Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

A statistical Variant of the Inductive Miner
eine statistische Variante des Inductive Miner

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)

eingereicht von: Martin Bauer
geboren am: 1.6.1994
geboren in: Ludwigsfelde

Gutachter/innen: Prof. Dr. Matthias Weidlich
Prof. Dr. Lars Grunske

eingereicht am: ..... verteidigt am: .....

Inhaltsverzeichnis

Scalability is one of the major challenges process discovery algorithms have to solve. In this work an adaption of the Inductive Miner infrequent is proposed that increases sscalability of the algorithm. The modified algorithm, statistical Inductive Miner infrequent, uses statistical preprocessing to reduce the input log to a smaller size while still maintaining all relevant data. It is shown that the algorithm is faster and more memory efficient than the original technique, while reaching the same fitness as Inductive Miner infrequent. Furthermore the impact workflow structure have on the runtime is analyzed.

1 Introduction

1.1 Process Mining

Today huge amounts of a data is produced in companies and computer-aided systems everywhere. Most of this data can be found in log files that provide an execution trail of the systems inner processes. Depending on the situation the analysis of these trails could provide different benefits, e.g. if and when errors appear in the system or whether security regulations are respected in the system. Examples for log files are operating system logs that are created when an error occurs, transactional data in online shops or records of a monitored workflow management system.
The analysis of log files can be useful for many reasons. A company could use logged data to verify that workers work compliant to regulation rules like the 4-eyes principle; to check if deviations occur or discover bottlenecks that slow down the system.
While the analysis of such logs can prove to be valuable for a user, analyzing by hand would be very tedious and error prone as logs usually contain thousands of entries. A solution is to use Process Mining techniques.

[1] defines process mining as a technique to extract information from event logs. Notable Process Mining architectures are the ProM Framework [2] and Apromore [3].

1.2 Process Discovery - a Subfield of Process Mining

The Process Mining Manifesto [4] defines 3 sub categories of Process Mining:

Process Discovery — given a log, discover a model that represents the behavior in the log as best as possible

Conformance Checking — given a log and a process model of the same process, check if the real life behaviour seen in the log represents the modelled behaviour in the log. Note that there are a multitude of Process Models that describe different aspects of a system, i.e. organizational models, business process models, policy rules or KPI's

Model Enhancement — given a log and a process model of the same process, enrich the model with data of the log. This may include changes to the structure of a business process model or the addition of process times seen in the log

Process discovery may be valuable for users if they do not have a model of their system at hand or if they want to analyze their system based on real data. In both cases they could use a process discovery algorithm to create a model and base their further analysis on the created model.
There are a multitude of possible models that can be discovered, like petri nets, finite state automaton or heuristic nets.

1.3 Scalability of Process Discovery Algorithms

Even if most of the algorithms work fast and reliably on small logs, [5] and [6] show that a lot of these behavioral log analysis tools have problems dealing with large logs. They either take a lot of time to complete or are not able to finish their computation at all. The design of scalable process discovery algorithms is also depicted as one of the major challenges of process mining in [4].
As more and more data gets created every day [7] this is a challenge that gets more important every day.

1.4 Contribution and Structure

The goal of this work is to create a variant of the Inductive Miner infrequent - IMi, called statistical Inductive Miner infrequent - sIMi, that uses statistical log analysis prior to the execution of IMi to reduce the size of the log file in order to enable IMi to analyze data faster. The reduced log will have all relevant information that is needed in order to discover the correct model and to approximate the average cycle time of the model
The algorithm will be evaluated upon the following questions:

- Does sIMi hold practical advantage against IMi in a real-life scenario and is it a more scalable solution?

- Are there behavioral properties in logs that have an impact on the algorithm's performance

Section 2 introduces needed terms, definitions and presents the basic algorithms sIMi is build upon on. Section 3 then presents the algorithmic framework of sIMi in detail, while section 4 contains a performance evaluation of the algorithm with regard to the two questions. The work is concluded with a synopsis that gives answers to the aforementioned questions in section 5.

2 Preliminaries and Related Works

2.1 Definitions and Notations

2.1.1 Events, Traces, Logs

Event Given a set of possible activities that may happen in a process, denoted as Σ, then an occurrence of an activity $e \in \Sigma$ is called an event.

Trace A trace t is an ordered sequence of events $e^* \in \Sigma$. A trace may be empty. It's size $|t|$ is the number of events it contains.
A trace corresponds to a process, which can be seen as an ordered sequence of activities.

Log A log is a finite multiset of traces $L \subseteq \Sigma*$. A log must not be empty. The size of a log $|L|$ is the number of traces it contains.
1 depicts a simple logfile and it's components.

user id	activity	timestamp
3456	A	03-07-2016
3456	B	03-07-2016
4788	A	04-07-2016
3456	C	04-07-2016
4788	D	04-07-2016

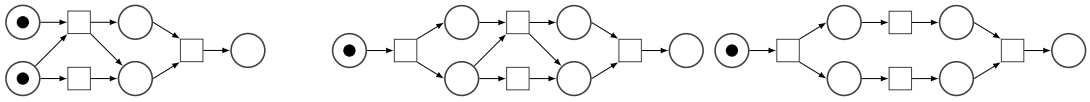
Figure 1: An exemplary log file. It consists of the two traces 3456 and 4788. Trace 3456 is an ordered secuence of the events <A,B,C>; trace 4788 is an ordered sequence of the events <AD>. Each event is annotated with a timestamp that shows when the event occured.

2.1.2 Block structured Workflow-Nets

A petri net is a bipartite graph consisting of places and transitions, where arcs exist only between places and transisions but not between places and places, or between transitions and transitions. [8] explains the standard semantics of petri nets.
Petri nets can be used to model behavior that is described through a log. Transitions correspond to an activity and places guarantee a certain order of these activities.
Traces have a single start- and end-activity, furthermore every trace is considered a finished process, as such a model that tries to represent this behavior has to ensure that all processes terminate and that the start-and end points are well defined.

Worklow nets are petri nets that have only a single start-and ending place [9]. If such a workflow net can be divided recursively into subnets that have single start- and ending places, then it is called a block structured workflow net [10].

Block structured workflow nets are guaranteed not to contain any dead- or lifelocks, so they always depict complete processes and have well-defined start- and ending places, therefore they are eligible to represent the behavior of a log. See 2 for examples of the three petri net classes.



(a) a Petri net (b) A workflow net (c) A block structured workflow net

Figure 2: Example for petri nets, workflow nets and block structured petri nets. Note that the workflow net does not prevent deadlocks, so it is can't represent a log-based model. The block structured approach on the other hand does prevent deadlocks and is therefore able to represent behavior of traces.

2.1.3 Directly Follows Graphs

The events in a log are part of ordering relations [11], e.g. for the trace <ABC> A directly preceeds B and B directly preceeds C. This relation can be expressed via a directly follows graph, that has an edge from node A to node B only if A directly preceeds B at least once in the log.

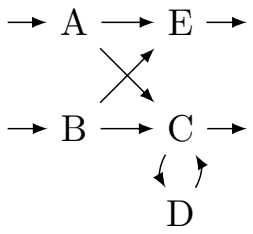


Figure 3: A directly follows graph made from the traces <AE>,<BC>,<ACDC> and <BE>. It captures all follows-realations and the events that are either a first or a last event in a trace.

2.1.4 Process Trees

A process tree, defined in [12], is a tree-like abstraction of a block structured workflow net. The trees inner nodes represent workflow operators, while the leaves represent events. The basic operators that a process tree supports are $\rightarrow$,$\bigwedge$,$\times$ and $\circlearrowleft$, which correspond to the workflow patterns sequence, parallel split & join, xor split & join and cycle defined in [13]. Note that $\bigwedge$ and $\times$already include their split and join.
Each operator has at least two children which are either events, operators or τ, the silent activity which corresponds to an empty or invisible event.
Each operator defines how it's children, denoted as $a_1, ..., a_n$ have to be combined as follows:

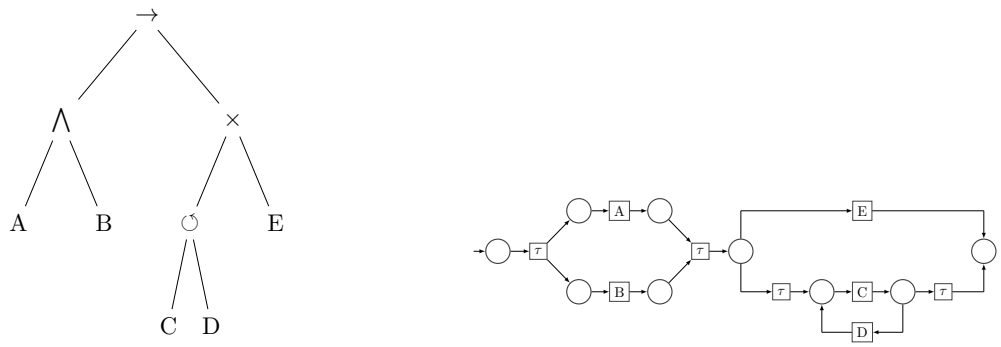
$\rightarrow (a_1, ..., a_n)$ - the children are executed from left to right, i.e first a_1, then a_2 and so on

$\bigwedge(a_1, ..., a_n)$ - all children are executed in parallel and intertwined

$\times(a_1, ..., a_n)$ - exactly one of the children $a_1, ..., a_n$ is executed

$\circlearrowleft(a_1, ..., a_n)$ - first a_1 is executed, then one of $a_2, ...a_n$ followed by a_1 may be be executed. The execution of one of $a_2, ...a_n$ followed by a_1 may repeat.

Process trees can easily be converted into other model languages. 4 shows a process tree and the corresponding block-structured petri-net.



(a) a Process Tree (b) a Petri Net

Figure 4: A process tree and the corresponding block structured petri net. Note that every operator in the process tree is a block in the petri net. τ is the silent transition that is not seen in a log. It bounds loop- and and-blocks to ensure correct behavior.

2.1.5 Quality of a Model

If a model has been discovered an important question is how good the algorithm fares in comparison to others. For this four quality criteria are used that describe different properties of the model the algorithm discovers:

Fitness Fitness describes to what extend the discovered model is able to replay the traces in the log

Simplicity Ideally an algorithm should produce simple models. Complex models may be correct, but hard to analyze

Precision While it is important that a model can produce all behavior in the log, it should not be able to replay too much additional behaviour. Precision captures this aspect. A model that can replay all behavior possible is fit but does not have any value for analysis.

Generalization Generalization depicts a models ability to reproduce future behavior, i.e. behavior that has not been seen but that will eventually happen.

The different criteria contradict themself partially. A precise model may not be a good generalization, while a fit model may not be precise. Different ways to measure these criteria have been proposed, e.g. for measuring fitness some different methods have been proposed in [14], [15] or [16].

2.2 Types of Process Discovery Algorithms

[17] gives an overview of process discovery algorithms that mine petri nets by dividing them into five categories and explaining the shortcomings or benefits of the categories as follows:

Abstraction-Based	these algorithms first abstract the log by finding ordering relations between events and then construct a model based on these relations. The α-algorithm [11] and it's extensions are the most prominent algorithms of this class
Heuristic-Based	heuristics-based algorithms take into account how frequent behaviour is in the log and are able to filter out low-frequent behavior, called noise, and create a model that represents the main behavior of the log. This is in contrast to abstraction-based algorithms who incorporate all behavior of the log into the final model. Heuristics Miner [18] is an example of such an algorithm
Search-Based	heuristics-based algorithms work on local information of the log, e.g. directly follows-relations and frequencies of these relations, but some workflow constructs are dependent on global information that is not accessible via direct relationships between events. Algorithms like proposed in [19] use genetic evolution of a population of models to find the most fitting model for the log. These genetic approaches generate a model and check how well it can parse the traces in the log and evolve the models that are able to reproduce the bahviour of the log the closest. As this procedure has to generate and test models multiple times it's runtime is usually higher than that of other methods
Language-Based Region Algorithms	such algorithms that view a log as a bag of words and that try to generate a model by using language-based region theory like in [20] are called language-based

State Discovery Algorithms — state discovery algorithms view each trace in a log as a transition system. All of these systems are then combined into a single transition system which is then converted into a petri net, like described in [21]

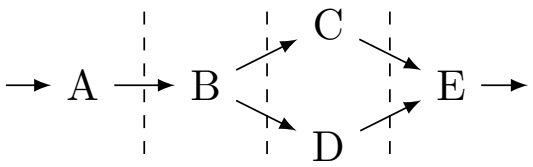
On the problem of scalability many proposals have been made. [5] uses MapReduce to make specification mining algorithms more scalable. [22] partitions the traces passages, which are sets of events X and Y for which X always preceeds Y and Y always follows X. These passages can be analyzed independently.
For algorithms that use the theory of region to discover a model the main bottleneck is memory consumption. [23] tries to minimize this bottleneck by splitting the input, analyzing each subset and combining the analyzed parts again.

2.3 Inductive Miner and Inductive Miner infrequent

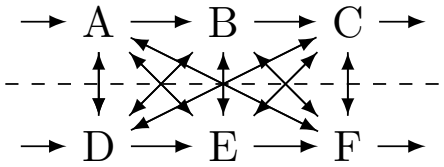
Inductive Miner Many abstraction-based algorithms are not able to rediscover models that are sound and fitting in finite time, e.g. the alpha-algorithm is not able to discover short loops and produces an unsound model, i.e. models containing deadlocks.
As a solution Inductive Miner [10] has been proposed. Inductive Miner is an abstraction-based process discovery algorithm that uses a divide-and-conquer-approach to discover a process tree. The algorithm is able to rediscover a sound and fitting model in finite time.

To do so, first the algorithm transforms the log L into a directly follows-graph $G(L)$. After creating the directly follows graph the algorithm tries to partition the graph $G(L)$ into smaller subsets $\Sigma_1, ..., \Sigma_n$ by finding the dominant workflow operator between the partitions.
This dominant workflow operator can be expressed as a cut of the graph $\bigoplus(\Sigma_1, ..., \Sigma_n)$, which is an operator that describes how the subsets would have to be combined in order to reproduce the original graph, i.e. for which $\bigoplus(\Sigma_1, ..., \Sigma_n) = G(L)$ holds true.

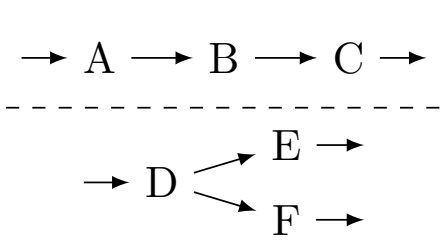
The 4 cut types that have been defined are $\rightarrow$,$\bigwedge$,$\times$ and $\circlearrowleft$, which are exactly the operators that a process tree uses for it's inner nodes.
If the dominant cut of a directly follows-graph is a sequence-cut $(\rightarrow (\Sigma_1, ..., \Sigma_n))$, then for any node $a \in \Sigma_i$ and $b \in \Sigma_j$ for $i < j$ there is a path from a to b but not the other way around.
For a xor-cut $(\times(\Sigma_1, ..., \Sigma_n))$ there is no edge between any two nodes $a \in \Sigma_i$ and $b \in \Sigma_j$ for which $i \neq j$.
In the case of $(\bigwedge(\Sigma_1, ..., \Sigma_n))$, an and-cut, there is a path between any two nodes $a \in \Sigma_i$ and $b \in \Sigma_j$ for which $i \neq j$ and for loop-cuts $(\circlearrowleft (\Sigma_1, ..., \Sigma_n))$ all start and ending-nodes are in Σ_1 while for all Σ_i with $i > 1$ incoming edges come from an ending node of Σ_1 and outgoing edges go to a starting node of Σ_1. 5 visualizes the four cuts.
Should the algorithm not be able to find a cut on a given graph, then it returns a flower model $\circlearrowleft (\tau, a_1, ..., a_n)$ for which $a_1, ..., a_n$ are the events that are part of $G(L)$.



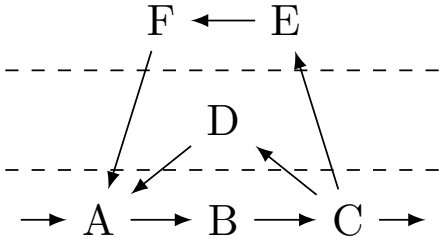
(a) A sequence cut. The arrows cross the cut lines only in one direction and an ordering sections is visible. The Σ are A, B, C, D and E



(b) An and cut. There is an edge between any nodes that are in different Σ. The Σ are A, B, C and E, F, G



(c) A xor cut. Each Σ is a single connected component of the graph. The Σ are A, B, C and D, E, F



(d) A loop cut. Σ_1 has all starting- and end-nodes while all other Σ have incoming edges from an ending node of Σ_1 and outgoing edges to a startinhow they appear ing node of Σ_1. The Σ are A, B, C,D and E, F.

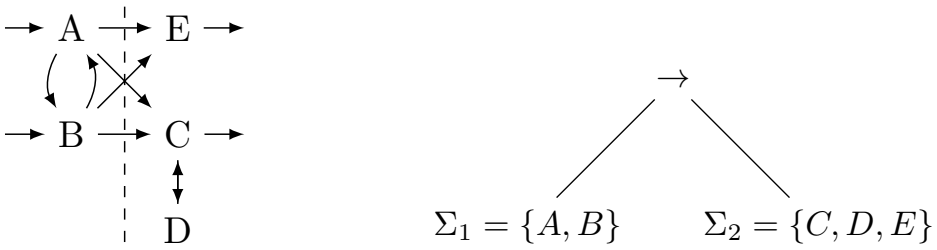
Figure 5: The different cut types how they appear in the df-graph. Dashed lines denote cuts.

With the chosen cut we can construct a node in the process tree. Assume we have G(L) and the chosen cut is $\circlearrowleft (\Sigma_1, ..., \Sigma_n)$, then $\circlearrowleft$ is the process tree operator and it's children are the results of the recursively analyzed $\Sigma_1, ..., \Sigma_n$.

See 6 for an example of how Inductive Miner transforms a directly follows graph into a process-tree.

Inductive Miner infrequent Inductive Miner considers all behavior of the log for the discovered model, even behavior that is infrequent or unwanted. In some cases, for example if this infrequent behavior is considered as noise, or if one wants just an overview of the main behavior of the log, then Inductive Miner is not a feasible option. To allow the usage of Inductive Miner in these use cases Inductive Miner infrequent has been proposed [24], a heuristics-based adaption of Inductive Miner that filters infrequent behavior and produces a reduced model, that is not perfectly fit anymore but more concise than a perfectly fit model. In each recursion step IMi first applies the normal steps that IM would do. If no cut could be applied and a flower model would be returned, it applies filter rules instead.

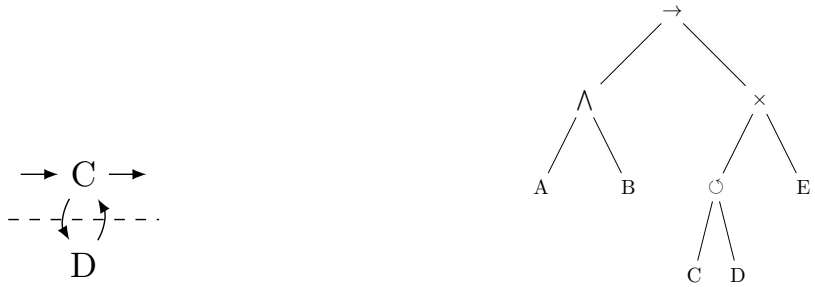
In order to do so IMi keeps track of how frequent traces and events are in the log



(a) The first detected cut is a sequence cut. The operator is added as the root node and the algorithm recurses on the partitions.

(b) The algorithm detects an and cut and recurses on $\Sigma_1\{A\}$ and $\Sigma_2\{B\}$. As the two partitions are base cases the event is returned and appended as a leaf node.

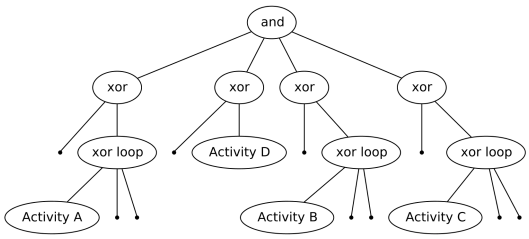
(c) In the next step an xor cut is detected. Upon recursion of Σ_2 the single event E is appended as a child.



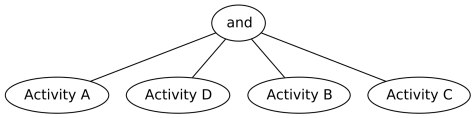
(d) Lastly a loop cut is detected. The partitions resemble base cases, so they are added as leaves. As there is no further partition that has to be analyzed the algorithm returns the process tree as the discovered model

Figure 6: Inductive Miner discovering a model of Log $L = \{< ABCDC >, < BAE >, < BAC >\}$

and uses a user-chosen frequency threshold k between 0 and 1. Upon cut detection infrequent edges whose frequencies are lower then k times the frequency of the most frequent edge in the graph are removed. Foror $\rightarrow$ cut detection a modified directly follows graph is used that amplifies the frequencies of frequent edges. This allows for a better distinction between wanted and unwanted behaviour.
Furthermore frequency-based filtering rules on base cases are introduced as the application of filter rules may introduce empty traces and repetitions of events in subsets. These rules try to select the most-fitting model for these distorted base cases.
Lastly after operator selection filter rules are applied on the partitioned subsets that remove the least operator infringing events from the log, to allow for a clean log for the next recursive step.
In evaluations IMi took longer than IM to finish, but was still faster than most other heuristics-based approaches.



(a) IM discovers a complex tree as each edge is considered.



(b) IMi is able to correctly discover the sequence.

Figure 7: Models as they are discovered by IM and IMi. The log represents a sequence of 4 events with added noise. IM creates a complex model while IMi discovers just the main behavior.

2.4 Incremental Process Discovery

Most Process Discovery algorithms base their analysis on the whole input log. Heuristics-based algorithms try to ignore infrequent bahaviour but still base their analysis on all traces. In a real-life scenario a log usually contains highly redundant information, either because some few processes are dominant in the log or because the underlying process only allows for so much behavior. Therefore it would be a good idea to not consider the whole log L but only a subset $L_{min} \subset L$ that contains the minimum information to discover the correct model. Stream-based discovery algorithms [25] have to take a similar approach, as the unbound size of log streams and the main memory boundary only allow for a window of traces, usually the newest ones, that can be considered for a representative model.

[26] proposes a statistical method that considers only a sublog of L based on the estimated probability p of new traces containing new information given a set of traces that have already been seen.
In the work the approach has been evaluated based on the k-tails algorithm [27] that uses k-sequences, sequences of consecutive events in the log.

The author defined the notion of δ-similarity as a criteria that tells when ënoughïnformation has been seen and when the analysis can be stopped. A log L is δ-similar if the total probability of k-sequences to be observed in the next trace that would be seen in the next trace is smaller than δ.
In order to precisely estimate δ one would have to know in advance how many k-sequences are remaining in the following traces at any given time. In a real scenario this is not reasonable as one would need to analyze the log in advance which destroys the purpose of the algorithm, which is not to consider everything for the discovery. One can not know the exact value of δ but one can try to estimate it using interval estimation methods by modeling the process of iteratively selecting a trace and checking whether it contains new information or not as a series of binomial experiments. With this view one can calculate the number of consecutive experiments that have to fail in order to safely conclude that no new information is expected in the remaining log.

By simulating a series of iterations in which one checks whether new information is seen or not the propbability p can be bound inside a binomial proportions interval with upper and lower bound $p_{lowerbound}$ and $p_{upperbound}$.
Let p be the proportion of succesful trials over N random trials and let z be the $1-0,5\cdot\alpha$ percentile of a normal distribution for a given α, then the binomial proportions interval

$$[\frac{1}{1+\frac{z^2}{N}} \cdot (p + \frac{z^2}{2N} - \sqrt{\frac{p\cdot(1-p)}{N} + \frac{z}{4N^2}}), \frac{1}{1+\frac{z^2}{N}} \cdot (p + \frac{z^2}{2N} + \sqrt{\frac{p\cdot(1-p)}{N} + \frac{z}{4N^2}})]$$

contains the real value p with a probability of $(1-\alpha)$. The used formula is Wilson's interval. Alternatively the paper cited Agresti-Coull interval or Jeffreys prior intervall as superior interval methods for different p [28].

Now one simply have to calculate how many failing trials N are needed until $p_{upperbound} < \delta$. Upon finding an N one knows that consecutive traces without new information are needed to conclude with a $(1-\alpha)$ confidence that the log is is δ-similar. As one wants to simulate all tests to fail $1-p$ will be set to 0, thus reducing the formula to

$$\frac{1}{1+\frac{z^2}{N}} \cdot (\frac{z^2}{2\cdot N} + \sqrt{\frac{z}{4\cdot N^2}})$$

For $\alpha = 0.01$ and $\delta = 0.05$ this would give $N = 128$ using the above formula. Thus after seeing 128 traces that do not contain new information one can stop the analysis knowing with a confidence of 0.99 that the probability of seeing new information is

less than δ.
One can calculate N before applying the algorithm . Then while analyzing traces one just has to keep track of the amount of consecutive traces without new information and stop the analysis after N consecute traces have been seen that yielded no new information.

3 The statistical Inductive Miner

3.1 Idea

Assume a user wants to get a general overwiev of the behavior in a log. The user also wants to measure the average cycle time of the model.
For this use case the Inductive Miner infrequent is an ideal choice.

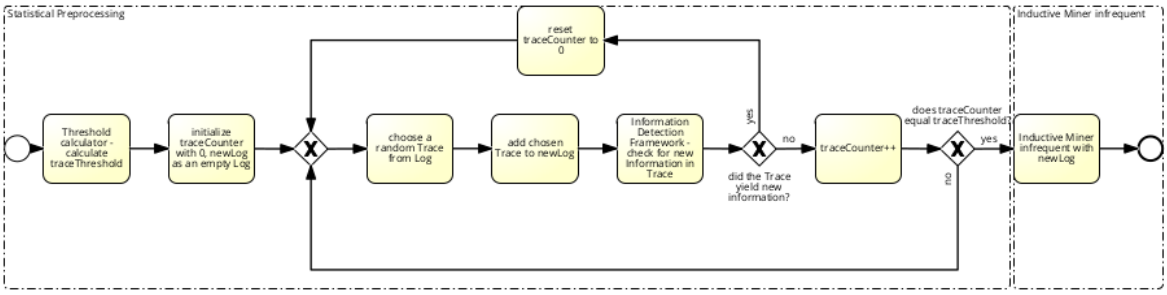
In [26] it was shown that the statisticallLog analysis is independent on the discovery algorithm it is used for, one only needs to know whether new information has been gained through a trace or not. Consequently one should be able to adapt the information detection so that it is usebale for a specific algorithm, then mine the reduced sublog and still discover a fitting model. If the log processing considers all trace properties that could change the underlying model representation, then the statistical pre-processing step should produce a sublog, that contains all relevant information for the algorithm.

If IMi is executed with a Log that has been pre-processed with criteria that are relevant for Inductive Miner, the runtime and memory consumption of the algorithm could be reduced and in turn enable the analysis of bigger data.

3.2 Algorithm

A statistical pre-processing step is added to IM, in which N is calculated, the number of consecutive traces without new information that have to be read to end the trace collection, then the algorithmg chooses random traces repeadedly, checks for new information in the chosen trace and increments a counter if no new information has been seen, or resets it to 0 otherwise. If the counter has reached N, the traces seen up to tis point form the new log for which IMi will discover a model. See Figure 8 for a visualization of the algorithmic setup.
As IM is already implemented in ProM [2] a statistical pre-processing framework is added that takes a log L and that returns a log $L_{new} \subseteq L$ and calls the already implemented IM with L_{new}. This will be slower than using the information extraction step that IM will do when it is called but allows to completely ignore the implementation details of IM.

Figure 8: The Algorithm



For the trace calculation any of the binomial proportion interval estimators is used. N is incremented repeadedly until the estimated upper bound of p is lower than δ.

3.3 The Information Detection Framework of sIMi

3.3.1 Defining a general Information Detection Framework

The general idea presented in [26] is to choose a random trace from the log and analyse whether this log contains any new information that change the inner representation of the log. If so, then the experimental chain starts again, otherwise it continues with the next randomly chosen trace until the threshold has been reached the complete log has been traversed.
For k-Tails the inner representation is the set of n-grams or k-tails seen so far. If a trace adds a new n-gram then the representation changes.

Having this general idea in mind one can formalize a general algorithm-independent framework F that detects new information in traces as follows:
Given a set of informations aspects $I_1, ..., I_n$, then each aspect has a corresponding information base, denoted as $B_{I_1}, ..., B_{I_n}$ that holds all information extracted from traces so far. Furthermore for every I_j there exists a function $DETECT_{I_j}(B_{I_j}, t)$, that, based on the current trace t and B_{I_j}, returns whether the traces yields new information for I_j or not.
Thus for information bases $B_{I_1}, ..., B_{I_n}$ and a trace t the function

$$G = DETECT_{I_1}(B_{I_1}, t) \vee DETECT_{I_2}(B_{I_2}, t) \vee ... \vee DETECT_{I_n}(B_{I_n}, t)$$

calcaulates whether t contains new information or not.
In the case of k-tails the framework $F_{k-tails}$ would contain

I_{n-gram}	n-grams seen in the traces
$B_{I_{n-gram}}$	a data structure, that contains the seen n-grams
$DETECT_{I_{n-gram}}(B_{I_{n-gram}}, t)$	the function that for a given trace t returns whether t

contains a n-gram that is not yet present in B_{I_1} and updates B_{I_1} with new n-grams

3.3.2 Notable properties of the Framework

Just from this setup one can conclude several properties of the framework:

Trace order dependent information detection Let $t_current$ denote the current trace, then $t_{current \pm i}$ denotes the trace that has been analyzed i steps prior or after $t_current$.
Whether F_{I_j} yields new information or not is highly dependent on the state of B_{I_j} while analyzing $t_current$. Depending on the order of chosen traces a trace could yield new information or not, e.g. if $< ABC >, < ABE >$ have been analyzed prior to the trace $< AB >$, then $< AB >$ does not contain new information, whereas in an other scenario it would contain new information.

The analysis of the first trace will always yield new information.

Extensibility of the framework The framework does give constraints on neither the type of information that will be analyzed, nor the amount of criteria that will be analyzed.
As such a user may add or remove analytical criteria as he sees fit. Removing an aspect will not affect the outcome of other aspects if they do not depend on the information base of the removed aspect.

Ordering of aspects As the $DETECT$-functions are only dependent on B_{I_j} they can be executed in any order.

Aspect dependent runtime Some information aspects see new information more frequently than others, some will also get more information out of a trace than others. Assuming that all but one aspect won't get any new information out of the log, but the remaining aspect still gets new information through the analysis of following traces, then this aspect prolongs the log preprocessing single handedly. The preprocessingwill only stop, when this remaining aspect does not gain new information anymore. One can say that this information aspect determines the total runtime of the preprocessing framework.

Aspect dependent reliability For each trace each I_j is analyzed.
The more I_j are present, the higher is the chance that $t_current$ will yield new information. Assuming that the order of $t_current, t_{current+1}, ..., t_{current+1}$, is known in advance, then there could be one ordering of traces for which each consecutive analysis yields no new information and the whole preprocessing chain comes to an end too early.

Adding more criteria decreases the chance of this happening and therefore increases the reliability of the framework.

Aspect-based runtime and memory The total runtime and memory of analyzing $t_current$ is $\sum_{j=1}^{n} O(B_{I_j})$ and therefore $\max(B_{I_j})$ for $j < n$.

3.4 Analytical Aspects for sIM

3.4.1 Model correctness

Simulating the directly follows graph during pre processing In the case of Inductive Miner a trace t yields new information if the directly-follows graph changed after reading in t. To ensure a correct model with the framework it should therefore be feasible to represent the directly follows graph through all aspects that could change the graph itself. The chosen aspects correspond to the I_j in our framework. By ensuring that we can recronstruct the graph by just using all B_{I_j} we can be sure that all relevant information is captured through our aspects. $DETECT_{I_j}$ detects a change in the graph by detecting a change in B_{I_j}, so all it has to do is to check, whether something new has been added to B_{I_j} by t or not.

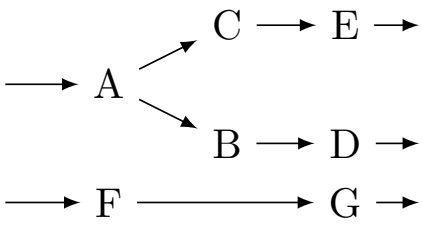
Using changeable graph properties as analysis aspects A directly follows graph is represented through a set of vertices, edges, start- and ending nodes. Given a trace t it may add

- a single starting node
- a single ending node
- multiple new vertices
- multiple new edges

to the graph. These graph aspects are the I_j for the sIM-framework. 9 shows that these criteria are able to detect a change in the knowledge base that IM uses. 10 further shows that it is possible to rebuild the directly follows graph using just these informations.

3.4.2 Cycle Time Correctness

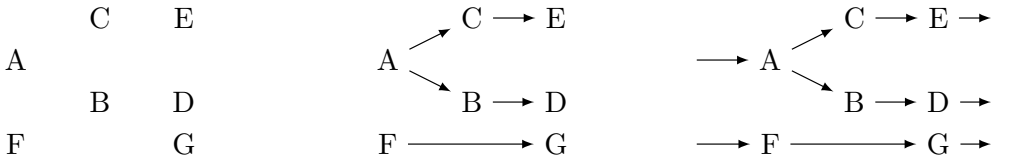
For model criteria a trace t yields new information if after reading the trace the corresponding knowledge base has changed.
For cycle times almost every trace will change the knowledge base, because every trace contains a cycle time and a small difference from the estimated cycle time will change the value of the new updated cycle time. Every cycle time but the one that is equals to the cycle time measured so far would yield new information. If this strict change would be used as $DETECT_{cycletime}$ then most likely all traces would have to be considered for the pre-processing step, making the whole framework too restrictive, making it



(a) A directly follows graph made of the traces <ABD> and <ACE>

(b) The same graph after <FG> has been added

Figure 9: Adding the trace <FG> to the initial graph adds a new starting- and end node, two new events and one new edge to the graph



(a) Adding nodes

(b) Adding edges

(c) Adding starting- and end nodes

Figure 10: Reconstructing the directly follows graph from 9b by using the information seen in the traces.

slower than IMi.
If one regards the calculated cycle times after each trace as a sequence of values, then onw can see that the sequence converges towards the correct average cycle time.
Let t_n, t_m denote the n'th and m'th trace that will be analyzed, $|t|$ the index of trace t in the sequence, and $cycletime_n$,$cycletime_m$ the estimated cycle time at t_n and t_m. Furthermore $d(cycletime_n, cycletime_m) := |cycletime_n - cycletime_m|$ is the function that calculates the distance between two cycle times. As this sequence converges and the space of possible cycle times is a complete metric space, it follows that the sequence is a Cauchy-sequence and therefore

$$\forall \epsilon \geq 0 \exists 0 < N < \max(|t|) \forall N \leq t_n, t_m < \max(|t|) : d(cycletime_n, cycletime_m) < \epsilon$$

holds true.
No matter what sequence of cycle times one has, for a given analysis one can choose an arbitrary ϵ and know that after an unknown number of traces n $d(cycletime_n, cycletime_{n+1})$ will be smaller than ϵ for every $m \geq n$.
Knowing this the correct cycle time can be approximated by choosing an ϵ and checking whether $d(cycletime_{n-1}, cycletime_n) \geq \epsilon$ is true or not. If it is true then that trace is treated as if it yields new information for $I_{cycletime}$.

Model based and Event based cycle time approximation There are multiple possibilities to approximate the average cycle time of a log. For a trace t one solution is to calculate the time of the trace by either getting it from it's information or calculating it manually by subtracting the beginning time of the first event from the completion time of the last event, updating the average cycle time of the model with the so calculated trace time and testing for $d(cycletime_{n-1}, cycletime_n) \geq \epsilon$.
The other possibility is to calculate the completion time of each event in t, updating the average cycle time for each event in the trace and then for each of them checking for the outcome of $d(cycletime_{n-1}, cycletime_n) \geq \epsilon$.
Both variants base their calculations on the timestamps present in the log, so two possible functions $DETECT_{cycletime_{model}}$ and $DETECT_{cycletime_{event}}$ exist, that could be used for $I_{cycletime}$. The second option needs more computation as it may has to do multiple calculations, first option needs to update a single value.

Finding an appropriate ϵ The outcome of $DETECT_{cycletime}$ is dependent on the chosen ϵ. It could be chosen so that the event time is approximated in the least traces as possible by setting it high. This would lower the amount of traces that have to be read for the experiment but results in an inaccurate estimated cycle time. If it is chosen to be small, then this increases the number of traces that have to be read during preprocessing but returns a more accurate result. Furthermore choosing an ϵ that is too small will result in almost every trace yielding new information. In such a case the whole log will be pre-processed and no speed gain is possible. On the other side, if ϵ is choosen too high, then almost no trace will yield new information, which means that the cycle time analysis does not provide meaningful results and is negligible. In this case one could remove the cycle time analysis as it's only accomplishment would be to slow down the general speed of the algorithm. The chosen ϵ is highly dependent on the distribution of times in the log. If all events have the same duration, then an event based analysis would finish after if has seen all events once. If all traces have the same duration, then a model based analysis would finish after one trace. How randomly the durations are determines what an appropriate ϵ may be. Therefore a user has to choose a fitting ϵ for each log based on the knowledge he or her has about the time distribution.

Even though the model based cycle time calculation does more work a user may only be able to quantify an appropriate ϵ for the events in the log, so it may be a good idea to give both options.

3.4.3 Summarizing all Aspects

Knowing that we want to guarantee a correct model and a approximately correct average cycle time following components for F_{sIMi} can be defined:

I_{event} the events seen in the log

I_{edges} the edges seen in the log

I_{start} the starting nodes seen in the log

I_{ending} the end nodes seen in the log

$I_{cycletime_{model}}$ the approximated cycle time based on trace times

$I_{cycletime_{event}}$ the approximated cycle time based on event times

3.5 Infrequent Behaviour

During the preprocessing the algorithm chooses a random trace for the next analysis. Assuming that trace t is infrequent in the log, then there are two possible outcomes:

2. t is not chosen then t is not represented in the graph

1. t is chosen then t is represented in the directly follows graph but is filtered out by the filtering rules of IMi

In both cases t does not have an impact on the mined model. Traces that appear frequently have a higher chance of being chosen than these that apper infrequently, so sIM can be regarded as a further filter step of IMi and as thus the addition of a statistical preprocessing step to the log should retain the relative frequency of unique traces.

3.6 Input parameters

Having defined all of the analysis criteria te input parameters of the algorithm are:

float α - the confidence of the preprocessing

float δ - the probability that the estimated p has to reach

float ϵ - the epsilon used for the cycle time analysis

boolean cycle time variant - 1 if cycle time will be measured on a model basis, 0 if on an event basis

4 Evaluating the Algorithm

4.1 Performance Analysis

4.1.1 Used Logs

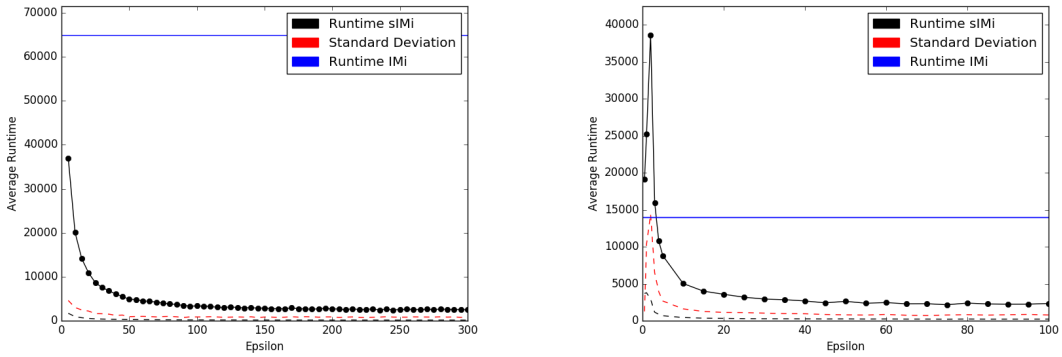
In order to measure how well the constructed algorithm fares compared to IMi measurements with 2 real life Logs have been performed. All of them were part of the BPI Challenge conducted by the International Workshop on Business Process Intelligence each year [29]. These logs are provided freely for anyone to use as they see fit and will be denoted as follows:

BPI2012 - a log by a dutch financial institute. It contains roughly 13000 traces and is 74.1MB big

BPI2014 - a log containing incident activity records of a bank of size 39.2MB

For each log runtime, memory consumption the number of traces that have been read and the fitness [16] with regards to the original model have been measured. All measurements have been conducted with varying ϵ on model and event view. α has been set to 0.01 and δ to 0.99. Wilson interval is used as the interval estimator. This sets the number of traces that have to be read to 127.
The chosen ϵ differs for all logs because approriate values for ϵ are dependant on the distribuion of trace and event times in the log.

4.1.2 Runtime

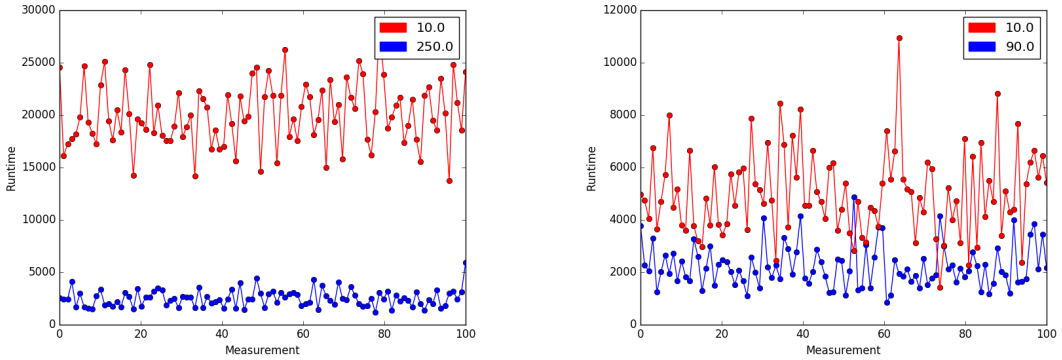


(a) BPI2012 (b) BPI2014

Figure 11: Average totalrRuntime (black continous) and average preprocessing time (black dotted) with varying ϵ, model-based

As shown in 11 sIMi is notably faster for most *epsilon*, for BPI2012 it produces a model 20 times faster than IMi for big ϵ. For BPI2014 the model can be produced up to 5 times as fast with preprocseeing being only a fraction of the total time. The impact on the chosen ϵ on the total runtime is very immense for small ϵ as then the cycle time analysis becomes the most restrictive information aspect during preprocessing.
For a small window of ϵ (for BPI2014 approximately $0.5m < \epsilon < 2$m) the runtime is notably higher than the runtime of IMi though. This is because the logs get partially ordered when they are importet into ProM, but the algorithm builds the new log randomly. As the implementation of IMi benefits from an ordered log the traces of the new log are ordered in the same way they are in the original log, so that the total runtime in cases where a large portion of the log gets seen is higher than IMi. Therefore we can say that sIMi is faster than IMi apart from cases where ϵ is very small.
One can clearly see that an ϵ that is choosen too high will not have any impact on the runtime. This is due to the cycle time analysis finishing much faster than at least

one of the model based information aspects, which then determines the runtime. For BPI2012 this seems to be the case for $epsilon > 150m$, for BPI2014 $\epsilon > 40m$.
Because of the higher amount of traces that have to be read for small ϵ and because the cycle time analysis is the only information aspect that prolongs the preprocessing in these cases the fluctuation of runtimes grows bigger, which can be seen in the standard deviation of the estimated runtimes that rises sharply for small ϵ. This leads to the assumption that the pre processing becomes unstable in cases where the cycle time is the most restrictive information aspect and where not all traces are seen.
If one compares runtimes for different ϵ like in 12 then one can see that the runtimes gets indeed more unstable for small ϵ, so differing runtimes are expected for small ϵ.



(a) BPI2012 (b) BPI2014

Figure 12: Comparison 100 measured runtimes for BPI2012 and BPI2014 using two different ϵ, model-based

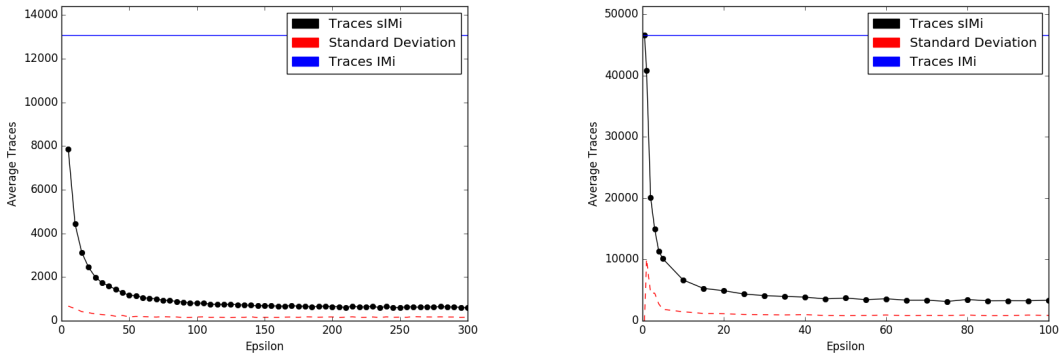
4.1.3 Traces

As the runtime is completely dependent on the number of traces that are seen in a run of the algorithm we can assume that the number of traces the algorithm has to read develops roughly the same for verying ϵ, which means that

- for small ϵ the number of traces rises sharply until the total log size is reached
- for big ϵ the number of traces will stay roughly the same

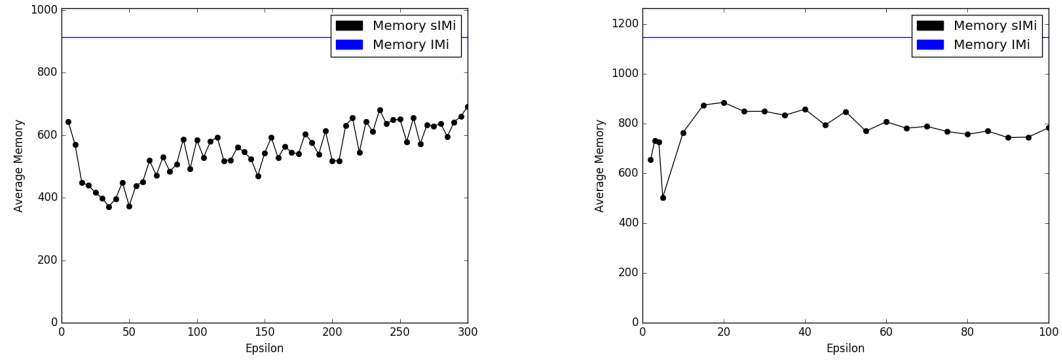
4.1.4 Memory

Even though runtime and traces are affected by the chosen ϵ the memory seems not to be influenced by ϵ. 14 shows that for both BPI2012 and BPI2014 memory consumption stayed roughly the same no matter the chosen ϵ. Due to a rather naive approach at measuring the memory consumption of sIMi the estimated memory differs by a lot for all measurements, but stays below IMi during all times.



(a) BPI2012 (b) BPI2014

Figure 13: Average number of traces per run with varying ϵ, model-based



(a) BPI2012 (b) BPI2014

Figure 14: Average memory consumption of sIMi with varying ϵ, model-based

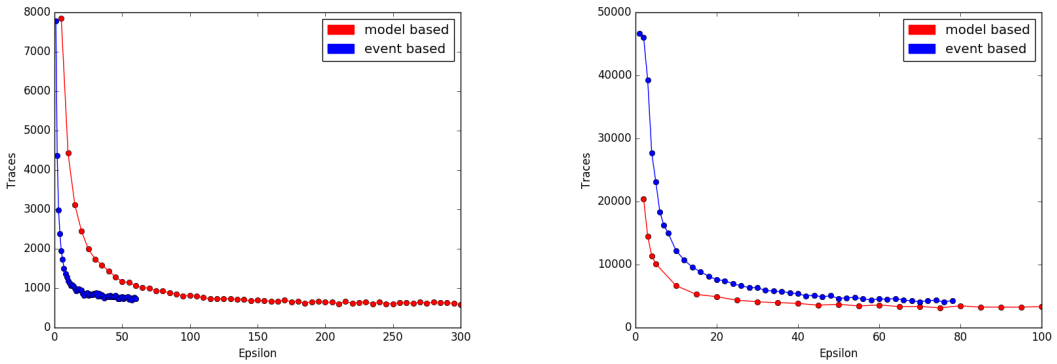
The question why memory is unaffected by ϵ can be answered easily. The used memory is dependent on the way the information base for an information aspect is stored. The chosen ϵ has an effect on the outcome of $DETECT_{cycletime}$, but The function will update $B_{I_{cycletime}}$ for every trace no matter what the value of ϵ is. In fact all DETECT-functions of the framework update their information base independent of ϵ. We can make a distinction between the change of memory over time for model based infromation aspects and for $B_{I_{cycletime}}$:

Cycle Time Information Aspect For $B_{I_{cycletime}}$ two values have to be stored, the amount of traces that affected the cycle time so far, and the average cycle time calculated so far. With these two values the next cycle time can be calculated. So no matter how much new information arises (which is affected by ϵ) the memory needed for $B_{I_{cycletime}}$ is either one cycle time object (for a model-based estimation) or one cycle

time object per event in the log (for an event based estimation).

Model based Information Aspects Adding to this constant sized pool of data come the information bases for the model based information aspects. For them multiple values have to be stored (i.e events, edges), so they change in size during the log analysis.
They are either more restrictive than the ϵ-based $I_{cycletime}$ or less restrictive. For the first case this means that the completion of the model based information aspects determines when the preprocessing ends, therefore almost all model based information has been seen upon completion, which means that no matter how high ϵ is, the average memory of the model based information aspects will be roughly the same size. On the other hand when ϵ is very small and therefore $I_{cycletime}$ is the most restrictive information aspect, then no new information gets added to the model based information aspects in the end, which means they are complete and therefore have roughly the same saze as in the first case.

4.1.5 Comparing model based and event based cycle time approximation

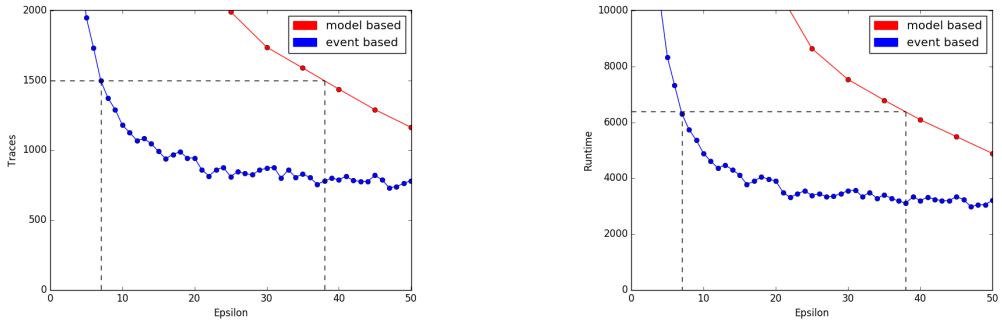


(a) BPI2012 (b) BPI2014

Figure 15: Comparison of model based and event based cycle time analysis and it's effect on the total traces with varying ϵ

Despite the question what cycle time approximation method a user may prefer it is interesting to see whether one method has an advantage over the other on an performance perspective.
In 15 one can see that one can not assume that one method has a smaller or bigger interval of appropriate ϵ, instead one sees that this is also dependent on the distribution of cycle times in te log, which could lead to any of the both methods to have a bigger assortment of viable ϵ. In 15a one sees that for BPI2012 the cycle times of events do not deviate by a lot. The trace times on the other hand deviate more, which may be a

indicator that the number of events in the traces differs a lot and that the log contains many differnt sized traces.
In contrast to BPI2014 the analysis of 15b yields that in BPI2012 event times are distributed more randomly, as shown through the impairment on higer ϵ and that traces have more similar completion times.
If one tries to compare the runtime of both analysis techniques then one will see that



(a) For a model based anylysis ϵ of 38 and for a event based analysis ϵ of 7 result in roughly 1500 traces that are seen during pre processing

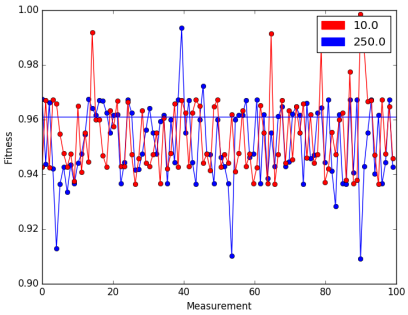
(b) For both model based and event based analysis the estimated epsilon result in a 6 second long preprocessing, even though event based analysis has to do more computations

Figure 16: Comparing the average runtimes of model based and event based cycle time approximation for ϵ that result in similar trace numbers for both analysis types for BPI2012.

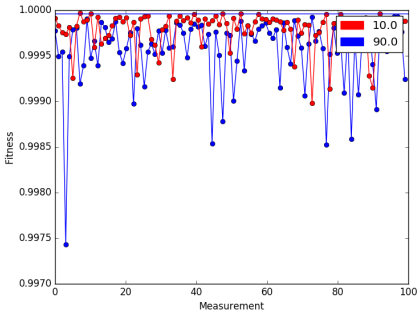
for ϵ where both see a similar amount of traces the runtimes are also similar, which contradicts the statement that event based analysis has to do more computations per trace.
The algorithm incorporates model based analysis, that has to analyse each event anyway. So no matter wat cycle time approximation is used, in both cases the algorithm has to analyse each event in each seen trace anyway because of the model criteria, so the drawback of event based analysis is actually irrelevant for the algorithm and none of both techniques has a visible computational advantage over the other.
As a computational difference is non existent both techniques should be presented to the user, who should then decide which method best fits his needs.

4.1.6 Fitness

sIMi does not loose a lot of fitness compared to IMi which seems to be due to the fact that sIMi is more likely to leave out the traces that are infrequent while all seen infrequent traces will be filtered out anyway, so in average both would yield the same result and a similar fitness score. Due to the random nature of picking traces there are also some cases where sIMi reaches a higher fitness score.



(a) Fitness comparison for a model based cycle time approximation for ϵ of 10 and 250, BPI 2012



(b) Fitness comparison for a model based cycle time approximation for ϵ of 10 and 90, BPI 2014

Figure 17: Fitness Measurements for BPI2012 and 2014

Even though the algorithm has similar fitness values as IMi the model differ slightly for multiple runs of the algorithm, which is again due to the random trace picking nature. In the worst case this could lead to a user being confused, as he would normally assume that the algorithm creates the same model each run. See 18 for an example. This should be prevented, e.g. by changing the algorithm to pseudo randomly pick traces by using seeds and forcing the algorithm to choose the same traces again for the same log by using the same seed again.

4.2 Analyzing a Log's Influence on Performance

4.2.1 Using Block Structures to determine Best Case Performance

So far analysis has been done on how well sIMi competes against IMi, but the logfiles influence on the runtime is still unknown.
As mentioned multiple times already, the runtime is dependent on the amount of new information seen in traces, which on a strict model basis is dependent on the number of events, starting-, end-events and edges that would be present in the final directly follows graph. If log a contains double the amount of edges or vertices than log b, one can easily assume that the analysis of log a will take longer, as the end of the preprocessing will be prolonged roughly twice as often as for log b.

[10] describes how all process tree operators are represented through certain edge combinations in the directly follows graph. Each of these operators represents a block in the block structured end model. Therefore one can try to analyze how much information is present in a certain type of block and try to conclude what

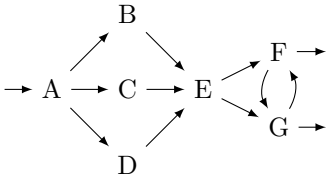


Figure 19: A Directly follows graph of an imaginary log containing 7 vertices, 9 edges, 1 starting node and 2 end nodes. The traces <ABEFG>,<ACEFG>,<ADEGF> contains all these informations and are therefore enough to represent this graph.

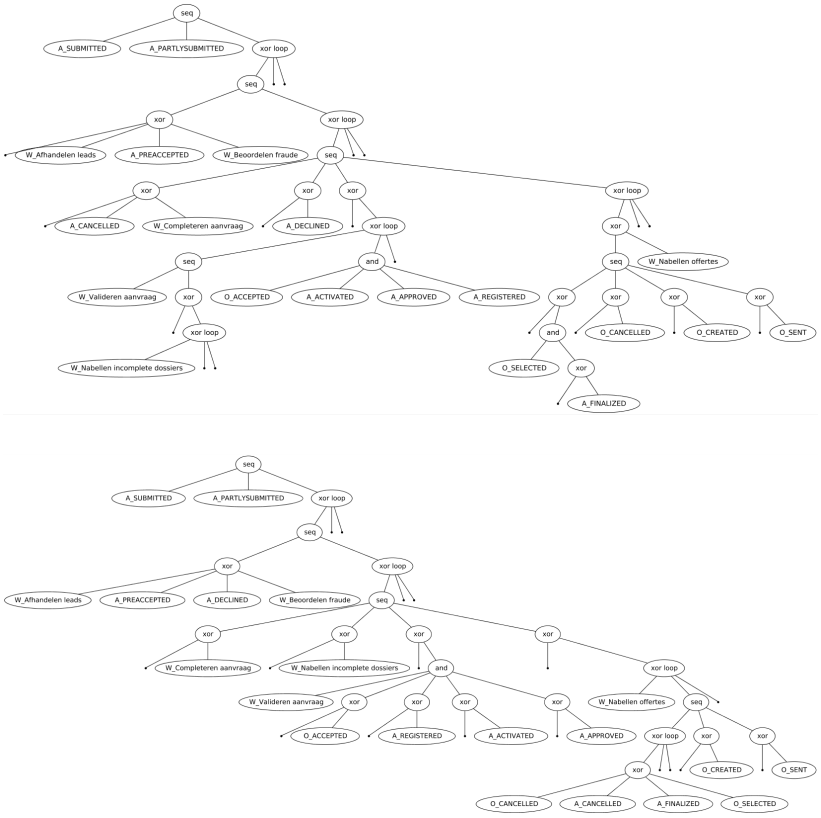


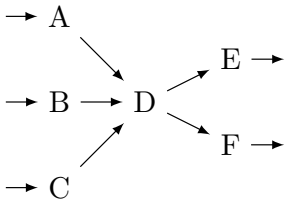
Figure 18: Two with sIMi mined Process Trees using 0.2 as the filter threshold, BPI2012. One can clearly see that the two trees are structurally different

the minimal amount of traces is that can describe the behaviour that is present in the block. Because behavior can be represented through edges that a trace takes through the graph, one just needs to find out the minimal number of traces that incorporate all edges of the block like shown in 19. As each block is described through the interactions between it's subblocks a prediction or at least approximation of the minimal amount of traces that IMi has to see can be done.

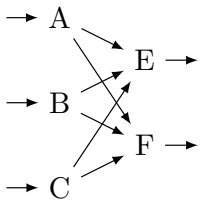
Single Events This is the base case as a single event does not contain other block stuctures. A single event takes at least one trace.

Sequences Sequences are concatenations of possible paths through all subblocks of the sequence block. Each path in a block begins at a start node and ends at an ending

node. As some subblocks have the need for more paths than others and as all blocks have to be traversed in a sequence, the block with the highest trace requirement determines how many traces are required in a block as this trace number could incorporate all other blocks minimal trace number. In 19 this has been used implicitly. The graph is a sequence of an event, a 3-event xor-cut, another event and a 2-event and-cut. In this scenario the xor-cut has the highest trace requirement, as 3 traces are needed to see each of B,C and E and the other possible paths through the other blocks can be covered in these 3 different traces.
For two blocks, where block A preceeds block B, each ending node of block A has an edge to each starting node of block B. As a path through a block ends with and starts with exactly one node of the appropriate set the algorithm needs at least as many traces as the product of ending nodes of block A times the starting nodes of block B. Therefore either the most requiring subblock or the most requiring node product between sequential blocks depicts the total minimal amount of traces needed for the sequence block. 20 visualizes these two possibilities.



(a) A sequence of a 3-event-xor, a single event and a 2-event-xor. Here the most requiring factor for the block is the first xor-block, thus requiring at least 3 traces to discover all information(i.e <ADE>,<BDF>,<CDE>)



(b) Two sequential xor-blocks. Here the 6 edges between the two blocks necessitate at least 6 traces to see all information in the graph (i.e <AE>,<AF>,<BE>,<BF>,<CE>,<CF>)

Figure 20: Two directly follows graphs that are structured as sequences with varying minimal trace requirements

Loops Loops are a special case. In theory no matter how interleaving loops are and no matter what subblocks they contain and how many paths they require in order to detect all information, a single trace can take multiple paths through the loop and so all information inside the loop can be detected in one trace. The more interleaved the loop is the more unlikely it becomes that a trace contains all information, but the possibility always persists. As seen exemplary in 21 1 can be assumed as the minimal amount of traces.

Xor-Blocks A xor-block has multiple paths and a trace has to take exactly one path. Therefore the minimal trace requirement of a xor-block is the sum of the requirements of all it's subblocks.

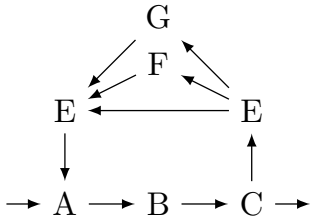


Figure 21: A directly follows graph depicting a loop that contains a second loop with a xor-block in it's optional part. Even though 2 traces are needed to see all information for the xor-block in isloation, the trace <ABCDEABCDFABCDGABC> conatins all information seen in the graph.

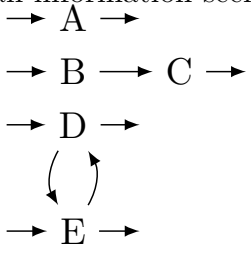


Figure 22: A directly foloows graph structured as a xor-block. For the first two paths one trace is needed, for the third path 2 traces are needed, so the traces <A>,<BC>,<DE> and <ED> include all information.

And-Blocks In and-blocks there exists a path between all nodes of different subblocks in the graph. As the possible edges in such a graph change drastically with the number of events in subblocks and depending on what type of subblocks are present, it is very difficult to give a correct estimation on the minimal number of traces that contain all information. The more events an and-block contains the more informaton is needed to detect it completely (as for $|V|$ vertices in the graph up to $|V|$ edges could be added we addif a new event is added), so one can assume the minimal required traces to grow factorial with the number of events in the and-block.

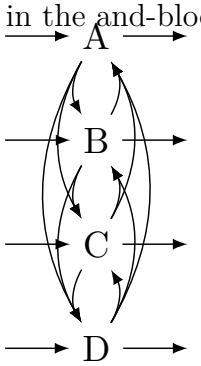


Figure 23: A directly follows graph containing a simple and-block. The traces <ACDB>,<BADC>,<CABD>,<DCBA>,<BCDA> can represent all information.

Having estimated the minimal number of traces a block needs to see in order to see all information one can conclude that and-structures in a log influence the runtime very badly the bigger they are, while sequences of complex structures and xor-blocks containing other complex stuctures add notably to the runtime.

4.2.2 Noise and Cycle Times Deviation

Noise Even though noise will be filtered out in the end and does not affect the end model, it prolongs the pre processing chain should it be seen in the log. The more noise a log contains the longer the runtime will be, as more traces have to be seen until the pre processing stops. 24 depicts this assumption.

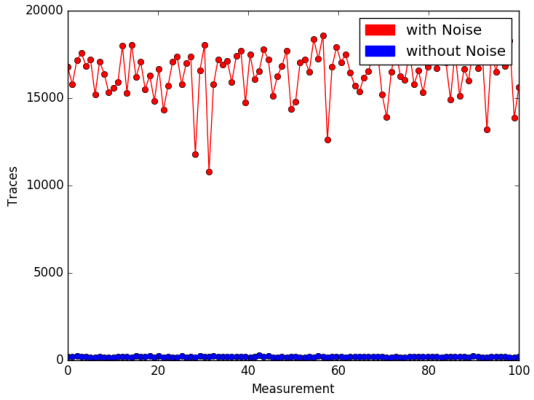


Figure 24: Comparison of read traces for a randomly generated log once with and once without added noise

Cycle Time Deviation Smaller ϵ increase the runtime of the algorithm. Logs for which the completion time of events or traces deviate by a lot will result in a longer runtime than those where the average deviation is smaller. This is depicted in 25, as the cycle time approximation takes longer in tese cases.

5 Conclusion

In this work I presented an alghorithmic framework that added a preprocessing step to the Inductive Miner infrequent algorithm that uses statistical log analysis in order to reduce the number of traces that IMi has to use, effectively reducing the input size. The preprocessing considered model correctness and an approximative cycle time correctness as the defining factors to conclude the size of the new sublog. I showed that the so created statistical Inductive Miner infrequent - sIMi, is faster and more memory efficient for a large amount of input data, while achieving similar fitness values as IMi.

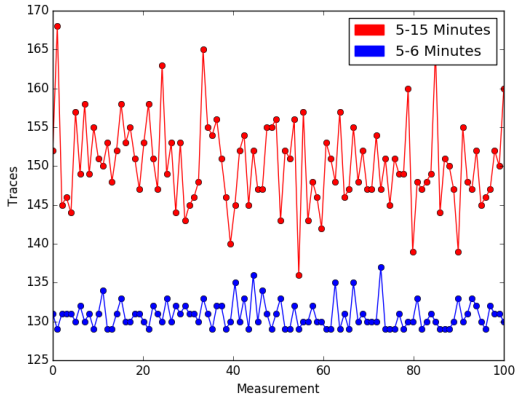


Figure 25: Comparison of read traces for a randomly generated log with differing deviations of cycle times. Measurements have been taken with ϵ of 30s and all deviations were uniformly distributed between the time limits.

Only for very strict cycle time approximation does the algorithm take longer than IMi, therefore the algorithm has an advantage against IMi in real life scenarios and is more scalable than aforementioned algorithm if the user is able to specify a good threshold for the cycle time approximation. The only redeeming factor is that sIMi produces differing models for same inputs even if the qualitative dimensions do not suffer.
Furthermore I showed that a too strict cycle time approximation is not beneficial to the algorithms runtime. Logs that contain big and-structures have the higest runtimes, followed by sequences of complex workflow constructs and xor-blocks containing complex event structures. Noise or high cycle time deviations also increase the runtime.

As an outlook, further work should be done on guaranteeing that the same model is discovered for same inputs. To achieve this the random number generator would have to be initialized in such a way that it uses the same seed for the same log. Further work has to be done on improving the runtime upon strict cycle time approximation.

6 Appendix

6.1 Using statistical Inductive Miner infrequent

The code of sIMi can be found at
`https://github.com/Martin-Bauer/StatisticalInductiveMinerInfrequent`.

For further instructions see the readme-file in the link.

Literatur

[1] "Process mining." `http://www.processmining.org/research/start`. Accessed: 2016-03-02.

[2] B. van Dongen, A. de Medeiros, H. Verbeek, A. Weijters, and W. van der Aalst, *The ProM Framework: A New Era in Process Mining Tool Support*. Springer-Verlag, 2005.

[3] "Apromore." `http://apromore.org/`. Accessed: 2016-03-02.

[4] W. M. van der Aalst et al., "Process mining manifesto," in *Business Process Management Workshops 2011, Lecture Notes in Business Information Processing*, vol. 99, Springer-Verlag, 2011.

[5] C. Bird, T. Menzies, and T. Zimmermann, *Scalable Parallelization of Specification Mining Using Distributed Computing*. Elsevier, 2015.

[6] J. Leemans, D. Fahland, and W. M. van der Aalst, "Scalable process discovery with guarantees," in *Lecture Notes in Business Information Processing*, vol. 214, Springer-Verlag, 2015.

[7] W. M. van der Aalst, "Data scientist: The engineer of the future," in *Proceedings of the I-ESA Conferences*, vol. 7, Springer-Verlag, 2014.

[8] T. Murata, "Petri nets: Properties, analysis and applications," in *Proceedings of the IEEE*, vol. 77, IEEE, 2002.

[9] W. M. van der Aalst, "The application of petri nets to workflow management," 1998.

[10] W. M. van der Aalst, S. J. Leemans, and D. Fahland, "Discovering block-structured process models from event logs - a constructive approach," 2014.

[11] W. M. van der Aalst, A. Weijters, and L. Maruster, "Workflow mining: discovering process models from event logs," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, IEEE, 2004.

[12] W. M. van der Aalst, J. Buijs, and B. van Dongen, *Towards Improving the Representational Bias of Process Mining*. Springer-Verlag, 2012.

[13] N. Russel, A. ter Hofstede, W. van der Aalst, and N. Mulyar, "Workflow control-flow patterns: A revised view,"

[14] C. Buijs, F. van Dongen, and W. M. van der Aalst, "On the role of fitness, precision, generalization and simplicity in process discovery," in *On the Move to Meaningful Internet Systems: OTM 2012*, Springer-Verlag, 2012.

[15] J. Leemans, D. Fahland, W. M. van der Aalst, A. Adriansyah, and B. van Dongen, "Replaying history on process models for conformance checking and performance analysis," in *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, John Wiley & Sons, Inc., 2012.

[16] J. Leemans, D. Fahland, and W. M. van der Aalst, "Scalable process discovery and conformance checking," in *Software and Systems Modeling*, Springer-Verlag, 2016.

[17] L. Wen, "Process mining: Overview and outlook of petri net discovery algorithms,"

[18] A. J. M. M. Weijters and A. K. A. D. Medeiros, "Process mining with the heuristicsminer algorithm."

[19] A. de Medeiros, A. Weijters, and W. M. van der Aalst, "Using genetic algorithms to mine process models: Representation, operators and results," 2005.

[20] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser, "Process mining based on regions of languages," in *Lecture Notes in Computer Science*, Springer-Verlag.

[21] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, "Synthesizing petri nets from state-based models,"

[22] W. van der Aalst and H. Verbeek, "Process discovery and conformance checking using passages," in *Fundamenta Informaticae*, vol. 131, no. 1, 2012.

[23] M. Solé and J. Carmona, *Incremental Process Discovery.* Springer-Verlag, 2012.

[24] D. Fahland, S. J. Leemans, and W. M. van der Aalst, "Discovering block-structured process models from event logs containing infrequent behaviour,"

[25] A. Burattin, A. Sperduti, and W. M. van der Aalst, "Control-flow discovery from event streams,"

[26] N. Busany and S. Maoz, "Behavioral log analysis with statistical guarantees," 2015.

[27] A. Biermann and J. Feldman, "On the synthesis of finite-state machines from samples of their behavior," in *IEEE Transactions on Computers*, IEEE.

[28] L. Brown, T. Cai, and A. DasGupta, "Interval estimation for a binomial proportion," in *Statistical Science*, Institute of Mathematical Statistics, 2010.

[29] "International workshop on business process intelligence." `https://www.win.tue.nl/bpi/doku.php?id=2016:start`. Accessed: 2016-03-04.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 6. März 2017 .....