Fuzzy Mining – Adaptive Process Simplification Based on Multi-Perspective Metrics

Christian W. Günther and Wil M.P. van der Aalst

Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
{c.w.gunther, w.m.p.v.d.aalst}@tue.nl

Abstract. Process Mining is a technique for extracting process models from execution logs. This is particularly useful in situations where people have an idealized view of reality. Real-life processes turn out to be less structured than people tend to believe. Unfortunately, traditional process mining approaches have problems dealing with unstructured processes. The discovered models are often "spaghetti-like", showing all details without distinguishing what is important and what is not. This paper proposes a new process mining approach to overcome this problem. The approach is configurable and allows for different faithfully simplified views of a particular process. To do this, the concept of a roadmap is used as a metaphor. Just like different roadmaps provide suitable abstractions of reality, process models should provide meaningful abstractions of operational processes encountered in domains ranging from healthcare and logistics to web services and public administration.

1 Introduction

Business processes, whether defined and prescribed or implicit and ad-hoc, drive and support most of the functions and services in enterprises and administrative bodies of today's world. For describing such processes, modeling them as graphs has proven to be a useful and intuitive tool. While modeling is well-established in process design, it is complicated to do for monitoring and documentation purposes. However, especially for monitoring, process models are valuable artifacts, because they allow us to communicate complex knowledge in intuitive, compact, and high-level form.

Process mining is a line of research which attempts to extract such abstract, compact representations of processes from their logs, i.e. execution histories [1–3, 5–7, 10, 14]. The α-algorithm, for example, can create a Petri net process model from an execution log [2]. In the last years, a number of process mining approaches have been developed, which address the various *perspectives* of a process (e.g., control flow, social network), and use various techniques to generalize from the log (e.g., genetic algorithms, theory of regions [12, 4]). Applied to explicitly designed, well-structured, and rigidly enforced processes, these techniques are able to deliver an impressive set of information, yet their purpose is somewhat limited to verifying the compliant execution. However, most processes in real life have not been purposefully designed and optimized, but have evolved over time or are not even explicitly defined. In such situations, the application of process mining is far more interesting, as it is not limited to re-discovering what we already know, but it can be used to *unveil previously hidden knowledge*.

Over the last couple of years we obtained much experience in applying the tried-and-tested set of mining algorithms to real-life processes. Existing algorithms tend to perform well on structured processes, but often fail to provide insightful models for less structured processes. The phrase “spaghetti models” is often used to refer to the results of such efforts. The problem is not that existing techniques produce incorrect results. In fact, some of the more robust process mining techniques guarantee that the resulting model is “correct” in the sense that reality fits into the model. The problem is that the resulting model shows all details without providing a suitable abstraction. This is comparable to looking at the map of a country where all cities and towns are represented by identical nodes and all roads are depicted in the same manner. The resulting map is correct, but not very suitable. Therefore, the concept of a roadmap is used as a metaphor to visualize the resulting models. Based on an analysis of the log, the importance of activities and relations among activities are taken into account. Activities and their relations can be clustered or removed depending on their role in the process. Moreover, certain aspects can be emphasized graphically just like a roadmap emphasizes highways and large cities over dirt roads and small towns. As will be demonstrated in this paper, the roadmap metaphor allows for meaningful process models.

In this paper we analyze the problems traditional mining algorithms have with less-structured processes (Section 2), and use the metaphor of maps to derive a novel, more appropriate approach from these lessons (Section 3). We abandon the idea of performing process mining confined to one perspective only, and propose a multi-perspective set of log-based process metrics (Section 4). Based on these, we have developed a flexible approach for *Fuzzy Mining*, i.e. adaptively simplifying mined process models (Section 5).

2 Less-Structured Processes – The Infamous Spaghetti Affair

The fundamental idea of process mining is both simple and persuasive: There is a process which is unknown to us, but we can follow the traces of its behavior, i.e. we have access to enactment logs. Feeding those into a process mining technique will yield an aggregate description of that observed behavior, e.g. in form of a process model.

In the beginning of process mining research, mostly artificially generated logs were used to develop and verify mining algorithms. Then, also logs from real-life workflow management systems, e.g. Staffware, could be successfully mined with these techniques. Early mining algorithms had high requirements towards the qualities of log files, e.g. they were supposed to be complete and limited to events of interest. Yet, most of the resulting problems could be easily remedied with more data, filtering the log and tuning the algorithm to better cope with problematic data.

While these successes were certainly convincing, most real-life processes are not executed within rigid, inflexible workflow management systems and the like, which enforce correct, predictive behavior. It is the inherent inflexibility of these systems which drove the majority of process owners (i.e., organizations having the need to support processes) to choose more flexible or ad-hoc solutions. Concepts like Adaptive Workflow or Case Handling either allow users to change the process at runtime, or define processes in a somewhat more “loose” manner which does not strictly define a specific path of execution. Yet the most popular solutions for supporting processes do not enforce any defined behavior at all, but merely offer functionality like sharing data and passing messages between users and resources. Examples for these systems are ERP (Enterprise Resource Planning) and CSCW (Computer-Supported Cooperative Work) systems, custom-built solutions, or plain E-Mail.

It is obvious that *executing a process within such less restrictive environments will lead to more diverse and less-structured behavior*. This abundance of observed behavior, however, unveiled a fundamental weakness in most of the early process mining algorithms. When these are used to mine logs from less-structured processes, the result is usually just as unstructured and hard to understand. These "spaghetti" process models do not provide any meaningful abstraction from the event logs themselves, and are therefore useless to process analysts. It is important to note that these "spaghetti" models are not incorrect. The problem is that *the processes themselves* are really "spaghetti-like", i.e., the model is an accurate reflection of reality.

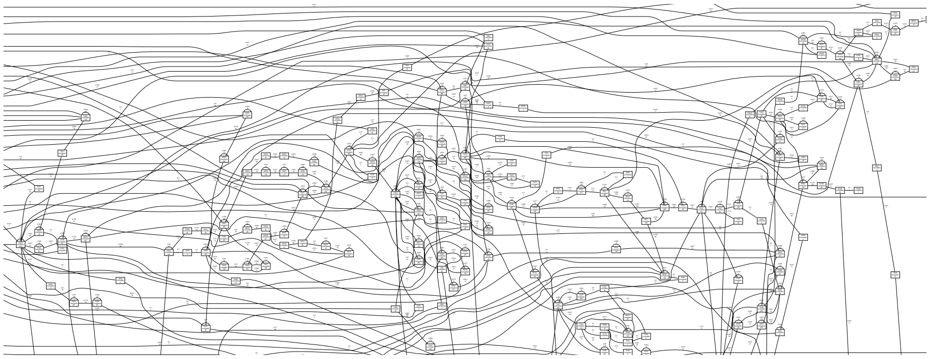


Fig. 1. Excerpt of a typical "Spaghetti" process model (ca. 20% of complete model).

An example of such a "spaghetti" model is given in Figure 1. It is noteworthy that this figure shows only a small excerpt (ca. 20%) of a highly unstructured process model. It has been mined from machine test logs using the Heuristics Miner, one of the traditional process mining techniques which is most resilient towards noise in logs [14]. Although this result is rather useful, certainly in comparison with other early process mining techniques, it is plain to see that deriving helpful information from it is not easy.

Event classes found in the log are interpreted as activity nodes in the process model. Their sheer amount makes it difficult to focus on the interesting parts of the process. The abundance of arcs in the model, which constitute the actual "spaghetti", introduce an even greater challenge for interpretation. Separating cause from effect, or the general direction in which the process is executed, is not possible because virtually every node is transitively connected to any other node in both directions. This mirrors the crux of flexibility in process execution – when people are free to execute anything in any given order they will usually make use of such feature, which renders monitoring business activities an essentially infeasible task.

We argue that the fault for these problems lies neither with less-structured processes, nor with process mining itself. Rather, it is the result of a number of, mostly implicit, assumptions which process mining has historically made, both with respect to the event logs under consideration, and regarding the processes which have generated them. While being perfectly sound in structured, controlled environments, *these assumptions do not hold in less-structured, real-life environments*, and thus ultimately make traditional process mining fail there.

Assumption 1: **All logs are reliable and trustworthy**. Any event type found in the log is assumed to have a corresponding logical activity in the process. However, activities in real-life processes may raise a random number of seemingly unrelated events. Activities may also go unrecorded, while other events do not correspond to any activity at all.

The assumption that logs are well-formed and homogeneous is also often not true. For example, a process found in the log is assumed to correspond to one logical entity. In less-structured environments, however, there are often a number of “tacit” process types which are executed, and thus logged, under the same name.

Also, the idea that all events are raised on the same level of abstraction, and are thus equally important, is not true in real-life settings. Events on different levels are “flattened” into the same event log, while there is also a high amount of informational events (e.g., debug messages from the system) which need to be disregarded.

Assumption 2: **There exists an exact process which is reflected in the logs**. This assumption implies that there is the one perfect solution out there, which needs to be found. Consequently, the mining result should model the process *completely*, *accurately*, and *precisely*. However, as stated before, spaghetti models are not necessarily incorrect – the models look like spaghetti, because they precisely describe every detail of the less-structured behavior found in the log. A more high-level solution, which is able to abstract from details, would thus be preferable.

Traditional mining algorithms have also been confined to a single *perspective* (e.g., control flow, data), as such isolated view is supposed to yield higher precision. However, perspectives are interacting in less-structured processes, e.g. the data flow may complement the control flow, and thus also needs to be taken into account.

In general, the assumption of a perfect solution is not well-suited for real-life application. Reality often differs significantly from theory, in ways that had not been anticipated. Consequently, useful tools for practical application must be *explorative*, i.e. support the analyst to tweak results and thus capitalize on their knowledge.

We have conducted process mining case studies in organizations like Philips Medical Systems, UWV, Rijkswaterstaat, the Catharina Hospital Eindhoven and the AMC hospital Amsterdam, and the Dutch municipalities of Alkmaar and Heusden. Our experiences in these case studies have shown the above assumptions to be violated in all ways imaginable. Therefore, to make process mining a useful tool in practical, less-structured settings, these assumptions need to be discarded. The next section introduces the main concept of our mining approach, which takes these lessons into account.

3 An Adaptive Approach for Process Simplification

Process mining techniques which are suitable for less-structured environments need to be able to provide a high-level view on the process, abstracting from undesired details. The field of cartography has always been faced with a quite similar challenge, namely to simplify highly complex and unstructured topologies. Activities in a process can be related to locations in a topology (e.g. towns or road crossings) and precedence relations to traffic connections between them (e.g., railways or motorways).

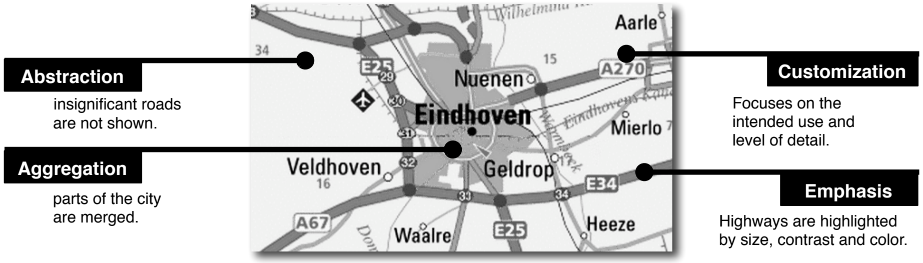

Abstraction
insignificant roads are not shown.
Aggregation
parts of the city are merged.
Aarle
Nuenen
E25
A270
Eindhoven
Mierlo
Veldhoven
Geldrop
E34
A67
Waalre
E25
Heeze
Customization
Focuses on the intended use and level of detail.
Emphasis
Highways are highlighted by size, contrast and color.

Fig. 2. Example of a road map.

When one takes a closer look at maps (such as the example in Figure 2), the solution cartography has come up with to simplify and present complex topologies, one can derive a number of valuable concepts from them.

Aggregation: To limit the number of information items displayed, maps often show *coherent clusters of low-level detail information* in an aggregated manner. One example are cities in road maps, where particular houses and streets are combined within the city's transitive closure (e.g., the city of Eindhoven in Figure 2).

Abstraction: Lower-level information which is *insignificant in the chosen context* is simply omitted from the visualization. Examples are bicycle paths, which are of no interest in a motorist's map.

Emphasis: More significant information is *highlighted* by visual means such as *color*, *contrast*, *saturation*, and *size*. For example, maps emphasize more important roads by displaying them as thicker, more colorful and contrasting lines (e.g., motorway "E25" in Figure 2).

Customization: There is no one single map for the world. Maps are specialized on a defined *local context*, have a specific *level of detail* (city maps vs highway maps), and a dedicated *purpose* (interregional travel vs alpine hiking).

These concepts are universal, well-understood, and established. In this paper we explore how they can be used to simplify and properly visualize complex, less-structured processes. To do that, we need to develop appropriate decision criteria on which to base the simplification and visualization of process models. We have identified two fundamental *metrics* which can support such decisions: (1) *significance* and (2) *correlation*.

Significance, which can be determined both for event classes (i.e., activities) and binary precedence relations over them (i.e., edges), measures the *relative importance* of behavior. As such, it specifies the level of interest we have in events, or their occurring after one another. One example for measuring significance is by frequency, i.e. events or precedence relations which are observed more frequently are deemed more significant.

Correlation on the other hand is only relevant for precedence relations over events. It measures *how closely related* two events following one another are. Examples for measuring correlation include determining the overlap of data attributes associated to two events following one another, or comparing the similarity of their event names. More closely correlated events are assumed to share a large amount of their data, or have their similarity expressed in their recorded names (e.g. "check_customer_application" and "approve_customer_application").

Based on these two metrics, which have been defined specially for this purpose, we can sketch our approach for process simplification as follows.

- *Highly significant* behavior is *preserved*, i.e. contained in the simplified model.
- *Less significant* but *highly correlated* behavior is *aggregated*, i.e. hidden in clusters within the simplified model.
- *Less significant* and *lowly correlated* behavior is *abstracted from*, i.e. removed from the simplified model.

This approach can greatly reduce and focus the displayed behavior, by employing the concepts of aggregation and abstraction. Based on such simplified model, we can employ the concept of *emphasis*, by highlighting more significant behavior.

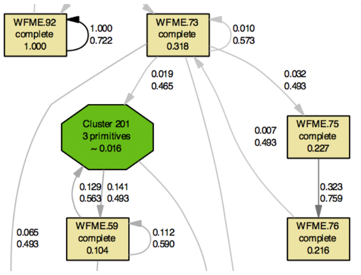


Fig. 3. Excerpt of a simplified and decorated process model.

Figure 3 shows an excerpt from a simplified process model, which has been created using our approach. Bright square nodes represent significant activities, the darker octagonal node is an aggregated cluster of three less-significant activities. All nodes are labeled with their respective significance, with clusters displaying the mean significance of their elements. The brightness of edges between nodes emphasizes their significance, i.e. more significant relations are darker. Edges are also labeled with their respective significance and correlation values. By either removing or hiding less significant information, this visualization enables the user to focus on the most interesting behavior in the process.

Yet, the question of what constitutes “interesting” behavior can have a number of answers, based on the process, the purpose of analysis, or the desired level of abstraction. In order to yield the most appropriate result, significance and correlation measures need to be configurable. We have thus developed a set of metrics, which can each measure significance or correlation based on different perspectives (e.g., control flow or data) of the process. By influencing the “mix” of these metrics and the simplification procedure itself, the user can *customize* the produced results to a large degree.

The following section introduces some of the metrics we have developed for significance and correlation in more detail.

4 Log-Based Process Metrics

Significance and correlation, as introduced in the previous section, are suitable concepts for describing the importance of behavior in a process in a compact manner. However, because they represent very generalized, condensed metrics, it is important to measure them in an *appropriate* manner. Taking into account the wide variety of processes, analysis questions and objectives, and levels of abstraction, it is necessary to make this measurement *adaptable* to such parameters.

Our approach is based on a configurable and extensible framework for measuring significance and correlation. The design of this framework is introduced in the next subsection, followed by detailed introductions to the three primary types of metrics: unary significance, binary significance, and binary correlation.

4.1 Metrics Framework

An important property of our measurement framework is that for each of the three primary types of metrics (unary significance, binary significance, and binary correlation) different implementations may be used. A metric may either be measured directly from the log (*log-based metric)*, or it may be based on measurements of other, log-based metrics (*derivative metric*).

When the log contains a large number of undesired events, which occur in between desired ones, actual causal dependencies between the desired event classes may go unrecorded. To counter this, our approach also measures *long-term relations*, i.e. when the sequence A, B, C is found in the log, we will not only record the relations $A \rightarrow B$ and $B \rightarrow C$, but also the *length-2-relationship* $A \rightarrow C$. We allow measuring relationships of arbitrary length, while the measured value will be *attenuated*, i.e. decreased, with increasing length of relationship.

4.2 Unary Significance Metrics

Unary significance describes the relative importance of an event class, which will be represented as a node in the process model. As our approach is based on removing less significant behavior, and as removing a node implies removing all of its connected arcs, unary significance is the primary driver of simplification.

One metric for unary significance is *frequency significance*, i.e. the more often a certain event class was observed in the log, the more significant it is. Frequency is a log-based metric, and is in fact the most intuitive of all metrics. Traditional process mining techniques are built solely on the principle of measuring frequency, and it remains an important foundation of our approach. However, real-life logs often contain a large number of events which are in fact not very significant, e.g. an event which describes saving the process state after every five activities. In such situations, frequency plays a diminished role and can rather distort results.

Another, derivate metric for unary significance is *routing significance*. The idea behind routing significance is that points, at which the process either forks (i.e., split nodes) or synchronizes (i.e., join nodes), are interesting in that they substantially define the structure of a process. The higher the number and significance of predecessors for a node (i.e., its incoming arcs) differs from the number and significance of its successors (i.e., outgoing arcs), the more important that node is for *routing* in the process. Routing significance is important as *amplifier metric*, i.e. it helps separating important routing nodes (whose significance it increases) from those less important.

4.3 Binary Significance Metrics

Binary significance describes the relative importance of a precedence relation between two event classes, i.e. an edge in the process model. Its purpose is to amplify and to isolate the observed behavior which is supposed to be of the greatest interest. In our simplification approach, it primarily influences the selection of edges which will be included in the simplified process model.

Like for unary significance, the log-based *frequency significance* metric is also the most important implementation for binary significance. The more often two event classes are observed after one another, the more significant their precedence relation.

The *distance significance* metric is a derivative implementation of binary significance. The more the significance of a relation differs from its source and target nodes' significances, the less its distance significance value. The rationale behind this metric is, that globally important relations are also always the most important relations for their endpoints. Distance significance locally amplifies crucial key relations between event classes, and weakens already insignificant relations. Thereby, it can clarify ambiguous situations in edge abstraction, where many relations “compete” over being included in the simplified process model. Especially in very unstructured execution logs, this metric is an indispensible tool for isolating behavior of interest.

4.4 Binary Correlation Metrics

Binary correlation measures the distance of events in a precedence relation, i.e. how closely related two events following one another are. Distance, in the process domain, can be equated to the magnitude of context change between two activity executions. Subsequently occurring activities which have a more similar context (e.g., which are executed by the same person or in a short timeframe) are thus evaluated to be higher correlated. Binary correlation is the main driver of the decision between *aggregation or abstraction* of less-significant behavior.

Proximity correlation evaluates event classes, which occur shortly after one another, as highly correlated. This is important for identifying clusters of events which correspond to one logical activity, as these are commonly executed within a short timeframe.

Another feature of such clusters of events occurring within the realm of one higher-level activity is, that they are executed by the same person. *Originator correlation* between event classes is determined from the names of the persons, which have triggered two subsequent events. The more similar these names, the higher correlated the respective event classes. In real applications, user names often include job titles or function identifiers (e.g.“sales_John” and “sales_Paul”). Therefore, this metric implementation is a valuable tool also for unveiling implicit correlation between events.

Endpoint correlation is quite similar, however, instead of resources it compares the *activity names* of subsequent events. More similar names will be interpreted as higher correlation. This is important for low-level logs including a large amount of less significant events which are closely related. Most of the time, events which reflect similar tasks also are given similar names (e.g., “open_valve13” and “close_valve13”), and this metric can unveil these implicit dependencies.

In most logs, events also include additional attributes, containing snapshots from the *data perspective* of the process (e.g., the value of an insurance claim). In such cases, the *selection of attributes* logged for each event can be interpreted as its context. Thus, the *data type correlation* metric evaluates event classes, where subsequent events share a large amount of data types (i.e., attribute keys), as highly correlated. *Data value correlation* is more specific, in that it also takes the values of these common attributes into account. In that, it uses relative similarity, i.e. small changes of an attribute value will compromise correlation less than a completely different value.

Currently, all implementations for binary correlation in our approach are log-based. The next section introduces our approach for adaptive simplification and visualization of complex process models, which is based on the aggregated measurements of all metric implementations which have been introduced in this section.

5 Adaptive Graph Simplification

Most process mining techniques follow an *interpretative* approach, i.e. they attempt to map behavior found in the log to typical process design patterns (e.g., whether a split node has AND- or XOR-semantics). Our approach, in contrast, focuses on high-level mapping of behavior found in the log, while not attempting to discover such patterns. Thus, creating the initial (non-simplified) process model is straightforward: All event classes found in the log are translated to activity nodes, whose importance is expressed by unary significance. For every observed precedence relation between event classes, a corresponding directed edge is added to the process model. This edge is described by the binary significance and correlation of the ordering relation it represents.

Subsequently, we apply three *transformation methods* to the process model, which will successively simplify specific aspects of it. The first two phases, *conflict resolution* and *edge filtering*, remove edges (i.e., precedence relations) between activity nodes, while the final *aggregation and abstraction* phase removes and/or clusters less-significant nodes. Removing edges from the model first is important – due to the less-structured nature of real-life processes and our measurement of long-term relationships, the initial model contains *deceptive* ordering relations, which do not correspond to valid behavior and need to be discarded. The following sections provide details about the three phases of our simplification approach, given in the order in which they are applied to the initial model.

5.1 Conflict Resolution in Binary Relations

Whenever two nodes in the initial process model are connected by edges in both directions, they are defined to be *in conflict*. Depending on their specific properties, conflicts may represent one of three possible situations in the process:

- *Length-2-loop:* Two activities A and B constitute a loop in the process model, i.e. after executing A and B in sequence, one may return to A and start over. In this case, the conflicting ordering relations between these activities are explicitly allowed in the original process, and thus need to be preserved.
- *Exception:* The process orders $A \rightarrow B$ in sequence, however, during real-life execution the exceptional case of $B \rightarrow A$ also occurs. Most of the time, the "normal" behavior is clearly more significant. In such cases, the "weaker" relation needs to be discarded to focus on the main behavior.
- *Concurrency:* A and B can be executed in any order (i.e., they are on two distinct, parallel paths), the log will most likely record both possible cases, i.e. $A \rightarrow B$ and $B \rightarrow A$, which will create a conflict. In this case, both conflicting ordering relations need to be removed from the process model.

Conflict resolution attempts to classify each conflict as one of these three cases, and then resolves it accordingly. For that, it first determines the *relative significance* of both conflicting relations.

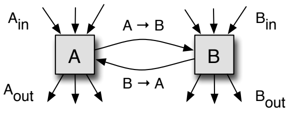


Fig. 4. Evaluating the relative significance of conflicting relations.

Figure 4 shows an example of two activities A and B in conflict. The relative significance for an edge $A \rightarrow B$ can be determined as follows.

Definition 1 (Relative significance). *Let $\mathcal{N}$ be the set of nodes in a process model, and let $sig : \mathcal{N} \times \mathcal{N} \rightarrow \mathbb{R}_0^+$ be a relation that assigns to each pair of nodes $A, B \in \mathcal{N}$ the significance of a precedence relation over them. $rel : \mathcal{N} \times \mathcal{N} \rightarrow \mathbb{R}_0^+$ is a relation which assigns to each pair of nodes $A, B \in \mathbb{N}$ the* relative importance *of their ordering relation:* $rel(A, B) = \frac{1}{2} \cdot \frac{sig(A,B)}{\sum_{X \in \mathcal{N}} sig(A,X)} + \frac{1}{2} \cdot \frac{sig(A,B)}{\sum_{X \in \mathcal{N}} sig(X,B)}$

Every ordering relation $A \rightarrow B$ has a set of *competing relations* $Comp_{AB} = A_{out} \cup B_{in}$. This set of competing relations is composed of A_{out}, i.e. all edges starting from A, and of B_{in}, i.e. all edges pointing to B (cf. Figure 4). Note that this set also contains the reference relation itself, i.e. more specifically: $B_{in} \cap A_{out} = \{A \rightarrow B\}$. By dividing the significance of an ordering relation $A \rightarrow B$ with the sum of all its competing relations' significances, we get the importance of this relation in its *local context*.

If the relative significance of both conflicting relations, $rel(A, B)$ and $rel(B, A)$ exceeds a specified *preserve threshold* value, this signifies that A and B are apparently forming a *length-2-loop*, which is their most significant behavior in the process. Thus, in this case, both $A \rightarrow B$ and $B \rightarrow A$ will be preserved.

In case at least one conflicting relation's relative significance is below this threshold, the *offset* between both relations' relative significances is determined, i.e. $ofs(A, B) = |rel(A, B) - rel(B, A)|$. The larger this offset value, the more the relative significances of both conflicting relations differ, i.e. one of them is clearly more important. Thus, if the offset value exceeds a specified *ratio threshold*, we assume that the relatively less significant relation is in fact an *exception* and remove it from the process model.

Otherwise, i.e. if at least one of the relations has a relative significance below the *preserve threshold* and their offset is smaller than the *ratio threshold*, this signifies that both $A \rightarrow B$ and $B \rightarrow A$ are relations which are of no greater importance for both their source and target activities. This low, yet balanced relative significance of conflicting relations hints at A and B being executed *concurrently*, i.e. in two separate threads of the process. Consequently, both edges are removed from the process model, as they do not correspond to factual ordering relations.

5.2 Edge Filtering

Although conflict resolution removes a number of edges from the process model, the model still contains a large amount of precedence relations. To infer further structure from this model, it is necessary to remove most of these remaining edges by *edge filtering*, which isolates the most important behavior. The obvious solution is to remove the globally least significant edges, leaving only highly significant behavior. However, this approach yields sub-optimal results, as it is prone to create small, disparate clusters of highly frequent behavior. Also, in the subsequent *aggregation* step, highly correlated relations play an important part in connecting clusters, even if they are not very significant.

Therefore, our *edge filtering* approach evaluates each edge $A \rightarrow B$ by its *utility* $util(A, B)$, a weighed sum of its significance and correlation. A configurable *utility ratio* $ur \in [0, 1]$ determines the weight, such that $util(A, B) = ur \cdot sig(A, B) + (1 - ur) \cdot cor(A, B)$. A larger value for ur will preserve more significant edges, while a smaller value will favor highly correlated edges.

Figure 5 shows an example for processing the incoming arcs of a node A. Using a utility ratio of 0.5, i.e. taking significance and correlation equally into account, the utility value is calculated, which ranges from 0.4 to 1.0 in this example.

Filtering edges is performed on a local basis, i.e. for each node in the process model, the algorithm preserves its incoming and outgoing edges with the highest utility value. The decision of which edges get preserved is configured by the *edge cutoff* parameter

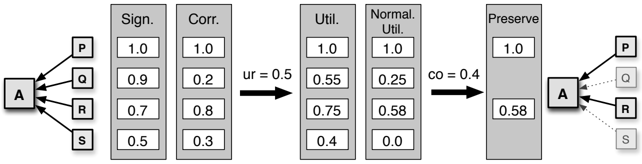


Fig. 5. Filtering the set of incoming edges for a node A.

$co \in [0, 1]$. For every node N, the utility values for each incoming edge $X \rightarrow N$ are normalized to $[0, 1]$, so that the weakest edge is assigned 0 and the strongest one 1. All edges whose normalized utility value exceeds *co* are added to the preserved set. In the example in Figure 5, only two of the original edges, are preserved, using a edge cutoff value of 0.4: $P \rightarrow A$ (with normalized utility of 1.0) and $R \rightarrow A$ (norm. utility of 0.56). The outgoing edges are processed in the same manner for each node.

The *edge cutoff* parameter determines the aggressiveness of the algorithm, i.e. the higher its value, the more likely the algorithm is to remove edges. In very unstructured processes, where precedence relations are likely to have a balanced significance, it is often useful to use a lower utility ratio, such that correlation will be taken more into account and resolve such ambiguous situations. On top of that, a high edge cutoff will act as an amplifier, helping to distinguish the most important edges.

Note that our edge filtering approach starts from an empty set of precedence relations, i.e. all edges are removed by default. Only if an edge is selected locally for at least one node, it will be preserved. This approach keeps the process model connected, while clarifying ambiguous situations. Whether an edge is preserved depends on its utility for describing the behavior of the activities it connects – and not on global comparisons with other parts of the model, which it does not even interact with.

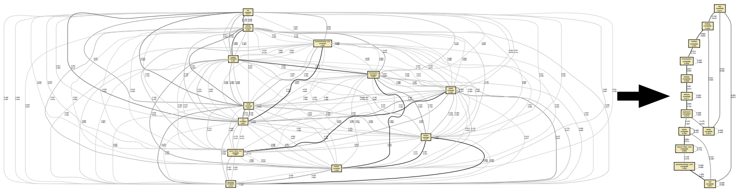


Fig. 6. Example of a process model before (left) and after (right) edge filtering.

Figure 6 shows the effect of edge filtering applied to a small, but very unstructured process. The number of nodes remains the same, while removing an appropriate subset of edges clearly brings structure to the previously chaotic process model.

5.3 Node Aggregation and Abstraction

While removing edges brings structure to the process model, the most effective tool for simplification is removing nodes. It enables the analyst to focus on an interesting subset of activities. Our approach preserves highly correlated groups of less-significant nodes as aggregated clusters, while removing isolated, less-significant nodes. Removing nodes is based on the *node cutoff* parameter. Every node whose unary significance is below this threshold becomes a *victim*, i.e. it will either be aggregated or abstracted from. The first phase of our algorithm builds initial clusters of less-significant behavior as follows.

- For each victim, find the most highly correlated neighbor (i.e., connected node)
- If this neighbor is a cluster node, add the victim to this cluster.
- Otherwise, create a new cluster node, and add the victim as its first element.

Whenever a node is added to a cluster, the cluster will "inherit" the ordering relations of that node, i.e. its incoming and outgoing arcs, while the actual node will be hidden. The second phase is *merging* the clusters, which is necessary as most clusters will, at this stage, only consist of one single victim. The following routine is performed to aggregate larger clusters and decrease their number.

- For each cluster, check whether all predecessors or all successors are also clusters.
- If all predecessor nodes are clusters as well, merge with the most highly correlated one and move on to the next cluster.
- If all successors are clusters as well, merge with the most highly correlated one.
- Otherwise, i.e. if both the cluster's pre- and postset contain regular nodes, the cluster is left untouched.

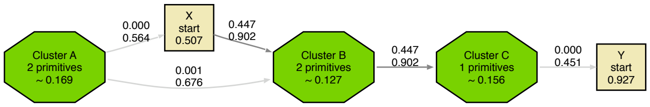


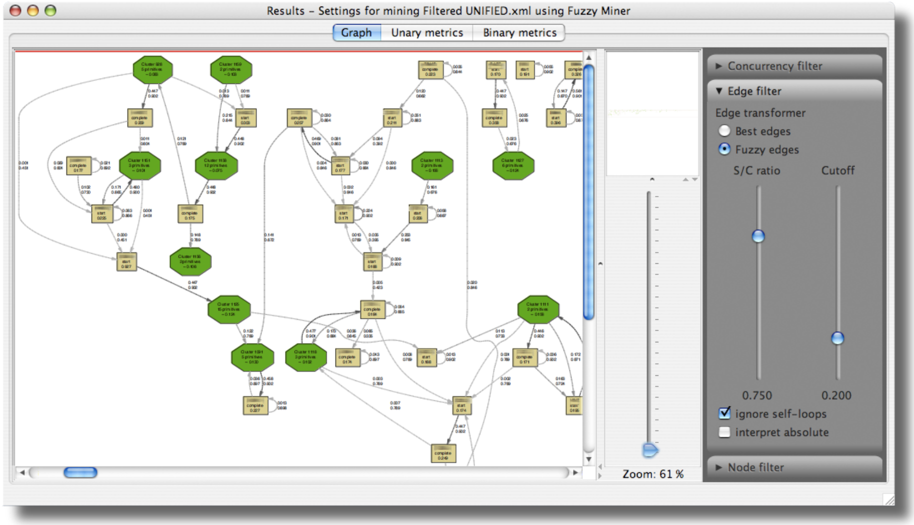
Fig. 7. Excerpt of a clustered model after the first aggregation phase.

It is important that clusters will only be merged, if the "victim" has only clusters in his pre- or postset. Figure 7 shows an example of a process model after the first phase of clustering. Cluster A cannot merge with cluster B, as they are also both connected to node X. Otherwise, X would be connected to the merged cluster in both directions, making the model less informative. However, clusters B and C can merge, as B's postset consists only of C. This simplification of the model does not lessen the amount of information, and is thus valid.

The last phase, which constitutes the *abstraction*, removes *isolated* and *singular* clusters. Isolated clusters are detached parts of the process, which are less significant and highly correlated, and which have thus been folded into one single, isolated cluster node. It is obvious that such detached nodes do not contribute to the process model, which is why they are simply removed. Singular clusters consist only of one, single activity node. Thus, they represent less-significant behavior which is not highly correlated to adjacent behavior. Singular clusters are undesired, because they do not simplify the model. Therefore, they are removed from the model, while their most significant precedence relations are transitively preserved (i.e., their predecessors are artificially connected to their successors, if such edge does not already exist in the model).

6 Implementation and Application

We have implemented our approach as the Fuzzy Miner plugin for the ProM framework [8]. All metrics introduced in Section 4 have been implemented, and can be configured by the user. Figure 8 shows the result view of the Fuzzy Miner, with the simplified graph view on the left, and a configuration pane for simplification parameters on the right. Alternative views allow the user to inspect and verify measurements for all metrics, which helps to tune these metrics to the log.



Edge transformer
○ Best edges
● Fuzzy edges

S/C ratio 0.750
Cutoff 0.200

☑ ignore self-loops
□ interpret absolute

Zoom: 61 %

Fig. 8. Screenshot of the Fuzzy Miner, applied to the very large and unstructured log also used for mining the model in Figure 1.

Note that this approach is the result of valuable lessons learnt from a great number of case studies with real-life logs. As such, both the applied metrics and the simplification algorithm have been optimized using large amounts actual, less-structured data. While it is difficult to validate the approach formally, the Fuzzy Miner has already become one of the most useful tools in case study applications.

For example, Figure 8 shows the result of applying the Fuzzy Miner to a large test log of manufacturing machines (155.296 events in 16 cases, 826 event classes). The approach has been shown to scale well and is of linear complexity. Deriving all metrics from the mentioned log was performed in less than ten seconds, while simplifying the resulting model took less than two seconds on a 1.8 GHz dual-core machine. While this model has been created from the raw logs, Figure 1 has been mined with the Heuristics Miner, after applying log filters [14]. It is obvious that the Fuzzy Miner is able to clean up a large amount of confusing behavior, and to infer and extract structure from what is chaotic. We have successfully used the Fuzzy Miner on various machinery test and usage logs, development process logs, hospital patient treatment logs, logs from case handling systems and web servers, among others. These are notoriously flexible and unstructured environments, and we hold our approach to be one of the most useful tools for analyzing them so far.

7 Related Work

While parting with some characteristics of traditional process mining techniques, such as absolute precision and describing the complete behavior, it is obvious that the approach described in this paper is based on previous work in this domain, most specifically control-flow mining algorithms [2, 14, 7]. The mining algorithm most related to Fuzzy Mining is the Heuristics Miner, which also employs heuristics to limit the set of precedence relations included in the model [14]. Our approach also incorporates concepts from log filtering, i.e. removing less significant events from the logs [8]. However, the foundation on multi-perspective metrics, i.e. looking at all aspects of the process at once, its interactive and explorative nature, and the integrated simplification algorithm clearly distinguishes Fuzzy Mining from all previous process mining techniques.

The adaptive simplification approach presented in Section 5 uses concepts from the domains of data clustering and graph clustering. Data clustering attempts to find related subsets of attributes, based on a binary distance metric inferred upon them [11]. Graph clustering algorithms, on the other hand, are based on analyzing structural properties of graphs, from which they derive partitioning strategies [13, 9]. Our approach, however, is based on a unique combination of analyzing the *significance* and *correlation* of graph elements, which are based on a wide set of process perspectives. It integrates abstraction and aggregation, and is also more specialized towards the process domain.

8 Discussion and Future Work

We have described the problems traditional process mining techniques face when applied to large, less-structured processes, as often found in practice. Subsequently, we have analyzed the causes for these problems, which lie in a mismatch between fundamental assumptions of traditional process mining, and the characteristics of real-life processes. Based on this analysis, we have developed an adaptive simplification and visualization technique for process models, which is based on two novel metrics, *significance* and *correlation*. We have described a framework for deriving these metrics from an enactment log, which can be adjusted to particular situations and analysis questions.

While the fine-grained configurability of the algorithm and its metrics makes our approach universally applicable, it is also one of its weaknesses, as finding the “right” parameter settings can sometimes be time-consuming. Thus, our next steps will concentrate on deriving higher-level parameters and sensible default settings, while preserving the full range of parameters for advanced users. Further work will concentrate on extending the set of metric implementations and improving the simplification algorithm.

It is our belief that process mining, in order to become more meaningful, and to become applicable in a wider array of practical settings, needs to address the problems it has with unstructured processes. We have shown that the traditional desire to model the *complete* behavior of a process in a *precise* manner conflicts with the original goal, i.e. to provide the user with *understandable, high-level information*. The success of process mining will depend on whether it is able to balance these conflicting goals sensibly. Fuzzy Mining is a first step in that direction.

Acknowledgements This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Dutch Ministry of Economic Affairs.

References

1. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
2. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
3. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
4. E. Badouel, L. Bernardinello, and P. Darondeau. The Synthesis Problem for Elementary Net Systems is NP-complete. *Theoretical Computer Science*, 186(1-2):107–134, 1997.
5. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
6. A. Datta. Automating the Discovery of As-Is Business Process Models: Probabilistic and Algorithmic Approaches. *Information Systems Research*, 9(3):275–301, 1998.
7. B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Process Mining: Building Instance Graphs. In P. Atzeni, W. Chu, H. Lu, S. Zhou, and T.W. Ling, editors, *International Conference on Conceptual Modeling (ER 2004)*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, Berlin, 2004.
8. B.F. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau, editors, *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
9. S. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.
10. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
11. A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.
12. A.K.A. de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic Process Mining: A Basic Approach and its Challenges. In C. Bussler et al., editor, *BPM 2005 Workshops (Workshop on Business Process Intelligence)*, volume 3812 of *Lecture Notes in Computer Science*, pages 203–215. Springer-Verlag, Berlin, 2006.
13. A. Pothen, H. D. Simon, and K. Liou. Partitioning sparse matrics with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, July 1990.
14. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.