

# 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

The goal of process mining is to extract semantic knowledge from a log consisting of process execution traces for the purposes of process understanding, innovation and improvement. In recent years many algorithms have been proposed to extract process models from 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 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. Our first finding is that **all** process logs can be explained by using *self-loop* and *optional* structures. Therefore, it is not difficult to build a fully accurate process model for *any* given log, even logs that contain inaccurate data or noise. Secondly, there is usually not one single, unique process model that can explain a log, i.e. the same log can be explained by a large number of *different* models. Thirdly, some models are of higher "quality" than others, and given that so many models can explain the same log, it is important to have a metric of quality for a model. Fourth, if a log contains noisy execution traces, a fully accurate process model that explains every trace in the log is not very meaningful because its quality is low. By controlling the use of self-loop and optional structures around tasks and blocks of tasks we can balance the quality and accuracy tradeoff to derive high-quality process models that explains a given percentage of traces in the log. Finally, we describe a novel quality-based algorithm for model extraction in the context of our noisy logs. 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 (Aalst, et al. 2007a, Aalst, et al. 2007b, Aalst, et al. 2003, Dustdar, et al. 2005, Weijters and Aalst 2003). 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 which shows several execution instances of this process. In each row is one instance showing the actual sequence in which the tasks were done. Thus, the first instance shows the execution of task A and then C. Other instances in this table show various other execution paths taken through the process from start to finish. 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 instances 10-12 in Table 1. These two new constructs play an important role in understanding noise in process logs.

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

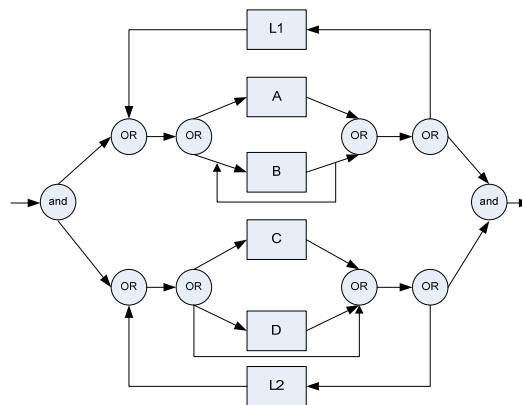


Figure 1: A business process model

Thus, a *process log* consists of execution traces or instances of a process (each row in Table 1 is a *trace* or an *instance*), and given such a log, the problem is to extract the process model. The log may consist of start and end times of each task; however, for simplicity we assume here that

the tasks are instantaneous and just include the start of a task in a log. Thus, each time a task (or activity) appears in the log, it is treated as the start and end of that task. 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 (i.e., fully accurate) models that reflect the actual workflow. Consequently, research efforts have focused on extracting general graphs (Agrawal, et al. 1998) and more specialized graphs such as Petri-nets (Aalst, et al. 2004) from logs.

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 a log of actual instances may differ drastically from the idealized process that might be 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. Thus, useful process knowledge acquired in this way can aid in better understanding of real-world processes, ensure process compliance and lead to better process design. 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, 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 (Mans, et al. 2008, Song and Aalst 2007). 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, et al. 2008). Process mining also has applications in social mining (Aalst, et al. 2005, Aalst, et al. 2007c). Since process logs usually capture information about who performed a task in a process instance, it is possible to extract information about how often a certain worker collaborates with another worker on a process instance, and determine how well two individuals in an organization work with each other. It is also useful to analyze cases or instances within a log, and identify patterns or relationships between the properties of a case. A detailed case study that analyzes an industrial application from the process, organizational and case perspectives is presented in (Aalst, et al. 2007c).

In this paper we introduce a new notion of *process model quality* and use it to evaluate various process models extracted from noisy data. A process log is *noisy* if it contains spurious trace instances that result from data collection errors such as a missed entry, skipped task or an exception entry that does not normally occur. Of course, we assume that the algorithm has no way of knowing which specific entries are noisy, however we allow a user to state a certain accuracy level, as the percentage of log instances that the user expects to be noise-free. Using this information, we then extract the highest quality process model that meets the accuracy requirements specified by the user. The user can also specify a range over which accuracy may vary, and in this case our algorithm can generate several models over this range and offer the user a quality-accuracy trade-off from which the user may select a model.

In our study, we focus on extracting block-structured process models (Schimm 2000, Schimm 2003). 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, block-structured models consisting of the sequence, loop, choice, and parallel structures 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 transaction log* (more on universal models shortly).

Of course, such models are not very useful since they can explain all logs. Therefore the quality of the final model should be based both on how well it explains the given log and how specific the model is to the log. Thus the universal models should have the lowest possible quality value. 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 X% of the instances in the log*, assuming it has X% correct instances.

Apart from the need to address the quality issue in process model mining, as further motivation of our approach, we note that on giving the log of Table 1 to several algorithms in the

well-known ProM toolset (Aalst, et al. 2007a), none of them were able to discover the process of Figure 1. The closest algorithm was Alpha and it produced the Petri-net process (where boxes are transitions and circles are places) shown in Figure 2 for the first 9 instances of the log. This model fails to explain instance 9 of the log and is also difficult to read. For all 12 instances it produced a model with many errors and failed to include task B. Therefore, there is still a need for new approaches for process mining. Unfortunately, there are no standard benchmarks for comparing one algorithm directly against another and there are also many different ways of describing processes.

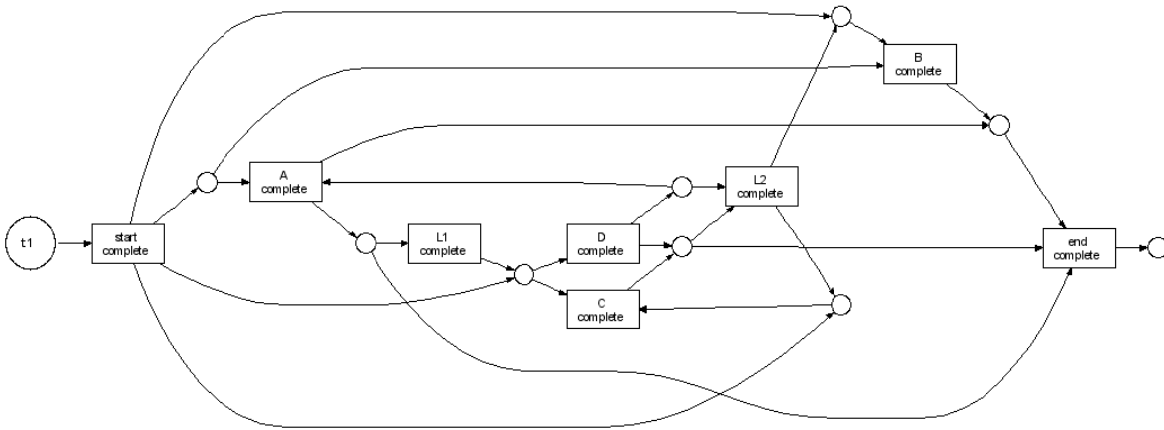


Figure 2: A model discovered by Alpha algorithm of ProM Toolset for 9 instances of Table 1

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. Preliminaries

We start this discussion by introducing four basic structures in block structured processes and two more structures called optional task and self loop. 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 Basic Structures – Sequence, Parallel, Choice, Loop, Optional and Self-loop Structures

A *block structured process* is created by using four basic structures as building blocks: *sequence*, *parallel*, *choice* and *loop*. These are shown in Figures 3(a)-(d). The four basic structures are abbreviated as "S", "C", "P" 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.

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 constructs – *optional (OP)* and *self-loop (SL)* structures are introduced. An optional structure (see Figure 3(e)) is a choice with one empty branch. A self loop (Figure 3(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. Next we describe our algorithm for extracting process models from logs. In the remaining discussion we use the terms *task* for atomic, lowest level tasks, and the term *sub-block or block* for either an individual task or an aggregate of two or more tasks created by using one of the four structures of Figure 3.

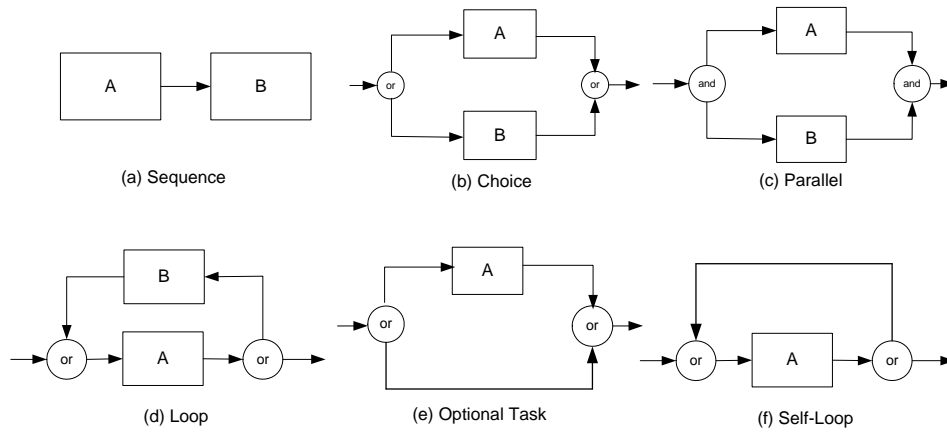
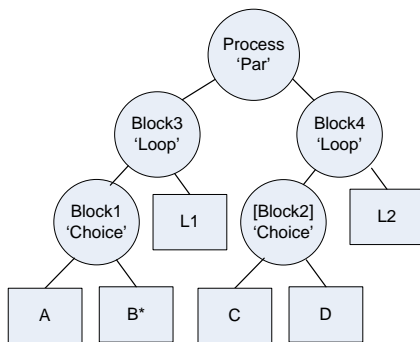


Figure 3: Types (or patterns) used in block structured workflows

The process of Figure 1 can be represented in a succinct form as a model tree and accompanying optional and self-loop vectors as shown in Figure 4(b). The optional and self-loop vectors correspond to the list of tasks and blocks: (A, B, C, D, L1, L2, Block1, Block2, Block3, Block4, Process). Figure 4(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 =

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

Self-loop SL vector: (0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Optional OP vector: (0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0)

SLOP vector= (0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0)

(b) Tuples and vectors

Figure 4: Model representation as tree and as tuples and vector

## 2.2 Universal Process Models

Figure 5(a) shows a model that can describe any given log, even a random one! Thus, given any log containing a certain number of instances of  $N$  tasks, say Task 1 through Task  $N$ , this model can explain that log. This model consists of a choice node with  $N$  branches to the  $N$  different

tasks. Note that to strictly follow our earlier discussions this choice node with  $N$  branches would actually be achieved through multiple choice nodes with two branches in a hierarchy. Any and exactly one of the  $N$  branches can 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 5(b) is another universal model. *These models can simulate any log.* Other universal process models can be made in a similar way. In fact, any given model can be converted into a universal model by making every task and block as both optional and self loop.

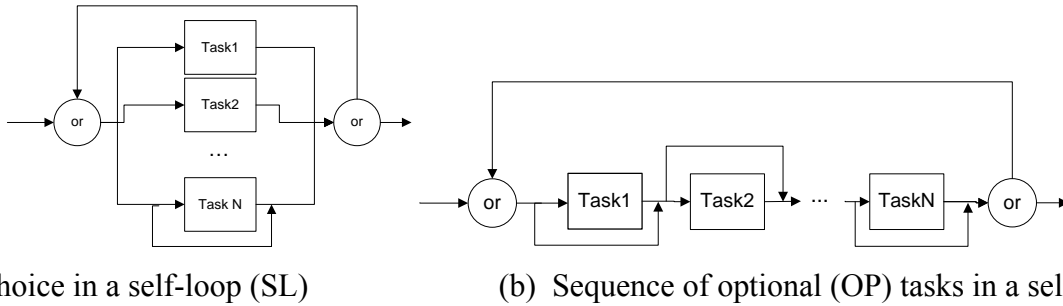


Figure 5: Two universal process models that can explain any log

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

There is a major problem with the universal model. The *quality* of this model is *bad* since it can simulate *every* log. Thus, there is no way to distinguish between a process model for a given log 1 versus the process model for another log 2. Hence, this model lacks *specificity*.

This has given rise to the issue of process model quality with a need for being able to distinguish good models from bad ones (Rozinat, et al. 2007). There are two aspects of quality of a model:

*Fidelity*: i.e. degree to which the model explains a log = 
$$\frac{\# \text{ log instances explained by the model}}{\text{Total \# of instances in the log}}$$

*Specificity*: i.e. degree to which a model is specific to a given log = 
$$\frac{\# \text{ log instances in the log}}{\text{Total \# of paths in the model}}$$

Ideally, a perfect model for a given log is one with both high fidelity and specificity. (Rozinat and Aalst 2008) use the term *fitness* for fidelity and *behavioral appropriateness* for 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. 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. Specifically the following axioms enumerate some principles for designing appropriate quality metrics:

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

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

Axiom 3: If two models explain the same log, the one that permits more traces or instances should have a lower quality value than the other one.

These axioms of quality are straightforward from an intuitive point of view. Axiom 1 simply states that if there is any model, other than the universal model, that does not explain every log but has a worse quality value than the universal model, the metric would not be sensible. Axiom 2 creates a benchmark whereby all universal models (since there are several of them) receive the same quality value (the worst) since they explain all logs, and are on par in quality terms. Finally, Axiom 3 is required because a log can be explained by many different models, and, in general, when two models can explain a log, we want to pick the one that is more specific to a log. 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 A and B 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. Figure 6 shows three different models. In fact, all three are identical and one can easily verify that they allow the same execution sequences. A sequence like "A B A A B A A B" can be executed on any of three models. Yet the SL and OP markings are different as shown on the figure labels in the format:

<{(A, B, L, BL1)}, (SL\_A, SL\_B, SL\_AB), (OP\_A, OP\_B, OP\_AB) >

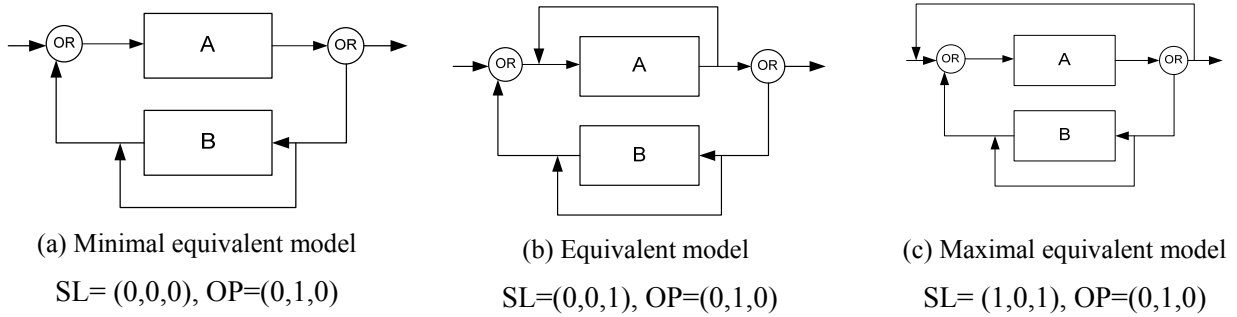


Figure 6: Three equivalent models

Arguably the model with the least number of markings shown in Figure 6(a) is the best in the sense that it is most succinct and yet has the same expressive power as the other two, while the one in Figure 6(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. Since the SL and OP tags affect the quality of a model, for every model we will determine its quality as the one for 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. Therefore as long as the quality metric is developed based on the SL and OP structures of the rewritten maximal equivalent model, the universal models (see Figure 4) will always be assigned the same quality value satisfying Axiom 1 and Axiom 2 in Section 2.3.

For visualizing the model one may use the minimal model also since it is equivalent to the maximal model. 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.

### 3.2 Badness Score

As mentioned above, 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

self-loops and optional tasks and blocks is "more bad" than one with fewer of them. 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  $n$ ,  $op()$ ,  $sl()$ , and  $p()$  are 0-1 functions to indicate whether a given block is of optional, self-loop, and parallel structures, 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 the parallel structure 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 structures because there no clear dominance of a parallel structure over a sequence is present.

This design also ensures that the Axiom 1 and Axiom 2 are still satisfied as the universal models scores are preserved to be the same. 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 coefficients (Spearman 1904) of 0.9318. We used the 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.

The badness score we proposed in this study is an approximate measure that is relatively easy to understand and compute, yet it adequately reflects the notion of model specificity. In future research we plan to further investigate more sophisticated ways to characterize model specificity. We recognize our badness score does not perfectly measure the specificity of a given process

model, i.e., the number of all possible log instances that can be generated. First of all, such a perfect measure may not exist as with the existence 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 or OP structure may produce an infinite number of log instances.

### 3.3 Quality Metrics

In this section we introduce three quality metrics. A simple metric Q0 is based on counting the number of self-loops and optional structures in a *maximally equivalent process model* of a given model as follows (where  $N$  is the number of tasks):

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

However, the Q0 metric does not reflect the level at which a marking appears in a process tree. Clearly a self-loop or optional marking at a higher level can create a lot more log instances, and hence hurt the quality more than such a marking at a lower level. Therefore we consider a new metric based on the badness score. This is defined as:

$$Q1 = \frac{\text{Log}(N)}{\text{Log}(\text{Badness score of actual model})}$$

The numerator represents the logarithm 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. A third metric Q2 is as follows:

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

All three metrics are based on the number of SL and OP markings of the pure block-structured model. Adding more of these markings increases fidelity and reduced specificity. For models with the same number of tasks, the universal models all have the worst (and equal) values for all three metrics. Q0 and Q2 are always zero for universal models, while with a larger number of tasks (say, more than 10) Q1 is also a small number for universal models and it gets close to zero as the number of tasks increases.

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. The correlation coefficient for Q1 and Q2 are identical as both are monotonic transformations of the badness score.

## 4. A Quality-based Process Mining Algorithm

There are three main parts in the algorithm:

**Create "pure" models:** by successively identifying pairs of tasks and S, P, C, L relationships to merge them using a criterion called the *mismmerge score*.

**Replay** the log against each model and determine a SLOP vector of SL and OP markings to apply to each pure model for each log instance that disagrees with the model.

**Optimize:** combine the SLOP vectors based on desired accuracy to obtain final models. Report best quality model or top-K models.

### 4.1 Algorithm

Our algorithm assumes that each instance is complete from start to end and is taken from a process model. Also, we assume that the log is sufficiently descriptive of the patterns that appear in the model, e.g. if tasks A and B are in parallel in the model, there should be log entries of the form ...AB... and ...BA... . Moreover if a task is in a loop it should appear multiple times in one or more log instances. The algorithm first builds an *adjacency matrix* called AdjM. This is an  $N \times N$  matrix for  $N$  tasks in the log. 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

Next we build a "pure" model using only the four constructs S, P, C and L by successively combining a pair of tasks using one of these four constructs. In order to find the best task pair and construct or relationship ( $R$ ), we evaluate all potential pairs for merging. Theoretically, given  $N$  tasks there are  $4N(N-1)$  possible combinations of task pairs and relationships in which they can combine. Thus for the example in Table 1, possible combinations are shown in Figure 7(b). In order to evaluate them, we associate a *mismerge (MM) score* with each combination as a count of the behavioral inconsistencies resulting from such a combination with respect to relationships of  $i$  and  $j$  with other tasks in the log. This score is calculated for combination  $(i, j, R)$  based on inconsistencies in the AdjM matrix found on applying the following rules):

Merge consistency rules ( $[i, j]$  refers to entry AdjM $[i, j]$ , and  $s$  is another task, 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**  $([i, s] = 1 \text{ or } [s, j] = 1) \rightarrow [i, s] = [j, s] = [s, i] = [s, j] = 1$
- (4) if  $R = 'L'$ , **then**  $[i, j] = 1$  **and**  $[j, i] = 1$  **and**  $([j, s] = 1 \text{ or } [s, j] = 1) \rightarrow [i, s] = [j, s] = [s, i] = [s, j] = 1$

These rules ensure that the correct semantic behavior of each merge type is reflected in the log. 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 requires 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 before  $i$ , or before  $j$ , should be in parallel with both  $i$  and  $j$ . Similarly, Rule 4 requires that 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  $j$  should be in parallel with both  $i$  and  $j$ . Table 1 shows how the calculation for the MM score is performed using the above rules for the possible merge of tasks C and D. Each column represents a condition of a rule and a row shows how a condition applies to a type of merge relationship (S, C, S, or P). A '-' means that a condition does not apply. A 'No' entry is marked for each condition violation, and the 'No' entries are added up to get the MM score that reflects the unsuitability of a candidate merge with respect to the AdjM matrix of the log.

Type	[C, D] = 1	[D, C] = 1	[C, D] = 0	[D, C] = 0	[C, s] = [D, s] = [s, C] = [s, D] = 1	[C, s] = [D, s] [s, C] = [s, D]	MM score
'P'	No	No	-	-	-	No for s = A, L2	4
'C'	-	-	Yes	Yes	-	No for s = A, L2	2
'S'	No	-	-	Yes	No for [A, D], [L2, D], [Start, C], [Start, D], [End, C], [End, D] ([C, s] = 1 → s = A, B, L2, End; [s, D] = 1 → s = Start, B, L1)	-	7
'L'	No	No	-	-	No for [A, D], [L2, D], [Start, C], [Start, D], [End, C], [End, D] ([D, s] = 1 → s = A, B, L2, End; [s, D] = 1 → s = Start, B, L1)	-	8

Table 1: Calculation of mismerge score for possible merges between tasks C and D

Note that these scoring rules assume a complete log of a given process model where all possible correct adjacency relationships between tasks are captured. In practice this seldom happens. Therefore, our algorithm searches for the *most probable pure models* and selects the best one from among them. Even with an incomplete or noisy log, a "bad" candidate merge will be penalized more than a "good" candidate merge by the MM score. The presence of SL and OP structures in the true model can also add noise to distort the AdjM matrix and the MM score. However, if these distortions are small then the relative ordering of the MM scores is preserved and the correct pure model is still discovered.

Figure 8 gives a high level overview of the algorithm. These rules can be applied to the AdjM matrix in a function called **ProposePair** to determine the MM score of each possible combination, and the one with the least score is returned. The merging tasks are replaced with a new block task (tasks A, B merge into A-B) in the AdjM Matrix and it has one fewer row and column. The merge is performed by means of a bit-wise logical OR operation on the rows and columns of the merging tasks. This procedure of finding two "best" candidates from the AdjM matrix is repeated until a complete process is built.

```

Algorithm Extract_Model (Log, AP)
Compute AdjM[][] Matrix from Log.
all_tasks = {set of all tasks in Log}
in_tasks = {}
while (all_tasks  $\not\subseteq$  in_tasks )
    (i, j, R) = ProposePair(AdjM)
    Model = Model U (i, j, R)
    AdjM = AdjM[][] U {OR(row i, row j)} U { OR(col i, col j)} – {row i, row j, col. i, col j}
    in_tasks = in_tasks U {i, j}
SLOP[][] = Replay(Log, Model)
Remove (100-AP)% rows from SLOP[][]
Model = Model U {SLOP[1][] OR SLOP[2][] OR ....}
Q = Quality(Model)
Return(Model, Q)

```

Figure 8: Overview of algorithm

After a basic model with S, P, C and L constructs is created, the log is **replayed** against it to check if they agree. Any inconsistency between the log and the basic model is resolved by applying optional and self-loop markings to the model. We maintain a list of expected tasks `next_tasks` in the model and update it after each log entry is matched with it. If a `next_tasks` entry in the model does not appear in the log, then an optional structure is placed on successive entries in the model until the desired entry is reached. Also, if an entry appears in the log that has already occurred previously but is not in `next_tasks` list, then a self-loop marker is inserted in the model to force transfer to the entry that reappears. 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' is explained by placing a self loop around task B. In general, this can always be done.

As each log entry is replayed the requisite OP and SL markings can be stored in a SLOP vector like the one in Figure 3 such that there is one such vector for each log entry. After the entire log is scanned all these vectors can be combined (by a union operation) or optimized and applied to the log. If the user provides an accuracy parameter AP (say, 95%) to indicate that (100– AP)% log entries may be discarded, then 5% SLOP vectors with markings with worst quality impact can be dropped and the union of the remaining SLOP vectors is applied to the basic model. The output of the algorithm is the model along with the markings from merged SLOP vector and the quality of the model. Finally, our algorithm backtracks to consider candidate merges in ascending order of MM scores, and generates other models. Eventually, the

model with the best quality among the candidate models examined is reported. It is also possible to report the Top-K quality models. A full algorithm using **Best-First** backtracking search (on the least MM score of a partial model) is given in Appendix A and an example in Appendix B.

As for most backtracking algorithms, the worst-case complexity of our algorithm is high at  $O(N^{2N})$  because there are N steps and in each step  $N(N-1)$  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.

## 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 in turn 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. In this section we report several experiments on simulated logs. The main purpose of these experiments is to examine how effectively our quality based algorithm is able to discover models that are similar to the original models when it is given logs produced from the original models of varying degrees of quality.

### 5.1 Simulated Log Data Generation

We first create a simulated process model, and then use it to generate a log. We start with a list of tasks and then pick two tasks at random to be combined in one of four basic structures (sequence, choice, parallel and loop). As an example, in Figure 9 tasks t3 and t6 are chosen from 10 tasks to combine in a P structure to create block b1, and then two other blocks b2 and b3 are formed similarly. Next b2 and b3 are combined to form b4, while t1 and b1 combine to form b5. In this way, pairs of unattached tasks or blocks are selected randomly and further combined until there is one large block that represents the final process. 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.

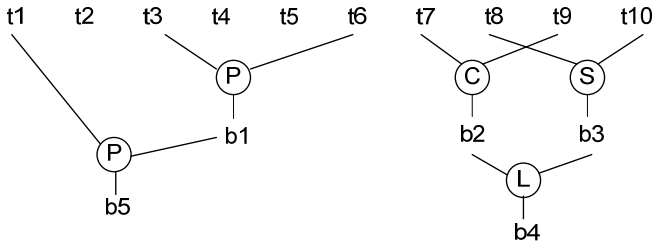


Figure 6: Generating a process model

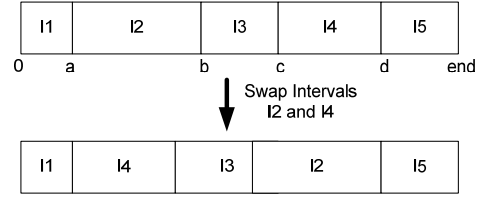


Figure 7: Simulating noise

In addition to the above experiments on noise-free logs, we also conducted experiments on simulated logs with *artificially induced noise*. We define a general noise operation on a log instance as (see Figure 10):

$$row = row[0:a] + row [c:d] + row [b:c] + row [a:b] + row[d:end]$$

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. However, a target accuracy of 100% was used with the noise-free logs.

## 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. We used 500 log instances as input in our experiments and set the stopping criterion for the backtracking algorithm as 100 models being evaluated. We used set the stopping criterion of 100 models for our experiments on both the synthetic and real data based on reasonable trade-off between running time and model quality obtained. For certain difficult process model, it may require more models to be evaluated to find the high-quality exact true model at the expense of additional computational time committed. It

should also be noted that in our experiments a value of  $W_1 = 1$  and  $W_2 = 5$  was used, while in practice users can adjust the weight based on relative importance of SL and OP structures.

In this experiment we used all three versions of 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, 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 were from 5 groups with the following numbers of randomly assigned SL and OP structures: (0,0), (0,1), (1,0), (1,1), and (2,2). The numbers of SL and OP structures reported in Table 2 are often greater than the number of structures randomly assigned. This results 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 *mismmerge 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 one 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 as can be seen 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 in prior sections. 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.

Closer examination reveals that the algorithm found an equivalent model as the true model with better quality metric. Specifically, a block consisting of three tasks in the true model  $\langle \{(a, b, 'C', B1), (c, B1, 'P', B2)\}, (1,1,0,1,0), (0,0,0,0,0) \rangle$  becomes  $\langle \{(c, b, 'P', B1), (a, B1, 'P', B2)\}, (1,1,0,0,0), (1,0,1,0,0) \rangle$ . 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 learned models with much more SL and OP structures than 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 be reached by evaluating more 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 5.1.2 from 10 random process models, 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 impose the SL and 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 table of results is omitted for brevity, but the findings are summarized next. A key criterion for evaluation was the quality ratio =  $\frac{Q2 \text{ of the learned model}}{Q2 \text{ of the true model}}$ .

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 Model group 1, 27 pure models were learned by the algorithm. However, 3 learned models had non-zero SL and OP structures with one outlier that contributed to the reduction in the quality ratio.

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 reverse direction from 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

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 in this section.

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 is 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 shows the top-10 most frequent tasks in our data. When counting the frequency, we considered the multiple occurrences within the same log instance. The tasks name are self-explanatory for the most part. Notice of allowance is a notification to the applicant that they are entitled to a patent under the law and requesting payment of a specified issue fee. It should also be noted that these were all instances in which patents were eventually issued, in some cases after first being rejected. However, these rejections were "non-final".

Table 3: Top 10 Most Frequent Tasks

Task Id	Task Description	Frequency
0	IFW (Image File wrapper) Scan & PACR (Patent Application Capture and Review) Auto Security Review	31,922
1	Issue Fee Payment Verified	33,608
2	Issue Notification Mailed	31,976
3	Mail Non-Final Rejection	42,067
4	Mail Notice of Allowance	32,968
5	"Non-Final" Rejection	42,067
6	Notice of Allowance Data Verification Completed	32,967
7	Receipt into Publications	76,931
8	Recordation of Patent Grant Mailed	31,799
9	Response after Non-Final Action	42,281

Table 4: Experimental results: patent log instances with varying target accuracies

Exp. #	Target accuracy	(Q2	Actual Accuracy	Best model at #	# SL	# OP
t1	1	0.4974	1	6	16	16
t2	0.95	0.6528	0.9755	44	9	10
t3	0.9	0.6528	0.9755	44	9	10
t4	0.85	0.6529	0.8630	44	9	9
t5	0.8	0.6577	0.8436	17	7	5
t6	0.75	0.6577	0.8436	17	7	5
t7	0.7	0.6577	0.8436	17	7	5
t8	0.65	0.6558	0.6863	10	6	4
t9	0.6	0.6558	0.6863	10	6	4
t10	0.55	0.6560	0.5586	10	5	5
t11	0.5	0.6591	0.5449	17	5	5

For a real logs like this, 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, we ran

the algorithm with varying target accuracies, using Q2 as the quality metric. Table 4 shows the Q2 and accuracy of the best model achieved within the first 100 searched models (and the sequence number at which the best model was found) with different target accuracies. When the target accuracy was set to 1, the best model had a quality of 0.4947 and 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 5 OP and 5 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.

Figure 11 shows three process models for Experiments 1, 2, and 8 of Table 4. The labels "S", "L", "C", and "P" denote the corresponding types of blocks. Each split node (with suffix 's') has a corresponding join or end node (with suffix 'e'). Thus, P17s is a parallel split node which matches with P17e. A legend is provided alongside to indicate the symbols used for OP tasks and blocks. The self-loop tasks and blocks are indicated using a self-loop in these diagrams. Across the three models, we observe consistent local structures such as the sequence structure consisting of t6 and t4, the sequence structure consisting of t5 and t3, the parallel structure consisting of t1 and t7. Moving up to a higher level, we see that the parallel block consisting of t0, t2, t8, and t9 and the loop block consisting of t5, t3, t1 and t7 were present across the four models as well. While these local structures make intuitive sense for non-experts, many differences are also there between the models which reflect the presence of noise.

The four models vary in how the local structures are assembled into the complete model. Model (a) differs from the other two models in that the root-level block in this model is a loop structure with an OP parallel sub-block. There are also many OP and SL structures at the lower level blocks/tasks clearly suggesting it is a low quality model. Model (b) has a parallel structure at the root consisting of the loop sub-block with t5, t3, t1 and t7, and the parallel sub-block comprising of the remaining tasks. The sequence structure consisting of t6 and t4 is now within a loop structure together with t0. The resulting model successfully explained 97.55% log instances with 9 SL structures and 10 OP structures, and a quality metric of 0.6528. Model (c) differs from Model (b) in that the non-OP sequence block consisting of t6 and t4 becomes one of four parallel branches from the root.

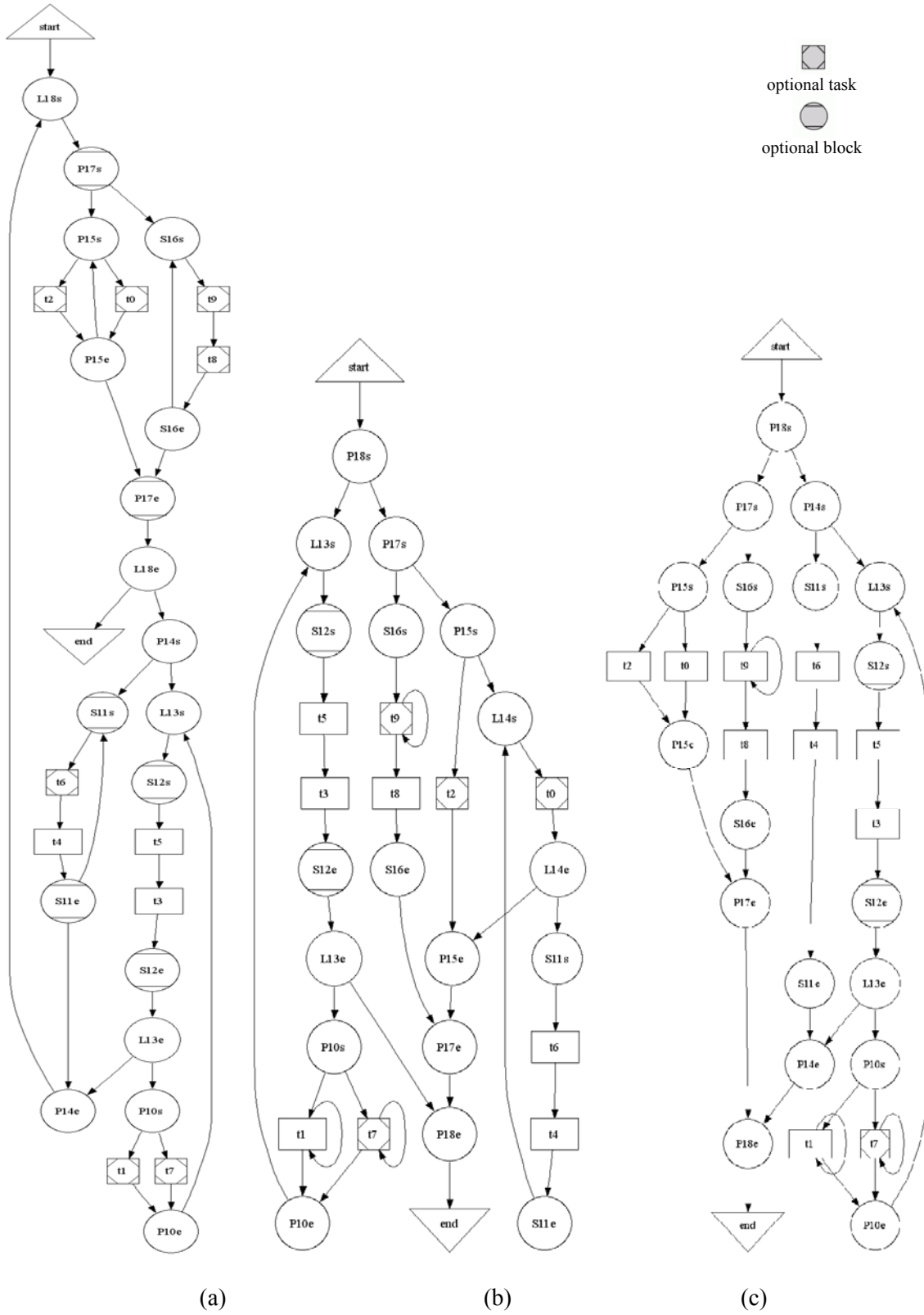


Figure 11: Patent application process models: (a) accuracy = 100%;  $Q = 0.4974$   
 (b) accuracy = 97.55%;  $Q = 0.6528$  (c) accuracy = 68.63%;  $Q = 0.6558$ .

There are several insights from this real log experiment. 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 despite varying the accuracy, some fundamental aspects of the process were retained in all models. A domain expert can even combine pieces of the models and rearrange them in order to obtain a more accurate model.

## 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 focus 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 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, 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 were found from their logs.

*There is an inverse relationship between noise and quality.* When there is more noise 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 tests 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 described in (Agrawal, et al. 1998) extracts dependency graphs from task logs based on a dependency analysis between tasks. This algorithm was applied in the context of the FlowMark workflow system from IBM. (Aalst, et al. 2003, Aalst, et al. 2004) have proposed the Alpha algorithm (see Figure 3) and variations of it for extracting Petri net models. These algorithms are based on extracting causal dependencies from logs, and combining tasks that depend on one another by treating them as transitions and placing tokens to connect them. The approach is implemented in the ProM tools (Dongen, 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 approaches to induce a final model. The initial work (Herbst and Karagiannis 1998) was limited to sequential models, but parallelism was introduced later (Herbst and Karagiannis 1999).

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, et al. 2004). A probabilistic model and a learning algorithm for workflow mining are discussed in (Silva, et al. 2005).

There has been very little focus on process mining in the context of noisy data. In this respect our approach based on the notion of a Q-metric is quite novel. Related work with a focus on

process quality is discussed in (Greco, et al. 2006, Rozinat and Aalst 2008, Rozinat, et al. 2007). In (Rozinat and Aalst 2008), the notion of quality is introduced in the context of Petri-nets. Metrics are presented there to examine fitness, and structural and behavioral appropriateness. These metrics are supposed to be used together in order to assess overall quality although a combined metric is not provided. The fitness metric in (Rozinat and Aalst 2008) is based about half on the proportion of missing tokens to consumed tokens and this part captures the idea of what we call OP tasks. The other half is based on the ratio of remaining tokens to produced tokens which captures the effect of self-loops in our terminology. The notion of replay is similar in both approaches. Another approach (Greco, et al. 2006) extracts unstructured And-Or graph models without loops. A new aspect in this approach is to develop multiple models by clustering log instances and then use these models together. However, the drawback is that users normally like to use one model, not multiple models in conjunction.

Compared to Petri-net models, block structured process models have less expressive power, but they are easier to describe and understand from an end-user point of view (compare Figure 1 and Figure 2 for instance). One key difference is that in Petri-net terminology, block structured models cannot describe non-free-choice processes. This would require additional variables to be associated with choice conditions. Invisible (or "silent") tasks in Petri-nets are similar to our OP tasks. Duplicate tasks are not allowed in our model. Finally, the quality metrics in a Petri-net context are described in terms of tokens, places and nodes, whereas our metrics are in term of SL and OP structures as discussed above. Our metric captures the notions not only of fitness but of structural and behavioral appropriateness also because we consider many initial models and pick one that requires the minimum SL and OP markings. A bad initial model will need more markings and result in less specific behavior which the metric will reflect. At the same time, another advantage of our approach is that it 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. This does not happen because we start with initial candidate models that are structurally sound.

Another line of work that dates back a few years (see for instance, (Mărușter, 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 based on frequency tables that maintain counts of how often one task succeeds

another one. While we do not keep frequencies explicitly in our approach, yet, they do influence our resulting model implicitly. This is because noisy instances are likely to generate more SL and OP structures, and these are the first ones our algorithm will discard while optimizing the model for 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 (Medeiros 2007, Song, 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, or perhaps use all of them in conjunction. The quality metrics that we have discussed can be applied to each model, and can inform the relative quality of each cluster.

## 8. Conclusions and Future Work

Process mining is an emerging research area of important 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 between 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 approach. The results were surprisingly encouraging in both scenarios. With simulated data where the true model was known, we were able to predict the model with considerable accuracy. With the real data from a patent approval process, we generated three good models at different degrees of accuracy. Our approach gave us valuable insights into the issue of quality-accuracy trade-off that is very important when noise is present in the process logs.

Our prototype tool takes a log as input and converts it into a process model. In future work, we will work on improving our Q-metric and assessing its effect on the performance of our

algorithm. We will also work on making our algorithm faster and more scalable. Finally, a later extension would be to find ways to extract models for more complex patterns.

## References

- Aalst, W. M. P. v. d., B. F. v. Dongen, C. W. Günther, R. S. Mans, A. K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, A. J. M. M. Weijters and H. M. W. Verbeek. 2007a. ProM 4.0: Comprehensive support for real process analysis, *Proceedings of the 28th International Conference on Applications and Theory of Petri Nets 2007*, Siedle, Poland 484-494.
- Aalst, W. M. P. v. d., A. K. A. d. Medeiros and A. J. M. M. Weijters. 2007b. Genetic process mining: an experimental evaluation, *Data Mining and Knowledge Discovery*, **14**(2) 245-304.
- Aalst, W. M. P. v. d., H. A. Reijers and M. Song. 2005. Discovering social networks from event logs, *Computer Supported Cooperative Work*, **14**(6) 549-593.
- Aalst, W. M. P. v. d., H. A. Reijers, A. J. M. M. Weijters, B. F. v. Dongen, A. K. A. d. Medeiros, M. Song and H. M. W. Verbeek. 2007c. Business process mining: An industrial application, *Information Systems*, **32**(5) 713-732.
- Aalst, W. M. P. v. d., B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm and A. J. M. M. Weijters. 2003. Workflow mining: A survey of issues and approaches, *Data and Knowledge Engineering*, **47**(2) 237-267.
- Aalst, W. M. P. v. d., A. J. M. M. Weijters and L. Maruster. 2004. Workflow mining: Discovering process models from event logs, *IEEE Transactions on Knowledge and Data Engineering*, **16**(9) 1128-1142.
- Agrawal, R., D. Gunopulos and F. Leymann. 1998. Mining process models from workflow logs, *Proceedings of the 6th International Conference on Extending Database Technology*, 469-483.
- Chesani, F., P. Mello, M. Montali, F. Riguzzi, M. Sebastianis and S. Storari. 2008. Checking compliance of execution traces to business rules, *Proceedings of the 4th Workshop on Business Process Intelligence (BPI 08) in conjunction with BPM 2008*, Milan, Italy.
- Dongen, B. F. v., A. K. A. d. Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters and W. M. P. v. d. Aalst. 2005. The ProM framework: A new era in process mining tool support, *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets*, 444-454.
- Dustdar, S., T. Hoffmann and W. M. P. v. d. Aalst. 2005. Mining of ad-hoc business processes with TeamLog, *Data and Knowledge Engineering*, **55**(2) 129-158.
- Greco, G., A. Guzzo, L. Pontieri and D. Sacca. 2004. Mining expressive process models by clustering workflow traces, *Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 52-62.
- Greco, G., A. Guzzo, L. Pontieri and D. Sacca. 2006. Discovering Expressive Process Models by Clustering Log Traces, *IEEE Transactions on Knowledge and Data Engineering*, **18**(8) 1010-1027.
- Herbst, J. and D. Karagiannis. 1998. Integrating machine learning and workflow management to support acquisition and adaptation of workflow models, *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, 745-752.

- Herbst, J. and D. Karagiannis. 1999. An inductive approach to the acquisition and adaptation of workflow models, *Proceedings of the Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business*, 52-57.
- Klein, M. and A. Bernstein. 2004. Towards high-precision service retrieval, *IEEE Internet Computing*, **8**(1) 30-36.
- Mans, R. S., M. H. Schonenberg, M. Song, W. M. P. v. d. Aalst and P. J. M. Bakker. 2008. Process mining in healthcare - A case study, *Proceedings of the HEALTHINF 2008*, Funchal, Madeira, Portugal 118-125.
- Märuşter, L., A. J. M. M. T. Weijters, W. M. P. v. d. Aalst and A. v. d. Bosch. 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.
- Medeiros, A. K. A. d. 2007. Process mining based on clustering: A quest for precision, *Proceedings of the BPM workshops in conjunction with BPM*, 17-29.
- Rozinat, A. and W. M. P. v. d. Aalst. 2008. Conformance checking of processes based on monitoring real behavior, *Information Systems*, **33**(1) 64-95.
- Rozinat, A., A. K. A. d. Medeiros, C. W. Günther, A. J. M. M. Weijters and W. M. P. v. d. Aalst. 2007. The need for a process mining evaluation framework in research and practice, *Proceedings of the 3rd Workshop on Business Process Intelligence*, 84-89.
- Schimm, G. 2000. Generic linear business process modeling, *Proceedings of the Workshops on Conceptual Modeling Approaches for E-Business and The World Wide Web and Conceptual Modeling: Conceptual Modeling for E-Business and the Web*, 31-39.
- Schimm, G. 2003. Mining most specific workflow models from event-based data, *Proceedings of the International Conference on Business Process Management*, Eindhoven, The Netherlands 25-40.
- Silva, R., J. Zhang and J. G. Shanahan. 2005. Probabilistic workflow mining, *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, Chicago, IL 275-284.
- Song, M. and W. M. P. v. d. Aalst. 2007. Supporting process mining by showing events at a glance, *Proceedings of the Seventeenth Annual Workshop on Information Technologies and Systems*, Montreal, Canada 139-145.
- Song, M., C. W. Günther and W. M. P. v. d. Aalst. 2008. Trace clustering in process mining, *Proceedings of the 4th Workshop on Business Process Intelligence (BPI 08) in conjunction with BPM 2008*, Milan, Italy.
- 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 and V. Kumar. 2006. *Introduction to data mining*, Addison Wesley,
- Weijters, A. J. M. M. and W. M. P. v. d. Aalst. 2003. Rediscovering workflow models from event-based data using Little Thumb, *Integrated Computer-Aided Engineering*, **10**(2) 151-162.
- 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.

## APPENDIX A

Long version of the Algorithm for finding the model with the highest quality among a given number of most probable candidate models with desired accuracy from a log.

```

Algorithm Extract_Model (Log, Taskset, Global Max_Limit, Global Desired_Accuracy)
  Build AdjM; set Global model_count = 0; set Global candidates = {};
  model0.tuples={}; model0.MMScore = 0//a tuple is (task i, task j, relationship R, mismerge (MM) score)
  SearchModel(model0, AdjM, Taskset, Log)
Function SearchModel(model, AdjM, Log)
  If model_count > max_limit then Exit
  tupleList = ProposePair(AdjM)
  foreach (tuple in tupleList)
    model1 = model U tuple
    model1.MMScore = model1.MMScore + tuple.MMScore
    candidates = candidates U model1
  sort candidates on MMScore
  foreach model2 in candidates
    AdjM = Merge(AdjM, model2.lastPair) // lastpair is the first two elements i, j of the last tuple
    If (all Tasks are included in model2)
      SLOP-vector = Replay(model2, Log, Desired_Accuracy)
      Output (model, SLOP-vector, Quality(model2, SLOP-vector))
      model_count ++
    Else: SearchModel(model2, AdjM, Log)
  End SearchModel
Function ProposePair(AdjM)
  newList = {}
  If (rel == 'P') MMScore = (# of mismatches in AdjM between the rows and columns for i, j) +
    (1 - AdjM[i][j]) + (1 - AdjM[j][i])
  If (rel == 'C') MMScore = (# of mismatches in AdjM between the rows and columns for i, j) +
    AdjM[i][j] + AdjM[j][i]
  If (rel == 'S') MMScore = # of zeros in {AdjM[i][j], AdjM[i][s], AdjM[j][s], AdjM[s][i], AdjM[s][j]}
    (where s ≠ i, j and (AdjM[i][s] == 1 or AdjM[s][j] == 1))
  If (rel == 'L') MMScore = # of zeros in {AdjM[i][j], AdjM[j][i], AdjM[i][s], AdjM[j][s], AdjM[s][i],
    AdjM[s][j]} (where s ≠ i, j and (AdjM[s][j] == 1 or AdjM[j][s] == 1))
  newList=newList + (i, j, R, MMScore) //this is a list of tuples
  Return(List)
End ProposePair
Function Merge(AdjM, pair (i, j))
  AdjM = AdjM[][] U {OR(row i, row j)} U {OR(col i, col j)} - {row i, row j, col. i, col j}
  Return AdjM
End Merge
Function Replay(model, Log, Accuracy)
  Compute list of next_tasks that can run from the model
  Initialize occurred tasks to {}
  For (each log instance l)
    If (next_entry e in next_tasks)
      update next_tasks; occurred = occurred U {e}; entry e = next_entry in log:
    Else
      If e in occurred then add a self-loop in model to make e reachable
      Scan Log while skipping tasks until first task that belongs to next_tasks is found.
      Mark SLOP vector for log instance l for the skipped tasks/sub-blocks
  SLOP-vector[] = combine/optimize SLOP vectors for each log instance
End Replay

```

## APPENDIX B

**Example 1:** To illustrate how our algorithm works, we ran it on the log of Table 1. This example is in two parts. In the first part, we ran our algorithm on the first 9 log instances of Table 1. Table B1(a) shows the first adjacency matrix constructed from this log.

Table B1. Initial AdjM matrix and revised AdjM matrix after merging tasks C and D

(a) AdjM matrix									(b) Merged AdjM								
	Start	A	B	C	D	L1	L2	End		S	A	B	C-D	L1	L2	End	
Start	0	1	1	1	1	0	0	0	St	0	1	1	1	0	0	0	
A	0	0	0	1	0	1	0	1	A	0	0	0	1	1	0	1	
B	0	0	0	1	1	1	0	1	B	0	0	0	1	1	0	1	
C	0	1	1	0	0	0	1	1	C-D	0	1	1	0	0	1	1	
D	0	1	1	0	0	0	1	1	L1	0	0	1	1	0	1	0	
L1	0	0	1	1	1	0	1	0	L2	0	0	1	1	1	0	0	
L2	0	0	1	1	0	1	0	0	End	0	0	0	0	0	0	0	
End	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	

Our algorithm merged C and D in the first step. The merged matrix is shown in Figure B2. After subsequent merges it produced the following model (the block numbers are from Fig. 3):  
 Model={ (C, D, 'Choice', Block2), (Block2, L2, 'Loop', Block4), (A, B, 'Choice', Block1), (Block1, L1, 'Choice', Block3), (Block3, Block4, 'Parallel', Process) }

The above model was based on the first 9 instances of Table 1 and does not contain any SL or OP tasks. Next we ran our algorithm to include all 12 instances of Table 1 because the last three instances require use of SL and OP structures. In this case, we were able to find the correct model (exactly as in Figure 1) at the 431st candidate model that was considered. Table B2 shows how the Replay algorithm operates while scanning each log for conformance with the model. The last row of this table shows a composite bit vector combining the three vectors through bitwise OR operation. From this row it is clear that task B is in a self-loop and block B1 (including tasks C and D) is optional. So, this is the final model shown in Figure 1.

Table B2: Self-loop and Optional vectors created by Replay function for log of Table 1

Instance number	Self-Loop Vector										Optional task/block vector										
	A	B	C	D	L1	L2	B1	B2	B3	B4	B5	A	B	C	D	L1	L2	B1	B2	B3	B4
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
11	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Union	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0