A MapReduce Approach for Mining Multi-Perspective Declarative Process Models

Christian Sturm, Stefan Schönig and Stefan Jablonski
University of Bayreuth, Germany

Keywords: Process Mining, Process Discovery, Multi-Perspective Process Model, Declare, MapReduce, Business Process Management.

Abstract: Automated process discovery aims at generating a process model from an event log. Such models can be represented as a set of declarative constraints where temporal coherencies can also be intertwined with dependencies upon value ranges of data parameters and resource characteristics. Existing mining tools do not support multi-perspective constraint discovery or are not efficient enough. In this paper, we propose an efficient mining framework for discovering multi-perspective declarative models that builds upon the distributed processing method MapReduce. Mining performance and effectiveness have been tested on several real-life event logs.

1 INTRODUCTION

Process mining is the area of research that embraces the automated discovery, conformance checking and enhancement of process models. Automated process discovery aims at generating a process model from an event log consisting of traces, such that each trace corresponds to one execution of the process. Each event in a trace consists as a minimum of an event class (i.e., the activity to which the event corresponds) and a timestamp. In some cases, other information may be available such as the performer of the activity as well as data produced by the event in the form of attribute-value pairs. Discovery is of particular value for processes that offer various options to execute them. Those processes are often referred to as flexible, unstructured or knowledge-intense. Frequently, procedural process models resulting from discovery are colloquially called Spaghetti models due to their complex structure (van der Aalst, 2011). Discovered process models can alternatively be represented as a set of declarative constraints, i.e., rules for directly representing the causality of the behaviour (Pichler et al., 2011). The advantages of declarative languages such as Declare (Pesic et al., 2007) or DPIL (Zeising et al., 2014) have been emphasized in the literature. It is also well known that behaviour is typically intertwined with dependencies upon value ranges of data parameters and resource characteristics (de Leoni et al., 2016). Therefore, Declare has been extended towards Multi-Perspective Declare (MP-Declare) (Burattin et al., 2015). However, state-of-the-art mining tools such as MINERful (Di Ciccio and Mecella, 2013; Di Ciccio and Mecella, 2015) and DeclareMiner (Maggi, 2013) do not support MP-Declare at this moment. In (Schönig et al., 2016) a first approach to enable the discovery of MP-Declare constraints has been proposed. However, it has not been investigated how this complex mining task can be performed in an efficient way.

In this paper, we address this open research problem by proposing an *efficient* mining framework for discovering MP-Declare models that leverages latest big data analysis technology and builds upon the distributed processing method *MapReduce*. We introduce parallelizable algorithms for discovering commonly used types of MP-Declare constraints. The proposed solution, however, can be applied to all other MP-Declare constraint types as well. In contrast to related solutions, the proposed framework considers traces in one direction solely which leads to a crucial benefit w.r.t. performance. Mining performance and effectiveness have been tested on several real-life event logs.

The paper is structured as follows. Section 2 introduces the language and semantics of MP-Declare. Section 3 describes the distributed framework we propose to speed up multi-perspective process discovery. Section 4 describes the implementation approach that we developed. Section 4.2 presents the evaluation of

Sturm, C., Schönig, S. and Jablonski, S.
A MapReduce Approach for Mining Multi-Perspective Declarative Process Models.
DOI: 10.5220/0006710305850595
In *Proceedings of the 20th International Conference on Enterprise Information Systems (ICEIS 2018)*, pages 585-595
ISBN: 978-989-758-298-1

our technique with real-life cases. Section 5 discusses related work before Section 6 that concludes the paper.

2 RESEARCH BACKGROUND

In this section, we first illustrate the research problem that we are addressing and summarize concepts of Declare, MP-Declare and MP-Declare mining.

2.1 Declarative Process Modelling

Declarative constraints are strong in representing the behaviour of flexible business processes. Modeling languages like Declare (Pesic et al., 2007) describe a set of *constraints* that must be satisfied throughout the process. Constraints, in turn, are based on *templates*. Templates are patterns that define parameterized classes of properties, and constraints are their concrete instantiations. Semantics of such languages are formalized using logics such as Linear Temporal Logic over finite traces (LTL_f) (Montali et al., 2010). Standard Declare only defines constraints like *response* ($\mathbf{G}(A \rightarrow \mathbf{F}B)$) or *alternateResponse* ($\mathbf{G}(A \rightarrow \mathbf{X}(\neg A \mathbf{U} B))$) that exclusively focus on behavioural aspects. Here, the **F**, **X**, **G**, and **U** LTL_f future operators have the following meanings: formula $\mathbf{F}\psi_1$ means that ψ_1 holds sometime in the future, $\mathbf{X}\psi_1$ means that ψ_1 holds in the next position, $\mathbf{G}\psi_1$ says that ψ_1 holds forever in the future, and, lastly, $\psi_1 \mathbf{U} \psi_2$ means that sometime in the future ψ_2 will hold and until that moment ψ_1 holds (with ψ_1 and ψ_2 LTL_f formulas). The **O**, **Y** and **S** LTL_f past operators have the following meaning: $\mathbf{O}\psi_1$ means that ψ_1 holds sometime in the past, $\mathbf{Y}\psi_1$ means that ψ_1 holds in the previous position, and $\psi_1 \mathbf{S} \psi_2$ means that ψ_1 has held sometime in the past and since that moment ψ_2 holds. Consider, for example, the *response* constraint $\mathbf{G}(A \rightarrow \mathbf{F}B)$. It indicates that if A *occurs*, B must eventually *follow*. Therefore, this constraint is fully satisfied in traces such as $\mathbf{t}_1 = \langle A,A,B,C \rangle$, $\mathbf{t}_2 = \langle B,B,C,D \rangle$, and $\mathbf{t}_3 = \langle A,B,C,B \rangle$, but not for $\mathbf{t}_4 = \langle A,B,A,C \rangle$ because, in this case, the second occurrence of A is not followed by a B. In $\mathbf{t}_2$, it is *vacuously satisfied* (Burattin et al., 2012), i.e., in a trivial way, because A never occurs. An *activation activity* of a constraint in a trace is an activity whose execution imposes, because of that constraint, some obligations on the execution of other activities (target activities) in the same trace. For example, A is an activation activity for the *response* constraint $\mathbf{G}(A \rightarrow \mathbf{F}B)$ and B is a target, because the execution of A forces B to be executed, eventually. An activation of a constraint leads to a *fulfilment* or to a *violation*. Consider, again, $\mathbf{G}(A \rightarrow \mathbf{F}B)$. In trace $\mathbf{t}_1$, the constraint is activated and fulfilled twice, whereas, in trace $\mathbf{t}_3$, it is activated and fulfilled only once. In trace $\mathbf{t}_4$, it is activated twice and the second activation leads to a violation (B does not occur subsequently).

A central shortcoming of languages like Declare is the fact that templates are not capable of expressing the connection between execution order of activities and other perspectives of the process. Consider the example of a loan application process. An analyst would be interested to learn about constraints such as the following:

1. Activation conditions: When a loan was requested and *account balance* $>$ *4,000 EUR*, the loan was subsequently granted in 95% of the cases.
2. Correlation conditions: When a loan was requested, the loan was subsequently granted and *amount requested* $=$ *amount granted* in 95% of the cases.
3. Target conditions: When a loan was requested, the loan was subsequently granted in 95% of the cases by a specific member of the financial board.
4. Temporal conditions: When a loan was requested, the loan was subsequently granted *within the next 30 days* in 95% of the cases.

The importance of more complex constraints that integrate activation, correlation, target and temporal dependencies has been emphasized by prior research and has led to the definition of a multi-perspective version of Declare (Burattin et al., 2015), multi-perspective Declare (MP-Declare) formally defined using LTL_f.

2.2 Mining Metrics

In this subsection, we describe the metrics that we use to discriminate those constraints that are fulfilled in the majority of cases in the event log, from those that are rarely satisfied, namely support and confidence. We consider two notions of support already defined in the literature, namely the event-based support (Di Ciccio and Mecella, 2015) and the trace-based support (Maggi et al., 2011).

Support. It is the number of fulfilments of a constraint divided by the number of occurrences of the condition of a constraint. In the example of Section 2.1, the four occurrences of A fulfil *response*(A,B), out of which 2 occur in $\mathbf{t}_1$, 1 in $\mathbf{t}_3$ and 1 in $\mathbf{t}_4$. Thereupon, it scales the number of events fulfilling the constraint by the number of events that fulfil the activation only. In the example, the five occurrences of A satisfy the activation. Therefore, the event-

based support of $response(A,B)$ is equal to $4/5$, namely 0.8.

Confidence. It is the product of support and the fraction of traces in the log where the condition (implications) or the constrained activity (not implications) occurs. In the example, the confidence is $4/5 \times 3/4 = 0.6$, A occurs in 3 traces over 4.

The work conducted in (Schönig et al., 2016) addressed the discovery of MP-Declare models. This research showed that the discovery of MP-Declare allows for the acquisition of knowledge that goes beyond classical declarative mining. However, it has not been investigated how this complex mining task can be performed in an efficient way. Until now, no scalable and high performant mining approach that can fully support MP-Declare is available.

3 MAP-REDUCE FOR MINING

In this section, we describe an efficient framework for discovering MP-Declare constraints. After giving insights into the internal infrastructure, we explain the parallelisable discovery algorithms for commonly used MP-Declare constraints that are used to discover models under consideration of further perspectives. In contrast to former solutions, this framework can be used out of the box and the algorithm has to consider the traces in one direction solely which leads to a crucial benefit with regard to performance.

3.1 Architecture and Infrastructure

The basic idea of the algorithm builds upon the MapReduce computation model. One key advantage is the inbuilt opportunity for executing the calculations in parallel, which gives an enormous performance boost, also in view of the analysis of an event log. At first, the scaffolding of the MapReduce algorithm is described briefly w.r.t. the discovery of a process model described later on. In the next section, we use an example log containing two traces defined in Equation 1.

$$t_0 = \langle a,b,b,c \rangle \qquad t_1 = \langle a,c,d \rangle \tag{1}$$

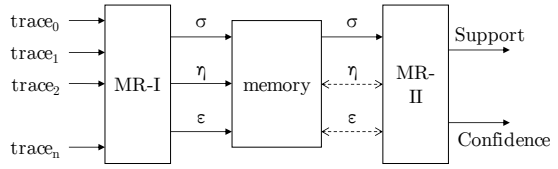

Figure 1: Infrastructure of the calculation.

To compute the support and confidence metrics, two MapReduce jobs are required, MR-I and MR-II (cf. Figure 1).

3.1.1 MR-I

In the **map**-phase of MR-I, key-value pairs are created from the given event data, i.e., a single trace of a log file. Each of the key-value pairs is assigned to a number for further processing. In the case of process discovery, this number is always 1. The challenge is to generate these key-value pairs and map them to 1 in order to address the logic for the MP-Declare constraints.

Example: Given a trace $t_0 = \langle a,b,b,c \rangle$. In the scope of the *response* template, the trace is mapped to four different key-value pairs in the map phase: $((a,b),1),((a,c),1),((b,c),1),((b,c),1)$. The keys are exactly those event pairs which fulfil the *response* template: a is followed by b and c, the first b is followed by c and the second b is followed by c. The underlying mapping algorithm containing the logic for all constraint templates is described in Section 3.3.

The **reduce**-phase finally obtains the key-value pairs that have been produced. The reduce-function must be declared by the user once again. In the case of constraint checking this phase depicts a summation of values. To continue the example above, the result of the reducer with trace t_0 is: $((a,b),1),((a,c),1),((b,c),2)$.

σ-function. The support metric is defined as the number of fulfilments of a constraint divided by the number of occurrences of the activation. The MR-I job in the example above calculates exactly the number of fulfilments, thus the numerator of the support formula. In the following we use a function $\sigma_\gamma : E \times E \to \mathbb{N}$, where E are events, for describing this figure, e.g. in t_0: $\sigma_{response}(b,c) = 2$. γ is the set of constraints.

η-function. To calculate the support of a constraint, the number of occurrences of the activation is necessary. For forward constraints ($\{*\}response$), this is the first event in the constraint template, e.g., b in the constraint $response(b,c)$. We define the number of occurrences of events as $\eta : E \to \mathbb{N}$, for instance in trace t_0: $\eta(b) = 2$. In order to obtain the correct values for the η-function, for each event e in the trace a key-value pair $(e,1)$ is additionally emitted in the map phase, e.g., for t_0: $(a,1),(b,1),(b,1),(c,1)$ which is reduced to $(a,1),(b,2),(c,1)$.

ε-function. A third value is necessary for determining the confidence, namely the amount of traces in which a given event occurs. We introduce the function $\varepsilon : E \rightarrow \mathbb{N}$, which holds this information. Taking into account the second trace t_1 (cf. Equation 1), MR-I outputs $\varepsilon(c) = 2$ or $\varepsilon(d) = 1$, as c occurs in t_0 and t_1, whereas d occurs in t_1 solely. Transferring this to MR-I, for each unique event e a key-value pair $(e,1)$ has to be produced, neglecting multiple occurrences of events, e.g., for trace t_0: $(a,1),(b,1),(c,1)$.

The Tables 1 and 2 shows the complete result of MR-I for the input log (cf. Equation 1) considering two constraint templates: *response* and *chainResponse*. The output of all mappers serves as the input for the reducers.

Table 1: Output of the Mapper in MR-I.

Trace	σ_R	σ_{CR}	η	ε
a,b,b,c	ab,1 bc,1	ab,1	a,1	a,1
	ac,1	bb,1	b,1	b,1
	bb,1	bc,1	b,1	c,1
	bc,1		c,1	
a,c,d	ac,1	ac,1	a,1	a,1
	ad,1	cd,1	c,1	c,1
	cd,1		d,1	d,1

Table 2: Output of the Reducer in MR-I.

σ_R	σ_{CR}	η	ε
ab,1 bc,2	ab,1 ac,1	a,2	a,2
ac,2 ad,1	bb,1 cd,1	b,2	b,1
bb,1 cd,1	bc,1	c,2	c,2
		d,1	d,1

3.1.2 MR-II

Two MapReduce jobs are performed where the event log only serves as input for the first MapReduce job. The output values of MR-I are used in MR-II to calculate support and confidence. Note that these calculations had to be extracted to a separate job because every single trace of the provided log needs to be tackled first in MR-I in order to obtain the σ-, η- and ε-functions. This makes MR-II mandatory; however, with a look on the performance, support and confidence can be computed in parallel again.

Using the functions introduced above, the support of a constraint $response(b,c)$ can be computed as $\mathcal{S}_R(b,c) = \frac{\sigma_R(b,c)}{\eta(b)} = \frac{2}{2} = 1$ (cf. Equation 2), thus as the fraction between the fulfilments of the constraint and the amount of its activations. Remember that forward constraints ($FWD = \{*\}response$) are activated with e_1 (Equation 2) and backward constraints ($BWD = \{*\}precedence$) are activated with e_2 (Equation 3).

$$\mathcal{S}_{FWD}(e_1,e_2) = \frac{\sigma_{FWD}(e_1,e_2)}{\eta(e_1)} \quad (2)$$

$$\mathcal{S}_{BWD}(e_1,e_2) = \frac{\sigma_{BWD}(e_1,e_2)}{\eta(e_2)} \quad (3)$$

The confidence of a constraint for an event pair (e_1,e_2) is the product of the support of (e_1,e_2) with the ratio between the amount of traces in the log in which event e_1 occurs (or e_2 in case of backward constraints) and the total number of traces in the log, denoted as $|l|$ in Equation 4 and 5.

$$\mathcal{C}_{FWD}(e_1,e_2) = \mathcal{S}_{FWD}(e_1,e_2) \cdot \frac{\varepsilon(e_1)}{|l|} \quad (4)$$

$$\mathcal{C}_{BWD}(e_1,e_2) = \mathcal{S}_{BWD}(e_1,e_2) \cdot \frac{\varepsilon(e_2)}{|l|} \quad (5)$$

In the running example, the confidence of the constraint $response(b,c)$ is calculated as $\mathcal{C}_R(b,c) = \mathcal{S}_R(b,c) \cdot (\frac{\varepsilon(b)}{|l|}) = 1 \cdot \frac{1}{2} = 0.5$.

In terms of MapReduce, the MR-II is structured rather trivial. In the **map**-phase, the output of MR-I is conducted directly to the reducer neglecting η and ε, i.e., all key-value pairs of the σ-function of all constraints are emitted and obtained by the reducer. The **reduce**-function then consults the *DB* (cf. Section 3.2) to look up the relevant η- and ε-value for a given key and calculates the corresponding support and confidence values (acc. to Equations 2, 3, 4 and 5).

3.2 Algorithm Overview

This section provides an overview on the algorithm for declarative process model discovery (Algorithm 1).

Algorithm 1: Overview.

```
for Trace t in l do
    KVP = mapToKVPairs(t)
    DB = accumulateKVPairs(DB, KVP)
end
for Constraint c do
    for Entry e in DB.σ.c do
        calcSupportAndConfidence(e, c)
    end
end
```

Given a log l with traces t, a first step is to create the key-value pairs *KVP* (MR-I-Mapper). These include the *KVP*s for each constraint (σ) and for the functions η and ε. These values are then accumulated into a database *DB* (MR-I-Reducer). The *DB* contains information about the three functions. For the

σ-function, it holds a list for each constraint separately and furthermore the entries of the lists comprising tuples $(EventTuple, fulfilments)$. This *DB* is then used in MR-II. For each constraint c, each tuple in the σ_c-list is considered (MR-II-Mapper). The discovered $EventTuple$ and the corresponding amount of fulfilments is then used to calculate support and confidence by consulting the corresponding η- and ε-values relevant to the $EventTuple$ (MR-II-Reducer).

3.3 Mapping MP-Declare Templates

We have to apply the logic of MP-Declare constraints into the MR-I mapping function to emit the necessary *KVP*s and calculate the correct values for support and confidence. For this purpose, we developed and derived algorithms from the support functions introduced in (Di Ciccio and Mecella, 2015). Therefore, we defined specific σ_γ functions for each of the MP-Declare relation constraints. Note that all the algorithms are working at only one trace instead of the whole log file, which ensures the capability of parallelization.

For reasons of readability, we use an abbreviated form for representing the event data in this section. We let the set of activities be $\{a,b,c,d\}$. Further on, we restrict to one single perspective, e.g., the organizational perspective, thus the defined resources which can execute the activities are $\{x,y,z\}$. For instance, trace t_1 in Equation 6 holds the information, that in the beginning a was executed by x, subsequently c was executed by z and so forth. In the end, the case is closed when again a was executed by x.

$$t_1 = \langle ax, cz, by, bx, dz, by, ax \rangle \tag{6}$$

The structure of the algorithm is built upon a nested for-loop, so that for each event in a given trace, every successor is considered. Henceforth i denotes the loop control variable for the outer loop and j is the counter variable for the inner loop.

In the case of t_1 (cf. Equation 6), all successors for ax are addressed in the inner loop ($i = 0$), whereas in the next step ($i = 1$) all successors for cz are considered and so forth. While iterating over the trace, different representations of the events are requested to match the multiperspective constraint templates. We denote the events for the outer loop as ${}_ie^\Gamma$ and for the inner loop as e^Γ_j, where Γ takes either A (activation) or T (target).
For instance, for $i = 2$ and $j = 5$ and in search of activation constraints (i.e. $A = (task, resource)$ and $T = (task)$) following representations are detected: ${}_1e^A = cz$, ${}_1e^T = c$, $e^A_4 = dz$ and $e^T_4 = d$.

In the following, we analyse the example trace t_1 (cf. Equation 6) and mark the pivot situations which are characteristic for each constraint template with $1_R, 2_R, \ldots$ for *response*, $1_{AR}, 2_{AR}, \ldots$ for *alternateResponse* and so forth.

3.3.1 Response

The initial assignment of (i, j) is $(0, 1)$, thus the events ax and cz are considered. For **activation** constraints, the activating event holds the additional condition solely; hence, $response(ax, c)$ is investigated in this first case. This constraint, activated with ${}_0e^A$ (ax) is fulfilled with e^T_1 (c) and thus $\sigma_R(ax, c)$ is incremented by 1. Also for $({}_0e^A, e^T_2)$ the value for $\sigma_R(ax, b)$ is incremented. In the next step, i.e., $({}_0e^A, e^T_3)$, the σ-value must not be modified, as the constraint $response(ax, b)$, activated with the event ${}_0e^A$ was already fulfilled with e^T_2 (cf. 1_R in Table 3). Similar are the cases 2_R to 5_R.

Table 3: MR-I results for *response* constraints (activation).

	c	b	b	d	b	a
ax	✓	✓	1_R	✓	2_R	✓
cz		✓	3_R	✓	4_R	✓
by			✓	✓	5_R	✓
bx				✓	✓	✓
dz					✓	✓
by						✓

For **target** constraints like $response(a, cz)$ the additional condition appears on the right-hand side. That means, the events in the outer loop must match the target template: ${}_ie^T$. Referring to Table 4, in the cases 6_R and 7_R, $\sigma_R(a, by)$ and $\sigma_R(c, by)$ respectively must not be increased, as the constraints are also already fulfilled (with $e^A_3 = by$).

Table 4: MR-I Results for *response* constraints (target).

	cz	by	bx	dz	by	ax
a	✓	✓	✓	✓	6_R	✓
c		✓	✓	✓	7_R	✓
b			✓	✓	✓	✓
b				✓	✓	✓
d					✓	✓
b						✓

3.3.2 AlternateResponse

The *alternateResponse* template shares the pivot constellations for (i, j) for already fulfilled constraints similar to the *response* template (cf. 1_{AR} to 5_{AR} in Table 5). For instance, the constraint $alternateResponse(ax, b)$ enforces that if the event a

occurs and is executed by x than the event b follows, and there is no recurrence of x executing a in between. In the case $i = 0$, this constraint is activated by ${}_0e^A$ (ax) and fulfilled with the event e_2^T (b). Therefore, additional events b must be ignored (e.g. e_3^T).
Besides the already-fulfilled-errors, another class of error type is introduced: *violations*. Consider 6_{AR} in Table 5. In this case, the constraint $alternateResponse(by,a)$ is checked. Although this constellation have not been occurred so far for this activation, the value $\sigma_{AR}(by,a)$ must not be modified, because it is violated by e_5^A (by): The activating event (by) recurs before a occurs. This is forbidden within the *alternateResponse* template. Note, that the resource is also decisive, thus $alternateResponse(by,d)$, activated with ${}_2e^A$ is fulfilled with e_4^T, although the event b recurs. However, this is executed by x and so the constraint is not violated (marked with an asterisk in Table 5.)

Table 5: MR-I res. for *alternateResponse* constraints (act.).

	c(z)	b(y)	b(x)	d(z)	b(y)	a(x)
ax	✓	✓	1_{AR}	✓	2_{AR}	✓
cz		✓	3_{AR}	✓	4_{AR}	✓
by			✓	✓*	5_{AR}	6_{AR}
bx				✓	✓	✓
dz					✓	✓
by						✓

The analysis of the **target** constraints (cf. Table. 6) shows the following anomalies: 7_{AR} and 8_{AR} are excluded because of the already-fulfilled-case and the cases 9_{AR} to 12_{AR} are excluded because of violations. For instance, 9_{AR} to 11_{AR} are activated with the event ${}_2e^A$ (b) and as the first event in the inner loop is also b (represented with the activation template, i.e. the activity solely (e_3^A)), all constraints with succeeding events in the inner loop are violated.

Table 6: MR-I res. for *alternateResponse* constraints (tar.).

	cz	bx	by	dz	by	ax
a	✓	✓	✓	✓	7_{AR}	✓
c		✓	✓	✓	8_{AR}	✓
b			✓	9_{AR}	10_{AR}	11_{AR}
b				✓	✓	12_{AR}
d					✓	✓
b						✓

3.3.3 ChainResponse

The logic for the *chainResponse* template is quite trivial and is located outside the inner loop. For each event ${}_ie^A$, the direct successor ${}_{i+1}e^T$ is considered and $\sigma_{CR}({}_ie^A,{}_{i+1}e^T)$ is incremented by one. Examples are an activation constraint like $chainResponse(ax,c)$ or a target constraint like $chainResponse(a,cz)$.

Table 7: MR-I res. for *chainResponse* constraints.

	cz	by	bx	dz	by	ax
ax	✓					
cz		✓				
by			✓			
bx				✓		
dz					✓	
by						✓

3.3.4 Precedence

Intuitively, one would iterate starting from the latest event for the backward constraints, e.g. the first (i,j)-tuple would be $(5,6)$ going on with $(4,6)$, i.e. the constraints $precedence(e_5^T,{}_6e^A)$ and $precedence(e_4^T,{}_6e^A)$ respectively. The former describes that whenever a occurs and was executed by x, then b has to precede (in the case of activation constraints $precedence(b,ax)$). Referring to the later, an example for a target constraint is if a occurs in a trace, then b has to precede and this has to be executed by y $(precedence(by,a))$.
For the sake of a performance boost, we propose an algorithm, which handle the backward constraints also by iterating through the events in a forward direction. To do so, the events of the outer loop (i) fills the role of the target events and the events of the inner loop (j) are now the activating events.
Consider Table 8. The first constraint under investigation will be $precedence(a,cz)$, activated with e_1^A (cz) and fulfilled with ${}_0e^T$ (a). After than, $precedence(a,by)$ is inspected. This one now is activated with e_2^A (by) but also fulfilled with the same outer loop event ${}_0e^T$ (a).
Interesting is the outer loop event ${}_2e^T$ (b) (cf. third row in Table 8). In the case e_4^A (dz), the value for $\sigma_P(b,dz)$ must not be modified (1_P). The reason is that this constraint, activated with dz is fulfilled with the outer loop event ${}_4e^T$ and thus, fulfilled in future (marked with an asterisk in Table 8).

Table 8: MR-I results for the *precedence* constraint template (activation).

	cz	by	bx	dz	by	ax
a(x)	✓	✓	✓	✓	✓	✓
c(z)		✓	✓	✓	✓	✓
b(y)			✓	1_P	2_P	3_P
b(x)				✓*	✓	4_P
d(z)					✓	✓
b(y)						✓

The **target** constraints show similar behaviour. Whenever the event ${}_ie^T$ occurs also in the inner loop in e_j^T, then the rest of the inner loop is neglected because the events are fulfilled later on, like $precedence(by,a)$ (5_P) is fulfilled in the asterisk-marked cell in Table 9.

Table 9: MR-I results for the *precedence* constraint template (target).

	c(z)	b(y)	b(x)	d(z)	b(y)	a(x)
ax	✓	✓	✓	✓	✓	✓
cz		✓	✓	✓	✓	✓
by			✓	✓	✓	5_P
bx				✓	✓	✓
dz					✓	✓
by						✓*

3.3.5 AlternatePrecedence

In addition, the *alternatePrecedence* constraints, as they are also backward constraints, are activated with the second given event in the template. The (activation) constraint $alternatePrecedence(a,by)$ at the marker 1_{AP} in Table 10 is violated because of the event e_2^A (by). Recurrences of the activating event are forbidden within the *alternatePrecedence* template. Similar is the case at 2_{AP}.
Consider now case 3_{AP} with the constraint $alternatePrecedence(b,dz)$. $\sigma_{AP}(b,dz)$ must not be incremented there, because this constraint activated with e_4^A (dz) is fulfilled with the event ${}_3e^T$ in the next run of the outer loop (note the asterisk in Table 10). Similar are the cases 4_{AP} to 6_{AP}. In

Table 10: MR-I results for the *alternatePrecedence* constraint template (activation).

	cz	by	bx	dz	by	ax
a	✓	✓	✓	✓	1_{AP}	✓
c		✓	✓	✓	2_{AP}	✓
b			✓	3_{AP}	4_{AP}	5_{AP}
b				✓*	✓	6_{AP}
d					✓	✓
b						✓

Table 11, the constraints at the cases 7_{AP} to 11_{AP} are violated, because of the reoccurrence of the events e_3^A (b) and e_5^A (b) in the events e_2^A and e_3^A.

3.3.6 ChainPrecedence

The *chainPrecedence* constraint template is implemented similar to the *chainResponse* template. However, the events of the constraints are integrated in reversed order starting with

Table 11: MR-I results for the *alternatePrecedence* constraint template (target).

	c	b	b	d	b	a
ax	✓	✓	7_{AP}	✓	8_{AP}	✓
cz		✓	9_{AP}	✓	10_{AP}	✓
by			✓	✓	11_{AP}	12_{AP}
bx				✓	✓	✓
dz					✓	✓
by						✓

$chainPrecedence(c,ax)$ (activation constraint) and in general terms $\sigma_{CR}({}_{i+1}e^T,{}_ie^A)$ is increased.

3.4 Pivot Characteristics Overview and Resulting Algorithm

The anomalies detected in the previous section can be traced to three certain pivot characteristics we have to take care. They include *already fulfilled (a)*, *violation (v)* and *fulfilled later (f)*, whereby the first one corresponds to forward constraints and latter appears only on backward constraints. In this section, the four anomaly classes are identified, described and the occurrence of problems regarding the classes are resolved.
Class I ($1_R - 7_R, 1_{AR} - 5_{AR}, 7_{AR} - 8_{AR}$). These situations occur when a pair of events is considered, where the activating event was already fulfilled in this case with a previous event. For instance, in a trace $\langle ax,b?,b?\rangle$, the constraint $R(ax,b)$ is fulfilled with the first event b and must not be considered in the next step ($j = 2$). For this activation constraint, the additional perspective of the fulfilling event is not crucial (note the ?). A similar case for a target constraint is $\langle a?,bx,bx\rangle$ where $R(a,bx)$ is fulfilled when reading the second bx in the inner loop. Also the *alternateResponse* template suffers from this anomaly: assuming a trace $\langle ax,\overline{ax},b?,\overline{ax},b?\rangle$, the value for $\sigma_{AR}(ax,b)$ referring to the constraint $AR(ax,b)$ would be incremented with the first b and the second b. Note that in this class it is forbidden for ax to recur as this would cause a violation (cf. Class II).
Solution. The problem is that the events in the inner loop filtered by the target template e_j^T are recurring. To prevent these Class I-failures, all events e_j^T are stored in a list L and σ is only incremented if the current e_j^T is not in L.

Class II ($6_{AR}, 9_{AR} - 12_{AR}$). Class II-errors hits the *alternateResponse* template solely. The definition of this template forbids the activating event to recur before the second event appears. As an example serves the trace $\langle ax,ax,b?\rangle$ with the constraint $AR(ax,b)$ for an activation constraint and $\langle a?,a?,b\rangle$ with $AR(a,bx)$ for a target constraint respectively.

Algorithm 2: Discovery of relational MP-Declare constraints.

```
   Input: Trace t
   Output: DB
 1 for i ← 0 to trace.events.Count do
 2 |  List<Event> eR, eAR, eAP;
 3 |  bool bAR, bP, bAP = false;
 4 |  db.η(ᵢe^A);
 5 |  db.η(ᵢe^T);
 6 |  db.ε(ᵢe^A);
 7 |  db.ε(ᵢe^T);
 8 |  for j ← i+1 to trace.events.Count do
   |  |  ; /* Response                    */
 9 |  |  if !eR.Contains(e^T_j) then
10 |  |  |  db.σ.R(ᵢe^A, e^T_j);
11 |  |  |  eR.Add(e^T_j);
12 |  |  end
   |  |  ; /* AlternateResponse           */
13 |  |  if !bAR ∧ !eAR.Contains(e^T_j) then
14 |  |  |  db.σ.AR(ᵢe^A, e^T_j);
15 |  |  |  eAR.Add(e^T_j);
16 |  |  |  if e^A_j = ᵢe^A then
17 |  |  |  |  bAR ← true;
18 |  |  |  end
19 |  |  end
   |  |  ; /* Precedence                  */
20 |  |  if !bP then
21 |  |  |  db.σ.P(ᵢe^T, e^A_j);
22 |  |  end
23 |  |  if ᵢe^T = e^T_j then
24 |  |  |  bP ← true;
25 |  |  end
   |  |  ; /* AlternatePrecedence        */
26 |  |  if !bAP ∧ !eAP.Contains(e^A_j) then
27 |  |  |  db.σ.AP(ᵢe^T, e^A_j);
28 |  |  |  eAP.Add(e^A_j);
29 |  |  end
30 |  |  if ᵢe^T = e^T_j then
31 |  |  |  bAP ← true;
32 |  |  end
33 |  end
34 |  if i < |t| − 1 then
35 |  |  db.σ.CR(ᵢe^A, ᵢ₊₁e^T);
36 |  |  db.σ.CP(ᵢ₊₁e^A, i^T_e);
37 |  end
38 end
```

Solution. If the activating event ${}_ie^A$ recurs in the inner loop as event e^A_j, then all succeeding constraints in the inner loop are violated by this recursion and thus the inner loop can be cancelled for this template.

Class III ($1_P - 5_P, 3_{AP} - 6_{AP}, 12_{AP}$). These anomaly is similar to Class I but for backward constraint templates. Some constraints must not be considered because they will be fulfilled later on. For instance, in a trace $\langle b?, b?, ax\rangle$ in the first outer loop run it is checked if the first $b?$ fulfils a constraint $P(b, ax)$. However, this is not true because this certain constraint is fulfilled in the second outer loop run.

Solution. The problem here is that the event of the outer loop ${}_ie^T$ recurs in the inner loop event e^T_j. That means that the succeeding inner loop events are fulfilled later on with succeeding outer loop events. In case of a recurrence, the consideration of succeeding events in this inner loop run can be cancelled.

Class IV ($1_{AP} - 2_{AP}, 7_{AP} - 11_{AP}$). Similar to Class II, errors corresponding to Class IV handle violations of constraints, viz. from the *alternatePrecedence* template in this particular case. In a trace $\langle a?, bx, bx\rangle$ the activation constraint $alternatePrecedence(a, bx)$, activated with the second bx event is violated, as bx recurs, before the fulfilling event a preceds.

Solution. As a solution, we store all events e^A_j in a list. If a next event e^A_j with a greater j occurs, the consideration of *alternatePrecedence* templates can be cancelled for a certain i.

4 IMPLEMENTATION

In this section, we particularly describe the implementation of MapReduce MP-Declare mining. As the algorithm is built upon the MapReduce computation paradigm, we took care to exploit this chance of parallelism by spreading the workload on multiple threads of a multicore CPU. As a first step, the XES log file has to be loaded into the applications memory. This is done by reading the file with the .NET LIBARY and parsing the information into POCOs. This step is also used to collect meta information about the event log to provide the user for instance a list of available perspectives but also some information on the expected performance of the analysis by means of the figures described in Section 4.2. As soon as the user has chosen the perspectives to consider, i.e. he has built the mining template (e.g. $task \wedge resource \rightarrow task$), a first parallel processing step builds the three characteristic functions σ, η and ε in MR-I. With the parallel extensions of the *Language Integrated Query*-.NET component (`PLINQ: Parallel.ForEach();`) we delegate the whole parallelization and task generation on to the framework itself and abstract from low level programming issues with all the upcoming advantages. We just had to take care of race hazards when joining to the global database (cf. `lock`-keyword). After then, when each of the traces has been considered and the characteristic functions are completed, the MR-II function can be invoked. Again, this step is executed in parallel with `PLINQ`.

4.1 Performance Influence Factors

Distribution of the Events. The most relevant index number for a performance analysis is the amount of loop runs the algorithm has to complete as they rise in a quadratic manner with the amount of events in a trace. For a trace with n events, the amount of loop runs $\mathcal{L}_t$ follows the formula in Equation 7.

$$\mathcal{L}_t = (n-1) + ... + 1 = \frac{n \cdot (n-1)}{2} = \frac{n^2 - n}{2} \quad (7)$$

That means, the higher the amount of traces with a huge amount of events in a log file, the higher is the computation time. For instance, first assume a log file containing 100 traces with a total amount of 1500 events, evenly distributed with 15 events per trace. Then the number of loop runs comes up to $\frac{15^2-15}{2} = 105$ for each trace, or summed up for the whole log $105 \cdot 100 = 10500$. Now consult a log file also with 100 traces, but now with 10 traces containing 90% of the events (945 events per trace) and the remaining 90 traces with the remaining 12 events per trace[1]. Then the total amount of loop runs is calculated as $10 \cdot \frac{945^2-945}{2} + 90 \cdot \frac{12^2-12}{2} = 4460400 + 5940 = 4466340$, which is more than 400 times higher than in the evenly distributed log. Hence, the crucial factor, i.e. the total number loop runs, can then be defined as the sum of the loop runs of each single trace (cf. Equation 8).

$$\sum_{Trace} \frac{|t|^2 \cdot (|t|-1)}{2} \quad (8)$$

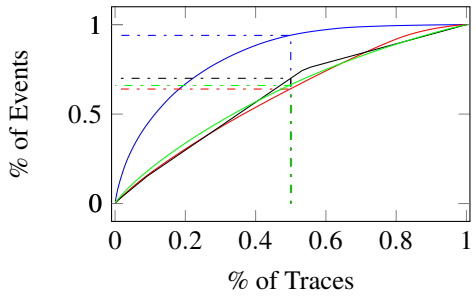

Figure 2: This caption has one line so it is centered.

Consider Table 12. Four real-life event logs were analysed while the traces where rearranged in descending order by the amount of events. It can clearly be seen that in the Hospital Log (blue line), half of the traces contains nearly all captured events (94%). Half of the events are included in the first 130 traces (11.4% of all traces). This raises the necessary loop runs to about $33 \cdot 10^6$. In contrast, in the Municipality Log (red line), the first half of the traces contains only about 64% of the events and 50% of the events are recorded in the first 438 traces (36% of all traces), similar to the Loan Application Log and the Traffic Log.

Since now, we have not seen a chance in breaking the quadratic dependency but as stated, compared to (Di Ciccio and Mecella, 2015) or (Sturm et al., 2017) we introduced an algorithm which is not in need of a two-directional investigation of the traces and thus saves half of the necessary loop runs.

Amount of Traces and Events. Apart from the dependency of the distribution of the events within the log discussed above, our implementation could handle log files with up to 150000 Traces or 1000000 captured events. Table 12 shows that the absolute amount of events and traces does not have a strong impact on the performance. For instance, the Traffic Log inherits nearly 600000 events and 150000 traces but is analysed within a mere fraction of time compared to logs either with less traces or less events.

Amount of Discovered Constraints. In order to investigate a performance restriction caused by a huge amount discovered constraints, we set the threshold of support and confidence on a minimal level, for instance 0.05 and 0.02 instead of 0.5 and 0.2. Shown with the Hospital Log, there is no noticeable change in performance when discovering about 35000 constraints instead of 2000. Therefore, this is not worth to be further analysed.

4.2 Performance Evaluation

We evaluated the effectiveness of our approach w.r.t. to several real-life event logs. We first evaluated our approach for the discovery of MP-Declare constraints using the **Hospital Log**[2], which records the treatment of patients diagnosed with cancer from a large Dutch hospital. In 5 of the traces at least one event does not include the additional perspective data (from *org:resource*) and thus they had to be excluded from the investigation. Furthermore, we applied our approach to the publicly available real-life event log (**Traffic Log**[3]) of an Italian local police office for managing fines for road traffic violations. This log file barely contains additional perspective data in a consistent way, so the *lifecycle:transition* attribute was considered as additional perspective, but the semantics does not affect the performance anyway. Additi-

[1] round up for simplicity

[2] https://doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54

[3] https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5

Table 12: Real-Life Event Logs under investigation.

Log Name	Events	Traces	Events/Traces	Loop Runs	Runtime in s
Traffic Log	561470	150370	3	0,97	133
Municipality Log	52217	1199	43	1,28	153
Loan Application Log	1202267	31509	38	26,73	697
Hospital Log	149489	1138	131	33,17	761

onally, we applied our approach to an event log pertaining to an administrative process in a Dutch municipality (**Municipality Log**[4]) as well as a log file containing **Loan Application**[5] information. All four event logs have been analysed w.r.t. all six described MP-Declare templates. The time measurements in Table 12 considers activation conditions solely, but the same tendencies could be recorded for target conditions.

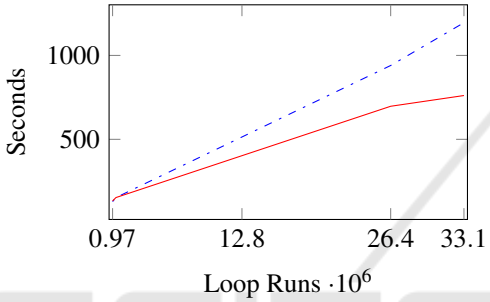

Figure 3: Performance of sequential execution compared to parallel execution.

The benchmark shows the expected behaviour with respect to the proposed performance influencing figures. Mapped into the coordinate system shown in Figure 3 with the total loop runs on the abscissa and the elapsed time on the ordinate one can clearly see the dependency. Furthermore, the impact of parallelization is revealed. When working on the Traffic Log (0.97), there is almost no difference between the sequential (dashed line) and the parallel execution (solid line). The performance gain through the distributed computing is cancelled out by the additional workload for task generation and joining the MR-I mapping results. Nevertheless, with more challenging log files in terms of loop runs (e.g. the Hospital Log (33.1)), the parallelization is exploited more and more efficiently.

5 RELATED WORK

Several approaches have been proposed for the discovery of declarative process models. In (Maggi et al., 2011), the authors present an approach that allows the user to select from a set of predefined Declare templates the ones to be used for the discovery. Other approaches to improve the performances of the discovery task are presented in (Di Ciccio et al., 2015b; Westergaard et al., 2013). Additionally, there are post-processing approaches that aim at simplifying the resulting Declare models in terms of redundancy elimination (Maggi et al., 2013a; Di Ciccio et al., 2015a) and disambiguation (Bose et al., 2013). An approach similar to the SQL-based one used in this paper is presented in (Räim et al., 2014) and is based on temporal logic query checking. In (Westergaard and Maggi, 2012), the authors define *Timed Declare*, an extension of Declare that relies on timed automata. In (Maggi, 2014), an approach for analysing event logs with Timed Declare is proposed. The *DPILMiner* (Schönig et al., 2016) exploits a discovery approach to incorporate the resource perspective and to mine for a set of predefined resource assignment constraints. In (Montali et al., 2013), the authors introduce for the first time a data-aware semantics for Declare and (Maggi et al., 2013b) first covered the data perspective in declarative process discovery, although this approach only allows for the discovery of *discriminative* activation conditions. (Schönig et al., 2016) proposes an approach to enable the discovery of MP-Declare constraints by querying event logs given in relational databases with SQL. Here, a performance evaluation has not been described.

6 CONCLUSIONS

In this paper, we proposed an efficient framework for the discovery of MP-Declare models based on the distributed processing method MapReduce. We introduced parallelizable algorithms for discovering six commonly used types of MP-Declare constraints. The mining performance and effectiveness have been tested with real-life event logs. The experiments show that our technique solves this complex mining task in reasonable time. The approach at hand represents a step into the direction of integrating process and data science and depicts a customisable and high performant declarative process mining technique. For future work, we plan to consider also correlation and

[4]https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1

[5]https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

time conditions as well as an additional integration of all MP-Declare constraints. Furthermore, we will examine how to improve performance even more, for instance with alternative MapReduce frameworks which can be set up and tested with the proposed algorithms.

REFERENCES

Bose, J. C., Maggi, F. M., and van der Aalst, W. (2013). Enhancing Declare Maps Based on Event Correlations. In *Business Process Management*, pages 97–112.

Burattin, A., Maggi, F. M., and Sperduti, A. (2015). Conformance checking based on multi-perspective declarative process models. *CoRR*, abs/1503.04957.

Burattin, A., Maggi, F. M., van der Aalst, W. M., and Sperduti, A. (2012). Techniques for a Posteriori Analysis of Declarative Processes. In *EDOC*, pages 41–50, Beijing. IEEE.

de Leoni, M., van der Aalst, W. M. P., and Dees, M. (2016). A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Inf. Syst.*, 56:235–257.

Di Ciccio, C., Maggi, F. M., Montali, M., and Mendling, J. (2015a). Ensuring model consistency in declarative process discovery. In *BPM*, volume 9253 of *Lecture Notes in Computer Science*, pages 144–159. Springer.

Di Ciccio, C. and Mecella, M. (2013). A two-step fast algorithm for the automated discovery of declarative workflows. In *CIDM*, pages 135–142. IEEE.

Di Ciccio, C. and Mecella, M. (2015). On the discovery of declarative control flows for artful processes. *ACM TMIS*, 5(4):1–37.

Di Ciccio, C., Schouten, M. H. M., de Leoni, M., and Mendling, J. (2015b). Declarative process discovery with MINERful in ProM. In *BPM Demos*, pages 60–64.

Maggi, F., Bose, R., and van der Aalst, W. (2013a). A knowledge-based integrated approach for discovering and repairing declare maps. In *CAiSE*.

Maggi, F. M. (2013). Declarative process mining with the declare component of prom. In *BPM Demo sessions 2013, 26-30, 2013*.

Maggi, F. M. (2014). Discovering metric temporal business constraints from event logs. In *BIR*, volume 194 of *Lecture Notes in Business Information Processing*, pages 261–275. Springer.

Maggi, F. M., Dumas, M., García-Bañuelos, L., and Montali, M. (2013b). Discovering data-aware declarative process models from event logs. In *BPM*, pages 81–96.

Maggi, F. M., Mooij, A., and van der Aalst, W. (2011). User-Guided Discovery of Declarative Process Models. In *CIDM*, pages 192–199.

Montali, M., Chesani, F., Mello, P., and Maggi, F. M. (2013). Towards data-aware constraints in declare. In *SAC*, pages 1391–1396. ACM.

Montali, M., Pesic, M., van der Aalst, W. M. P., Chesani, F., Mello, P., and Storari, S. (2010). Declarative Specification and Verification of Service Choreographies. *ACM Transactions on the Web*, 4(1).

Pesic, M., Schonenberg, H., and van der Aalst, W. M. P. (2007). Declare: Full support for loosely-structured processes. In *IEEE International EDOC Conference 2007*, pages 287–300.

Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., and Reijers, H. A. (2011). Imperative versus declarative process modeling languages: An empirical investigation. In *BPM Workshops*, pages 383–394.

Räim, M., Di Ciccio, C., Maggi, F. M., Mecella, M., and Mendling, J. (2014). Log-based understanding of business processes through temporal logic query checking. In *OTM Conferences*, volume 8841, pages 75–92. Springer.

Schönig, S., Cabanillas, C., Jablonski, S., and Mendling, J. (2016). A Framework for Efficiently Mining the Organisational Perspective of Business Processes. *Decision Support Systems*.

Schönig, S., Di Ciccio, C., Maggi, F. M., and Mendling, J. (2016). Discovery of multi-perspective declarative process models. In *Service-Oriented Computing, ICSOC, Banff, Canada*, pages 87–103.

Sturm, C., Schönig, S. S., and Ciccio, C. D. (2017). Distributed multi-perspective declare discovery. In *BPM Demos*.

van der Aalst, W. (2011). *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer.

Westergaard, M. and Maggi, F. M. (2012). Looking into the future: Using timed automata to provide a priori advice about timed declarative process models. In *OTM*, volume 7565 of *LNCS*, pages 250–267. Springer.

Westergaard, M., Stahl, C., and Reijers, H. (2013). UnconstrainedMiner: Efficient Discovery of Generalized Declarative Process Models. *BPM CR, No. BPM-13-28*.

Zeising, M., Schönig, S., and Jablonski, S. (2014). Towards a Common Platform for the Support of Routine and Agile Business Processes. In *Collaborative Computing: Networking, Applications and Worksharing*.