Online Discovery of Cooperative Structures in Business Processes

S.J. van Zelst, B.F. van Dongen, and W.M.P. van der Aalst

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{s.j.v.zelst,b.f.v.dongen,w.m.p.v.d.aalst}@tue.nl

Abstract. Process mining is a data-driven technique aiming to provide novel insights and help organizations to improve their business processes. In this paper, we focus on the cooperative aspect of process mining, i.e., discovering networks of cooperating resources that together perform processes. We use online streams of events as an input rather than event logs, which are typically used in an off-line setting. We present the Online Cooperative Network (OCN) framework, which defines online cooperative resource network discovery in a generic way. A prototypical implementation of the framework is available in the open source process mining toolkit ProM. By means of an empirical evaluation we show the applicability of the framework in the streaming domain. The techniques presented operate in a real time fashion and are able to handle unlimited amounts of data. Moreover, the implementation allows to visualize network dynamics, which helps in gaining insights in changes in the execution of the underlying business process.

Keywords: Process Mining · Process Enhancement · Event Streams · Cooperative Resource Networks

1 Introduction

The goal of *process mining* [1] is to improve business processes by analyzing event logs. An event log captures the behavior of a business process by means of sequences of executed business process *events*. Process mining consists of three fields: *process discovery*, *conformance checking* and *process enhancement*. Process discovery aims at discovering a process model based on an event log. Conformance checking aims at assessing whether a process model conforms w.r.t. behavior recorded in an event log. Finally, process enhancement aims at improving business process models, e.g., by discovering resource interaction, applying bottleneck analysis, etc. Most work within process mining is devoted to process discovery, however, various other perspectives are of interest in context of business processes. In particular, the *organizational perspective* is of interest, i.e., interaction of employees, resources, devices etc. within the business process. Analyzing this perspective is useful for process enhancement as it reveals deficiencies in resource utilization, cooperation, sub-contracting schemes etc.

Work aiming at the organizational perspective [2], [15], [10], [14], typically use event logs as a primary input, and thus result in a historical/static view. Processes are subject to change and evolution, hence, we are often interested in a recent/dynamic view of the process. The tight alignment of IT and business processes enables us to capture events at the moment they occur. Analyzing such real-time, online *streams of events* enables us to inspect a recent view of the process. However, the high emission rate and potential unboundedness of the event stream forces us to design novel efficient methods that are able to cope with infinite data. Additionally, as the number of events recorded for operational processes increases tremendously each year, and, new data sources become intertwined with operational processes (sensor networks, mobile devices, etc.), event logs may exceed a computer's physical memory and hard disk(s). As we assume event streams to be infinite, the techniques presented in this paper additionally allow us to analyze conventional event logs of arbitrary size.

In this paper, we present the ***O**nline **C**ooperative **N**etworks* (OCN) framework, i.e., a new organizational process enhancement method that describes online cooperative resource network discovery. The framework allows the user to view multiple cooperative resource networks, based on a single event stream. Events on the stream are assumed to (at least) contain information about (1) the *case* for which the event occurred, (2) the activity that was performed and (3) the resource that performed the activity. The framework defines means to organize the data observed on the stream in a finite data structure. The user inspects a cooperative network in which new events are incorporated within the network as fast as possible, i.e., in a real-time and incremental fashion. Whenever the user decides to inspect a different network, e.g., changing from a *handover of work* to a *reassignment* network, an initial network is computed on the basis of the current data structure. Hence, a property of the proposed approach is that such switch does not need any form of *warm-up period*, i.e., after initialization of the network new events are immediately incorporated in the network.

The remainder of this paper is organized as follows. In Section 2, we present background material covering conventional discovery of cooperative resource networks. In Section 3 we present the OCN framework. Section 4 details on the implementation of the OCN framework within the process mining toolkit ProM [16]. In Section 5, we present empirical results w.r.t. event processing time, memory usage etc. of some example cooperative metrics. Section 7 concludes the paper and discusses future work.

2 Background

This section provides a brief summary of the typical input and output artifacts of conventional cooperative resource network discovery. To ease the reader, we build on top of the following mathematical notation. Let $X = \{e_1, e_2, ..., e_n\}$ denote a set of n elements. $\mathbb{N} = \{1, 2, 3, ...\}$ denotes natural numbers, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ includes 0, and $\mathbb{R}$ denotes real numbers. A sequence σ of length n over set X is a function $\sigma : \{1, 2, ..., n\} \rightarrow X$. A sequence is written as $\sigma = \langle \sigma(1), \sigma(2), ...,$

Table 1: Fictional example of events stored within an information system.

Identifier	*Case*	*Activity*	*Time-Stamp*	*Resource*
...	...	...	...	...
1	c_1	Register Payment	2016/07/07 09:05:34	John
2	c_2	Create Fine	2016/07/07 09:07:34	Lucy
3	c_1	Close Case	2016/07/07 09:08:15	Rob
4	c_2	Send Fine	2016/07/07 09:15:23	Rob
5	c_2	Register Payment	2016/07/07 10:02:23	John
...	...	...	...	...

$\sigma(n)\rangle$. For concatenation we write $\sigma_1 \cdot \sigma_2$, and, ϵ denotes the empty sequence. X^* denotes the set of all possible sequences over X.

2.1 Event Logs

A single execution of a process is referred to as a *process instance*. A process instance is related to a specific *case* for which the process is being executed. As an example consider a business process handling road fines. Typically, the process is executed per driver's offense, hence, an offense can be considered a case. The actual activities performed for a case may depend on data attributes related to the case. If the offense is related to illegal parking, sending a fine to the offender suffices. However, if the offense is related to speeding, apart from sending a fine, the driver's license might need to be revoked.

The *activities* executed for a process are stored in a company's information system's database. Consider Table 1 as an example, where a row describes the individual execution of an activity, which we refer to as an *event*. Events are unique and have a *unique identifier*. Events describe the *case*, at what *time*, and, by which *resource* an activity was performed. Often other data attributes are available for an event, i.e., an event is a vector/tuple of data values, e.g., *(5, c_2, Register Payment, 2016/07/07 10:02:23, John)*. Based on the case column we are able to extract a *sequence of events* executed for a specific case, also referred to as a *trace*. In this paper we are merely interested in the activity performed for a case, and, the resource executing the activity. Let $\mathcal{I}$ denote the universe of unique identifiers, let $\mathcal{C}$ denote the universe of cases, $\mathcal{A}$ the universe of activities and $\mathcal{R}$ the universe of resources. We define the universe of events as $\mathcal{E} = (\mathcal{I} \times \mathcal{C} \times \mathcal{A} \times \mathcal{R})$. An event $(\iota, c, a, r) \in \mathcal{E}$ is a quadruple describing (1) the event's unique identifier ι, (2) the corresponding *case* c, (3) activity a that was performed, and, (4) resource r that performed the activity. To access individual elements of an event, we define projection functions, i.e., given event $e = (\iota, c, a, r) \in \mathcal{E}$, we have $\pi_{\mathcal{I}}(e) = \iota$, $\pi_{\mathcal{C}}(e) = c$, $\pi_{\mathcal{A}}(e) = a$ and $\pi_{\mathcal{R}}(e) = r$.

An *event log* acts as the main source of input for most conventional process mining techniques. An event log is typically extracted from a company's information system and is defined as a set of traces. The order of events within a trace is usually imposed by time-stamp information.

Definition 1 (Event Log). *Let $\mathcal{E}$ denote the universe of events. An event log L is a set of traces, i.e., $L \subseteq \mathcal{E}^*$. A case $c \in \mathcal{C}$ uniquely defines a trace, i.e., for*

all $\sigma \in L$ *and* $1 \leq i, j \leq |\sigma|$ *we have* $\pi_{\mathcal{C}}(\sigma(i)) = \pi_{\mathcal{C}}(\sigma(j))$*, and, for all* $\sigma, \sigma' \in L$ *s.t.* $\sigma \neq \sigma'$*, and,* $1 \leq i \leq |\sigma|$*,* $1 \leq j \leq |\sigma'|$*, we have* $\pi_{\mathcal{C}}(\sigma(i)) \neq \pi_{\mathcal{C}}(\sigma'(j))$*. Event are unique, i.e., for all* $\sigma \in L$ *and* $1 \leq i < j \leq |\sigma|$ *we have* $\sigma(i) \neq \sigma(j)$*, and, for any* $\sigma, \sigma' \in L$ *and* $1 \leq i \leq |\sigma|$*,* $1 \leq j \leq |\sigma'|$*, we have* $\sigma(i) \neq \sigma'(j)$*.*

As an example of a trace consider case c_2 in Table 1 describing $\sigma_{c_2} = \langle(2, c_2,$ Create Fine, Lucy$), (4, c_2,$ Send Fine, Rob$), (5, c_2,$ Register Payment, John$)\rangle$. Note that multiple traces might exist that share the same type of behavior, i.e., there may be more traces for some case c_i of the form $\langle(x, c_i,$ Create Fine, Lucy$), (y, c_i,$ Send Fine, Rob$), (z, c_i,$ Register Payment, John$)\rangle$.

2.2 Discovering Cooperative Resource Networks

In this section we present a general definition of resource networks based on event logs which we, in Section 3, adopt into an event stream context.

In [2] a wide range of *cooperative metrics* is defined on the basis of event logs. In essence, all these metrics are defined over pairs of resources, e.g., given a certain pair of resources, what is the value for the *handover of work metric*?, how strong is the *working together metric*?, what is the *Minkowski distance* for the *joint case metric*? etc. Given the universe of resources $\mathcal{R}$, we define a cooperative metric μ as $\mu : \mathcal{R} \times \mathcal{R} \to \mathbb{R}$. A *cooperative resource network* is simply a weighted (un)directed graph, having some $V \subseteq \mathcal{R}$ as vertices, a set $E \subseteq \mathcal{R} \times \mathcal{R}$ as edges, and, μ as an edge weight function.

Definition 2 (Cooperative Resource Network). *Let* $\mathcal{R}$ *denote the universe of resources, let* $V \subseteq R$*, let* $E \subseteq \mathcal{R} \times \mathcal{R}$*, and, let* $\mu : \mathcal{R} \times \mathcal{R} \to \mathbb{R}$ *be a cooperative metric. Resource Network* N *is a weighted (un)directed graph* $N = (V, E, \mu)$*.*

In Definition 2, V, E and μ are not connected. In practice, μ often defines V and E, e.g., given a metric μ s.t. $0 \leq \mu(r_1, r_2) \leq 1, \forall r_1, r_2 \in \mathcal{R}$, then $V = \{r \in \mathcal{R} \mid \exists_{r' \in \mathcal{R}}(\mu(r, r') > 0 \vee \mu(r', r) > 0)$, and, $E = \{(r_1, r_2) \in \mathcal{R} \times \mathcal{R} \mid \mu(r_1, r_2) > 0\}$.

Consider the following *handover of work* metric as an example. Given a non-empty event log L and a pair of resources $r_1, r_2 \in \mathcal{R}$, for any $\sigma \in L$ we define:

$$|r_1 \succ_\sigma r_2| = \sum_{i=1}^{|\sigma|-1} \begin{cases} 1, \text{ iff } \pi_{\mathcal{R}}(\sigma(i)) = r_1 \wedge \pi_{\mathcal{R}}(\sigma(i+1)) = r_2 \\ 0, \text{ otherwise} \end{cases} \tag{1}$$

The relation $|r_1 \succ_\sigma r_2|$ denotes the number of times a handover occurs from resource r_1 to resource r_2 within σ. Reconsider the event data depicted in Table 1 where, for trace σ_{c_2} related to case c_2, we have $|\text{Lucy} \succ_{\sigma_{c_2}} \text{Rob}| = 1$, $|\text{Rob} \succ_{\sigma_{c_2}} \text{Lucy}| = 0$ etc. In general $|r_1 \succ_\sigma r_2|$ describes a *local trace-based metric*. To express handover of work in a global, event log based metric, we define $r_1 \rhd^L r_2$:

$$r_1 \rhd^L r_2 = \left(\sum_{\sigma \in L} |r_1 \succ_\sigma r_2|\right) \Big/ \left(\sum_{\sigma \in L} (|\sigma| - 1)\right) \tag{2}$$

As a simple example, assume that cases c_1 and c_2 are the only two traces in the example event data of Table 1. Observe that we have John $\rhd^L$ Rob = Lucy $\rhd^L$

Rob = Rob $\triangleright^L$ John = $\frac{1}{3}$. In [2] a more generalized definition is presented that allows us to incorporate relations between resources at longer distances, e.g., given a distance of 2, Lucy also hands over work to John in case c_2. For simplicity we only consider relations of distance 1 within this paper.

3 Discovering Cooperative Structures Online

When we apply $\triangleright^L$ on a real event log, e.g., the BPI Challenge 2012 event log [9], we get the network depicted in Fig. 1.

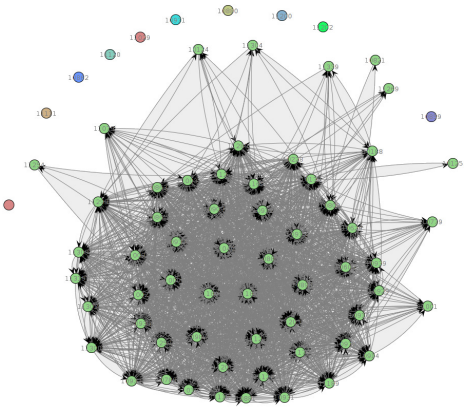


Fig. 1: Result of calculating $\triangleright^L$ based on the BPI Challenge 2012 [9].

Apart from observing that various resources hand over work to each other, we are not able to gain any valuable insights. *Moreover, we are not able to deduce whether certain resources became more/less active, stopped working etc.* Due to the tight alignment of IT and business processes we are able to capture events in an real-time fashion, i.e., at the moment they occur. Therefore, we propose to discover social networks on the basis of such *event streams*. To this purpose, we present the Online Cooperative Network (OCN) framework, which allows us to discover and visualize a live view of different cooperative networks, based on event streams. Additionally we discuss the update complexity of resource network metrics in a streaming context. Prior to this, we define the notion of *event streams*, which we formalize as a possibly infinite *sequence of events*.

Definition 3 (Event Stream). *Let $\mathcal{E}$ denote the universe of events. An event stream is a (infinite) sequence of events, i.e., $\mathcal{S} : \mathbb{N} \to \mathcal{E}$. Given some $i \in \mathbb{N}$, $\mathcal{S}(i)$ denotes the i^{th} event on the event stream. Each event in an event stream is unique, i.e., $\mathcal{S}$ is injective: for $i, j \in \mathbb{N}$ $\mathcal{S}(i) = \mathcal{S}(j) \implies i = j$.*

Note that event logs and event streams differ significantly. First, an event log has *final information* regarding cases, i.e., each trace is completed and thus the information on some case c does not change. For event streams this is not the case, i.e., after receiving the first i events we have seen j events related to c, $(j \leq i)$. Within the next i' activities we may see j' events related to case c, $(j' \leq i')$. Thus, the trace of case c after receiving i events may differ from the trace for case c after receiving $i + i'$ events. Second, an event log is assumed to be finite, whereas we assume event streams to be infinite. Hence, we need ways to temporarily store traces seen on the event stream, and moreover,

ways to remove events and/or traces at some point in time. Since transforming conventional event logs to an event stream is trivial, we are able to analyze any event log with the proposed techniques.

3.1 The OCN Framework

Given a stream of events, we are interested in constructing resource networks. Since multiple cooperative metrics are defined, we aim at designing a framework that allows us to discover several different networks on an event stream. Moreover, the user needs to be able to seamlessly switch between different networks. Hence, we present the OCN framework, depicted in Fig. 2, which describes this in a generic way.

From each new event, (activity,resource)-pair information (i.e. (a, r)) is extracted. In *case administration* $\mathcal{D}^{\mathcal{C}}$, we store for each case the (partial) sequence of (activity,resource)-pairs seen so far. Update function λ constantly updates $\mathcal{D}^{\mathcal{C}}$, and handles case/event deletion. At any point in time, we are able to query $\mathcal{D}^{\mathcal{C}}$ what sequences of events are stored for a (collection) of case(s). Due to the finite nature of $\mathcal{D}^{\mathcal{C}}$, enforced by λ, the answer to such query changes over time, i.e., $\mathcal{D}^{\mathcal{C}}$ *discards old data*. After receiving i events, the user inspects some cooperative network N_i^j. Network N_i^j is constantly updated by γ^j, in a real-time fashion. The information stored in $\mathcal{D}^{\mathcal{C}}$ allows the user to switch the current network N_i^j to an other network $N_i^{j'}$, e.g., changing from a *handover of work* network to a *reassignment* network.

We identify four main components: (1) case administration $\mathcal{D}^{\mathcal{C}}$, (2) case administration update function λ, (3) resource network builders γ^1, γ^2, ..., γ^n, and, (4) resource networks N^1, N^2, ..., N^n. Only one network builder is active at a specific point in time (γ^j in Fig. 2). In the remainder of this section we discuss the components in more detail.

Case Administration Most cooperative metrics are defined on the basis of resource information present in traces within event logs. Hence, in an event

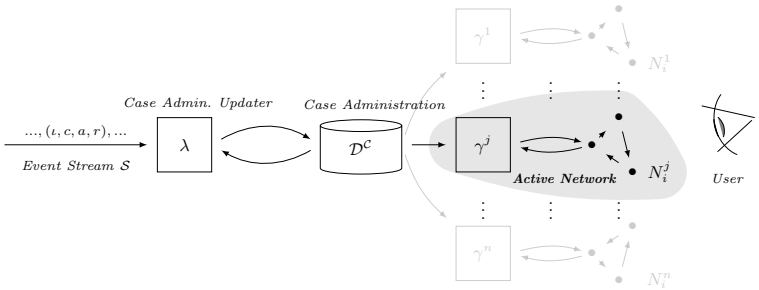


Fig. 2: Schematic overview of the *OCN framework*.

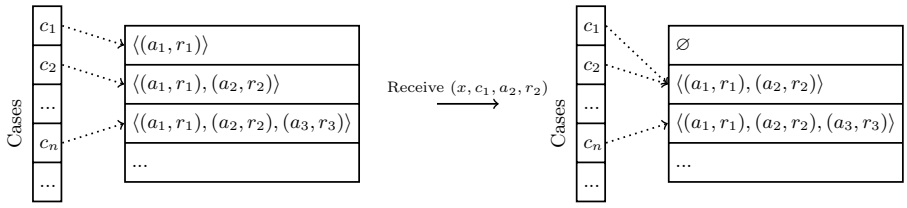


Fig. 3: A simple example of a case administration $\mathcal{D}^{\mathcal{C}}$.

stream setting, we need a data structure, i.e., a *case administration*, that allows us to determine, for a given case identifier, what sequence of events has thus far been emitted on the event stream. Since we receive events over time, the case administration evolves over time. We define the case administration as a function $\mathcal{D}^{\mathcal{C}} : (\mathbb{N}_0 \times \mathcal{C}) \rightarrow (\mathcal{A} \times \mathcal{R})^*$. Given $i \in \mathbb{N}_0$ and $c \in \mathcal{C}$, $\mathcal{D}^{\mathcal{C}}(i, c)$ is the sequence of (activity,resource)-pairs related to case c observed on the event stream, after i events. This sequence is not necessarily containing all events ever observed/observable for case c, i.e., as the memory is finite, events related to case c might be removed, and/or, new events related to case c might arrive later.

To keep the case administration up to date, we need some update function λ, which updates the case administration based on the i^{th} event. A simple initial design of a case administration is depicted in Fig. 3. We maintain a collection, e.g., an array, consisting of case identifiers seen earlier on the event stream. Each case identifier points to a corresponding sequence. For case c_1 we have seen $\langle a_1, r_1\rangle$. If we receive new event (x, c_1, a_2, r_2), we update c_1's pointer to $\langle(a_1, r_1), (a_2, r_2)\rangle$. As there is no case identifier pointing to $\langle(a_i, r_i)\rangle$ we remove it from the collection of sequences. Since the stream is possibly infinite, we need an *aging mechanism* that allows us to forget some of the data elements stored in the case administration. We identify two aging strategies, i.e., *case level* and *event level*. In the case level strategy, we store all events for a given case identifier until we decide to completely remove the case, and all its related events, from the administration. In the event level strategy, we remove individual events from the case administration.

There are several ways to design case level strategy data structures, primarily based on the field of data stream analysis [13], [3], [11]. For example, if we are interested in maintaining a view of the most recent cases, we use a *decay function* on the cases, that allows us to determine whether a case is eligible for deletion [8]. If we are on the other hand interested in the most active, or, most frequently updated cases, we are able to use frequency approximation algorithms defined for data streams [7]. When the underlying technique removes a case due to its age or relative infrequency, the associated sequence is removed.

In the theoretical case where we only have one active case within the process that has an infinite sequence of different events, the case level strategy does not work. In such case we adopt an event based strategy. As an example consider the following approach. We maintain three data structures, two of which as depicted

in Fig. 3, i.e., a collection of cases pointing to a collection of sequences. The third data structure is simply an array consisting of case identifiers that have been emitted on the stream. Note that we store each case identifier of each event, i.e., if we have received n events related to case c, we have (at most) n occurrences of c within the array. We use a decay function to apply aging on the elements of the case identifier array [8]. Whenever the decay function indicates that a case is eligible for deletion we remove the first (activity,resource)-pair of the sequence that the case is currently pointing to. If the case now points to the empty sequence, we remove it from the collection of cases.

Networks & Builders We maintain knowledge about the cases seen so far in memory to be able to switch from one resource network to the other. Hence, the network discovery components $\gamma^1, \gamma^2, ..., \gamma^n$, consist of two separate tasks: (1) *initialization* of the network based on the current state of $\mathcal{D}^{\mathcal{C}}$ and (2) *update* of the network based on new events flowing in. The first step, i.e., initialization, equals the conventional computation of the resource network. The second step, i.e., updating, is of particular interest. Consider that we obtain some network N_{i-1}, based on $\mathcal{D}^{\mathcal{C}}$ after $i-1$ events, and, we receive i^{th} event (ι, c, a, r). The new event either introduces, or, updates a network metric value for a pair of resources. Also, the metric value for a pair of resources usually depends on a global property, e.g., the divisor of $\rhd^L$, hence the new event also affects the metric value of other pairs of resources within the network. Thus, network N_i, based on $\mathcal{D}^{\mathcal{C}}$ after i events, is very likely to differ from the previous network N_{i-1}, i.e., a new handover is added/removed due to the new event/removed data. From an implementation perspective, we need to refresh the internal data structures that maintain the network after each new event. The complexity of this operation is expected to grow in terms of the network size. For most metrics we additionally need to design supporting data structures that allow us to recompute the actual metric value. For example, to be able to compute the $\rhd^L$ metric, for each resource pair, we need to maintain a denominator and a divisor. The denominator is resource pair specific, whereas the divisor can be maintained globally.

Some supporting data structures turn out to be computable in an incremental fashion. If the incremental computation of the data structures is inexpensive, we are able to update them in a real-time fashion. However, some metrics do not support inexpensive incremental updates of their supporting data structures, i.e., we potentially need to recompute them from scratch after a new event. In such case we need a more periodic update scheme of the network in order to maintain the network in a real-time fasion. Hence, we propose two network update strategies: *right-time*, i.e. at the user's request, and *real-time*, i.e. updating after each event, which define how we need to update our network over time.

3.2 Network Update Strategies

If we reconsider the *handover of work* metric as presented in Section 2.2, designing supporting data structures that allow us to maintain the metric is straightforward. We maintain a map $M_{\succ} : \mathcal{R} \times \mathcal{R} \rightarrow \mathbb{N}_0$, that maintains the denominator of

$r_1 \rhd_i^{\mathcal{D}^{\mathcal{C}}} r_2$, i.e., $\sum_{c \in \mathcal{C}} |r_1 \succ_{\mathcal{D}^c(i,c)} r_2|$ as the number of received events i increases over time. Moreover, since the divisor of $r_1 \rhd_i^{\mathcal{D}^{\mathcal{C}}} r_2$ has the same value for all possible combinations of r_1 and r_2 we maintain it i.e., $\sum_{c \in \mathcal{C}} (|\mathcal{D}^{\mathcal{C}}(i,c)| - 1)$, as a single integer $d \in \mathbb{N}$. Initially we have $M_{\succ}(r_1, r_2) = 0,\ \forall r_1, r_2 \in \mathcal{R}$. Assume that after a while, we receive some i^{th} event (ι', c', a', r') s.t. we have $\mathcal{D}^{\mathcal{C}}_{i-1}(c') = \sigma$, with $\sigma \neq \epsilon$. Thus, at time i, we have $\mathcal{D}^{\mathcal{C}}_i(c') = \sigma \cdot \langle (a', r') \rangle$. Moreover, assume that no case is removed from the case administration after receiving the i^{th} event. Observe that for any $r_1, r_2 \in \mathcal{R}$, for the denominator of $r_1 \rhd_i^{\mathcal{D}^{\mathcal{C}}} r_2$, we have:

$$\sum_{c \in \mathcal{C}} |r_1 \succ_{\mathcal{D}^c(i,c)} r_2| = \left(\sum_{c \in \mathcal{C} \setminus \{c'\}} |r_1 \succ_{\mathcal{D}^c(i,c)} r_2| \right) + |r_1 \succ_{\sigma \cdot \langle (a', r') \rangle} r_2| \quad (3)$$

We assumed that no sequence was removed from the case administration, thus:

$$\sum_{c \in \mathcal{C} \setminus \{c'\}} |r_1 \succ_{\mathcal{D}^c(i,c)} r_2| = \sum_{c \in \mathcal{C} \setminus \{c'\}} |r_1 \succ_{\mathcal{D}^c(i-1,c)} r_2| \quad (4)$$

We also have $|r_1 \succ_{\sigma \cdot \langle (a', r') \rangle} r_2| = |r_1 \succ_{\sigma} r_2| + b$, where $b = 1$ if $r' = r_2$, and, $\pi_{\mathcal{R}}(\sigma(|\sigma|)) = r_1$, else, $b = 0$. Combing this with Equations 3 and 4 yields:

$$\left(\sum_{c \in \mathcal{C} \setminus \{c'\}} |r_1 \succ_{\mathcal{D}^c(i-1,c)} r_2| \right) + |r_1 \succ_{\sigma} r_2| + b = \left(\sum_{c \in \mathcal{C}} |r_1 \succ_{\mathcal{D}^c(i-1,c)} r_2| \right) + b \quad (5)$$

Thus, the only action we need to perform is increasing the value of $M_{\succ}(r, r')$ by one, where r denotes the resource that executed the last event of σ. Note that, for the divisor, we are able to deduce $d_i = d_{i-1} + 1$. If we drop the assumption that no case is removed from the case administration, we need to reduce the values of $M_{\succ}$ and d accordingly. Updating d based on a dropped trace σ' is trivial, i.e., we simply subtract $|\sigma'| - 1$. For $M_{\succ}$ we iterate over σ' and for each pair of resources we decrease the corresponding $M_{\succ}$ value by one.

Clearly, the $\rhd$ metric is computable incrementally. Hence, the $\rhd$ metric is an example of a metric that we classify as a *real-time metric*. We define cooperative metrics to be real-time if it is possible to update the metric's supporting data structures by means of an incremental update. However, since the divisor changes, we need to recompute all metric values for those $r_1, r_2 \in \mathcal{R}$ with a non-zero $M_{\succ}$ value. Thus, in worst case this operation has complexity $O(|\mathcal{R}|^2)$.

There are examples of cooperative metrics that do not meet the real-time requirement. An example of such metric is the *boolean-causal variation* of the handover of work metric [2]. First, we introduce the notion of causality, after which we introduce the metric. Consider example event log L_2, in which we only record *sequences of activities*: $L_2 = \{\langle a_1, a_2, a_3, a_4 \rangle, \langle a_1, a_3, a_2, a_4 \rangle\}$. In the event log, a_1 is *directly followed* by a_2, which we write as $a_1 > a_2$. We additionally have $a_1 > a_3$, $a_2 > a_3$, $a_2 > a_4$, $a_3 > a_2$, and, $a_3 > a_4$. Two activities a_i and a_j are said to be in a *causal relation*, i.e., $a_i \rightarrow a_j$, iff $a_i > a_j$ and $a_j \not> a_i$. Thus, for the example event log we have $a_1 \rightarrow a_2$, $a_1 \rightarrow a_3$, $a_2 \rightarrow a_4$, and, $a_3 \rightarrow a_4$.

Given a sequence σ of activity-resource pairs, we define relation $r_1 \succeq_\sigma r_2$ which specifies that resource r_1 hands over work of a case to resource r_2, given that the corresponding executed activities are in a causal relation. Moreover, relation $r_1 \succeq_\sigma r_2$ does not count the number of occurrences of such handover, i.e., it only captures the fact that such handover occurred at least once in the trace.

$$r_1 \succeq_\sigma r_2 = \begin{cases} 1 & \text{iff } \exists_{1 \leq i < |\sigma|-1}(\pi_{\mathcal{R}}(\sigma(i)) = r_1 \wedge \pi_{\mathcal{R}}(\sigma(i+1)) = r_2 \wedge \pi_{\mathcal{A}}(\sigma(i)) \rightarrow \pi_{\mathcal{A}}(\sigma(i+1))) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

We now define a corresponding cooperative network metric $r_1 \trianglerighteq_i^{\mathcal{D}^{\mathcal{C}}} r_2$ that captures this globally, based on event streams. Let i denote the index of the latest received event on the event stream, then:

$$r_1 \trianglerighteq_i^{\mathcal{D}^{\mathcal{C}}} r_2 = \left(\sum_{c \in \mathcal{C}} r_1 \succeq_{\mathcal{D}^{\mathcal{C}}(i,c)} r_2 \right) \Big/ |\{c \in \mathcal{C} \mid \mathcal{D}^{\mathcal{C}}(i,c) \neq \epsilon\}|$$

Maintaining the divisor is in this case trivial. We again maintain $M_\succ : (\mathcal{R} \times \mathcal{R}) \rightarrow \mathbb{N}_0$, which represents the denominator of the $\trianglerighteq$ metric. However, to maintain the metric we need to additionally keep tack of the causal relations present within the traces. To maintain the causal relations we maintain a map $M_> : (\mathcal{A} \times \mathcal{A}) \rightarrow \mathbb{N}_0$ that maintains the $>$ relation. Based on $M_>$ we maintain a set $M_\rightarrow$ that maintains causal pairs. When we receive i^{th} event $e = (\iota, c, a, r)$ we increment entry $M_>(a', a)$ with one, where a' denotes the last activity of $\mathcal{D}^{\mathcal{C}}(i-1, c)$. Next, we update $M_\rightarrow$ and $M_\succ$. There are four different possibilities related to the causality of a' and a, all having different implications on the way $\trianglerighteq$ is computed, and, the corresponding computational complexity.

1. *Before processing e, we have* $M_>(a', a) > 0$ *and* $M_>(a, a') = 0$*, i.e.,* $a' \rightarrow a$ *already holds.* We check whether σ contains structure $\langle ..., (a'', r'), (a''', r), ... \rangle$, s.t. $a'' \rightarrow a'''$. If so, we do not update $M_\succ(r', r)$, otherwise, we increase $M_\succ(r', r)$ with one.
2. *Before processing e, we have* $M_>(a', a) = 0$ *and* $M_>(a, a') = 0$*, i.e.,* $a' \rightarrow a$ *starts to hold.* We apply the same procedure as described in 1.
3. *Before processing e, we have* $M_>(a', a) > 0$ *and* $M_>(a, a') > 0$*, i.e.,* $a' \rightarrow a$ *did not and still does not hold.* There is no need to update $M_\succ$.
4. *Before processing e, we have* $M_>(a', a) = 0$ *and* $M_>(a, a') > 0$*, i.e.,* $a \rightarrow a'$ *did hold, though no longer holds.* We now need to remove the contribution of all cases that have a trace containing structure $\langle ..., (a, r''), (a', r'''), ... \rangle$. We only need to reduce the $M_\succ(r'', r''')$ value if within the same trace there is no structure of the form $\langle ..., (a'', r''), (a''', r'''), ... \rangle$ s.t. $a'' \rightarrow a'''$. Hence, *we need to loop over all traces currently present in the case administration.*

The fourth case is problematic from an incremental point of view. Within all other cases, it suffices to analyze the trace related to c within the case administration. If the fourth case applies we need to check all cases within the case administration in order to update $M_\succ$. Depending on the size of the case administration this procedure might be time consuming. Moreover, the nature of the

underlying process, i.e., stable without a lot of variation vs. unstable/changing with variation in terms of causalities influences the complexity. When we remove cases from the case administration, we again need to check the four possible scenario's w.r.t. modifications in causality. In case of removal, a casual relation that is added results in the need to rescan the case administration.

The $\trianglerighteq$ metric is an example of a *right-time* metric, i.e., instead of continuously maintaining the supporting data structures, it should be recomputed at regular intervals, or, at the user's request. In Section 5 we empirically assess the computational complexity of both metrics $\triangleright$ and $\trianglerighteq$. We also assess how often we need to recompute the internal data structures of $\trianglerighteq$ based on real data, i.e., how often does a causal change happen? Prior to this, in Section 4, we present details regarding the implementation of the OCN framework.

4 Implementation

The OCN framework is implemented in the ProM [16] (`http://promtools.org`) framework (Fig. 4). ProM is the open-source standard for implementing process mining techniques. The OCN framework is distributed within the `StreamSocialNetworks` package.[1] The current implementation provides support for: *Generalized Handover of Work Metrics* [2, Definition 4.4], *Generalized In-Between Metrics* [2, Definition 4.7], *Working Together Metrics* [2, Definition 4.8], and, *Joint Activity Metrics* based on *Minkowski distance*, *Hamming distance* and *Pearson's correlation coefficent* [2, Definition 4.10]. Moreover, the framework is *easily extensible* to support more cooperative network metrics.

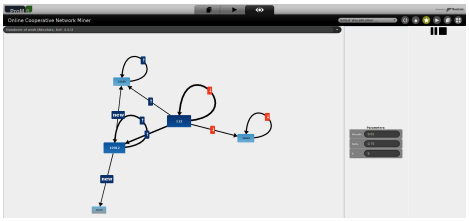


Fig. 4: The OCN framework in ProM.

Within the implementation, instead of maintaining sequences of events, a *prefix-tree* of (activity,resource)-pairs is built in memory. Each case within the case administration points to a node in the tree, that represents the latest (activity,resource)-pair received for the case. By walking from the root to a node pointed at by a case identifier, we get the corresponding trace. The user can choose between two visualizations: (1) visualize each consecutive network, or, (2) makes use of windows in order to *visualize changes* within the network(s). To visualize changes, the visualizer stores two windows each containing w networks. For each link within the network it computes the average corresponding metric value for both windows. The width of the links within the network are based on the relative change of the average link values. A link labeled with `new` indicates a new relation, a $\uparrow$ indicates an increase w.r.t. the previous window, a $\downarrow$ indicates a decrease, and a $\times$ label indicates that the relations was present in the first window, though is no longer present in the second window. The size and

[1] `https://svn.win.tue.nl/repos/prom/Packages/StreamSocialNetworks/Trunk`

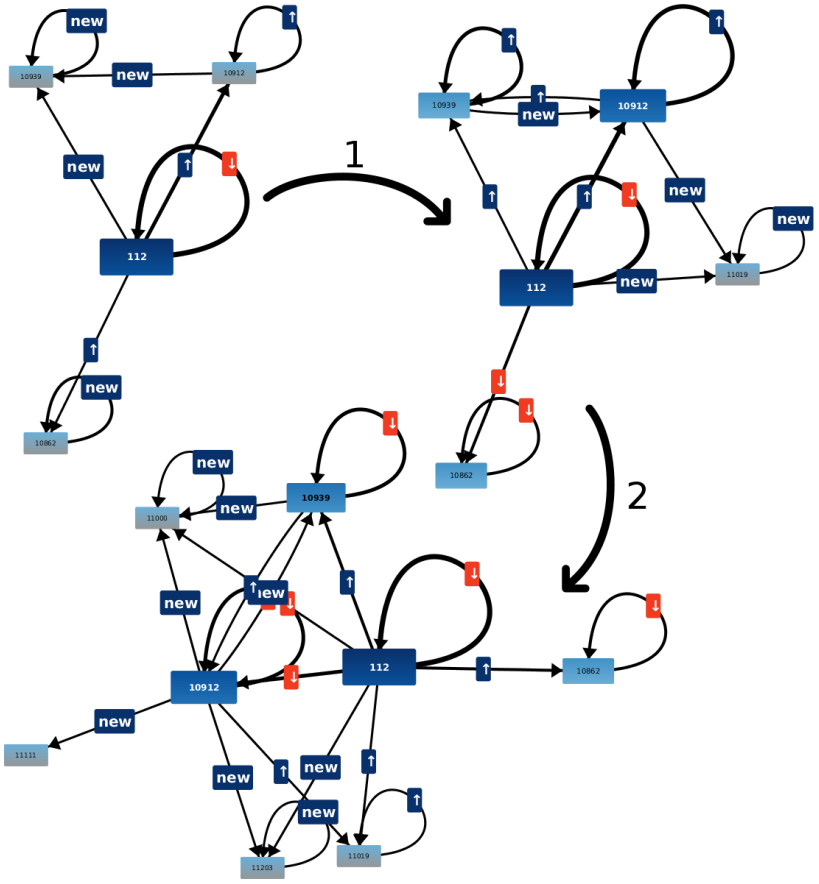


Fig. 5: Three consecutive window-based cooperative networks based on an event stream of the BPI Challenge 2012 event log [9].

color of the resources within the network are based on their relative involvement within the network as a whole.

To illustrate the usefulness and applicability of the framework, consider Fig. 5 which depict a sequence of subsequent window based cooperative networks. The networks depict the handover of work metric with a maximum distance of 5 events, a fading factor of 0.75, and, a window-size w of 50. In the first network, all resources hand over work to *themselves*. Some new relations become present, and, resource 10939 (top left) is completely new. Resource 112 is the most active resource, and involved in the most dominant relations (although its self-loop decreased in relative frequency). In the second network, we observe that resource 10912 became more active. Resource 10939 now also hands over work to resource 10912. Again a new resource became active, i.e., resource 11019 (bottom right), and, the relations that involve resource 10862 became relatively less frequent. In

the third network, again a lot of new behavior becomes apparent. Due to the self-loops, we conclude that most resources execute multiple subsequent activities for a case, or, execute subsequent activities *within a span of at most five activities*. There are several other interesting observations that we can derive from Fig. 5.

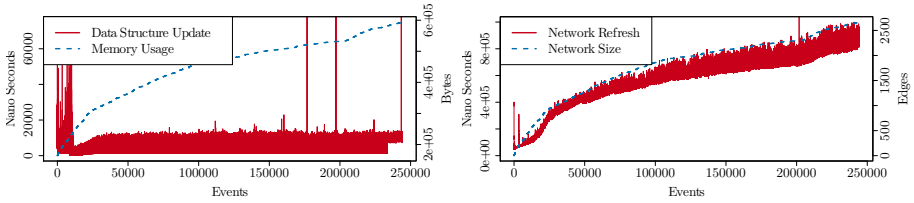
5 Evaluation

In this section we assess the scalability of the metrics described in Section 3.2, i.e., $\triangleright$ and $\trianglerighteq$.[2] For the experiments we created an event stream based on the BPI Challenge 2012 event log [9]. The event log contains *real event data*, related to a loan application process. In total, the event log contains 262.200 events. Of these events, 244.190 events actually describe what resource executed the activity.

Within the event stream, events are ordered based on timestamps. Using the event stream, for each metric we measured: (1) time needed to refresh the supporting data structure(s) (in nano seconds), (2) time needed to refresh the whole network (in nano seconds), (3) total memory consumption of the data structure(s) (in bytes), (4) network size (in number of edges). Moreover, for each metric we investigated two scenario's: (1) no restriction on the case administration's available memory, (2) finite memory restriction on the case administration (*case level removal strategy*) using a forward decay model [8] with an *exponential decay function* with a *decay rate* of 0.01 and a removal threshold of 0.01. To reduce the effects of outliers in terms of run time measurements, e.g. caused by the java garbage collector, we ran each experiment ten times, removed the minimal and maximal measurement values, and, averaged the remaining eight values. Additionally, in some charts the y-axes are restricted in order to highlight the overall trends, instead of outliers.

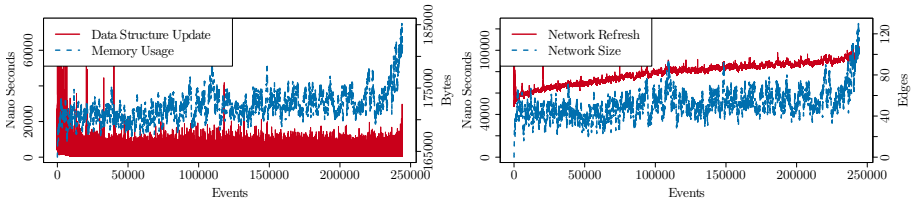
In Fig. 6 the results of discovering the $\triangleright$ metric, without any restriction on the available memory, are depicted. In Fig. 6a the memory usage combined with the data structure update time are depicted. Note that, since we do not restrict memory usage, no *cases are removed from the case administration*. As a result, the memory usage is steadily increasing. The data structure update time only consists of incrementally updating the divisor and the denominator values of the resource pairs affected by the update. We observe that initially there are some high values in terms of update time w.r.t. the overall trend. This is likely to be related to initialization of the underlying data structure. Then, after a short period of increasing update time values, the update time seems to stabilize. In Fig. 6b the network size combined with the network refresh time is depicted. When comparing the network size with the memory usage depicted in Fig. 6a we observe that they follow the exact same pattern. This makes sense since the

[2] Experiments are performed on four `Dell PowerEdge R520, 2 x Intel Xeon E5-2407 v2 2.40GHz, 8 x 8 8GB RDIMM` machines running `Ubuntu 14.04 LTS`. Experiment source code is available at: `https://svn.win.tue.nl/repos/prom/Packages/StreamSocialNetworks/Branches/publications/2016_coopis/`. Raw experiment results are available at `https://github.com/s-j-v-zelst/research/releases/download/final/2016_coopis_experiments.tar.gz`.



(a) Data structure updates and memory. (b) Network refresh time and size.

Fig. 6: Results of discovering the $\triangleright$ metric without memory restrictions.



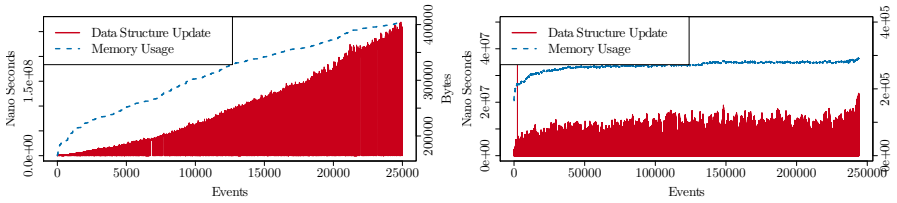
(a) Data structure updates and memory. (b) Network refresh time and network size.

Fig. 7: Results of discovering the $\triangleright$ metric with memory restrictions.

larger the network, the more absolute and relative values we need to store. The time to refresh the network follows the same shape. This is as expected as we need to calculate more relative values as the network size increases.

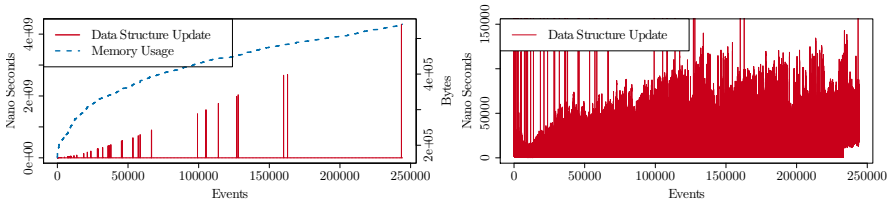
In Fig. 7 the results of discovering the $\triangleright$ metric, with finite memory restriction, are depicted. In Fig. 7a the memory usage combined with the data structure update time is depicted. In this case we observe that memory usage is fluctuating. When we compare the time needed for data structure updates, we observe that this is now slightly higher than in Fig. 6a. This is explained by the fact that when restricting memory to be finite, cases are dropped from the case administration. Again, the network refresh rate follows the behavior of the memory/network size. Due to the restrictions on memory usage, the refresh rate of the network is now comparable to the data structure update time.

For metric $\trianglerighteq$ we first computed a baseline in terms update times and memory usage, depicted in Fig. 8. Within the baseline we recompute the complete data structure and network after each event that we receive. In Fig. 8a the baseline without memory usage restrictions is depicted. Due to the high computational costs, we only performed measurements for the first 25.000 events. Moreover, we took the median values out of four experiments. The time needed to refresh the data structure grows tremendously fast. As the figure shows, refreshing the data structure constantly is not feasible in a streaming setting. In Fig. 8b the



(a) No memory restrictions. (b) Restricted memory.

Fig. 8: Baseline measurements for metric $\trianglerighteq$.



(a) Data structure updates and memory. (b) Data structure updates (zoomed in).

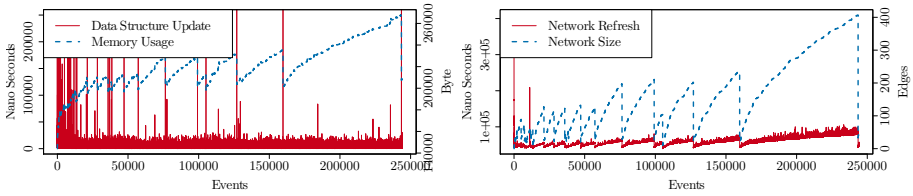
Fig. 9: Results of discovering the $\trianglerighteq$ metric without memory restrictions.

baseline with restricted memory use is depicted. In this case, the use of restricted memory limits the growth in terms of the data structure update time. Still, the times needed to update the data structure are infeasible in a streaming setting.

As explained in Section 3.2, we do not need to update the internal data structure every single time. Hence in Fig. 9 we depict the results of the $\trianglerighteq$ metric when only applying a data structure recalculation when it is indeed needed, without memory restrictions. In Fig. 9a we observe that a recalculation of the internal data structure does not happen very often, i.e., there is a limited number of excessive peeks in the chart. In Fig. 9b we zoomed in on the data structure update times. Note that, on average the time needed is much higher than in the case of $\triangleright$. This makes sense as in case of $\trianglerighteq$, we need to traverse the whole trace as opposed to a single constant update in case of $\triangleright$.

In Fig. 10 we depict the results of the $\trianglerighteq$ metric with restricted memory. The drops in memory usage in Fig. 10a correlate with the peeks in data structure update time, indicating that a recalculation of the data structure is needed at these points in time. Again, the network size, depicted in Fig. 10b follows the same shape as the memory usage. Likewise the time needed to refresh the network follows the network size.

In Table 2, for both $\triangleright$ and $\trianglerighteq$, the average values for the data structure update time and the network refresh time are presented. The results show that



(a) Data structure updates and memory. (b) Network refresh time and network size.

Fig. 10: Results of discovering the $\trianglerighteq$ metric without memory restrictions.

Table 2: Average values and standard deviations for the data structure update time and network refresh time.

Metric	*Memory*	*Avg./stdv. Data Structure Update (ns.)*	*Avg./stdv. Network Refresh (ns.)*
$\triangleright$	no restriction	8.459 / 2.436	567.652 / 199.991
$\triangleright$	restricted	2.176 / 2.285	78.817 / 9.896
$\trianglerighteq$	no restriction	133.333 / 14.834.991	393.190 / 138.375
$\trianglerighteq$	restricted	4.045 / 40.333	57.803 / 9.907

memory restriction has a positive influence on the time needed to update the data structures and refresh the network. Moreover, the $\trianglerighteq$ metric, in case of restricted memory, seems to need twice as much time compared to the $\triangleright$ metric. The experiments indicate that total recalculation of the network is not often needed. Hence, it is feasible to update the metric real-time, and, whenever we need to recompute the data structure, temporarily buffer new incoming events. The events can be processed in real-time after the data structures are refreshed.

6 Related Work

For an elaborate overview of process mining we refer to [1]. Here, we primarily focus on work related to the organizational perspective of process mining, and, applications of event streams within process mining.

In [2] a collection of social network metrics is defined in context of process mining, i.e., metrics based on event log data. The work can be regarded as one of the foundational works of the application of social network analysis in context of process mining. In [15] the authors extend the work of [2] with organizational model mining and information flow mining. In [10] the authors identify that applying conventional techniques as defined in [2], [15] result in complex networks that are not easy to understand. The authors propose to solve this by applying hierarchical clustering on the social networks to form clusters of similar resources. In [14] an extensible framework is proposed that allows for extracting resource behavior time series. It allows process owners to visualize their resource's performance over time, using event logs as an objective source of information. Recently, in [4], Appice et al. propose a method to analyze evolv-

ing communities within event logs. The method transforms the event log into a finite stream of events, and, subsequently divides the stream into windows. The method computes a resource community for each window and assess the changes within the communities of successive windows.

In online, event stream based process mining, the majority of the work is devoted to stream based process discovery, exclusively focusing on control-flow. In [6] an adaptation of the Heuristics Miner [17] for the purpose of event stream based process discovery is presented. In [12] the authors improve the implementation of [6] by the internal use of prefix-trees. In [5] methods for finding declarative process models on the basis of event streams are proposed.

To the best of our knowledge, this is the first work that considers the organizational perspective using *real time*, *online*, and, *infinite* event streams.

7 Conclusion

In this paper we presented the Online Cooperative Networks (OCN) framework, which allows us to discover several cooperative resource networks on the basis of event streams. The OCN framework in very general and in principle allows for many different types of analysis based on online/real-time event stream data. Moreover, it enables us to gain insights in cooperative networks over time, rather than providing one monolithic view. Due to the assumptions on the data, i.e., event streams might be infinite, the OCN Framework additionally allows us to analyze event logs that exceed a computer's physical memory. Therefore it can be used to speed up the analysis of large event logs and is not limited to streams.

Our experiments show that we are able to compute cooperative networks in an event stream setting. We have shown that there are cooperative network metrics suitable for real-time incremental update strategies. For these metrics, updating of the supporting data structures converges to a constant amount of time. The time needed to update the resource network however grows in terms of the size of the network. Additionally, we have shown that limiting the available memory has a positive impact both on the use of memory as on the network refresh times. However, as consequence of limited available memory we remove cases, which slightly increases the data structure update times.

Future Work The time needed to refresh the network is strongly related to the network size. However, in some cases, the effect of an increase in a metric's denominator/divisor might have little effect on the value of the metric, e.g. $\frac{1.001.337}{2.133.700} = 0,47 \approx \frac{1.001.338}{2.133.700}$. An interesting direction for future work is the integration of fast identification whether the network should be recalculated.

Within this paper we primarily focus on the feasibility of adopting cooperative resource network metrics in an event stream setting. To gain more value out of stream based resource network analysis, an interesting direction for future work is the assessment of different visualization methods of the evolution of the networks. Also, extensions such as community detection etc. are of interest.

References

1. Aalst, W.M.P. van der: Process Mining - Data Science in Action, Second Edition. Springer (2016)
2. Aalst, W.M.P. van der, Reijers, H.A., Song, M.: Discovering Social Networks from Event Logs. Computer Supported Cooperative Work 14(6), 549–593 (2005)
3. Aggarwal, C.C. (ed.): Data Streams - Models and Algorithms, Advances in Database Systems, vol. 31. Springer (2007)
4. Appice, A, Pietro M. di, Greco, C., Malerba, D.: Discovering and Tracking Organizational Structures in Event Logs. In: Ceci, M., Loglisci, C., Manco, G., Masciari, E., Ras, Z.W. (eds.) ECML-PKDD NFMCP, Revised Selected Papers. LNCS, vol. 9607, pp. 46–60. Springer (2015)
5. Burattin, A., Cimitile, M., Maggi, F.M., Sperduti, A.: Online Discovery of Declarative Process Models from Event Streams. IEEE Trans. Services Computing 8(6), 833–846 (2015)
6. Burattin, A., Sperduti, A., Aalst, W.M.P. van der: Control-Flow Discovery from Event Streams. In: IEEE CEC 2014. pp. 2420–2427. IEEE (2014)
7. Cormode, G., Hadjieleftheriou, M.: Methods for Finding Frequent Items in Data Streams. VLDB J. 19(1), 3–20 (2010)
8. Cormode, G., Shkapenyuk, V., Srivastava, D., Xu, B.: Forward Decay: A Practical Time Decay Model for Streaming Systems. In: Ioannidis, Y.E., Lee, D.L., Ng, R.T. (eds.) IEEE ICDE. pp. 138–149. IEEE Computer Society (2009)
9. Dongen, B.F. van: BPI Challenge 2012 (2012), `http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`
10. Ferreira, D.R., Alves, C.: Discovering User Communities in Large Event Logs. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops, Revised Selected Papers. Lecture Notes in Business Information Processing, vol. 99, pp. 123–134. Springer (2011)
11. Gama, J.: Knowledge Discovery from Data Streams. Chapman and Hall / CRC Data Mining and Knowledge Discovery Series, CRC Press (2010)
12. Hassani, M., Siccha, S., Richter, F., Seidl, T.: Efficient Process Discovery From Event Streams Using Sequential Pattern Mining. In: IEEE SSCI 2015. pp. 1366–1373. IEEE (2015)
13. Muthukrishnan, S.: Data Streams: Algorithms and Applications. Foundations and Trends in Theoretical Computer Science 1(2) (2005)
14. Pika, A., Wynn, M.T., Fidge, C.J., Hofstede, A.H.M. ter, Leyer, M., Aalst, W.M.P. van der: An Extensible Framework for Analysing Resource Behaviour Using Event Logs. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 564–579. Springer (2014)
15. Song, M, Aalst, W.M.P. van der: Towards Comprehensive Support for Organizational Mining. Decision Support Systems 46(1), 300–317 (2008)
16. Verbeek, H.M.W., Buijs, J.C.A.M., Dongen, B.F. van, Aalst, W.M.P. van der: XES, XESame, and ProM 6. In: Soffer, P., Proper, E. (eds.) Information Systems Evolution - CAiSE Forum 2010, Selected Extended Papers. LNBIP, vol. 72, pp. 60–75. Springer (2010)
17. Weijters, A.J.M.M., Aalst, W.M.P. van der: Rediscovering Workflow Models from Event-Based Data using Little Thumb. Integrated Computer-Aided Engineering 10(2), 151–162 (2003)