, Σ, ν), where $S^a = \{e \in S \mid e(agent) = a\}$, Σ is the set of traces in the agent trace set Δ of S performed by a, that is, $\Sigma = \{\delta \in \Delta \mid \forall\, i \in [1 .. |\delta|] : \delta(i)(agent) = a\}$, and $\nu = \{(e, (a, e(activity))) \mid e \in S^a\}$ is the naming function that names an event by the pair comprising its agent and activity attributes. ⌟

The agent log of agent *a1* and the event selection from Table 1 is defined by the triple $\big(S^{a1}, \Sigma, \nu\big)$, where $S^{a1} = \{e^a, e^b, e^c, e^d, e^e, e^q, e^r, e^s, e^t\}$, $\Sigma = \{\langle e^a, e^b, e^e \rangle, \langle e^c, e^d, e^t \rangle, \langle e^q, e^r, e^s \rangle\}$, and ν maps events in S^{a1} to their names and contains, for instance, it holds that $\big\{(e^a, (a1, check)), (e^d, (a1, analyze))\big\} \subset \nu$.

Next, we discuss several classes of workflow nets used by Agent Miner. An interaction net describes the structure of interactions between agents.
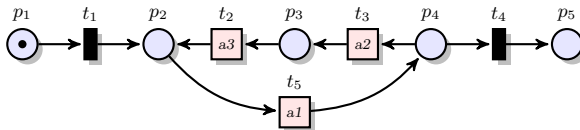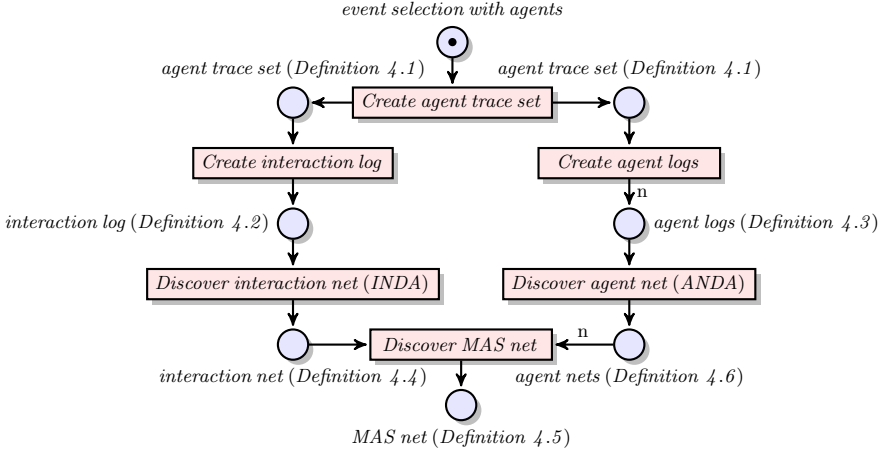

Fig. 5: An i-net.

Fig. 6: The Agent Miner algorithm.

Definition 4.4 (Interaction nets)
An *interaction net*, or an *i-net*, of event selection S is a workflow net $(P, T, F, \Lambda, \lambda)$, where $\Lambda = \{a \in \mathcal{V} \mid \exists e \in S : e(agent) = a\} \cup \{\tau\}$. ⌟

Figures 3a and 5 show two i-nets of the event selection from Table 1. They describe alternative ways the agents could have interacted to generate the event data. For example, the i-net in Figure 5 suggests that agent *a1* starts the interaction, and then any number of sequences of interactions of *a1* with agent *a2*, then of *a2* with agent *a3*, and finally of *a3* again with agent *a1* can occur.

A MAS net describes how agents perform activities in a collaborative setting.

Definition 4.5 (Multi-Agent System nets)
A *Multi-Agent System (MAS) net* of event selection S is a workflow net $(P, T, F, \Lambda, \lambda)$, with $\Lambda = \{(a, b) \in \mathcal{V} \times \mathcal{V} \mid \exists e \in S : e(agent) = a \wedge e(activity) = b\} \cup \{\tau\}$. ⌟

A MAS net of events of a single agent is an agent net of this agent.

Definition 4.6 (Agent nets)
A MAS net of event selection S such that the agent attribute of every event in S is equal to a, that is, $\forall e \in S : e(agent) = a$, is an *agent net* of a. ⌟

4.2 Algorithm

Figure 6 defines the Agent Miner algorithm as a workflow net. It is parameterized by two conventional control flow discovery algorithms: an Agent Net Discovery Algorithm (ANDA) and an Interaction Net Discovery Algorithm (INDA). Agent Miner takes an event selection as input and produces an interaction net, agent nets, and a MAS net. Similar to standard process discovery algorithms, the latter explains how the discovered interaction and agent nets represent the traces identified by the case attribute. The algorithm has six steps detailed below.

Create agent trace set. The agent trace set of the input event selection is created by associating each event with an agent trace, see Definition 4.1.
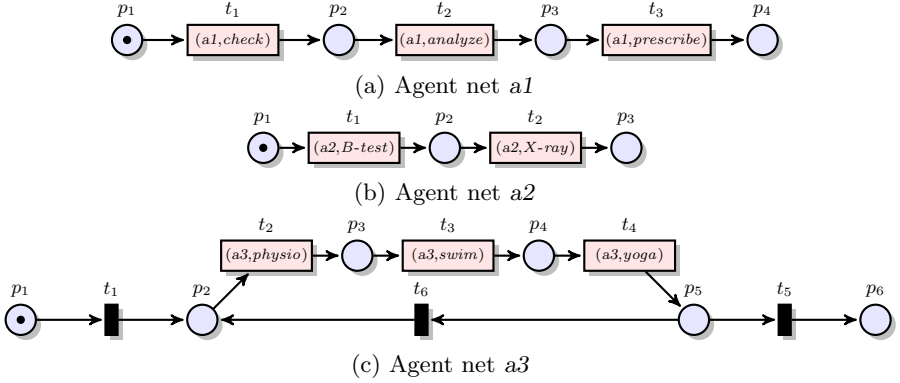
(a) Agent net *a1*

(b) Agent net *a2*

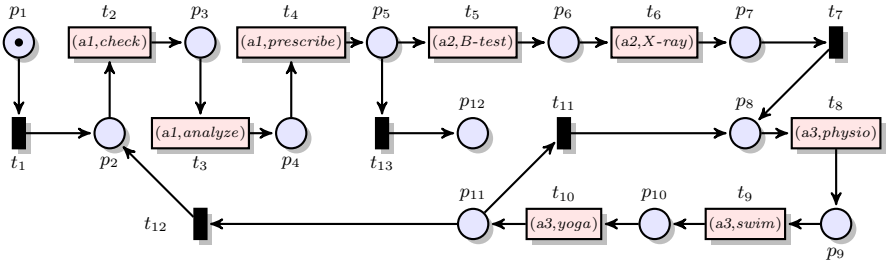(c) Agent net *a3*

Fig. 7: Three agent nets.

Fig. 8: A MAS net.

Create interaction log. The interaction log is created by selecting the first events of all traces in the agent trace set and constructing the case trace set of this event selection, see Definition 4.2.

Discover interaction net. In this step, INDA is used to discover an i-net, see Definition 4.4, from the interaction log. For example, the i-net in Fig. 5 was discovered from the interaction log of the event selection in Table 1.

Create agent logs. Multiple (n) agent logs are created, one for each agent encountered in the input event selection, by extracting the corresponding agent traces from the agent trace set, see Definition 4.3.

Discover agent net. In this step, ANDA is used to discover n agent nets (one net is discovered from each agent log), see Definition 4.6. Figure 7 shows three agent nets discovered from the agent logs of the event selection in Table 1.

Discover MAS net. Finally, a MAS net, see Definition 4.5, is constructed by "embedding" the agent nets into the i-net and applying the Fusion of Series Places reduction [14] to the refined i-net. Fig. 8 shows the embedding of the agent nets in Fig. 7 into the i-net in Fig. 5. The embedding is performed by refining each observable transition in the i-net with the corresponding agent net. The resulting MAS net describes how agents (doctors) interact to support the execution of cases (treatment of patients) in the health surveillance process introduced in Section 2 discovered based on the event selection from Table 1.

Thus, Agent Miner is a divide-and-conquer algorithm that "divides" the input event selection into interaction and agent logs and "conquers" these logs

using ANDA and INDA. In this work, as ANDA, we used DFG translation to Petri nets (DFG-PN) [12], while as INDA, we used Inductive Miner (IM) [11] and Split Miner (SM) [2] and removed iterations of observable transitions in the obtained i-nets. This approach follows the agent system paradigm, where agents are sequential machines, and parallelism emerges through collaborations of agents. If the algorithms guarantee that the constructed models are safe and sound workflow nets, which holds for DFG-PN and IM, every constructed MAS net is guaranteed to be a safe and sound workflow net (cf. Theorem 2 in [18]).

5   Evaluation

The Agent Miner algorithm (Figure 6) and its evaluation pipeline (Figure 9), including the code, the discovered models, and detailed results, are publicly available [25]. This section presents and discusses the design (Section 5.1) and results (Section 5.2) of an evaluation of Agent Miner using real-world datasets.

5.1   Design

Figure 9 presents our five-step evaluation pipeline. We executed this pipeline for each real-world dataset used in this evaluation. The steps are explained below.
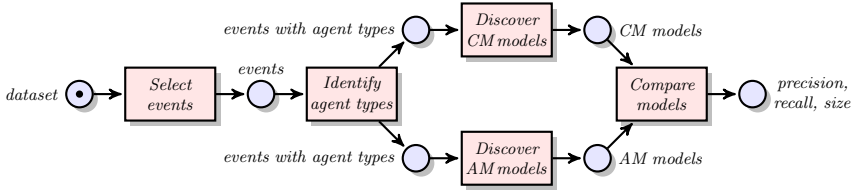

Fig. 9: Evaluation pipeline.

Select events. In this step, we select events from the dataset to ensure events have required attributes and remove infrequent case traces. Specifically, every event must have a *case*, *timestamp*, *activity*, and *agent* attribute. Moreover, each selected event must be part of a frequent case variant (a trace within *vff%* of the most common case traces, where *vff* is the *variant frequency filter* parameter).

Identify agent types. For the event selection, we identify agent instances using agent attribute values of events and group them into agent types by clustering them based on distances between agent instance DFGs, where an agent instance DFG is constructed from the corresponding agent log (Definition 4.3). For a pair of agents (a_1, a_2), we calculate the distance between the DFGs of a_1 and a_2 as:

$$1 - \max(|DF_{a_1} \cap DF_{a_2}|/|DF_{a_1}|, |DF_{a_1} \cap DF_{a_2}|/|DF_{a_2}|),$$

where DF_{a_1} and DF_{a_2} are the directly-follows relations (sets of edges) in the corresponding DFGs. Thus, the more one DFG subsumes the directly-follows relations of the other DFG, the more likely the corresponding agent instances belong to the same agent type. Once the agent types are identified, the agent attribute values of all the events in the input event selection are updated from instances to types. For example, this step leads to the identification of three

agent types *a1* (doctor *d1*), *a2* (doctors *d2* and *d4*), and *a3* (doctors *d3* and *d5*) in the motivating example from Section 2.

Discover CM models. We refer to a discovery algorithm based on the conventional process mining paradigm as a Conventional Miner (CM) and a model discovered by a CM algorithm as a CM model. We use two state-of-the-art CM algorithms, Inductive Miner IMf variant (IM) [11] and Split Miner (SM) [2], to construct CM models from the case traces induced by the event selection, that is, traces identified by the case attributes, see Definition 3.3. Given an event selection S, to discover CM models, we use two naming functions: i) activity-only labeling (AOL) defined by $\nu_{aol} = \{(e, e(activity)) \mid e \in S\}$ and ii) agent and activity labeling (AAL) defined by $\nu_{aal} = \{(e, (e(agent), e(activity))) \mid e \in S\}$. Thus, for each event dataset, we construct four CM models. For example, the model in Fig. 4 is the IM model constructed using the ν_{aol} naming function. Hence, agent attributes of events were not used to construct the model, and then observable transitions were annotated with agent info. For each CM algorithm and naming function, we construct ten CM models, one for each configuration of the *threshold* parameter of the algorithm, *noise threshold* for IM and *frequency threshold* for SM (in the range from 0.0 to 0.9 in 0.1 increments). This approach ensures that the discovered CM models show a range of quality characteristics.

Discover AM models. For the event selection over agent types, we use the Agent Miner (AM) algorithm (refer to Section 4) to discover agent nets, the model of agent interactions, and the MAS net that defines the semantics of the resulting agent systems. DFG translation to Petri nets (DFG-PN) [12] algorithm is used as an ANDA. Inductive Miner (IM) [11] and Split Miner (SM) [2] are used as INDAs. For each configuration of ANDA and INDA, we run AM ten times with ten different parameter pairs (ff_i, th_i), with $i \in [1 \mathinner{.\,.} 10]$, $ff_i = i \times 0.1$, and $th_i = 1 - i \times 0.1$. Parameter ff_i is the *activity frequency filter* parameter used by the ANDA. Parameter th_i is the *threshold* parameter used by the INDA, *noise threshold* for IM and *frequency threshold* for SM. The lower the value of i, the more filtered the event selection is and the less the discovered models are fit to the input data.

Compare models. To compare the discovered CM and AM models, we calculate their *recall* and *precision* with respect to the case trace set of the original event selection, and *size* (as the number of nodes and arcs). To compute precision and recall, we use the entropy-based measures [16,17]. These measures fulfill all the desired properties for these classes of measures [23]. For example, the entropy-based precision measure guarantees that a model that describes fewer traces not in the event log has a better precision score. For CM models constructed using AOL and AAL naming functions of events, we measure their precision and recall with respect to the event logs that identify events using the corresponding naming functions. Note that labels of observable transitions of AM models are composed of agent-activity pairs, see, for example, the MAS net in Fig. 8. Hence, to compare MAS nets to CM models constructed using the AOL naming function, we also measure precision and recall of MAS nets after rewriting their transition labels to only mention activity names.

Table 2: Size, recall, and precision of IM nets and MAS nets discovered by Agent Miner (Inductive Miner as INDA) that rely on activity-only labeling of observable transitions.

BPIC dataset	Variant frequency filter (*vff*%)	Inductive Miner						Agent Miner (MAS nets)					
		lowest size			greatest precision			lowest size			greatest precision		
		size	recall	prec.	size	recall	prec.	size	recall	prec.	size	recall	prec.
2011	10%	592	0.72	**0.41**	592	0.72	**0.41**	**466**	**0.92**	0.35	**466**	**0.92**	0.35
2012	80%	420	**1.00**	0.06	454	**0.88**	**0.32**	**333**	0.80	**0.18**	**333**	0.80	0.18
2013	80%	**54**	**0.70**	0.53	105	**0.78**	0.62	69	0.62	0.64	**69**	0.62	**0.64**
2014	10%	**229**	**0.85**	0.28	335	0.55	**0.45**	231	0.78	**0.36**	**231**	**0.78**	0.36
2015	80%	199	**0.95**	**0.35**	199	**0.95**	**0.35**	**122**	0.95	0.34	**122**	0.95	0.34
2017	80%	**216**	**0.85**	0.22	**241**	0.91	**0.26**	340	0.76	0.15	597	**0.94**	0.15
2018	10%	**350**	**0.88**	0.13	**350**	**0.88**	0.13	535	0.85	0.09	535	0.86	0.09
2019	10%	270	0.61	0.24	375	0.54	**0.46**	**191**	**0.65**	**0.34**	**191**	**0.65**	0.34
2020	80%	204	0.72	0.18	220	0.69	**0.25**	**200**	**0.75**	**0.19**	**200**	**0.75**	0.19

5.2   Datasets and Results

To evaluate Agent Miner, we used publicly available real-world Business Process Intelligence Challenge (BPIC) datasets and assumed they stem from agent systems. These datasets are widely used to evaluate conventional process discovery algorithms. We assumed that the resource attributes of events specify agent instances that triggered them and selected all BPIC datasets that specify events with resource attributes, leading to nine selected datasets.[1]

Initially, we used the *vff*% parameter of 80% and completed the evaluation pipeline (cf. Fig. 9) for datasets BPIC 2012, 2013, 2015 (Municipality 1), 2017, and 2020 (Travel Permit Data). Due to performance reasons, to process the other datasets, we lowered the *vff*% parameter to 10%. Table 2 summarizes the quality measurements for workflow nets discovered using Inductive Miner and MAS nets constructed by Agent Miner, using activity-only labeling (AOL) of transitions in the discovered nets. For each dataset, the table lists size, recall, and precision values for the models that scored the lowest size and the greatest precision.

These results confirm that the quality of the MAS nets discovered by Agent Miner is comparable to the quality of CM models. This observation is remarkable for at least two reasons. First, as stated above, we had no background knowledge of whether the datasets stem from agent-driven business processes. Still, for most datasets, we discovered interesting (in terms of size, recall, and precision) MAS nets. Second, in addition to high-quality MAS nets, Agent Miner constructs agent nets and i-nets that can be used as a starting point for analysis and improvement of ways the process participants work individually and together.

To support the above conclusions, Fig. 10 shows three types of trade-offs as two-dimensional Pareto fronts for the nets discovered from the BPIC 2015 dataset. Each point in the plots denotes two quality measurements for one net generated by the evaluation pipeline. The Pareto fronts indicate the nets with

---

(a) recall/prec. trade-off   (b) size/prec. trade-off   (c) size/recall trade-off

Fig. 10: BPIC 2015 Pareto fronts: IM–AOL nets ("●") and AM–AOL MAS nets ("○").

(a) recall/prec. trade-off   (b) size/prec. trade-off   (c) size/recall trade-off

Fig. 11: BPIC 2013 Pareto fronts: IM–AOL nets ("●") and AM–AOL MAS nets ("○").

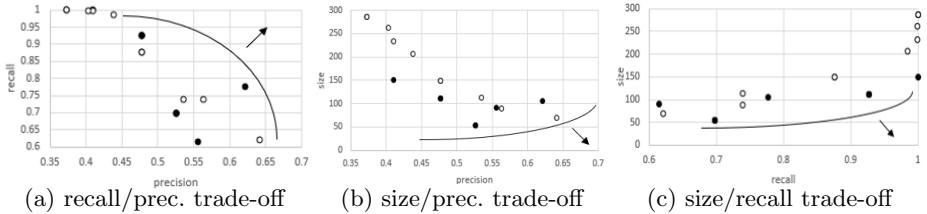better quality measurements. The MAS nets discovered by Agent Miner complement workflow nets discovered by Inductive Miner to result in more saturated Pareto fronts, that is, fronts with more models of interesting qualities. For the BPIC15 dataset, MAS nets 1 to 8 discovered by Agent Miner belong to the recall/precision Pareto front and demonstrate combinations of these quality measurements better than most CM models discovered by Inductive Miner. MAS net 8 belongs to the size/precision Pareto front and is better in terms of size/precision than most of the CM models. MAS nets 1 to 5, 7, and 8 demonstrate a better combination of size/recall measurements than almost all CM models. Overall, the Pareto fronts contain more AM models than CM models, except in the size/precision case, when the front is represented by one AM model and one CM model. Similar to the BPIC 2015 results, the Pareto fronts for the BPIC 2013 dataset, presented in Fig. 11, include points for the MAS nets. The Pareto front plots for all the datasets are included in the evaluation results [25].

Agent Miner uses an additional event attribute that specifies the *agent* that triggered the event. Hence, it is reasonable to expect it to construct comparable or better models than conventional algorithms, which do not require this attribute. To obtain generalizable conclusions, for all the datasets, we analyzed Pareto fronts and performed paired samples t-tests to establish whether CM and AM models are of the same qualities, for CM models discovered using both activity-only labeling (AOL) and agent and activity labeling (AAL). When measuring precision and recall of AOL and AAL models, both AM and CM models, events in the logs were identified correspondingly. The null hypotheses used in the t-tests are such that means of quality measurements for AM and CM models are equal. Table 3 summarizes comparisons of Pareto fronts for IM nets and AM nets (IM as INDA), while Table 4 shows the results of the tests, where results are for size (s), recall (r) and precision (p) measurements. In Table 3, AM, CM, and AM&CM entries stand for situations when the Pareto front is composed of AM only, CM only, or AM and CM models, respectively. In Table 4, for the

Table 3: Comparison of Pareto fronts.

BPIC data	AM vs CM fronts (AOL)			AM vs CM fronts (ALL)		
	r/p	s/p	s/r	r/p	s/p	s/r
2011	AM	AM&CM	AM	AM&CM	CM	AM&CM
2012	AM&CM	AM&CM	AM&CM	AM&CM	AM&CM	AM
2013	AM&CM	AM&CM	AM&CM	AM&CM	AM&CM	AM&CM
2014	AM&CM	AM&CM	CM	AM&CM	AM&CM	AM&CM
2015	AM&CM	AM&CM	AM&CM	AM&CM	CM	CM
2017	AM&CM	CM	CM	AM&CM	AM&CM	CM
2018	CM	CM	CM	CM	CM	CM
2019	AM&CM	AM&CM	AM&CM	CM	CM	AM&CM
2020	AM&CM	AM&CM	AM&CM	AM&CM	AM&CM	AM&CM

Table 4: Results of t-tests.

BPIC data	AM vs CM tests (AOL)			AM vs CM tests (ALL)		
	s	r	p	s	r	p
2011	CM	CM	CM	CM	AM	AM&CM
2012	CM	CM	AM&CM	AM&CM	AM	AM&CM
2013	CM	AM&CM	AM&CM	AM&CM	AM	AM&CM
2014	CM	CM	AM&CM	CM	AM	AM&CM
2015	CM	AM&CM	AM&CM	CM	AM	AM&CM
2017	CM	AM&CM	AM&CM	AM&CM	AM&CM	AM&CM
2018	CM	CM	AM&CM	AM&CM	AM&CM	AM&CM
2019	CM	CM	AM	CM	AM	AM&CM
2020	CM	CM	AM&CM	CM	AM	CM

significance level of 0.05, AM, CM, and AM&CM entries stand for situations when AM models are significantly better, CM models are significantly better, and the null hypothesis was not rejected, respectively. We accept the AM and AM&CM entries as situations when Agent Miner provides additional value to CM models and highlight them in bold (the majority of the entries).

The results confirm that Agent Miner discovers interesting models to complement models constructed by conventional discovery algorithms that also rely on event attributes that specify agents that triggered the events and describe this information in the constructed models. The results for Split Miner, CM models, and AM models (Split Miner as INDA), demonstrate similar conclusions as for IM models presented above.

6   Related Work

This section reviews business process modeling with agents, process discovery addressing agents, and agent system modeling in a broad context.

Agent Modeling in Business Process Management. The relationship between business process management and agent-based modeling was first explored in the '90s by Jennings et al. [7]. In an extension of their work, dubbed ADEPT, the authors propose to model a business process as a negotiation system between agents, similar to our interaction nets [8]. However, their model of negotiating agents is conceptual in its nature, while our model is formal and executable. Several authors advocate for an agent-based perspective on business process modeling [6,22]. In these works, the process is seen as such composed of interacting agents [7]. However, they do not provide automated discovery of models from data.

Process Discovery. Traditional process discovery techniques assume a case perspective when learning models from data while often ignoring additional perspectives such as resources. Several process discovery techniques extended traditional methods beyond the case perspective. Rozinat et al. [20] propose a multi-perspective approach for mining simulation models from event data that includes the resource perspective. The resulting models are executable and can be used for performance analysis of the underlying system. However, resources are considered static entities and not active agents. Van der Aalst et al. [1] address the modeling of behavior and availability of resources. Yet, resources play a secondary role, with cases being the dominant perspective that defines business processes. Klijn et al. [9] develop a technique for querying event logs to uncover interactions between process entities. Yet, they do not provide a formal model that can be evaluated for its correctness or goodness-of-fit. Discovering

functional architecture models (FAM) that comprise interacting modules that internally perform various activities was proposed in [28]. Unlike the modules in FAMs, our agents are dynamic, decentralized, and may interact not only with other agents, but also with the environment. Fettke and Reisig presented an approach to system mining called Heraklit [5]. Heraklit proceeds by constructing distributed runs of participating agents and then combining them into the overall system, whereas Agent Miner constructs agent nets and an additional net that explicitly describes agents' interactions.

Tour et al. [24] have shown that by shifting process mining paradigm from case-based to agent-based, one can discover less complex models of business processes. Within this agent paradigm, Nesterov et al. [15] have recently proposed a process discovery solution. Their algorithm constructs sound generalized workflow nets that capture agents' behavior. In contrast to our approach, Nesterov et al. assume to know the interaction patterns between agents and aim to implement this given interaction pattern, while we discover the interactions from the data. Moreover, their algorithm can only discover pre-defined workflow patterns, while Agent Miner is generic and can model any pattern its ANDA component can discover. Furthermore, we evaluated Agent Miner over real-life logs, while Nesterov et al. tested their approach over synthetically generated data.

Our approach can also be viewed as a log-decomposition process discovery approach [27]. Such techniques propose to localize the event log to discover different models tailored to the data in these local logs. In these techniques, the log decomposition is usually driven by the case attribute, whereas we propose an agent-driven log decomposition.

Multi-Agent System Modeling. Multi-agent systems (MASs) have been studied extensively in the past; cf. [4] for a recent overview. Here, we focus on approaches that model multi-agent systems using Petri nets, our formalism of choice. A seminal paper by Moldt et al. [13] proposed to use colored Petri nets (CPNs) to model agent systems. Their model captures three crucial components in a MAS: communication, independence (between agents), and intelligence. However, given the richness of CPNs, formal results on correctness of constructed models were not obtain. Celaya et al. [3] model a multi-agent system using elementary Petri nets to capture interactions between agents and use Petri net analysis to ensure the models are deadlock-free. In our work, we discover MAS models from event data rather than relying on human expertise.

7   Conclusion

This paper presents and evaluates Agent Miner, a divide-and-conquer algorithm for discovering models of agents and their interactions from event data. The algorithm "divides" the input data into special parts and then "conquers" the parts using conventional process discovery algorithms. The constructed agent and interaction models provide a new, modular perspective on the data, suitable for analyzing process participants and their interactions. These artifacts can be integrated into a model that describes process control flow. Such integrated models are often smaller and represent the event data more faithfully than cor-

responding process models constructed using conventional discovery algorithms. The configuration of Agent Miner used in the evaluation reported in this paper ensures that the obtained integrated models are safe and sound.

Agent Miner has several limitations representing areas of interest for future work. First, the interaction logs are constructed by taking the first event in each agent trace of each agent. This approach ignores the information on the duration of agent activities and interactions. For instance, information on the durations of activities between interactions can be obtained by considering the first and the last event from agent traces between those interactions. Consequently, one can apply lifecycle-aware process discovery [10] or queue mining techniques [21] to infer agent interactions. Next, Agent Miner associates each event with only one agent. One can relax this limitation and study the effects of multiple agents sharing the same event. Finally, the evaluation approach used in this article is limited to the traditional model quality measures used in process mining. The use of new agent-specific quality measures for discovered models may highlight additional benefits of Agent Miner and other agent system mining algorithms in the context of agent-based business process management.

References

1. van der Aalst, W.M.P., Nakatumba, J., Rozinat, A., Russell, N.: Business process simulation. In: Handbook on Business Process Management 1, pp. 313–338. Springer (2010)
2. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowledge and Information Systems **59**(2), 251–284 (2019)
3. Celaya, J.R., Desrochers, A.A., Graves, R.J.: Modeling and analysis of multi-agent systems using Petri nets. In: 2007 IEEE International Conference on Systems, Man and Cybernetics. pp. 1439–1444 (2007)
4. Dorri, A., Kanhere, S., Jurdak, R.: Multi-agent systems: a survey. IEEE Access **6**, 28573–28593 (2018)
5. Fettke, P., Reisig, W.: Systems mining with Heraklit: the next step. In: BPM Forum. LNBIP, vol. 458, pp. 89–104. Springer (2022)
6. Halaška, M., Šperka, R.: Is there a need for agent-based modelling and simulation in business process management? Organizacija **51**(4) (2018)
7. Jennings, N.R., Faratin, P., Johnson, M.J., Norman, T.J., O'Brien, P., Wiegand, M.E.: Agent-based business process management. International Journal of Cooperative Information Systems **5**(2&3), 105–130 (1996)
8. Jennings, N.R., Norman, T.J., Faratin, P.: ADEPT: an agent-based approach to business process management. ACM SIGMOD Record **27**(4), 32–39 (1998)
9. Klijn, E.L., Mannhardt, F., Fahland, D.: Aggregating event knowledge graphs for task analysis. In: ICPM Workshops. LNBIP, vol. 468, pp. 493–505. Springer (2022)
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Using life cycle information in process discovery. In: BPM Workshops. LNBIP, vol. 256, pp. 204–217. Springer (2015)

11. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: BPM Workshops. LNBIP, vol. 171, pp. 66–78. Springer (2013)
12. Leemans, S.J.J., Poppe, E., Wynn, M.T.: Directly follows-based process mining: exploration & a case study. In: ICPM. pp. 25–32. IEEE (2019)
13. Moldt, D., Wienberg, F.: Multi-agent-systems based on coloured Petri nets. In: Petri Nets. LNCS, vol. 1248, pp. 82–101. Springer (1997)
14. Murata, T.: Petri nets: properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (Apr 1989)
15. Nesterov, R., Bernardinello, L., Lomazova, I., Pomello, L.: Discovering architecture-aware and sound process models of multi-agent systems: a compositional approach. Software and Systems Modeling pp. 1–25 (2022)
16. Polyvyanyy, A., Alkhammash, H., Di Ciccio, C., García-Bañuelos, L., Kalenkova, A., Leemans, S.J.J., Mendling, J., Moffat, A., Weidlich, M.: Entropia: a family of entropy-based conformance checking measures for process mining. In: ICPM Tools. CEUR workshop proceedings, vol. 2703, pp. 39–42. CEUR-WS.org (2020)
17. Polyvyanyy, A., Solti, A., Weidlich, M., Di Ciccio, C., Mendling, J.: Monotone precision and recall measures for comparing executions and specifications of dynamic systems. ACM Transactions on Software Engineering and Methodology **29**(3), 17:1–17:41 (2020)
18. Polyvyanyy, A., Weidlich, M., Weske, M.: Connectivity of workflow nets: the foundations of stepwise verification. Acta Informatica **48**(4), 213–242 (2011)
19. Reisig, W.: Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies. Springer (2013)
20. Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.M.P.: Discovering simulation models. Information Systems **34**(3), 305–327 (2009)
21. Senderovich, A.: Queue mining. In: Encyclopedia of Big Data Technologies. Springer (2019)
22. Sulis, E., Di Leva, A.: An agent-based model of a business process: the use case of a hospital emergency department. In: BPM Workshops. LNBIP, vol. 308, pp. 124–132. Springer (2017)
23. Syring, A.F., Tax, N., van der Aalst, W.M.P.: Evaluating conformance measures in process mining using conformance propositions. Transactions on Petri Nets and Other Models of Concurrency **14**, 192–221 (2019)
24. Tour, A., Polyvyanyy, A., Kalenkova, A.: Agent system mining: vision, benefits, and challenges. IEEE Access **9**, 99480–99494 (2021)
25. Tour, A., Polyvyanyy, A., Kalenkova, A., Senderovich, A.: Agent miner: implementation and evaluation results. Tech. rep., Melbourne University (2023). https://doi.org/10.26188/21127273
26. van der Aalst, W.M.P.: Process mining – Data Science in Action. Springer (2016)
27. van der Aalst, W.M.P., Kalenkova, A., Rubin, V., Verbeek, E.: Process discovery using localized events. In: Petri Nets. pp. 287–308. Springer (2015)
28. van der Werf, J.M.E., Kaats, E.: Discovery of functional architectures from event logs. In: PNSE@Petri Nets. pp. 227–243 (2015)
29. Wolfram, S.: Computation theory of cellular automata. Communications in Mathematical Physics **96**, 15–57 (1984)