A Study of Quality and Accuracy Tradeoffs in Process Mining

Zan Huang and Akhil Kumar
Smeal College of Business, Pennsylvania State University, University Park, PA 16802, USA,
{zanhuang,akhilkumar}@psu.edu

In recent years, many algorithms have been proposed to extract process models from process execution logs. The process models describe the ordering relationships between tasks in a process in terms of standard constructs like *sequence, parallel, choice* and *loop*. Most algorithms assume that each trace in a log represents a correct execution sequence based on a model. In practice, logs are often noisy, and algorithms designed for correct logs are not able to handle noisy logs. In this paper we share our key insights from a study of noise in process logs both real and synthetic. We found that **all** process logs can be explained by a block-structured model with two special *self-loop* and *optional* structures, making it trivial to build a fully accurate process model for *any* given log, even one with inaccurate data or noise present in it. However, such a model suffers from low *quality*. By controlling the use of self-loop and optional structures of tasks and blocks of tasks, we can balance the quality and accuracy tradeoff to derive high-quality process models that explain a given percentage of traces in the log. Finally, new quality metrics, and a novel quality-based algorithm for model extraction from noisy logs are described. The results of the experiments with the algorithm on real and synthetic data are reported and analyzed at length.

Keywords : process mining, knowledge discovery, quality metric, quality-accuracy trade-off, quality-based algorithm.

1. Introduction

Recently, there has been considerable interest in extracting process models from logs of actual execution instances (van der Aalst, van Dongen et al. 2003; Weijters and van der Aalst 2003; Dustdar, Hoffmann et al. 2005; van der Aalst, de Medeiros et al. 2007; van der Aalst, van Dongen et al. 2007). There are several reasons for the need to extract a model from an actual log. First, it enables better understanding of the process. Second, it also helps in the comparison of the actual process with a conceptualized process, and allows suitable modifications to be made

and discrepancies reported. In many real-world applications the process model extracted from actual logs may differ drastically from the idealized process that might have been designed many years ago, suggesting that the idealized process is not being enforced and the extracted model can provide guidance for modifying it. Thirdly, one may improve an existing process by finding opportunities for exploiting parallelism by examining the process model to increase throughput. Finally, it is also possible to run queries against a process model (Klein and Bernstein 2004).

Process mining is an emerging research area that has opened up new possibilities of knowledge discovery. It complements data mining (Tan, Steinbach et al. 2006) with many practical applications. In healthcare, it is possible to extract the process followed to provide care to a patient and identify anomalies that may result in lack of proper treatment (Song and van der Aalst 2007; Mans, Schonenberg et al. 2008). Moreover, based on such analysis corrective action may be taken. It has also been applied in healthcare fraud and abuse detection (Yang and Hwang 2006), and in compliance (Chesani, Mello et al. 2008). Process mining even has applications in social mining (van der Aalst, Reijers et al. 2005; van der Aalst, Reijers et al. 2007), but that is not the focus of this paper.

Number	Log instance
1	A C
2	B D
3	C L2 C A
4	D A L1 B
5	A L1 C L2 C B
6	A C L2 L1 C B
7	B D L2 L1 D B
8	D A L1 L2 B C
9	D L2 B L1 C A
10	A
11	D A L1 B B
12	A L1 A

Table 1: An example log

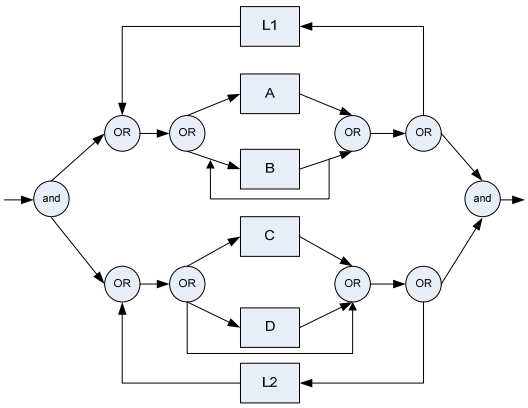


Figure 1: A business process model

Figure 1 shows a process model consisting of six tasks, A, B, C, D, L1 and L2. This model is drawn using **and-splits, and-joins, or-splits** and **or-joins**. The **and**-node pairs can capture *parallelism* while the **or**-node pairs capture *choice* and *loops*. Notice how the *choice* structures are nested inside *loop* structures, and two loops in turn are inside a *parallel* structure. This model was derived (using our algorithm to be described later) from the execution log of Table 1. In each row, there is one log instance (or trace) showing the actual sequence in which the tasks were performed. Though not shown, each instance has a *start* task and an *end* task. Two other features in Figure 1 are the *optional block* around tasks C and D, and the *self-loop* around task B.

This means that the C-D block may optionally be skipped, while B can be repeated multiple times. These constructs are required to explain the log instances 10-12 in Table 1. These two new constructs play an important role in accommodating noise in process logs.

Thus, the problem of process mining is to extract the process model from a log. It should be noted that we do not assume a "complete" log; therefore the observed log instances are treated as positive examples, while an unobserved instance is *not* interpreted as a negative example. In other words, we intend to find a model that can sufficiently explain the log instances observed but do not restrict the model from allowing any unobserved behavior. As will be discussed in detail later, the quality of a model is influenced by the amount of unobserved behavior it also permits. The log may consist of *start* and *end times* of each task; however, for simplicity we assume here that the tasks are *atomic* and the start event of a task which appears in a log is treated as both its start and end. This treatment is common for the vast majority of work in process mining. We consider a log with both start and end times as a special case of atomic task logs, as a task A with start and end times can be treated as a sequence block consisting of two artificial tasks, *A-start* and *A-end*. In general, under the atomic task representation with, say, tasks A and B, both patterns 'A B' and 'B A' need to be present in the log to determine they are in parallel. On the other hand, under the presence of start/end time, just one instance containing 'A-start B-start A-end B-end' suffices.

In recent years, there have been several efforts in this area. The main objective is to develop algorithms that can analyze logs and construct correct models that reflect the actual workflow. Consequently, research efforts have focused on extracting general graphs (Agrawal, Gunopulos et al. 1998) and more specialized graphs such as Petri-nets (van der Aalst, Weijters et al. 2004) from logs.

In our study, we focus on extracting block-structured process models (Schimm 2000; Schimm 2003) from noisy data. Block-structured models are easier to describe and understand from an end-user point of view compared to other process models in the literature, such as Petri net and graph models. More importantly, they correspond closely to the widely adopted *Unified Modeling Language (UML)* constructs for describing business processes, such as the *Activity Diagram*. In our study, we extend the basic block-structured model by introducing two special *self-loop* and *optional* structures in order to represent the commonly observed execution patterns in real-world transaction logs. An important consequence of these two special structures is that

the abuse of these two structures can make any given model of the four basic structures (referred to as a *pure model* in this paper) become *universal models that can explain any log*. Of course, such models are not very useful since they can explain all logs and are overly general. Therefore we introduce in this paper a new *quality metric* that assesses how specific a mined block-structured model is to the input log. An overly general universal model has the lowest possible quality value, while models permitting fewer log execution patterns have higher quality values.

Allowing self-loop and optional structures leads to less specific models, but at the same time our algorithm is tolerant to noisy data. A process log is *noisy* if it contains spurious trace instances that result from data collection errors and execution exceptions such as a missed task and exceptional repetition of some tasks or subprocesses. Of course, we assume that the algorithm has no way of knowing which specific log instances are noisy. Our algorithm operates by first finding the most likely pure model with the four basic structures and then determining the minimal additional self-loop and optional structures needed for explaining a pre-specified percentage of log instances. With an appropriately designed quality metric, our problem of learning a block-structured process model from a noisy log becomes one of *finding the highest quality model to correctly explain, say, X% of the instances in the log*.

Note that the notion of process model quality in our paper is a technical concept dealing with the specificity of a given block-structured process model. Our quality metric does not relate to the subjective quality perceived by users such as understandability (Mendling, Reijers et al. 2007). It is also different from the notion of quality in conceptual modeling (Lindland, Sindre et al. 1994), which consists of syntactic, semantic and pragmatic quality. It is in line with the specificity or precision measures in the process mining literature that characterize how much more behavior is captured in the model than was observed in the log (Rozinat, de Medeiros et al. 2007; Rozinat, de Medeiros et al. 2007; Rozinat and van der Aalst 2008). Also, our quality metric is intended to be used together with the accuracy measure to select the best model from the ones that satisfy the accuracy requirement. It is not intended to be a general process model quality measure.

As further motivation for our approach, we note that current approaches fall short in certain scenarios. We tried the contrived log of Table 1 on several algorithms in the well-known ProM toolset (van der Aalst, van Dongen et al. 2007), including Alpha++ (Wen, Wang et al. 2006), Heuristic Miner (Weijters and van der Aalst 2003), Multi-Phase Miner (van Dongen and van der

Aalst 2004), Parikh language-based region miner (van der Werf, van Dongen et al. 2008) and Genetic Miner (van der Aalst, de Medeiros et al. 2007). Despite varying parameter settings over a wide range, none of them was able to discover the process of Figure 1. The best performing algorithms among these seemed to be the Genetic Miner (if running time is not considered), for which in one parameter setting the resulting model could explain ten of the 12 instances. The next best were Heuristic Miner and Multi-phase Miner, which resulted in models that can successfully explain one of the twelve log instances. We will report additional detailed results on the performance of these existing algorithms in the experimental study section of the paper. We do not claim any superiority of our approach to existing ones; rather, we offer it only as a somewhat new perspective on process mining that works well with certain kinds of log where other methods may not work so well.

In subsequent sections we explain our approach in detail, and also give results of testing our algorithm on both synthetic and real data. Section 2 gives a background about how block structured workflows are constructed using modeling structures and the principles for developing quality metrics. Section 3 defines our metrics. Then Section 4 describes our algorithm for process discovery from log instances. Next, we give actual results and analysis from running the algorithm on synthetic logs (Section 5) and a large-scale real-world log (Section 6) on the patent application domain. Later, Section 7 gives further discussion, analysis and insights, and also an overview of related work. Finally, Section 8 concludes the paper.

2. Block-structured Process Model

We start this discussion by introducing four basic structures in block structured processes along with the optional and self-loop structures. It should be noted at the outset that every workflow process also has two distinguished "Start" and "End" tasks (exactly one each) to respectively denote the start and end of a process. We limit ourselves to structures that combine only two tasks or blocks at a time; however, there is no loss of generality because structures that represent higher branching factors can be represented in terms of these structures.

2.1 Block Structures – Sequence, Parallel, Choice, Loop, Optional and Self-loop

A *block structured process* is created by using four basic structures as building blocks: *sequence, parallel, choice* and *loop* ("S", "P", "C" and "L"). In a block structured model atomic tasks are combined into sub-blocks, which are further combined into larger sub-blocks using one of these four basic structures, until the full model is created. If a log corresponds exactly to such a pure block structured model, then one can limit oneself to just these four structures and discover a model from a log. Such a model is called a "pure" model.

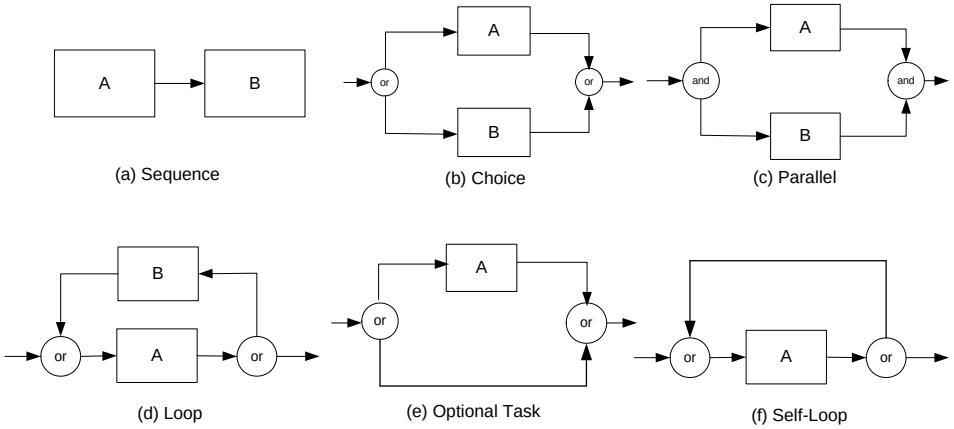
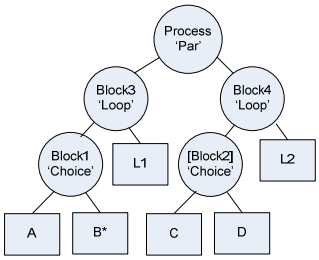


Figure 2: Types (or patterns) used in block structured workflows

However, in real logs that we observed, it was seldom the case that a log could be explained perfectly with a pure model. In particular, we found that some tasks are often inherently optional in the business process, and appear in some log instances but not in others. Another common situation in real logs is that some tasks are repeated. Thus, a log may contain an entry like AABCBC to suggest that task A was done twice followed by two occurrences of tasks B and C in succession. Such a log cannot be described using the four basic primitives of the block structured model because it has a self-loop and we disallow duplicate tasks. In order to capture these two common scenarios two additional structures, *optional* (*OP*) and *self-loop* (*SL*), are introduced. An optional structure (see Figure 2(e)) is a choice with one empty branch. A self-loop (Figure 2(f)) is a loop of a single task/block. These can create severe complications in the discovery process and have to be distinguished from normal loops. In the remaining discussion we use the terms *task* for atomic, lowest level tasks, and the general term *sub-block or block* for both an individual task and an aggregate structure. It should also be noted that the self-loop in

Figure 2(f) can create a *livelock*. However, it is reasonable to assume that a correct business process will allow exit from the self-loop when an exit condition is satisfied.

The process of Figure 1 can be represented in a succinct form as a model tree and accompanying list of optional and self-loop markings as shown in Figure 3(b). Figure 3(a) gives the graphical representation of the model tree. Here a * is used to indicate a self-loop, while square brackets around a task or block name mean that it is optional.



(a) A process tree

Model.tuples =
{(A, B, 'Choice', Block1),
(C, D, 'Choice', Block2),
(Block1, L1, 'Loop', Block3),
(Block2, L2, 'Loop', Block4),
(Block3, Block4, 'Parallel', Process)}

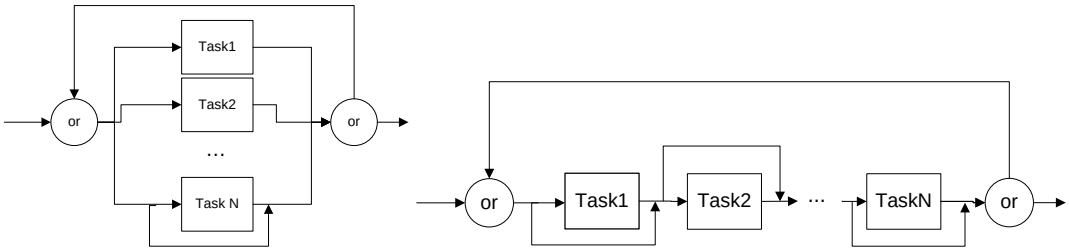
Model.SLOP = {B_sl, Block2_op}

(b) Tuple representation of the model

Figure 3: Model representation as tree and as merge tuples and SL/OP markings

The six block structures included in our model correspond to the basic workflow patterns identified in (van der Aalst, ter Hofstede et al. 2004). For example, our P structure corresponds to the parallel split and synchronization patterns; our C structure corresponds to the exclusive choice and simple merge patterns; and our SL structure corresponds to the arbitrary cycle pattern. There are many advanced workflow patterns that are not captured in our model, such as multi-choice in (van der Aalst, ter Hofstede et al. 2004), the so-called "wheatstone bridge" structure, and non-free-choice. Not allowing duplicate tasks, our approach does not lead to very specific, but otherwise meaningless, models where each log instance is trivially converted into a simple sequence of tasks, and each such sequence is combined in a choice pattern. Yet we believe our structures are expressive enough to capture most workflow patterns considered by existing process mining algorithms. In fact, our block-structure model can be considered as a *restricted* Petri net that allows invisible tasks (both self-loop and optional structures can be expressed in Petri net using invisible tasks). Our model is restricted in the sense that invisible transitions are only introduced for self-loop and optional structures, while the general Petri nets allow invisible transitions in much more complicated patterns. We believe this restriction is actually an advantage as it makes the model easier to learn and there is little or no sacrifice of expressiveness.

2.2 Universal Process Models



(a) Choice in a self-loop (SL) (b) Sequence of optional (OP) tasks in a self-loop

Figure 4: Two universal process models that can explain any log

Figure 4(a) shows a model that can describe any given log, even a random one! This model consists of a choice node with *N* branches to the *N* different tasks. Note that to strictly follow our earlier discussion this choice node with *N* branches would actually be achieved through multiple choice nodes with two branches in a hierarchy. Any one of the *N* branches may be taken at the choice node. Moreover, the entire process is in a self-loop. Thus, this process keeps repeating as many times as the number of entries in a log instance. Figure 4(b) is another universal model. Both models in Figure 4 correspond to the so-called "flower" model (Rozinat, de Medeiros et al. 2007; Rozinat and van der Aalst 2008) often used as an example to illustrate overly general Petri nets. *These models can simulate any log.* Other universal process models can be made in a similar way. In fact, any given pure model in our context can be converted into a universal model by marking every task and block as both optional and self-loop.

2.3 Desirable Properties of a Quality Metric and Quality-accuracy Trade-off

A major problem with the universal model is that its *quality* is *bad* since it can simulate *every* log. Thus, two logs with the same set of tasks can always be explained by the same model! Hence, this model lacks *specificity*. This has given rise to the issue of process model quality with a need for distinguishing good models from bad ones (Rozinat, de Medeiros et al. 2007). Process models mined from logs are mainly evaluated in two aspects: *fidelity* (degree to which the model explains a log) and *specificity* (degree to which a model is specific to a given log). Fidelity is also referred to as *fitness* (Rozinat, de Medeiros et al. 2007; Rozinat and van der Aalst 2008) and specificity is also referred to as *appropriateness* and *precision* (Rozinat, de Medeiros et al. 2007; Rozinat and van der Aalst 2008).

(Rozinat, de Medeiros et al. 2007) provides a summary of existing metrics for these concepts. Fitness measures the percentage of log instances (e.g., *completeness* (Greco, Guzzo et al. 2006)) or percentages of individual tasks (e.g., $PF_{complete}$ (van der Aalst, de Medeiros et al. 2007)) that can be explained by the model. The appropriateness measures capture how much behavior is allowed by the model which was not necessary for the log (*behavioral appropriateness* (Rozinat and van der Aalst 2008)), not included in the true model (*structural precision* (van der Aalst, de Medeiros et al. 2007)), or not included in the true model evaluated at the execution of the log (*behavioral precision* (van der Aalst, de Medeiros et al. 2007)). Further, *structural and behavioral recall* measures (van der Aalst, de Medeiros et al. 2007) define how much behavior of the true model is actually captured in the mined model. These measures are defined on a Petri net.

Ideally, a perfect model for a given log is one with both high fidelity and specificity. A unique feature of the block-structured models in our study is that given any pure model we can always impose appropriate self-loop and optional structures to achieve a model with perfect fidelity or fitness, or 100% *accuracy* (the percentage of log instances that can be explained by the model). Therefore, our quality metric focuses on the specificity aspect of a given model. There is a natural trade-off between accuracy and quality of a process model. End-users can decide whether it is beneficial to require less-than-perfect accuracy, discarding possible noisy or legitimate but rare log instances to find models of better quality. Existing model specificity measures are defined with respect to the true model and/or the log. For practical applications, the true model is usually not available. Further, coupling the model with the log to define the quality metric may bring unnecessary conceptual and computational burden. The following intuitive axioms enumerate some principles to define a quality metric for a process model independent of the log or the true model:

Axiom 1: A universal model that can describe **every log** has the worst quality.

Axiom 2: All universal models should have an equal quality score.

Axiom 3: The model that permits more unique instances should have a lower quality value.

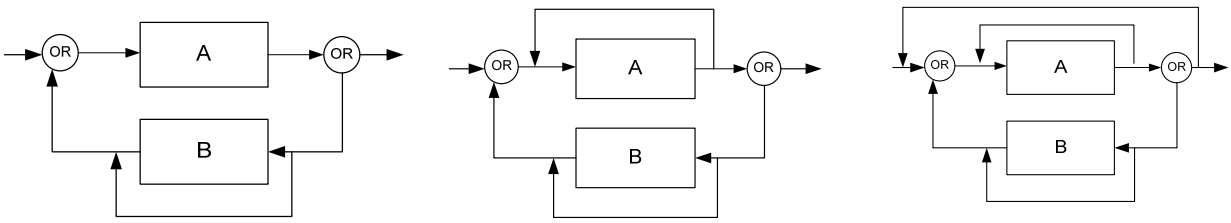
Although we define the quality metric independent of the log, it is important to note that our quality metric is intended to be used in the context of evaluating different models mined from a given log. It is not intended to serve as a general quality metric for process models. In the context of our block-structured models with SL and OP structures, because a log can be explained by

many different models, when two models can explain a log we want to pick the one that is more specific. Thus, a log consisting of just two tasks A and B can be explained by a model in which A and B are in sequence, and also one in which they are in parallel. However, the latter model also allows a trace "B A". Hence, in this case we would assign a higher quality value to the first model than to the second one. These axioms will apply to our quality metric which is developed in the next section.

3. An Approach for Process Model Quality Metrics

3.1 Model Equivalence

Our quality metrics are based on the notion of *equivalent models.* Two models are *equivalent* if they allow exactly the same execution sequences to occur (Huang and Kumar, 2008). Figure 5 shows three different models. In fact, all three are identical and one can also verify that they allow the same execution sequences. A sequence like "A B A A B A A B A" can be executed on any of the three models. Yet the SL and OP markings are different as shown on the figure labels.



(a) Minimal equivalent model
SL= (0,0,0), OP=(0,1,0)

(b) Equivalent model
SL=(1,0,0), OP=(0,1,0)

(c) Maximal equivalent model
SL= (1,0,1), OP=(0,1,0)

Figure 5: Three equivalent models

Arguably the model with the least number of markings shown in Figure 5(a) is the best in the sense that it is most succinct, and yet it has the same expressive power as the other two. The one in Figure 5(c) is the most verbose. Therefore, among multiple models, the one with the least SL and OP markings is called the *minimal equivalent model* and the one with the most is called the *maximal equivalent model*. By enumerating sequences of reasonable length for all (S, P, C, or L) combinations of two tasks, and SL and OP markings (256 combinations of the atomic structures), we can determine equivalent models and also the maximal and minimal equivalent models in a set of equivalent models. We use the maximum equivalent model to compute the quality metric for a model as explained shortly, but the minimal model is easier to visualize. In general, in a

hierarchical block-structured model, the translation of a model into an equivalent maximal (or minimal) one is done recursively from the leaf level of the process model tree. Because the rewriting can cause the child self-loop/optional (SLOP) structures to affect the parent SLOP structures and vice-versa, we iterate until convergence is achieved, i.e. there is no change in successive iterations. For a different concept of equivalence between two EPC diagrams based on similar events and function in two models, see (Mendling and Simon, 2006).

3.2 Badness Score and Quality Metrics

Our *Quality metric* relies on a *Badness score* (a.k.a. *Q-metric* and *B-value*, respectively). This score measures the badness of a model where, intuitively, a model with more SL and OP structures is "more bad" than one with fewer of them. We choose to focus on the SL and OP structures as they are mainly responsible for additional behavior allowed by the model (universal models as the extreme). We determine the badness score of a model based on its maximal equivalent model in order to maintain consistency across equivalent models. The reason for rewriting each combination with the maximal one, as opposed to the minimal, is to ensure that all universal models have the same worst quality. Even though the models in Figures 4 and 5 look different, yet these models and many other universal models are equivalent. Without rewriting, it is impossible to design a consistent scoring rule to assign the same quality value to them. The *maximal equivalent universal model* is one where *every task and block is both in a self-loop and is optional*. It is guaranteed to have the worst value if the quality metric is defined on the SL and OP structures by Axiom 1 and Axiom 2 in Section 2.3. We use a recursive formula as follows:

$$B(n)=\begin{cases}(1+W_1\cdot op(n)+\ W_2\cdot sl(n))\cdot(B(n_1)+B(n_2)), & n \text{ is a block and } op(n)+\ sl(n)>0\\ (1+W_3)\cdot(B(n_1)+B(n_2)), & n \text{ is a block}, op(n)+\ sl(n)=0,\ p(n)=1\\ 1+W_1\cdot op(n)+W_2\cdot sl(n), & n \text{ is a task}\end{cases}$$

where n_1, n_2 are child nodes of node *n,* and *op*(), *sl*(), and *p*() are 0-1 functions to indicate whether a given block has an optional, self-loop, or parallel structure, respectively. W_1 and W_2 are weighting factors to vary the influence of SL and OP tasks/blocks. W_3 attempts to penalize the parallel structure as it completely dominates the sequence structure. In other words, when the underlying model has a sequence structure both sequence and parallel structures can perfectly explain the log instances, but the sequence structure is more specific. (A, B, 'Parallel') allows two paths, while (A, B, 'Sequence') only one. Hence, the latter is more specific and it should receive a better score than the former (see Axiom 3). However, in our calculation this adjustment is not

applied to the blocks that are of OP or SL structure because no clear dominance of a parallel structure over a sequence is present. If we think of a process as a tree the B-value of the root will give the badness of the process.

In order to satisfy Axiom 3, we have analyzed the correlation between the number of possible log instances of the 256 atomic structures introduced in Section 3.1 and their badness scores with different combinations of W_1, W_2, and W_3. Based on this analysis, we set W_1, W_2, and W_3 to be 1, 5, and 0.5, which produced a Spearman rank order correlation coefficient (Spearman 1904) of 0.9318. We used Spearman rank order correlation for this analysis because we consider the badness score to be appropriate if it successfully ranks the atomic structures according to their numbers of possible log instances. After all, the number of log instances is generally dependent on the maximum log instance length allowed. From the parameter settings we can see that the SL structure has much greater influence on the specificity of the model than the OP structure.

Our proposed badness score is an approximate measure that is relatively easy to understand and compute, yet it adequately reflects the notion of model specificity. In general, a model with a smaller badness score is considered "better" than one with a higher score at the same accuracy level. If the true model does contain legitimate SL and OP structures then the pure models that have the minimum badness scores (i.e. which are best in terms of specificity), but a low accuracy, would not be considered good models. While recognizing that our badness score does not perfectly measure the specificity of a given process model, we must also note that such a perfect measure may not even exist. This is because, with the presence of SL and OP structures, the number of possible log instances of a process model is not well defined. Strictly speaking, any model with at least one SL structure may produce an infinite number of log instances.

Based on the badness score, we can define the quality metric of a block-structured model. As a common practice, we impose a range of [0, 1] on the metric with 1 representing the best quality, and 0 the worst. We consider two versions of the quality metric based on the badness score:

$$\text{Q1} = \frac{\text{Log}(N)}{\text{Log}(\text{Badness score of actual model})}, \quad \text{Q2} = 1 - \frac{\text{Log}(\text{Badness score of actual model} - N + 1)}{\text{Log}(\text{Badness score of universal model} - N + 1)}$$

For Q1, the numerator represents the logarithmic transformation of the badness score for the best model, which is N since it does not have any markings and does not contain any parallel structure. This measure reflects how much worse the given model is from the best model.

Similarly, Q2 reflects how much better a given model is from the worst universal model. Both Q1 and Q2 are monotonic transformations of the badness score. In fact, there are numerous ways to design a quality metric from monotonic transformations of the badness score. All such designs may work equally well for our purpose as the rank order is maintained. It is only the specific value of the quality metric that is interpreted differently.

We also include a simple quality metric based on simple counts of SL and OP structures in the model as follows:

$$Q0 = 1 - \frac{\text{Number of self loops} + \text{number of optional markings}}{4N - 2}$$

As validation for Axiom 3, we compared the number of log instances of models consisting of only 2 tasks (with the maximum length of an instance set to 6) and the quality metrics of these models. The Spearman rank order correlation coefficients for Q0, Q1, and Q2 are –0.7649, –0.9318, and –0.9318, respectively, all with *p-values* smaller than 0.001, indicating statistical significance of the association between all three quality metrics and number of possible log instances. This analysis shows that Q1 and Q2 based on the badness score are better metrics as will also be confirmed in our experimental results.

4. A Quality-based Process Mining Algorithm

4.1 Algorithm Overview

Recognizing that a universal model can explain every log and produce 100% accuracy, it is clear that an algorithm that focuses exclusively on fidelity in disregard of specificity is not very useful. We propose a quality-based process mining algorithm that generates candidate models using an intelligent heuristic and evaluates them by their quality. The algorithm does not make any strict assumptions regarding the input log as long as it is representative of the process. Given any arbitrary log, our algorithm can always find a block-structured model with the six structures (S, C, L, P, SL, and OP) to explain a pre-specified percentage of the log instances. This results in a *search problem* of finding the model with the highest quality metric that can fully explain the given log. When noise is present in a log, then the accuracy requirement is relaxed suitably, say, to 90% if the estimate of noisy log instances is 10%. An exhaustive algorithm that considers all

possible models is not feasible. Our proposed algorithm does better than exhaustive search by providing heuristics to find accurate and high-quality models faster in the search process.

Our overall strategy is based on a **Best-First** tree search (see the tree in Figure 6 for a log with four tasks, t1 through t4). Starting with an empty root node, we successively determine pairs of tasks to be merged into a block in one of the four basic relationships (S, C, L, and P). Each path from the root node (or an internal node), represents a new pair of tasks or blocks to be merged in S, C, L or P relationship. When the algorithm terminates, an internal node represents a partial model, while a leaf node represents a complete one (unless a path is prematurely aborted).

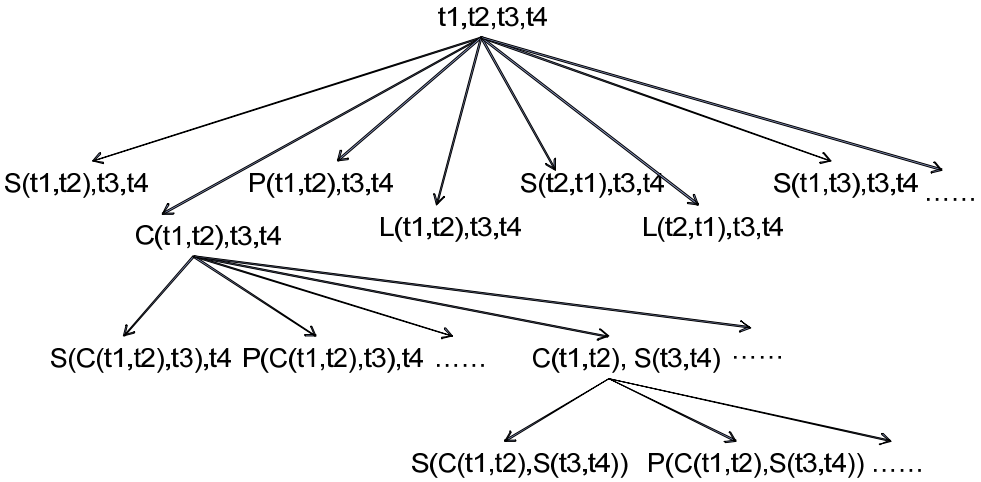


Figure 6: Illustrative example of a search tree for four tasks

The Best-First search, also called a "search and score" approach, uses a criterion to decide what nodes of a partially developed tree are most promising to explore further. For our case, a non-leaf node represents a partial process model. We define a penalty criterion called *mismerge* (MM) score at each node to determine how promising a partial model that the node represents is. A higher MM value represents a bad model. Intuitively, it reflects the inconsistency of the merges along the edges of a selected path from the root to a node with respect to the relationships in the adjacency matrix **AdjM** that is made by simply scanning the log and setting AdjM[i,j] =1 when the task sequence 'i j' appears in the log. The search tree node with the least mismerge score that corresponds to an incomplete model is explored further at every iteration of the algorithm to grow the tree.

When a complete model is found, a procedure called **Replay** "replays" each log instance against the model. For each instance, any SL and OP structures ("adornments") needed in the model to resolve inconsistencies between the model and the log instance are added. Recall from

previous discussion that this is always possible. A procedure **Optimize** is then called to discard log instances with uncommon "adornments", and determine the "best" set (with respect to effect on quality) of SL and OP structures to add to the pure model such that a pre-specified percentage of the log instances is fully explained. The quality metric of this complete model is computed and stored. The best-first search is continued until a pre-specified number of complete models have been reached and the model with the best quality is selected. More details are given next.

4.2 Algorithm Details

The algorithm first builds an *adjacency matrix* called AdjM. This is an $N \times N$ matrix for N tasks in the log (including two artificially added tasks *start* and *end).* The AdjM matrix for the first 9 instances of Table 1 is shown in Figure 7(a). Initially, all entries are set to 0. Then, for every pair of consecutive or adjacent entries (i, j) in the log, AdjM[i][j] is set to 1.

	Start	A	B	C	D	L1	L2	End
Start	0	1	1	1	1	0	0	0
A	0	0	0	1	0	1	0	1
B	0	0	0	1	1	1	0	1
C	0	1	1	0	0	0	1	1
D	0	1	1	0	0	0	1	1
L1	0	0	1	1	1	0	1	0
L2	0	0	1	1	0	1	0	0
End	0	0	0	0	0	0	0	0

(a) AdjM matrix

Possible merges
(Start, A, 'S'), (A, Start, 'S')…
(A, B, 'S'), (B, A, 'S), (A, B, 'P'), (B, A, 'L')…
(B, C, 'S'), (C, B, 'S'), …
…. (C,D,'S'), (D, C,'S'), …(C, D, 'C')…
(A, L1, 'S'), (L1, A, 'S), …(L1, A, 'L')…
(A, L2, 'S'), (L2, A, 'S), …(L2, A, 'L')..
……(L2, End, 'L')

(b) Possible candidate merges in matrix

Figure 7: Initial AdjM matrix and revised AdjM matrix after merging tasks C and D

The AdjM matrix is used to calculate the mismerge scores for a partial model using a heuristic formula. This score is calculated for every potential combination of two tasks and a relationship (i.e. 4N(N-1) combinations for N tasks) based on inconsistencies in the AdjM matrix. Thus, for combination (i, j, R) the score is calculated by applying the following rules:

Merge consistency rules ([i, j] refers to entry AdjM[i, j], and for all s s.t. $s \neq i, j$

(1) if R = 'P', **then** [i, j] = 1 **and** [j, i] = 1 **and** [i, s] = [j, s] **and** [s, i] = [s, j]

(2) if R = 'C' , **then** [i, j] = 0 **and** [j, i] = 0 **and** [i, s] = [j, s] **and** [s, i] = [s, j]

(3) if R = 'S', **then** [i, j] = 1 **and** [j, i] = 0 **and** ([i, s] = 1 **or** [s, j] = 1) ➔ [i, s] = [j, s] = [s, i] = [s, j]= 1

(4) if R = 'L', **then** [i, j] = 1 **and** [j, i] = 1 **and** ([j, s] = 1 **or** [s, j] = 1) ➔ [i, s] = [j, s] = [s, i] = [s, j]=1

These rules ensure that the correct semantic behavior of each merge type is reflected in the log, and is generated from a pure model with S, C, P, and L structures (i.e there are no SL or OP

structures). Thus, if tasks *i* and *j* are to merge in parallel, **Rule 1** requires that they should appear in both orders ([*i*, *j*] = 1 and [*j*, *i*] = 1) in the log. In addition every other task *s* should have exactly the same adjacency relationship with them ([*i*, *s*] = [*j*, *s*] and [*s*, *i*] = [*s*, *j*]). Similarly, **Rule 2** says that if tasks *i* and *j* are to merge in choice, then they should never be adjacent to each other, and again every other task *s* should have exactly the same precedence relationship with both *i* and *j*. **Rule 3** requires that if tasks *i* and *j* are to be merged in sequence then *i* should be adjacent before *j*, but not after *j*. In addition, every other task *s* that is adjacent after *i,* or before *j,* should be in parallel with both *i* and *j*. By **Rule 4**, if tasks *i* and *j* are to be merged in a loop, then *i* should be adjacent before *j,* and also *j* before *i*. Additionally, every other task *s* that is adjacent to *j* (either *before* or *after*) should be in parallel with both *i* and *j*. Strictly, for these rules to work perfectly, the log must be "complete" (i.e. describe all possible paths through the process), but this is unrealistic for non-trivial processes. Nevertheless, the MM score reflects the deviation of a representative log for a potential merge (and, hence, the desirability) with respect to these rules. Appendix A (in the online supplement on the journal website) gives an example of a MM score calculation. The software for our algorithm is available on our website (see Appendix F).

Figure 8 gives a high level pseudo code of the algorithm. For illustrative simplicity, we show the first model reached by the search process that has the desired accuracy, and its quality. In Appendix B, we show the detailed pseudo code for evaluating a specified number of models and the details of sub functions in the algorithm. The **ProposePair** function applies the merge consistency rules to obtain mismerge scores for **all** candidate merges for a partial model. The **Merge** function obtains the *new* AdjM matrix after two tasks/blocks merge by means of a bit-wise logical OR operation on the rows and columns of the merging tasks/blocks.

After a basic model with S, P, C and L constructs is created, the **Replay** function finds a set of SL and OP markings on tasks/blocks needed for each log instance to be fully explained. We maintain a list of currently expected tasks next_tasks in the model, and a list of occurred_tasks, and update them after each log entry *e* is scanned. If *e* is not contained in next_tasks but is contained in occurred_tasks, then a self-loop structure is placed on an appropriate block in the model such that the updated next_tasks contains *e*. Also, if *e* is not contained in either next_tasks or occurred_tasks, then an optional structure is placed on an appropriate block in the model such that the updated next_tasks contains *e*. In this way we ensure that the log is explained by the model by adding SL and OP tags on the basic model. In the example log of Table 1, entry 10 ('A')

is explained by putting an optional path around the C-D block, and entry 11 ('D A L1 B B') by placing a self-loop around task B. In general, this can always be done. The **Replay** function returns a set containing the SL and OP markings for each instance in the log. A simple heuristic chooses (*100 – Desired_Accuracy*)% of log instances that have uncommon SL/OP structures to discard, and takes the union of the rest as the SL/OP structures for the final model, which is returned together with its quality metric. The problem of which X% of log instances to remove such that the union of the SL/OP markings on the rest results in the highest quality metric can be formulated as an optimization problem, which we plan to investigate in future research. Further details on **Replay** and the computation of SL/OP markings are given in Appendix C.

```
Algorithm Extract_Model (Log, Taskset, Global Desired_Accuracy%)
  Build AdjM; set Global candidates = {};
    model0.tuples = {}; model0.AdjM = AdjM; model0.MMscore = 0
    //a tuple is (task/block i, task/block j, relationship R, mismerge (MM) score)
  SearchModel(model0, Taskset, Log)
End Extract _Model
Function SearchModel(model, Taskset, Log, evaluated_models)
 tupleList = ProposePair(model.AdjM)
 remove model from candidates
 Foreach (tuple in tupleList)
   model1.tuples = model.tuples ∪ tuple
   model1.MMscore = model1.MMscore + tuple.MMscore
   model1.AdjM = Merge(AdjM, model2.lastTuple.i, model2.lastTuple.j)
 candidates = candidates ∪ model1
 sort candidates on MMScore
 model2 = candidates.pop(0)
 If (all Tasks are included in model2)
        SLOP_set = Replay(Log, Model)
        Remove (100 – Desired_Accuracy)% elements from SLOP_set \\ simple heuristic is currently
        employed to choose the elements corresponding to worst quality metrics to remove
        SLOP = SLOP[1] ∪ SLOP[2] ∪ ….
        model2.Q = Quality(model2, SLOP)
        Return model2
 Else: SearchModel(model2, Taskset, Log)
End SearchModel
```

Figure 8: High level pseudo code of the **Extract_Model** algorithm (later referred to as HK)

For a search algorithm, the worst-case complexity of our algorithm is high at $O(N^{2N})$ because there are N steps and in each step $4N(N\text{-}1)$ merge combinations are generated. But, in practice, our algorithm is able to find an optimal or near optimal solution by examining a small number of candidate models as we show from the experiments in the next section.

Our algorithm is related to the Genetic Mining approach (van der Aalst, de Medeiros et al. 2007) in that both adopt the "search-and-score" strategy. The **Genetic Miner** relies on the

genetic principles like mutation and cross-over to simulate the evolution of a population of random models to find models with a high score. Our algorithm relies on the mismerge score heuristics to guide the search process, where the effectiveness of the heuristics is the key. In our experiments, the Genetic Miner was significantly more computationally intensive and produced worse models than those extracted using our algorithm when comparable amount of search time was used. More details on this comparison are reported later in Section 6.

5. Experiments with Synthetic Data

In this section we report experiments in which we use our algorithm to generate process models from simulated logs and then evaluate the quality of these models. The simulated logs were generated based on simulated process models. In real-world settings, the correct process model is often unknown. However, a simulated log allows us to compare the true process model with the one produced by our algorithm.

5. 1 Simulated Log Data Generation

We use a simulated process model to generate a log. Starting with a list of tasks, two tasks at random are combined in one of four basic structures (sequence, choice, parallel and loop). We then also randomly mark certain tasks and blocks as SL and OP structures. When generating logs from the simulated model, for a choice structure a branch is chosen with equal probability (0.5); for a loop structure 1-3 times of iterations are executed; for a parallel structure the tasks under the two parallel branches are mixed randomly into a single sequence.

Experiments were also conducted on simulated logs with *artificially induced noise*. We define a general noise operation on a log instance as:

$$row = row[0{:}a] + row\,[c{:}d] + row\,[b{:}c] + row\,[a{:}b] + row[d{:}end]$$

where a, b, c, d: are randomly chosen positions of log entries in the log such that $a \leq b \leq c \leq d$.
As an example, consider a log entry or trace like "t1 t7 t9 t8 t6 t4 t3 t4 t2". If a = 1, b = 2, c=5, d=7, then after performing this operation, the log entry would be "t1 t4 t3 t4 t8 t6 t7 t9 t2" which is an example of noisy log trace. This is a general transformation resulting in a swap of any pair of segments from a log instance. In our experiments, we randomly chose 5% of the instances or rows in the simulated logs and performed the noise operation up to three times on them. Correspondingly, we set the target accuracy to be 95% when running the process mining

algorithm on these noisy logs. A target accuracy of 100% was used with the noise-free logs. We describe random process models and give examples of synthetic log instances produced from them in Appendix D.

5.2 Experiments on 10 Process Models

In this set of experiments, we constructed 10 random process models, each containing 10 tasks with a varying number of SL and OP structures. 500 log instances were used as input in our experiments and the stopping rule was set to 100 evaluated models. This rule was based on a reasonable trade-off between running time and the quality of the model obtained. Of course, for certain difficult process models, this rule may be modified. It should also be noted that in our experiments weights W_1 =1 and W_2 = 5 were used, but in practice users can adjust the weights based on the relative importance of SL and OP structures.

We used all three versions of the quality metrics (Q0, Q1, and Q2) introduced in Section 4. The experimental results are presented in Tables 2(a), 2(b), and 2(c). In each table, and for each of the 10 random process models, we show the corresponding Q-metric of the true model (together with number of SL and OP tasks/blocks assigned to the models) and the same information for the best models produced by the algorithm. The Q0, Q1, and Q2 metric values lie in the range of 0.6842 - 1, 0.2612 - 0.9602, and 0.5912 - 0.9678, respectively. Note that the 10 process models came from five groups (SL, OP) structure counts of as: (0,0), (0,1), (1,0), (1,1), and (2,2). The numbers of SL and OP structures reported in Table 2 are often larger than the number of structures randomly assigned. This arises from the cascading equivalent model rewriting procedure mentioned in Section 3.1, which propagates these structures to other levels of the process tree. For the noise-free logs we report the best Q found, the number of models evaluated before finding the best one, and the accuracy achieved. Note that in some cases the actual accuracy is higher than the target accuracy.

In general, we observed similar performance across the three quality metrics. In our approach the search process for finding pure models is based on the *mismerge scores* discussed in Section 3, and is not influenced by the quality metrics. Therefore, the same 100 models were evaluated for all three metrics, and the specific quality metric was used to determine which of these 100 models was chosen to be the best model. Furthermore, because Q1 and Q2 are both monotonic transformations of the badness scores, the models chosen by the two metrics were exactly the

same ones as in Table 2, but the quality metric values were different. The similar performances in Table 2 are therefore reasonable.

From the experimental results on the noise-free data, we observe that using Q1 and Q2 metrics our algorithm recovered the exact true model for Models 1, 2, 3, 4, and 5. When using Q0 our algorithm recovered the exact true model for Models 1, 2, 3, and 5. For Model 4, although Q0 identified the model with the same quality metric (same number of SL and OP structures) as the true model, closer examination reveals that a sequence structure in the true model becomes a parallel structure in the learned model. This is due to the simplicity of the Q0 metric as discussed earlier. For Model 6, 7, and 10, the learned models were very close to the true model as explained next. In particular, for Model 6 the learned model actually had better Q1 and Q2 values than the true model. Further examination reveals that the algorithm found an equivalent model as the true model with a better quality metric. Specifically, a block consisting of three tasks in the true model <{(a, b, 'C', B1), (c, B1, 'P', B2)}, (1,1,0,1,0), (0,0,0,0,0)> becomes <{(c, b, 'P', B1), (a, B1, 'P', B2)}, (1,1,0,0,0), (1,0,1,0,0)>. The two tasks with SL and OP structures resulted in better Q1 and Q2 metrics than the one self-loop block containing two SL tasks. Our algorithm had difficulty with Models 8 and 9, and it discovered models with more SL and OP structures than were present in the true models. Further investigation showed that both models have self-loop structures on the root block, thus creating a very large number of traces and making them particularly difficult to learn accurately from the logs. Such special cases are not very likely to occur in the real world.

From the results for noisy logs, it was heartening to observe that the *same* models were learned for Models 2, 3, 4, 5, and 6 as those learned using the noise-free data. For Model 1, the true model was not achieved within the 100 models evaluated. We found that the true model could however be found by evaluating still more candidate models. For Models 7, 8, 9, and 10, the models learned with noisy data had better quality metrics than the ones learned from noise-free data. Furthermore, for Models 7, 8, and 10 the learned models actually had better quality metrics than the true model. The better quality metrics of learned models were the result of the overall optimization process taking into consideration the quality calculation, equivalent model structures, and log instances selected to discard. Overall both sets of results are very encouraging because they show that the quality based algorithm is able to discover original models of varying degrees of quality from the logs of these models with surprising accuracy.

Table 2: Experimental Results with 10 simulated logs

(a) Metric Q0

Model	True model Q* (# SL, # OP)	Noise-free data with 100% target accuracy		Noisy (5%) data with 95% target accuracy		
		Best Q (# SL, # OP)	Best model found at #	Best Q (# SL, # OP)	# model at	Accuracy achieved
1	1 (0,0)	1 (0,0)	4	0.9736 (1,0)	14	0.982
2	1 (0,0)	1 (0,0)	1	1 (0,0)	11	0.96
3	0.9736 (0,1)	0.9736 (0,1)	1	0.9736 (0,1)	1	0.966
4	0.9210 (0,3)	0.9210 (0,3)	1	0.8947 (0,4)	16	0.97
5	0.9736 (1,0)	0.9736 (1,0)	1	0.9736 (1,0)	1	0.966
6	0.9210 (3,0)	0.8947 (2,2)	1	0.8947 (2,2)	2	0.982
7	0.7105 (3,8)	0.6842 (4,8)	1	0.7631 (4,5)	58	0.966
8	0.6842 (6,6)	0.7105 (10,1)	42	0.3947 (15,8)	8	0.954
9	0.8421 (4,2)	0.3684 (16,8)	22	0.5263 (8,10)	46	0.952
10	0.7894 (4,4)	0.7894 (4,4)	78	0.8947 (3,1)	41	0.99

(b) Metric Q1

Model	True model Q* (# SL, # OP)	Noise-free data with 100% target accuracy		Noisy (5%) data with 95% target accuracy		
		Best Q (# SL, # OP)	Best model found at #	Best Q (# SL, # OP)	# model at	Accuracy achieved
1	0.7966 (0,0)	0.7966 (0,0)	4	0.6820 (1,0)	16	0.982
2	0.9602 (0,0)	0.9602 (0,0)	1	0.9602 (0,0)	11	0.96
3	0.8846 (0,1)	0.8846 (0,1)	1	0.8846 (0,1)	1	0.966
4	0.8127 (0,3)	0.8127 (0,3)	15	0.7343 (0,3)	79	0.968
5	0.8044 (1,0)	0.8044 (1,0)	1	0.8044 (1,0)	48	0.966
6	0.4839 (3,0)	0.6450 (2,2)	1	0.6318 (2,2)	5	0.982
7	0.3391 (3,8)	0.3312 (4,8)	3	0.3398 (4,5)	54	0.966
8	0.2612 (6,6)	0.2412 (10,6)	41	0.2852 (12,15)	96	0.968
9	0.3501 (4,2)	0.2103 (16,8)	22	0.2654 (8,11)	56	0.952
10	0.4341 (4,4)	0.4285 (4,4)	78	0.6048 (2,3)	12	0.95

(c) Metric Q2

Model	True model Q* (# SL, # OP)	Noise-free data with 100% target accuracy		Noisy (5%) data with 95% target accuracy		
		Best Q (# SL, # OP)	Best model found at #	Best Q (# SL, # OP)	# model at	Accuracy achieved
1	0.8980 (0,0)	0.8980 (0,0)	4	0.8604 (1,0)	16	0.982
2	0.9678 (0,0)	0.9678 (0,0)	1	0.9678 (0,0)	11	0.96
3	0.9302 (0,1)	0.9302 (0,1)	1	0.9302 (0,1)	1	0.966
4	0.9035 (0,3)	0.9035 (0,3)	15	0.8775 (0,3)	79	0.968
5	0.9007 (1,0)	0.9007 (1,0)	1	0.9007 (1,0)	48	0.966
6	0.7830 (3,0)	0.8479 (2,2)	1	0.8434 (2,2)	5	0.982
7	0.6855 (3,8)	0.6779 (4,8)	3	0.6862 (4,5)	54	0.966
8	0.5912 (6,6)	0.5573 (10,6)	41	0.6256 (12,15)	96	0.968
9	0.6956 (4,2)	0.4922 (16,8)	22	0.5977 (8,11)	56	0.952
10	0.7561 (4,4)	0.7527 (4,4)	78	0.8337 (2,3)	12	0.95

5.3 Experiment on 30 Process Models with various SL/OP markings

In order to verify the robustness of the findings reported in Section 5.2 from 10 random process models, we report a larger experiment on 30 randomly generated models in this section. We first randomly generated 30 pure models of 10 tasks. Based on these 30 models, we variously imposed the (SL, OP) structures with respective counts of (0, 0), (0, 1), (1, 0), (1, 1), and (2, 2) to generate five groups of models similar to the ones in the experiments of Section 5.2. The base pure models were maintained to be the same. Altogether 150 different process models were used in this experiment. The detailed results are omitted for brevity, but the findings are summarized next. A key criterion for evaluation was the *quality ratio* = $\frac{\text{Q2 of the learned model}}{\text{Q2 of the true model}}$.

With the noise-free data, the *average quality ratio* varied continuously between 0.986 for group 1 to 0.970 for group 5. Closer examination revealed that of the 30 pure models in group 1, 27 models were successfully learned by the algorithm. The remaining three learned models had SL and OP structures, and one outlier was mainly responsible for quality ratio of less than 1.

For models learned from the noisy data, the quality ratio varied between 0.942 for group 1 and 1.075 for group 5. The quality ratio here varies in a contrary way from its behavior in the noise-free situation. When working with pure models (with no OP or SL structures), the presence of noise consistently decreases the Q-value in the learned model. But as the number of SL and OP structures increased in the true models, the algorithm actually had greater room to optimize and learned models with better quality metrics. These results further validate the algorithm.

6. Experience with Real Data – A Patent Application Process

6.1 Data and Results

In order to evaluate the practical applicability of the algorithm, we accessed the United States patent application process data (available at http://portal.uspto.gov/external/portal/pair) and discuss our experiments with this data next.

We collected transaction data for patents of the category 435 (Chemistry: molecular biology and microbiology) under the United States Patent Classification issued between 2000 and 2005. There are a total of 31,682 patent transaction instances containing 518 unique tasks. As an initial attempt towards building a process model for all 518 tasks, we focus our experiments on the top-

10 most frequent tasks. These tasks help to identify the highest-level structure of the underlying process model with manageable complexity. As additional less frequent tasks are included into the analysis, the resulting model will provide more details of the process.

Table 3: Experimental results: patent log instances with varying target accuracies

Exp. #	Target accuracy	Q2	Actual Accuracy	Best model at #	# SL	# OP
1	1	0.4974	1	6	16	16
2	0.95	0.6528	0.9755	44	9	10
3	0.9	0.6528	0.9755	44	9	10
4	0.85	0.6529	0.8630	44	9	9
5	0.8	0.6577	0.8436	17	7	5
6	0.75	0.6577	0.8436	17	7	5
7	0.7	0.6577	0.8436	17	7	5
8	0.65	0.6558	0.6863	10	6	4
9	0.6	0.6558	0.6863	10	6	4
10	0.55	0.6560	0.5586	10	5	5
11	0.5	0.6591	0.5449	17	5	5

For a real world log like this one, it is difficult to estimate how many instances are noisy in order to set a target accuracy level for the algorithm. Thus if it is set too high, the algorithm produces a low-quality model, while, if it is set too low, the algorithm over-simplifies the model. Hence, the algorithm was run with varying target accuracies, using *Q2* as the quality metric. Table 3 shows *Q2* and *accuracy* of the best model found within the first 100 searched models (and the sequence number at which the best model was found) for different target accuracies. When the target accuracy was set to 1, the quality of the best model was 0.4947 and it was the 6th model evaluated. As target accuracy decreased, the quality increased as expected. We observed that the largest increase occurred as the target accuracy changed from 1 to 0.95. Afterwards, the quality increased only slightly as the target accuracy was reduced in steps to 0.5, while the number of (SL, OP) structures fell from (9, 10) to (5, 5). The fact that five OP and five SL structures remain even after discarding 50% of the log instances also suggests that the actual patent application process for these top 10 tasks is a low quality model. We provide the detailed description of the tasks, along with figures and discussion of the mined models in Appendix E.

Several insights were gained from this experiment with a real world log. Notably, the models help to make sense of the underlying process even in a situation where the process is ill-defined and the log is noisy. By comparing the various models at different levels of accuracy a domain expert can determine the model that reflects the best combination of accuracy and quality in the actual real-world process. We also observed that some fundamental aspects of the process were

retained in models of different accuracy level. A domain expert can even combine pieces of the models and rearrange them in order to obtain a more accurate model.

6.2 Comparison with Existing Algorithms on Existing Quality Measures

To further demonstrate the effectiveness of our proposed framework of quality metric and model search algorithm, we compared our algorithm with comparable existing algorithms (van Dongen, et al. 2009). In the introduction, we briefly discussed our experience with several algorithms including Alpha++, Heuristic Miner, Multi-Phase Miner, Parikh language-based region and Genetic Miner on the simple log in Table 1. Here we give results of comparison of different algorithms on our patent application data. The algorithms are described briefly first.

6.2.1 Background on existing algorithms

The **Heuristic Miner** or **HM** (Weijters and van der Aalst 2003) algorithm computes a number of metrics from the log for all task pairs such as frequency of each task, frequency of direct succession (i.e. immediate precedence of two tasks), and frequency of indirect succession, and uses these metrics to compute dependency scores. The algorithm uses heuristic rules based on the dependency scores and frequency of direct succession to detect direct causal relationships between tasks, and form a dependency graph. Another set of heuristic rules again based on the dependency scores are applied to differentiate parallel and choice structures to derive a workflow net (a special case of a Petri net). The heuristic miner represents the typical rule-based algorithm, which outputs a *single* model that is deemed to be most probable according to the observed log. Such an approach relies heavily on how closely the observed execution patterns align with the heuristic rules used. No possibility of adjustment of the heuristics and revision of the resulting model is built into the algorithm. For example, there is no way of knowing that by just changing a choice structure to a parallel structure the model can be improved greatly and therefore the change should be made to revise the model.

The **Alpha ++** and its other variants are of a similar nature to HM, but they make different assumptions about the completeness of the log. As discussed in Section 4.2, our algorithm is somewhat similar to **Genetic Miner** (van der Aalst, de Medeiros et al. 2007) in that both adopt the "search-and-score" approach, where multiple candidate models are generated and evaluated. Although the genetics principle provides the mechanisms to guide the search process toward better models over time, from a modeling perspective no heuristics can be incorporated to aid the

search. Our approach can be viewed as a hybrid in that our Best-first search is guided by the mismerge score heuristics. This leads to a more directed and faster search process. Meanwhile, because we do search for multiple candidate models, the algorithm is robust in the sense that we should be able to find the best model even if the actual execution patterns deviate from the heuristics used, as long as the deviation is small and a sufficient number of models are evaluated.

We also included in our experiment the **Multi-Phase miner** (van Dongen and van der Aalst 2005), which first derives models for individual instances and then aggregates the instance models to obtain the complete model for the process; and the **Parikh language-based region miner** (van der Werf, van Dongen et al. 2008) which exploits the similarity between process model and language models and builds on the theory of region that tries to synthesize a Petri net which can reproduce the language as precisely as possible. Both algorithms claim that a Petri net that is consistent with the log can always be found, which is also true for our algorithm due to the possibility of universal models.

6.2.2 Results

To provide a common platform to compare different algorithms, we converted resulting models of different algorithms into Petri nets so that many existing quality and fitness metrics can be used to evaluate different models. We used $PF_{complete}$ (van der Aalst, de Medeiros et al. 2007), which is calculated based on missing and extra tokens found while replaying the log against the Petri-net model, as the fidelity measure. For model specificity, the *behavioral* and *structural precision,* and *recall*, measures (van der Aalst, de Medeiros et al. 2007) are quite relevant to our quality metric. One issue is that these measures require a reference model. In our experiments, we used a universal model as the reference model, but doing so makes the *precision measures always 1,* and thus not helpful. The recall measures now reflect how much behavior is allowed by the mined model, and therefore, actually correspond nicely with our quality metric in an inverse way, i.e., a small recall measure means that less behavior is allowed, and corresponds to higher specificity, or in effect higher quality. Note that behavioral recall is different from structural recall in that the former captures the amount of behavior allowed in the context of the execution of every log instance, while the latter captures the behavior allowed by the model in general.

We used the ProM framework for our experiments. The reported measures are for the best model obtained across different settings. For the HM algorithm, different parameter settings

were tried and measures for the best model obtained are reported. For the Genetic Miner (GM), we tried different settings of population size and generations with which the program could complete within 20 hours, e.g. 10 individuals in a population for 100 generations, and 100 individuals for 10 generations. The model measures were obtained using the Control Flow Benchmark plug-in. In our experiments with the complete patent dataset, the Parikh miner resulted in an infeasible model and the Multi-Phase miner resulted in a Petri net with an excessive number of hidden tasks and in invalid quality metrics. We understand that the output of the Multi-Phase miner is represented as Event-driven Processing Chains (Curran and Ladd, 2000). The translation to Petri net may be problematic. Since it is not straightforward to compare EPCs with Petri nets, we leave the direct comparison between our method and multi-phase miner to future work.

Table 4. Comparison of our algorithm (HK) with Benchmark Algorithms

	Fidelity/fitness metrics		Specificity metrics		
	Ref. metric	Our metric	Reference metrics		Our metric
Algorithm	$PF_{complete}$	Accuracy	Behavioral Recall on Universal Model	Structural Recall on Universal Model	Quality
Heuristic Miner	0.0956	-	0.3526	0.1240	-
Alpha++	0	-	0.5151	0.4215	-
Genetic Miner	0	-	0.1105	0.0496	-
HK-100%	1	1	0.7831	0.7603	0.4974
HK-95%	0.8387	0.9755	0.5176	0.0992	0.6528
HK-65%	0.8203	0.6863	0.4796	0.0744	0.6558

In Table 4, we report $PF_{complete}$, *behavior recall on universal model,* and *structural recall on universal model* measures for models produced by HM, Alpha ++, GM, and our algorithm (HK) at the 100%, 95%, and 65% desired accuracies (i.e. HK-100%, HK-95%, and HK-65%). For our models, we also report the *accuracy* and *quality* measures. $PF_{complete}$, and our accuracy metrics are measures of fidelity/fitness of the model, and high metric values indicate high fidelity. Our quality metric and the behavioral and structural recall on universal model metrics are measures of specificity; thus, high quality and low recall values indicate high specificity. We observe that while existing algorithms may be able to output specific models, they have difficulty in finding process models with fidelity for the kind of log we experimented with. The HM model is the best among the three in fidelity, but it can explain just about 10% of the task executions in the log. In contrast, the models produced by our algorithm are of much higher fidelity. Our HK-100%

model had the worst specificity, which is to be expected as the model is quite close to a universal model. The HK-95% and HK-65% models had a value greater than 82% on the $PF_{complete}$ measure. Note that the larger difference in the accuracy measure arises in part due to the measurement unit being a task versus a log instance. The recall measure is comparable or better relative to the HM model. In fact, on $PF_{complete}$ and structural recall, both HK-65% and HK-95% dominate the HM and Alpha++ models for this log.

Of course, this analysis applies only to a single log and is not generalizable without further systematic experimentation. Yet it shows how different approaches can be compared. It also highlights the potential and need for new approaches in accommodating the process patterns found in real-world noisy logs, and striking a better balance between model accuracy and quality.

7. Discussion and Related Work

In this paper we developed a quality-based algorithm for analyzing process logs and extracting block-structured process models from the task precedence relationships observed in the logs. We focused on designing an algorithm that can help us make sense out of noisy logs because real world logs are noisy and an assumption of a 100% correct log is unrealistic. We noted that the noise in the log of a process that is designed mainly from basic structures like sequence, parallel, choice and loop can be explained by adding self-loop (SL) and optional (OP) structures on an initial candidate model. In fact, these latter two structures can be used to explain every log given an arbitrary process model consisting of the four basic structures.

However, adding these structures hurts the quality of a model. Using this reasoning we developed quality metrics (Q-metrics) for measuring the quality of a model and developed an algorithm for extracting the highest quality model from a log. The problem of extracting process models from a noisy log is then formulated as one of finding the model with the best quality metric that can explain a desired proportion of log instances. Our algorithm performs Best-First search using a *mimserge score* metric to determine the best task combination and relationship to merge at each successive step to build a model. The experimental results showed that when the true models were of high quality (containing less SL and OP structures), say above 0.9, the algorithm was able to discover them from their logs after examining only about 100 candidate models. Also, good approximations were found of lower quality true models from their logs.

There is an inverse relationship between noise and quality. When more noise is present in the log, the model required to explain it uses more SL and OP structures, thus lowering the model quality. In a practical situation, a user can run our algorithm at different levels of target accuracy and produce several process models that are X% faithful to the log for a given accuracy level X. The results of the experiments with both simulated and real data were surprisingly encouraging and provided valuable insights. However, the Q-metric, as any other metric, is not perfect. More sensitivity analysis to determine the effect of the weighting coefficients in it would be useful. We also need to develop better optimization heuristics for the **Replay** function for removing log instances when less than 100% accuracy is desired.

In recent years, there has been considerable interest in process mining algorithms to extract workflow models from logs. There are techniques for extracting three kinds of models: graph-oriented models, Petri-nets, and block structured models. The approach of (Agrawal, Gunopulos et al. 1998) extracts dependency graphs from task logs based on a dependency analysis between tasks. It was applied in the context of the FlowMark workflow system from IBM. The alpha algorithm and its variants for extracting Petri net models were proposed in (van der Aalst, van Dongen et al. 2003; van der Aalst, Weijters et al. 2004). These algorithms are based on extracting causal dependencies from logs, and connecting tasks (or transitions) that depend on one another by inserting output and input places between them. The approach is implemented in the ProM tools (van Dongen, de Medeiros et al. 2005). The approach of (Herbst and Karagiannis 1998; Herbst and Karagiannis 1999) starts with a very general initial model and applies a series of merges and splits using probabilistic methods to induce a final model.

Techniques for block structured workflows are proposed in (Schimm 2000; Schimm 2003). This approach defines an algebra for workflow traces, then it groups traces into classes, extracts precedence relationships, creates sub-models and finally synthesizes them using the algebra. It is also a bottom-up approach like ours, which starts at the instance level, and by applying rewriting rules it clusters the instances more and more until no further rewriting is possible and a block-structured model is returned. It handles loops by relabeling which is not required in our approach. It also assumes that tasks have a start and end time while we assume that they are atomic. Another technique based on clustering of traces to identify global constraints is discussed in (Greco, A.Guzzo et al. 2004). A probabilistic model and a learning algorithm for workflow mining are discussed in (Silva, Zhang et al. 2005).

Process mining in the context of noisy data has received less attention. We described several existing algorithms (Heuristic Miner, Multi-phase Miner, and Genetic Miner) that can deal with noisy data in Section 6.2. A different approach to mining models from noisy (unstructured) real-world logs is the fuzzy mining approach (Günther and van der Aalst 2007), which is an adaptive technique based on significance and correlation metrics defined on tasks and task pairs to allow filtering of edges in a task precedence graph. It results in summary models with different degrees of simplicity. One significant difference in this approach is that it does not attempt to identify specific process model structures such as choice and parallel.

Another line of work (see for instance, (Măruşter, Weijters et al. 2006)) points out that when noise is present, frequencies should influence the decision of a process mining algorithm regarding what instances to discard. Therefore, they develop heuristic algorithms (e.g., (Weijters and van der Aalst 2003)) based on frequency tables that maintain task-task precedence frequencies. While we do not keep frequencies explicitly in our current design, yet, they do influence our resulting model implicitly because noisy instances are likely to generate more SL and OP structures, and these will successively be discarded while optimizing the model for noise at various levels of accuracy. Thus, in the quest for higher quality we are able to eliminate infrequent (and quite likely, noisy) process patterns. Another approach called process mining based on clustering (de Medeiros 2007; Song, Günther et al. 2008) creates related clusters of log instances first, and then applies processing mining techniques to each cluster separately. This results in several alternative models from which a user may choose one. Finally, the method of (Greco, Guzzo et al. 2006) extracts multiple unstructured And-Or graph models without loops, but users prefer a single model.

8. Conclusions and Future Work

Process mining is an emerging research area of considerable significance within the broader domain of knowledge discovery. The new quality based process mining algorithm described in this paper uses the approach of creating an adjacency matrix for all pairs of tasks to capture the relationships embodied in the log. Then it assigns *mismerge scores* to candidate pairs of task, and selects the best pair to merge. This procedure is repeated within a backtracking algorithm to derive multiple candidate workflow models. The one with the highest quality is selected. A novel

aspect of our work is the notion of a *Quality metric* for a process model and the emphasis on the quality-accuracy trade-off in the context of noisy logs.

Through extensive experiments we have shown that our approach based on a relatively simple notion of quality lends itself very well for making sense out of noisy data. A model for generating noisy data was created, and experiments were run on both simulated and real data to test our technique. The results were surprisingly encouraging in both scenarios. With simulated data where the true model was known we were able to extract the model with considerable accuracy. With the real data from a patent approval process we generated three good models at different degrees of accuracy. We gained valuable insights into the issue of quality-accuracy trade-off that is very important when noise is present in the process logs.

In future work, we will work on improving the mismerge scoring heuristics and our Q-metric, and assessing their effects on the performance of our algorithm. We will also work on making our algorithm faster and more scalable. In addition, a later extension would be to find ways to extract models for more complex patterns. Under a different direction, further investigation is needed to compare Petri net, block-structured model, and other representations of business process with respect to the ease of understanding and use of these models by end-users.

Acknowledgments

The authors are grateful to the associate editor and the reviewers for their valuable comments and insights that helped to improve the paper. The research of Akhil Kumar was funded in part by the Smeal College of Business.

References

Agrawal, R., D. Gunopulos, et al. (1998). Mining process models from workflow logs. 6th International Conference on Extending Database Technology.

Chesani, F., P. Mello, et al. (2008). Checking compliance of execution traces to business rules. 4th Workshop on Business Process Intelligence (BPI 08) in conjunction with BPM 2008, Milan, Italy, Springer.

Curran T. and Ladd A. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model* (2nd ed.). Prentice Hall PTR, Upper Saddle River, 2000.

de Medeiros, A. K. A. (2007). Process mining based on clustering: A quest for precision. BPM workshops in conjunction with BPM, Springer Berlin.

Dustdar, S., T. Hoffmann, et al. (2005). "Mining of ad-hoc business processes with TeamLog." Data and Knowledge Engineering **55**(2): 129-158.

Greco, G., A.Guzzo, et al. (2004). Mining expressive process models by clustering workflow traces. 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining.

Greco, G., A. Guzzo, et al. (2006). "Discovering Expressive Process Models by Clustering Log Traces." IEEE Transactions on Knowledge and Data Engineering **18**(8): 1010-1027.

Günther, C. W. and W. M. P. van der Aalst (2007). Fuzzy mining: Adaptive process simplification based on multi-perspective metrics. 5th International BPM 2007 Conference, Brisbane, Australia, Springer Berlin.

Herbst, J. and D. Karagiannis (1998). Integrating machine learning and workflow management to support acquisition and adaptation of workflow models. 9th International Workshop on Database and Expert Systems Applications.

Herbst, J. and D. Karagiannis (1999). An inductive approach to the acquisition and adaptation of workflow models. Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business.

Klein, M. and A. Bernstein (2004). "Towards high-precision service retrieval " IEEE Internet Computing **8**(1): 30-36.

Huang, Z. and A. Kumar. 2008. New quality metrics for evaluating process models. *Proceedings BPM 2008 International Workshops*, Milan, Italy, Sept. 2008, 164-170, Springer.

Lindland, O. I., G. Sindre, et al. (1994). "Understanding quality in conceptual modeling." IEEE Software **11**(2): 42-49.

Mans, R. S., M. H. Schonenberg, et al. (2008). Process mining in healthcare - A case study. HEALTHINF 2008, Funchal, Madeira, Portugal.

Măruşter, L., A. J. M. M. Weijters, et al. (2006). "A rule-based approach for process discovery: Dealing with noise and imbalance in process logs." Data Mining and Knowledge Discovery **13**(1): 67-87.

Mendling, J., H. Reijers, et al. (2007). What makes process models understandable? 5th International Conference on Business Process Management Brisbane, Australia, Springer.

Rozinat, A., A. K. A. de Medeiros, et al. (2007). The need for a process mining evaluation framework in research and practice. 3rd Workshop on Business Process Intelligence.

Rozinat, A., A. K. A. de Medeiros, et al. (2007). Towards an evaluation framework for process mining algorithms, BPM Center.

Rozinat, A. and W. M. P. van der Aalst (2008). "Conformance checking of processes based on monitoring real behavior." Information Systems **33**(1): 64-95.

Schimm, G. (2000). Generic linear business process modeling. Workshops on Conceptual Modeling Approaches for E-Business and The World Wide Web and Conceptual Modeling: Conceptual Modeling for E-Business and the Web.

Schimm, G. (2003). Mining most specific workflow models from event-based data. International Conference on Business Process Management, Eindhoven, The Netherlands.

Silva, R., J. Zhang, et al. (2005). Probabilistic workflow mining. International Conference on Knowledge Discovery and Data Mining, Chicago, IL.

Song, M., C. W. Günther, et al. (2008). Trace clustering in process mining. 4th Workshop on Business Process Intelligence (BPI 08) in conjunction with BPM 2008, Milan, Italy.

Song, M. and W. M. P. van der Aalst (2007). Supporting process mining by showing events at a glance. Seventeenth Annual Workshop on Information Technologies and Systems Montreal, Canada.

Spearman, C. (1904). "The proof and measurement of association between two things." The American Journal of Psychology **15**(1): 72-101.

Tan, P.-N., M. Steinbach, et al. (2006). Introduction to data mining, Addison Wesley.

van der Aalst, W. M. P., A. K. A. de Medeiros, et al. (2007). "Genetic process mining: An experimental evaluation." Data Mining and Knowledge Discovery **14**(2): 245-304.

van der Aalst, W. M. P., H. A. Reijers, et al. (2005). "Discovering social networks from event logs." Computer Supported Cooperative Work **14**(6): 549-593.

van der Aalst, W. M. P., H. A. Reijers, et al. (2007). "Business process mining: An industrial application." Information Systems **32**(5): 713-732.

van der Aalst, W. M. P., A. H. M. ter Hofstede, et al. (2004). "Workflow patterns." Distributed and Parallel Databases **14**(1): 5-51.

van der Aalst, W. M. P., B. F. van Dongen, et al. (2007). ProM 4.0: Comprehensive support for real process analysis. 28th International Conference on Applications and Theory of Petri Nets 2007, Siedcle, Poland, Springer-Verlag.

van der Aalst, W. M. P., B. F. van Dongen, et al. (2003). "Workflow mining: A survey of issues and approaches." Data and Knowledge Engineering **47**(2): 237-267.

van der Aalst, W. M. P., A. J. M. M. Weijters, et al. (2004). "Workflow mining: Discovering process models from event logs." IEEE Transactions on Knowledge and Data Engineering **16**(9): 1128-1142.

van der Werf, J. M. E. M., B. F. van Dongen, et al. (2008). Process discovery using integer linear programming 29th International Conferenc on Applications and Theory of Petri Nets, Xi'an, China, Springer Berlin.

van Dongen, B. F., A. K. A. de Medeiros, et al. (2005). The ProM framework: A new era in process mining tool support 26th International Conference on Applications and Theory of Petri Nets

van Dongen, B. F. and W. M. P. van der Aalst (2004). Multi-phase process mining: Building instance graphs International Conference on Conceptual Modeling, Shanghai, China.

van Dongen, B. F. and W. M. P. van der Aalst (2005). Multi-phase process mining: Aggregating instance graphs into EPCs and Petri nets. the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management (PNCWB).

Weijters, A. J. M. M. and W. M. P. van der Aalst (2003). "Rediscovering workflow models from event-based data using Little Thumb." Integrated Computer-Aided Engineering **10**(2): 151-162.

Wen, L., J. Wang, et al. (2006). Detecting implicit dependencies between tasks from event logs. Asia-Pacific Web Conference on Frontiers of WWW Research and Development, Springer.

Yang, W.-S. and S.-Y. Hwang (2006). "A process-mining framework for the detection of healthcare fraud and abuse." Expert Systems with Applications **31**: 56-68.