

# Visualizing Token Flows Using Interactive Performance Spectra

Wil M.P. van der Aalst<sup>1,2</sup>[0000-0002-0955-6940], Daniel Tacke genannt Unterberg<sup>1</sup>,  
Vadim Denisov<sup>2,3</sup>, and Dirk Fahland<sup>2</sup>[0000-0002-1993-9363]

<sup>1</sup> Process and Data Science (Informatik 9), RWTH Aachen University, Aachen, Germany

<sup>2</sup> Analytics for Information Systems group, Technische Universiteit Eindhoven, The Netherlands

<sup>3</sup> Vanderlande Industries, Veghel, The Netherlands

wvdaalst@rwth-aachen.de

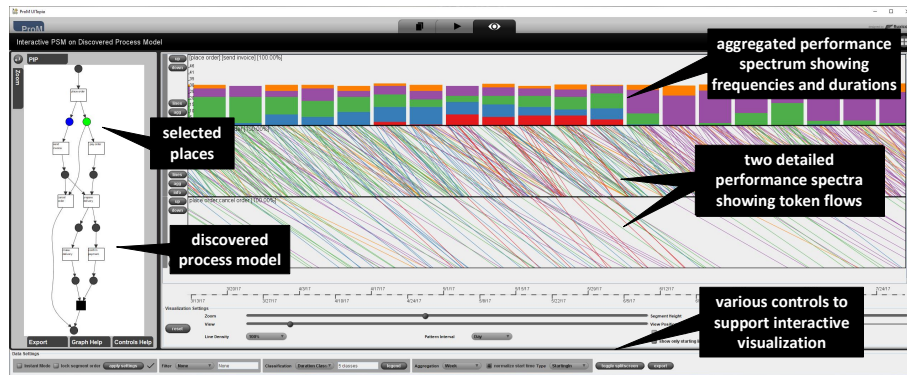
**Abstract.** Process mining techniques can be used to discover process models from event data and project performance and conformance related diagnostics on such models. For example, it is possible to automatically discover Petri nets showing the bottlenecks in production, administration, transport, and financial processes. Also basic statistics (frequencies, average delays, standard deviations, etc.) can be projected on the places and transitions of such nets to reveal performance and compliance problems. However, real-life phenomena such as overtaking, batching, queueing, concept drift, and partial blocking of multiple cases remain invisible when considering basic statistics. This paper presents an approach combining Petri-net-based discovery techniques and so-called performance spectra based on token flows. Token production and consumption are visualized such that the true dynamics of the process are revealed. Our ProM implementation supports a range of visual-analytics features allowing the user to interact with the underlying event data and Petri net. Event data related to the handling of orders are used to demonstrate the functionality of our tool.

**Keywords:** Process mining · Visual analytics · Petri nets · Performance spectrum.

## 1 Introduction

Process mining techniques can be used to automatically discover process models, diagnose compliance and performance problems, predict behaviors, and recommend process improvements [1]. Currently, there are over 30 commercial process mining tools (Celonis, ProcessGold, Disco, Everflow, Lana, Logpickr, Mehrwerk, Minit, MyInvenio, PAFnow, PuzzleData, Timeline, QPR, etc.) and thousands of organizations are using process mining to improve their processes.

In process mining, there are two main types of artifacts: *event logs* and *process models* [1]. Each *event* in an event log refers to an *activity* possibly executed by a *resource* at a particular *time* and for a particular *case*. An event may have additional attributes such as transactional information, costs, customer, location, and unit. Process discovery techniques can be used to discover process models [1]. Although most of the



**Fig. 1.** Screenshot visualizing the token flows of the two selected places using three interactive performance spectra (based on an event log with 39.022 orders and 210.459 events).

commercial systems start by deriving a so-called *directly-follows graph* (a Markovian model with activities as states), there is consensus that *concurrency* should be uncovered to avoid Spaghetti-like underfitting process models [2]. Therefore, all of the more advanced tools provide ways of discovering *higher-level process models* such as Petri nets, process trees, statecharts, or BPMN models [1].

The approach presented in this paper assumes that we have an event log and a corresponding Petri net that was automatically discovered. Existing techniques are able to replay the event log on such a process model to show performance-related diagnostics by *aggregating* over all cases, e.g., the frequency of a transition and the average or median waiting time [1, 10, 11]. This is done by showing numeric values next to places, transitions, and arcs. It is also possible to use colors and line thickness, e.g., transitions with long waiting times are colored red and frequent paths are thicker than infrequent ones. Although existing tools also report minimum times, maximal times, and standard deviations, they have problems revealing the following phenomena that require *comparing all individual cases*:

- *Overtaking*: Some cases are handled much faster, thereby bypassing cases that started earlier. This can only be observed through pairwise comparison.
- *Batching*: Groups of events are combined into batched activities.
- *Queueing*: Cases are delayed waiting for shared resources.
- *Concept drift*: The dynamic behavior is changing over time (periodic, gradual, or sudden). This can only be observed by plotting the fine-grained behavior over time.
- *Partial blocking*: Specific subclasses of cases are delayed, whereas others are not.

For example, if the reported average delay in a part of the process is 8.5 days (computed over the past year), then we cannot say anything about phenomena such as overtaking, batching, queueing, concept drift, and partial blocking. These phenomena are caused by interactions between cases and resources that may change over time. Therefore, they cannot be identified by looking at aggregate data.

To address these limitations, we use so-called *performance spectra* based on *token flows*. A token flow refers to the production of a token for a place and the later consump-

tion and also the corresponding two timestamps. Note that a token flow has a *duration*. Performance spectra were introduced in [6] to provide fine-grained visualizations of the performance of business processes and material handling systems along *sequences* of activities (rather than process models).

In this paper, we use performance spectra in the context of a Petri net and provide extensive tool support. We provide a tight integration between process models and performance spectra. By aligning model and log, we can visualize performance spectra for token flows in concurrent processes instead of along sequences of interleaved events, thereby overcoming a significant limitation of our prior work [6]. We can now also handle concurrent processes, thereby exploiting the alignment of model and log.

The approach has been implemented as a ProM plug-in and can be downloaded from *promtools.org* (install the package “PerformanceSpectrumIntegration”). Figure 1 shows the tool while analyzing the detailed behavior of token flows of two places (left) over time (right). The performance spectra reveal concept drift, overtaking, and queueing. These phenomena would not be visible if one would consider basic place-related statistics such as average waiting times and standard deviations.

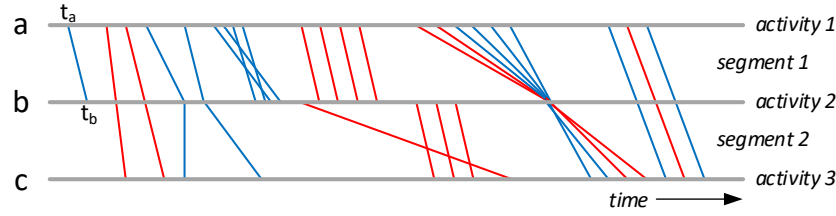
Our work can be seen as part of *visual analytics* [9], also called “the science of analytical reasoning facilitated by interactive visual interfaces”, since we combine automated process discovery with interactive visualization. Therefore, we aim to apply best practices from visual analytics, e.g., interaction, supporting the identification of patterns, dealing with many data points, and the ability to drill down.

The remainder is organized as follows. Section 2 introduces performance spectra and Section 3 introduces token flows and how these correspond to performance spectra. Section 4 shows how token flows can be extracted from event logs, thus enabling the computation of performance spectra in the presence of concurrency. The implementation and its application are presented in Section 5. Section 6 concludes the paper.

## 2 Performance Spectra

Performance spectra were first introduced in [6]. They use the notion of *segments*. A segment is defined by a pair of activities  $(a, b)$ , e.g., two subsequent sensors on a conveyor belt or two subsequent stations of an assembly line. Each object (e.g., a bag or product) flowing from  $a$  directly to  $b$  is an *observation* and is characterized by two timestamps  $(t_a, t_b)$  where  $t_a$  is the time  $a$  occurs directly followed by  $b$  at time  $t_b$ . These segments can be concatenated and it is possible to show the *detailed performance spectrum* with one line for every pair  $(t_a, t_b)$  or the *aggregate performance spectrum* aggregating results per time period (yielding a kind of bar chart with periods on the horizontal axis). The aggregate performance spectrum is based counting the number of lines (i.e., observations) per time period. Observations can be extended with a class label  $c$  (e.g., fast/slow or compliant/deviating) leading to characterizations of the form  $(t_a, t_b, c)$ . Label  $c$  can be used to filter or color the lines in the detailed performance spectrum. See Figure 2 for an example and [6, 5] for more details (e.g., a taxonomy of patterns in performance spectra).

Figure 3 shows a few common patterns that should be self-explanatory. For example, crossing lines correspond to overtaking, e.g., a LIFO (Last-In-First-Out) queuing



**Fig. 2.** Example of a detailed performance spectrum composed of two segments and three activities  $a$ ,  $b$ , and  $c$ . Each observation (i.e., a line) connects two correlated events. The horizontal axis represents time. The color indicates the class.

discipline. If the tokens in a place are always consumed in FIFO (First-In-First-Out) order there are no crossing lines.

Batching corresponds to converging lines. We can also see how patterns change over time, e.g., due to concept drift. Figure 3(g-i) shows three aggregate performance spectra. Here, information about frequencies, durations, etc. is aggregated over predefined time intervals. In all performance spectra, colors can be used to distinguish observations related to different classes.

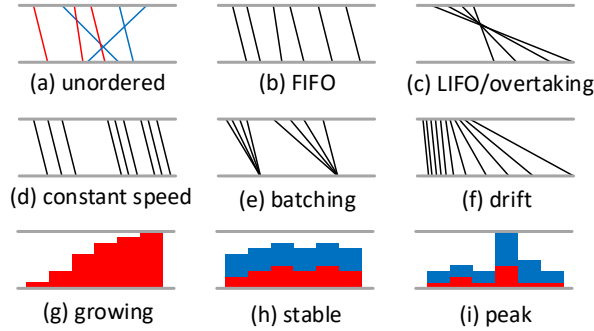
Performance spectra have been used to analyze materials-handling systems such as the handling of luggage in airports and parcels in distribution centers. However, they can be applied to any operational process to gain novel insights regarding overtaking, batching, queueing, concept drift, and partial blocking that remain hidden when looking at traditional performance indicators like frequencies and average durations.

Performance spectra are created using *event data*. Each observation (i.e., a line) corresponds to two *correlated events*. In this paper, we use standard event data where each event has a timestamp and refers to at least an activity and a case.

**Definition 1 (Log Events, Event Log).**  $\mathcal{U}_{LE}$  is the universe of log events. A log event  $e \in \mathcal{U}_{LE}$  can have any number of attributes.  $\pi_x(e)$  is the value of attribute  $x$  for event  $e$ .  $\pi_x(e) = \perp$  if there is no such value. We assume that each log event  $e$  has a timestamp  $\pi_{time}(e) \neq \perp$  and refers to a case  $\pi_{case}(e) \neq \perp$  and an activity  $\pi_{act}(e) \neq \perp$ .  $L \subseteq \mathcal{U}_{LE}$  is an event log describing a collection of recorded events.

Note that events can have many more attributes than the mandatory ones, e.g., for  $e \in L$ ,  $\pi_{case}(e)$ ,  $\pi_{act}(e)$ , and  $\pi_{time}(e)$  need to exist, but also  $\pi_{resource}(e)$ ,  $\pi_{costs}(e)$ , and  $\pi_{location}(e)$  may be defined.

The case identifier is used to correlate two subsequent events belong to the same case. The activities are used to build segments. Consider the leftmost observation (first blue line) in Figure 3.  $t_a$  and  $t_b$  refer to the timestamps of the two corresponding events  $e_1$  and  $e_2$ , i.e.  $\pi_{act}(e_1) = a$ ,  $\pi_{time}(e_1) = t_a$ ,  $\pi_{act}(e_2) = b$ , and  $\pi_{time}(e_2) = t_b$ . These events are correlated because they have the same case identifier (i.e.,  $\pi_{case}(e_1) = \pi_{case}(e_2)$ ). For example, events  $e_1$  and  $e_2$  refer to the same bag in a baggage handling system or to a particular car that is assembled. The class label  $c$  of an observation  $(t_a, t_b, c)$  may be based on the event attributes, e.g.,  $\pi_{resource}(e_1)$ ,  $\pi_{costs}(e_1)$  or  $\pi_{time}(e_2) - \pi_{time}(e_1)$ .



**Fig. 3.** Six detailed performance spectra (a-f) and three aggregate performance spectra (g-i).

Although the Performance Spectrum Miner (PSM) presented in [5, 6] uses event data, the link to process models (e.g. Petri nets) was not supported thus far. Moreover, the PSM has difficulties dealing with concurrency since direct successions of *causally unrelated* activities are hiding the real causalities and durations. Consider, for example, a process where the traces  $\langle a, b, c, d \rangle$  and  $\langle a, c, b, d \rangle$  occur both 100 times. Although  $a$  is always followed by  $b$  (i.e., 200 times), without preprocessing, the PSM will only show this for half of the cases (and show misleading delays).

### 3 Token Flows

To combine process discovery with performance spectra, we introduce the notion of *token flows*. This way, we are able to relate performance spectra to process models and visualize concurrent processes.

Petri nets are defined as usual [7] and can be discovered using standard process mining techniques [1]. We assume that the reader is familiar with the basic concepts.

**Definition 2 (Petri Net).** A Petri net is a tuple  $N = (P, T, F)$  with  $P$  the set of places,  $T$  the set of transitions such that  $P \cap T = \emptyset$ , and  $F \subseteq (P \times T) \cup (T \times P)$  the flow relation.  $\bullet x = \{y \mid (y, x) \in F\}$  is the preset of a place or transition  $x \in P \cup T$ , and  $x \bullet = \{y \mid (x, y) \in F\}$  its postset.

In a *marking*, i.e., a state of the Petri net, places may contain *tokens*. A transition  $t \in T$  is *enabled* if the input places are marked. A transition that *occurs* (i.e., fires) consumes one token from each output place and produces one token for each output place.

We need to relate transition occurrences to events in the event log. Therefore, we differentiate between the different occurrences of a transition and use *events* as a unifying notion. Events may refer to *transition occurrences* (cf. Definition 3) and/or events recorded in the *event log* (cf.  $L \subseteq \mathcal{U}_{LE}$  in Definition 1).

**Definition 3 (Transition Events).**  $\mathcal{U}_{TE}$  is the universe of transition events, i.e., all events having a timestamp and referring to a model transition. A transition event  $e \in$

$\mathcal{U}_{TE}$  has a timestamp  $\pi_{time}(e) \neq \perp$  and refers to a transition  $\pi_{trans}(e) \neq \perp$ , and can have any number of attributes. (as before,  $\pi_x(e)$  is the value of attribute  $x$  for event  $e$  and  $\pi_x(e) = \perp$  denotes that there is no such value).

A transition event  $e \in \mathcal{U}_{TE}$  refers to the occurrence of a transition  $\pi_{trans}(e)$  at time  $\pi_{time}(e)$ . Note that transition events can have any number of attributes (just like log events). These attributes will be used to determine delays and to correlate transitions occurrences. Alignments will be used to relate log events and transition events (see Section 4). Therefore, we only consider the Petri net structure  $N = (P, T, F)$  and abstract from the initial marking, token colors, guards, arc expressions, etc.

A *binary token flow* refers to the production of a particular token by some transition occurrence and the subsequent consumption by another transition occurrence. This relates to the notion of a place in an occurrence net and the notion of token flows defined in [8]. However, we adapt terminology to be able to relate binary token flows to pairs of events in event logs. Section 4 shows how this is done. Binary token flows correspond to observations in performance spectra (i.e., the individual lines in Figure 2).

A binary token flow  $tf = (e_{prod}, p, e_{cons})$  refers to a transition occurrence  $e_{prod}$  producing a specific token for place  $p$  that is later consumed by another transition occurrence  $e_{cons}$ .

**Definition 4 (Binary Token Flow).** Let  $N = (P, T, F)$  be a Petri net.  $\mathcal{U}_{BTF}^N = \{(e_{prod}, p, e_{cons}) \in \mathcal{U}_{TE} \times P \times \mathcal{U}_{TE} \mid \pi_{trans}(e_{prod}) \in \bullet p \wedge \pi_{trans}(e_{cons}) \in p \bullet \wedge \pi_{time}(e_{cons}) \geq \pi_{time}(e_{prod})\}$  are the possible binary token flows of  $N$ .

Note that a binary token flow corresponds to a *condition* in a *partially-ordered run* of the Petri net [7]. It is possible to describe an entire run of a net as a set of binary token flows. A necessary requirement is that the token flows in this set can only produce and/or consume one token for each place (of course, the same transition can consume or produce multiple tokens and a token cannot be consumed before it is produced). We call such a set a *valid token flow set*.

**Definition 5 (Valid Token Flow Set).** Let  $N = (P, T, F)$  be a Petri net structure.  $TFS \subseteq \mathcal{U}_{BTF}^N$  is a valid token flow set if for any pair of token flows  $(e_{prod}, p, e_{cons}) \in TFS$  and  $(e'_{prod}, p, e'_{cons}) \in TFS$  sharing the same place  $p$ :  $e_{prod} = e'_{prod}$  if and only if  $e_{cons} = e'_{cons}$  (i.e., tokens are produced and consumed only once).

A run of the Petri net  $N$  corresponds to a valid token flow set. Such a run may involve many different cases (e.g., all bags handled in a baggage handling system). Therefore, we do not define an initial marking or additional annotations (like in colored Petri nets). Section 4 explains how a valid token flow set can be extracted from an event log  $L \subseteq \mathcal{U}_{LE}$ .

Token flows abstract from tokens that are never produced or never consumed (e.g., tokens in the initial and final marking). If needed, it is always possible to add dummy source and sink transitions. Since an event log may contain information about many cases, this is more natural for process mining.

**Definition 6 (Notations).** For any binary token flow  $tf = (e_{prod}, p, e_{cons})$ , we use the following shorthands:  $\pi_{dur}(tf) = \pi_{time}(e_{cons}) - \pi_{time}(e_{prod})$  is the duration of a token flow,  $\pi_{tp}(tf) = \pi_{trans}(e_{prod})$  is the transition starting the token flow,  $\pi_{time}^{tp}(tf) =$

$\pi_{time}(e_{prod})$  is the production time,  $\pi_{tc}(tf) = \pi_{trans}(e_{cons})$  is the transition completing the token flow,  $\pi_{tc}^{time}(tf) = \pi_{time}(e_{cons})$  is the consumption time, and  $\pi_{pl}(tf) = p$  is the corresponding place. Let  $TFS \subseteq \mathcal{U}_{BTF}^N$  be a valid token flow set. For any subset of places  $Q \subseteq P$ :  $TFS \upharpoonright_Q = \{tf \in TFS \mid \pi_{pl}(tf) \in Q\}$  are the token flows through these places. We can also compute statistics over sets of token flows, e.g.,  $avgdur(TFS) = \sum_{tf \in TFS} \pi_{dur}(tf) / |TFS|$  is the average delay.

The above notions facilitate computations over token flows, e.g.,  $avgdur(TFS \upharpoonright_{\{p\}})$  is the average time tokens spend in place  $p$ . Minimum, maximum, median, variance, and standard deviation can be computed in a similar way.

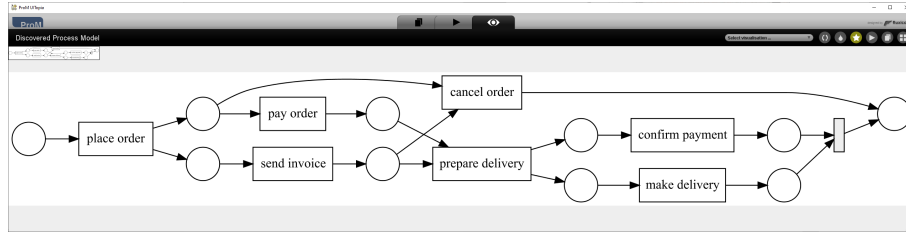
Each binary token flow  $tf \in TFS$  needs to be converted into an observation of the form  $(t_a, t_b, c)$  where  $a$  represents the start of the segment (i.e., the top line),  $b$  represents the end of the segment (i.e., the bottom line),  $t_a$  is the start time of the observation,  $t_b$  is the end time of the observation, and  $c$  is the class. This can be done as follows. For  $a$  we can pick all the transitions producing tokens for place  $p$  or a specific transition in the preset of  $p$ , i.e.,  $a = \bullet p$  or  $a \in \bullet p$ . For  $b$  we can pick all the transitions consuming tokens from place  $p$  or a specific one, i.e.,  $b = p \bullet$  or  $b \in p \bullet$ . For  $a = \bullet p$  and  $b = p \bullet$ , we create the *grouped multiset of observations*  $Obs_p = [(t_a, t_b, c) \mid tf \in TFS \upharpoonright_{\{p\}} \wedge t_a = \pi_{tp}^{time}(tf) \wedge t_b = \pi_{tc}^{time}(tf) \wedge c = class(tf)]$ . For  $a \in \bullet p$  and  $b \in p \bullet$ , we create the *ungrouped multiset of observations*  $Obs_{a,b,p} = [(t_a, t_b, c) \mid tf \in TFS \upharpoonright_{\{p\}} \wedge a = \pi_{tp}(tf) \wedge b = \pi_{tc}(tf) \wedge t_a = \pi_{tp}^{time}(tf) \wedge t_b = \pi_{tc}^{time}(tf) \wedge c = class(tf)]$ . We use the notations from Definition 6 to define these two types of multisets. We leave the definition of the classification function  $class$  open. It may be based on properties of the case (e.g. the customer involved), events (e.g., the outcome), or some performance related computation (e.g., overall case duration). A simple example is  $class(tf) = slow$  if  $\pi_{dur}(tf) \geq 24$  hours and  $class(tf) = fast$  if  $\pi_{dur}(tf) < 24$  hours.

Using the above, we can create one grouped performance spectrum for place  $p$  based on  $Obs_p$  or  $|\bullet p| \times |p \bullet|$  ungrouped spectra based on  $Obs_{a,b,p}$  with  $a \in \bullet p$  and  $b \in p \bullet$ .

## 4 Extracting Token Flows From Event Logs

Each binary token flow  $tf = (e_{prod}, p, e_{cons})$  in a valid token flow set (i.e., a run of the Petri net) corresponds to an observation (i.e., a line) in the corresponding performance spectrum. Attributes and computations such as  $\pi_{dur}(tf)$  can be used to add class labels (coloring of lines). Hence, given a valid token flow set  $TFS \subseteq \mathcal{U}_{BTF}^N$ , it is easy to construct performance spectra. In this section, we show how to obtain valid token flow sets from event data.

We start with an event log  $L \subseteq \mathcal{U}_{LE}$  and first discover a Petri net  $N = (P, T, F)$  using existing process discovery techniques (e.g., inductive mining, region-based discovery, etc.) [1]. The Petri net  $N$  may also be a discovered model that was modified or created manually. The event log  $L$  is replayed on the Petri net  $N$  by computing the so-called *alignments* [3, 4]. Alignments make it possible to link each case in the event log (i.e., tokens having a common case identifier) to a path through the model. This may imply that events are ignored because they do not fit (called a “move on log”) or that artificial events need to be introduced (called a “move on model”). In the latter case, also



**Fig. 4.** The Petri net model automatically discovered based on the event log with 39.022 orders and 210.459 events.

a timestamp needs to be added in-between the synchronous moves (i.e., fitting events before and after).

Alignments are typically computed per case and several approaches are available [3, 4]. They all map each case in the log onto a path through the model such that the differences are minimized. Let  $c$  be a case in event log  $L \subseteq \mathcal{U}_{LE}$ .  $\sigma_c = \{e \in L \mid \pi_{case}(e) = c\}$  are the events of  $c$  (often represented as sequence of activities). Replaying the firing sequence  $\sigma_c$  over  $N$  yields a collection of transition events  $\gamma_c \subseteq \mathcal{U}_{TE}$ . Note that events in  $\sigma_c$  may be ignored (“move on log”) and events may be added (“move on model”) due to silent transitions or deviating behavior.  $\gamma_c$  defines a partially-ordered run [7] for a particular case  $c$ . Each condition in such a run describes a token in place  $p$  with its producing event  $e_{prod}$  and its consuming event  $e_{cons}$ . This yields the binary token flows for case  $c$ . Taking the union over all cases yields the overall valid token flow set  $TFS \subseteq \mathcal{U}_{BTF}^N$  for event log  $L$ . As shown in Section 3, such a  $TFS$  can be visualized in terms of performance spectra.

## 5 Implementation

The approach has been implemented as a ProM plug-in. Figure 1 already showed a screenshot analyzing a Petri net discovered from an event log with information about 39.022 orders and 7 activities (210.459 events in total). Figure 4 shows a larger visualization of the discovered process model.

To use the plug-in, install the package “PerformanceSpectrumIntegration” in combination with ProM’s latest Nightly Build from *promtools.org*. The plug-in “Interactive Performance Spectrum” takes as input an event log and a Petri net.<sup>4</sup> A possible scenario is to take an arbitrary XES event log, run the “Inductive Visual Miner”, save the result as an “Accepting Petri Net”, and then run the “Interactive Performance Spectrum” on the event log and accepting Petri net. Using these simple steps, one immediately gets the desired result and can now interactively show performance spectra side-by-side with the process model.

Alignments [3, 4] play a key role in constructing token flows. The “Interactive Performance Spectrum” plug-in first applies the standard alignment techniques to the event

<sup>4</sup> Note that there are several variants depending on the type of Petri net, whether the alignment remains to be constructed, and to configure the class attribute.





**Fig. 5.** Screenshot showing three performance spectra marked ❶, ❷, and ❸. The blue diamond was added to the process model to analyze the token flow between *pay order* and *make delivery*. The colors differentiate between frequent variants (orange) and infrequent variants (green).

log and the discovered Petri net. This is typically the biggest performance bottleneck and not specific to our approach (alignments are also needed to compute simple statistics for places such as the average waiting time). After this initialization step, the plug-in shows the Petri net and a variable number of performance spectra. Figure 1 shows three performance spectra. The top one is an aggregated performance spectrum with information about the token flow between activities *place order* and *send invoice* (the blue place). The two other performance spectra show details about the token flow of the green place, i.e., the place connecting activities *place order* and *pay order* and *cancel order*. The colors of the lines and bars are based on the overall flow time which is used as a class attribute.

Figure 5 shows another screenshot of the “Interactive Performance Spectrum” while analyzing the same process in a different period. Three performance spectra (marked ❶, ❷, and ❸) are shown. The first one (i.e., ❶) is a detailed performance spectrum for the place connecting *prepare delivery* with *confirm payment*. As can be seen, work is sometimes queueing for *confirm payment* (see the gaps and diverging lines) and things are handled in FIFO (First-In-First-Out) order (there are no crossing lines). The second performance spectrum (i.e., ❷) shows the token flow for the place connecting *prepare delivery* with *make delivery*. Again we see queueing. However, now things are handled in a LIFO (Last-In-First-Out) order as illustrated by the crossing lines. The orange lines refer to token flows belonging to frequent variants and the green lines refer to token flows belonging to infrequent variants (i.e., cases following a less common path through the model). The third performance spectrum (i.e., ❸) refers to an artificially added place (indicated by the blue diamond in the Petri net). The virtual place connects *pay order* and *make delivery* and was added to analyze the token flow between these two activities. The tool facilitates the addition of such virtual places without recomputing alignments (the places do not need to be implicit and do influence the alignments; they are merely used to generate additional diagnostic token flows). Moreover, both detailed and aggregated performance spectra can be generated for such places.

By simply selecting places on the left-hand side, spectra are added on the right-hand side. It is also possible to add multiple virtual measurement places interactively. Given a place  $p$ , one can show one aggregated token-flow spectrum based on  $Obs_p$  or show individual spectra for all pairs of transitions in  $\{(a, b) \mid a \in \bullet p \wedge b \in p \bullet\}$  based on  $Obs_{a,b,p}$  (see Section 3). Moreover, there are many ways to adapt the view as illustrated by the bottom part of Figure 5:

- It is possible to seamlessly rescale the time axis and to zoom-in on selected time periods using two sliders.
- The density of the lines can be adapted and it is possible to overlay the spectra with weeks, days, hours, and seconds.
- For smaller event logs, the views are immediately updated, for larger event logs it is possible to make multiple changes and then update the view.
- Performance spectra can be filtered based on attributes of events in the log or derived attributes such as duration or variant.
- Performance spectra use colors that may be based on event attributes or derived attributes, i.e., the  $class(tf)$  introduced before. This way one can, for example, see differences between products, customer groups, and departments. However, it can also be used to highlight exceptional or slow cases.

For aggregated performance spectra (like ⑤ in Figure 5) we need to select the time period used for binning the token flows. We can use years, quarters, months, weeks, days, days, hours, minutes or a customized time period. The aggregated performance spectrum in Figure 5 uses day as a time period. Each stacked bar corresponds to one day. The height refers to the number of token flows and the colors refer to classes involved. One can clearly see the weekends and the varying load and changing mix of cases. Obviously, one cannot see such phenomena using traditional performance visualizations.

The “Interactive Performance Spectrum” plug-in has been applied to most of the publicly available real-life event logs from the 4TU Center for research data.<sup>5</sup> Experiments show that the biggest bottleneck is in the discovery (getting a reasonable Petri net) and alignment computation (which may be very time consuming).

## 6 Conclusion

This paper presented the novel “Interactive Performance Spectrum” plug-in (available as part of ProM’s Nightly Build *promtools.org*) that visualizes token flows using performance spectra while providing a range of interaction mechanisms. This allows us to uncover real-life phenomena such as overtaking, batching, queueing, concept drift, and partial blocking that would otherwise remain invisible. By using a Petri net as the “lens” to look at event data, we overcome the limitations of our earlier work [6, 5] where concurrency was not handled and we did not support the link between performance spectra and process models. This results in problems similar to using directly-follows graphs for process discovery [2]. Due to the “Interactive Performance Spectrum” plug-in we are now able to handle concurrency and tightly integrate process models and event data.

<sup>5</sup> The site [https://data.4tu.nl/repository/collection:event\\_logs\\_synthetic](https://data.4tu.nl/repository/collection:event_logs_synthetic) provides 24 sets of real-life event data and 15 synthetic data sets.

The applicability of the approach is mostly limited by the potential complexity of alignment computations. For large event logs and models, preprocessing may be time-consuming. Therefore, as future work, we would like to explore approximative alignments and apply various forms of sampling.

**Acknowledgments** We thank the Alexander von Humboldt (AvH) Stiftung and Vanderlande Industries for supporting our research.

## References

1. W.M.P. van der Aalst. *Process Mining: Data Science in Action*. Springer-Verlag, 2016.
2. W.M.P. van der Aalst. A Practitioner’s Guide to Process Mining: Limitations of the Directly-Follows Graph. In *International Conference on Enterprise Information Systems (Centeris 2019)*, volume 164 of *Procedia Computer Science*, pages 321–328. Elsevier, 2019.
3. W.M.P. van der Aalst, A. Adriansyah, and B. van Dongen. Replaying History on Process Models for Conformance Checking and Performance Analysis. *WIREs Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.
4. J. Carmona, B. van Dongen, A. Solti, and M. Weidlich. *Conformance Checking: Relating Processes and Models*. Springer-Verlag, 2018.
5. V. Denisov, E. Belkina, D. Fahland, and W.M.P. van der Aalst. The Performance Spectrum Miner: Visual Analytics for Fine-Grained Performance Analysis of Processes. In *Proceedings of the BPM Demo Track*, volume 2196 of *CEUR Workshop Proceedings*, pages 96–100. CEUR-WS.org, 2018.
6. V. Denisov, D. Fahland, and W.M.P. van der Aalst. Unbiased, Fine-Grained Description of Processes Performance from Event Data. In *International Conference on Business Process Management (BPM 2018)*, volume 11080 of *Lecture Notes in Computer Science*, pages 139–157. Springer-Verlag, 2018.
7. J. Desel and W. Reisig. Place/Transition Nets. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 122–173. Springer-Verlag, 1998.
8. G. Juhas, R. Lorenz, and J. Desel. Unifying Petri Net Semantics with Token Flows. In G. Franceschinis and K. Wolf, editors, *Proceedings of the 30th International Conference on Applications and Theory of Petri Nets (Petri Nets 2009)*, volume 5606 of *Lecture Notes in Computer Science*, pages 2–21. Springer-Verlag, 2009.
9. D. Keim, J. Kohlhammer, G. Ellis, and F. Mansmann, editors. *Mastering the Information Age: Solving Problems with Visual Analytics*. VisMaster, <http://www.vismaster.eu/book/>, 2010.
10. A. Rogge-Solti, W.M.P. van der Aalst, and M. Weske. Discovering Stochastic Petri Nets with Arbitrary Delay Distributions from Event Logs. In *Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2013)*, volume 171 of *Lecture Notes in Business Information Processing*, pages 15–27. Springer-Verlag, 2014.
11. A. Rozinat, R.S. Mans, M. Song, and W.M.P. van der Aalst. Discovering Colored Petri Nets From Event Logs. *International Journal on Software Tools for Technology Transfer*, 10(1):57–74, 2008.