Event Interval Analysis: Why Do Processes Take Time?

Suriadi Suriadi[c,a,*], Chun Ouyang[a], Wil M. P. van der Aalst[b,a], Arthur H. M. ter Hofstede[a,b]

[a] *Queensland University of Technology, GPO Box 2434, Brisbane, Australia*
[b] *Eindhoven University of Technology, Eindhoven, The Netherlands*
[c] *Massey University, Albany, New Zealand*

Abstract

Through the application of process mining, valuable evidence-based insights can be obtained about business processes in organisations. As a result, the field has seen an increased uptake in recent years as evidenced by success stories and increased tool support. However, despite this impact, current performance analysis capabilities remain somewhat limited in the context of information-poor event logs. For example, natural daily and weekly patterns are not considered but they are vital for understanding the performance of processes and resources. In this paper, a new framework for analysing event logs is defined. Our framework is based on the concept of *event interval*. The framework allows for a systematic approach to sophisticated performance-related analysis beyond the capabilities of existing log-based analysis techniques, even with information-poor event logs. The paper formalises a range of event interval types and then presents an implementation as well as an evaluation of the proposed approach.

Keywords: process mining, ProM, data mining, business process management

1. Introduction

Process mining [21] aims to exploit the massive amount of event data recorded by today's information systems to gain valuable insights into business processes by unearthing, among others, actual process behaviour, model deviations, performance characteristics, and bottlenecks. The use of process mining in practice is becoming more and more widespread as confirmed by the growing number of success stories of its application [17, 9, 14, 25] and the increasing number of tools offering process mining capabilities [8] (e.g. ProM [26], ARIS[1], Fluxicon[2], Bizclarity[3]).

The practical application of process mining is often hampered by the limited information available in events logs, especially in those logs which are not generated by process-aware information systems. A typical problem is that not both start and complete times are recorded for activities. This information is expected by current process mining software, such as the *performance analysis with Petri nets* and *alignment-based performance analysis* plug-ins of the ProM environment [7, 20, 18, 2] and the performance analysis component of Disco (a commercial tool), in order to derive metrics such as waiting times and case utilisation. The reliance on the existence of a *clean and simple* process models to derive more detailed performance information is also not very realistic as they are often not realizable in practice. Furthermore, existing approaches, such as the fuzzy-mining-based performance analysis approaches [23, 1, 5, 6, 16], and the temporal trace language approach [13], analyse process performance

*Corresponding author
Email addresses: s.suriadi@massey.ac.nz, +64-9-414 0800 ext 43581 (Suriadi Suriadi), c.ouyang@qut.edu.au (Chun Ouyang), w.m.p.v.d.aalst@tue.nl (Wil M. P. van der Aalst), a.terhofstede@qut.edu.au (Arthur H. M. ter Hofstede)

[1] www.softwareag.com/corporate/products/aris_platform/aris_controlling/aris_process_performance/overview/

[2] www.fluxicon.com/disco

[3] www.bizclarity.com.au

separately from resource performance, thus preventing the discovery of certain types of insight, such as idle periods for processes and resources. Our approach proposed in this article, on the other hand, is able to combine these two perspective, thus allowing us to extract insights about the idle periods of processes. Identifying idle periods of processes is important as the ability to minimize them is likely to lead to an increase in work throughput.

While existing data mining technology and spreadsheet tools can, to a certain extent, derive performance metrics, these tools and techniques *do not take into account the temporal and resource constraints that are inherent within any type of process-based analysis* (more details in Section 2.1). The approach proposed in this article is specifically developed to extract performance insights that take into account those constraints properly.

In summary, *despite the proliferation of process mining algorithms, especially in the area of process discovery, emphasis in the field has been on comparatively basic performance metrics* (e.g. throughput times, working times, and waiting times) *and not so much on analysis of more advanced performance-related behaviour of processes* (e.g. resource productivity trends, correlation of performance with log variables, workload estimation, and discovery of and comparison between working patterns of resources).

This paper proposes a novel performance analysis framework based on the concept of *event intervals*. An event interval is a time interval between two adjacent events that fulfils certain conditions. By choosing the concept of event interval as the building block of the approach and by considering a variety of such intervals, problems with varying degrees of information in logs (e.g. in relation to time stamps) can be addressed in a *systematic* manner. A range of different types of event intervals is distinguished where the type is determined by attribute values of the events for the case involved, the resource involved, and the type of transaction (e.g. started, completed, scheduled). Depending on the kind of information available in a log, some types of event intervals may or may not apply. By taking into account both the case and the resource perspectives, we are able to tackle more complex forms of performance-related analysis, such as recurring working patterns of resources, waiting time distributions over time, and resource performance comparison. In addition, the framework allows the discovery of correlations between event interval durations and attributes recorded in a log. Equally important, our approach has been implemented in the open-source process mining analysis tool ProM to automate the various types of analyses just mentioned, thus, delivering a *dedicated framework, with tool-support, for extracting sophisticated and refined process-related performance information from information-poor logs*.

This paper is structured as follows. Section 2 describes and formalizes the event interval analysis approach, while Section 3 describes an implementation of this approach as a plug-in of ProM [24]. Section 4 details the evaluation of the approach and Section 5 compares our approach with a number of existing performance analysis approaches. Section 6 summarises the paper and presents potential future work.

2. Approach

We start this section with an introduction to process mining which is the background knowledge necessary to understand the approach we propose in the paper. This is then followed by discussions about the rationale and the main ideas behind our approach and finally detailed explanation and formalization of the approach.

2.1. Background: Process Mining

The starting point of any process mining analysis is an *event log*, a sample of which is shown at the top part of Figure 1. An event log consists of a set of events and each row in an event log corresponds to a single event. For illustration purposes, the log in Fig. 1 captures the events related to a purchase order process. A *process* can be interpreted

as the execution of a sequence of activities that follow a certain order to achieve a particular goal, e.g. to handle a purchase order or an insurance claim. A process definition is often specified in the form of a *process model* using a well-defined modelling language. For example, the process shown in the middle-left part of Fig. 1 was described in terms of a Petri net. A *case* refers to one particular execution (or instance) of a process, which has or is assigned a unique identifier ('caseID'). In the sample event log in Fig. 1, we can see that there are three cases, each is made up of 3-4 events. An *event* contains information concerning the *activity* that is captured by the event, the *resource* that performs an action which triggers the occurrence of the event, the *timestamp* of the event, and the *transaction type* of the event.

An event's *transaction type* captures the 'state' of the activity when the event occurred. Examples include 'create' (when the activity becomes available to be executed), 'assign' (when the activity is given to a particular resource to perform), 'start' (when the assigned resource starts the execution of the activity), and 'complete' (when the resource completes the activity). The caseID, activity, resource, timestamp, and transaction type are often called *event attributes*. Finally, an instance of a particular activity that belongs to a particular case is referred to as a *work item*, and a list of work items is known as a *worklist* which is usually associated with a particular resource.

Process mining takes an event log to extract insights about *how the process was actually executed*. With the support of existing process mining techniques, an event log can be used to extract temporal patterns formed by the events in the log and thus to build (the structure of) a process model (which is known as process discovery). For example, the process model in Fig. 1 may be discovered from an event log of which a short snippet is shown in this figure. Performance analysis is another important area in which process mining techniques have been proven to be useful. An event log often contains information relating to other perspectives such as the case, the time, and the resource perspectives. Mining such information, in the context of business process, can provide insight into the understanding of different perspectives. For example, log data such as timestamps and frequencies of activities can be used to identify bottlenecks, throughput, and efficiency of a process. Furthermore, data about activities being carried out by certain resources can be used to analyse working behaviour of resources.

A distinct characteristic of the problems addressed by process mining, which differentiates the process mining discipline from other types of data analysis, is the existence of concurrency and *temporal constraints* among events, both from a case perspective and a resource perspective. The former restricts the types of work items that can be executed at any given point in time (as they are dependent on the completion of the preceding activities), while the latter restricts the resources who can execute certain work items and the time at which the work items can be executed, depending on task-resource assignment and resource availability respectively. Consequently, each row in an event log (i.e. an event) has temporal relationships with other events, through resource and case constraints. This is in contrast to other types of logs typically used in traditional data analytics (such as data mining or even basic spreadsheet analysis) whereby the concept of case or resource temporal constraint does not exist or is not taken into account.

2.2. Rationale: Reasoning about Time in Processes

In this paper we are interested in process performance analysis using event logs. The existing process mining techniques focus on the analysis of basic performance metrics (e.g. working time, waiting time, and cycle time/throughput) and assume that an event log contains sufficient transactional information, e.g. both 'start' and 'complete' information, of activities. Basically, working time of an activity a_x is the time elapsed from the start to the completion of

the activity a_x, waiting time of a_x is the time elapsed from the completion of the last activity that preceded a_x to the start of the activity a_x, and cycle time of a_x is comprised of both the working time and waiting time of a_x.

In reality an event log, however, often records limited transactional information of an activity. For example, the event log shown in Fig. 1 only contains information about activity completion, and we cannot see when the activity started. With the existing process mining techniques, we can only extract, from such a log, the information of cycle time (i.e. the time interval between the completion of two subsequent activities) with little knowledge of the working time and waiting time. This prevents us from carrying out a range of typical performance-related analyses, such as analysis of process (cycle time) efficiency and resource utilisation. Hence, the following question arises:

- If an event log contains limited activity transactional information, e.g. either 'start' or 'complete' information only, is it still possible to extract rich and meaningful insights about process performance from the log?

Based on business process execution principles, we exploit an important observation that for a work item to be executable in a case, two conditions must hold: 1) the progress of the case must reach a state where the work item itself is available for execution, and 2) the resource responsible for carrying out the work item must be available. The former condition can be checked by focusing on the case perspective, while the latter requires to take into account the resource perspective.

Again, consider the example in Fig. 1. Let's pick an arbitrary timestamp, say 7 Jan 2014 at 14:30:00.[4] On 7 Jan 2014 at 14:30:00, we can deduce that the work item

[4]Note that this timestamp is not among those listed in Figure 1 because it is a rather 'random' point in time. The log shown in Figure 1 only shows timestamps of completed work items.

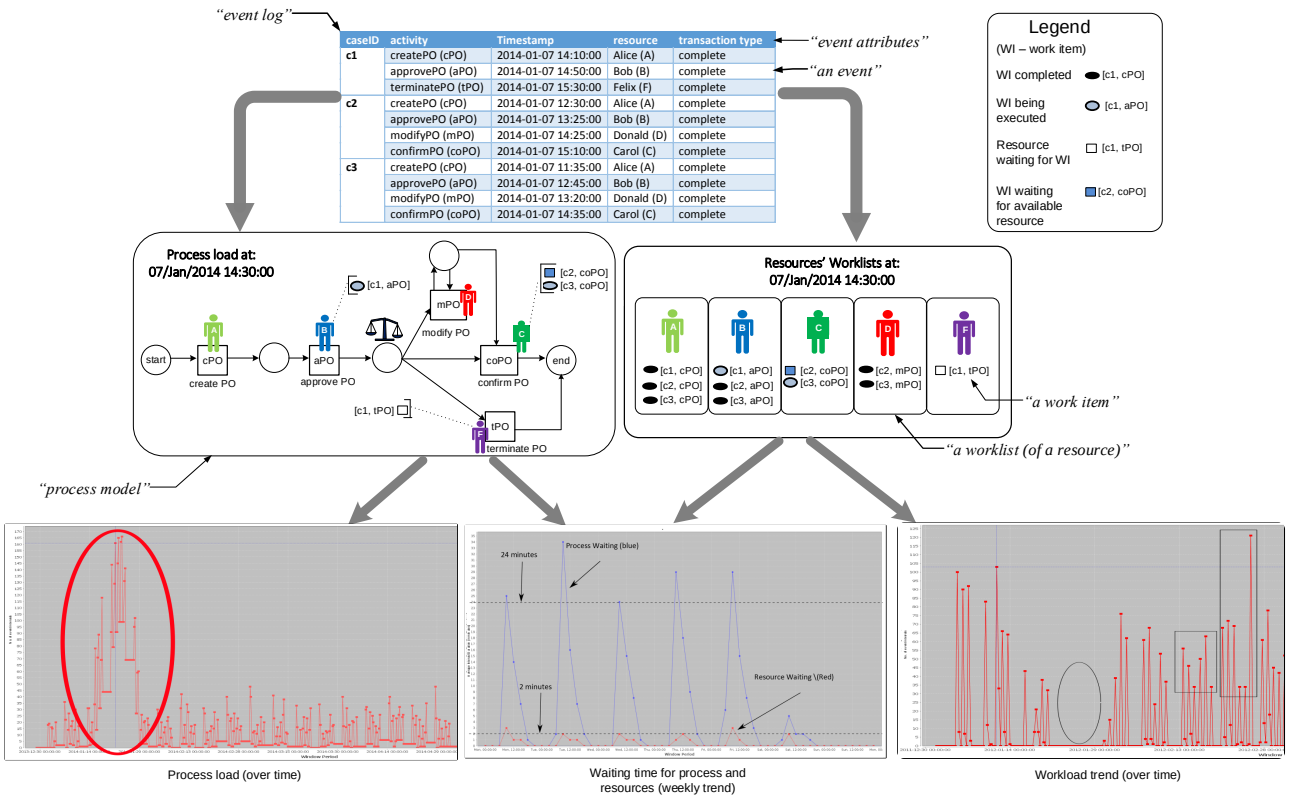

Figure 1: Process mining: an example of event log, process model, resources' worklists, and performance analysis

'approvePO' of case c_1 (i.e. [c_1, aPO]) is most likely being executed because 1) it is available upon the completion of the preceding work item 'createPO' ([c_1, cPO]) at 14:10:00 and 2) the resource 'Bob' (B), who is responsible for 'approvePO', is available upon the completion of 'approvePO' in case c_2 ([c_2, aPO]) at 13:25:00.

Now let us discuss a couple of scenarios where the case and resource constraints apply. Firstly, since work item [c_1, aPO] is still in progress at 14:30:00, none of the subsequent activities (i.e. 'modifyPO', 'confirmPO', or 'terminatePO') can be executed in case c_1. As a result, resource 'Felix' (F), who is responsible for 'terminatePO' and is free at 14:30:00, has to *wait* until the completion of work item [c_1, aPO] at 14:50:00 before he can execute work item [c_1, tPO]. Resource 'Carol' (C), who is responsible for 'confirmPO', is working on case c_3 (on work item [c_3, coPO]) at 14:30:00 and thus is not available for other cases. As a result, although work item 'confirmPO' of case c_2 ([c_2, coPO]) has been available for execution since 14:25:00 (when the preceding activity 'modifyPO' was completed), it has to *wait* until 'Carol' completes work item [c_3, coPO] at 14:35:00. In summary, the first scenario reveals that there is a waiting time of at least 20 minutes for resource 'Felix' before he can execute [c_1, tPO] due to a case constraint, while the second scenario reveals that there is a waiting time of at least 10 minutes in case c_2 before work item [c_2, coPO] can actually be executed due to resource constraints.

What we can learn from the above scenarios is that even in the absence of activity start (or complete) information, it is still possible to derive meaningful insights about the waiting time and working time of a resource or a process activity by *examining the various points of time in the log data and the intervals between these time points from both the case and resource perspectives*. This finding provides the underlying rationale for the proposal of a new log data analysis approach that can be used to extract important information about process performance that is hidden in an (information-poor) event log. Furthermore, we also hope to explore the potential of this approach for supporting sophisticated performance-related analysis beyond the capabilities of current process mining techniques. For example, by taking multiple 'snapshots' of both the process and resource load over time, we could build a picture that describes the changes in the load of the processes (see bottom-left graph in Fig. 1) and the resources (bottom-right graph). Another example shown in the bottom-middle graph of Fig. 1 shows how we can observe the variation of process waiting time and the resource waiting time over a recurring period of time (such as a day or week) to discover the daily or weekly pattern of the waiting times for both processes and resources.

2.3. Approach: Described in More Detail

The central notion used in this paper is that of an *event interval*. Event intervals can be calculated from *event logs*. In essence, an event interval is a period of time between the occurrences of two events that satisfy certain conditions, e.g. same transaction type (e.g. 'start' or 'complete'), same case identifier, and/or same resource identifier. Different types of event intervals can be derived by imposing different set of conditions. Furthermore, depending on the conditions imposed, one can also derive meaningful interpretation in terms of work performance by the collection of event intervals that exist in an event log.

For example, one can define a type of event interval where the events involved concern activities which are performed by the same resource. This type of event interval is referred to as a *resource interval*. As an illustration consider the interval between events e_1 and e_2 in Fig. 2. The figure depicts an elaboration of the second scenario discussed in Section 2.2, where the relevant events are displayed in ascending order along a timeline. Events e_1 and e_2 capture the completion of activity 'confirmPO' by resource 'Carol' in two *different* cases c_3 and $c2$, respectively. Note that there is *no event* involving 'Carol' that occurred

in between the occurrences of e_1 and e_2. Under the assumption that a resource does not work on multiple activities at the same time, one can interpret the period represented by this event interval as the *maximum* time that it could have taken 'Carol' to perform activity 'confirmPO' in case c_2. This maximum would actually be realised if 'Carol' started to work on 'confirmPO' in case c_2 immediately after completing the activity in case c_3, i.e. immediately after the occurrence of event e_1. The example further illustrates that even in the context of an information-poor event log (only the completions of activities are recorded) one can still derive meaningful information about resource performance using the notion of a specific type of event interval.

The main driver behind our approach is that *the existence of various types of intervals in an event log may reveal important information about process performance.* In fact, by displaying various types of intervals unearthed from an event log over a timeline, information can be inferred that cannot be obtained through the application of current process mining approaches. To illustrate this phenomenon, consider another type of event interval where the timestamps of the events involved indicate the completion of their activities and where these activities belong to the same case. This type of event interval is referred to as a *case interval*. Let's revisit Fig. 2 where events e'_1 (capturing the completion of activity 'modifyPO' in case c_2) and e_2 form a case interval as both events belong to the same case, c_2, and there are no intermediate events belonging to this case. Given that e'_1 occurred before e_1, it can be inferred that the time difference between events e'_1 and e_1 corresponds to a waiting time for case c_2. This can be explained as follows. Immediately after the occurrence of event e'_1, the next activity ('confirmPO') in case c_2 could start were it not for the fact that this activity is to be performed by 'Carol' who is still occupied with case c_3. The completion of 'confirmPO' by 'Carol' in case c_3 is signalled by the occurrence of event e_1, hence case c_2 was delayed after the occurrence of event e'_1 till the occurrence of event e_1.

It should be noted that in the current version of event interval analysis, as presented in this paper, we make a couple of assumptions with regards to the way in which work items are created and the way resources execute work items. The analysis focuses on activities within a log that have causal relationships, and considers resources working on one work item at any given time. In case that these assumptions are violated, our event interval analysis framework will still be able to deliver meaningful insights, however, their interpretation would become more involved.

Having created an intuition behind the notion of event interval and having made explicit our assumptions, we are in a position to lay the formal foundation for event interval analysis by providing definitions for the notion of event log and for five basic types of event intervals, each of which captures a different performance perspective. The five types of event intervals form the basic building blocks of our approach to performance analysis. More types of

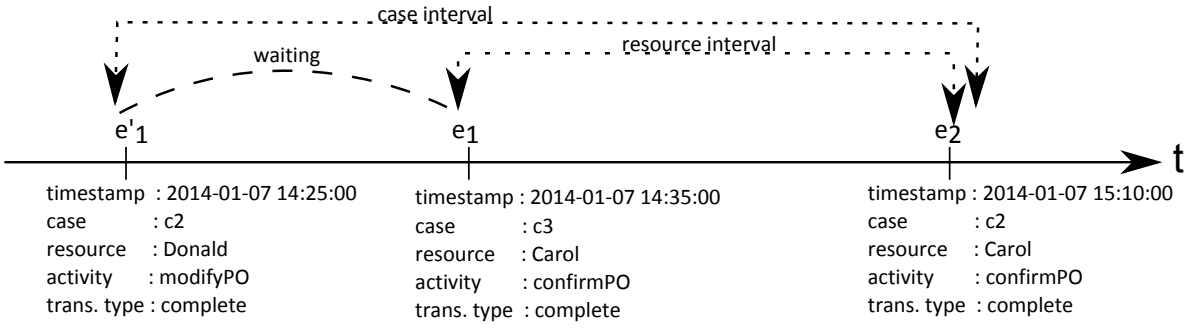

Figure 2: An illustrative example of event intervals (in the context of the event log shown in Fig. 1)

intervals can be defined for event logs that capture more transaction types (e.g. create, assign) leading to richer forms of analysis. The approach can thus be seen as a framework providing a range of analysis techniques tailored to the type of information present in a log.

2.3.1. Event Log

An *event log* ($\mathcal{L}$) consists of a set of *events*. Each event (e) is associated with a range of attributes capturing the information of caseID ($case(e)$), activity ($act(e)$), timestamp ($time(e)$), resource ($res(e)$), transaction type ($type(e)$), etc.[5] An event log can be partitioned into smaller event logs based on an event attribute. For example, an *event-typed log* ($\mathcal{L}_{tp}$) is the result of partitioning the original event log based on transaction type tp and hence it consists of events that have the same transaction type. Also, events are ordered by unique numbers, which are referred to as *event order identifier*s, that are related to their timestamps, and numbers take precedence over timestamps where two events occur concurrently. Below we formally define event and event log to avoid any ambiguity that may arise when introducing the relevant concepts.

Definition 1 (Event). Let *Case* be the set of case identifiers, *Act* the set of activity identifiers, *Time* the set of possible timestamps, *Res* the set of resource identifiers and *Type* the set of transaction types (e.g. create, assign, start and complete).

$\mathscr{E}$ is the set of events.

For any $e \in \mathscr{E}$: $case(e) \in Case$ is the case identifier of e, $act(e) \in Act$ is the activity identifier of e, $time(e) \in Time$ is the timestamp of e, $res(e) \in Res$ is the resource identifier of e and $type(e) \in Type$ is the transaction type of e. If an attribute is missing, a $\perp$ value is returned, e.g., $res(e) = \perp$ means that no resource is associated with event e. □

[5] An event can have more attributes but these are not considered in this paper.

Definition 2 (Event Log). An event log $\mathcal{L} \subseteq \mathscr{E}$ is a set of events. □

Definition 3 (Event-typed Log). Let $\mathcal{L} \subseteq \mathscr{E}$ be an event log and let *Type* be the set of transaction types. Given $tp \in Type$, $\mathcal{L}_{tp} = \{e \in \mathcal{L} \mid type(e) = tp\}$ is an event-typed log of transaction type tp. □

Definition 4 (Event Order Identifier). Given an event log $\mathcal{L} \subseteq \mathscr{E}$, $id : \mathcal{L} \rightarrow \{1, ..., |\mathcal{L}|\}$ is a bijective function that maps each event $e \in \mathcal{L}$ to a unique natural number where $\forall e_1, e_2 \in \mathcal{L} : id(e_1) < id(e_2)$ if $time(e_1) < time(e_2)$ and $e_1 \leq e_2$ if and only if $id(e_1) \leq id(e_2)$. □

2.3.2. Event Intervals

The general notion of an *event interval* (e_1, e_2) (or *interval* for short) refers to the interval between any two events in a log. More specifically, we can identify five basic types of event intervals.

Case Interval (cg). This is an interval between two adjacent events of the same transaction type in the *same case*. Given an event e, the case interval cg_e captures the maximum period the activity $act(e)$ of $case(e)$ becomes available until it is completed.

Consider the example of a case interval in Fig. 3(a). At the moment of $time(e_1)$, activity $act(e_1)$ of case c_2 is completed and the subsequent activity $act(e_3)$ is available for execution. The latest this remains in the case is the moment when $act(e_3)$ is completed, which is $time(e_3)$. Hence, event interval (e_1, e_3) defines case interval cg_{e_3}.

A special scenario that should be considered is when a case appears for the first time, i.e. the starting point of the case, in an event log. This is when an initial case interval is defined. For example, in Fig. 4, event e_1 signals the starting point of case c_2, and the initial case interval of c_2, written $cg^i_{e_1}$, is formed by a pair comprising event e_1 and itself, i.e. (e_1, e_1). As another example, event interval (e_3, e_3) defines the initial case interval of c_3, written $cg^i_{e_3}$. Obviously, an initial interval has a zero time duration.

Resource Interval (rg). This is an interval between two adjacent events of the same transaction type that involve the *same resource*. Given an event e, the resource interval rg_e captures the maximum period the resource $res(e)$ is available for performing the activity $act(e)$ in $case(e)$ under the assumption that a resource does not work on more than one activity at a time.

Consider the example of a resource interval in Fig. 3(a). At the moment of $time(e_2)$ when resource r_2 completes the activity $act(e_2)$ of case c_1, r_2 is then ready to carry out activity $act(e_3)$ of case c_2. The latest this holds true is the moment of $time(e_3)$, when r_2 completed $act(e_3)$ in c_2. Hence, event interval (e_2, e_3) defines resource interval rg_{e_3}.

Similarly to case interval, an initial resource interval is defined when a resource appears for the first time in an event log. Again, consider the illustration in Fig. 4. Event e_1 indicates the first appearance of resource r_2, and the initial resource interval of r_2, written $rg^i_{e_1}$, is interval (e_1, e_1); and for resource r_1, the initial resource interval $rg^i_{e_2}$ is interval (e_2, e_2).

Working Interval (wg). Given an event e, the working interval wg_e captures the period the activity $act(e)$ of $case(e)$ is performed by a specific resource $res(e)$. It starts from the moment when the activity and the resource are both available for the execution of that activity, and *finishes* the moment when the resource completes the activity.

A working interval is represented by either a case interval or a resource interval depending on which becomes available *later*, the activity or the resource. For example, in Fig. 3(a), though activity $act(e_3)$ of case c_2 is ready to be performed at $time(e_1)$, resource r_2 is not available until later at $time(e_2)$. Hence, the event interval (e_2, e_3) defines working interval wg_{e_3} as well as the corresponding resource interval rg_{e_3}. In Fig. 3(b), on the other hand, activity $act(e_3)$ of case c_2 becomes available later than resource r_2, and thus working interval wg_{e_3} is the same as the corresponding case interval cg_{e_3}.

In the situation when only *completion*s of activities are recorded, a working interval provides us with 1) an upper bound for the amount of time that the resource

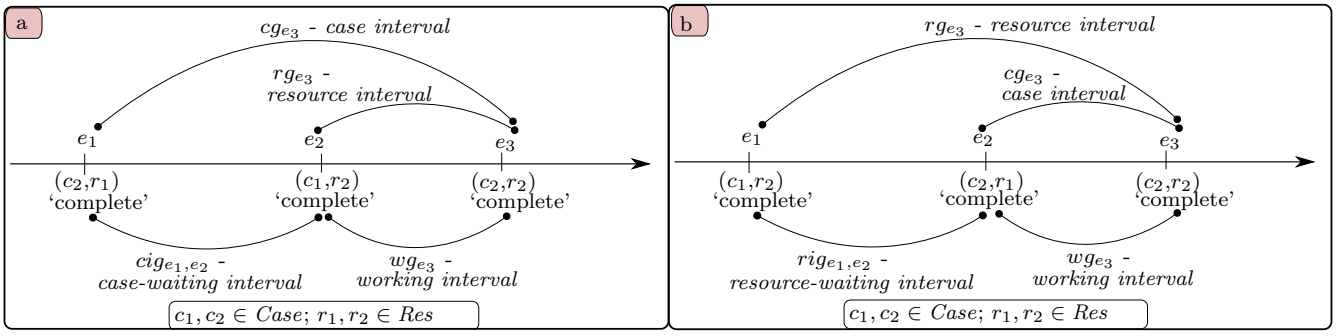

Figure 3: Illustration of five basic types of event intervals in two different scenarios (a) and (b)

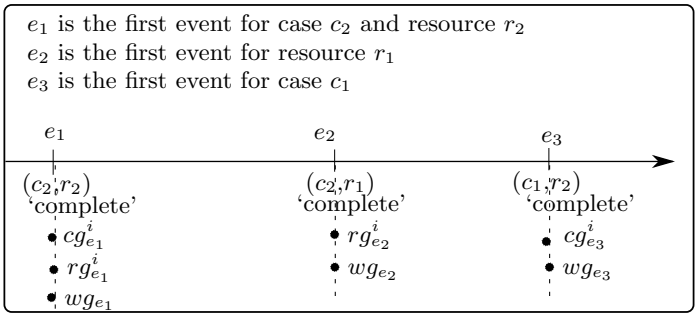

Figure 4: Illustration of initial case intervals, initial resource intervals, and the corresponding working intervals

worked on the activity (the maximum occurs when the resource starts to work on the activity immediately after the recorded completion of the previous activity), and 2) an upper bound for the amount of time that the resource is idle, i.e. able to work on the activity but not actually doing so (this maximum occurs when the resource started working on the activity just before its completion is recorded).

A special scenario is, when either a case interval or a resource interval is an initial interval, the corresponding working interval is specified by that initial interval. Examples can be seen as wg_{e_1}, wg_{e_2}, and wg_{e_3} in Fig. 4.

Case Waiting Interval (*cig*). This captures the period an activity of a certain case that is ready to be performed waits for the availability of the corresponding resource. It *starts* from the moment when only the activity is ready and *ends* the moment when the resource is also ready.

In Fig. 3(a) cig_{e_1,e_2} is an example of a case waiting interval during which activity $act(e_3)$ of case c_2 waits for resource r_2 to become available (from case c_1).

Resource Waiting Interval (*rig*). This captures the period that an available resource waits for the corresponding activity of an expected case to be ready for its performance. It *starts* from the moment when only the resource is available and *ends* the moment when the activity in the corresponding case also becomes available.

In Fig. 3(b) rig_{e_1,e_2} is an example of a resource waiting interval during which resource r_2 waits for activity $act(e_3)$ of case c_2 to become available.

Finally, it is possible that both the activity of a certain case and the required resource become available at the same time. In this situation, case interval, resource interval, and working interval all correspond to the same event interval, and both case waiting interval and resource waiting interval have a zero time duration.

Formal Definitions. Below we formally define the five basic types of event intervals to avoid any ambiguity that may arise in the above informal description.

Definition 5 (Event Interval). Let $\mathcal{L} \subseteq \mathscr{E}$ be an event log. Let $e_1, e_2 \in \mathcal{L}$ with $e_1 \leq e_2 : (e_1, e_2)$ is an event interval. $\mathscr{G} = \{(e_1, e_2) \in \mathcal{L} \times \mathcal{L} \mid e_1 \leq e_2\}$ is the set of all possible event intervals, and $dur(e_1, e_2) = time(e_2) - time(e_1)$ is the (time) duration of interval (e_1, e_2). □

Definition 6 (Case Interval). Let $\mathcal{L} \subseteq \mathscr{E}$ be an event log and $\mathcal{L}_{tp} \subseteq \mathcal{L}$ (where $tp \in Type$) be an event-typed log. $CG(\mathcal{L}_{tp}) = CG^n(\mathcal{L}_{tp}) \cup CG^i(\mathcal{L}_{tp})$ is the set of case intervals in $\mathcal{L}$ based on transaction type tp, where 1) $CG^n(\mathcal{L}_{tp}) = \{(e_1, e_2) \in \mathcal{L}_{tp} \times \mathcal{L}_{tp} \mid e_1 < e_2 \wedge case(e_1) = case(e_2) \wedge \nexists_{e' \in \mathcal{L}_{tp}}(e_1 < e' \wedge e' < e_2 \wedge case(e') = case(e_2))\}$ are (normal) case intervals; and 2) $CG^i(\mathcal{L}_{tp}) = \{(e, e) \in \mathcal{L}_{tp} \times \mathcal{L}_{tp} \mid \nexists_{e' \in \mathcal{L}_{tp}}(e' < e \wedge case(e') = case(e))\}$ are initial case intervals (one for each case). □

We use the shorthand notation $cg_{e,tp}$ to denote case interval $(e', e) \in CG(\mathcal{L}_{tp})$, where e is the end event of the case interval and tp specifies the transaction type of e. It is sufficient to only use the end event (e.g. e) in the notation to refer to a case interval since the start event (e.g. e') can easily be identified by the definition of a case interval. If the transaction type (tp) is *complete*, the notation $cg_{e,tp}$ is further simplified to cg_e.

Definition 7 (Resource Interval). Let $\mathcal{L} \subseteq \mathscr{E}$ be an event log and $\mathcal{L}_{tp} \subseteq \mathcal{L}$ (where $tp \in Type$) be an event-typed log. $RG(\mathcal{L}_{tp}) = RG^n(\mathcal{L}_{tp}) \cup RG^i(\mathcal{L}_{tp})$ is the set of resource intervals in $\mathcal{L}$ based on transaction type tp, where 1) $RG^n(\mathcal{L}_{tp}) = \{(e_1, e_2) \in \mathcal{L}_{tp} \times \mathcal{L}_{tp} \mid e_1 < e_2 \wedge res(e_1) = res(e_2) \wedge \nexists_{e' \in \mathcal{L}_{tp}}(e_1 < e' \wedge e' < e_2 \wedge res(e') = res(e_2))\}$ are (normal) resource interval; and 2) $RG^i(\mathcal{L}_{tp}) = \{(e, e) \in \mathcal{L}_{tp} \times \mathcal{L}_{tp} \mid \nexists_{e' \in \mathcal{L}_{tp}}(e' < e \wedge res(e') = res(e))\}$ are initial resource intervals (one for each resource). □

We use the shorthand notation $rg_{e,tp}$ to denote resource interval $(e', e) \in RG(\mathcal{L}_{tp})$, where e is the end event of the resource interval and tp is the transaction type of e. Similar to case intervals, the transaction type is dropped if it is *complete*. This convention also applies to the remaining types of event intervals.

Definition 8 (Working Interval). Given an event log $\mathcal{L} \subseteq \mathscr{E}$ and an event-typed log $\mathcal{L}_{tp} \subseteq \mathcal{L}$ (where $tp \in Type$), $CG(\mathcal{L}_{tp})$ is the set of case intervals and $RG(\mathcal{L}_{tp})$ is the set of resource intervals that can be observed in $\mathcal{L}$ based on transaction type tp. $WG(\mathcal{L}_{tp}) = \{(e_1, e_2) \in CG(L_{tp}) \cup RG(L_{tp}) \mid \nexists_{e_3 \in \mathcal{L}_{tp}}(e_1 < e_3 \wedge e_3 < e_2 \wedge (res(e_3) = res(e_2) \vee case(e_3) = case(e_2))\}$ is the set of working intervals. □

Similar to case intervals and resource intervals, we use the shorthand notation $wg_{e,tp}$ to denote working interval $(e', e) \in WG(\mathcal{L}_{tp})$, which can be further simplified to wg_e if only the transaction type *complete* is present.

Definition 9 (Case Waiting Intervals). Let $\mathcal{L} \subseteq \mathscr{E}$ be an event log and $\mathcal{L}_{tp} \subseteq \mathcal{L}$ ($tp \in Type$) be an event-typed log. $CIG(\mathcal{L}_{tp}) = \{(e_1, e_2) \in \mathcal{L}_{tp} \times \mathcal{L}_{tp} \mid \exists_{e_3 \in \mathcal{L}_{tp}} (e_2 \leq e_3 \wedge (e_1, e_3) \in CG(\mathcal{L}_{tp}) \wedge (e_2, e_3) \in RG(\mathcal{L}_{tp}))\}$ is a set of case waiting intervals. □

When only the transaction type *complete* is present, we can use the shorthand notation $cig_{e',e}$ to denote case waiting interval $(e', e) \in CIG(\mathcal{L}_{tp})$, where e' is the starting event of the corresponding case interval, e is the starting event of the corresponding resource interval, and event e' occurs no later than event e.

Definition 10 (Resource Waiting Intervals). Let $\mathcal{L} \subseteq \mathscr{E}$ be an event log and $\mathcal{L}_{tp} \subseteq \mathcal{L}$ ($tp \in Type$) be an event-typed log. $RIG(\mathcal{L}_{tp}) = \{(e_1, e_2) \in \mathcal{L}_{tp} \times \mathcal{L}_{tp} \mid \exists_{e_3 \in \mathcal{L}_{tp}}(e_2 \leq e_3 \wedge (e_2, e_3) \in CG(\mathcal{L}_{tp}) \wedge (e_1, e_3) \in RG(\mathcal{L}_{tp}))\}$ is the set of case waiting intervals. □

When only the transaction type *complete* is present, we can use the shorthand notation $rig_{e',e}$ to denote resource waiting interval $(e', e) \in RIG(\mathcal{L}_{tp})$, where e' is the starting event of the corresponding resource interval, e is the starting event of the corresponding case interval, and event e' occurs no later than event e.

2.3.3. Operations on Event Intervals

Being able to identify the existence of various event intervals is an important first step in enabling event interval-based performance analysis. Now it is time to introduce a number of operations that can be subsequently applied in order to derive useful performance-related information from the event intervals identified.

Grouping. It is obvious that an event interval on its own only provides a micro-level view of the performance of a process. In order to obtain a more complete and overall understanding of a process' performance, it is necessary to gather together the event intervals that share common features useful to address specific performance-related analysis questions. This can be realised via *group by* operation.

Definition 11 (Group By). A set of event intervals $\mathcal{G} \subseteq \mathscr{G}$ can be grouped by event attributes (e.g. case, resource, activity). Let $Attr$ be the set of values carried by a specific event attribute, $\mathcal{G}\uparrow_{Attr}$ defines the collection of intervals grouped by that event attribute. A special case is $\mathcal{G}\uparrow_{\perp} = \{(\perp, \mathcal{G})\}$, where $\perp$ represents the null attribute. □

The operation of grouping by event attributes supports collecting individual event intervals into clusters of intervals in a way that the intervals in each cluster carry the same value in regard to a specific event attribute. For example, $\mathcal{G}\uparrow_{Res}$ represents the grouping of intervals by the *resource* attribute of events. Given any resource $r \in Res$, $\mathcal{G}\uparrow_{Res}(r) = \{(e_1, e_2) \in \mathcal{G} \mid res(e_2) = r\}$ is the set of intervals related to r. Hereafter we use $\mathcal{G}\uparrow^{x}_{Attr}$ as a more compact notation for $\mathcal{G}\uparrow_{Attr}(x)$, e.g. $\mathcal{G}\uparrow^{r}_{Res}$ means $\mathcal{G}\uparrow_{Res}(r)$.

The *group by* operation can be applied to the various types of event intervals. For example, grouping *case intervals* by *resource* attribute yields clusters of case intervals, where each cluster contains all the case intervals of which the corresponding work items are executed by one particular resource. Given an event log $\mathcal{L}$, $CG(\mathcal{L})$ is the overall set of case intervals in $\mathcal{L}$. The above clusters of case intervals can be specified as $CG(\mathcal{L})\uparrow_{Res}$ and the cluster for each resource $r \in Res$ as $CG(\mathcal{L})\uparrow^{r}_{Res}$.

Furthermore, the *group by* operation, when applied to a specific value of an event attribute, yields a *set* of intervals

(e.g. $\mathcal{G}\restriction^{r}_{Res}$) and hence the operation can be applied again. For example, this way one can obtain all intervals that involve a certain resource r and a certain activity a, either by writing $(\mathcal{G} \restriction^{r}_{Res}) \restriction^{a}_{Act}$ or $(\mathcal{G} \restriction^{a}_{Act}) \restriction^{r}_{Res}$.

By choosing the right combination of an event interval type and a *group by* event attribute, meaningful clusters of event intervals can be derived for conducting analysis operations which are to be introduced next.

Analysis Operations. Eventually, the goal of defining, identifying, and grouping event intervals is to enable the extraction of useful (and hopefully, representative) performance information from an event log. This can be achieved by applying a number of performance analysis operations on the clusters of event intervals obtained through application of the *group by* operation. In this paper, we propose three different analysis operations: *metrics analysis*, *evolution graph analysis*, and *decision tree analysis*.

Firstly, we define metric functions that can be used to compute performance measures, such as frequency and time duration, of event intervals for metrics analysis.

Definition 12 (Metric Functions). Let $\mathbb{G} \subseteq \mathscr{P}(\mathscr{G})$ be a set of grouped collections of event intervals where each $\mathcal{G} \in \mathbb{G}$ is a finite and non-empty set (e.g. $\mathbb{G}$ can be a set of collections of event intervals obtained from the *group by* operation in Definition 11). We introduce $\mathcal{F}$ as a set of metric functions that have $\mathbb{G}$ as their domain, and $f_{metric} \in \mathcal{F}$ refers to any metric function in $\mathcal{F}$. Two concrete examples of f_{metric}, namely $f_{\texttt{freq}}$ and $f_{\texttt{dur}}$, are defined. For any set of event intervals $\mathcal{G} \in \mathbb{G}$, $f_{\texttt{freq}} : \mathbb{G} \to \mathbb{N}$ computes the number of the intervals in $\mathcal{G}$, and $f_{\texttt{dur}} : \mathbb{G} \to \mathbb{R}$ computes the average duration of the intervals in $\mathcal{G}$. □

We can apply a metrics analysis to, theoretically, any cluster of event intervals. For example, given an event log $\mathcal{L}$ and a resource $r \in Res$, $f_{\texttt{dur}}(WG(\mathcal{L})\restriction^{r}_{Res})$ returns the *average working time* of any work items involving resource r in case only activity completions are recorded. Similarly, for any case $c \in Case$, $f_{\texttt{dur}}(WG(\mathcal{L})\restriction^{c}_{Case})$ returns the *average working time* of any work items that belong to case c.

Next, we propose two different methods for counting event intervals within a time window (i.e. a time interval). The difference lies in the way how event intervals that span across multiple time windows are counted. One is called the *Load* (L) method in which intervals are counted as long as they exist in a given time window. As a result, an interval will be counted more than once if it spans across multiple time windows. The other, namely the *Unique* (U) method, only counts those intervals that appear for the first time in a time window. As a result, each interval will be counted exactly once.

Definition 13 (Count by Time). Let $\mathcal{G} \subseteq \mathscr{G}$ be a set of event intervals. Given $t \in Time$ a point of time and $d \in TimeDuration$ a (non-zero) time duration, $[t, t+d\rangle$ defines a (valid) time window. The *Load* method counts all the intervals, $\mathcal{G}^{L}_{d}(t) = \{(e_1, e_2) \in \mathcal{G} \mid (time(e_1) < t + d \wedge time(e_2) > t) \vee (time(e_1) = time(e_2) = t)\}$, that exist in $[t, t+d\rangle$. The *Unique* method counts only the intervals, $\mathcal{G}^{U}_{d}(t) = \{(e_1, e_2) \in \mathcal{G} \mid t \leq time(e_1) < t + d\}$, that appear in $[t, t+d\rangle$ for the first time. □

Both methods are valid in counting the number of event intervals and for certain analysis questions one suits better than the other. For example, we consider the *Load* method a more reasonable means to address the analysis questions concerning case or resource occupancy, e.g. how busy a resource is during a certain period, while the *Unique* method provides a better approach to incremental analysis of performance, e.g. how much new work emerges during a time window.

While the metrics analysis supports the computation of metric measures in a static manner, it is often more interesting to understand how performance metrics evolve over a period of time. The results are usually represented and visualised in the form of evolution graphs for analysis. We consider two styles of time progression. One is the

linear time progression which is quite common and often used as a default style of time progression for analysis. The other is the *cyclic* time progression which can be used to exhibit the performance behaviour of a system in a period relative to a recurring time window. For example, we may be interested in what happens daily over a period of a month or weekly over a period of a year.

Definition 14 (Evolution Graph Analysis). Let $\mathcal{G} \subseteq \mathscr{G}$ be a set of event intervals, $t \in \mathit{Time}$ a point of time, $d \in \mathit{TimeDuration}$ a non-zero time duration, $\triangleright \in \{L, U\}$ the interval counting method, $f_{metric} \in \mathcal{F}$ a metric function for computing a certain metric measure over a set of intervals. Then, $f_{metric}(\mathcal{G}_d^{\triangleright}(t))$ computes a certain metric measure over the group of intervals within time window $[t, t+d\rangle$ over a *linear time progression*.

Let $\omega \in \mathit{TimeDuration}$ (where $\omega > d$ and ω `modulo` $d = 0$) be the duration of a time cycle, $t_0 \in \mathit{Time}$ the initial point of time of ω, $\{0, ..., n\}$ a set of integers, and $\chi \in \{\chi_{\texttt{mean}}, \chi_{\texttt{median}}, \chi_{\texttt{max}}, \chi_{\texttt{min}}, \chi_{\texttt{sum}}\}$ for calculating average, median, maximal, minimal, or total value over a set of values. Then, $\chi(f_{metric}(\mathcal{G}_d^{\triangleright}(\hat{t} + i * \omega)))$, where $t_0 \leq \hat{t} < t_0 + \omega$ and $i \in \{0, ..., n\}$, computes a certain metric measure over the group of intervals within (recurring) time window $[\hat{t}, \hat{t} + d\rangle$ over a *cyclic time progression*. □

Let's use an example to explain the evolution analysis defined above. Given an event log $\mathcal{L}$, we want to observe how the *number* of *new* resource intervals for a particular resource r, which exist in every time window $[t, t+d\rangle$ of *linear* progression, evolves over a certain time period.

Firstly, the *group by* operation is applied to collect the set of resource intervals for resource r, i.e. $RG(\mathcal{L})\restriction^r_{Res}$. Next, the *Unique* method is used to count the new resource intervals related to r that appear in each time window $[t, t+d\rangle$ of *linear* progression, resulting in $(RG(\mathcal{L})\restriction^r_{Res})^U_d(t)$. As the last step, the $f_{\texttt{freq}}$ metric function is applied to compute the number of new resource intervals of r over time t, thus $f_{\texttt{freq}}(RG(\mathcal{L})\restriction^r_{Res})^U_d(t)$. The result can be plotted into an evolution graph shown in Fig. 5(a).

Alternatively, by considering a daily (i.e. $\omega = 1$ day) *cyclic* progression, we can obtain an aggregation of evolution graphs of $f_{\texttt{freq}}(RG(\mathcal{L})\restriction^r_{Res})^U_d(\hat{t} + i * \omega)$ as Fig. 5(b) shows, and then calculate the average number of new re-

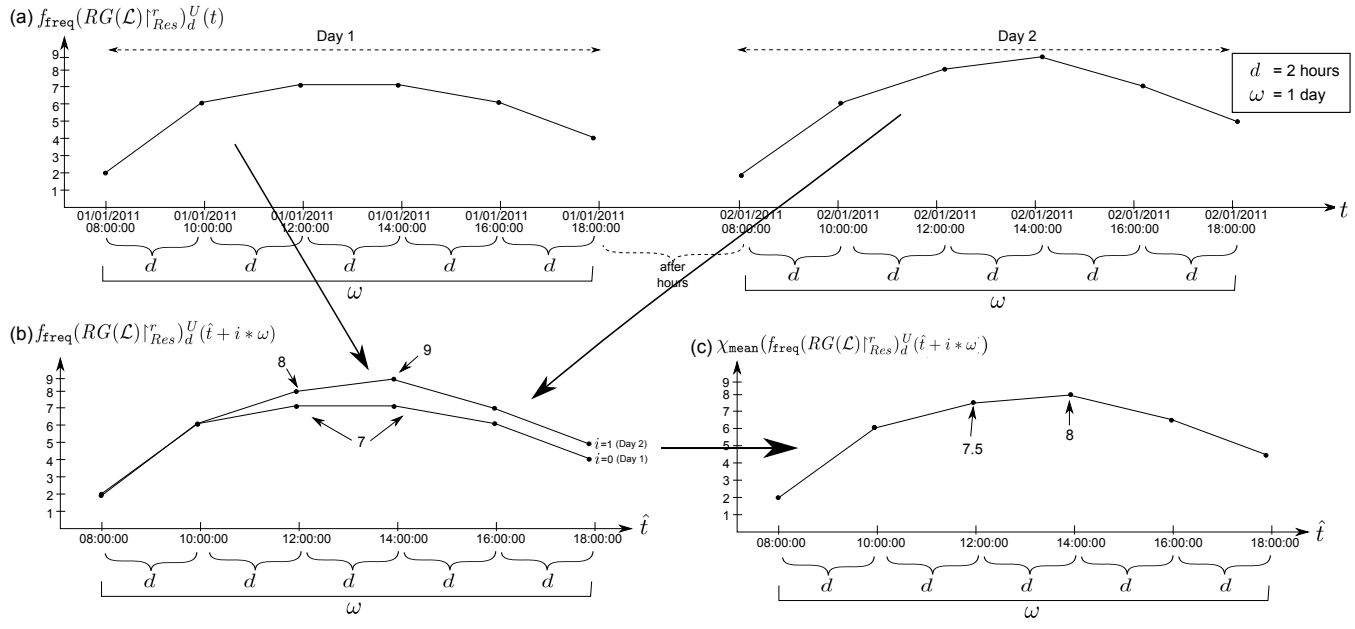

Figure 5: Examples of evolution graphs: (a) evolution over linear time progression, (b) aggregation of evolutions over cyclic time progression, and (c) average of evolutions over cyclic time progression.

independent variables					*dependent variable*
Case Id	*Activity*	*Day of the Week*	*Resource*	*Duration*	*Duration Category (RG)*
102	createPO	Wednesday	felix	20 mins	short
102	approvePO	Thursday	donald	3 mins	short
84	createPO	Monday	carol	37 mins	long
80	createPO	Wednesday	alice	18 mins	long
79	approvePO	Tuesday	bob	4 mins	short
102	modifyPO	Friday	bob	53 mins	long

Table 1: An example of event data that can be used for conducting a decision tree analysis. The *predictor variables* are those attributes that may explain the duration of intervals (which is, in turn, the *response variable*). The duration of the intervals is classified based on domain knowledge. In the table above, we classify the duration of an event interval as 'long' if it lasts more than 30 minutes for activity 'createPO', 5 minutes for activity 'approvePO', and 25 minutes for activity 'modifyPO'

.

source intervals for r at each moment of $\hat{t}$ over cyclic time progression, i.e. $\chi_{\texttt{mean}}(f_{\texttt{freq}}(RG(\mathcal{L})\restriction^{r}_{Res})^{U}_{d}(\hat{t}+i*\omega))$. This can be plotted into the graph shown in Fig. 5(c), which exhibits a daily pattern how average number of new resource intervals of r evolves, during a given time period.

Definition 15 (Decision Tree). Let $\mathcal{G} \subseteq \mathscr{G}$ be a set of event intervals. Then, $dt(\mathcal{G}) = \{((case(e'), act(e'), time(e'), res(e'), ...), durCategory(e, e')) \mid (e, e') \in \mathcal{G}\}$ defines a learning problem to identify the correlation between a set of event attributes (including such as caseID, activity, timestamp, resource, and so on, which are known as *independent variables*) and the time duration of event intervals (known as *dependent variable*).

This analysis aims to discover the impact of certain event attributes on interval duration through the application of decision tree analysis or regression analysis. As the dependent variable in this learning problem, *durCategory* captures different categories of the intervals as a result of classification of the intervals based on their duration. The value of a dependent variable such as *durCategory* is always presented in a categorial (i.e. nominal) form.

Table 1 lists an example of event data that can be used for conducting a decision tree analysis. It helps to find out how each of the attributes, including the case and the activity being executed, day of the week when a particular activity being carried out, and the resource by whom the activity being performed, affect the duration of the resource intervals.

By now we have defined the concept of event intervals and a new performance analysis framework that builds upon various operations on the event intervals, which, as currently proposed, include aggregation operation, metrics analysis, evolution of performance metrics over time of linear or cyclic time progression, and decision tree analysis. Discussion about tool implementation to support this framework follows in the next section.

3. Implementation

The approach detailed in Section 2 has been implemented as a plug-in of the ProM Tool [24], an open-source process mining environment.[6] This section describes briefly the implemented plug-in.

3.1. Interface

The input required by our plug-in is an XES/MXML [4, 22] log that satisfies the minimal requirements for process mining analysis (i.e. the log contains activity information

[6]The name of the plug-in is *Event interval Analysis* and it can be installed automatically via ProM Package Manager or manually by following the instruction available from `https://www.dropbox.com/s/lddagfz8i9yzwp0/InstallationInstructions.pdf`

related to a process, each activity can be linked to a process instance, and the occurrences of the activities can be ordered [19]). While the presence of *resource* information in the log is essential, our approach is still applicable even when such information is missing by, for example, assigning a unique resource to each activity. Furthermore, each event in the log may be tagged with its transaction type (e.g. schedule, allocate, start, complete, and/or others) in accordance with the XES/MXML specification [4, 22]. If such information is missing, it is trivial to add that using existing tools (e.g. the Disco tool can tag each event with the event transaction type of 'start' or 'complete').

As shown in Figure 6 (left), the implemented plug-in allows users to choose the time frame within which the extracted event intervals are to be analysed. Furthermore, given that our approach works with logs with a number of different transaction types, users will have to specify the transaction type that they would like to focus on in the extraction of event intervals (see Definition 3). Finally, we also allow users to specify the *interval types* that they would like to consider in their analysis (as per Definition 6 to Definition 10 in Section 2.3.2).

Once a user has applied the necessary configuration options, the plug-in will firstly extract all the applicable event intervals that exist in the log. Then it will display an interactive window that consists of a set of configuration options on the left-hand side of the panel (for the purposes of performing various *analysis types* as defined in Section 2.3.3) and a result visualisation panel on the right-hand side of the panel (see Fig. 6 - right). To make the tool more usable, configuration options that are not relevant for a particular analysis type are disabled. The details of these panels are provided in the remainder of this section.

3.2. Interval Analysis Interface

Figure 7 (left) shows a screenshot of the interactive window through which users can conduct event interval analyses interactively based on the extracted event intervals. The configuration options panel allows users to specify a particular *base analysis type*, i.e. the combination of an *analysis type*, *interval type*, and *group by* mechanism (as per Section 2.3.3).

Decision Tree. If one chooses $dt(\mathcal{G}\restriction_{Attr})$ as the *base analysis type* (see Definition 15), then the list of available attributes that can be used as *predictor variables*, as present in the log, is displayed in the drop-down box. Users then need to choose one or more *predictor* variables - see Fig. 7 (right). The *response variable* in such an analysis is always the duration of whichever *interval type* the user chooses.

Furthermore, based on the chosen predictor variables, a number of drop-down boxes is also displayed in the result

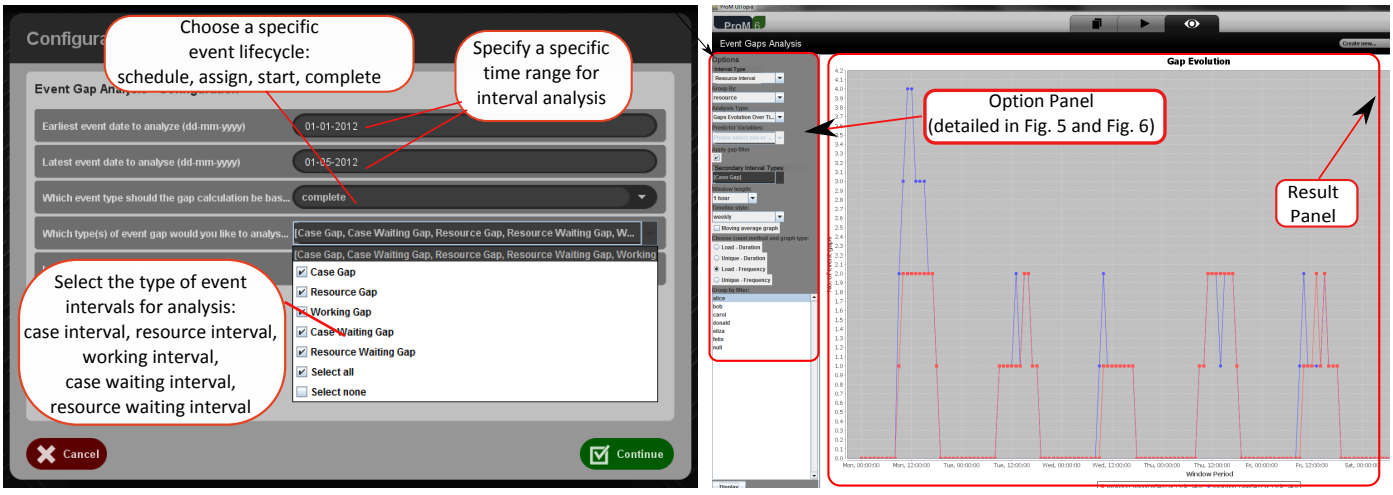

Figure 6: Event interval analysis plug-in - configuration panel and visualisation panel overview.

visualisation panel, one for each of the selected predictor variables. These drop-down boxes allow users to specify the data type for each of the selected *predictor variables* (i.e. numeric, nominal, or string) - see Fig. 7 (right). The users also need to choose if the response variable, that is, the duration of the selected *interval type*, is to be discretized based on the values themselves (that is, classes will be equally split based on the minimum and maximum values seen) or the distribution of values (that is, classes will be equally split based on the distribution of the values). The former technique may result in imbalanced classes, while the latter ensures that classes are balanced. Finally, users need to specify if the response variable is to be discretized into two classes or into three.

The decision tree analysis type invokes the `J.48` classification algorithm [15] from the WEKA library to produce one decision tree for each *interval cluster* (as determined by the *group by* parameter) to explain the correlation between the selected predictor variables and the interval duration. This analysis type produces two artifacts: (1) the decision tree(s) with the corresponding 'fitness' results mined from the data (one tree for each *interval cluster*), and (2) a set of WEKA data files (i.e. in `arff`-compliant format) that users can directly use as input into the WEKA tool for further data mining analysis (one file for each *interval cluster*). The mined decision tree(s) and the corresponding results are displayed in a text format as per the output from the WEKA library. An example of the result of a decision tree analysis can be seen in the evaluation section of this article (Fig. 20 in Section 4.2.3).

Simple Metrics. If one chooses $f_{metric}(\mathcal{G}\restriction_{Attr})$ as the *base analysis type* (see Definition 12), then the only further parameter to configure is whether the *interval filter* parameter should be set - see Fig. 8. By specifying a *interval filter*, we remove any event intervals that are made up from a pair of events whose timestamps do not share the same date. Such a filter is sometimes needed to disregard intervals that partially or wholly lay outside business hours. The simple metrics *base analysis type* produces a bar graph that displays the average, median, and standard deviation metrics for each *interval cluster*. An example of the result of a *simple metric* analysis is available in the evaluation section of this article (Fig. 16 in Section 4.2.2).

Evolution. If one chooses $f_{metric}(\mathcal{G}\restriction_{Attr})^{\triangleright}_{d}(t)$ or $\chi(f_{metric}(\mathcal{G}\restriction_{Attr})^{\triangleright}_{d}(\hat{t}+i*\omega))$ as the *base analysis type* (see Definition 14), further parameters need to be specified. Firstly, users may choose one or more 'secondary' *interval types* to be used in the analysis (see Fig, 8), in addition to the main interval type they have chosen earlier as part of

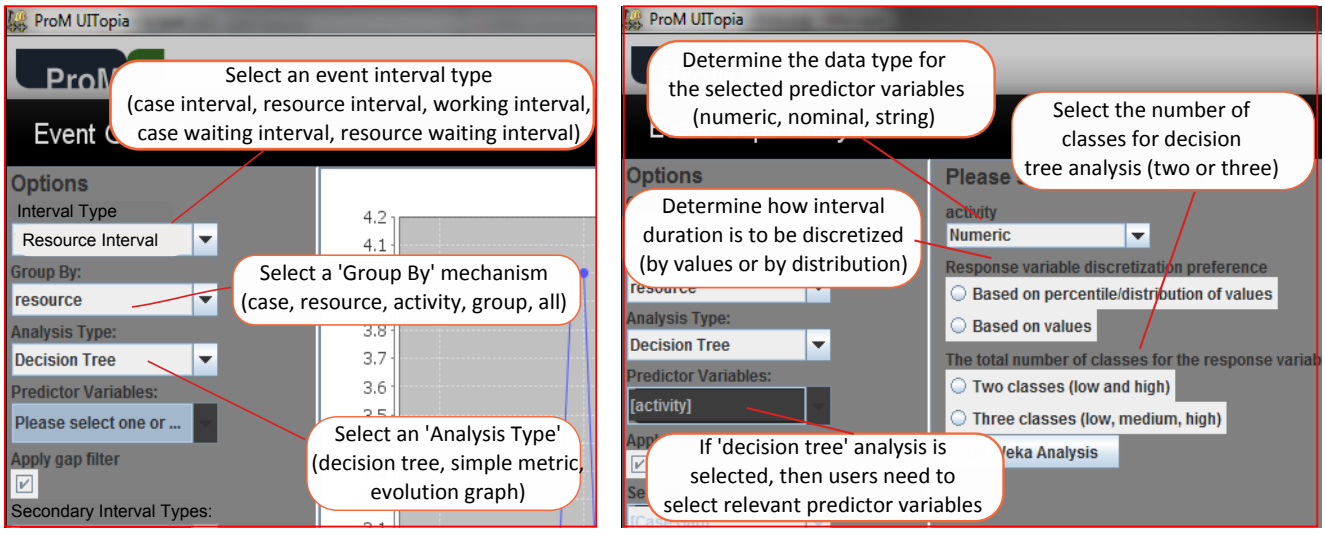

Figure 7: Option panel to configure *base analysis type* (left) and to set *decision tree* analysis options (right).

the *base analysis type*. In any case, the resulting evolution graph contains one series for each possible interval cluster, i.e. if users choose n secondary *interval types* in addition to the main *interval type*, and the chosen *group by* mechanism has m *interval clusters*, then by default the resulting graph has $(n+1) \times m$ series.

Other parameters (as defined in Section 2.3.3) to choose include (see Fig. 8):

- `count method`: *load* or *unique* (as per Definition 13),
- `graph type`: *duration* (average interval duration) or *frequency* (number of intervals) (i.e. $f_{\texttt{dur}}$ or $f_{\texttt{freq}}$ metrics functions in Definition 12),
- `window length`: from 0.5 hour to 24 hours (i.e. the value for parameter d in Definition 13),
- `timeline`: *linear* time progression or *daily*, *weekly*, *monthly*, and *year* being the set of specific values of *cyclic* time progression (i.e. the value for parameter ω in Definition 14),
- `isintervalFiltered`: *yes* or *no*, and
- `group by filter`: no specific value (*none*) or a set of chosen values $(x_1, ..., x_n)$ of an event attribute, that is, each of the chosen value x_i $(i \in \{1, ..., n\})$ corresponds to a particular *interval cluster* (i.e. group of intervals) $\mathcal{G}\restriction^{x_i}_{Attr}$, and if *none* is chosen, then all values will be used, i.e. $\mathcal{G}\restriction_{Attr}$.

Sample evolution graphs generated using two *interval types* (case interval and resource interval), group by *null*, using *unique* as the `count method` parameter, and *duration* as the `graph type` parameter are shown in Fig. 9.

4. Evaluation

To demonstrate the applicability of our approach, we show that we can use event interval analysis framework to answer a number of typical performance-related questions detailed in Section 2.2, such as the distribution of resources' workload over time, the pattern in which resources carry on their tasks, the utilisation rate of resources, comparison of resources performances, and working time estimation.

The evaluation of our approach is performed on both synthetic logs and a real-life log: the former logs are used to demonstrate the correctness of our approach, while the latter is used to demonstrate its usefulness in extracting performance information from a real event log.

The evaluation of our approach focuses on the *evolution graph* analysis (that is, the analysis defined in Definition 14) as we find it to be quite powerful in gaining insights into the performance of resources and processes. The *simple metrics* analysis is used, when needed, to provide an overview of interval duration statistics. Evaluation of the *decision tree* analysis is provided in Section 4.2.3 when we attempt to find correlations between duration of intervals and other variables seen in the event logs used.

To simplify presentation, Table 2 shows a number of configuration parameters and the corresponding short-hand notations for *evolution graph* analyses that we use in the remainder of this section.

4.1. Synthetic Log

Evaluating our approach with synthetic logs (which we have generated ourselves) is needed to show that our approach can detect certain phenomena that we know exist in the logs, which is essential to demonstrate the correctness of our approach and the related implementation.

We have generated three synthetic logs[7], each with a distinctive trend in terms of resource working patterns and process load:

- *Log 1 - Cyclic*: new cases arrive at a regular pace, and resources working patterns also follow a regular

[7] These logs can be obtained from `https://www.dropbox.com/s/0d5trj1f5qqlyuj/artificialLogs.zip`

trend dicated by the alternation between business-hours and outside-hours cycle. This log is used to represent a 'normal' process load and resource working patterns.

- *Log 2 - Disturbance Load*: new cases arrive at a regular pattern most of the time but with a sudden distinctive increase in the arrival of new cases over a short period of time, hence causing an increase in the workload as well. This log also captures a disturbance that occurred when half of the resources were not available for a period of two weeks. This log is used to capture a one-off phenomena, e.g. the occurrence of a natural disaster that causes a sudden spike in the number of new insurance claims, or the onset of a school holiday period that results in a number of resources taking leave.

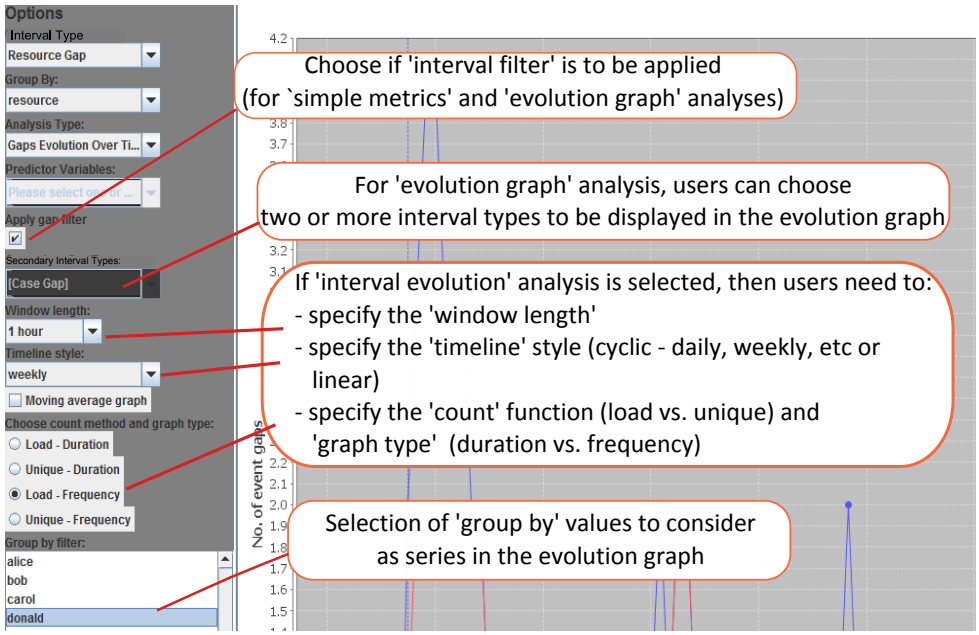

Figure 8: Option panel for *evolution graph* analysis configurations.

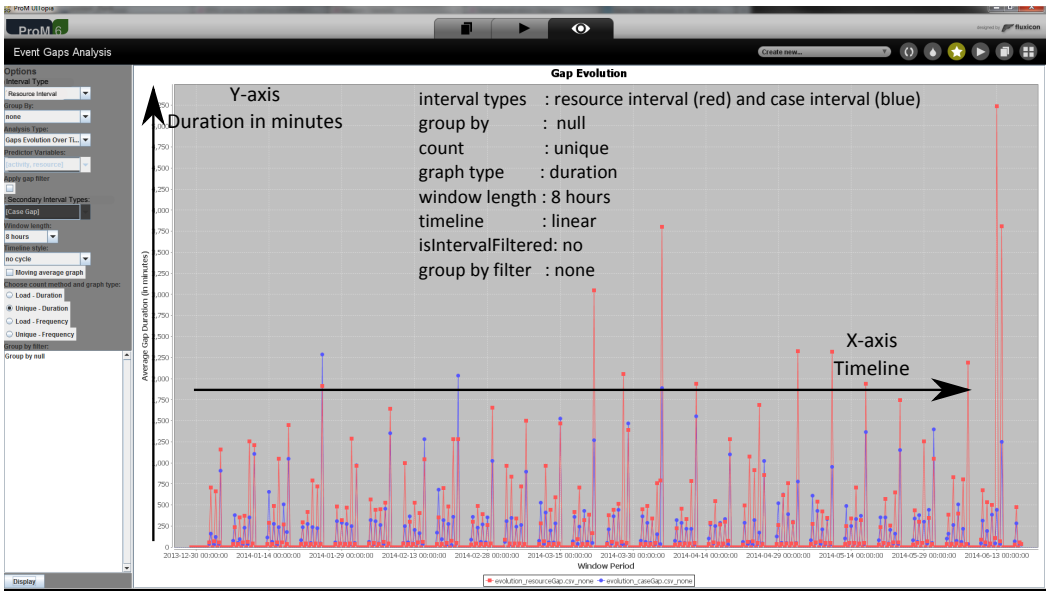

Figure 9: A sample output of evolution graphs for two interval types: *case interval* and *resource interval*.

Ref. Id	Expression of Short Notation	interval Type	Group By	Count Method	Graph Type	Window size	Timeline Prog.	interval Filtered
EV1	$f_{\texttt{freq}}(CG{\uparrow_{\perp}})^{L}_{6hr}(t)$	case	null	load	freq.	6 hrs	linear	no
EV2	$f_{\texttt{freq}}(RG{\uparrow_{Res}})^{L}_{6hr}(\text{t})$	resource	*Res*	load	freq.	6 hrs	linear	no
EV3	$f_{\texttt{freq}}(CG{\uparrow_{\perp}})^{L}_{8hr}(t)$	case	null	load	freq.	8 hrs	linear	no
EV4	$f_{\texttt{freq}}(RG{\uparrow_{Res}})^{L}_{8hr}(t)$	resource	*Res*	load	freq.	8 hrs	linear	no
EV5	$\chi_{\texttt{mean}}(f_{\texttt{dur}}(CG{\uparrow_{\perp}})^{L}_{1hr}(\hat{t}+i*1w))$	case	null	load	dur.	1 hr	weekly	yes
	$\chi_{\texttt{mean}}(f_{\texttt{dur}}(RG{\uparrow_{\perp}})^{L}_{1hr}(\hat{t}+i*1w))$	resource	null	load	dur.	1 hr	weekly	yes
	$\chi_{\texttt{mean}}(f_{\texttt{dur}}(AG{\uparrow_{\perp}})^{L}_{1hr}(\hat{t}+i*1w))$	activity	null	load	dur.	1 hr	weekly	yes
EV6	$f_{\texttt{freq}}(RG{\uparrow_{\perp}})^{L}_{8hr}(t)$	resource	null	load	freq.	8 hrs	linear	yes
EV7	$f_{\texttt{freq}}(RG{\uparrow_{Res}})^{L}_{8hr}(t)$	resource	*Res*	load	freq.	8 hrs	linear	yes
EV8	$\chi_{\texttt{mean}}(f_{\texttt{freq}}(RG{\uparrow_{Res}})^{L}_{1hr}(\hat{t}+i*1d))$	resource	*Res*	load	freq.	1 hr	daily	yes
EV9	$\chi_{\texttt{mean}}(f_{\texttt{freq}}(RG{\uparrow_{Res}})^{L}_{1hr}(\hat{t}+i*1w))$	resource	*Res*	load	freq.	1 hr	weekly	yes
EV10	$f_{\texttt{freq}}(CG{\uparrow_{\perp}})^{L}_{8hr}(t)$	case	null	load	freq.	8 hrs	linear	yes
EV11	$f_{\texttt{freq}}(CG{\uparrow_{Res}})^{L}_{8hr}(t)$	case	*Res*	load	freq.	8 hrs	linear	yes
EV12	$f_{\texttt{dur}}(RG{\uparrow_{\perp}})^{U}_{12hr}(t)$	resource	null	unique	dur.	12 hrs	linear	yes
EV13	$f_{\texttt{dur}}(RG{\uparrow_{Res}})^{U}_{1hr}(t)$	resource	*Res*	unique	dur.	1 hr	linear	yes
EV14	$\chi_{\texttt{mean}}(f_{\texttt{dur}}(RG{\uparrow_{Res}})^{U}_{1hr}(\hat{t}+i*1w))$	resource	*Res*	unique	dur.	1 hr	weekly	yes
EV15	$f_{\texttt{dur}}(CIG{\uparrow_{\perp}})^{U}_{8hr}(t)$	case waiting	null	unique	dur.	8 hrs	linear	yes
	$f_{\texttt{dur}}(RIG{\uparrow_{\perp}})^{U}_{8hr}(t)$	resource waiting	null	unique	dur.	8 hrs	linear	yes
EV16	$\chi_{\texttt{mean}}(f_{\texttt{dur}}(CIG{\uparrow_{\perp}})^{U}_{3hr}(\hat{t}+i*1w))$	case waiting	null	unique	dur.	3 hrs	weekly	yes
	$\chi_{\texttt{mean}}(f_{\texttt{dur}}(RIG{\uparrow_{\perp}})^{U}_{3hr}(\hat{t}+i*1w))$	resource waiting	null	unique	dur.	3 hrs	weekly	yes

Table 2: Configuration Parameters for *Evolution Graph* Analysis

- *Log 3 - Gradually-incrementing Load*: new cases arrive at an increasing rate, causing a gradual increase in the workload of resources as well. This log is used to represent, e.g. a gradual take-up of a new insurance product line.

All synthetic logs simulate cases that started within a 6-month period (that is, the starting date for all cases in the log fall within a particular six-month period; however, the total duration of the log is longer than six months as some cases may take a few weeks to complete). There are 6 resources in the log and each of them can execute any activity. Each new work item is allocated to the earliest-available resource, estimated from workload distribution of all resources at the time the work item is scheduled.

We have evaluated our approach using the three logs described above. Our tool managed to detect the trends that we have built into the logs. Furthermore, insights related to resources' workload and utilization can also be extracted. In this section, we only show results from Log 2 (disturbance load) and Log 3 (gradually-incrementing load). We do not show results from our evaluation with the *cyclic* log simply because the real-life log that we used for the second evaluation round (detailed in Section 4.2) already contains such behaviour.

4.1.1. Disturbance Load

Log 2 has two disturbance features built-in: (1) a sudden spike in the number of new cases in the first-half period of the log, and (2) the unavailability of half of the resources

in the second-half period of the log. Using our event interval analysis approach, we expect these two phenomena to be detected through the use of evolution graphs.

To capture process load, we can use $f_{\texttt{freq}}(CG{\restriction_{\perp}})^L_d(t)$ as the *base analysis type* (as per Definition 14). Recall that a *case interval* is defined by the time period between the time when an event for a particular case was recorded to the immediate following event belonging to the same case. Thus, by applying the above analysis, we can see, for a given period of time, the total number of work items in the process that will eventually become 'active' and be completed. Furthermore, if we refine the analysis to $f_{\texttt{freq}}(CG{\restriction_{Res}})^L_d(t)$, then we essentially obtain the workload of each resource at any given point in time. Similarly, to detect how 'busy' a resource has been, we can use $f_{\texttt{freq}}(RG{\restriction_{Res}})^L_d(t)$ as the *base analysis type* to obtain, at any given point in time, the average number of intervals (or, in this context, work items) performed by a given resource.

Fig. 10 (top-left) shows an evolution graph generated using configuration EV1 (see Table 2). The figure shows that there is a sudden spike of new *case intervals* in the first-half of the graph, thus demonstrating the ability of our approach to make it easier for process analysts to discover the existence of such disturbance. Here, it is worth emphasizing that such a trend is made evident by exploiting the *building block* of our approach, i.e. event interval, from which the load of a process, and how it changes over time, can be estimated. We contend that the systematic approach proposed in this article facilitates the extraction of such knowledge which would have been easily 'hidden' in the data otherwise.

The top-right graph in Fig. 10 shows the result of a similar *evolution graph* analysis using configuration EV2 with a focus on the resource 'bob' who, as we can see, is not available for a particular period of time, thus we can see that there is no resource interval at all for a period of two weeks. Finally, Fig. 10 (bottom-right) shows the same *evolution graph* analysis with EV2 configuration, but with a focus on another resource 'eliza' who is available throughout the whole period of the log. Interestingly, the absence of some resources for that two week period has an impact on those resources who are available: their resource intervals increase during that time period. Furthermore, for both resources 'bob' and 'eliza', they increase their throughput as evidenced by an increase in the number of case intervals during the spike period.

4.1.2. Gradual Increase of Load

Through the use of logs, we can demonstrate how our approach and the related tool are able to detect a gradual increase in the load of a process and of resources - behaviours that are built into Log 3. Fig. 11 (left) shows an *evolution graph* generated using configuration EV3 (see Table 2). In that figure, which we can see a gradual increase in the number of case intervals over the duration of the log.

The behaviour of resources in Log 3 is such that they attempt to increase the speed of their work as their workload increases. Such resource behaviour is captured in Fig. 11 (right) whereby, through the use of an *evolution graph* analysis based on configuration EV4 we can see a gradual increase in the number of resource intervals. However, this figure also shows that the number of resource intervals stabilizes towards the end of the graph - this can be explained by the fact that a resource can only speed up their work to the minimum amount of time required to complete any work items - this forms the limit on the number of maximum resource intervals that a resource can possibly have within a period of time. In other words, at full capacity, there is a limit on the maximum number of work items that a resource can complete within a given time period.

4.1.3. Estimating the Working Time

It is a challenge to obtain an estimate of task working times when we only have an event log with one event trans-

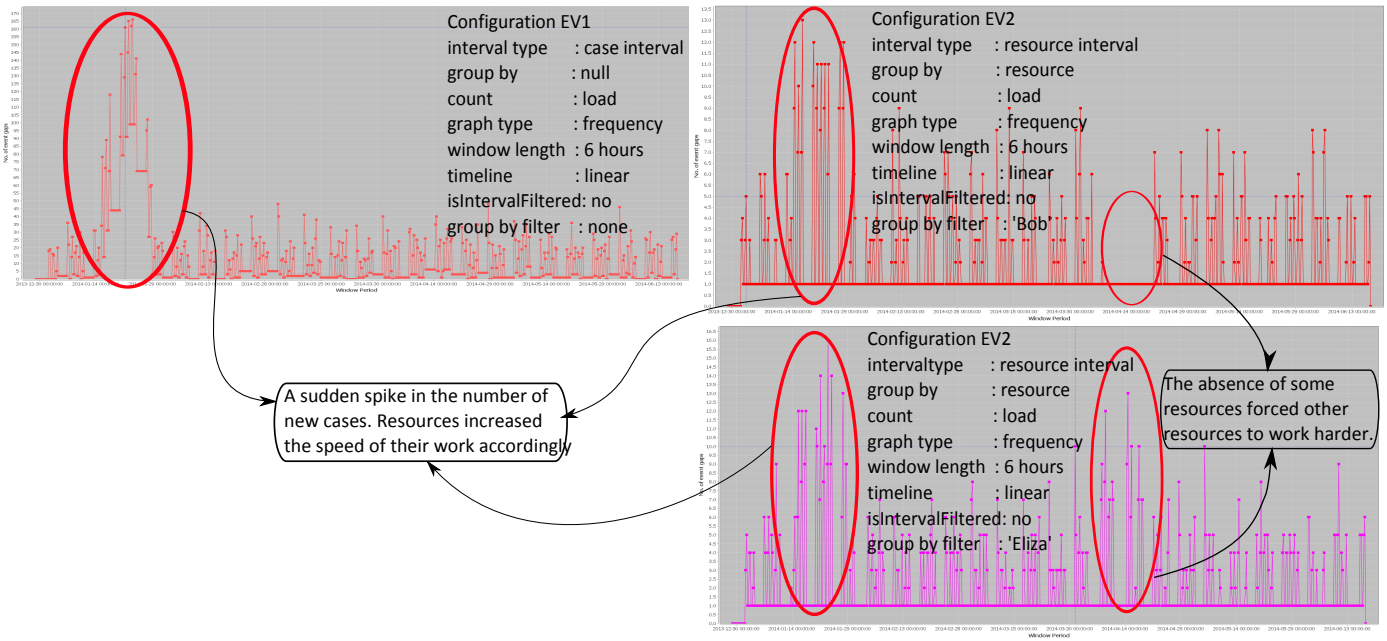

Figure 10: Evolution graphs to detect disturbances in both process load and resource workload.

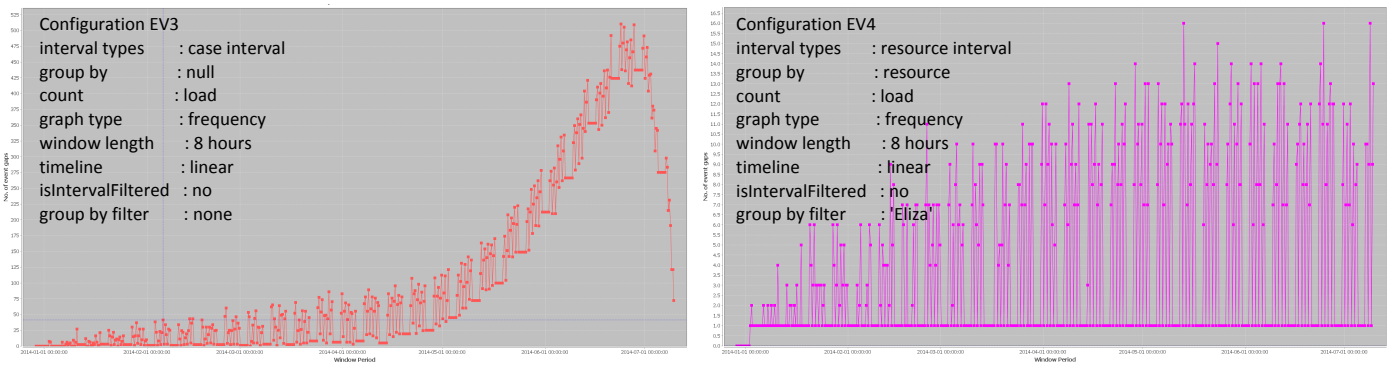

Figure 11: Evolution graphs to detect a gradual increase in process load and resource workload.

action type (e.g. complete). Existing approaches (such as the Basic Performance Analysis [7]) attempt to do so by calculating the duration between two events according to our case interval definition. However, such an estimation is most of the time too coarse, e.g. a work item related to a case may be left untouched for an extended period due to a variety of reasons (e.g. low importance or unavailability of resources). By just using case intervals, the estimated working time will also include the shelf-time of those work items.

Our approach attempts to arrive at a *better estimate* of working time through the introduction of the concept of *working interval*. By combining both *case* and *resource* perspective (as captured by the definition of *working interval*), we can get a more accurate estimation of working time even though we are still working with an information-poor log. In particular, by reasoning about the earliest time a work item is available and the earliest point in time when a resource could possibly work on the work item, we can increase the accuracy of our working time estimate. This approach links back to our earlier discussion (see Section 2.2) about the uniqueness of process analysis and, by extension, the event interval approach proposed in this article.

Fig. 12 shows an *evolution graph* derived from configuration EV5 which shows the weekly trend of the inter-

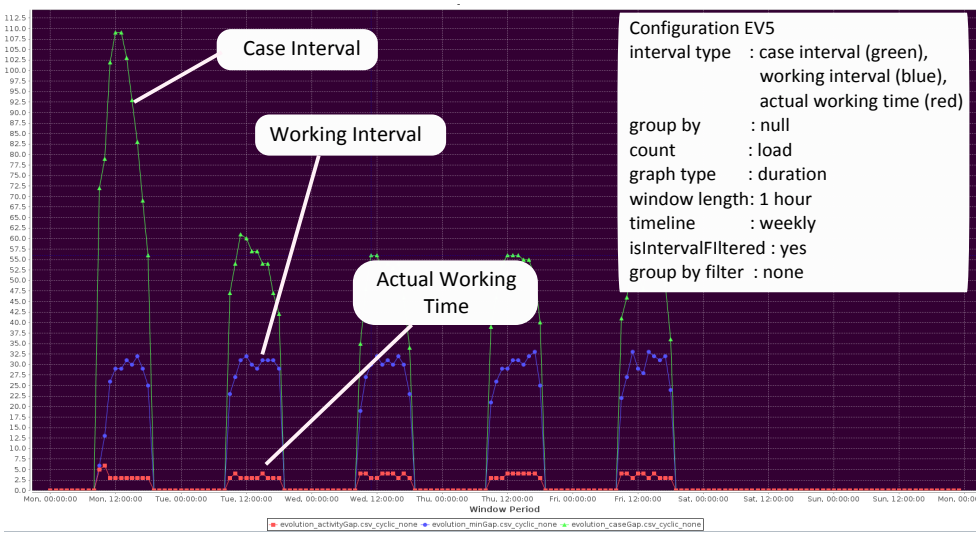

Figure 12: Comparison of case interval duration and working interval duration with actual working time duration.

val duration of *case intervals* and *working intervals*, and compares them with the actual working time of the corresponding work items.[8] As we can see, the distance between the *working interval* graph and the *actual working time* graph is, on average, always lower than the distance between the *case interval* graph and the *actual working time* graph. This figure therefore demonstrates that we can indeed obtain a more accurate picture of the working time of variuos work items by exploiting the concept of *working interval*.

Furthermore, Table 3 shows that the distance in the interval duration is larger between the *actual working* duration and the *case interval* duration than between the *actual working* duration and the *working interval* duration.

4.2. Logs from Industry

Given that we have demonstrated that our tool behaves as expected, we can now evaluate the tool using a real-life event log. To this end, we use a log from Suncorp, one of the largest insurance organizations in Australia. In particular, given the relatively long time-span of the data (there are events from as early as 2008 to as late as August 2012 in the log), we only use events from the months of January 2012 to April 2012. Furthermore, we only use the *complete* event transaction type in the evaluation.

4.2.1. Workload

We can apply the same analysis to gauge the workload of resources by using the concept of *resource interval*. Fig. 13 (top) (generated using the configuration EV6 shows typical resource behaviour over a number of weeks where we can see that the workload of resources (as captured by the total number of intervals within any 8-hour window) reached their peaks about 5 times a week (i.e. one peak per day). It is also interesting to note that we can clearly see the impact of public holidays on resources' productivity from the graph as well, e.g. during New Year's Day, Australia Day, and Easter holidays there were hardly any activities detected.

Fig. 13 (bottom) shows another *evolution graph* derived from configuration EV7 (for a particular resource R_B only). This graph shows the peculiarity of the resource's working pattern, e.g. we can see that R_B had two periods of leave, and that there were a number of weeks where this

[8] Because we used a synthetic log that also contained *start* timestamp for each work item, we could obtain the actual working times of all work items seen in the log (calculated as the duration between the start of a work item to its completion), and use this to evaluate the quality of our estimates.

Log Type	Avg. Duration			Std. Dev		
	Act. Work Time	Case interval	Work. interval	Act. Work Time	Case interval	Work interval
Log 1	1036	8223	4662	9633	26571	20996
Log 2	1247	18946	4842	11329	47850	21325
Log 3	534	130559	4682	6930	197491	21224

Table 3: Average and standard deviation of actual working time duration vs. case interval and working interval durations.

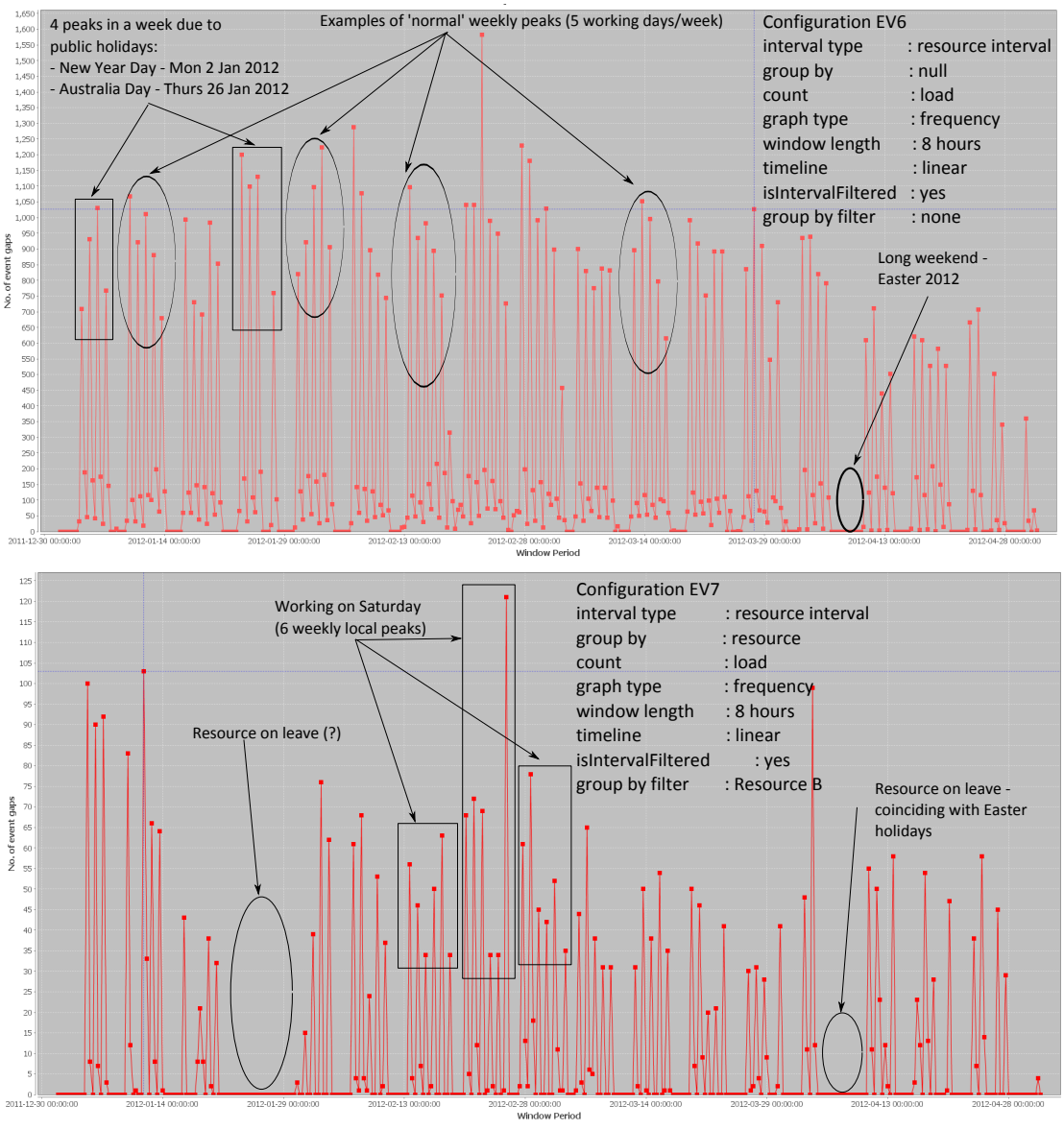

Figure 13: Workload estimation using *resource intervals*.

resource also worked on Saturday.

Despite the occasional variations in the way resources worked, we can 'average out' the variations to obtain daily and weekly patterns to extract long-term trends of resources' working patterns. For example, from the daily *evolution graph* analysis shown in Fig. 14 (top) which is derived from applying configuration EV8 it is quite evident that Resource A (R_A - identified with colour 'blue') had a lot more *resource intervals* at any given period of time, which means that this 'blue' resource completed many

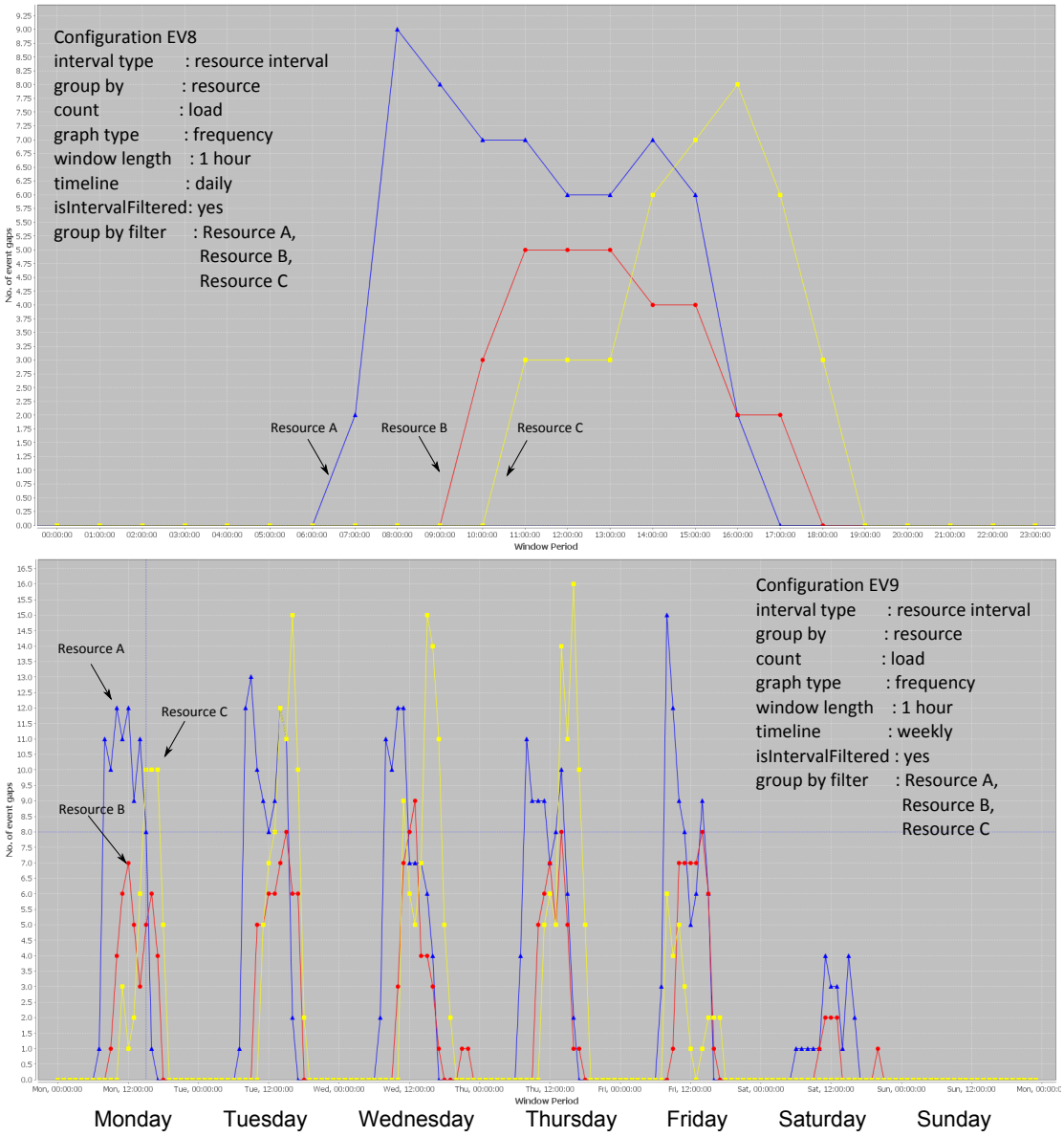

Figure 14: Daily (top) and weekly (bottom) resource interval *evolution graph* analysis for Resource A (black), Resource B (red), and Resource C (yellow).

work items. In contrast, Resource B (R_B - identified with colour 'red') had the fewest *resource intervals* at any given period of time. Of course, a straightforward interpretation would be to say that, assuming that all work items required similar amount of efforts to complete, R_A was considerably more productive than R_B. However, the lower number of resource intervals for R_B may also be explained by the fact that R_B worked on more difficult work items than R_A, hence, we see fewer work items being completed by R_B.

Additionally, we can see that R_A, on average, started work much earlier than any other resource (the first non-zero interval was recorded between 6 am and 7 am daily). In contrast, Resource C (R_C) was the last one to start his/her working day (the first non-zero interval was recorded between 10 am and 11 am daily), although R_C also finished the latest among the three (between 6 pm and 7 pm).

However, if we look at the weekly pattern shown on Fig. 14 (bottom) which is derived by applying configuration EV9 we can obtain more detailed insights. While R_A

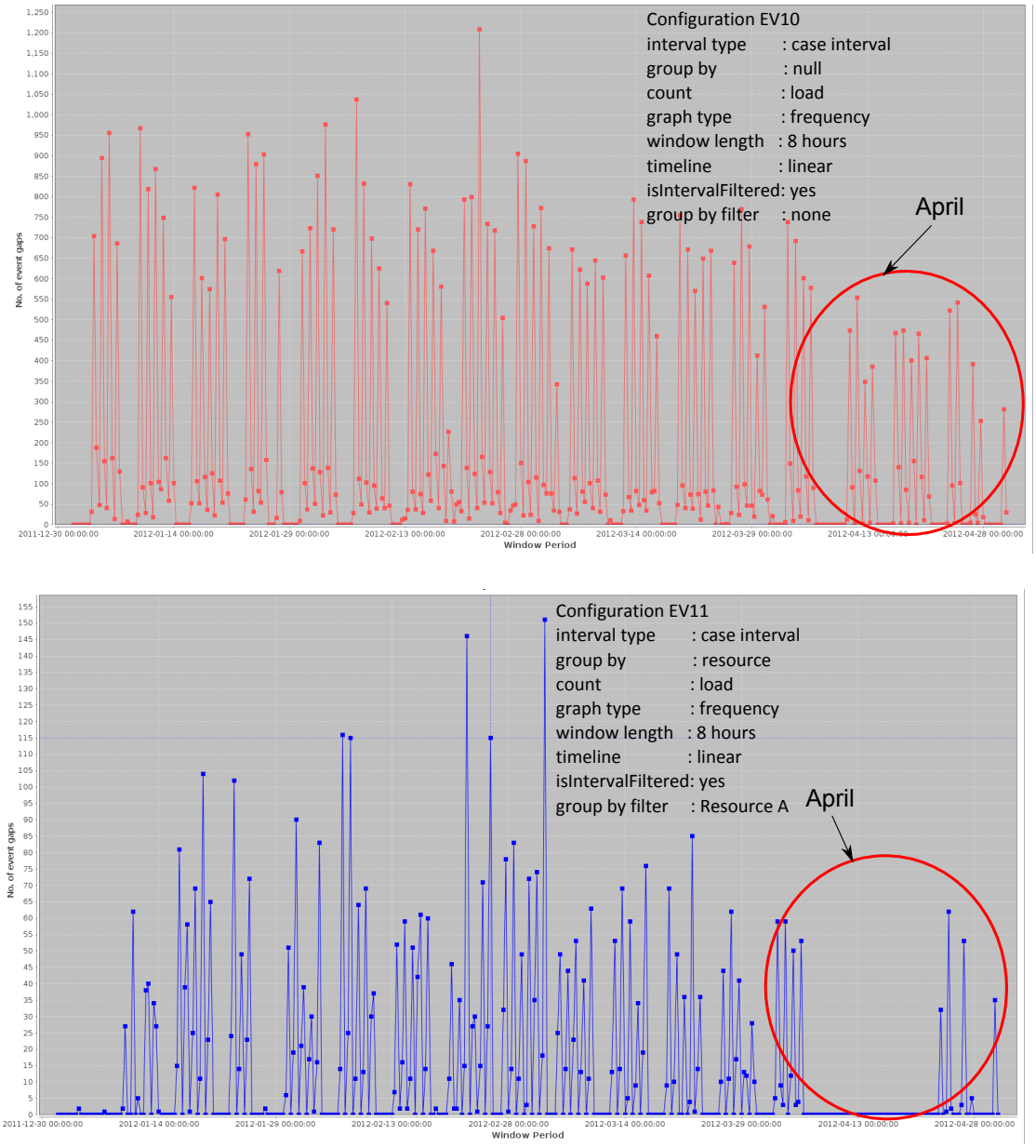

Figure 15: The case interval *evolution graph* analysis for the process (top) and for Resource A (bottom).

still maintained a high number of resource intervals, R_C seemed to have more intervals than R_A between Tuesday to Thursday; however, the productivity of R_C seemed to dip on Friday. Furthermore, we can also see that both R_A and R_B conducted work on Saturday, while this was not the case for R_C.

Fig. 15 depicts the workload of resources seen in the Suncorp log. The top part of this figure shows the aggregate workload of all resources (obtained by applying *evolution graph* analysis using configuration EV10 Fig. 15 (bottom), derived from configuration EV11, shows the *case interval* evolution for Resource A. The shape of both graphs shows that on average, the overall workload of the system remains more or less constant, except towards the month of April whereby there is a noticeable dip in the workload for both the overall process and for Resource A.

4.2.2. Resource Utilization

Similar to workload analysis, we can apply a number of event interval analyses to gauge the utilization of re-

sources. For an overview, we can always start with a *simple metric* analysis, such as $f_{metric}(RG{\uparrow}_{Res})$ (see Fig. 16) to get a sense of the amount of time taken by resources to complete a work item. This figure shows that the longest median resource interval duration was roughly 205 minutes and it was performed by a particular Resource X. Furthermore, one can roughly say that 30 minutes seems to be a good median for *resource interval* duration in general.

Of course, to get a more detailed view, we can also apply our *evolution graph* analysis. Fig. 17(a) shows an *evolution graph* derived from configuration EV12 whereby we can see that on average, resources did not take too long to complete a work item (as captured by the relatively low *resource interval* duration for all resources - about 15 minutes).

We can also focus closer on a number of resources to compare their speed in completing work items. Fig. 18(a) shows the *evolution graph* (derived from configuration EV13) for two resources: Resource A and Resource D. As can be seen, on average, Resource D took longer to complete work items.

To check if it is true that Resource D took longer to complete work items, we can of course use a similar *evolution graph* analysis, however this time, by using weekly graph configuration (as derived by applying configuration EV14 as shown in Fig. 17(b). In this figure, the average peak interval duration for Resource A was around 10 minutes (with the exception of, interestingly, Friday whereby the average duration was higher on average). Nevertheless, when we compare Resource A and Resource D, we can see that on most days, notably Mondays, Tuesdays, and Wednesdays, Resource A performed quicker than Resource D. The exceptions to this observation are for Wednesdays and Fridays.

There is a very simple explantion to the phenomenon just observed. Fig. 18(b) shows a zoomed-in picture of Fig. 18(a) (the section bounded in a red-rectangle). From this figure, if we look closely, we can see that Resource D actually worked part-time and he/she often took Wednesday as a non-working day (we can conclude this due to the fact that there was no resource interval at all for many Wednesdays for Resource D), thus, lowering the average resource interval duration for Wednesdays for Resource D. The phenomenon seen on Fridays requires further investigation and consultation with domain experts to be able to explain it properly.

Another aspect of utilization is the duration and frequency of 'waiting times' that both resources and cases need to endure (higher waiting times often signal suboptimal scheduling and/or deployment of resources). We can use both *resource waiting interval* (RIG) and *case waiting interval* (CIG) to gauge resources and cases waiting times respectively. Fig. 19 (top) shows an evolution graph derived by applying configuration EV15. This figure shows that, most of the time, the case waiting times (as deduced from *case waiting interval*) were longer than resource waiting times (from *resource waiting interval*). Such a difference becomes evident when we look at the weekly pattern of the case- and resource- waiting times as shown in Fig. 19 (bottom) which was derived from configuration EV16. One possible interpretation of this phenomenon is that the system was under-resourced, hence, more often than not, cases needed to wait for resources to be available.

4.2.3. Explaining the Interval Duration

Decision tree analysis is useful when we want to extract factors that may influence the length of the duration of intervals. For example, we may want to find out if the combination of certain activity types and resources may influence the working time of work items (i.e. the duration of *working intervals*).

Using the same log as the one used in Section 4.2.1 and Section 4.2.2, we applied our decision tree analysis (using activity types and resources as the *predictor variables*) to understand the impact of the *predictor variables*

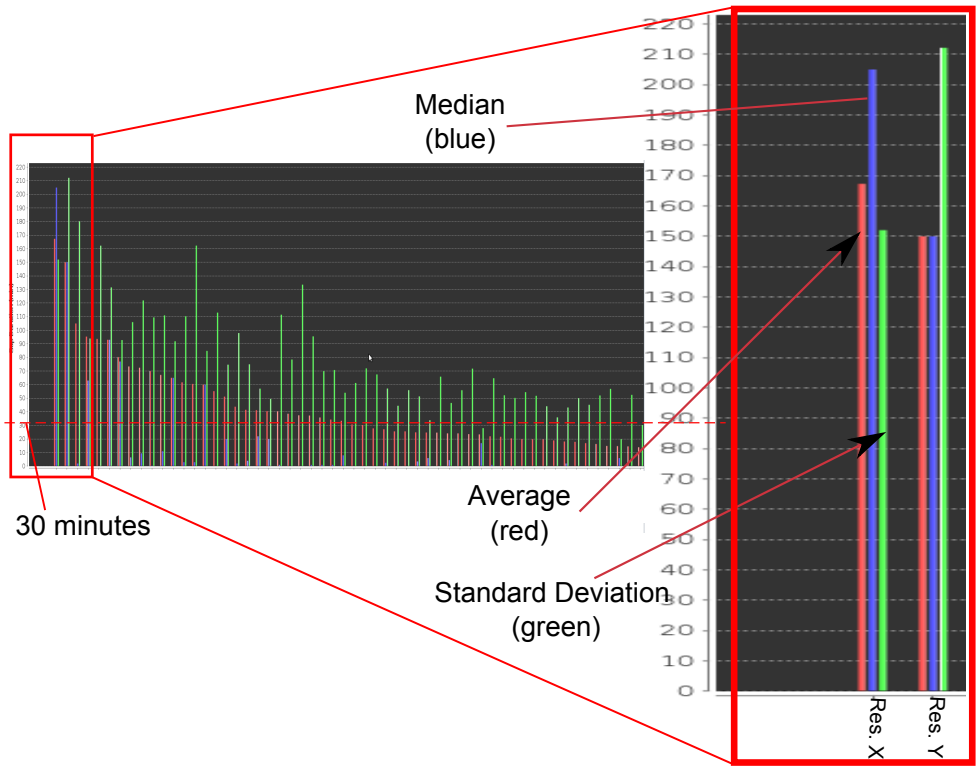

Figure 16: Simple metrics analysis $f_{metric}(RG{\uparrow}_{Res})$.

on the length of the *working interval* durations.The working intervals used in this analysis were grouped by all (i.e. $WG{\uparrow}_{\perp}$). The data types for both the *predictor variables* were set to nominal. Furthermore, we also used the tool to automatically label the intervals in the data into two classes ('long' and 'short') based on the distribution of the durations (that is, those intervals with durations shorter than the median interval duration are labelled as 'short', and vice versa). An example of the labelled data produced by the tool is provided in Table 4.

Using the labelled data, the tool performed a decision tree analysis to check if the duration of *working intervals* can be explained by activities and/or resources. The result of the analysis is shown in Fig. 20. In this case, the decision tree managed to correctly classify about three quarters of all the *working interval* durations seen in the log with the Kappa statistics [3] of 0.5 (which is commonly interpreted as having a 'moderate' agreement that the classification result is not the result of pure chance). Without domain knowledge, it is quite difficult to interpret if this classification is 'good' or 'bad'; however, the interesting point from this analysis is that we can extract insight about the correlation between duration of event interval with other factors, including the activity type and the resources who executed them. For example, we can see that activities '*Follow up requested information*' and '*Contact Assessor*' were likely to have relatively 'short' interval durations, while other activities, such as '*Re-issue document*' and '*Follow-up stakeholders (decline)*', were likely to have 'long' interval durations. For other activities, such as *Consider closing claim*, the working interval durations were dependent on the resources who executed the activity. For example, as shown in Fig. 21, when *Resource K* executed the activity, the interval durations were likely to be 'short', however, when *Resource L* or *Resource M* executed the activity, the durations were likely to be 'long'.

By applying a decision tree analysis on event interval durations, we can thus gain preliminary insights into those

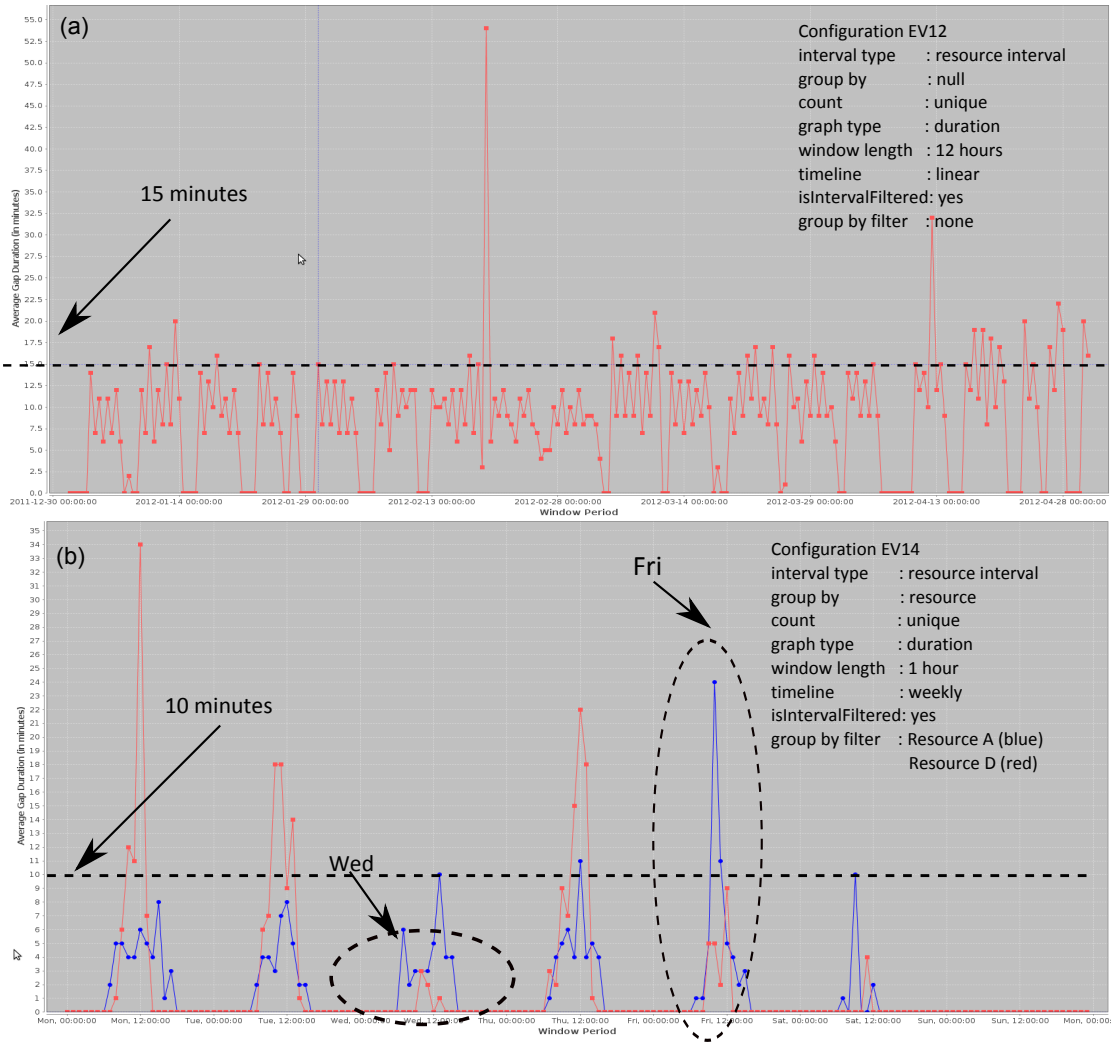

Figure 17: *Resource interval* duration for all resources (a), and weekly resource interval duration (b) for Resource A and Resource D.

Resource	**Activity**	**Label**
Resource J	Note received	Short
Resource P	Follow up from repairer	Long
Resource O	Review and approve new payment	Long
Resource C	Follow up insured	Short
.....	.....	.....

Table 4: An example of labelled data produced from the event interval analysis tool.

factors that influence the working or waiting times of various work items.

4.3. Summary

As demonstrated, the event interval analysis proposed in this article can be used to answer a variety of performance-related questions about resources as well as processes using event logs containing minimal information. Using synthetic logs that were built with particular characteristics in mind, we have shown that the implementation of the event interval analysis is correct in that expected insights about process-related performance could be extracted suc-

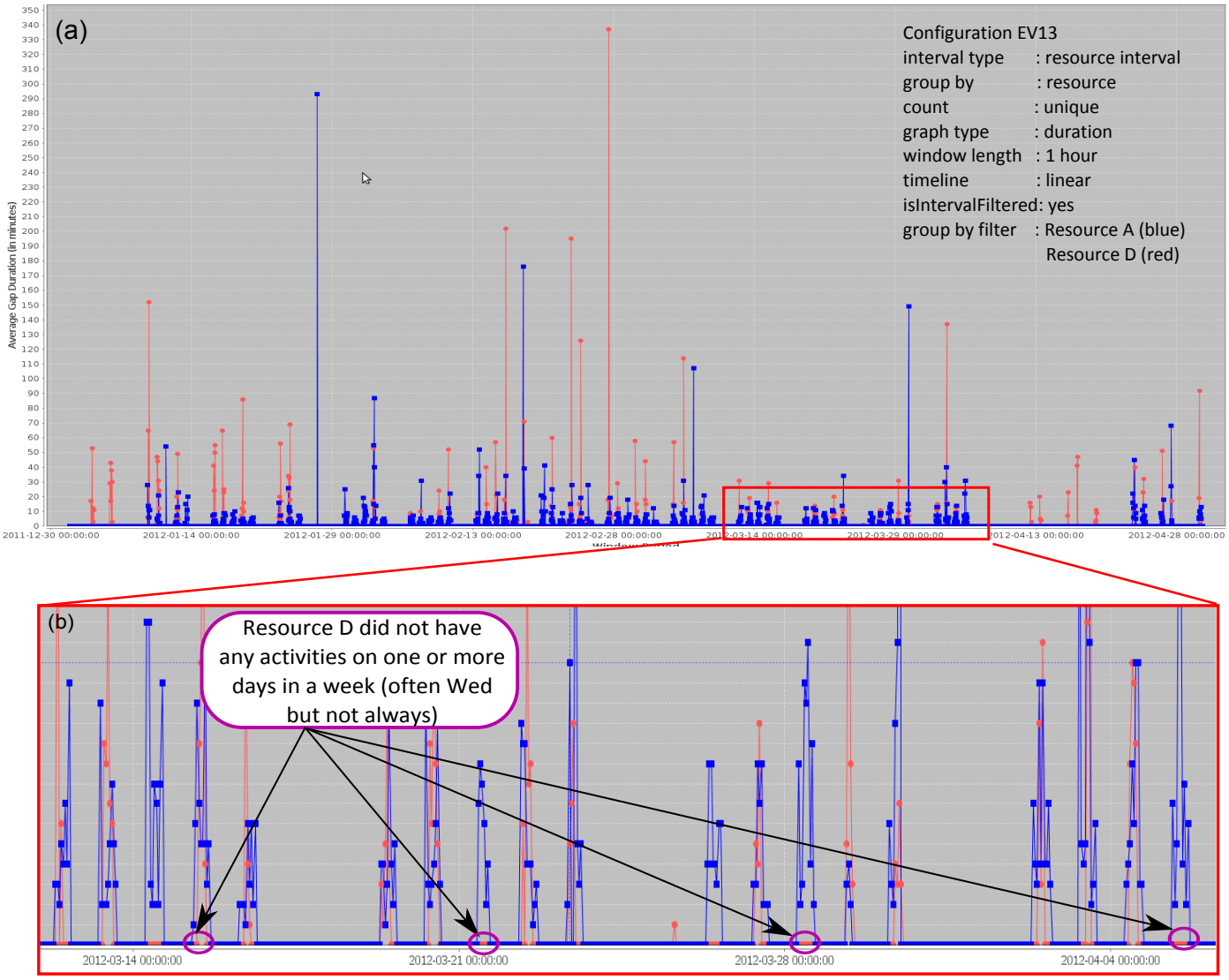

Figure 18: *Resource interval* duration for Resource A and Resource D (a), and the corresponding zoomed-in region (b).

cessfully. More importantly, by applying our event interval analysis on a real-world information-poor event log, we have shown that our tool is able to extract interesting insights into the performance of an insurance organisation's processes and their resources. In other words, we have shown that the proposed event interval analysis is *fit for the purpose* of answering typical questions related to process performance.

In particular, we have shown how the concept of working interval, which takes into account both the case and resource perspectives, can be used to arrive at a more precise estimation of working time. Through the use of evolution graph analysis, we can compare the throughput of different resources over the same period of time, as well as compare trends related to the way in which resources carry out their tasks. Through the use of resource- and case-waiting intervals, we can discover idle periods in both resources and processes. Finally, we have shown that our analysis approach allows one to learn correlation between the duration of event intervals and other attributes that exist in event logs.

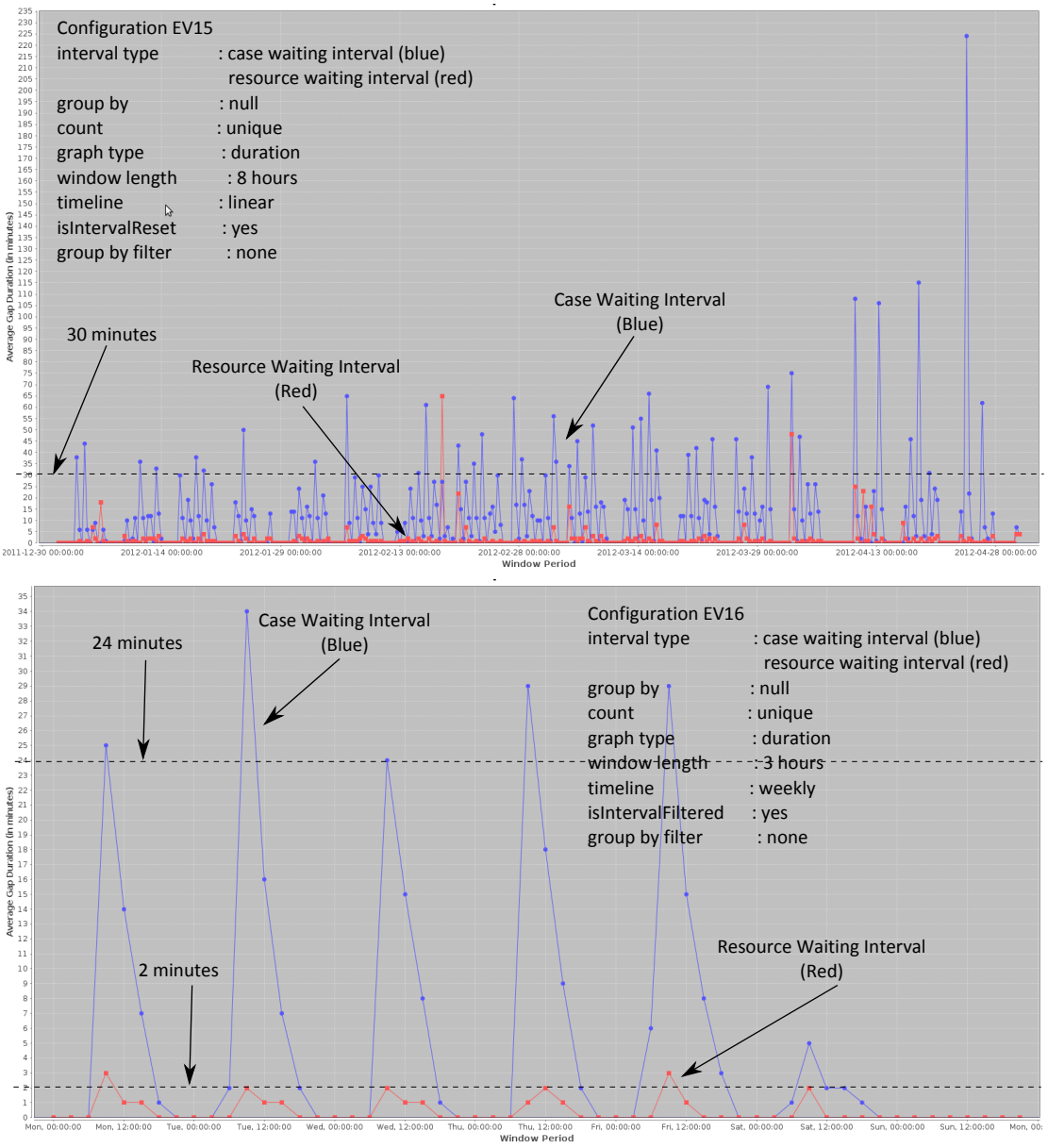

Figure 19: Case and resource waiting times using *case waiting intervals* (CIG) and *resource waiting intervals* (RIG).

5. Related Work

In the domain of process mining, there have been a number of research approaches that look at extracting performance information from event logs. One of the earliest pieces of work in this space includes the *performance analysis with Petri nets* apprach by van der Aalst and van Dongen [20], later extended by Hornix [7]. This work looks at how one can replay an event log on a process model (expressed as a Petri net) to extract performance information, such as sojourn time, waiting time, and working time of the work items related to the process. Essentially, the underlying technique applied by this approach is similar to the concept of *case interval* defined in this paper. Furthermore, this approach uses process models to compute performance information; however, it is often impossible to discover good process models due to infrequent behaviours and/or incomplete log. Our event interval analysis approach attempts to overcome this issue by not relying on the existence of a process model in order to extract more refined performance information.

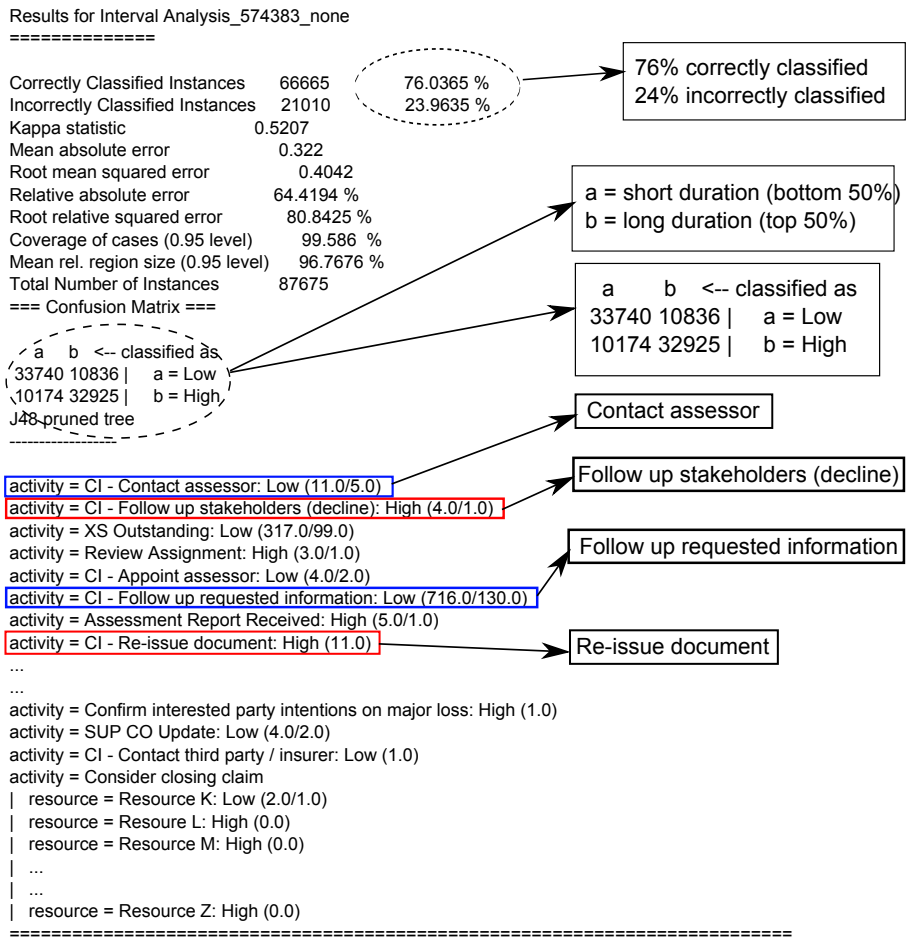

Figure 20: Decision tree analysis result detailing the correlation between *working interval* duration and activity types/resources.

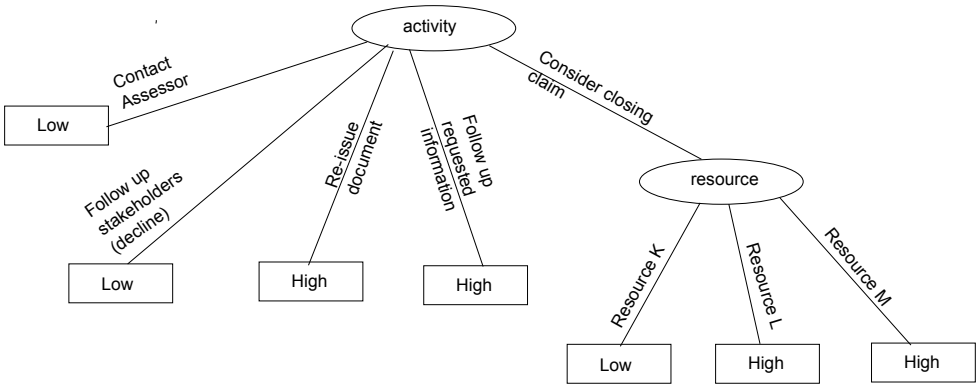

Figure 21: Graphical representation of the decision tree

Moreover, the *performance analysis with Petri nets* approach does not exploit the concept of *resource interval*. One can argue that the concept of *resource interval* can always be supported by the *performance analysis with Petri nets* approach by 're-tagging' the resource identifier attribute in an event log as the case identifier. While this may be the case, the *performance analysis with Petri nets* approach still does not support the consideration of both the case and the resource perspectives simultaneously in the manner proposed in this article. As a result, estimation of waiting times can only be performed if the log contains at least two event transaction types (either 'sched-

ule'/'start' or 'start'/'complete').[9]

Other approaches to performance analysis within the domain of process mining exploit the 'fuzzy mining' concept [23, 1, 5, 6, 16]. These fuzzy-based approaches tend to be rather intuitive as they use graphical cues to convey performance information (e.g. the use of thicker edges to represent highly-traversed paths and the use of colours to reprsent high/low/medium throughput times). Furthermore, the projection of performance information onto process models improves the intuitiveness and understandability of the performance information extracted. However, these fuzzy-based approaches only consider *case intervals*, and the interplay between proces and resource perspectives is not supported.

Popova and Sharpanskykh [13] propose the use of Temporal Trace Language to extract various process-related properties (including conformance and performance metrics) from event logs. Upon a closer look at the approach, we can also see that this approach extracts performance information strictly from the case perspective only.

Adriansyah [18, 2] proposes a robust performance analysis approach by removing 'deviant' events that cannot be paired to form a proper 'interval' for the purpose of counting interval duration. This is achieved by 'aligning' a known process model with the events seen in the corresponding log. Such an approach provides rather reliable performance information, assuming the existence of the corresponding process model. Nevertheless, similar to other approaches discussed before, the use of alignment for performance analysis only considers performance from the case perspective and not in combination with the resource perspective.

[9]The authors argued that their approach can estimate the upper bound value for waiting times, sojourn times, and execution times even when the log only contains one event transaction type, namely 'start' [7]. Nevertheless, as stated in the documentation of this plug-in in the ProM 5.1 tool, the calculation of waiting times, sojourn times, and execution times will overlap, thus we do not consider such an approach to allow proper waiting time estimations.

The work by Nakatumba [12, 11] is the closest to the event interval analysis framework proposed in this article. This work extracts performance information from both the case and resource perspectives. Concepts equivalent to the various types of intervals defined in this paper (e.g. *case interval*, *resource interval*, and *working interval*) are defined and manipulated to obtain deeper insights into not only process performance, but also performance-related behaviour (such as resources availability analysis). While the base concepts applied in this work are similar to our event interval analysis framework, our work diverges in terms of how the concept of intervals can be used. In the work of Nakatumba, the concept of event intervals is applied to extract resource availability, as well as to repair logs that miss certain event transaction timestamps. In our work, however, we generalized the concept of event intervals, and built a flexible framework that allows the manipulation of event intervals (through various *group by* mechanisms and *analysis types*) such that wider ranges of insights can be extracted. Moreoever, we are not aware of any process mining techniques that extract daily and weekly patterns of process behaviours.

Finally, questions often arise regarding the difference between our approach and other forms of data analytics, including data mining and spreadsheet analysis. We highlight here that the key distinguishing feature of our approach boils down to *the very nature of the event log data and the type of problems that we address which require the application of process perspective to solve.* In Section 2.1, we have elaborated the distinguishing feature of an event log whereby there exists temporal constraints between events in the log allowing the notion of a *process* to be captured and analysed. Here, we would also like to argue the type of questions that our approach attempts to address are process-related whereby different perspectives (such as case and resource perspectives) need to be considered. For example, to address questions such as the changes in resources' workload over time, or the seasonal

pattern of process loads, one needs to engage in a rather elaborate data analysis to provide an accurate answer. Most importantly, such analysis is not readily support, nor is it typically performed, by traditional data mining or spreadsheet analysis.

6. Conclusions and Future Work

In this article, we have proposed, formalised, implemented, and evaluated a new framework for extracting performance information from information-poor event logs. In particular, we have shown how we can uncover various types of *event intervals* from such a log and apply them in various ways to extract useful insights about performance-related behaviours from the perspectives of both process and resources involved. As detailed in Section 2.1, we have highlighted the temporal constraints that one needs to tackle in addressing process-related problems. Existing data analytics (including data mining and spreadsheet analysis) cannot extract the required insights from event data.

The application of our framework to a real-life event log (from Suncorp) has demonstrated that our framework can indeed extract interesting and useful insights that have thus far previously difficult to extract. Our analysis shows that extracting cyclical process behaviours (e.g. daily or weekly) provides us with interesting and valuable insights about seasonal behaviours of resources as well as process loads; yet this is rarely investigated because the majority of current process mining techniques tend to focus on the control-flow perspective. More importantly, the insights obtained from the use of our event interval analysis are invaluable for the purpose of making *well-informed decisions*. For example, by understanding the weekly variations in employees' workload (obtained through the evolution graph analysis), one can better manage employee's working hours, especially those part-time workers. Furthermore, through decision tree analysis, one may gain insights into the factors influencing the existence of large chunks of idle periods for resources, thus, allowing one to implement well-targeted remedial actions.

A key limitation of our framework is the assumption that, within a case, all work items occurred sequentially. This assumption has the unwanted effect of including intervals with pairs of events that actually occurred in parallel. For example, assume that the completion of an event e_a in a case c_a triggers a parallel execution of two events e_b and e_c within the same case. As recorded in the log, we see the following ordering of events: $e_a < e_b < e_c$. Possible case intervals that can be extracted from such a sequence of events include (e_a, e_b) and (e_b, e_c). As a result, the interpretation of the time period between those intervals as the maximum sojourn times for the corresponding work items is misleading. Obviously, the 'correct' case intervals in this scenario are (e_a, e_b) and (e_a, e_c).

We argue that the above limitation is surmountable by refining our definition of a *interval* with an additional restriction: any two events e_0 and e_1 ($e_0 < e_1$) can only form a interval if e_0 and e_1 have a direct causal relationship. Such relationships can be derived through the analysis of input and output data as conducted by Lu [10]. We can also discover direct causal relationships by applying well-established algorithms in the field of process mining, such as the alpha algorithm [20].

Finally, our tool currently relies on human users to identify interesting patterns in the analysis results (which are often displayed as graphs). As part of the future work, we plan to extend the tool to be able to automatically highlight regions in the graphs that may capture interesting process performance-related phenomenon (e.g. a change in a user's throughput).

Acknowledgement.. This work was supported by the Australian Research Council Discovery Project entitled *Risk-aware Business Process Management* (DP110100091).

[1] Arya Adriansyah. Performance analysis of business processes from event logs and given process models. Master's thesis, Eindhoven University of Technology, 2009.

[2] Arya Adriansyah. *Aligning Observed and Model Behavior*. PhD thesis, Eindhoven University of Technology, 2014.

[3] Jean Carletta. Assessing agreement on classification tasks: The kappa statistic. *Comput. Linguist.*, 22(2):249–254, June 1996.

[4] Technische Universiteit Eindhoven. *XES Standard Definition*, 1.4 edition, October 2012.

[5] Christian W. Günther. *Process Mining in Flexible Environments*. PhD thesis, Eindhoven University of Technology, 2008.

[6] Christian W. Günther and Wil M.P. van der Aalst. Fuzzy mining- adaptive process simplification based on multi-perspective metrics. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management*, volume 4714 of *LNCS*, pages 328–343. Springer, 2007.

[7] Peter T.G Hornix. Performance analysis of business processes through process mining. Master's thesis, Technische Universiteit Eindhoven, 2007.

[8] Teresa Jones. Identify abpd's business benefits and understand vendor strengths. *Gartner*, (G00247367), May 2013.

[9] Ruth Liew. Suncorp banks on data mining to slash claims. *Australian Financial Review*, 2013.

[10] Xixi Lu. Artifact-centric log extraction and process discovery. Master's thesis, Eindhoven University of Technology, 2013. `http://www.processmining.org/_media/blogs/pub2013/finalthesis_-_lu.pdf`.

[11] Joyce Nakatumba. *Resource-aware business process management: analysis and support*. PhD thesis, Eindhoven University of Technology, 2013.

[12] Joyce Nakatumba and Wil M. P. van der Aalst. Analyzing resource behavior using process mining. In Stefanie Rinderle-Ma et al., editor, *Business Process Management Workshops*, volume 43 of *LNBIP*, pages 69–80. Springer, 2009.

[13] Viara Popova and Alexei Sharpanskykh. Formal analysis of executions of organizational scenarios based on process-oriented specifications. *Applied Intelligence*, 34(2):226–244, 2011.

[14] QPR Software. From opportunity to delivery, end-to-end process transparency for ruukki. 2013. `http://www.win.tue.nl/ieeetfpm/lib/exe/fetch.php?media=:casestudies:processmining_casestudy_ruukki_qpr.pdf`. Last accessed 26 March 2014.

[15] John R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[16] Anne Rozinat. *Disco User's Guide*. Fluxicon, 2012. `https://fluxicon.com/disco/files/Disco-User-Guide.pdf`. Last accessed 28 March 2014.

[17] Suriadi Suriadi, Moe T. Wynn, Chun Ouyang, Arthur H. M. ter Hofstede, and Nienke J. van Dijk. Understanding process behaviours in a large insurance company in australia: A case study. In Camille Salinesi, Moira C. Norrie, and Oscar Pastor, editors, *Advanced Information Systems Engineering*, volume 7908 of *LNCS*, pages 449–464. Springer, 2013.

[18] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.

[19] Wil M. P. van der Aalst et al. Process mining manifesto. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops*, volume 99 of *LNBIP*, pages 169–194. Springer, 2011.

[20] Wil M. P. van der Aalst and Boudewijn F. van Dongen. Discovering workflow performance models from timed logs. In Yanbo Han, Stefan Tai, and Dietmar Wikarski, editors, *Engineering and Deployment of Cooperative Information Systems*, volume 2480 of *LNCS*, pages 45–63. Springer, 2002.

[21] Wil M.P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[22] Boudewijn van Dongen. MXML - a meta model for process mining data. Presentation at Enterprise Modelling and Ontologies for Interoperability Workshop 2005, June 2005. `http://www.processmining.org/_media/presentations/miningmetamodelimoa2005.ppt` - Last accessed 27 Feb 2014.

[23] Boudewijn F. van Dongen and Arya Adriansyah. Process mining: Fuzzy clustering and performance visualization. In *Business Process Management Workshops*, volume 43 of *LNBIP*, pages 158–169. Springer, 2009.

[24] Boudewijn F. van Dongen, Ana K.A. De Medeiros, H.M.W. Verbeek, A.J.M.M Weijters, and Wil M. P. van der Aalst. The ProM framework: A new era in process mining tool support. In Gianfranco Ciardo and Philippe Darondeau, editors, *Applications and Theory of Petri Nets*, volume 3536 of *LNCS*, pages 444–454. Springer, 2005.

[25] Bram Vanschoenwinkel. Case study - process mining: Package delivery. 2012. `http://www.win.tue.nl/ieeetfpm/lib/exe/fetch.php?media=:casestudies:ae_case_process_mining.pdf`. Last accessed 26 March 2014.

[26] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. XES, XESame, and ProM 6. In *Information Systems Evolution*, volume 72 of *LNBIP*, pages 60–75. Springer, 2011.