Change Your History: Learning from Event Logs to Improve Processes

Wil M.P. van der Aalst[*†] Wei Zhe Low[†] Moe T. Wynn[†] Arthur H.M. ter Hofstede[†*]

[*]Technische Universiteit Eindhoven (TU/e), Eindhoven, The Netherlands
Email: w.m.p.v.d.aalst@tue.nl

[†]Queensland University of Technology (QUT), Brisbane, Queensland, Australia
E-mail: w4.low|m.wynn|a.terhofstede@qut.edu.au

***Abstract*—The abundance of event data enables new forms of analysis that facilitate process improvement. Process mining provides a novel set of tools to discover the real process, to detect deviations from some normative process, and to analyze bottlenecks and waste. The lion's share of process mining focuses on the "as-is" situation rather than the "to-be" situation. Clearly, analysis should aim at actionable insights and concrete suggestions for improvement. However, state-of-the-art techniques do not allow for this. Techniques like simulation can be used to do "what-if" analysis but are not driven by event data, and as a result, improvements can be very unrealistic. Techniques for predictive analytics and combinatorial optimization are data-driven but mostly focus on well-structured decision problems. Operational processes within complex organizations cannot be mapped onto a simulation model or simple decision problem. This paper provides a novel approach based on event logs as used by process mining techniques. Instead of trying to create or modify process models, this approach works directly on the event log itself. It aims to "improve history" rather than speculate about a highly uncertain future. By showing concrete improvements in terms of partly modified event logs, the stakeholders can learn from earlier mistakes and inefficiencies. This is similar to analyzing a soccer match to improve a team's performance in the next game. This paper introduces the idea using event logs in conjunction with flexible "compatibility" and "utility" notions. An initial prototype –serving as a proof-of-concept– was realized as a ProM plug-in and tested on real-life event logs.**

I. Introduction

In 2014, the German national soccer team won the World Cup Championship. The German team and SAP developed and used the tool "Match Insights" to obtain a competitive advantage. The tool was used to analyze former soccer matches in detail. The idea of analyzing soccer matches to improve performance based on facts rather than misguided beliefs is not new. Already in 1950, Charles Reep created a toolkit to analyze soccer games. Reep developed a notational analysis system of soccer in an attempt to provide empirical evidence for superior playing strategies. Today, lots of detailed soccer data are collected and, recently, various innovative visualizations have been developed, for example the Wave visualization tool developed in a collaboration between TU/e and Infostrada Sports. The tool visualizes low-level spatio-temporal soccer data thus providing novel insights at different levels of granularity [1]. Figure 1 shows a high-level overview of a match, but it is also possible to zoom-in on particular players, situations, etc. The potential of analyzing games to learn from mistakes can also be seen in other sports. In baseball, the term "sabermetrics" refers to the collection of statistics to search for objective knowledge about baseball. In "Moneyball" [2] Michael Lewis reports on the successes of Oakland Athletics due to their superior understanding of statistics to outperform better-resourced competitors.

Performing operational processes in organizations is in many ways comparable to playing soccer matches. One organization is competing with other organizations, just as teams are competing. Within an organization, people need to feel part of the same team to be most effective. An improvement in one department may cause problems in other departments. Analyzing operational processes based on historic event data is as useful as analyzing a match after the fact. This explains the growing interest in analytics and data science for process improvement.

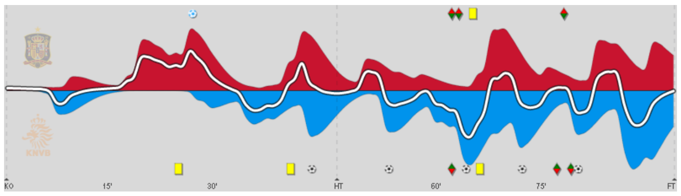


Fig. 1. Visualization of the remarkable soccer match Spain versus Netherlands on June 13th 2014 (1-5) [1].

The term *process mining* refers to techniques that extract knowledge from event logs [3]. Process mining techniques form a family of *a-posteriori* analysis techniques exploiting the information recorded in audit trails, transaction logs, databases, etc. Process mining includes (automated) process discovery (i.e., learning process models from raw event data), conformance checking (i.e., monitoring deviations by comparing model and log [5]), social network/organizational mining, automated construction of simulation models, model extension, model repair, case prediction, and history-based recommendations. Most of the current process mining techniques aim to provide insights and do not (yet) support the automation of process improvements (apart from initial work on operational support [3]).

Process mining complements traditional model-driven approaches like business process simulation [4]. Simulation can be used for "what-if" analysis. However, simulation models are often at a higher abstraction-level and not directly connected to the low-level event data. Hence, recommendations based on simulation may be naïve or overly optimistic.

Data-driven approaches like predictive analytics [6] and combinatorial optimization [7] tend to focus on specific decision problems. Predictive analytics approaches use machine learning or data mining to make predictions about future events. The context may be fuzzy and answers may be inconclusive, but the goal is to analyze a specific variable, e.g., will a customer buy a product or not. Combinatorial optimization assumes a more reliable problem setting and within this setting provides some guarantees (e.g., lowest costs).

The approach presented in this paper differs from the above approaches in the sense that we stay close to the original event data and do not reduce things to a process model, stochastic model, decision tree, or regression formula.

Starting point is an *event log* as shown in Table I. Activities are executed for *cases*, i.e., *process instances*. In Table I each row corresponds to an *activity instance*. Here it is assumed that each activity has a *start time* and a *completion time* and is executed by a *resource*. (In Section II we will provide a more general definition.) XES (eXtensible Event Stream) [8] and related standards such as MXML provide a tool- and process-independent way to store event data as event logs. Process mining tools such as ProM and Disco can handle such event logs and automatically discover process models or check conformance. *Instead of discovering a process model highlighting performance or conformance problems, we propose a more detailed analysis of the event log also showing improvements at the instance level.* This can be compared to analyzing a soccer match based on concrete game situations, e.g., missed scoring opportunities or defensive errors. This is done by changing the event log, e.g., activities are reordered or resource allocations are changed to improve the process in terms of flow time or costs.

Modifying event logs to better understand process improvements can be operationalized using the following two notions: *compatibility* and *utility*. Event log L_1 is *compatible* with L_2 (notation: $L_1 \sim_c L_2$) if L_1 is a possible alternative for L_2. Most of the event log must remain invariant and only selected parts that could actually be influenced can be modified, e.g., activities can only be transferred to people having the same role and activities cannot start before all inputs are available. Moreover, a *utility function* needs to be defined. $util_g(L)$ is the utility of an event log. Positive changes lead to a higher utility. Negative changes (e.g., higher costs, longer flow times, more rejects, etc.) lead to a lower utility. L_1 *potentially improves* L_2 if and only if $L_1 \sim_c L_2$ and $util_g(L_1) \geq util_g(L_2)$. L_1 is an *optimal improvement* of L_2 if no better improvement can be found when considering all compatible alternatives. After finding potential or optimal improvements, the differences are visualized for stakeholders (managers, analysts and end-users). Improvements can be shown at the process level (e.g., the number of transfers between two resources or the reduced waiting time for a bottleneck activity) and at the instance level (e.g., visualizing time shifts within individual cases).

We feel that any redesign effort at the process level should be preceded by attempts to "change histories" in event logs. By showing alternative scenarios based on real historic cases, a reality check is possible and stakeholders are stimulated to think about process improvements. This is comparable to a soccer player learning from video recordings of his own mistakes.

The remainder of this paper is organized as follows. Section II provides a detailed introduction to the core concepts in event logs. Section III briefly introduces the basic types of process mining: process discovery, conformance checking, and performance analysis. The approach to change event logs in a controlled manner is described in Section IV. An example of the approach is given in Section V. A prototype implementation is described in Section VI followed by related work (Section VII) and a conclusion (Section VIII).

II. Event Logs

The "change your history" approach proposed in this paper heavily relies on detailed event logs as depicted in Table I. In this section, we formalize the key notions used in such logs.

In the remainder, we assume several universes. $\mathcal{U}_{ET} = \{start, complete, schedule, suspend, resume, \ldots\}$ is the universe of event types (providing transactional information). A *start* event signals the start of an activity instance and a corresponding *complete* event signals its completion. $\mathcal{U}_{TS}$ is the universe of *timestamps*. One can measure a duration by taking the difference between two timestamps. $\mathcal{U}_{Res}$ is the universe of *resources*, i.e., people, machines, teams, organizational units, etc. $\perp \in \mathcal{U}_{Res}$ represents a "dummy resource" and is used for automated events that do not require a real resource. $\mathcal{U}_{Attr}$ is the universe of *attribute names*. $\mathcal{U}_{Val}$ is the universe of *attribute values*. Cases, activity instances and events may have any number of attributes represented by name-value pairs, e.g., $(age, 49)$.

Definition 1 (Event Log): An event log $L = (E, AN, AI, C, act, type, time, res, case, name, attr, \preceq)$ is a tuple such that:

- E is the set of *events*,
- AN is the set of *activity names*,
- AI is the set of *activity instances*,
- C is the set of *cases* (i.e., process instances),
- $act \in E \rightarrow AI$ is a surjective function mapping *events* onto *activity instances*,
- $type \in E \rightarrow \mathcal{U}_{ET}$ is a function mapping *events* onto *event types*,
- $time \in E \rightarrow \mathcal{U}_{TS}$ is a function mapping *events* onto *timestamps*,
- $res \in E \rightarrow \mathcal{U}_{Res}$ is a function mapping *events* onto *resources*,
- $case \in AI \rightarrow C$ is a surjective function mapping *activity instances* onto *cases*,
- $name \in AI \rightarrow AN$ is a surjective function mapping *activity instances* onto *activity names*,
- $attr \in (E \cup AI \cup C) \rightarrow (\mathcal{U}_{Attr} \not\rightarrow \mathcal{U}_{Val})$ maps events, activity instances, and cases onto a partial function assigning values to some attributes,[1]

[1] $f \in X \not\rightarrow Y$ is a partial function with domain $dom(f) \subseteq X$.

TABLE I. A FRAGMENT OF SOME SIMPLIFIED EVENT LOG: EACH LINE CORRESPONDS TO AN ACTIVITY INSTANCE.

case id	activity	resource	start time	completion time	...
order1	accept order	Pete	29-01-2015:11.02	29-01-2015:15.12	...
order1	send invoice	Mary	30-01-2015:09.07	30-01-2015:09.13	...
order2	accept order	Pete	30-01-2015:09.08	30-01-2015:14.32	...
order1	handle payment	Mary	30-01-2015:09.14	30-01-2015:10.00	...
order2	send invoice	Mary	30-01-2015:10.08	30-01-2015:11.17	...
...	...	...	...	...	...

- relation $\preceq\ E \times E$ defines a partial order on events,[2] and
- the timestamps respect the partial order: $e_1 \preceq e_2$ implies $time(e_1) \leq time(e_2)$ for any pair of events e_1 and e_2.

$\mathcal{U}_L$ is the universe of all possible event logs.

As Definition 1 shows, an event log consists of *events*, *activities*, and *cases*. As usual, we distinguish between activity *names* (sometimes called tasks) and activity *instances* (i.e., the execution of a task for a particular case). Multiple events can be associated with an activity instance. For example, in Table I every activity instance has a *start* event and a *complete* event. Definition 1 is more general as it allows for additional *event types* (e.g., suspend) and does not require a particular number of events per activity instance. Each activity instance belongs to a case. Note that, transitively, an event also belongs to a case (i.e., $case(act(e))$). Events have a *timestamp* and refer to a *resource*, e.g., the person triggering the event. If $res(e) = \perp$, then no real resource is involved (e is an automated event). Events are partially ordered. Note that the timestamps also induce an ordering and this ordering should respect $\preceq$: $time(e_1) \leq time(e_2)$ if $e_1 \preceq e_2$. Partial order $\preceq$ can be used to express causal dependencies, e.g., maybe $e_1 \preceq e_2$, because e_2 uses data produced by e_1. Cases, activity instances, and events can have additional attributes, but these are not in the focus of this paper.

Our approach assumes that part of the event log is fixed and part of the event log can be modified, e.g., functions $time$ and res can be changed, but the rest of event log $L = (E, AN, AI, C, act, type, time, res, case, name, attr, \preceq)$ may not be changed. This will be formalized using the compatibility notion. In addition, the utility of an event log will defined. However, before doing so we briefly introduce three basic types of process mining.

III. PROCESS MINING

Process mining starts from event data as specified in Definition 1. Note that not all attributes are needed for all types of process mining, e.g., explicit timestamps are only needed for performance analysis and resources are typically not used during control-flow discovery.

A. Process Discovery

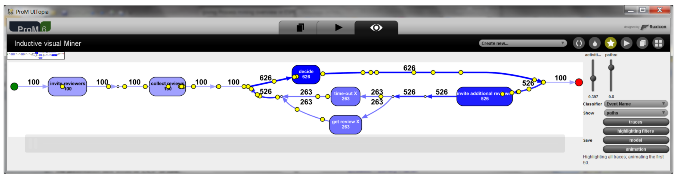
The first type of process mining is *discovery*. A discovery technique takes an event log and produces a process model (e.g., Petri net or BPMN model) without using any a priori information. Process discovery is the most prominent process-mining technique. For many organizations it is often surprising to see that existing techniques are indeed able to discover real processes merely based on example behaviors stored in event logs.

B. Conformance Checking

The second type of process mining is *conformance*. Here, an existing process model is compared with an event log of the same process. Conformance checking can be used to check if reality, as recorded in the log, conforms to the model and vice versa.

C. Performance Analysis

State-of-the-art conformance checking techniques create so-called *alignments* between the observed event data and the process model. In an alignment each case is mapped onto the closest path in the model, also in case of deviations and non-determinism. As a result, event logs can always be replayed on process models. This holds for both discovered processes and hand-made models. Next to showing deviations, alignments can also be used to reveal *bottlenecks* using timestamps of events. Hence, process mining provides novel forms of performance analyses that combine event data with process-centric visualizations.



Classifier Event Name
Show paths
Highlighting all traces; animating the first 50.

Fig. 2. A process model automatically discovered from event data. Replaying the event log reveals the bottleneck.

Figure 2 shows a process model discovered based on an event log. ProM's Visual Inductive Miner [9] was used to discover a block structured model that can be mapped onto Petri nets and BPMN. The model was automatically simplified to show only the most frequent activities. The event log is replayed on the discovered model to show bottlenecks.

The growing number of commercial *process mining tools* (Disco, Perceptive Process Mining, Celonis Process Mining, QPR ProcessAnalyzer, Software AG/ARIS PPM, Fujitsu Interstage Automated Process Discovery, etc.) and the numerous successful applications of process mining illustrate the rising interest in process mining. See for example the twenty *case studies* on the webpage of the IEEE Task Force on Process Mining [10]. However, the state-of-the-art in process mining revolves around identifying performance- or compliance-related problems rather than solving them. Therefore, the next

[2] A partial order is a binary relation that is (1) reflexive, i.e. $e \preceq e$ for any event e, (2) antisymmetric , i.e. if $e_1 \preceq e_2$ and $e_2 \preceq e_1$ then $e_1 = e_2$, and (3) transitive, i.e. $e_1 \preceq e_2$ and $e_2 \preceq e_3$ implies $e_1 \preceq e_3$.

section returns to the event log itself as a starting point for analysis.

IV. Change Your History

An event log $L = (E, AN, AI, C, act, type, time, res, case, name, attr, \preceq)$ (Definition 1) provides a unified representation of the history of a process. *Rather than proposing an improved process model, we aim to create an improved history.* To do this we need to specify which parts of the event log can be modified and how we would like to evaluate the changed history. Therefore, we introduce the notions of *compatibility* and *utility*.

A. Compatibility

It is undesirable to be able to change the event log completely. For example, it is unrealistic to assume that resources can perform all activities and that all activities can be skipped. Therefore, we carefully need to consider which parts of the event log are *invariant* and which parts are *changeable*.

Definition 2 (Compatible): $\sim_c \subseteq \mathcal{U}_L \times \mathcal{U}_L$ is a *compatibility* relation. $L_1 \sim_c L_2$ if and only if event logs L_1 and L_2 are compatible with respect to constraints c.

If $L_1 \sim_c L_2$, then history L_2 can be replaced by a new, hopefully improved, history L_1. The compatibility relation defines what should remain invariant while exploring alternatives. The relation does not need to be symmetric. Section V provides a concrete compatibility relation. When defining compatibility one needs to think about the things that can be realistically influenced and avoid changes that lead to unrealistic scenarios.

B. Utility

Performance measurement is a topic in itself. One can define *Key Performance Indicators* (KPIs) with respect to time, costs, and quality. Consider for example KPIs related to time. One can measure flow times, waiting times, service times, service levels, etc. Costs and quality can also be quantified in different ways. Process improvement is clearly a multi-criteria decision making problem. Hence, one may want to look for Parato optimal decisions, i.e., alternatives that are not dominated by alternatives that are better (or at least as good) in all dimensions.

Here we take a simplified approach and assume that performance can be captured in a single value: *utility* (inverse of a cost).

Definition 3 (Utility): $util_g \in \mathcal{U}_L \to \mathbb{R}$ is a *utility* function. $util_g(L)$ is the utility of event log L given goal g.

Given an event log it is easy to compute a utility function based on time. For example, $util_g(L)$ can be defined as the average flow time of cases in L. To incorporate costs or quality, additional information is needed. For example, costing approaches such as Activity Based Costing (ABC) and Resource Consumption Accounting (RCA) need to assign costs to activities in some way [11]. In [12] a framework for cost-aware process mining is proposed.

C. Creating an Optimization Problem

Based on a compatibility relation and a utility function, one can reason about *improved* and *optimal event logs*. The compatibility relation limits the space of alternative event logs. The utility function helps to select the interesting candidates.

Definition 4 (Improved and Optimal Event Logs): Let $L_1, L_2 \in \mathcal{U}_L$ be two event logs, $\sim_c \subseteq \mathcal{U}_L \times \mathcal{U}_L$ a compatibility relation, and $util_g \in \mathcal{U}_L \to \mathbb{R}$ a utility function.

- L_1 *potentially improves* L_2 with respect to constraints c and goal g (notation: $L_1 \trianglerighteq_{c,g} L_2$) if and only if $L_1 \sim_c L_2$ and $util_g(L_1) \geq util_g(L_2)$,
- L_1 is an *optimal improvement* of L_2 (notation: $L_1 \triangle_{c,g} L_2$) with respect to constraints c and goal g if $L_1 \sim_c L_2$ and for all $L \in \mathcal{U}_L$: $L \sim_c L_2$ implies $util_g(L_1) \geq util_g(L)$.

Definitions 2, 3 and 4 are very generic. Combined with the precise definition of event logs, they form a framework for systematically discussing process improvements at the instance level.

V. Example: Two Types of Shifts

Compatibility and utility are generic, but also rather abstract, notions. To illustrate the "change your history" approach we now define a concrete compatibility relation $\sim_c$ and utility function $util_g$.

First, we define a compatibility relation that keeps everything invariant except the timing of events and the allocation of events to resources.

Definition 5 (Example Compatibility Relation): Let $L_1, L_2 \in \mathcal{U}_L$ be two event logs with $L_i = (E_i, AN_i, AI_i, C_i, act_i, type_i, time_i, res_i, case_i, name_i, attr_i, \preceq_i)$. $L_1 \sim_c L_2$ if and only if the following properties hold:

- $E = E_1 = E_2$, $AN = AN_1 = AN_2$, $AI = AI_1 = AI_2$, $C = C_1 = C_2$, $act = act_1 = act_2$, $type = type_1 = type_2$, $case = case_1 = case_2$, $name = name_1 = name_2$, $attr = attr_1 = attr_1$, $\preceq = \preceq_1 = \preceq_2$,
- for all $e_1, e_2 \in E$ with $AI(e_1) = AI(e_2)$: $time_1(e_1) - time_1(e_2) = time_2(e_1) - time_2(e_2)$ (i.e., time differences within an activity remain unchanged),
- $E^c = \{e \in E \mid case(act(e)) = c\}$ denotes the set of events corresponding to case $c \in C$,
- $E^r_i = \{e \in E \mid res_i(e) = r\}$ denotes the set of events performed by resource $r \in \mathcal{U}_{Res}$ in L_i,
- for all $c \in C$: $\mathbf{min}_{e \in E^c}\ time_1(e) \geq \mathbf{min}_{e \in E^c}\ time_2(e)$ (i.e., cases do not start earlier),
- for all $r \in \mathcal{U}_{Res}$: $\{(name(act(e)), type(e)) \mid e \in E^r_1\} \subseteq \{(name(act(e)), type(e)) \mid e \in R^r_2\}$ (i.e., a resource cannot perform activities it did not perform before), and
- there cannot be four different events $e^s_1, e^c_1, e^s_2, e^c_2 \in E$ such that $res_1(e^s_1) = res_1(e^c_1) = res_1(e^s_2) = res_1(e^c_2)$, $type(e^s_1) = type(e^s_2) = start$, $type(e^c_1) = type(e^c_2) = complete$, $time_1(e^s_2) < time_1(e^c_1)$ and $time_1(e^s_1) <$

$time_1(e_2^c)$ (i.e., the same resource cannot work on multiple activities at the same time).

Note that the example compatibility relation explicitly considers activity instances with a *start* and *complete* event. All log elements are invariant except the resource allocation and the times at which events occur. Note that:

- The time differences between events in the same activity instance cannot change. This implies that durations (time difference between start event and complete event) do not change.
- Cases cannot start earlier, i.e., the first event of a case cannot be moved to an earlier time (only a later time is allowed).
- Resources cannot perform activities they did not perform before (taking into account event types). For example, if resource r never completed an activity a in the original event log, then r cannot complete a in the modified log. Also note the role of $\perp \in \mathcal{U}_{Res}$ here, i.e., things that always required a resource in the old log, also require a resource in the new log.
- In the modified event log, the same resource can only perform one activity at a time. Here start and completion events are used to demarcate the activity instances.

An example of a utility function is the average flow time.

Definition 6 (Example Utility Function): Let $L = (E, AN, AI, C, act, type, time, res, case, name, attr, \preceq)$ be an event log. $util_g(L) = \sum_{c \in C}(\mathbf{max}_{e \in E^c}\ time_1(e) - \mathbf{min}_{e \in E^c}\ time_2(e))/|C|$ for $L \in \mathcal{U}_L$.

The start time of a case is assumed to be the timestamp of the first event of the case in the original event log (i.e., $\mathbf{min}_{e \in E^c}\ time_2(e)$). The completion time of a case is the timestamp of the last event of the case in the modified event log (assuming activity instances having just a *start* and *complete* event).

Definitions 2, 3 and 4 provide a generic framework that can be instantiated as just described. However, many alternatives are possible [13]. Moreover, it is important to visualize the key differences between the original event log L_2 and the improved or optimal event log L_1. Improvements can be shown at the process (i.e., type) level and at the instance level. This is illustrated in the next section.

VI. Prototype Implementation

Multiple ProM plug-ins were developed to illustrate the proposed approach. Bearing in mind the goal of cost reduction, a hybrid genetic algorithm approach was developed and used together with a utility function that takes into account trade-offs between time, cost and resource utilization to generate an improved event log which is compatible with a given event log [13]. Everything in the improved event log is kept invariant except the timing of events and the allocation of events to resources. Various visualizations have been created to highlight key differences between two compatible logs in order to gain insights into potential improvement scenarios.

The change in the average start times of events for all activities in the two logs at the process level is visualized in Figure 3. Each bar in a row represents an activity, and its average difference in start times across all cases that contain an instance of that activity. A green bar that extends towards the left shows that, on average, instances of the activity tend to start earlier. A red bar that extends towards the right shows activity instances starting later. The configuration panel on the left enables the user to filter the cases and activities of interest and to sort the activities. The statistics panel on the right provides detailed information about the timing differences. Figure 3 shows that instances of the activities *APT* and *CWR* on average could have started a lot earlier.

Figure 4 visualizes the time differences in activities within each case and compares these differences at the instance level. Each twin-row represents the “before” and “after” behavior of a case in terms of execution times. The activity instances that were executed within the case are portrayed by bars, and are distinguishable by their colors. By observing Figure 4, it is clear that the throughput time of *Case 144* can be significantly shortened by minimizing idle times between activity instances in that case. This is made possible due to under-utilized and free resources being reallocated to undertake these activity instances in the improved log.

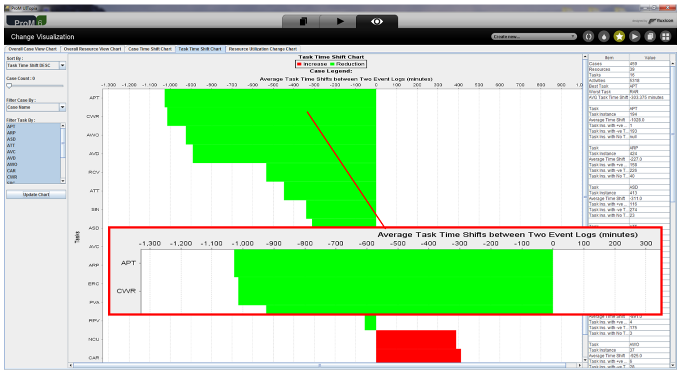


Fig. 3. An example visualization that aggregates the changes in average start times for activities between two event logs.

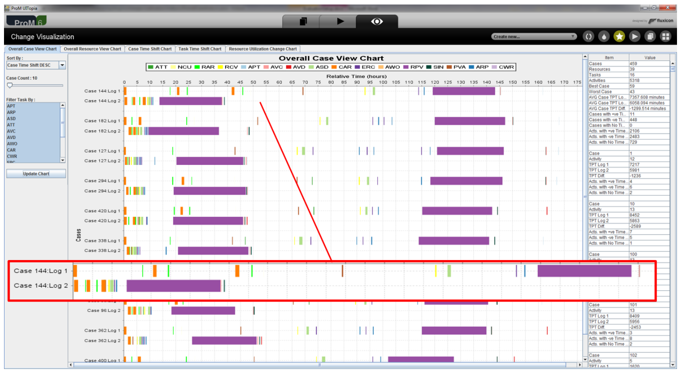


Fig. 4. An example visualization that outlines the timing differences of cases between two event logs.

In order to examine the changes in resources executing various activities between two compatible logs, Figure 5 visualizes the resources as nodes and the transfer of activities between resources as directed arcs between these nodes. For the resource that a node represents, the size of the node shows the total number of activity instances that did not have their resource reallocated, and the colored segments group the activity instances into their respective activities. The configuration panel on the left allows the user to customize the visualization by filtering the activities or by adjusting the details displayed on the graph. In Figure 5, it can be observed that a number of activity instances were shifted from *IS2* to *IS1* and from *SC1* to *SC2*. It is also possible to observe the stability of a resource by comparing the activities executed by the resource in the original log with those retained by the same resource in the improved log. The visualization provides us with insights into possible improvements in workload allocation. These insights can be used to better utilize the available resources in an organization.

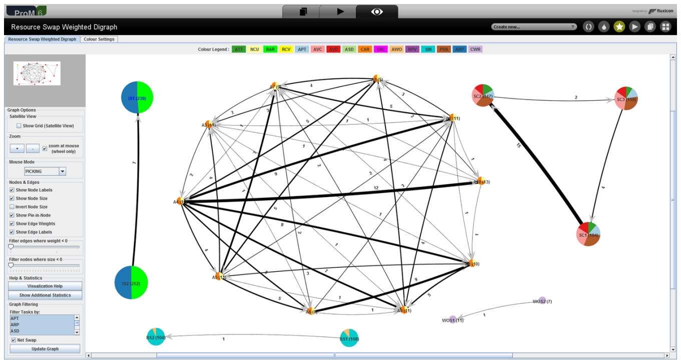


Fig. 5. An example visualization that highlights resource shifts between two event logs.

Through these ProM plug-ins, we explored techniques to generate improved event logs in order to provide evidence at the instance level of how processes could be improved by shifting start times of activities and by allocating a different resource to work on an activity. In order to better understand the key differences between the two logs, we developed multiple visualizations that highlight the shifts in time and changes in resource allocations between two compatible event logs. These insights can then be used to facilitate discussions around possible process improvements within the organization.

VII. Related Work

See [3] for an introduction to process mining. The XES standard is described in [8]. The extraction of event data is discussed in Chapter 4 of [3] and in [14].

As mentioned, our approach complements process mining [3], simulation [4], predictive analytics [6], and combinatorial optimization [7]. It is also related to *comparative process mining*. For example, the notion of "process cubes" can be used to systematically compare subsets of event data [15].

Some initial work has been done based on the notions discussed in this paper [16], [13], [17]. For example, [13] describes a hybrid genetic algorithm that explores alternatives in a smart manner thus allowing for larger search spaces. In [17] different visualizations are described (cf. Section VI).

VIII. Conclusion

This paper presented a generic framework for reengineering event logs in a controlled manner. By exploring different alternatives at the instance level, stakeholders are able to reflect and learn. This is similar to soccer players analyzing historic matches and practicing particular game situations.

Future work aims at providing a wide range of compatibility relations ($\sim_c$) and utility functions ($util_g$), and corresponding change visualizations. Moreover, we aim to efficiently prune the search space using monotonicity results for processes. This is needed to make the approach feasible in large-scale real-life settings.

References

[1] M. Grootjans, "Visualization of Spatio-Temporal Soccer Data", Master Thesis, Eindhoven, 2014.

[2] M. Lewis, *Moneyball: The Art of Winning an Unfair Game*. W. W. Norton & Company, 2003.

[3] W. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin, 2011.

[4] ——, "Business Process Simulation Survival Guide," in *Handbook on Business Process Management*, J. Brocke and M. Rosemann, Eds. Springer-Verlag, Berlin, 2015, pp. 337–370.

[5] W. van der Aalst, A. Adriansyah, and B. van Dongen, "Replaying History on Process Models for Conformance Checking and Performance Analysis," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 182–192, 2012.

[6] E. Siegel, *Predictive Analytics: The Power to Predict Who Will Click, Buy, Lie, or Die*. Wiley, 2013.

[7] G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization*. Wiley, 1988.

[8] IEEE Task Force on Process Mining, "XES Standard Definition," www.xes-standard.org, 2013.

[9] S. Leemans, D. Fahland, and W. van der Aalst, "Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour," in *International Workshop on Business Process Intelligence (BPI 2013)*, LNBIP, vol. 171. Springer-Verlag, Berlin, 2014, pp. 66–78.

[10] IEEE Task Force on Process Mining, "Process Mining Case Studies," http://www.win.tue.nl/ieeetfpm/doku.php?id=shared:process_mining_case_studies, 2013.

[11] B. Clinton and A. van der Merwe, "Management Accounting: Approaches, Techniques, and Management Processes," *Cost Management*, vol. 20, no. 3, pp. 14–22, 2006.

[12] M. Wynn, W. Low, A. ter Hofstede, and W. Nauta, "A framework for cost-aware process management: Cost reporting and cost prediction," *Journal of Universal Computer Science*, vol. 20, no. 3, pp. 406–430, 2014.

[13] W. Low, J. De Weerdt, M. Wynn, A. ter Hofstede, W. van der Aalst, and S. Broucke, "Perturbing Event logs to Identify Cost Reduction Opportunities: A Genetic Algorithm-Based Approach," in *IEEE Congress on Evolutionary Computation (CEC 2014)*. IEEE Computer Society, 2014, pp. 2428–2435.

[14] W. van der Aalst, "Extracting Event Data from Databases to Unleash Process Mining," in *BPM: Driving Innovation in a Digital World*, J. vom Brocke and T. Schmiedel, Eds. Springer-Verlag, 2015, pp. 105–128.

[15] ——, "Process Cubes: Slicing, Dicing, Rolling Up and Drilling Down Event Data for Process Mining," in *Asia Pacific Conference on Business Process Management (AP-BPM 2013)*, ser. Lecture Notes in Business Information Processing, M. Song, M. Wynn, and J. Liu, Eds., vol. 159. Springer-Verlag, Berlin, 2013, pp. 1–22.

[16] ——, "Log-Based Cost Analysis and Improvement," technical Note, January 29, 2013.

[17] W. Low, M. Wynn, A. ter Hofstede, J. De Weerdt, and W. van der Aalst, "Change Visualisations: Analysing the Resource and Timing Differences between Two Event Logs," technical Note, January 15, 2015.