Applied Soft Computing 11 (2011) 1697–1710

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Process discovery in event logs: An application in the telecom industry

Stijn Goedertier[a], Jochen De Weerdt[a,*], David Martens[a,b], Jan Vanthienen[a], Bart Baesens[a,c]

[a] Department of Decision Sciences and Information Management, Katholieke Universiteit Leuven, Naamsestraat 69, B-3000 Leuven, Belgium
[b] Department of Business Administration and Public Management, Hogeschool Gent, Universiteit Gent, Voskenslaan 270, B-9000 Ghent, Belgium
[c] School of Management, University of Southampton, Highfield Southampton, SO17 1BJ, United Kingdom

ARTICLE INFO

Article history:
Received 14 December 2009
Received in revised form 4 February 2010
Accepted 30 April 2010
Available online 8 May 2010

Keywords:
Process discovery
AGNEs
HeuristicsMiner
Event logs
Genetic Miner
Data mining
Workflow management systems (WfMS)

ABSTRACT

The abundant availability of data is typical for information-intensive organizations. Usually, discerning knowledge from vast amounts of data is a challenge. Similarly, discovering business process models from information system event logs is definitely non-trivial. Within the analysis of event logs, process discovery, which can be defined as the automated construction of structured process models from such event logs, is an important learning task. However, the discovery of these processes poses many challenges. First of all, human-centric processes are likely to contain a lot of noise as people deviate from standard procedures. Other challenges are the discovery of so-called non-local, non-free choice constructs, duplicate activities, incomplete event logs and the inclusion of prior knowledge. In this paper, we present an empirical evaluation of three state-of-the-art process discovery techniques: Genetic Miner, AGNEs and HeuristicsMiner. Although the detailed empirical evaluation is the main contribution of this paper to the literature, an in-depth discussion of a number of different evaluation metrics for process discovery techniques and a thorough discussion of the validity issue are key contributions as well.

1. Introduction

Organizations currently face an information paradox: the more they automate their processes, the less they are capable of monitoring and understanding them. A good understanding of processes is nonetheless vital for fulfilling business requirements such as verifying and guaranteeing business process compliance [26], setting up a coherent access control policy [14] and optimizing and redesigning business processes [17]. A better understanding will eventually enable organizations to provide better, automated support for their business processes in flexible, process-aware information systems [8,42].

Traditionally, practitioners have been obtaining insight into processes using interviewing techniques. A new and promising way of acquiring insights into business processes is the analysis of the event logs of information systems [33]. In many organizations, such event logs conceal an untapped reservoir of knowledge about the way employees and customers conduct every-day business transactions. Event logs are already available in many organizations. Popular Enterprise Resource Planning (ERP) systems such as SAP R/3, Oracle e-Business Suite and workflow management systems (WfMS) such as ARIS, TIBCO and Microsoft Biztalk already keep track of these event logs.

The topic of process discovery is relatively new and can be situated at an intersection of the fields of Business Process Management (BPM) and data mining [27]. It is inherently related to data mining and to the more general domain of knowledge discovery in databases (KDD) since the nature of its objectives is extracting useful information from large data repositories. Likewise, process discovery is strongly associated with BPM because of its purpose of gaining insight into business processes. As a result, process mining fits flawlessly into the BPM life cycle framework [34,41,47].

Because of the rather novelty of process discovery, it is definitely valuable to discuss various state-of-the-art discovery algorithms and assess them in a real-life setting. In order to do so, the remainder of this paper is structured as follows. In Section 2, process discovery and its main challenges are discussed and some basic concepts of Petri net theory are briefly introduced. Section 3 outlines a number of state-of-the-art process discovery techniques. Section 4 provides a discussion on evaluation metrics. In Section 5, three of the discussed algorithms will be applied on a real-life event log. Finally, the conclusions are formulated in Section 6.

2. Preliminaries

2.1. Process discovery

The basic idea of process discovery or control-flow discovery is straightforward: given an event log, automatically compose a

* Corresponding author. Tel.: +32 16 32 68 87; fax: +32 16 32 66 24.
E-mail addresses: stijn.goedertier@econ.kuleuven.be (S. Goedertier), jochen.deweerdt@econ.kuleuven.be (J. De Weerdt), david.martens@econ.kuleuven.be (D. Martens), jan.vanthienen@econ.kuleuven.be (J. Vanthienen), bart.baesens@econ.kuleuven.be (B. Baesens).

1568-4946/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2010.04.025

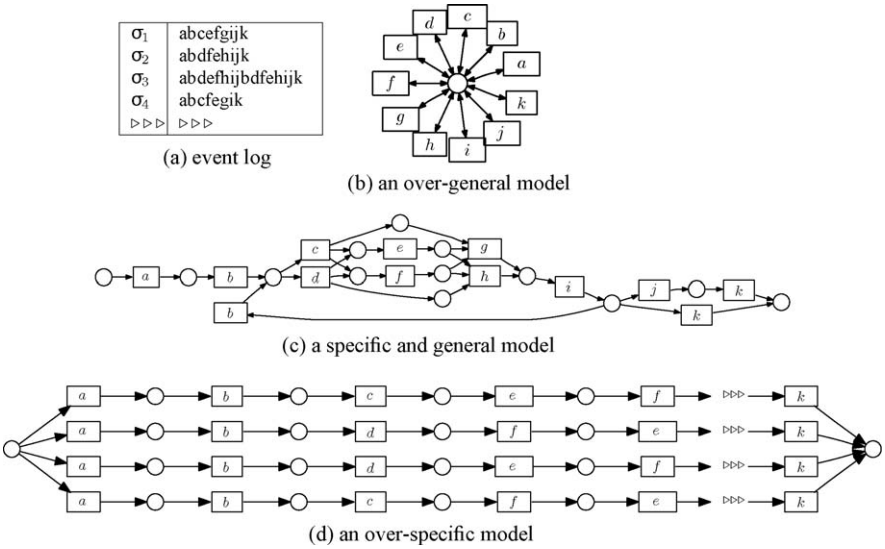

Fig. 1. DriversLicensel – discovery of a driver's license application process – (the transitions correspond to activity types that have the following meaning: *a* start, *b* apply for license, *c* attend classes cars, *d* attend classes motor bikes, *e* obtain insurance, *f* theoretical exam, *g* practical exam cars, *h* practical exam motor bikes, *i* get result, *j* receive license, and *k* end). (a) Event log, (b) an over-general model, (c) a specific and general model and (d) an over-specific model.

suitable process model that describes the behavior seen in the log. Auspiciously, processes occur in a more or less structured fashion, containing structures such as or-joins, or-splits, and-joins, and-splits, and loops. More accurately, the learning task can be formulated as follows: given an event log that contains the events about a finite number of process instances, find a model that correctly summarizes the behavior in the event log, striking the right balance between generality (allowing enough behavior) and specificity (not allowing too much behavior). For the purpose of process discovery, processes are often represented as workflow nets, a special subclass of Petri nets. Fig. 1 illustrates the learning problem for a Driver's License application process. Given the event log in Fig. 1(a), different process models can be conceived that portray similar behavior as the event log. The Petri net in Fig. 1(b) is capable of parsing every sequence in the event log. However, it can be considered to be overly general as it allows any activity to occur in any order. In contrast, the Petri net in Fig. 1(d) is overly specific, as it provides a mere enumeration of the different sequences in the event log. The Petri net in Fig. 1(c) is likely to be the more suitable process model. It is well-structured, and strikes a reasonable balance between specificity and generality, allowing for instance an unseen sequence *abcefgik*, but disallowing random behavior.

Process discovery is particularly useful in the context of human-centric processes for which an information system does not enforce the activities to be carried out in a particular order [30]. The extraction of useful knowledge from human-centric event logs often goes beyond the ability of descriptive statistics. What is additionally required is a process model that correctly summarizes the event log and that describes how business processes have actually taken place. Process discovery can provide answers to many business questions. In the first place, process discovery enables organizations to unveil implicit processes and tacit knowledge. Moreover, the availability of a model that accurately describes the behavior in an event log allows for advanced analysis such as the identification of performance bottlenecks or the localization of paths in the process model that are not compliant to existing regulations. The business value of process discovery is well illustrated by the plugins within the ProM Framework [31,37]. In analogy with the WEKA toolset for data mining [12,46], the ProM Framework consists of a large number of plugins for the analysis of event logs. The *Conformance Checker* plugin [25], for instance, allows identifying the discrepancies between an idealized process model and an event log. Moreover, with a model that accurately describes the event log, it becomes possible to use the time-information in an event log for the purpose of performance analysis, using, for instance, the *Performance Analysis with Petri nets* plugin.

2.2. Challenging problems

The discovery of process models from event logs faces many challenges [35]:

- **Incomplete logs**: Process discovery is inherently a descriptive learning task that aims at accurately summarizing an event log such that the discovered process model models all allowable behavior (recall) but does not allow for behavior that is not present in the event log (precision). Nonetheless, the ability to generalize beyond observed behavior can also be important to the problem of process discovery.
- **Noise**: Human-centric processes are prone to exceptions and logging errors. This causes additional low-frequent behavior to be present in the event log that is unwanted in the process model to be discovered. Consequently, process discovery algorithms face the challenge of not overfitting this noise.
- **Unsupervised learning**: An inherent difficulty of process discovery is that it is often limited to the much more difficult setting of unsupervised learning. Event logs rarely contain negative information about state transitions that were prevented from taking place.
- **History-dependent behavior**: Human-centric processes portray behavior that is dependent on the non-immediate history. An example is the occurrence of history-based joins in the control flow of a business process [39]. A history of related events can therefore be a strong predictor and is readily available in event logs. However, *non-local* behavior of business processes already presents many challenges for process modeling [2,40]. For process discovery, the inclusion of such non-local, non-free choice in the hypothesis space of process mining algorithms raises difficulties with regard to search space complexity and hypothesis visualization [45].
- **Inclusion of case data**: The routing choices that are made in business processes can also be dependent on the value of its data

properties. The inclusion of case data in discovered process models is therefore potentially useful. It nonetheless poses difficulties because of the time-varying nature of case data.

- **Prior knowledge**: An essential requirement for process discovery, and data mining in general, is that it produces justifiable models. Justifiability refers to the extent to which an induced model is in line with the existing domain knowledge. In the context of process discovery, domain knowledge might refer to prior knowledge about concurrency (parallelism), locality or exclusivity of activities.
- **Duplicate activities**: Duplicate or homonymic activities are activities that are identically labeled but that are used in different execution contexts within the process. In particular, the problem of duplicate tasks refers to the situation that the same activity appears multiple times in the process model. Often, duplicate activities are embedded in different contexts, and thus surrounded by different activities in the process model.

2.3. Petri net semantics

Example 1(c) is a Petri net representation of a simplified driver's license application process. Petri nets represent a graphical language with a formal semantics that has been used to represent, analyze, verify, and simulate dynamic behavior [23]. Petri nets consist of places, tokens, and arcs. *Places* (drawn as circles) can contain *tokens* and are a distributed representation of state. Each different distribution of tokens over the places of a Petri net indicates a different state. Such a state is called a *marking*. *Transitions* (drawn as rectangles) can consume and produce tokens and represent a state change. *Arcs* (drawn as arrows) connect places and transitions and represent a flow relationship. More formally, a marked Petri net is a pair $((P,T,F),s)$ where,

- P is a finite set of places,
- T is a finite set of transitions such that $P \cap T = \varnothing$, and
- $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of direct arcs, and
- $s \in P \rightarrow \mathbb{N}$ is a bag over P denoting the marking of the net [28,29].

Petri nets are bipartite directed graphs, which means that each arc must connect a transition to a place or a place to a transition. The transitions in a Petri net can be labeled or not. Transitions that are not labeled are called *silent transitions*. Different transitions that bear the same label are called *duplicate transitions*.

The behavior of a Petri net is determined by the *firing rule*. A transition t is *enabled* iff each input place p of t contains at least one token. When a transition is enabled it can *fire*. When a transition fires, it brings about a state change in the Petri net. In essence, it consumes one token from each input place p and produces one token in each output place p of t. To evaluate the extent to which a Petri net is capable of parsing an event sequence, transitions might be forced to fire. A transition that is not enabled, can be *forced to fire*. When a transition is forced to fire, it consumes one token from each input place that contains a token – if any – and produces one token in each output place. Petri nets are capable of representing structures such as sequences, or-splits, and-splits (parallel threads), or-joins, and-joins, loops (iterations), history-dependent or-splits, and duplicate activities that are typical for organizational processes.

3. Discussion of process discovery techniques

In this section, we elaborate on a number of state-of-the-art process discovery techniques that have been applied to the context of workflow management systems (WfMS). Table 1 provides a non-comprehensive overview of process mining algorithms.

Table 1
Overview of process discovery algorithms.

Algorithm (Ref.)	Summary
α, α⁺, α⁺⁺ van der Aalst et al. [36] Alves de Medeiros et al. [3]	Derives a Petri net from local, binary ordering relations detected within an event log. α⁺ extends the α algorithm to discover short loops. α⁺⁺ extends the α algorithm to discover non-local, non-free choice constructs.
HeuristicsMiner Weijters and van der Aalst [43] Weijters et al. [44]	Extends the α algorithm by taking into account the frequency of the follows relationship, to calculate dependency/frequency tables from the event log and uses heuristics to convert this information into a heuristics net.
Genetic Miner Alves de Medeiros et al. [4]	A genetic algorithm that selects the more complete and precise heuristics nets over generations of nets.
AGNEs Goedertier et al. [13]	A configurable technique representing process discovery as a multi-relational classification problem on event logs supplemented with artificially generated negative events.

3.1. The α-algorithms

The α-algorithm can be considered to be a theoretical learner for which van der Aalst et al. [36] prove that it can learn an important class of workflow nets, called structured workflow nets, from complete event logs. The α-algorithm assumes event logs to be complete with respect to all allowable binary sequences and assumes that the event log does not contain any noise. Therefore, the α-algorithm is sensitive to noise and incompleteness of event logs. Moreover, the original α-algorithm was incapable of discovering short loops or non-local, non-free choice constructs. Alves de Medeiros et al. [3] have extended the α-algorithm to mine short loops and called it α^+. Wen etal. [45] made an extension for detecting implicit dependencies, so-called non-local, non-free choice constructs. None of the algorithms can detect duplicate activities. The main reason why the α-algorithms are sensitive to noise, is that they do not take into account the frequency of binary sequences that occur in event logs. Note that because of the lack of robustness to noise, none of the α-algorithms were included in the real-life comparative study.

3.2. Genetic Miner

Due to the limitations of local search, early approaches to process mining, were generally not able to discover non-trivial constructs like non-free choice, invisible tasks and duplicate tasks. The main motivation of Alves de Medeiros et al. [4] to apply a genetic algorithm for process discovery is to benefit from global search. This global search property is the most important characteristic of genetic algorithms in general, as for example illustrated in [9,11]. The conception of a genetic process mining algorithm requires three main concerns to be resolved: the definition of an appropriate internal representation, a suitable fitness measure and genetic operators that guarantee to cover the entire search space. Genetic Miner defines its search space in terms of causal matrices. These matrices express task dependencies only, yet they are closely related to Petri nets. Additionally, the genetic algorithm should be provided with an initial population. Although a random initial population would not influence the outcome of the algorithm, using a heuristic to build it, will result in a significant improvement of the calculation time. As such, Genetic Miner makes use of a so-called hybrid genetic algorithm.

Once the internal representation is specified and the initial population is determined, a suitable fitness function is required. Alves de Medeiros et al. propose a measure that incorporates both a recall and a precision dimension. The fitness function evaluates how well an individual (i.e. a process model) is able to reproduce the behavior in the event log and as a result drives the algorithm towards suitable models. Mimicking the process of evolution, the algorithm builds new individuals in each generation by using genetic operators. Both crossover and mutation are applied by Genetic Miner. Crossover implies the recombination of existing material, whereas mutation can be defined as the insertion of new material in the population. It is required that all the points in the search space, as defined by the internal representation, may be reached.

Because of the global search property, Genetic Miner is capable of detecting non-local patterns in the event log. Moreover, the algorithm ensures a fair robustness to noise because of the arc post-pruning step. In this step, arcs in a mined process model that are used fewer times than a given threshold are removed from the model. Because the detection of non-local patterns and the robustness to noise are especially useful when dealing with real-life event logs, the selection of Genetic Miner for the comparative study is self-evident.

3.3. AGNEs

AGNEs [13] addresses the difficulties of process mining by representing the discovery task as first-order classification learning on event logs, supplemented with artificially generated negative events. Like Genetic Miner, the AGNEs algorithm is capable of constructing Petri net models from event logs and has been implemented as a mining plugin in the ProM Framework.

The starting point of the AGNEs process discovery algorithm is the analysis of frequent constraints that hold in the event log. These constraints are used to gain insight in local dependency, non-local dependency, and parallelism relationships that exist between pairs of activities. These relationships are used in the language bias of the classification learner, to bias the classification towards locally discriminating preconditions as much as possible. The aim is to facilitate the construction of a graphical model from the learned preconditions.

A second step in the AGNEs process discovery algorithm is the induction of artificial negative events. AGNEs represents process discovery as a classification problem. In particular, AGNEs tries to find a theory to classify whether a transition can take place or not. Unfortunately, event logs rarely contain information about transitions that are not allowed to take place. Obviously, the reason is that most information systems do not keep track of the activities that they could not have been doing at particular moments. Therefore, AGNEs artificially generates negative events, quite similarly as is done with active learning in a data mining context (see e.g. [20]). We refer to [13] for a detailed explanation of the induction of negative events.

Given an event log supplemented with artificially generated negative events, it becomes possible to learn the conditions that discriminate between the occurrence of either a positive or a negative event. As such, process mining is represented as a classification learning problem. The AGNEs process discovery algorithm is designed to learn the discriminating conditions that determine whether an event can take place or not, given a history of events of other activities. To induce this knowledge, AGNEs makes use of an existing multi-relational classification learner: Tilde [5]. Supplied with the appropriate language bias, Tilde constructs for each activity type a binary logical decision tree (LDT) that best partitions the negative and positive events. Blockeel and De Raedt show how a LDT can be transformed into an equivalent logical program from which the corresponding Petri net representation is derived.

The benchmark study in [13] shows that AGNEs is expressive, robust to noise and has the ability to generalize. Furthermore, the experiments indicate that AGNEs performs well in comparison with other state-of-the-art process mining techniques. The main reason why AGNEs can be described as robust to noise is that the constraints in the algorithm's language bias allows it to come up with structured patterns and to some extent prevents the construction of arbitrary connections between transitions. What is more, the formulation of process discovery as a classification task allows for the application of an already robust classification algorithm (TILDE).

3.4. HeuristicsMiner

Weijters et al. [44] have developed a heuristic-based method for process discovery, called HeuristicsMiner. In particular, HeuristicsMiner extends the formal α-algorithm [36] in that it applies frequency information with regard to three types of relationships between activities in an event log: direct dependency, concurrency and not-directly-connectedness.

The starting point of the algorithm is the construction of a so-called dependency graph. Therefore, the dependency frequencies between the events of an event log are used in a heuristic search for the correct relations. In order to distinguish between noise and low-frequent behavior, three threshold parameters are available: the dependency threshold, the positive observations threshold and the relative to best threshold. In a second step, a solution is provided for handling short loops.

Then, in a third step, HeuristicsMiner's main idea is to derive a causal matrix of preconditions from the dependency graph whereby only the highest dependency measures are taken into account. The use of a causal matrix as internal representation is entirely similar to the Genetic Miner algorithm. The causal matrix representation is required in order to distinguish between different dependency relations between activities in the process model. As such, AND/XOR-splits/joins can be successfully identified and represented. These different types of relationships cannot be represented in a simple dependency graph. In [21], a detailed description about the semantics of causal matrices is given, as well as an in-depth discussion on how to translate a causal matrix into a Petri net.

Finally, Weijters et al. [44] define their heuristic approach to visually represent the derived causal matrix in the form of a heuristic net. The translation between a heuristic net and a Petri net is relatively straightforward and is implemented in the ProM Framework.

According to its specification, HeuristicsMiner can discover short loops and non-local dependencies, but lacks the capability of detecting duplicate activities. As a result of the threshold parameter setting, the heuristic algorithm is prone to be noise resilient. This is because noise can be defined as low-frequent behavior and the threshold configuration intrinsically prevents the discovery of low-frequent patterns in the process model. Conform its conception, HeuristicsMiner has the advantage to be able to focus on all behavior in the event log, or only on the main behavior.

3.5. Other process discovery techniques

Although not included in the comparative study in Section 5, a number of other process mining techniques are discussed briefly. van Dongen and van der Aalst [38] developed Multi-phase miner, which is a multi-phase approach to process mining that starts from the individual process sequences, constructs so-called instance graphs for each sequence that account for parallelism, and then aggregates these instance graphs according to previously detected binary relationships between activity types. Interestingly,

the aggregation ensures that every discovered process model has a perfect recall, but generally scores less on specificity.

With the FSM/Petrify miner, van der Aalst et al. [32] present a two-phase approach to process discovery that allows to configure when states or state transitions are considered to be similar. The ability to manipulate similarity is a good approach to deal with incomplete event logs. In particular, several criteria can be considered for defining similarity of behavior and states: the inclusion of future or past events, the maximum horizon, the activities that determine state, whether ordering matters, the activities that visibly can bring about state changes, etcetera. Using these criteria, a configurable finite state machine can be constructed from the event log. In a second phase, the finite state machine is folded into regions using the existing theory of regions [7]. For the moment, the second phase of the algorithm still poses difficulties with respect to constructing suitable process models.

Finally, Günther and van der Aalst [15] propose Fuzzy Miner, an adaptive simplification and visualization technique based on significance and correlation measures to visualize the behavior in event logs at various levels of abstraction. The main contribution of Fuzzy Miner is that it can also be applied to less structured, or unstructured processes of which the event logs cannot easily be summarized in concise, structured process models.

4. Evaluation metrics

Discovered process models preferably allow the behavior in the event log (recall) but no other, unobserved, random behavior (specificity). Once a strong assumption of completeness (i.e. an event log portrays the complete behavior of the underlying process) is weakened, it also becomes a challenge to determine the right level of generalization. Therefore, it is not trivial to define appropriate metrics to compare different process discovery algorithms. For the purpose of this comparative study, we will use a number of metrics calculated on the basis of a Petri net representation of the discovered process model. In addition, we will evaluate the techniques on the basis of metrics that are related to Machine Learning. Therefore, it is required that, beside positive examples, also negative examples are available. Because the AGNEs Process Discovery technique inherently uses the generation of artificial negative events, we will use these negative examples to calculate the according metrics for each of the applied discovery techniques.

4.1. Petri net based metrics

Weijters et al. [44] define the parsing measure *PM*, which quantifies the percentage of traces in the log that can be replayed by the discovered process model. The measure is defined as follows:

- **Parsing measure** *PM*: The number of sequences in the event log that are correctly parsed by the process model, divided by the total number of sequences in the event log. For efficiency, the similar sequences in the event log are grouped. Let k represent the number of grouped sequences, n_i the number of process instances in a grouped sequence i, c_i a variable that is equal to 1 if grouped sequence i can be parsed correctly, and 0 if grouped sequence i cannot be parsed correctly. The parsing measure can be defined as follows Weijters et al. [44]:

$$PM = \frac{\sum_{i=1}^{k} n_i c_i}{\sum_{i=1}^{k} n_i}.$$

PM is a coarse-grained metric. A single missing arc in a Petri net can result in parsing failure for all sequences. A process model with a single point of failure is generally better than a process model with more points of failure. This is not quantified by the parsing measure *PM*.

Rozinat-IS-2008 define two other metrics to evaluate a discovered process model: the fitness metric f and the advanced behavioral appropriateness metric a'_B:

- **Fitness** f: Fitness is a metric that is obtained by trying whether each (grouped) sequence in the event log can be reproduced by the generative model. This procedure is called *sequence replay*. The metric assumes the generative model to be a Petri net. At the start of the sequence replay, f has an initial value of one. During replay, the transitions in the Petri net will produce and consume tokens to reflect the state transitions. However, the proportion of tokens that must additionally be created in the marked Petri net, so as to *force* a transition to fire, is subtracted from this initial value. Likewise, the fitness measure f punishes for extra behavior by subtracting the proportion of remaining tokens relative to the total number of produced tokens from this initial value. Let k represent the number of grouped sequences, n_i the number of process instances, c_i the number of tokens consumed, m_i the number of missing tokens, p_i the number of produced tokens, and r_i the number of remaining tokens for each grouped sequence i ($1 \le i \le k$). The fitness metric can be defined as follows [25]:

$$f = \frac{1}{2}\left(1 - \frac{\sum_{i=1}^{k} n_i m_i}{\sum_{i=1}^{k} n_i c_i}\right) + \frac{1}{2}\left(1 - \frac{\sum_{i=1}^{k} n_i r_i}{\sum_{i=1}^{k} n_i p_i}\right).$$

- **Behavioral appropriateness** a'_B: Behavioral appropriateness is a metric that is obtained by an exploration of the state space of a Petri net and by comparing the different types of *follows* and *precedes* relationships that occur in the state space with the different types of *follows* and *precedes* relationships that occur in the event log. The metric is defined as the proportion of number of *follows* and *precedes* relationships that the Petri net has in common with the event log in regard to the number of relationships allowed by the Petri net. Let S_F^m be the S_F relation and S_P^m be the S_P relation for the process model, and S_F^l the S_F relation and S_P^l the S_P relation for the event log. The advanced behavioral appropriateness metric a'_B is defined as follows [25]:

$$a'_B = \left(\frac{|S_F^l \cap S_F^m|}{2.|S_F^m|} + \frac{|S_P^l \cap S_P^m|}{2.|S_P^m|}\right).$$

Fitness f and behavioral appropriateness a'_B are particularly useful measures to evaluate the performance of a discovered process model. Moreover, these metrics have been implemented in the ProM framework. However, the interpretation of the fitness measure requires some attention: although it accounts for recall as it punishes for the number of missing tokens that had to be created, it also punishes for the number of tokens that remain in a Petri net after log replay. The latter can be considered *extra* behavior. Therefore, the fitness metric f also has a specificity semantics attached to it. Furthermore, it is to be noted that the behavioral appropriateness a'_B metric is not guaranteed to account for all non-local behavior in the event log (for instance, a non-local, non-free, history-dependent choice construct that is part of a loop will not be detected by the measure). In addition, the a'_B metric requires an analysis of the state space of the process model or a more or less exhaustive simulation to consider all allowable sequences by the model.

4.2. Machine Learning metrics

Since Goedertier et al. [13] formulated process discovery as a binary classification problem on event logs supplemented with arti-

ficial negative events, the application of Machine Learning related metrics becomes available. For example, one can use the true positive and true negative rate from classification learning theory to quantify the recall and specificity of a process model:

- **True positive rate** TP_{rate} or **recall**: the percentage of correctly classified positive events in the event log. This probability can be estimated as follows: $TP_{rate} = TP/(TP + FN)$, where TP is the amount of correctly classified positive events and PN is the amount of incorrectly classified positive events.
- **True negative rate** TN_{rate} or **specificity**: the percentage of correctly classified negative events in the event log. This probability can be estimated as follows: $TN_{rate} = TN/(TN + FP)$, where TN is the amount of correctly classified negative events and FP is the amount of incorrectly classified negative events.

From the literature, it is known that accuracy can be derived from the combination of TP_{rate} and TN_{rate} [27]. Accuracy is the sum of the true positive and true negative rate, weighted by the respective class distributions. The fact that accuracy is relative to the underlying class distributions can lead to counterintuitive interpretations. Moreover, in process discovery, these class distributions can be quite different and have no particular meaning. In order to make abstraction of the proportion of negative and positive events, Goedertier et al. [13] propose to attach equal importance to both recall and specificity: $acc = (1/2)\,recall + (1/2)\,specificity$. According to this definition, the accuracy of a majority-class predictor is 1/2. Flower models, such as the one in Fig. 1(b), are an example of such majority-class predictors. Because a flower model represents random behavior, it has a perfect recall of all the behavior in the event log but it also has much additional behavior compared to the event log. Because of the latter fact, the flower model has zero specificity, and an accuracy of 1/2. Any useful process model should have an accuracy higher than 1/2.

The availability of an event log supplemented with artificial negative events, allows for the definition of a specificity metric that does not require a state space analysis. Instead, specificity can be calculated by parsing the (grouped) sequences, supplemented with negative events. Accordingly, Goedertier et al. define:

- **Behavioral recall** r_B^p: The behavioral recall r_B^p metric is obtained by parsing each grouped event sequence. The values for TP and FN are initially zero. Starting from the initial marking, each sequence is being parsed. Whenever an enabled transition fires during parsing, the value for TP is increased by one. Whenever a transition is not enabled, but must be forced to fire the value for FN is increased. As an optimization, identical sequences only are to be replayed once. Let k represent the number of grouped sequences, n_i the number of process instances, TP_i the number of events that are correctly parsed, and FN_i the number of events for which a transition was forced to fire for each grouped sequence i $(1 \le i \le k)$. At the end of the sequence replay, r_B^p is obtained as follows:

$$r_B^p = \left(\frac{\sum_{i=1}^{k} n_i TP_i}{\sum_{i=1}^{k} n_i TP_i + \sum_{i=1}^{k} n_i FN_i} \right).$$

In the case of multiple enabled duplicate transitions, sequence replay fires the transition of which the succeeding transition is the next positive event (or makes a random choice). In the case of multiple enabled silent transitions, log replay fires the transition of which the succeeding transition is the next positive event (or makes a random choice). Unlike the fitness metric f, r_B^p does not punish for remaining tokens. Whenever after replaying a sequence, remaining tokens cause *additional behavior* by enabling particular transitions, this is punished by our behavioral specificity metric s_B^n.

Table 2
Telecom – properties of the original and preprocessed event log.

Property	Definition
Real-life process	Second-line customer-initiated processes regarding the resolution of internet, television, and telephony service problems.
Event log	The event log recorded by a WfMS, containing all *dispatch* events of 17,812 cases that were closed between January 6, 2008 and February 6, 2008.
Activity types	127 different queues of which processes involving the 40 most frequent queues were retained. In addition, the *create* and *close case* event types are considered to be activity types.
Event types	From the original four event types (*create*, *notes*, *dispatch*, and *case close*) only the *dispatch* event type was retained.
Process instances	The *case id* field allowed to identify 17,821 process instances of which 17,812 remained after filtering out processes that did not involve the retained activity types.
Case data	The following case data fields were given a value for more than 20% of the cases: *problem code*, *problem cause*, and *service type*.
$\neq$ process instances	After grouping according to the same follows relationship 1375 different process instances were retained.

- **Behavioral specificity** s_B^n: The behavioral specificity s_B^n metric can be obtained during the same replay as r_B^p. The values for TN and FP are initially zero. Whenever during replay, a negative event is encountered for which no transitions are enabled, the value for TN is increased by one. In contrast, whenever a negative event is encountered during sequence replay for which there is a corresponding transition enabled in the Petri net, the value for FP is increased by one. As an optimization, identical sequences only are to be replayed once. Let k represent the number of grouped sequences, n_i the number of process instances, TN_i the number of negative events for which no transition was enabled, and FP_i the number of negative events for which a transition was enabled during the replay of each grouped sequence i $(1 \le i \le k)$. At the end of the sequence replay, s_B^n is obtained as follows:

$$s_B^n = \left(\frac{\sum_{i=1}^{k} n_i TN_i}{\sum_{i=1}^{k} n_i TN_i + \sum_{i=1}^{k} n_i FP_i} \right).$$

Because the behavioral specificity metric s_B^n checks whether the Petri net recalls negative events, it is inherently dependent on the way in which these negative events are generated. For the moment, the negative event generation procedure included in the AGNEs algorithm is configurable by the negative event injection probability π, and whether or not it must account for the parallel variants of the given sequences of positive events. For the purpose of uniformity, negative events are generated in the test sets with π equal to 1.0 and account for parallel variants = true.

5. Case study at a European telecom provider

5.1. The obtained event log

The event log consists of events about customer-initiated processes that are handled at three different locations by the employees of the telecom provider. The handling of cases is organized in a first line and a second line. First-line operators are junior

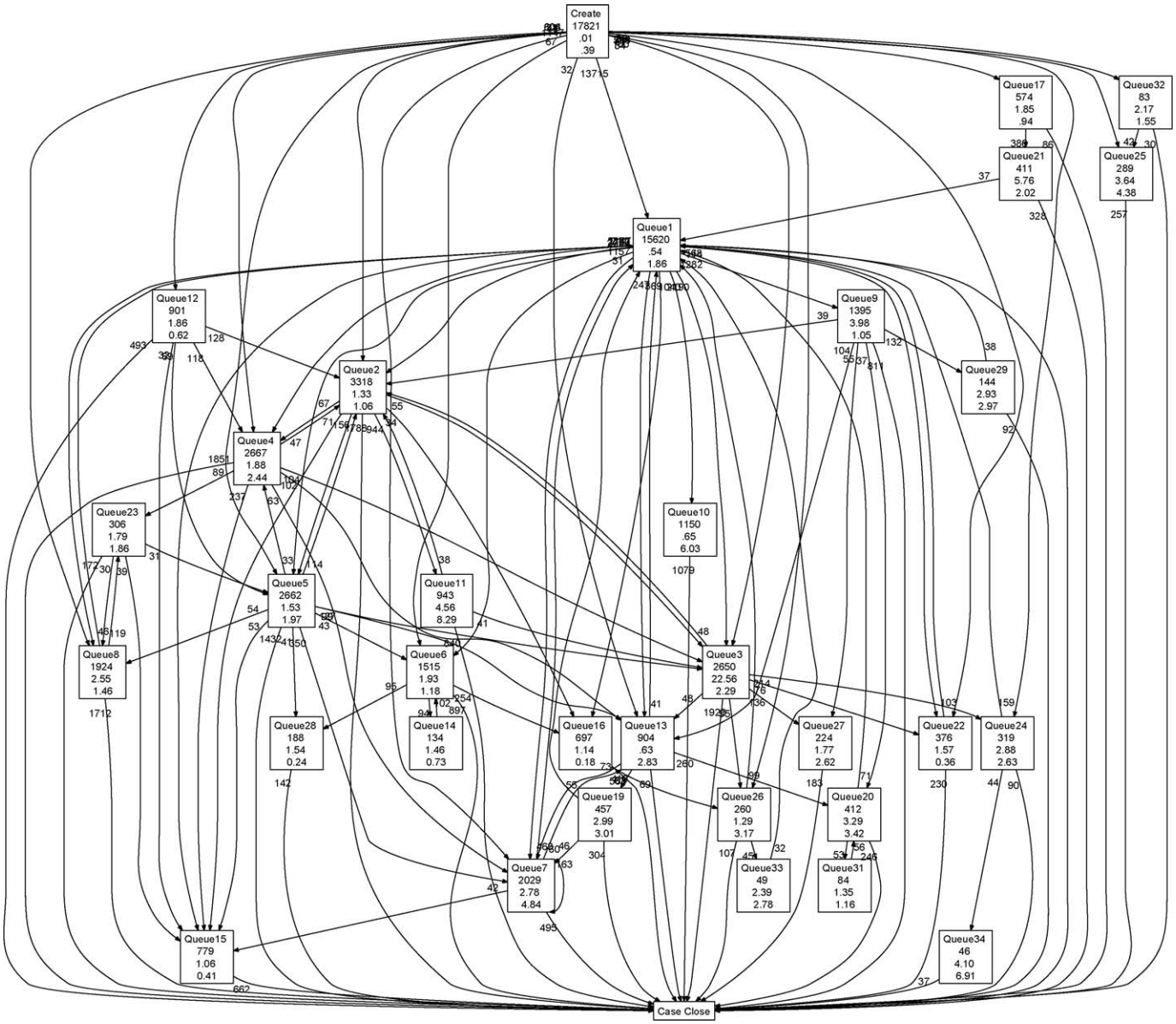

Fig. 2. Telecom – a visualization of queue routing using descriptive statistics only.

operators that deal with frequent customer requests for which standardized procedures have been put in place. When a first-line operator cannot process a case, it is routed to a queue of the second line. Second-line case handling is operated by senior experts who have the authority to make decisions to solve the more involved cases. The second-line processes are coordinated and supported by means of a workflow management system (WfMS). The obtained event log consists of these second-line case handling events. The second-line WfMS is organized as a system of 127 logical queues. Each queue corresponds to a number of similar types of activities that are to be carried out. At any given moment each active case resides in exactly one queue. Employees can process a case by taking it out of the queue into their personal work bin. Every evolution of a case is documented by adding notes. Moreover, employees can classify the nature of the case according to a number of data fields. In addition, a worker or dispatcher has the ability to reroute cases to different queues whenever this is necessary. The system imposes no restrictions with regard to the routing of cases. Queues represent a work distribution system and are akin to roles in WfMS. For the purpose of this analysis, queues are considered to be activity types. To handle cases that have a common cause, a parent–child structure exists among cases. Occasionally, rules can perform updates on cases, such as automatically closing a case, and automatically rerouting a case.

Table 2 summarizes the properties of the original event log and of the event log that was obtained after preprocessing. Because the goal of the case study is to map the routing choices, only *dispatch* event types were retained from the event log. In this context, queues can be seen as activity types.

First, an exploratory study was made using descriptive statistics. A number of plain vanilla SQL queries on the event log were used to calculate the average and standard deviation of the queue times, the total number of cases that were routed to a queue, and the inflow and outflow to other queues. This data is visualized by means of graphs such as the one in Fig. 2. Each rectangle represents a queue. Each arc represents cases being routed from one queue to another. For each queue, the average cycle time, standard deviation of the cycle time, and the total number of cases is indicated. On each arc, the outflow of cases from one queue to another is indicated in absolute quantities. This visualization makes it immediately appar-

Table 3
Telecom – parameter settings.

Algorithm (Ref.)	Parameter settings	
Genetic Miner Alves de Medeiros et al. [4]	Population size = 10 Max number generations = 5000 Initial population type = possible duplicates Power value = 1 Elitism rate = 0.2 Selection type = tournament 5 Extra behavior punishment with $\kappa = 0.025$ Enhanced crossover type with crossover probability = 0.8 Enhanced mutation type with mutation probability = 0.2	
AGNEs Goedertier et al. [13]	Prior knowledge	$\forall a, b \in A$: *PriorSerial(a,b)*
	Temporal constraints	$t_{absence} = 0.9$, $t_{chain} = 0.08$, $t_{succ} = 0.8$, $t_{ordering} = 0.08$, $t_{triple} = 0.1$
	Negative event generation	Injection probability $\pi = 0.04$ Calculate parallel variants = true
	Language bias:	Include skip sequences = false Include global sequences = false Include occurrence count = false Data conditions = none
	TILDE	Splitting heuristic: gain Minimal cases in tree nodes = 5 C4.5 pruning with confidence level = 0.25
	Graph construction	$t_{connect} = 0.0$
HeuristicsMiner Weijters et al. [44]	Relative to best threshold = 0.05 Positive observations = 10 Dependency threshold = 0.9 Length-one-loops threshold = 0.9 Length-two-loops threshold = 0.9 Long-distance threshold = 0.9 Dependency divisor = 1 AND threshold = 500,000.0 Use all-activities-connected heuristic = true Use long-distance dependency heuristic = false	

ent that cases in a given queue can be routed to a great number of queues. To reduce the information overload on the graph, arcs with less than 30 cases were left out. Disconnected queues are left out as well. A lean sigma black belt expert, who is currently making improvements to the second-line case handling, confirmed that the graph was a useful visualization of the queue routing choices that were currently made. Moreover, he asserted that the cross-table representation of the same data, allowing the expert to drill down to individual cases whenever required, was a useful tool in understanding the complexity of the underlying processes.

5.2. Validity of assumptions

Although many process discovery algorithms have been developed, only a few authors report on the practical application of process discovery algorithms on real-life event logs [4,10,15,31]. Practical applications are important, because they give an indication of the validity of the assumptions that are made by process discovery algorithms. In particular, authors like Günther and van der Aalst [15] and Ferreira et al. [10] have identified that process discovery algorithms make a number of assumptions that are not always guaranteed for realistic event logs. In the next paragraphs, the validity of these assumptions with respect to the obtained event log is evaluated.

Assumption 1. There is a one-to-one mapping between a system event and a business event.

This assumption underestimates the importance of preprocessing an event log. Günther and van der Aalst [15] assert that system logs may contain events of different levels of abstraction. Moreover, the authors point out that in many processes it is not meaningful to separate the control flow, data flow, and resource perspectives. System logs can indeed have an ontology that is different from the process modeling ontology that is used in this text. Moreover, the interpretation of a system event can differ in function of the intended analysis. Another issue is that the most useful information to classify a system event may reside in unstructured data, such as free text fields or e-mails. The labeling of unstructured data for the purpose of activity recognition can be believed to be very difficult. For these reasons, it is often not possible to automatically convert a system event into a relevant business event. In the obtained event log, the resource and control flow perspective are intertwined: the queue to which a particular case is being dispatched (resource perspective) is considered to be the activity type (control flow perspective). Undoubtedly, the textual annotations that employees add to each case conceal a lot of information about the nature of the underlying activity, but we did not attempt to use this text field to discover or recognize more fine-grained activity types.

Assumption 2. It is possible to identify meaningful process instances in an event log.

This assumption entails that each system event can be related to a meaningful process instance by means of a suitable identifier, such as a customer id, an employee number, or a case id. Ferreira et al. [10] argue that for many system logs the existence of such an identifier is not guaranteed. Consequently, a lot of preprocessing can be required to identify clusters of events that represent process instances. In the obtained event log, process instances are identified using the *case id* identifier of the WfMS.

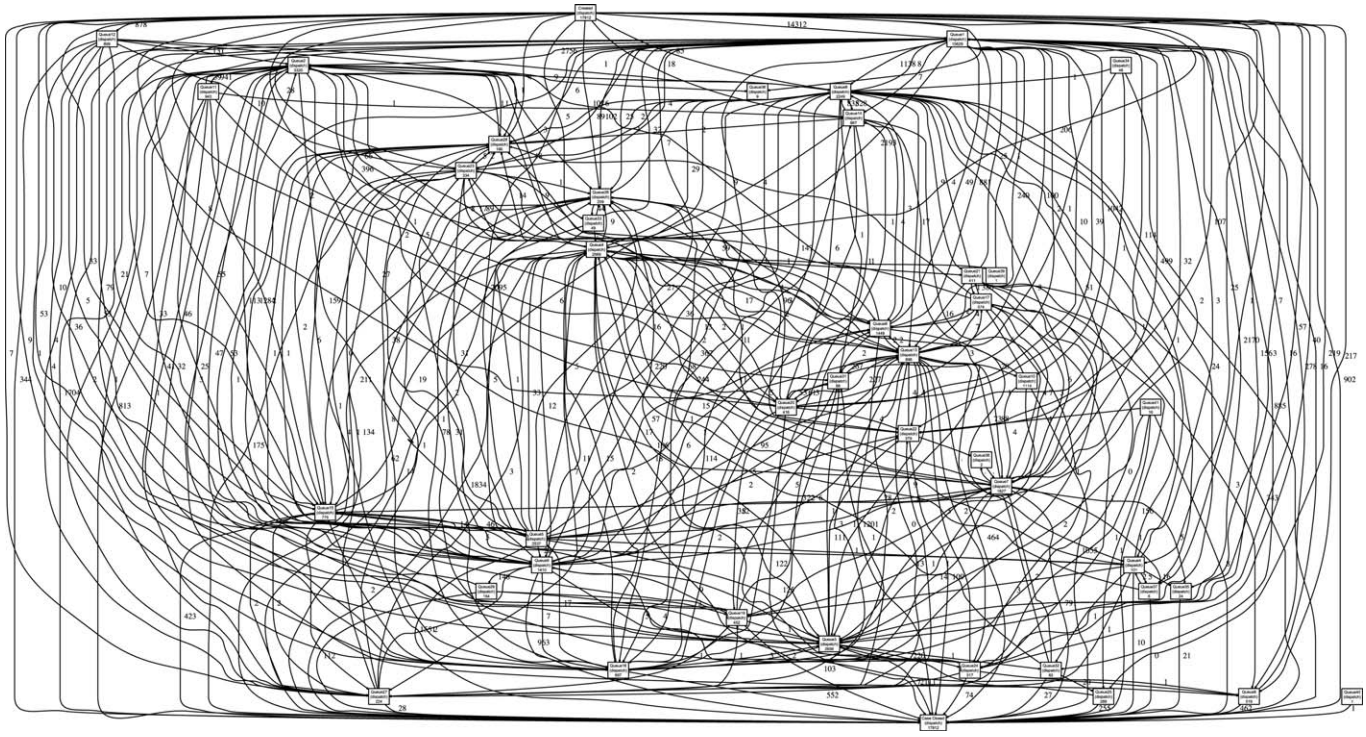

Fig. 3. Telecom – Genetic Miner mining results as a heuristics net.

Assumption 3. The events in the log are generated by exactly one underlying process.

Even when meaningful process instances can be identified from an event log, it is possible that the event log represents the history of multiple underlying, real-life processes. Ferreira et al. [10] demonstrate that the use of sequence clustering algorithms can be useful to detect clusters of frequently occurring sequences. As the obtained event log contains cases that deal with a variety of customer problems – there are over one hundred different *problem codes* – it can be assumed that the event log is actually generated by many underlying processes. Unfortunately, due to poor data quality of case data fields, we did not find a useful filtering criterion. Sequence clustering can be a useful technique to identify clusters of underlying processes [10]. However, due to the high number of underlying processes, the application of the expectation maximization algorithm for mixtures of first-order Markov models [6] did not lead to the identification of meaningful clusters of process instances.

Assumption 4. The processes take place in a structured fashion.

Structured processes take into account concerns such as prerequisites, synchronization, parallelism, and exclusiveness. Consequently, it is assumed that it is possible to summarize the underlying processes in structured process modeling languages such as workflow nets. Günther and van der Aalst [15] report on a number of real-life event logs, for which these assumptions were not valid. The authors promote a series of multi-perspective metrics and adaptive simplification and visualization techniques to bring structure to these seemingly unstructured event logs. From Fig. 2, it can be observed that there are few dominant queue routing choices. The underlying processes apparently have little structure. The latter can be attributed to the fact that the high number of unidentifiable, underlying processes each observe a different routing and by themselves are not guaranteed to occur in a structured fashion. Another reason why the routing occurs in an unstructured fashion is that the WfMS does not impose any restriction on the routing of cases.

The obtained event log to some extent differs from ideal event logs. It does not really record which activities take place, but rather records the different queues each case is routed to. Therefore, some of the assumptions that process discovery algorithms make are not valid for the event log. In particular, Assumptions 3 and 4 are clearly problematic for the obtained event log. Another difference is that the underlying processes contain no parallel behavior. Detecting concurrency is one of the more difficult aspects of process discovery. It is interesting to see how deterministic process discovery algorithms cope with these differences.

5.3. Results

In this section, we compare the mining results of Genetic Miner, AGNEs, and HeuristicsMiner. Firstly, the parameter settings, depicted in Table 3, are discussed. Foremost, it is important to try to account for the prior knowledge that no concurrent behavior is present in the obtained event log. This prior knowledge is valid because no case can be routed to or reside in several queues at the same time. As such, HeuristicsMiner's AND threshold was set to a sufficiently large value, in this case 500,000. As for AGNEs, the formulation of the impossibility of concurrent activities is even more straightforward. The reason is that this technique explicitly allows the formulation of prior knowledge, which for this case results in: $\forall a, b \in A : PriorSerial(a, b)$. In addition, the $t_{connect}$ parameter was lowered to a value of 0. The other parameter settings correspond to the empirical setup in [13]. Finally, Genetic Miner has been running for 5000 generations, with a population size of 10. These parameter settings correspond to the parameter settings in the case study described by Medeiros2007. Note that Genetic Miner does not allow to include the prior knowledge that no activities can occur concurrently.The graphical representations of the results are depicted in Figs. 3–5 respectively. Although the visual representations of the mining results already reveal a lot of information, a more thorough analysis is provided. In the next paragraphs, we address the question whether the discovered models produce use-

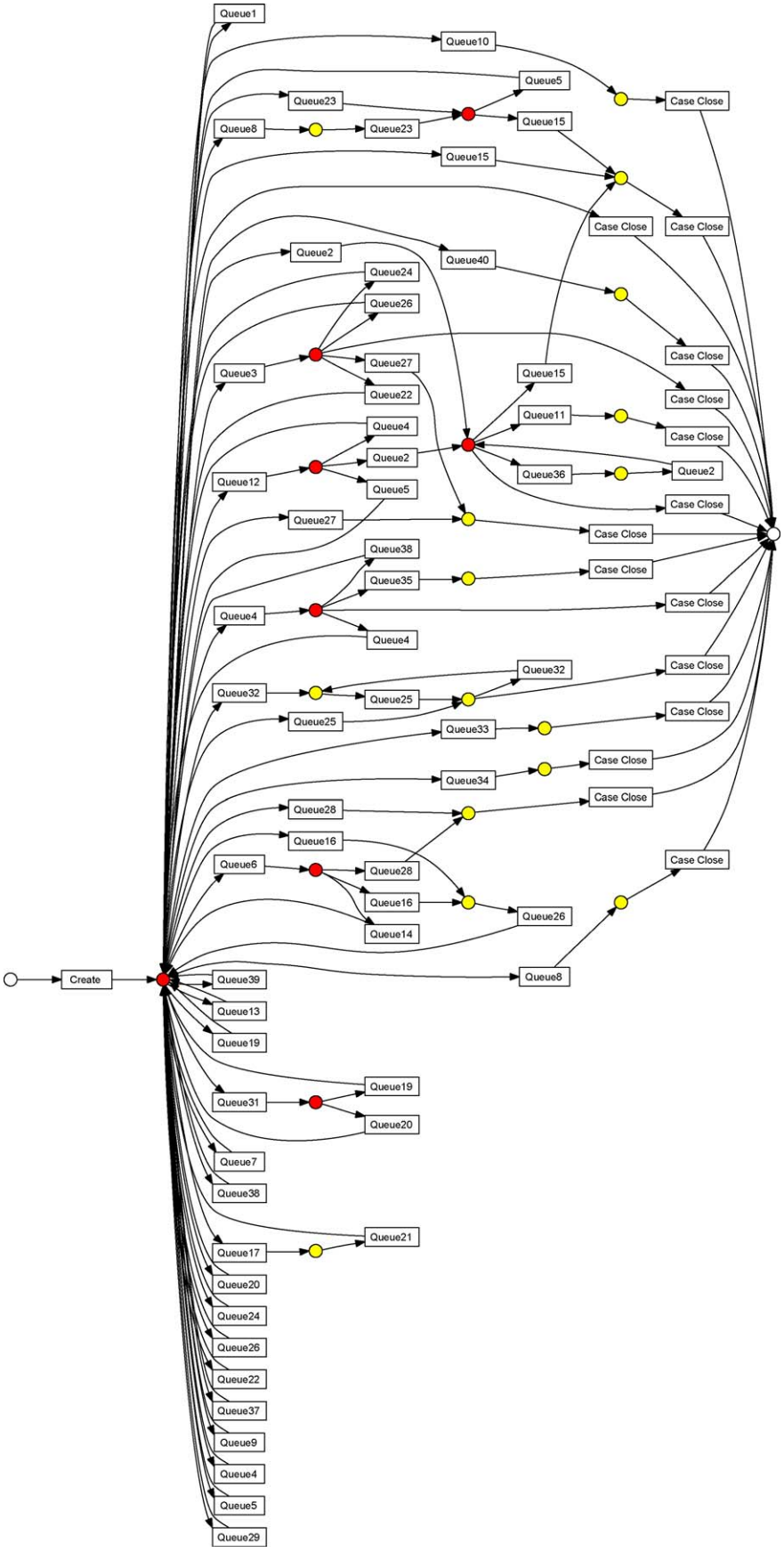

Fig. 4. Telecom – AGNEs mining results as a Petri net.

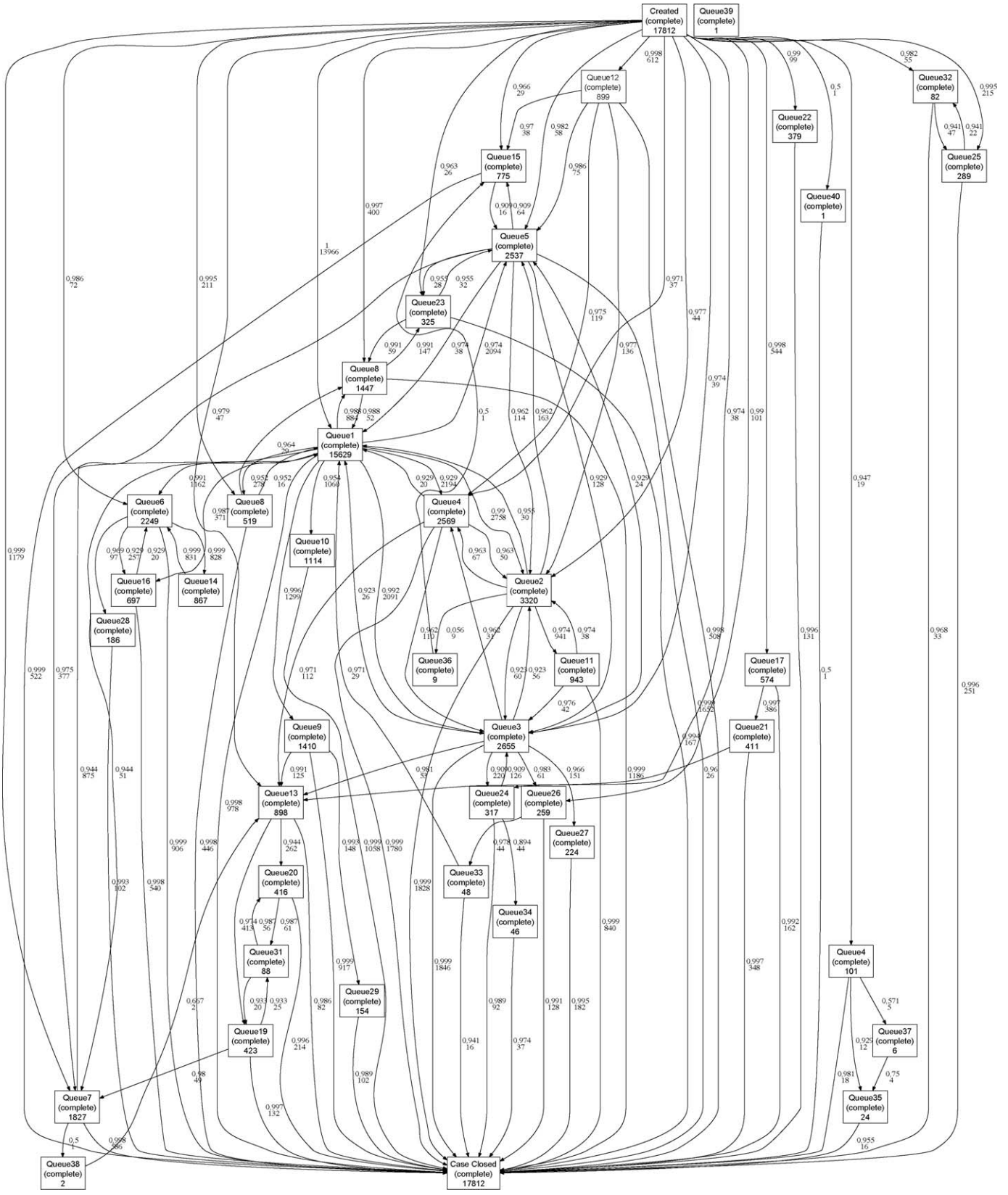

Fig. 5. Telecom – HeuristicsMiner mining results as a heuristics net.

ful information for an expert to get a good idea of the queue routing choices that are made within the second-line handling of customer problems. In particular, it is determined whether, the discovered models provide additional information compared to the graph with descriptive statistics displayed in Fig. 2. Martens [18] suggests that three general requirements make an induced classification model acceptable: accuracy, comprehensibility and justifiability. Also for process mining techniques, it is required that the learned process

Table 4
Telecom – mining results.

	PM	f	a'_B	*acc*	r_B^p	s_B^n	acc_B^{pn}	#nodes	#arcs	Run time
AGNEs	0.06	0.94	0.67	0.80	0.93	0.67	0.80	129	153	0 d:10 h:06 m:34 s
Descriptive statistics	0.73	0.86	–	–	0.94	0.80	0.87	42	139	<1 m
Flower model	0.00	1.00	0.76	0.88	1.00	0.00	0.50	47	88	00 s:032 ms
Genetic Miner	0.03	–	–	–	0.83	0.62	0.73	42	14,392	7 d:00 h:08 m:26 s
HeuristicsMiner	0.80	0.97	0.72	0.85	0.96	0.88	0.92	42	132	01 s:578 ms

models not only accurately describe the patterns in the event log, but are also comprehensible to end-users and take into account the specific domain knowledge and requirements of the end-user. Therefore, we do not solely focuss on accuracy.

5.3.1. Accuracy

Accuracy refers to the extent to which the induced model fits the behavior in the event log and can be generalized towards unseen behavior. Accuracy refers both to recall (allowing all observed behavior in the event log) and specificity (disallowing all unobserved or disallowed behavior).

The results of applying the process discovery algorithms on the obtained event log are displayed in Table 4. To calibrate the metrics, we also report their evaluation of the so-called flower model. Because the flower model represents random behavior, it has a perfect recall of all the behavior in the event log but it also has much *additional* behavior compared to the event log. Because of the latter fact, the flower model has zero specificity. These properties are to some extent reflected in the metrics in Table 4. The fitness measure f and the behavioral recall measure r_B^p are both 1.0, whereas the behavioral specificity metric s_B^n amounts to 0. The parsing measure *PM* does not reflect the recall of the flower model, most likely because the implementation requires a conversion into a heuristics net, which is very complex for a flower model. The behavioral appropriateness measure a'_B does not really seem to quantify the lack of specificity of the flower model.

Table 4 also shows the results of the visualization of queue routing of Fig. 2 that was obtained from descriptive statistics. To calculate these results, the visualization has been converted into a Petri net in the following way. Every rectangle (queue) is mapped to a labeled transition with exactly one input and output place. Each arc (queue routing) is mapped into an silent transition that connects the output place of the from-queue to the input place of the to-queue. This mapping is justifiable, because the event log contains no concurrent behavior. If the underlying processes do portray concurrent behavior, there would be no straightforward mapping of descriptive statistics into Petri nets.

Compared to the flower model, AGNEs has a fairly good recall with a fitness f of 0.94 and a behavioral recall r_B^p of 0.93. However for the sensitivity dimension, HeuristicsMiner outperforms all techniques with a fitness f of 0.97 and a behavioral recall r_B^p of 0.96. According to the r_B^p, Genetic Miner ranks last. Note that for Genetic Miner, the ProM 4.2 implementations of f, and a'_B did not produce an outcome. To calculate these metrics, a conversion of the heuristics nets into Petri nets is required. The resulting Petri nets, which have many invisible transitions, are seemingly too complex to determine a good firing sequence or to perform a state space analysis [25]. Because simple heuristics are used to determine a good firing sequence of invisible and duplicate transitions, it is possible to calculate the r_B^p and s_B^n recall and specificity metrics.

In contrast with the flower model, each of the applied process discovery techniques is able to improve significantly in regard to the specificity dimension. Looking at the available a'_B and s_B^n measures, it can be observed that the Genetic Miner result is less good than the AGNEs mining result. Nonetheless, HeuristicsMiner again outperforms the other algorithms.

Finally, when taking into account that accuracy can be seen as a weighted average of specificity and recall, it can be observed that the discovered process model by HeuristicsMiner has an accuracy of 92%, whereas the models discovered by Genetic Miner and AGNEs have an accuracy of 73% and 80% respectively. HeuristicsMiner is the only algorithm that outperforms the overall accuracy of 87% of the model obtained from the descriptive statistics.

5.3.2. Comprehensibility

Comprehensibility refers to the extent to which an induced model is comprehensible to end-users. Whereas comprehensibility is more difficult to quantify, it is an important evaluation criterium for process models. Table 4 indicates the number of arcs and nodes in each of the mining models. These numbers should be treated with care, because the number of nodes is dependent on the used process modeling languages. For Genetic Miner and HeuristicsMiner the number of places, transitions, and arcs in the heuristics net are reported, whereas for AGNEs this is done with the number of nodes and arcs in the discovered Petri net. Mendling et al. [22] have evaluated graph-based business process model understandability and quality by means of surveys and experiments. In general, the understandability of a process model deteriorates with the number of arcs and nodes in the model. Although the AGNEs mining result contains several duplicate, homonymic transitions, it has less arcs than the Genetic Miner result. However, in terms of comprehensibility, the simple visualizations in Fig. 2, obtained by descriptive statistics, and Fig. 5, obtained by HeuristicsMiner, already give a clear idea of the queue routing frequencies and queuing times. By allowing the user to filter out the arcs and nodes of which the frequency is below a particular threshold, the end-user obtains a comprehensible and configurable image of the routing choices. Notice that the comprehensibility of the descriptive statistics model is mainly due to the lack of concurrent behavior. The visualization does not allow to comprehensively represent concurrency.

5.3.3. Justifiability

Justifiability refers to the extent to which an induced model is aligned with the existing domain knowledge [19]. Domain knowledge can refer to knowledge about concurrency (parallelism), impossibility (mutually exclusive activities), etcetera. For the obtained event log, a domain expert might be surprised to see a process model in which a routing to several queues is allowed at the same time. Therefore, the inclusion of domain knowledge can be seen as a good indication of justifiability because it clearly improves the intuitiveness of mined models. For example, the AGNEs Process Discovery algorithm can be provided with the requirement that all queue routings must occur in serial. In general AGNEs allows to provide parallelism information for individual pairs of activity types. This fine-grained a priori knowledge cannot be provided to HeuristicsMiner, although fine-tuning the AND threshold accounts for a more coarse-grained induction of the parallelism information. Furthermore, it is currently not possible to constrain the search space of Genetic Miner with this a priori knowledge, inducing a discovered model containing no parallel transitions.

Table 5
Telecom – evaluation of process discovery algorithms.

	Accuracy	Comprehensibility	Justifiability	Run time
AGNEs	+	+	+	–
Descriptive statistics	+	++	+	+
Genetic Miner	+/–	–	–	–
HeuristicsMiner	++	+	+	++

5.3.4. Run time

Run time refers to the time that is required to run a particular process discovery algorithm. Table 4 indicates considerable differences in required run time on the filtered event log. Whereas a single run of HeuristicsMiner takes less than two seconds, running AGNEs and Genetic Miner takes over 10 h and over 7 days respectively. In general, process discovery is not a real-time data mining application. However, an end-user is likely to prefer process discovery algorithms that are less stringent on computational requirements. In practice, shorter run times will encourage the end-user to interactively explore the event log, applying various filters, and parameter settings [16,37]. Reporting run times is perhaps unfair. The implementations of AGNEs, HeuristicsMiner, and Genetic Miner can be considered to be scientific prototypes, built as a proof-of-concept, not as a commercial process mining algorithm. A commercial version of Genetic Miner, for instance, contains a much faster implementation of Genetic Miner [24]. The descriptive statistics from which model 2 has been constructed, were implemented as a number of SQL queries. By constructing a number of indexes on the event log, run time could be reduced to less than 30 s.

5.4. Conclusion

Table 5 summarizes the evaluation of the process discovery algorithms according to the above-mentioned criteria. A four-level scoring has been applied: ++ for the top-ranking performance, + for good performance, +/– for average performance, and – for low-ranking performance. Overall, the visualization with descriptive statistics, displayed in Fig. 2, scores very good in terms of accuracy, comprehensibility, justifiability, and run time.

Nevertheless, it can be concluded that the HeuristicsMiner algorithm provides slightly better accuracy results in comparison to the model obtained using descriptive statistics. Furthermore, one must put this outcome in the right perspective. When the underlying processes contain concurrent behavior, it is no longer possible to construct an accurate process model from descriptive statistics. Process discovery aims at discovering more complex structures such as and-splits, synchronization, exclusiveness, and non-local dependencies. Such issues are not present in the obtained event log. As such, when a real-life event log contains for example non-local, non-free choice constructs or duplicate activities, HeuristicsMiner will certainly not outperform the other process mining algorithms as it does in this case study.

6. Conclusion

Process discovery is defined as the automated construction of structured process models from event logs. In the domain of process discovery, case studies can give an idea of the validity of the assumptions and the scalability of the mining algorithms. In this paper, an empirical evaluation of three state-of-the-art process discovery techniques is brought forward. The case study revealed that different algorithms are certainly scalable towards real-life event logs. However, like other case studies [10,15,31], it was found that human-centric processes can take place in an unstructured fashion. Bringing order in the chaos of unstructured processes probably requires a different search strategy and process modeling language than the ones used by existing process discovery algorithms. As for the comparative study, it can be concluded that HeuristicsMiner outperformed the other algorithms on a majority of performance dimensions. The discovered model was more accurate and comprehensible. On the other hand, the AGNEs process discovery algorithm scores very good in terms of justifiability. It is definitely valuable to evaluate process discovery techniques not solely on accuracy, but also on other dimensions such as comprehensibility and justifiability in order to discover useful and acceptable process models. This thorough discussion of the validity issue is one important contribution of this paper. Furthermore, an in-depth discussion of a number of different evaluation metrics for process discovery techniques is definitely valuable. Finally, the paper clearly contributes to the literature by providing a detailed empirical evaluation and benchmark study of different process discovery algorithms.

Acknowledgment

We would like to thank the Flemish Research Council for financial support under Odysseus grant B.0915.09.

References

[1] G. Alonso, P. Dadam, M. Rosemann (Eds.), Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24–28, 2007, Proceedings, vol. 4714 of Lecture Notes in Computer Science, Springer, 2007.
[2] R. Alur, T.A. Henzinger, A really temporal logic, Journal of the ACM 41 (1) (1994) 181–204.
[3] A.K. Alves de Medeiros, B.F. van Dongen, W.M.P. van der Aalst, A.J.M.M. Weijters, Process mining: extending the alpha-algorithm to mine short loops, BETA working paper series 113, Eindhoven University of Technology, 2004.
[4] A.K. Alves de Medeiros, A.J.M.M. Weijters, W.M.P. van der Aalst, Genetic process mining: an experimental evaluation, Data Mining and Knowledge Discovery 14 (2) (2007) 245–304.
[5] H. Blockeel, L. De Raedt, Top-down induction of first-order logical decision trees, Artificial Intelligence 101 (1–2) (1998) 285–297.
[6] I. Cadez, D. Heckerman, C. Meek, P. Smyth, S. White, Model-based clustering and visualization of navigation patterns on a web site, Data Mining and Knowledge Discovery 7 (4) (2003) 399–424.
[7] J. Cortadella, M. Kishinevsky, L. Lavagno, A. Yakovlev, Deriving Petri nets from finite transition systems, IEEE Transactions on Computers 47 (8) (1998) 859–882.
[8] M. Dumas, W.M.P. van der Aalst, A.H.M. ter Hofstede, Process Aware Information Systems: Bridging People and Software Through Process Technology, Wiley–Interscience, 2005.
[9] A. Eiben, J. Smith, Introduction to Evolutionary Computing (Natural Computing Series), Springer-Verlag, 2003.
[10] D.R. Ferreira, M. Zacarias, M. Malheiros, P. Ferreira, Approaching process mining with sequence clustering: experiments and findings, in: [1], 2007, pp. 360–374.
[11] Z.-p. Gao, J. Chen, X.-s. Qiu, L.-m. Meng, QoE/QoS driven simulated annealing-based genetic algorithm for Web services selection, The Journal of China Universities of Posts and Telecommunications 16 (September) (2009) 102–107.
[12] S. Garner, WEKA: the waikato environment for knowledge analysis, 1995, http://www.cs.waikato.ac.nz/ml/weka/.
[13] S. Goedertier, D. Martens, J. Vanthienen, B. Baesens, Robust process discovery with artificial negative events, Journal of Machine Learning Research 10 (2009) 1305–1340.
[14] S. Goedertier, C. Mues, J. Vanthienen, Specifying process-aware access control rules in SBVR, in: A. Paschke, Y. Biletskiy (Eds.), Proceedings of the International Symposium Advances in Rule Interchange and Applications (RuleML 2007), vol. 4824 of Lecture Notes in Computer Science, Springer, 2007, pp. 39–52 (Best Paper Award).
[15] C.W. Günther, W.M.P. van der Aalst, Fuzzy mining – adaptive process simplification based on multi-perspective metrics, in: [1], 2007, pp. 328–343.
[16] M. Hammori, J. Herbst, N. Kleiner, Interactive workflow mining – requirements, concepts and implementation, Data & Knowledge Engineering 56 (1) (2006) 41–63.
[17] S. Mansar, H.A. Reijers, Best practices in business process redesign: validation of a redesign framework, Computers in Industry 56 (5) (2005) 457–471.
[18] D. Martens, Building acceptable classification models for financial engineering applications: thesis summary, SIGKDD Explorations 10 (2) (2008) 30–31.
[19] D. Martens, M. De Backer, R. Haesen, B. Baesens, C. Mues, J. Vanthienen, Ant-based approach to the knowledge fusion problem, in: Proceedings of the Fifth International Workshop on Ant Colony Optimization and Swarm Intelligence. Lecture Notes in Computer Science, Springer, 2006, pp. 85–96.

[20] D. Martens, T. Van Gestel, B. Baesens, Decomposi-tional rule extraction from support vector machines by active learning, IEEE Transactions on Knowledge and Data Engineering 21 (2) (2009) 178–191.
[21] A.K.A.D. Medeiros, A.J.M.M. Weijters, Genetic process mining: a basic approach and its challenges, in: Busines Process Management 2005 Workshops, Springer-Verlag, 2006, pp. 203–215.
[22] J. Mendling, H.A. Reijers, J. Cardoso, What makes process models understandable? in: [1], 2007, pp. 48–63.
[23] T. Murata, Petri nets: properties, analysis and applications, Proceedings of the IEEE 77 (4) (1989) 541–580.
[24] P. Athena, BPM|Protos – process modeling and mining tool, 2008, http://www.pallas-athena.com.
[25] A. Rozinat, W.M.P. van der Aalst, Conformance checking of processes based on monitoring real behavior, Information Systems 33 (1) (2008) 64–95.
[26] S. Sadiq, G. Governatori, K. Namiri, Modeling control objectives for business process compliance, in: [1], 2007, pp. 149–164.
[27] P.-N. Tan, M. Steinbach, V. Kumar, Introduction to Data Mining, Addison-Wesley, 2005.
[28] W.M.P. van der Aalst, Verification of workflow nets, in: P. Azéma, G. Balbo (Eds.), Proceedings of the 18th International Conference on the Application and Theory of Petri Nets 1997 (ICATPN'97), vol. 1248 of Lecture Notes in Computer Science, Springer, 1997, pp. 407–426.
[29] W.M.P. van der Aalst, The application of Petri nets to workflow management, Journal of Circuits, Systems, and Computers 8 (1) (1998) 21–66.
[30] W.M.P. van der Aalst, Exploring the CSCW spectrum using process mining, Advanced Engineering Informatics 21 (2) (2007) 191–199.
[31] W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, H.M.W. Verbeek, Business process mining: an industrial application, Information Systems 32 (5) (2007) 713–732.
[32] W.M.P. van der Aalst, V. Rubin, B.F. van Dongen, E. Kindler, C.W. Günther, Process mining: a two-step approach using transition systems and regions, BPM-06-30, BPM Center Report, 2006.
[33] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, A.J.M.M. Weijters, Workflow mining: a survey of issues and approaches, Data & Knowledge Engineering 47 (2) (2003) 237–267.
[34] W.M.P. van der Aalst, K.M. van Hee, Workflow Management: Models, Methods, and Systems, MIT Press, Cambridge, MA, 2002.
[35] W.M.P. van der Aalst, A.J.M.M. Weijters, Process mining: a research agenda, Computers in Industry 53 (3) (2004) 231–244.
[36] W.M.P. van der Aalst, A.J.M.M. Weijters, L. Maruster, Workflow mining: discovering process models from event logs, IEEE Transactions on Knowledge and Data Engineering 16 (9) (2004) 1128–1142.
[37] B.F. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, W.M.P. van der Aalst, The ProM framework: a new era in process mining tool support, in: G. Ciardo, P. Darondeau (Eds.), Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005), vol. 3536 of Lecture Notes in Computer Science, Springer, 2005, pp. 444–454.
[38] B.F. van Dongen, W.M.P. van der Aalst, Multi-phase process mining: aggregating instance graphs into EPCs and Petri nets, in: Proceedings of the 2nd International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management (PNCWB), 2005.
[39] K.M. van Hee, O. Oanea, A. Serebrenik, N. Sidorova, M. Voorhoeve, History-based joins: semantics, soundness and implementation, in: S. Dustdar, J. Fiadeiro, A. Sheth (Eds.), Business Process Management, vol. 4102 of Lecture Notes in Computer Science, Springer, 2006, pp. 225–240.
[40] K.M. van Hee, O. Oanea, A. Serebrenik, N. Sidorova, M. Voorhoeve, Modelling history-dependent business processes, in: MSVVEIS, 2006, pp. 76–85.
[41] K. Vergidis, a. Tiwari, B. Majeed, R. Roy, Optimisation of business process designs: an algorithmic approach with multiple objectives, International Journal of Production Economics (1–2) (2007) 105–121.
[42] B. Weber, S. Rinderle, M. Reichert, Change patterns and change support features in process-aware information systems, in: J. Krogstie, A.L. Opdahl, G. Sindre (Eds.), CAiSE, vol. 4495 of Lecture Notes in Computer Science, Springer, 2007, pp. 574–588.
[43] A.J.M.M. Weijters, W.M.P. van der Aalst, Rediscovering workflow models from event-based data using little thumb, Integrated Computer-Aided Engineering 10 (2) (2003) 151–162.
[44] A.J.M.M. Weijters, W.M.P. van der Aalst, A.K. Alves de Medeiros, Process mining with the heuristicsminer algorithm, BETA working paper series 166, Eindhoven University of Technology, 2006.
[45] L. Wen, W.M.P. van der Aalst, J. Wang, J. Sun, Mining process models with non-free-choice constructs, Data Mining and Knowledge Discovery 15 (2) (2007) 145–180.
[46] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.
[47] M. zur Muehlen, Workflow-based process controlling, in: Foundation, Design, and Implementation of Workflow-driven Process Information Systems, vol. 6 of Advances in Information Systems and Management Science, Logos, Berlin, 2004.

Stijn Goedertier received a Master's degree in Business Engineering and a PhD in Applied Economic Sciences from K.U.Leuven (Belgium) in 2004 and 2008 respectively. He has done research in the field of Business Process Management, declarative process modeling and process discovery. He is currently employed as a business advisor at Pricewaterhouse-Coopers.

Jochen De Weerdt received a Master's degree in Business Engineering from the Katholieke Universiteit Leuven (K.U.Leuven), Leuven, Belgium. He is currently employed as a doctoral researcher at the Department of Decision Sciences and Information Management at the Katholieke Universiteit Leuven (K.U.Leuven). His research interests include data mining, process mining, and web intelligence.

David Martens received a Master's degree in computer engineering and a PhD in Applied Economic Sciences from K.U.Leuven (Belgium) in 2003 and 2008 respectively, and a Master of Business Administration (M.B.A.) in 2005 from Reims Management School (France). He is lecturer at Hogeschool Gent, and post-doctoral researcher at K.U.Leuven. His research is focused on the development of comprehensible data mining techniques, using SVM rule extraction and Ant Colony Optimization. The main application of his research can be found in the financial engineering domain with the development of Basel II-compliant credit scoring systems.

Jan Vanthienen received the Ph.D. degree in Applied Economics (information systems) from the Katholieke Universiteit Leuven (K.U.Leuven), Leuven, Belgium. He is a Full Professor of information systems with the Department of Decision Sciences and Information Management, K.U.Leuven. He is also Chairholder of the Pricewaterhouse-Coopers Chair on E-Business at K.U.Leuven. He is the author or coauthor of numerous papers published in international journals and conference proceedings. His current research interests include information and knowledge management, business intelligence and business rules, and information systems analysis and design.

Professor Bart Baesens is an associate professor at K.U.Leuven (Belgium), and a lecturer at the University of Southampton (United Kingdom). He has done extensive research on predictive analytics, data mining, customer relationship management, fraud detection, and credit risk management. His findings have been published in well-known international journals (e.g. Machine Learning, Management Science, IEEE Transactions on Neural Networks, IEEE Transactions on Knowledge and Data Engineering, IEEE Transactions on Evolutionary Computation, Journal of Machine Learning Research, Ẽ) and presented at international top conferences. He is also co-author of the book Credit Risk Management: Basic Concepts, published in 2008.