A

On the Discovery of Declarative Control Flows for Artful Processes

Claudio DI CICCIO, Wirtschaftsuniversität Wien
Massimo MECELLA, Sapienza Università di Roma

Artful processes are those processes in which the experience, intuition, and knowledge of the actors are the key factors in determining the decision making. They are typically carried out by the "knowledge workers", such as professors, managers, researchers. They are often scarcely formalized or completely unknown a priori. Throughout this paper, we discuss how we addressed the challenge of discovering declarative control flows, in the context of artful processes. To this extent, we devised and implemented a two-phase algorithm, named MINERful. The first phase builds a knowledge base, where statistical information extracted from logs is represented. During the second phase, queries are evaluated on that knowledge base, in order to infer the constraints that constitute the discovered process. After an outline of the overall approach and an insight on the adopted process modeling language, we describe in detail our discovery technique. Thereupon, we analyze its performances, both from a theoretical and an experimental perspective. A user-driven evaluation of the quality of results is also reported, on the basis of a real case study. Finally, a study on the fitness of discovered models with respect to synthetic and real logs is presented.

Categories and Subject Descriptors: H.2.8 [**Database Applications**]: Data Mining

General Terms: Algorithms, Management

Additional Key Words and Phrases: Process Mining, Declarative Process Model, Control-Flow Discovery, Artful Processes, MailOfMine

ACM Reference Format:
Claudio Di Ciccio and Massimo Mecella, 2014. On the Discovery of Declarative Control Flows for Artful Processes. *ACM Trans. Manag. Inform. Syst.* V, N, Article A (January YYYY), 38 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Knowledge workers [Warren et al. 2009] are those professionals working for the creation of intangible products, whose value resides in the knowledge they provide, let be it in favor of their company, university, or the whole society. The processes that they carry out fall under the category of knowledge-intensive processes (KIPs) [Gronau and Weber 2004], as their value is meant to be created through the fulfillment of the knowledge requirements of the process participants. Some characteristics of KIPs are

This work has been partly supported by the Sapienza grants TESTMED, SUPER, Premio "Ricercatori Under-40" (2012), and the InfoSapienza center. The work of Massimo Mecella has been also partly supported by the Italian MIUR project RoMA (Smart Cities and Communities SCN_00064) . The work of Claudio Di Ciccio has been partly performed when a PhD candidate in Sapienza Università di Roma. With Wirtschaftsuniversität Wien, his research has received funding from the EU Seventh Framework Programme (FP7) under grant agreement 318275 (GET Service).
Author's addresses: C. Di Ciccio, Wirtschaftsuniversität Wien, Department of Information Systems and Operations, Building D2, Welthandelsplatz 1, 1020 Vienna, Austria. E-mail: claudio.di.ciccio@wu.ac.at. M. Mecella, Sapienza Università di Roma, Dipartimento di Ingegneria Informatica Automatica e Gestionale "Antonio Ruberti", via Ariosto 25, 00185 Roma, Italy. E-mail: mecella@diag.uniroma1.it

the diversity and uncertainty of process input and output [Davenport et al. 1996] and the variability of exceptional conditions [Di Ciccio et al. 2014]. Artful processes [Hill et al. 2006] are a class of knowledge-intensive processes, where the decisions taken over during the enactment of the process are usually fast and based on the expertise and intuition of the main actors [Di Ciccio et al. 2012]. Therefore, there is an “art” in their execution: their name, “artful”, stems from this. A typical example of artful process is the management of a research project: knowledge workers such as project managers, professors, technical managers, contribute to the final outcome. To this extent, they bring into play their competence, together with the best practices gathered during their respective careers. Due to their nature, artful processes are rarely formalized – let alone defined formally [Hill et al. 2006]. Therefore, though frequently repeated, they are not exactly reproducible, even by their originators – since they are not written down – and cannot be easily shared either. Furthermore, hardly any process management systems are currently used during the execution of such workflows.

Our objective is the automated discovery of artful processes. Understanding artful processes involving knowledge workers can lead to valuable improvements in many scenarios. For instance, it can be crucial in *enterprise engineering*, where it is important to preserve more than just the actual documents making up the product data: knowing the “soft knowledge” of the overall process (the so-called product life-cycle) is of critical relevance for knowledge-heavy industries.

[Di Ciccio and Mecella 2013a] describes MAILOFMINE, namely the approach we devised and the system we designed for discovering such processes out of semi-structured texts, i.e., email messages. In this paper, we specifically focus on the control-flow discovery algorithm that MAILOFMINE is based on, named MINERful.

Due to their nature, artful processes are highly flexible. Therefore, their representation by means of graphs depicting all the possible paths leading to the final outcome is likely to result in vast entangled models: the more alternatives are allowed, the more graphical objects have to be drawn. Intricated models lead to less comprehensibility [Mendling et al. 2007b] and more errors [Mendling et al. 2007a]. Therefore, we opted for the “declarative” workflow modeling [Pesic and van der Aalst 2006], which specifies workflows through a listing of constraints: every execution is assumed to be valid, as long as it respects such constraints. As stated in [van der Aalst et al. 2009a], this leads to an effective way of representing flexible workflows, though ensuring a balanced level of support for the process management. Hence, the output of MINERful is a declarative model of the control flow.

The input for the discovery algorithm is an *event log*, i.e., a textual representation of a temporarily ordered linear sequence of tasks. Each recorded *event* reports the execution of a *task* (i.e., a well-defined step in the workflow) in a *case* (i.e., a workflow instance).

A very concise description of a preliminary version of the technique was presented in [Di Ciccio and Mecella 2012b], without the definition of important reliability and relevance metrics for the discovered constraints. An enhanced version, with performance evaluation over synthetic data, has been introduced in [Di Ciccio and Mecella 2013c]. The current paper gives an holistic presentation of the technique in a more refined version. It extensively analyzes its performances, both from a theoretical and an experimental perspective, and reports a user-driven evaluation of the quality of the results, conducted on a real case study.

The characteristics of the approach are: *(i)* modularity, as it is based on two steps, where the first step builds a knowledge base for the second, effectively verifying the constraints as the results of specific queries;*(ii)* independence from the specific formalism adopted for representing constraints; *(iii)* probabilistic approach to the inference of constraints; *(iv)* capability of eliminating the redundancy of subsumed constraints.

MINERful has been designed in order to be reasonably fast. The main reason for keeping its computation time low mainly resides in its usage inside MAILOFMINE. Since it runs over logs which are built on top of uncertain information (the semi-structured text of email messages), the output strongly depends on the reliability of the log. The log can indeed contain several outliers or misinterpreted information. Thus, if users or experts considered the resulting process model as poorly compliant to the reality, the mining phase might need to be repeated over refined logs, so as to improve the overall quality of the result. A slow algorithm might undermine such an iterative approach, making it impractical.

The remainder of the paper is organized as follows. Section 2 discusses relevant work. The declarative process model, which is adopted by our technique, is presented in Section 3. MINERful is described extensively in Section 4. Section 5 presents the evaluation of the technique, and finally Section 6 concludes the paper.

2. RELATED WORK

Process Mining, a.k.a. *Workflow Mining* [van der Aalst 2011], is the set of techniques that allow the extraction of process descriptions, stemming from a set of recorded real executions. Such executions are meant to be stored in so called *event log*s, i.e., textual representations of a temporarily ordered linear sequence of tasks [van der Aalst 2012]. In event logs, each recorded *event* reports the execution of a *task* (i.e., a well-defined step in the workflow) in a *case* (i.e., a workflow instance). Events are always recorded sequentially, even though tasks could be executed in parallel: the algorithm has to infer the actual structure of the workflow, by identifying the causal dependencies between tasks (*condition*s). ProM [van der Aalst et al. 2009b] is one of the most used plug-in based software environment for implementing workflow mining techniques.

The idea to apply process mining in the context of workflow management systems was introduced in [Agrawal et al. 1998], where processes were modeled as directed graphs in which vertexes represented individual activities and edges stood for dependencies between them. Cook and Wolf, at the same time, investigated similar issues in the context of software engineering processes. In [Cook and Wolf 1998] they described three methods for process discovery: *(i)* neural network-based, *(ii)* purely algorithmic, *(iii)* adopting a Markovian approach. The authors consider the last two as the most promising. The purely algorithmic approach builds a finite state machine where states are fused if their *futures* (in terms of possible behaviors for the next *k* steps) are identical. The Markovian approach uses a mixture of algorithmic and statistical methods and is able to deal with noise. However, the results presented in [Cook and Wolf 1998] are limited to sequential behavior only.

From [Agrawal et al. 1998] onwards, many techniques have been proposed, in order to address specific issues: pure algorithmic (e.g., α algorithm, drawn in [van der Aalst et al. 2004] and its evolution α^{++} [Wen et al. 2007]), heuristic (e.g., [Weijters and van der Aalst 2003]), genetic (e.g., [de Medeiros et al. 2007]), etc. Heuristic and genetic algorithms have been introduced to cope with noise, which the pure algorithmic techniques were not able to manage. Algorithmic techniques rely on footprints of traces (i.e., tables reporting whether events appeared before or afterwards) to determine the workflow net that could have generated them. Heuristic approaches build a representation similar to causal nets. When they construct the process model, they take into account the frequencies of events and sequences, in order to ignore infrequent paths. Genetic process mining adopts an evolutionary approach to the discovery. It differs from the other two in that its computation evolves in a non-deterministic way: the final output, indeed, is the result of a simulation of the process of natural selection. The evolutionary reproduction of the procedures is used to determine the final outcome. [Buijs et al. 2012] discusses in depth the user-tunable metrics adopted for the genetic algorithm, in order to make it return qualitatively better workflows in terms of replay fitness, precision, generalization and simplicity [van der Aalst 2011]. The accurate results are valuable, though such an algorithm suffers from unpredictability in terms of the returned process: it can change from run to run, due to the nature of evolutionary algorithms itself. Another drawback is the time the algorithm takes, which is generally high.

A very smart extension to the previous research work has been achieved by the two-steps algorithm proposed in [van der Aalst et al. 2010]. Differently from the former approaches, which typically provide a single process mining step, it splits the computation in two phases: *(i)* the tunable mining of a Transition System (TS) representing the process behavior and *(ii)* the automated construction of a Petri Net bisimilar to the TS [Cortadella et al. 1998; Desel and Reisig 1996]. The first phase is "tunable", so that it can be either more strictly adhering (or more permissive) w.r.t. the analyzed log traces behavior. As a consequence, the user can balance between overfitting and underfitting. The second phase has no parameter to set, since its only aim is to synthesize the TS into an equivalent Workflow Net. Thus, it is fixed, while the former step could be realized exploiting one among many of the previously proposed "one-step" algorithms. [Weijters and van der Aalst 2003], e.g., is claimed to integrate well.

The need for flexibility in the definition of some types of process, such as artful processes, leads to an alternative to the classical "imperative" approach: the "declarative" approach. Rather than using a procedural language for expressing the allowed sequences of activities ("closed" models), it is based on the description of workflows through the usage of constraints: the idea is that every task can be performed, except what does not respect such constraints ("open" models). [van der Aalst et al. 2009a] showed how the declarative approach (such as the one adopted by Declare [Pesic et al. 2007]) can help in obtaining a fair trade-off between flexibility in managing collaborative processes and support in controlling and assisting the enactment of workflows.

[Maggi et al. 2011] outlines an algorithm for mining Declare processes implemented in ProM (Declare Miner). The approach works as follows. The user is asked to specify a set of Declare constraint templates. Then, the system generates all the possible constraints stemming from them, i.e., obtained by the instantiation of those templates to all the activities in the process. The user is required to set an additional parameter named PoE (Percentage of Events). It is meant to be used as a threshold, in order to avoid that "rare" events determine the final outcome. Constraints are indeed pruned out, if the constrained activities appear less than PoE% times in the log. Thereafter, every candidate constraint is translated into the related accepting finite automata, according to the rules defined in [Pesic et al. 2010]. For the optimization of this task, the tool is integrated with the technique described in [Westergaard 2011]. Traces are thus replayed on the resulting automata. Each constraint among the candidates becomes part of the discovered process if and only if the percentage of traces accepted by the related automaton exceeds a user-defined threshold, named PoI (Percentage of Instances).

[Maggi et al. 2012] proposes an evolution of [Maggi et al. 2011], with the adoption of a two-phase approach. The first phase is based on the Apriori algorithm, developed by Agrawal and Srikant for mining association rules [Agrawal and Srikant 1994]. During this preliminary phase, the frequent sets of correlated activities are identified. The candidate constraints are computed on the basis of the correlated activity sets only. During the second phase, the candidate constraints are checked as in [Maggi et al. 2011]. Therefore, the search space for the second phase is reduced. In output, constraints constituting the discovered process are weighted according to their *Support*, i.e., the probability of such constraints to hold in the mined workflow. It is calculated as the proportion of traces where the constraint is satisfied. To filter out irrelevant constraints, more metrics are introduced, based on the appearances of the activities involved within the log: they are *Confidence*, *Interest Factor* and *CPIR* (Conditional-Probability Increment Ratio). Since we also adopted such metrics, with slight modifications, they will be described further in the following sections. The recent work of Westergaard and Stahl [**?**] presents their "Unconstrained Miner", a fast tool for mining declarative constraints. The boost in performance is obtained thanks to the parallelization of the computing tasks (having each thread working on different slices of the log), and the simultaneous verification of multiple constraints (through the usage of compound automata, built to this aim). As such, it serves as a rapid tool for computing the support of constraints.

[Pichler et al. 2011] presents a first empirical study on the cognitive effort required for understanding declarative models, in comparison with the imperative ones. Since the imperative is a well-established approach, known to the vast majority of the Business Process Management experts and practitioners, the declarative approach turns out to be less intuitive to them. Therefore, comprehensibility of models is a key factor for establishing the new approach. The seminal work on declarative modeling itself [van der Aalst et al. 2009a] states that declarative workflow specifications may be less readable if many (interacting) constraints are added. Thus, the minimization of constraints returned by declarative process mining techniques has become a main challenge. [Maggi et al. 2011] first tackles it by removing vacuously satisfied constraints. Constraints are considered as vacuously satisfied when no trace in the log violates them, yet no trace shows the effect of their application either. A vacuously satisfied constraint is, e.g., that every request is eventually acknowledged, in a process instance that does not contain requests. Vacuity detection techniques were originally proposed by Vardi et al. [Kupferman and Vardi 2003], in the field of LTL model checking. Maggi et al. successfully adapted them in the context of process discovery. In addition, [Maggi et al. 2013] deals with the issue of simplifying the so called Declare "maps", keeping only the most significant constraints in the discovered workflows. The idea is indeed that the discovered Declare maps can often result in cluttered diagrams, due to the presence of constraints which are proven to hold true in logs, but redundant or uninteresting. The proposed technique, implemented as a ProM plug-in and named "Declare Maps Miner', works as follows: after the application of the technique described in [Maggi et al. 2012], the pruning phase takes place. It is based on three main methods: *(i)* the removal of weaker constraints implied by stronger constraints; *(ii)* the reparation of predefined basic Declare maps; *(iii)* an ontology-guided search for constraints, linking activities that either belong to different groups of interest, or to the same group. The last two require the user input, whereas the first does not. Due to the nature of artful processes, we could rely neither on predefined models nor on activities' grouping. For the removal of implied constraints, we made use of a methodology presented in [Di Ciccio and Mecella 2013c], based on the subsumption hierarchy of constraints.

[Lamma et al. 2007a; Chesani et al. 2009] describe the usage of inductive logic programming techniques to mine models expressed as a SCIFF [Alberti et al. 2008] first-order logic theory, consisting of a set of implication rules named Social Integrity Constraints (IC's for short). Finally, the learned theory is automatedly translated into the ConDec [Pesic and van der Aalst 2006] notation. [Chesani et al. 2009] proposes the implementation of the framework, named DPML (Declarative Process Model Learner [Lamma et al. 2007b]) as a ProM plug-in. [Bellodi et al. 2010b; 2010a] extend this technique by weighting in a second phase the constraints with a probabilistic estimation. The learned IC's are indeed translated from SCIFF, discovered by DPML, into Markov Logic formulae [Richardson and Domingos 2006]. Their probabilistic-based weighting is computed by the Alchemy tool [Bellodi et al. 2010a]. Both the techniques in [Lamma et al. 2007a] and [Bellodi et al. 2010a], rely on the availability of compliant and non-compliant traces of execution, w.r.t. the process to mine. For instance, a real log from cervical cancer screening careflows is considered in [Lamma et al. 2007a]. All the traces have been analyzed by a domain expert and labeled as compliant or non compliant with respect to the protocol adopted in the screening center. [Bellodi et al. 2010b] takes as a case study the records of the student careers belonging to the same university of the authors. In this case, positive traces are represented by graduated students, whilst negative traces are related to students who did not conclude their studies. For a comprehensive insight on the logic-based approaches to declarative workflow mining, the reader can refer to [Montali 2010].

As in the aforementioned logic-based approaches, we preferred to elaborate a technique which avoided the replay of every trace on automata in the log. Thanks to this, the time for computing the result diminishes with respect to, e.g., the approach of [Maggi et al. 2011]. On the other hand, we had to deal with traces which were not labeled in advance. Therefore, our technique does not require the user's specification of positive and negative past executions.

3. SPECIFICATION OF DECLARATIVE WORKFLOWS AS CONSTRAINTS

Our control-flow discovery algorithm represents the mined processes as declarative models. In particular, we adopt only a subset of Declare constraints, as in [Maggi et al. 2011].

Constraints are temporal rules constraining the execution of activities. E.g., $\mathit{Response}(\rho, \sigma)$ is a constraint on the activities ρ and σ, forcing σ to be executed if the activity ρ was completed before. Such rules are meant to adhere to specific constraint *templates*. $\mathit{RespondedExistence}$ is the template of $\mathit{RespondedExistence}(\rho, \sigma)$. We further categorize constraint templates into *constraint types*. For instance, $\mathit{RespondedExistence}$ belongs to the $\mathit{RelationConstraint}$ type. $\mathit{MutualRelation}$ and $\mathit{NegativeRelation}$ are subtypes of $\mathit{RelationConstraint}$.

In the following, we briefly summarize the Declare constraint templates we use (see Table I). The reader may find further information in [Maggi et al. 2011; Pesic et al. 2007]. Figure 1 [Di Ciccio and Mecella 2013c] depicts the subsumption hierarchy of Declare constraints. Such hierarchy will be exploited by the mining algorithm in order to prune out redundant constraints, as explained later in this paper (Section 4.4). Such subsumption hierarchy of constraint templates builds upon our preliminary work [Di Ciccio and Mecella 2012a; 2012b; 2013c]. The contemporary works of [Maggi et al. 2013] and [Schunselaar et al. 2012] made analogous observations about the implications among constraint templates. Their studies lead to conclusions that mainly match with the considerations drawn here, from different perspectives. [Maggi et al. 2013] includes the concept of transitivity in the discussion. In other words, Maggi et al. investigate the case of those constraints that, if applied to pairs of activities $\{A, B\}$, $\{B, C\}$, constrain $\{A, C\}$ as well. [Schunselaar et al. 2012] investigates the case of constraint templates instantiated to sets of activities, rather than single ones. That is to say, Schunselaar et al. also consider the case of constraints referring to, e.g., $\{A, \{B, C\}\}$, or $\{\{A, B\}, \{C, D\}\}$. In this paper, we focus on the concept of subsumption, its impact on reliability metrics and the interaction between templates.

Declare constraints are always referred to an activity at least, which they constrain. The ***Existence***(M, ρ) constraint imposes ρ to appear at least M times in the trace. We rename $\mathit{Existence}(1, \rho)$ as $\mathit{Participation}(\rho)$. The ***Absence***$(N, \rho)$ constraint holds if ρ occurs at most $N - 1$ times in the trace. We call $\mathit{Absence}(2, \rho)$ as $\mathit{AtMostOne}(\rho)$. $\mathit{Init}(\rho)$ makes each trace start with ρ. $\mathit{End}(\rho)$ makes each trace end with ρ.

The aforementioned constraints fall under the type of $\mathit{ExistenceConstraint}$s, as each of them relates to a single activity. The following are named $\mathit{RelationConstraint}$s,

Constraint	**Explanation**	**Positive examples**		**Negative examples**	
Existence constraints					
Existence(n, a)	Activity a occurs at least n times in the trace				
Participation(a) $\equiv$ *Existence*(1, a)	a occurs at least *once*	✓ bcac	✓ bcaac	× bcc	× c
Absence($m + 1$, a)	a occurs at most m times				
AtMostOne(a) $\equiv$ *Absence*(2, a)	a occurs at most *once*	✓ bcc	✓ bcac	× bcaac	× bcacaa
Init(a)	a is the *first* to occur	✓ acc	✓ abac	× cc	× bac
End(a)	a is the *last* to occur	✓ bca	✓ baca	× bc	× bac
Relation constraints					
RespondedExistence(a, b)	If a occurs in the trace, then b occurs as well	✓ bcaac	✓ bcc	× caac	× acc
Response(a, b)	If a occurs, then b occurs after a	✓ caacb	✓ bcc	× caac	× bacc
AlternateResponse(a, b)	Each time a occurs, then b occurs afterwards, before a recurs	✓ cacb	✓ abcacb	× caacb	× bacacb
ChainResponse(a, b)	Each time a occurs, then b occurs immediately afterwards	✓ cabb	✓ abcab	× cacb	× bca
Precedence(a, b)	b occurs only if preceded by a	✓ cacbb	✓ acc	× ccbb	× bacc
AlternatePrecedence(a, b)	Each time b occurs, it is preceded by a and no other b can recur in between	✓ cacba	✓ abcaacb	× cacbba	× abbabcb
ChainPrecedence(a, b)	Each time b occurs, then a occurs immediately beforehand	✓ abca	✓ abaabc	× bca	× baacb
Mutual relation constraints					
CoExistence(a, b)	If b occurs, then a occurs, and viceversa	✓ cacbb	✓ bcca	× cac	× bcc
Succession(a, b)	a occurs if and only if it is followed by b	✓ cacbb	✓ accb	× bac	× bcca
AlternateSuccession(a, b)	a and b if and only if the latter follows the former, and they alternate each other in the trace	✓ cacbab	✓ abcabc	× caacbb	× bac
ChainSuccession(a, b)	a and b occur if and only if the latter immediately follows the former	✓ cabab	✓ ccc	× cacb	× cbac
Negative relation constraints					
NotChainSuccession(a, b)	a and b occur if and only if the latter does not immediately follows the former	✓ acbacb	✓ bbaa	× abcab	× cabc
NotSuccession(a, b)	a can never occur before b	✓ bbcaa	✓ cbbca	× aacbb	× abb
NotCoExistence(a, b)	a and b never occur together	✓ cccbbb	✓ ccac	× accbb	× bcac

Table I: Declare constraints

since they constrain pairs of activities. $\mathit{RespondedExistence}(\rho, \sigma)$ holds if, whenever ρ is executed, σ is either already executed or going to be executed – i.e., no matter if before or afterwards. $\mathit{Response}(\rho, \sigma)$, instead, imposes a temporal ordering, since it requires that if ρ is performed, then σ will be executed eventually ***afterwards***, i.e., before the enacted process ends. On the other hand, $\mathit{Precedence}(\rho, \sigma)$ states that σ cannot be executed if ρ was not executed ***beforehand***, at least once. We remark here that $\mathit{Precedence}(\rho, \sigma)$ does not specialize $\mathit{RespondedExistence}(\rho, \sigma)$, but $\mathit{RespondedExistence}(\sigma, \rho)$. This is due to the semantics of the constraints themselves. Therefore, $\mathit{Response}(\rho, \sigma)$ is subsumed by $\mathit{RespondedExistence}(\rho, \sigma)$, whilst $\mathit{Precedence}(\rho, \sigma)$ is subsumed by $\mathit{RespondedExistence}(\sigma, \rho)$. $\mathit{AlternateResponse}(\rho, \sigma)$ and $\mathit{AlternatePrecedence}(\rho, \sigma)$ strengthen respectively $\mathit{Response}(\rho, \sigma)$ and $\mathit{Precedence}(\rho, \sigma)$ by requiring that ***each*** ρ (σ) must be followed (preceded) by at least one occurrence of σ (ρ). The "alternation" is in that the trace cannot have two ρs (σs) in a row before σ (after ρ). $\mathit{ChainResponse}(\rho, \sigma)$ and $\mathit{ChainPrecedence}(\rho, \sigma)$, in turn, specialize $\mathit{AlternateResponse}(\rho, \sigma)$ and $\mathit{AlternatePrecedence}(\rho, \sigma)$, declaring that only σ (resp. ρ) can be performed immediately after ρ (before σ).

The constraints of type $\mathit{MutualRelation}$ are verified if and only if two $\mathit{RespondedExistence}$ (or descendant) constraints are satisfied, as it follows. $\mathit{CoExistence}(\rho, \sigma)$ holds if both $\mathit{RespondedExistence}(\rho, \sigma)$ and $\mathit{RespondedExistence}(\sigma, \rho)$ are respected. $\mathit{Succession}(\rho, \sigma)$ is valid if $\mathit{Response}(\rho, \sigma)$ and $\mathit{Precedence}(\rho, \sigma)$ are verified. The same holds with $\mathit{AlternateSuccession}(\rho, \sigma)$, equivalent to the conjunction of $\mathit{AlternateResponse}(\rho, \sigma)$ and $\mathit{AlternatePrecedence}(\rho, \sigma)$, and $\mathit{ChainSuccession}(\rho, \sigma)$, with respect to $\mathit{ChainResponse}(\rho, \sigma)$ and $\mathit{ChainPrecedence}(\rho, \sigma)$. The aforementioned relations

$\mathit{MutualRelation}$ constraint	*forward* constraint	*backward* constraint
$\mathit{CoExistence}(\rho, \sigma)$	$\mathit{RespondedExistence}(\rho, \sigma)$	$\mathit{RespondedExistence}(\sigma, \rho)$
$\mathit{Succession}(\rho, \sigma)$	$\mathit{Response}(\rho, \sigma)$	$\mathit{Precedence}(\rho, \sigma)$
$\mathit{AlternateSuccession}(\rho, \sigma)$	$\mathit{AlternateResponse}(\rho, \sigma)$	$\mathit{AlternatePrecedence}(\rho, \sigma)$
$\mathit{ChainSuccession}(\rho, \sigma)$	$\mathit{ChainResponse}(\rho, \sigma)$	$\mathit{ChainPrecedence}(\rho, \sigma)$

Table II: ***Forward*** and ***backward*** relations for $\mathit{MutualRelation}$ constraints

constitute the *forward* and *backward* associations in Figure 1. There, they are only drawn for $\mathit{CoExistence}$, for the sake of readability. The comprehensive list is reported in Table II. Finally, we consider $\mathit{NegativeRelation}$ constraints: they are satisfied when the related $\mathit{MutualRelation}$s (*negated*, in Figure 1) are not. $\mathit{NotChainSuccession}(\rho, \sigma)$ expresses the impossibility for σ to be executed immediately after ρ (the opposite of $\mathit{ChainSuccession}(\rho, \sigma)$). $\mathit{NotSuccession}(\rho, \sigma)$ generalizes the previous. It requires that, if ρ is performed, no other σ can be executed afterwards ($\mathit{Succession}(\rho, \sigma)$, then, is the *negated* constraint). $\mathit{NotCoExistence}(\rho, \sigma)$ is even more restrictive: if ρ is performed, not any σ can be executed, and vice-versa (the opposite of $\mathit{CoExistence}(\rho, \sigma)$). The "negated" relation is depicted in Figure 1 only for the $\mathit{MutualRelation}$ and $\mathit{NegatedRelation}$ types. It extends to the constraint templates as described in Table III.

Process Description 1 The example process

$\mathit{Response}$(r, p)
$\mathit{RespondedExistence}$(c, p)
$\mathit{Succession}$(p, n)
$\mathit{Participation}$(n), $\mathit{AtMostOne}$(n), End(n)

$NegativeRelation$ constraint	*negated* constraint
$NotCoExistence(\rho, \sigma)$	$CoExistence(\rho, \sigma)$
$NotSuccession(\rho, \sigma)$	$Succession(\rho, \sigma)$
$NotAlternateSuccession(\rho, \sigma)$	$AlternateSuccession(\rho, \sigma)$
$NotChainSuccession(\rho, \sigma)$	$ChainSuccession(\rho, \sigma)$

Table III: *Negated* relations for $NegativeRelation$ constraints

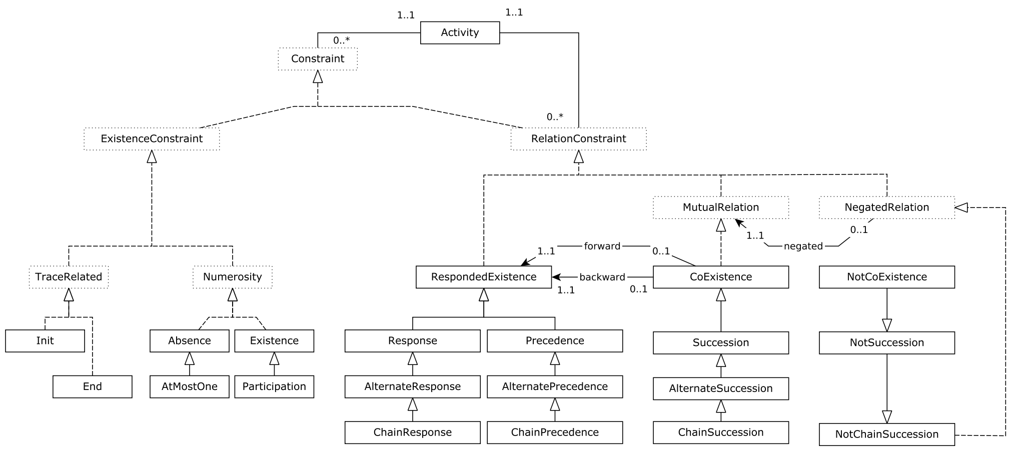


Fig. 1: The declarative process model's hierarchy of constraints. Taking into account the UML Class Diagram graphical notations, the Generalization ("is-a") relationships represent the subsumption between constraint templates. The subsumed is on the tail, the subsuming on the head. For example, $AlternateSuccession$ is subsumed by $Succession$ and $AlternateSuccession$ subsumes $ChainSuccession$. The Realization relationships indicate that the constraint template (as well as the subsumed ones in the hierarchy) belong to a specific type. Constraint templates are drawn as solid boxes, whereas the constraint types' boxes are dashed.

Here we outline a brief example: we want to model the process of defining an agenda for a research project meeting. The schedule is discussed by email among the participants. We suppose that a final agenda will be committed ("confirm" – n) after that requests for a new proposal ("request" – r), proposals themselves ("propose" – p) and comments ("comment" – c) have been circulated.

The aforementioned activities are bound to the following constraints (cf. Process Description 1). If a request is sent, then a proposal is expected to be prepared afterwards (cf. $Response$(r, p)). Comments can be given in order to review a proposed agenda, or for soliciting the formulation of a new proposal. Thus, the presence of c in the trace is constrained to the presence of p (cf. $RespondedExistence$(c, p)). A confirmation is supposed to be mandatorily given after the proposal. Conversely, any proposal is expected to precede a confirmation (cf. $Succession$(p, n)). We suppose the confirmation to be the $final$ activity (cf. End(n)). This mandatory task (cf. $Participation$(n)) is not expected to be executed more than once (cf. $AtMostOne$(n)).

As an example, the following traces would be compliant: pn, pcn, rpcn, rpcpn, rrpcrpcrcpcn, rpprpcccrpcn.

4. MINERFUL

MINERful, the control-flow discovery technique that we propose, is based on the concept of *MINERfulKB*: it keeps specific information extracted from the event log. Such knowledge base represents statistical data needed to discover the constraints of the declarative process represented by the log. In fact, MINERfulKB is queried during the execution of the technique.

Therefore, MINERful is a two-step algorithm. The first step consists of the construction of MINERfulKB (Section 4.3). The second step infers the declarative model through the analysis of MINERfulKB (Section 4.4). The final output is a set of constraints, verified on the knowledge base. In the following, formal definitions are provided.

4.1. MINERfulKB

Let Σ be a finite alphabet: the symbols in the alphabet are meant to correspond to *activities* in a process. Therefore, we will interchangeably use terms "activity", "character" and "symbol". A log is a collection of traces, i.e., a finite sequence of activities. We consider $T \subset \Sigma^*$ as the log, where Σ^* is the set of traces of elements in Σ. We will interchangeably use the terms "trace" and "string" for denoting every $t \in T$. We introduce six functions mapping to integers a log T and either one character $\rho \in \Sigma$ or two characters $\rho, \sigma \in \Sigma$. Such numbers are interpreted in MINERful as the result of specific quantitative analyses performed on collections of strings (logs). The six functions are grouped into *MINERful ownplay* and *MINERful interplay*. The former collects information referring to single characters. The latter describes the relation between couple of characters. We will call the possible occurrences of ρ and σ in the string as either *pivot* or *searched* characters, depending on the role they play in the definition of the following functions. We will denote by ρ' (resp., σ') any occurrence of ρ (resp., σ) in a string other than the pivot or searched character. In the examples, we will assign $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ to the alphabet Σ. ρ will be assigned as a and σ as b. The assigned log $\mathsf{T} \subset \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}^*$ for T will change, case by case.

Definition 4.1 (*MINERful interplay*). A tuple $\mathcal{I} = \langle \Sigma, \delta, \beta^{\rightarrow}, \beta^{\leftarrow} \rangle$, where:

$\delta(T, \rho, \sigma, d)$. $\delta : \Sigma^* \times \Sigma \times \Sigma \times \mathbb{Z} \rightarrow \mathbb{N}^+$ is the *distance* function. It maps a distance[1] $d \in \mathbb{Z}$ between pivot $\rho \in \Sigma$ and searched $\sigma \in \Sigma$ to the number of cases when they appeared at distance d, in the traces of log T (e.g., $\delta(\mathsf{T}, \mathsf{a}, \mathsf{b}, 2) = 4$ means that we have the evidence of a searched b appearing 2 characters after the pivot a in 4 cases, given $\mathsf{T} = \{\mathsf{cacbcc}, \mathsf{acbcacba}, \mathsf{acbaaa}\}$); we recall that $\mathbb{N}^+$ is the set of natural integers excluding zero[2];

$\beta^{\rightarrow}(T, \rho, \sigma)$. $\beta^{\rightarrow} : \Sigma^* \times \Sigma \times \Sigma \rightarrow \mathbb{N}$ is the *in-between onwards appearance* function. Given a pivot ρ and the closest *following* occurrence of searched σ, it counts the number of ρ's in-between, for every string of log T in which both ρ and σ occur. By closest following occurrence we mean that no other σ' is read after ρ and before σ. $\beta^{\rightarrow}(\mathsf{T}, \mathsf{a}, \mathsf{b}) = 2$ means, e.g., that a appeared 2 times between the preceding occurrence of pivot a and the *closest* following occurrence of searched b, as in $\mathsf{T} = \{\mathsf{accaacb}\}$, $\mathsf{T} = \{\mathsf{accabcaab}\}$, or $\mathsf{T} = \{\mathsf{accab}, \mathsf{caab}\}$);

$\beta^{\leftarrow}(T, \rho, \sigma)$. $\beta^{\leftarrow} : \Sigma^* \times \Sigma \times \Sigma \rightarrow \mathbb{N}$ is the *in-between onwards appearance* function. Given a pivot σ and the closest *preceding* occurrence of searched ρ, it counts

[1]The *distance* represents the number of characters between ρ and σ. It is a positive value if σ *follows* ρ, negative if σ *precedes* ρ.

[2]Thus, we do not consider the concurrency of events in a log, i.e., no pair of characters is read in the same position.

Function	Extended	Abbreviated
Distance	$\delta(T, \rho, \sigma, d)$	$\delta_{\rho,\sigma}(d)$
In-between onwards appearance	$\beta^{\rightarrow}(T, \rho, \sigma)$	$\beta^{\rightarrow}_{\rho,\sigma}$
In-between backwards appearance	$\beta^{\leftarrow}(T, \rho, \sigma)$	$\beta^{\leftarrow}_{\rho,\sigma}$
Global appearance	$\gamma(T, \rho, n)$	$\gamma_{\rho}(n)$
Initial appearance	$\alpha(T, \rho)$	α_{ρ}
Final appearance	$\omega(T, \rho)$	ω_{ρ}

Table IV: Abbreviations for the functions of MINERfulKB

the number of σ's in-between, for every string of log T in which both ρ and σ occur. By closest preceding occurrence we mean that no other ρ' is read before σ and after ρ. $\beta^{\leftarrow}(\mathsf{T}, \mathsf{a}, \mathsf{b}) = 3$ means, e.g., that b appeared 3 times between the following occurrence of pivot b and the *closest* preceding occurrence of searched a, as in $\mathsf{T} = \{\mathsf{acbcbbcb}\}$, $\mathsf{T} = \{\mathsf{accbbcbcabb}\}$, or $\mathsf{T} = \{\mathsf{accbbcb}, \mathsf{cabb}\}$).

Definition 4.2 (*MINERful ownplay*). A tuple $\mathcal{O} = \langle \Sigma, \gamma, \alpha, \omega \rangle$, where:

$\gamma(T, \rho, n)$. $\gamma : \Sigma^* \times \Sigma \times \mathbb{N} \to \mathbb{N}$ is the *global appearance* function. It maps the pivot ρ and a natural number $n \in \mathbb{N}$ to the number of traces of T in which ρ was read n times (e.g., $\gamma(\mathsf{T}, \mathsf{a}, 4) = 2$ means that the pivot a happens to be read exactly four times in only two strings in the log, as in $\mathsf{T} = \{\mathsf{aabbabcca}, \mathsf{babacaa}\}$);
$\alpha(T, \rho)$. $\alpha : \Sigma^* \times \Sigma \to \mathbb{N}$ is the *initial appearance* function. It represents the number of strings where the pivot ρ appeared as the *initial* symbol (e.g., if $\alpha(\mathsf{T}, \mathsf{a}) = 5$, five traces started with a, as in $\mathsf{T} = \{\mathsf{abc}, \mathsf{abbc}, \mathsf{aca}, \mathsf{aa}, \mathsf{a}\}$);
$\omega(T, \rho)$. $\omega : \Sigma^* \times \Sigma \to \mathbb{N}$ is the *final appearance* function. It represents the number of strings where the pivot ρ appeared as the *last* symbol (e.g., if $\omega(\mathsf{T}, \mathsf{a}) = 0$, no trace ended with a, as in $\mathsf{T} = \{\mathsf{abc}, \mathsf{abbc}\}$).

Definition 4.3 (*MINERfulKB*). A tuple $\mathcal{KB} = \langle \mathcal{I}, \mathcal{O} \rangle$ where $\mathcal{I} = \langle \Sigma, \delta, \beta^{\rightarrow}, \beta^{\leftarrow} \rangle$ is the MINERful interplay, and $\mathcal{O} = \langle \Sigma, \gamma, \alpha, \omega \rangle$ is the MINERful ownplay.

Notational convention. For the sake of readability, we will put input characters as indexes in the subscript of the function symbols. We will remove the explicit reference to log T, too. Hence, we will have the abbreviations listed in Table IV.

With a slight abuse of notation, we consider $\delta_{\rho,\sigma}(+\infty)$, i.e., $\delta(T, \rho, \sigma, +\infty)$, and $\delta_{\rho,\sigma}(-\infty)$, i.e., $\delta(T, \rho, \sigma, -\infty)$, as the number of cases in which the searched σ, respectively, did not appear in a string *after* the pivot ρ, and did not appear in a string *before* ρ. $\delta_{\rho,\sigma}(\pm\infty)$, alias $\delta(T, \rho, \sigma, \pm\infty)$, represents the number of cases in which the searched σ did not appear at all in the strings where ρ occurred, i.e., neither before nor after.

For the sake of brevity, here we also define the following function:

$$\Gamma_{\rho} = \sum_{n>0} \gamma_{\rho}(n) \cdot n$$

It is meant to count the number of appearances of ρ in log T. □

As an example, let us suppose to interpret MINERfulKB over $\mathbf{T} = \{\mathsf{aabbac}\}$ (for simplicity, a log with a single trace). Then, for what a, b and c are concerned, $\mathcal{I}$ is

	$-\infty$	$\cdots$	-5	-4	-3	-2	-1	$\pm\infty$	$+1$	$+2$	$+3$	$+4$	$+5$	$\cdots$	$+\infty$
$\delta_{\mathsf{a},\mathsf{b}}$	2	0	0	0	0	1	1	0	1	2	1	0	0	0	1
$\delta_{\mathsf{a},\mathsf{c}}$	3	0	0	0	0	0	0	0	1	0	0	1	1	0	0
$\delta_{\mathsf{b},\mathsf{a}}$	0	0	0	0	1	2	1	0	1	1	0	0	0	0	0
$\delta_{\mathsf{b},\mathsf{c}}$	2	0	0	0	0	0	0	0	0	1	1	0	0	0	0
$\delta_{\mathsf{c},\mathsf{a}}$	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1
$\delta_{\mathsf{c},\mathsf{b}}$	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1

(a) The δ function

$$
\begin{array}{lcllcl}
\beta^{\rightarrow}_{\mathsf{a},\mathsf{b}} & = & 1; & \beta^{\leftarrow}_{\mathsf{a},\mathsf{b}} & = & 1 \\
\beta^{\rightarrow}_{\mathsf{a},\mathsf{c}} & = & 2; & \beta^{\leftarrow}_{\mathsf{a},\mathsf{c}} & = & 0 \\
\beta^{\rightarrow}_{\mathsf{b},\mathsf{a}} & = & 1; & \beta^{\leftarrow}_{\mathsf{b},\mathsf{a}} & = & 0 \\
\beta^{\rightarrow}_{\mathsf{b},\mathsf{c}} & = & 1; & \beta^{\leftarrow}_{\mathsf{b},\mathsf{c}} & = & 0 \\
\beta^{\rightarrow}_{\mathsf{c},\mathsf{a}} & = & 0; & \beta^{\leftarrow}_{\mathsf{c},\mathsf{a}} & = & 0 \\
\beta^{\rightarrow}_{\mathsf{c},\mathsf{b}} & = & 0; & \beta^{\leftarrow}_{\mathsf{c},\mathsf{b}} & = & 0
\end{array}
$$

(b) The $\beta^{\rightarrow}$ and $\beta^{\leftarrow}$ functions

Table V: An example of MINERful interplay, interpreted over $\mathbf{T} = \{\mathsf{aabbac}\}$

shown in Table V, and $\mathcal{O}$ is as follows:

$$\left\langle \gamma_{\mathsf{a}}(n) = \left\{ \begin{array}{ll} 1 & n = 3 \\ 0 & n \in \mathbb{N} \setminus \{3\} \end{array} \right\}, \alpha_{\mathsf{a}} = 1, \omega_{\mathsf{a}} = 0 \right\rangle$$

$$\left\langle \gamma_{\mathsf{b}}(n) = \left\{ \begin{array}{ll} 1 & n = 2 \\ 0 & n \in \mathbb{N} \setminus \{2\} \end{array} \right\}, \alpha_{\mathsf{b}} = 0, \omega_{\mathsf{b}} = 0 \right\rangle$$

$$\left\langle \gamma_{\mathsf{c}}(n) = \left\{ \begin{array}{ll} 1 & n = 1 \\ 0 & n \in \mathbb{N} \setminus \{1\} \end{array} \right\}, \alpha_{\mathsf{c}} = 0, \omega_{\mathsf{c}} = 1 \right\rangle$$

4.2. The algorithm: a bird's eye view

Algorithm 1 presents a bird's eye view of the technique. The algorithm accepts in input a set of strings T and an alphabet Σ. It requires that the characters of the strings in T belong to the alphabet Σ. The output is a set of discovered constraints, $\mathcal{B}^{+}$. The algorithm computes the values of metrics specifying reliability (Support) and relevance (Confidence Level and Interest Factor) for each constraint in $\mathcal{B}^{+}$. Optionally, the user can specify thresholds for those metrics, in order to remove those constraints that are not associated to a sufficient degree of reliability or relevance. The different steps of the algorithm will be detailed in the following sections. We will describe how the MINERfulKB is computed in Section 4.3. Section 4.4 will explain the procedure for discovering constraints. The construction of the knowledge base is delegated to the

Algorithm 1 The MINERful pseudo-code algorithm, in its simplest form (bird's eye view)

```
$\mathcal{KB} \leftarrow$ computeKBOnwards($T, \Sigma, \emptyset$)
$\mathcal{KB} \leftarrow$ computeKBBackwards($T, \Sigma, \mathcal{KB}$)
$\mathcal{B}^+ \leftarrow$ discoverConstraints$^+$($\mathcal{KB}, \Sigma, |T|$)
return $\mathcal{B}^+$
```

computeKBOnwards and computeKBBackwards procedures. They are designed to be completely on-line. Indeed, they refine the MINERfulKB as new strings occur and new characters are read. discoverConstraints leverages on the outcome of computeKBOnwards and computeKBBackwards, in order to return the discovered process.

4.3. Construction of MINERfulKB

The input of this algorithm is an alphabet of activities and a log, which are possible assignments for (resp.) Σ and T, referring back to the definitions of Section 4.1. Here we call the input log L and the input process alphabet A. For each activity $a \in A$, a unique identifier is considered. For each trace $l \in L$, a string of unique activities' identifiers is taken into account. We recall that Σ pertains to the interpretation of MINERful interplay $\mathcal{I}$ and MINERful ownplay $\mathcal{O}$, whereas T is the collection of strings passed as a parameter to all functions in $\mathcal{I}$ and $\mathcal{O}$. Therefore, the algorithm computes an interpretation function $\left({}^{\mathcal{L},\mathcal{A}}\cdot\right)$ for MINERfulKB $\mathcal{KB}$ over L and A, considering the activities in A as the characters of Σ and the traces of L as the strings of T. At the end of the run, we have the interpretations for both MINERful interplay and MINERful ownplay on the basis of L and A, i.e.,

$${}^{\mathcal{L},\mathcal{A}}\mathcal{KB} = \left\langle {}^{\mathcal{L},\mathcal{A}}\mathcal{I}, {}^{\mathcal{L},\mathcal{A}}\mathcal{O} \right\rangle$$

where

$${}^{\mathcal{L},\mathcal{A}}\mathcal{I} = \langle {}^{\mathcal{L},\mathcal{A}}\Sigma, {}^{\mathcal{L},\mathcal{A}}\delta, {}^{\mathcal{L},\mathcal{A}}\beta^{\rightarrow}, {}^{\mathcal{L},\mathcal{A}}\beta^{\leftarrow} \rangle$$

and

$${}^{\mathcal{L},\mathcal{A}}\mathcal{O} = \langle {}^{\mathcal{L},\mathcal{A}}\Sigma, {}^{\mathcal{L},\mathcal{A}}\gamma, {}^{\mathcal{L},\mathcal{A}}\alpha, {}^{\mathcal{L},\mathcal{A}}\omega \rangle$$

(see Section 4.1).

The ${}^{\mathcal{L},\mathcal{A}}\cdot$ notation will be omitted in the remainder, in order to ease the reading of text and formulae. Thus, we will implicitly refer to the interpreted MINERfulKB, when mentioning $\mathcal{KB}$ and all the related functions.

Before starting the description of Algorithm 2, we resume here the adopted notation. *Sets* differ from *lists* in that they cannot have multiple copies of the same value. Therefore, if, e.g., $X = \{x, y\}$ then $X \cup \{x\} = \{x, y\}$, i.e., unions only consider distinct items (cf. line 13 in Algorithm 2). Lists have an explicit positional indexing over the inserted values. Hence, $\overrightarrow{\mathrm{p}_\rho}[j]$ (see line 27 in Algorithm 2), is pointing at the j-th element in the $\overrightarrow{\mathrm{p}_\rho}$ list. Strings are considered as lists of characters: thus, $t[i]$ refers to the i-th character in the string t (see line 12 in Algorithm 2), where i ranges from 1 to $|t|$. Lists and strings are provided with a concatenation function $\circ$: for instance, the effect of $\overrightarrow{\mathrm{p}_\sigma} \leftarrow \overrightarrow{\mathrm{p}_\sigma} \circ \{i\}$ is to add i as the last element in $\overrightarrow{\mathrm{p}_\sigma}$ (see line 14 in Algorithm 2). For pointing at a specific element in a map (indexed multi-set), we specify the "coordinates" between pairs of brackets, as for a bi-dimensional array: e.g., $\mathtt{N}[r][s]$ is the element in $\mathtt{N}$ corresponding to r and s (see line 7 in Algorithm 2). When pointing at the whole sub-map corresponding to a single character, we insert only the target symbol, as selecting a row in a bi-dimensional array: e.g., $\mathtt{N}[r]$ is the sub-map in $\mathtt{N}$ corresponding to r (see line 41 in Algorithm 2).

Algorithm 2 The COMPUTEKBONWARDS procedure's pseudo-code

```
procedure COMPUTEKBONWARDS($T, \Sigma, \mathcal{KB}$)
    $\forall d \in \mathbb{Z}\ \forall \rho \in \Sigma\ \forall \sigma \in \Sigma\ .\quad \delta_{\rho,\sigma}(d) \leftarrow 0$
    $\forall n \in \mathbb{N}^{+}\ \forall \rho \in \Sigma\ .\quad \gamma_{\rho}(x) \leftarrow 0$
    for all $t \subseteq T$ do
        $\alpha_{t[1]} \leftarrow \alpha_{t[1]} + 1$
        $\mathtt{R} := \emptyset$    # $\mathtt{R}$: set of characters already appeared in $t$
        $\forall r, s \in \Sigma\ .\quad \mathtt{N}[r][s] := 0$    # $\mathtt{N}$: bi-indexed map, counting the missing $s$'s after $r$
        $\forall r \in \Sigma\ .\quad \overrightarrow{\mathtt{p}_r} := \{\}$    # $\overrightarrow{\mathtt{p}_r}$: list of indexes where $r$ appears in $t$
        $\forall r, s \in \Sigma\ .\quad \mathtt{W}[r][s] := 0$    # $\mathtt{W}$: counts the $r$'s repeated before the next $s$
        $\forall r, s \in \Sigma\ .\quad \widehat{\mathtt{W}[r][s]} := \bot$    # $\widehat{\mathtt{W}}$: flags granting the update of $\mathtt{W}$
        for $i = 1 \rightarrow |t|$ do
            $\sigma := t[i]$
            $\mathtt{R} := \mathtt{R} \cup \{\sigma\}$
            $\overrightarrow{\mathtt{p}_\sigma} := \overrightarrow{\mathtt{p}_\sigma} \circ \{i\}$
            for all $\rho \in \mathtt{R}$ do
                if $\rho = \sigma$ then
                    for all $s \in \Sigma \setminus \{\rho\}$ do
                        $\mathtt{N}[\rho][s] := \mathtt{N}[\rho][s] + 1$
                        if $\widehat{\mathtt{W}[\rho][s]} = \bot$ then
                            $\widehat{\mathtt{W}[\rho][s]} := \top$
                        else
                            $\mathtt{W}[\rho][s] := \mathtt{W}[\rho][s] + 1$
                        end if
                    end for
                else
                    for $j = 1 \rightarrow |\overrightarrow{\mathtt{p}_\rho}|$ do
                        $\delta_{\rho,\sigma}(i - \overrightarrow{\mathtt{p}_\rho}[j]) \leftarrow \delta_{\rho,\sigma}(i - \overrightarrow{\mathtt{p}_\rho}[j]) + 1$
                    end for
                    $\mathtt{N}[\rho][\sigma] := 0$
                    if $\widehat{\mathtt{W}[\rho][\sigma]} = \top$ then
                        $\beta_{\overrightarrow{\rho,\sigma}} \leftarrow \beta_{\overrightarrow{\rho,\sigma}} + \mathtt{W}[\rho][\sigma]$
                        $\widehat{\mathtt{W}[\rho][\sigma]} := \bot$, $\mathtt{W}[\rho][\sigma] := 0$
                    end if
                end if
            end for
        end for
        for all $r \in \mathtt{R}$ do
            for all $\bar{s} \in \Sigma \setminus \mathtt{R}$ do
                $\delta_{r,\bar{s}}(\pm\infty) \leftarrow \delta_{r,\bar{s}}(\pm\infty) + |\overrightarrow{\mathtt{p}_r}|$
            end for
            for all $\bar{s} \in \Sigma \setminus \{r\}$ do
                $\delta_{r,\bar{s}}(+\infty) \leftarrow \delta_{r,\bar{s}}(+\infty) + \mathtt{N}[r][\bar{s}]$
            end for
        end for
        for all $s \in \Sigma$ do
            $\gamma_s(|\overrightarrow{\mathtt{p}_s}|) \leftarrow \gamma_s(|\overrightarrow{\mathtt{p}_s}|) + 1$
        end for
        $\omega_{t[|t|]} \leftarrow \omega_{t[|t|]} + 1$
    end for
    return $\mathcal{KB}$
end procedure
```

In order to make it easier for the reader to distinguish between assignments of temporary variables and the update of the interpretation for MINERfulKB, we denote the former with $:=$ (see Algorithm 2 from line 6 to line 10), the latter with $\leftarrow$ (see e.g., line 5 in Algorithm 2).

From line 2 to line 3, the interpretations of the γ and δ functions are initialized, supposing that they are constant and equal to 0. Then, for each string t in T (line 4), the first character appearing ($t[0]$) is recorded into the related $\alpha_{t[0]}$ as the first one (line 5). After the initialization of auxiliary data structures (whose role is briefly explained in-line in the code itself and later in this Section), the analysis of the single characters in the string begins (line 11).

First of all, the encountered character σ is added to the set of appeared characters in t, namely $\mathtt{R}$. Then, the current index is concatenated ($\circ$ operation) to the list of positions where σ was read in t ($\overrightarrow{\mathrm{p}_\sigma}$), at line 14. On line 15 the algorithm starts the computation of interleaving statistics between characters.

Within a cycle over each character already found in the string, $\rho \in \mathtt{R}$, the algorithm proceeds differently, depending on whether the condition at line 16 is satisfied or not. Specifically, the statement at line 16 checks whether the currently read character, σ, has already appeared in the string (i.e., σ coincides with ρ). If this is the case, $\mathtt{N}[\rho][s]$ is incremented by 1 (line 18), where s is any character in the alphabet besides ρ ($s \in \Sigma \setminus \rho$). $\mathtt{N}[\rho][s]$ indeed is a temporary structure counting the number of times in which ρ is read but s does not occur any later in the string. Such counter will be reset if s appears afterwards (see line 29). Otherwise, its value is going to be "flushed" into $\delta_{\rho,s}(+\infty)$ at the end of string t (see line 42). We recall that $\delta_{\rho,s}(+\infty)$ counts the number of times in which ρ is read in the string whilst s does not occur afterwards. A running example of such operations is shown in Table VI.

From line 19 to line 23, the algorithm updates the counters for repeated occurrences of ρ before the next occurrence of s: $\widehat{\mathtt{W}[\rho][s]}$ is the flag for incrementing the $\mathtt{W}[\rho][s]$ counter; hence, if it is set to false, it gets true, whereas if it is already true, $\mathtt{W}[\rho][s]$ is incremented by 1. This is due to the fact that when the next occurrence of s is found in the string, the value of $\mathtt{W}[\rho][s]$ will be flushed as an increment to $\beta^{\rightarrow}_{\rho,s}$ (see line 31). Thereafter, $\widehat{\mathtt{W}[\rho][s]}$ and $\mathtt{W}[\rho][s]$ will be reset, respectively, to $\perp$ and 0 (line 32). A running example showing the evolution of $\widehat{\mathtt{W}}$, $\mathtt{W}$, and $\beta^{\rightarrow}_{\cdot,\cdot}$ is shown in Table VII.

At line 27, $\delta_{\rho,\sigma}(i - \overrightarrow{\mathrm{p}_\rho}[j])$ is incremented by 1 for every element in $\overrightarrow{\mathrm{p}_\rho}$. $\delta_{\rho,\sigma}(d)$ counts how many times σ followed ρ after d characters along the log. The update instruction is entered because the condition at line 16 is not satisfied, i.e., the encountered σ differs from ρ in the loop over $\mathtt{R}$. Therefore, the value assumed by $\delta_{\rho,\sigma}$ at the current distance between ρ and σ has to be incremented by 1. However, we may have not only one position where ρ occurred, but many. E.g., let us consider aaccccacab...: there, the pivot a was read at position 1, 2, 7 and 9, and the searched b at position 10. Thus, b must be recorded to appear at distance 1, 3, 8 and 9 from a. Recalling that $\overrightarrow{\mathrm{p}_\rho}$ collects all the indexes where ρ is read (see line 14), this is what happens at line 27. Such operation is repeated for each position of ρ in $\overrightarrow{\mathrm{p}_\rho}$, i.e., inside the loop starting at line 26. In the example, i would point at the last character read, b. Therefore, let us assume that i is equal to 10. $\overrightarrow{\mathrm{p_a}} = \{1, 2, 7, 9\}$, index j in loop at line 26 ranges from 1 to 4 and, thus, we have:

— for $j = 1$, $(i - \overrightarrow{\mathrm{p_a}}[j])$ is equal to 9,
— for $j = 2$, $(i - \overrightarrow{\mathrm{p_a}}[j])$ is equal to 7,
— for $j = 3$, $(i - \overrightarrow{\mathrm{p_a}}[j])$ is equal to 2,
— for $j = 4$, $(i - \overrightarrow{\mathrm{p_a}}[j])$ is equal to 1.

$\langle \mathtt{N}, \delta_{\cdot,\cdot}(+\infty)\rangle \backslash^{\sigma \in t}$	a	a	b	b	a	c	
$\langle \mathtt{N}[\mathsf{a}][\mathsf{b}],\ \delta_{\mathsf{a},\mathsf{b}}(+\infty)\rangle$	$\langle 1,-\rangle$	$\langle 2,-\rangle$	$\langle 0,-\rangle$	$\langle 0,-\rangle$	$\langle 1,-\rangle$	$\langle 1,-\rangle$	$\langle 0,+1\rangle$
$\langle \mathtt{N}[\mathsf{a}][\mathsf{c}],\ \delta_{\mathsf{a},\mathsf{c}}(+\infty)\rangle$	$\langle 1,-\rangle$	$\langle 2,-\rangle$	$\langle 2,-\rangle$	$\langle 2,-\rangle$	$\langle 3,-\rangle$	$\langle 0,-\rangle$	$\langle 0,+0\rangle$
$\langle \mathtt{N}[\mathsf{b}][\mathsf{a}],\ \delta_{\mathsf{b},\mathsf{a}}(+\infty)\rangle$	$\langle 0,-\rangle$	$\langle 0,-\rangle$	$\langle 1,-\rangle$	$\langle 2,-\rangle$	$\langle 0,-\rangle$	$\langle 0,-\rangle$	$\langle 0,+0\rangle$
$\langle \mathtt{N}[\mathsf{b}][\mathsf{c}],\ \delta_{\mathsf{b},\mathsf{c}}(+\infty)\rangle$	$\langle 0,-\rangle$	$\langle 0,-\rangle$	$\langle 1,-\rangle$	$\langle 2,-\rangle$	$\langle 0,-\rangle$	$\langle 0,-\rangle$	$\langle 0,+0\rangle$
$\langle \mathtt{N}[\mathsf{c}][\mathsf{a}],\ \delta_{\mathsf{c},\mathsf{a}}(+\infty)\rangle$	$\langle 0,-\rangle$	$\langle 0,-\rangle$	$\langle 0,-\rangle$	$\langle 0,-\rangle$	$\langle 0,-\rangle$	$\langle 1,-\rangle$	$\langle 0,+1\rangle$
$\langle \mathtt{N}[\mathsf{c}][\mathsf{b}],\ \delta_{\mathsf{c},\mathsf{b}}(+\infty)\rangle$	$\langle 0,-\rangle$	$\langle 0,-\rangle$	$\langle 0,-\rangle$	$\langle 0,-\rangle$	$\langle 0,-\rangle$	$\langle 1,-\rangle$	$\langle 0,+1\rangle$

Table VI: The evolution of $\mathtt{N}$ and $\delta_{\cdot,\cdot}(+\infty)$, over the reading of a string $t = \mathsf{aabbac}$

Therefore, the following values have to be incremented by 1: $\delta_{\mathsf{a},\mathsf{b}}(9)$, $\delta_{\mathsf{a},\mathsf{b}}(7)$, $\delta_{\mathsf{a},\mathsf{b}}(2)$ and $\delta_{\mathsf{a},\mathsf{b}}(1)$. This is probably one of the most difficult steps of the algorithm. However, it prevents the analysis to be repeated like a transitive closure, on each string for every appeared character.

The final part of the outermost cycle updates the counters on the basis of the previously gathered information. The instruction of line 39 records the number of times in which the read character, r, occurred in the current string t, but $\bar{s}$ did not. In fact, $\overrightarrow{\mathsf{p}_r}$ records the indexes where character r occurred in the string. $\delta_{r,\bar{s}}(\pm\infty)$ counts how many times r is read in the strings where $\bar{s}$ does not occur. Therefore, it is incremented by $|\overrightarrow{\mathsf{p}_r}|$. For instance, if the alphabet Σ comprises a, b and c, and aababbb is read, we have that a occurred at positions 1, 2 and 4. Therefore, $\overrightarrow{\mathsf{p}_\mathsf{a}} = \{1, 2, 4\}$ and thus $|\overrightarrow{\mathsf{p}_\mathsf{a}}| = 3$. Following the same rationale, $|\overrightarrow{\mathsf{p}_\mathsf{b}}| = 4$ and $|\overrightarrow{\mathsf{p}_\mathsf{c}}| = 0$. Since no c was read, $\Sigma \setminus \mathtt{R} = \{\mathsf{c}\}$. Therefore, $\delta_{\mathsf{a},\mathsf{c}}$ is incremented by 3 (3 is the value of $|\overrightarrow{\mathsf{p}_\mathsf{a}}|$) and $\delta_{\mathsf{b},\mathsf{c}}$ is incremented by 4 ($|\overrightarrow{\mathsf{p}_\mathsf{b}}|$).

On line 46, $\gamma_s(|\overrightarrow{\mathsf{p}_s}|)$ is updated for each $s \in \Sigma$. In fact, $\gamma_s(n)$ counts in how many strings the character s appeared n times. Therefore, the number of occurrences of s in t, namely $|\overrightarrow{\mathsf{p}_s}|$, is the argument, and the referred value is incremented by 1. $|\overrightarrow{\mathsf{p}_s}|$ can amount to 0 as well, if s was never read in t. Again, if, e.g., the alphabet Σ comprises a, b and c, and aaabbbb is read, then $|\overrightarrow{\mathsf{p}_\mathsf{a}}| = 3$, $|\overrightarrow{\mathsf{p}_\mathsf{b}}| = 4$ and $|\overrightarrow{\mathsf{p}_\mathsf{c}}| = 0$. Therefore, $\gamma_\mathsf{a}(3)$ is incremented by 1, as well as $\gamma_\mathsf{b}(4)$ and $\gamma_\mathsf{c}(0)$.

In the end, (line 48), $\omega_{t[|t|]}$ is incremented by 1. This is because $\omega_\cdot$ stores the number of appearances of a given character as the last in the string, and $t[|t|]$ is the last character appearing in t. For instance, when a string $t = \mathsf{aabbac}$ is read, ω_c increments its value by 1.

COMPUTEKBBACKWARDS is the analog of COMPUTEKBONWARDS, as both build the knowledge base. However, the latter reads the strings from left to right, whereas the former parses the strings from right to left. Therefore, here we only reported the pseudo-algorithm of COMPUTEKBONWARDS (Algorithm 2), since COMPUTEKBBACKWARDS differs from it in few details. The only differences are in that COMPUTEKBBACKWARDS:

— does not update either the γ, nor the α nor the ω functions (namely, it does not contribute to give an interpretation to MINERful ownplay, being this task already fulfilled by COMPUTEKBONWARDS);

$\langle\langle \widehat{\mathtt{W}}, \mathtt{W}\rangle, \beta^{\rightarrow}_{\cdot,\cdot}\rangle \backslash^{\sigma \in t}$	a	a	b	b	a	c
$\langle\langle \widehat{\mathtt{W}[\mathsf{a}][\mathsf{b}]}, \mathtt{W}[\mathsf{a}][\mathsf{b}]\rangle, \beta^{\rightarrow}_{\mathsf{a},\mathsf{b}}\rangle$	$\langle\langle\top,0\rangle,-\rangle$	$\langle\langle\top,1\rangle,-\rangle$	$\langle\langle\bot,0\rangle,+1\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\top,0\rangle,-\rangle$	$\langle\langle\top,0\rangle,-\rangle$
$\langle\langle \widehat{\mathtt{W}[\mathsf{a}][\mathsf{c}]}, \mathtt{W}[\mathsf{a}][\mathsf{c}]\rangle, \beta^{\rightarrow}_{\mathsf{a},\mathsf{c}}\rangle$	$\langle\langle\top,0\rangle,-\rangle$	$\langle\langle\top,1\rangle,-\rangle$	$\langle\langle\top,1\rangle,-\rangle$	$\langle\langle\top,1\rangle,-\rangle$	$\langle\langle\top,2\rangle,-\rangle$	$\langle\langle\bot,0\rangle,+2\rangle$
$\langle\langle \widehat{\mathtt{W}[\mathsf{b}][\mathsf{a}]}, \mathtt{W}[\mathsf{b}][\mathsf{a}]\rangle, \beta^{\rightarrow}_{\mathsf{b},\mathsf{a}}\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\top,0\rangle,-\rangle$	$\langle\langle\top,1\rangle,-\rangle$	$\langle\langle\bot,0\rangle,+1\rangle$	$\langle\langle\bot,0\rangle,-\rangle$
$\langle\langle \widehat{\mathtt{W}[\mathsf{b}][\mathsf{c}]}, \mathtt{W}[\mathsf{b}][\mathsf{c}]\rangle, \beta^{\rightarrow}_{\mathsf{b},\mathsf{c}}\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\top,0\rangle,-\rangle$	$\langle\langle\top,1\rangle,-\rangle$	$\langle\langle\top,1\rangle,-\rangle$	$\langle\langle\bot,0\rangle,+1\rangle$
$\langle\langle \widehat{\mathtt{W}[\mathsf{c}][\mathsf{a}]}, \mathtt{W}[\mathsf{c}][\mathsf{a}]\rangle, \beta^{\rightarrow}_{\mathsf{c},\mathsf{a}}\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\top,0\rangle,-\rangle$
$\langle\langle \widehat{\mathtt{W}[\mathsf{c}][\mathsf{b}]}, \mathtt{W}[\mathsf{c}][\mathsf{b}]\rangle, \beta^{\rightarrow}_{\mathsf{c},\mathsf{b}}\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\bot,0\rangle,-\rangle$	$\langle\langle\top,0\rangle,-\rangle$

Table VII: The evolution of $\widehat{\mathtt{W}}$, $\mathtt{W}$, and $\beta^{\rightarrow}_{\cdot,\cdot}$ over the reading of a string $t = \mathsf{aabbac}$

— does not update $\delta_{\cdot,\cdot}(\pm\infty)$, since COMPUTEKBBACKWARDS already detected characters never appeared in the string, if any;
— reverses the sign of i, the counter of the current index in the string (namely, it is initialized with -1 and proceeds being decremented by 1 at each step);
— updates the δ function for $-\infty$ values, instead of $+\infty$, whenever the same conditions of line 41 in Algorithm 2 are verified;
— updates the $\beta^{\leftarrow}_{\rho,\sigma}$ function at line 31, instead of $\beta^{\rightarrow}_{\rho,\sigma}$.

The procedure for building the knowledge base of MINERful (Algorithm 2), is

(1) linear time w.r.t. the number of strings in the log,
(2) quadratic time w.r.t. the size of strings in the log,
(3) quadratic time w.r.t. the size of the alphabet;

therefore, the complexity is $O(|T| \cdot |t_{max}|^2 \cdot |\Sigma|^2)$. The proof is reported in the on-line appendix. Intuitively, we recall here that the nature of MINERfulKB itself allows us to specify an algorithm which is on-line, i.e., it refines MINERfulKB as new strings occur (1) and new characters in the string are read, with no need to go back on already processed data in the end. For each symbol picked up from the string, the temporary data structures related to the occurrences of characters that have already appeared are updated (2). Finally, every activity is linked to every other activity in the alphabet 3, within the knowledge base.

4.4. Discovery of constraints and their metrics

The set of constraints mined by our technique is listed in Table VIII. In particular, Table VIII shows the functions used in order to compute Support, with respect to MINERfulKB. Support is the normalized fraction of cases in which the constraint is verified, over the set of traces in T. Its value ranges from 0 to 1 and estimates the likelihood for a constraint to hold true in the discovered process. The functions in Table VIII are all based on mathematical operations performed on data coming from the MINERfulKB. In addition, they consider the size of T, i.e., how many strings were read. For ease of readability, we omit the $\mathcal{KB}$ and $|T|$ parameters from the list of each function, since they can be considered as a common shared knowledge. The rationale of such functions is based on the semantics of constraints and classical probability definitions, as largely discussed in the on-line appendix.

Support is a metric adopted in [Maggi et al. 2012] as well, but with a slight difference in the computation. There, it corresponds to the number of traces where the constraint is satisfied, w.r.t. the number of traces in the log. Here, instead, we changed the

Constraint	Support function
$Existence(n, \rho)$	$1 - \frac{\sum_{i=0}^{n-1} \gamma_\rho(i)}{\lvert T \rvert}$
$Participation(\rho)$	$1 - \frac{\gamma_\rho(0)}{\lvert T \rvert}$
$Absence(m, \rho)$	$\frac{\sum_{i=0}^{m} \gamma_\rho(i)}{\lvert T \rvert}$
$AtMostOne(\rho)$	$\frac{\gamma_\rho(0)+\gamma_\rho(1)}{\lvert T \rvert}$
$Init(\rho)$	$\frac{\alpha_\rho}{\lvert T \rvert}$
$End(\rho)$	$\frac{\omega_\rho}{\lvert T \rvert}$
$RespondedExistence(\rho, \sigma)$	$1 - \frac{\delta_{\rho,\sigma}(\pm\infty)}{\Gamma_\rho}$
$Response(\rho, \sigma)$	$1 - \frac{\delta_{\rho,\sigma}(+\infty)}{\Gamma_\rho}$
$Precedence(\rho, \sigma)$	$1 - \frac{\delta_{\rho,\sigma}(-\infty)}{\Gamma_\sigma}$
$AlternateResponse(\rho, \sigma)$	$1 - \frac{\beta^{\rightarrow}_{\rho,\sigma}+\delta_{\rho,\sigma}(+\infty)}{\Gamma_\rho}$
$AlternatePrecedence(\rho, \sigma)$	$1 - \frac{\beta^{\leftarrow}_{\rho,\sigma}+\delta_{\sigma,\rho}(-\infty)}{\Gamma_\sigma}$
$ChainResponse(\rho, \sigma)$	$\frac{\delta_{\rho,\sigma}(1)}{\Gamma_\rho}$
$ChainPrecedence(\rho, \sigma)$	$\frac{\delta_{\sigma,\rho}(-1)}{\Gamma_\sigma}$
$CoExistence(\rho, \sigma)$	$1 - \frac{\delta_{\rho,\sigma}(\pm\infty)+\delta_{\sigma,\rho}(\pm\infty)}{\Gamma_\rho+\Gamma_\sigma}$
$NotCoExistence(\rho, \sigma)$	$\frac{\delta_{\rho,\sigma}(\pm\infty)+\delta_{\sigma,\rho}(\pm\infty)}{\Gamma_\rho+\Gamma_\sigma}$
$Succession(\rho, \sigma)$	$1 - \frac{\delta_{\rho,\sigma}(+\infty)+\delta_{\sigma,\rho}(-\infty)}{\Gamma_\rho+\Gamma_\sigma}$
$NotSuccession(\rho, \sigma)$	$\frac{\delta_{\rho,\sigma}(+\infty)+\delta_{\sigma,\rho}(-\infty)}{\Gamma_\rho+\Gamma_\sigma}$
$AlternateSuccession(\rho, \sigma)$	$1 - \frac{\beta^{\rightarrow}_{\rho,\sigma}+\delta_{\rho,\sigma}(+\infty)+\beta^{\leftarrow}_{\rho,\sigma}+\delta_{\sigma,\rho}(-\infty)}{\Gamma_\rho+\Gamma_\sigma}$
$ChainSuccession(\rho, \sigma)$	$\frac{\delta_{\rho,\sigma}(1)+\delta_{\sigma,\rho}(-1)}{\Gamma_\rho+\Gamma_\sigma}$
$NotChainSuccession(\rho, \sigma)$	$1 - \frac{\delta_{\rho,\sigma}(1)+\delta_{\sigma,\rho}(-1)}{\Gamma_\rho+\Gamma_\sigma}$

Table VIII: Functions computing constraints' Support

perspective for Relation Constraints, making Support related to single events rather than to the entire trace. As a clarifying example, the following trace can be considered: acbcacbaabac. MINERful would assign a Support equal to 0.80 to $Response(\mathsf{a}, \mathsf{b})$, because four "a" out of five respect the constraint. According to the approach of [Maggi et al. 2012], Support would be 0 instead.

Taking inspiration from [Maggi et al. 2012], we also associated to Support the Confidence Level (or Confidence for short) and Interest Factor metrics. Both estimate a level of relevance for a constraint, based on the assumption that the more the constrained activities appear in the log, the more their constraints should be taken into account.

Roughly speaking, if an activity a appears once in the whole log, made of hundreds of thousands of events, there is likely to be a glitch in the normal execution. Our adaptation of such metrics does not completely match the version of [Maggi et al. 2012], though. The reader can find the procedure computing Confidence and Interest Factor in Algorithm 4, discussed later in this Section. The overall DISCOVERCONSTRAINTS^{+} algorithm is presented in Algorithm 3. It consists of three procedure calls.

Algorithm 3 The pseudo-code of the DISCOVERCONSTRAINTS^{+} algorithm

```
Require: $\tau_s = 1.0$ , $\tau_c = 0.0$ , $\tau_i = 0.0$      # Optional user-defined thresholds
1: procedure DISCOVERCONSTRAINTS$^{+}$($\mathcal{KB}, \Sigma, |T|$)
2:     $\mathcal{B}^{+} \leftarrow$ CALCMETRICSFORCONSTRAINTS($\mathcal{KB}, \Sigma, |T|$)
3:     $\mathcal{B}^{+} \leftarrow$ CLEANOUTPUT($\mathcal{B}^{+}$)
4:     $\mathcal{B}^{+} \leftarrow$ FILTEROUTPUTBYTHRESHOLD($\mathcal{B}^{+}, \tau_s, \tau_c, \tau_i$)
5:     return $\mathcal{B}^{+}$
6: end procedure
```

Algorithm 4 The pseudo-code of the CALCMETRICSFORCONSTRAINTS procedure

```
1: procedure CALCMETRICSFORCONSTRAINTS($\mathcal{KB}, \Sigma, |T|$)
2:     $\mathcal{B}^{+} \leftarrow \emptyset$      # Inizialization of the extended bag of constraints
3:     for all $\rho \in \Sigma$ do
4:         if $\Gamma_\rho > 0$ then
5:             for all $b^x(\rho) \sqsubseteq \{ExistenceConstraint\}$ do
6:                 $s^x_b \leftarrow$ CALCSUPPORT($b^x(\rho)$)
7:                 $c^x_b \leftarrow s^x_b \cdot \left(1 - \frac{\gamma_\rho(0)}{|T|}\right)$
8:                 $i^x_b \leftarrow c^x_b \cdot \left(1 - \frac{\gamma_\rho(0)}{|T|}\right)$
9:                 $\mathcal{B}^{+} \leftarrow \mathcal{B}^{+} \cup \langle b^x(\rho), s^x_b, c^x_b, i^x_b \rangle$
10:            end for
11:            for all $\sigma \in \Sigma$ do
12:                for all $b^y(\rho,\sigma) \sqsubseteq \{RelationConstraint\}$ do
13:                    $s^y_b \leftarrow$ CALCSUPPORT($b^y(\rho,\sigma)$)
14:                    if $\neg(b^y(\rho,\sigma) \sqsubseteq \{Precedence, AlternatePrecedence, ChainPrecedence\})$ then
15:                        $c^y_b \leftarrow s^y_b \cdot \left(1 - \frac{\gamma_\rho(0)}{|T|}\right)$
16:                    else
17:                        $c^y_b \leftarrow s^y_b \cdot \left(1 - \frac{\gamma_\sigma(0)}{|T|}\right)$
18:                    end if
19:                    if $\neg(b^y(\rho,\sigma) \sqsubseteq \{NotCoExistence\})$ then
20:                        $i^y_b \leftarrow \left(1 - \frac{\gamma_\rho(0)}{|T|}\right) \cdot \left(1 - \frac{\gamma_\sigma(0)}{|T|}\right)$
21:                    else
22:                        $i^y_b \leftarrow \left(1 - \frac{\gamma_\rho(0)}{|T|}\right) \cdot \left(\frac{\gamma_\sigma(0)}{|T|}\right)$
23:                    end if
24:                    $\mathcal{B}^{+} \leftarrow \mathcal{B}^{+} \cup \langle b^y(\rho,\sigma), s^y_b, c^y_b, i^y_b \rangle$
25:                end for
26:            end for
27:        end if
28:    end for
29:    return $\mathcal{B}^{+}$
30: end procedure
```

The first one, CALCMETRICSFORCONSTRAINTS (Algorithm 4), populates the $\mathcal{B}^+$ bag. $\mathcal{B}^+$ is a collection of tuples $\langle b, s_b, c_b, i_b \rangle$, each associating to a constraint b the related

(1) Support, s_b,
(2) Confidence Level, c_b,
(3) Interest Factor, i_b.

For each constraint, let it be either *(i)* $b^x(\rho)$ if it belongs to the type of *ExistenceConstraint*s and constrains ρ (line 5), *(ii)* $b^y(\rho, \sigma)$ if it belongs to the type of *RelationConstraint*s and constrains ρ and σ (line 12). The $\sqsubseteq$ operator specifies whether a given constraint belongs to a type or a template.

The Support value is computed by the CALCSUPPORT procedure (resp., lines 6 and 13). Here we do not report its pseudo-code, because it applies the functions listed in Table VIII, according to the constraint template that $b^x(\rho)$ or $b^y(\rho, \sigma)$ belong to. In order to explain how the computations of Confidence Level and Interest Factor work, we need to introduce two notions:

— *activity-related log fraction*, namely the fraction of traces in the log where a given activity appears at least once;
— *activity-unrelated log fraction*, namely the fraction of traces where a given activity does not occur.

We consider Confidence Level, c_b, as the product of the constraint's Support and one constrained activity-related log fraction. The activity-related log fraction is chosen on the basis of the constraint type. $\mathit{ExistenceConstraint}$s constrain a single activity (ρ). Due to this, the second term of the product is the ρ-related log fraction (line 7). $\mathit{RelationConstraint}$s constrain two activities. Therefore, the second term of the product is:

(1) the *ρ-related log fraction* (line 15), for all constraints but $\mathit{Precedence}$ or $\mathit{Precedence}$-subsumed ones;
(2) the *σ-related log fraction*, otherwise (line 17).

$\mathit{Precedence}$ and $\mathit{Precedence}$-subsumed constraints have an impact on the structure of the string when σ occurs: if σ does not appear, no condition on the string is imposed (see the definition in Section 3). Therefore, Confidence for $\mathit{Precedence}(\rho, \sigma)$ is computed on the basis of the σ-related log fraction. The same holds for $\mathit{AlternatePrecedence}(\rho, \sigma)$ and $\mathit{ChainPrecedence}(\rho, \sigma)$. For all the other constraints, the ρ-related log fraction is taken into account.

The Interest Factor is computed on the basis of the product of Support and two activity-related log fractions. Indeed, it as either:

(1) Confidence Level multiplied by the constrained activity-related log fraction, if the constraint is an $\mathit{ExistenceConstraint}$, or
(2) the product of the activity-related log fractions, if the constraint is either a $\mathit{RelationConstraint}$ or a $\mathit{NegativeRelationConstraint}$, besides $\mathit{NotCoExistence}$,
(3) the multiplication of *activity-related log fraction* of one constrained activity and the *activity-unrelated log fraction* of the other, if the constraint is a $\mathit{NotCoExistence}$.

The reasons why the definition of Interest Factor changes according to the type of the constraint template are that:

(1) $\mathit{ExistenceConstraint}$s affect only one activity.
(2) $\mathit{RelationConstraint}$s and $\mathit{NegativeRelationConstraint}$s tie pairs of activities.
(3) The default formulation of Interest Factor for $\mathit{RelationConstraint}$s does not suit $\mathit{NegativeRelationConstraint}$s.

For what the last point is concerned, we recall here that $\mathit{NotCoExistence}(\rho, \sigma)$ holds when the occurrence of ρ implies the absence of σ in the trace. Therefore, the higher the number of traces where ρ is read, the lower the *activity-related log fraction* for the other activity. This is the reason why we consider the *activity-related log fraction* for ρ and the *activity-unrelated log fraction* for σ. The remaining $\mathit{NegativeRelationConstraint}$s do not impose the absence of σ in the trace, but only in the part that follows ρ (*NotSuccession*(ρ, σ)), or in the next character ($\mathit{NotChainSuccession}(\rho, \sigma)$): therefore, the common computation of the Interest Factor for $\mathit{RelationConstraint}$s can be considered valid.

Finally, we want to focus on line 4 of Algorithm 4. Given that *ex falso quod libet*, a character that was never read might be declared as supporting each constraint. However, it would be senseless to the mining purpose, as it would add no bit of information to the gathered knowledge. Because of this, the condition stated in line 4 avoids that missing activities in the log get involved in any inferred constraint.

In order to filter the irrelevant constraints out of the output, we make use of two methods, the aim of which is: *(i)* not to show trivially deducible constraints[3]; *(ii)* let the user decide thresholds of reliability and relevance, i.e., decide what are the least Support, Confidence and Interest Factor for a constraint to be considered valid and significant.

The first objective is fulfilled by Algorithm 5, cleanOutput, which requires no user intervention. The latter is obtained by Algorithm 6, filterOutputByThreshold, which expects a triple of parameters, (optionally) provided by the user (see line 0 in Algorithm 3):

(1) τ_s, the Support threshold;
(2) τ_c, the Confidence threshold;
(3) τ_i, the Interest Factor threshold.

In Algorithm 5, the block between lines 4 and 17 involves every $\mathit{RelationConstraint}$ (see the hierarchy in Figure 1). When a constraints b has a subsuming constraint p and its support (s_b) is less than p's one ($s_p > s_b$), it is removed. Otherwise, p and all its subsuming constraints (the "ancestors") are removed. By definition (see Table VIII), the Support of a subsumed constraint is in fact less than or equal to the subsuming's one. Therefore, the block from line 6 to line 9 raises along the hierarchy of Figure 1, from the current constraint to the subsuming one. Due to the monotonic increase of Support along the hierarchy, the loop from line 7 to line 9 stops when either *(i)* a subsuming constraint has a Support which is greater than the constraint under analysis, or *(ii)* no more "ancestors" along the hierarchy exist (i.e., the whole hierarchy share the same Support). In the first case, the current constraint is removed from bag $\mathcal{B}^+$. In the second case, its "parent" is deleted. Applying this selection to all constraints ensures that only one constraint along the hierarchy is kept in bag $\mathcal{B}^+$. Owing to this, the number of returned constraints is dramatically reduced.

$\mathit{MutualRelation}$ constraints are based on the conjunction of two $\mathit{RelationConstraint}$s (see Section 3). They are managed within the block from line 18 to line 25: see Table II to see how they are connected. If a *MutualRelation* constraint is known to have a Support which is not lower than *both* of the involved $\mathit{RelationConstraint}$s, the $\mathit{RelationConstraint}$s can be removed. Otherwise, no action is taken. From line 26 to line 33, a selection between each $\mathit{NegativeRelation}$ constraint and its negated is made: the constraint having the least Support is removed.

[3]e.g., it is enough to say that $\mathit{ChainPrecedence}(\mathsf{a}, \mathsf{b})$ holds, rather than explicitly return as valid constraints $\mathit{ChainPrecedence}(\mathsf{a}, \mathsf{b})$, $\mathit{AlternatePrecedence}(\mathsf{a}, \mathsf{b})$ and $\mathit{Precedence}(\mathsf{a}, \mathsf{b})$. In fact, the last two constraints are directly implied by the first: see Figure 1.

Algorithm 5 The pseudo-code of the CLEANOUTPUT procedure

```
1: procedure CLEANOUTPUT($\mathcal{B}^+$)
2:     $\overline{\mathcal{B}^+} := clone\ \mathcal{B}^+$
3:     for all $\langle b, s_b, c_b, i_b \rangle \in \overline{\mathcal{B}^+}$ do
4:         if $b \sqsubseteq RelationConstraint$ then
5:             if $hasParent(\overline{\mathcal{B}^+}, b)$ then
6:                 $p := b$
7:                 repeat
8:                     $\langle p, s_p, c_p, i_p \rangle := getParent(\overline{\mathcal{B}^+}, p)$
9:                 until $(s_p = s_b) \wedge hasParent(\overline{\mathcal{B}^+}, p)$
10:                if $s_p > s_b$ then
11:                    $\mathcal{B}^+ \leftarrow \mathcal{B}^+ \setminus \{\langle b, s_b, c_b, i_b \rangle\}$
12:                else
13:                    $\langle p, s_p, c_p, i_p \rangle := getParent(\overline{\mathcal{B}^+}, b)$
14:                    $\mathcal{B}^+ \leftarrow \mathcal{B}^+ \setminus \{\langle p, s_p, c_p, i_p \rangle\}$
15:                end if
16:            end if
17:        end if
18:        if $b \sqsubseteq MutualRelation$ then
19:            $\langle f, s_f, c_f, i_f \rangle := getForward(\overline{\mathcal{B}^+}, b)$
20:            $\langle r, s_r, c_r, i_r \rangle := getBackward(\overline{\mathcal{B}^+}, b)$
21:            if $\neg(s_f > s_b \vee s_r > s_b)$ then
22:                $\mathcal{B}^+ \leftarrow \mathcal{B}^+ \setminus \{\langle f, s_f, c_f, i_f \rangle\}$
23:                $\mathcal{B}^+ \leftarrow \mathcal{B}^+ \setminus \{\langle r, s_r, c_r, i_r \rangle\}$
24:            end if
25:        end if
26:        if $b \sqsubseteq NegativeRelation$ then
27:            $\langle n, s_n \rangle := getNegated(\overline{\mathcal{B}^+}, b)$
28:            if $s_n \leqslant s_b$ then
29:                $\mathcal{B}^+ \leftarrow \mathcal{B}^+ \setminus \{\langle n, s_n, c_n, i_n \rangle\}$
30:            else
31:                $\mathcal{B}^+ \leftarrow \mathcal{B}^+ \setminus \{\langle b, s_b, c_b, i_b \rangle\}$
32:            end if
33:        end if
34:    end for
35:    return $\mathcal{B}^+$
36: end procedure
```

The *hasParent*, *getParent*, *getForward*, *getBackward* and *getNegated* functions explore the subsumptions and the associations between constraints as described in Section 3. They all require as input *(i)* the $\mathcal{B}^+$ bag, and *(ii)* a constraint b. Their behavior is as follows.

— *hasParent* and *getParent* traverse the subsumption hierarchy (see Figure 1): the first returns the subsuming constraint, the second returns the subsumed constraint;
— *getForward* and *getBackward* return the tuples in $\mathcal{B}^+$ that are implied by the given *MutualRelation* constraint (see Table II): with *ChainSuccession*(ρ, σ), e.g., *getForward* returns *ChainResponse*(ρ, σ), whereas *getBackward* returns *ChainPrecedence*(ρ, σ);
— *getNegated* returns the *MutualRelation* tuple (like *CoExistence*) that is negated by the given *NegativeRelation* constraint (like *NotCoExistence*) (see Table III).

These functions do not depend on the interpretation of the MINERful interplay and the MINERful ownplay, but only on the semantics of constraints. The behavior of

Constraint	Symbol	*hasParent*	*getParent*	*getNegated*	*getForward*	*getBackward*
$RespondedExistence(\rho,\sigma)$	$\top_{\rho,\sigma}$	*false*	-	-	-	-
$Response(\rho,\sigma)$	$\top^{\rightarrow}_{\rho,\sigma}$	*true*	$\top_{\rho,\sigma}$	-	-	-
$AlternateResponse(\rho,\sigma)$	$\top^{\Rightarrow}_{\rho,\sigma}$	*true*	$\top^{\rightarrow}_{\rho,\sigma}$	-	-	-
$ChainResponse(\rho,\sigma)$	$\top^{\Rrightarrow}_{\rho,\sigma}$	*true*	$\top^{\Rightarrow}_{\rho,\sigma}$	-	-	-
$Precedence(\rho,\sigma)$	$\top^{\leftarrow}_{\rho,\sigma}$	*true*	$\top_{\sigma,\rho}$	-	-	-
$AlternatePrecedence(\rho,\sigma)$	$\top^{\Leftarrow}_{\rho,\sigma}$	*true*	$\top^{\leftarrow}_{\rho,\sigma}$	-	-	-
$ChainPrecedence(\rho,\sigma)$	$\top^{\Lleftarrow}_{\rho,\sigma}$	*true*	$\top^{\Leftarrow}_{\rho,\sigma}$	-	-	-
$CoExistence(\rho,\sigma)$	$\top^{\rho}_{\sigma}$	*false*	-	-	$\top_{\rho,\sigma}$	$\top_{\sigma,\rho}$
$Succession(\rho,\sigma)$	$\top^{\leftrightarrow}_{\rho,\sigma}$	*true*	$\top^{\rho}_{\sigma}$	-	$\top^{\rightarrow}_{\rho,\sigma}$	$\top^{\leftarrow}_{\sigma,\rho}$
$AlternateSuccession(\rho,\sigma)$	$\top^{\Leftrightarrow}_{\rho,\sigma}$	*true*	$\top^{\leftrightarrow}_{\rho,\sigma}$	-	$\top^{\Rightarrow}_{\rho,\sigma}$	$\top^{\Leftarrow}_{\sigma,\rho}$
$ChainSuccession(\rho,\sigma)$	$\top^{\Lleftarrow\!\!\!\Rrightarrow}_{\rho,\sigma}$	*true*	$\top^{\Leftrightarrow}_{\rho,\sigma}$	-	$\top^{\Rrightarrow}_{\rho,\sigma}$	$\top^{\Lleftarrow}_{\sigma,\rho}$
$NotChainSuccession(\rho,\sigma)$	$\top^{\not\Lleftarrow\!\!\!\Rrightarrow}_{\rho,\sigma}$	*false*	-	$\top^{\Lleftarrow\!\!\!\Rrightarrow}_{\rho,\sigma}$	-	-
$NotSuccession(\rho,\sigma)$	$\top^{\nleftrightarrow}_{\rho,\sigma}$	*true*	$\top^{\not\Lleftarrow\!\!\!\Rrightarrow}_{\rho,\sigma}$	$\top^{\leftrightarrow}_{\rho,\sigma}$	-	-
$NotCoExistence(\rho,\sigma)$	$\bot^{\rho}_{\sigma}$	*true*	$\top^{\nleftrightarrow}_{\rho,\sigma}$	$\top^{\rho}_{\sigma}$	-	-

Table IX: The functions navigating the constraints' hierarchy of subsumptions

hasParent, *getParent*, *getForward*, *getBackward* and *getNegated* functions is detailed in Table IX.

Algorithm 6 The pseudo-code of the FILTEROUTPUTBYTHRESHOLD procedure

```
1: procedure FILTEROUTPUTBYTHRESHOLD($\mathcal{B}^{+}, \tau_s, \tau_c, \tau_i$)
2:     for all $\langle b, s_b, c_b, i_b \rangle \in \mathcal{B}^{+}$ do
3:         if $s_b < \tau_s \ \vee \ s_c < \tau_c \ \vee \ s_i < \tau_i$ then
4:             $\mathcal{B}^{+} \leftarrow \mathcal{B}^{+} \setminus \{\langle b, s_b, c_b, i_b \rangle\}$
5:         end if
6:     end for
7:     return $\mathcal{B}^{+}$
8: end procedure
```

The FILTEROUTPUTBYTHRESHOLD procedure (Algorithm 6) finally filters out those constraints whose Support, Confidence and Interest Factor are below the user-defined thresholds (resp., τ_s, τ_c, τ_i).

The procedure for discovering the processes' constraints out of MINERful knowledge base, along with their Support, Confidence Level and Interest Factor is *(i)* quadratic in time w.r.t. the size of the alphabet, *(ii)* linear in time w.r.t the number of constraint templates, which is fixed and equal to 18 (thus, constant); therefore, the complexity is $O(|\Sigma|^2)$. The formal proof is presented in the on-line appendix.

Finally, we note that the complexity of the MINERful algorithm is $O(|T| \cdot |t_{max}|^2 \cdot |\Sigma_T|^2)$, being the algorithm *(i)* linear time w.r.t. the number of strings in log, *(ii)* quadratic time w.r.t. the size of strings in the log, *(iii)* quadratic time w.r.t. the size of the alphabet. The formal proof is provided in the on-line appendix.

5. EXPERIMENTS AND EVALUATION

In order to evaluate MINERful, we considered *(i)* its efficiency, in terms of computation time, and *(ii)* its efficacy, in terms of conformance of the discovered processes to reality.

We first produced synthetic logs, stemming from predefined workflow models. Then, we processed such logs. For every log, we measured the time it took to discover the originating workflow model, and analyzed its performance with respect to the input size. Thereafter, we compared the performance of MINERful to the performance of the current state-of-the-art tools, w.r.t. the synthetic logs and benchmarks taken from the BPI Challenge logs [van Dongen 2011; 2012].

In order to inspect the quality of results and validate the approach, we also verified the whole MAILOFMINE system on a real case study (Section 5.2). Data were extracted from the mailbox of an authors' colleague, known to be an expert in the area of the process to discover. As usual for artful processes, the process behind the analyzed email messages was not known a priori. Therefore, we could not apply an automated comparison between the resulting workflow model and the originating process, since no definition for the originating process was available at all. Thus, the expert was requested to analyze and assess the discovered workflow model by categorizing the mined constraints. We also assessed the level of the fitness of the discovered models to the analyzed event logs. Fitness is a metric for assessing to what extent the behavior seen in the event log is reported in the discovered model. In order to do so, we compared the fitness of the discovered control flow with respect to different levels of Support, used as a threshold for filtering out constraints. Finally, we compared the output models of MINERful to the Declare maps discovered by Declare Maps Miner [Maggi et al. 2013] (see Section 2).

5.1. Performance of MINERful

Figure 2 shows the experimental results that we obtained through the running of MINERful on synthetic logs, changing their input w.r.t. the number of traces in the log, their length, and the number of activities that the originating process comprised.

All the tests were conducted on a Sony VAIO VGN-FE11H (Intel Core Duo T2300 1.66 GHz, 2 MB L2 cache, with 2 GB of DDR2 RAM at 667 Mhz), having Ubuntu Linux 10.04 as the operating system and Java JRE v1.6. The synthetic logs' random traces were created by integrating our tool with Xeger,[4] a Java open-source library which generates strings according to the given regular expressions. We set up two different experiments. In Setup 1 (see Table X), synthetic logs were generated by simulating the execution of diversified versions of the example process, outlined in Section 3. Such process was modified by altering the number of constraints involved in the definition of the workflow. The constraints ranged from a minimum set of four ($Unique(\mathsf{n})$, $Participation(\mathsf{n})$, $End(\mathsf{n})$, $Succession(\mathsf{p}, \mathsf{n})$) to the maximum set of seven (including $Response(\mathsf{r}, \mathsf{p})$, $RespondedExistence(\mathsf{c}, \mathsf{p})$, $AlternatePrecedence(\mathsf{r}, \mathsf{c})$). For each altered process, different logs were created by varying: *(i)* alphabet size (i.e., activities appearing in the log), *(ii)* number of traces, and *(iii)* range of the number of events per trace (see Setup 1 in Table X). In order to consider the performances' degradation

[4]http://code.google.com/p/xeger/

Setup	Min. length	Max. length	Num. of traces	Alphabet size	Runs
1	$[0, 8]$	$[5, 20]$	$[10^2, 10^6]$	$[2, 5]$	29 000
2	$[0, 2]$	$[10, 25]$	$[10^3, 16 \cdot 10^3]$	$[10, 50]$	13 536

Table X: Performance experiments setup

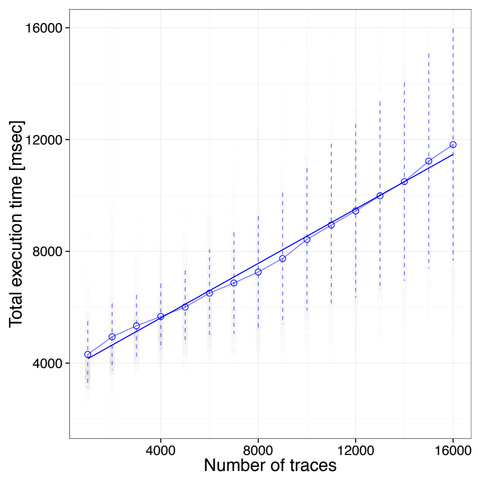
Source	Activities	Traces	Events processed		Total time	Engine
Synth. log, Setup 1	5	100 000	1 676 447	(avg. 16.764)	00:00:04	MINERful
					00:09:04	Dec. Miner
					12:00:00	D. M. Miner
					00:00:03	Unc. Miner
Synth. log, Setup 2	52	16 000	296 277	(avg. 18.517)	00:00:08	MINERful
					03:59:47	Dec. Miner
					00:09:27	D. M. Miner
					00:00:11	Unc. Miner
Financial log	24	13 087	262 200	(avg. 20.035)	00:00:03	MINERful
[van Dongen 2012]					00:42:29	Dec. Miner
					00:08:39	D. M. Miner
					00:00:04	Unc. Miner
Hospital log	624	1 143	150 291	(avg. 131.488)	00:04:20	MINERful
[van Dongen 2011]					02:52:03	Dec. Miner
					02:04:13	D. M. Miner
					00:15:10	Unc. Miner

Table XI: Performances of MINERful over synthetic and real cases. For all tools, the loading phase of logs is not included.

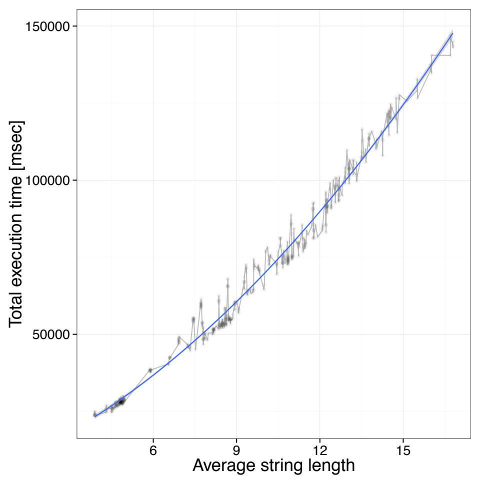
over increasing alphabets, we also executed a new experiment, according to Setup 2 (Table X).

Figure 2a shows the time taken by the algorithm to run, in comparison with the number of traces in the logs. The fitting curve is linear. The time taken for the algorithm to mine constraints, with respect to the average length of the traces is depicted in Figure 2b. There, the alphabet size is fixed and equal to 5. The dependency is quadratic. The same dependency holds between the computation time and the size of the alphabet of activities, as drawn in Figure 2c. Figure 2d separates the analysis of the time taken by the algorithm for its computation into its two main procedures: *(i)* the construction of MINERfulKB (Section 4.3) and *(ii)* the discovery of constraints, obtained by queries over MINERfulKB itself (Section 4.4). The graph gives an evidence of the fact that the second phase is faster than the first one, since it does not depend on the size of the log given in input (cf. Section 4.4 and Lemma B.2 in the online appendix). Figures 2a, 2b and 2c show by experimental evidence that the trend of computation time comply to the complexity analysis made on the algorithm (cf. Appendix B.3).

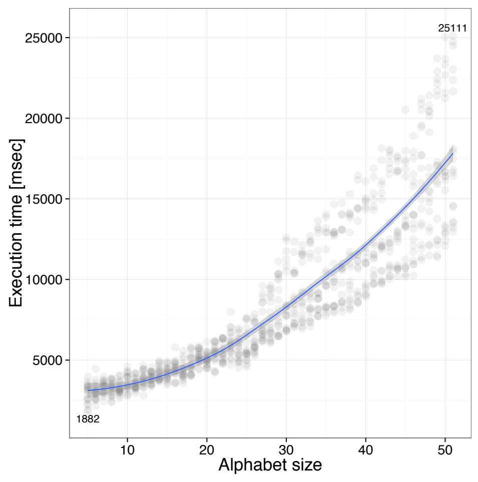
In order to test the efficiency of MINERful when dealing with real-life cases, we tested it with on well known benchmarks, taken from Business Process Intelligence Challenges (BPIC) (cf. "Dutch academic hospital log", BPIC'11 [van Dongen 2011] and



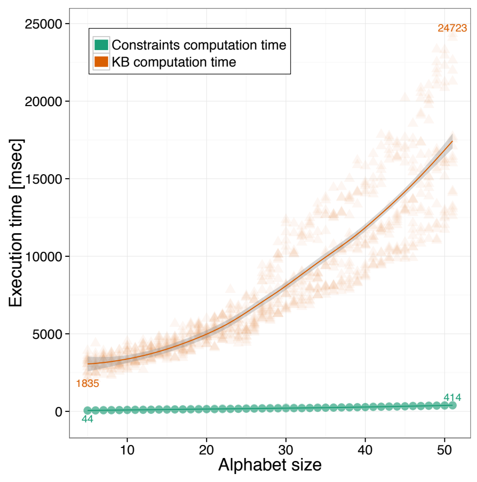
(a) Time needed for the execution, with respect to the number of traces (from Setup 2: only the tests where the size of the alphabet is greater than 25 are considered).



(b) Time needed for the execution, with respect to the trace length (from Setup 1: only the tests where the size of the alphabet is equal to 5 are plotted)



(c) Time needed for the execution, with respect to the size of the alphabet (from Setup 2)



(d) Time taken for the construction of MINERfulKB and the discovery of constraints by queries over MINERfulKB, with respect to the size of the alphabet (from Setup 2)

Fig. 2: Experimental results

"Dutch financial institute log", BPIC'12 [van Dongen 2012]). Table XI summarizes the execution of MINERful on both synthetic and benchmarking data. The same logs have been used with other state-of-the-art algorithms, namely Declare Miner [Maggi et al. 2011], Declare Maps Miner [Maggi et al. 2012], Unconstrained Declare [**?**] – see Section 2, in order to evaluate the efficiency of MINERful. We tuned each mining algorithm in order to check for the same constraints that MINERful is able to discover, establishing a threshold for the minimum Support at 85%. These tests were run on a Lenovo ThinkPad T430 (Intel i5-3320M CPU @ 2.60GHz, 16 GB RAM, 4096 MB of which was dedicated to the Java Virtual Machine). For Unconstrained Miner, we enabled the parallelization option, setting it up to 2 threads. In addition, we enabled the Super-Scalar mode (see [Westergaard and Stahl 2013] for further information). All timings refer to the mining phase only, regardless of the input reading. Therefore, the time needed for unpacking, opening and loading the XES log files [Günther and Verbeek 2012] is not included. Experimental results confirm that MINERful is among the fastest tools for discovering declarative processes.

5.2. A real case study

In order to evaluate MINERful in terms of its efficacy, we verified the quality of control flows mined from real data. The input data pertained to the artful process of managing European research projects. To this extent, we made use of the MAILOFMINE system [Di Ciccio and Mecella 2013a] in order to extract a log out of 6 mailbox IMAP folders containing email messages. The email messages concerned the management of 5 different European research projects. Such folders belonged to a domain expert. Together with him, we analyzed the discovered process. More than 350 constraints were found to hold true in the log, with a Support higher than 80% (i.e., over the threshold we set, in agreement with the user). However, the simplification techniques adopted by MINERful (see Section 4.4) reduced the number of constraints shown to the user, finally returning a set of ca. 200 constraints.

In order to assess the validity of the mined process, we checked every constraint with the expert. As usual in the context of artful processes, the workflow was not known a priori. However, the expert could classify as right or wrong a guessed constraint, on the basis of his experience. As a matter of fact, such situation of partial knowledge reproduces a real case, where the artful process had not ever been formalized before, although the main actor is aware of the best practices adopted. For each constraint in the list, we asked the expert whether it was either: *(i)* right, i.e., it made sense with respect to his experience; *(ii)* noticeably right, i.e., it not only made sense but also suggested some surprising mechanisms in the workflow; *(iii)* wrong, i.e., not necessarily corresponding to reality; *(iv)* utterly wrong, i.e., not corresponding to reality, unreasonable. The last level was assigned to very few constraints (7 out of 173), a half of how many were considered noticeably right (14) – see Figure 3. For the rest of the analysis, we categorize right and noticeably right constraints as good guesses, while wrong and utterly wrong constraints are considered bad guesses. Being the classification based on the judgment of an expert, we will name them resp. as *perceived true positives* ($TP^{\mathcal{P}}$) and *perceived false positives* ($FP^{\mathcal{P}}$). As said, the original process was unknown. Therefore, there was no viable way to identify those constraints that were known to hold true, but are ignored by the discovery algorithm. For the same reason, it was not feasible to identify those constraints that were known not to hold true and were not part of the discovered process. Thereby, we have no notion of *perceived false negatives* or *perceived true negatives* in this context.

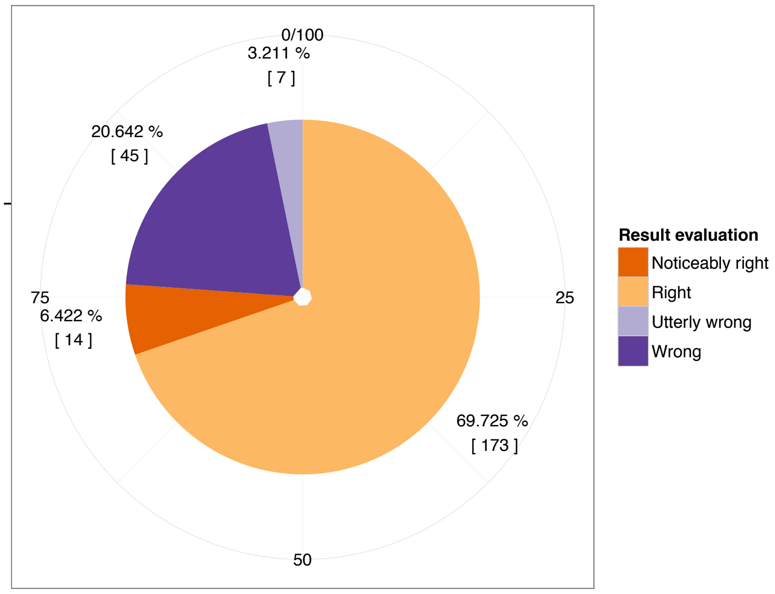


Fig. 3: Evaluation of MINERful: appropriateness of the discovered results in the case study

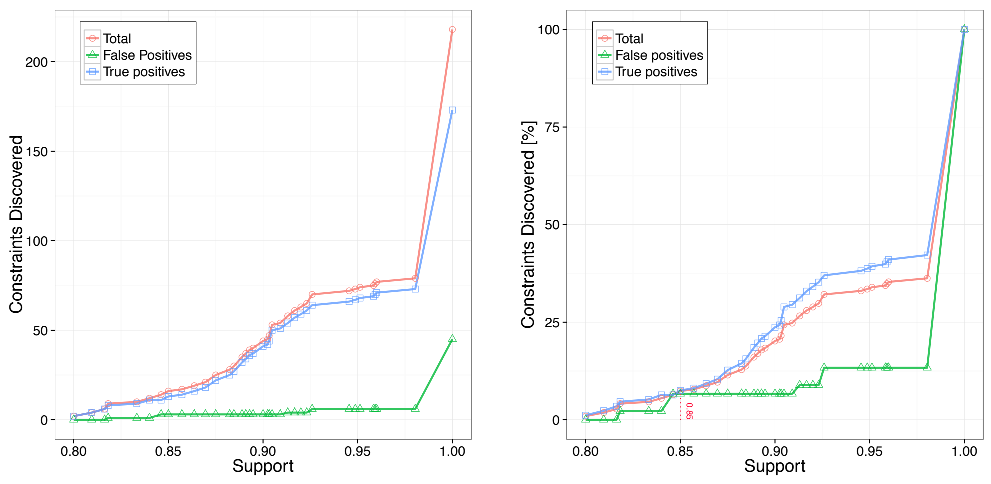
Building upon the definition of Precision in Information Retrieval,[5] we consider the *perceived precision* as a quality measure for the discovered process from the user's perspective. It specifies the fraction of constraints that are relevant in the discovered process. It is formulated as follows:

$$PerceivedPrecision = \frac{TP^{\mathcal{P}}}{TP^{\mathcal{P}} + FP^{\mathcal{P}}}$$

The algorithm was proved to obtain a $PerceivedPrecision$ of 0.794 over the real case study. In other words, about 80% of the inferred constraints were compliant to a realistic model of the process. Figure 3 summarizes the results of this real case study evaluation.

For the sake of readability, we will omit the adjective "perceived" in the remainder of this Section. Figure 4a shows the trend of true positives, false positives and all constraints found, with respect to their Support. The quantities on the ordinates are cumulative, i.e., they represent the sum of the values which are gained up to the current value on the abscissae. The curves show how, as Support increases, the distance between the cumulated false positives and the true positives rises as Support increases. On Figure 4b, a line puts in evidence where the relative percentage of confirmed constraints overtakes the wrong, i.e., a "break-point" after which the rate of hits, in terms of accepted guesses, is higher than the rate of misses, in terms of wrong guesses. Such break-point corresponds to a Support value of 0.85, i.e., 5% higher than the threshold established a priori. However, the number of true positives below that

[5]"Precision" in the context of Process Model Discovery [Buijs et al. 2012] differs from the classical definition of Information Retrieval [Manning et al. 2008]. In the first case, it "quantifies the fraction of the behavior allowed by the model which is not seen in the event log". In Information Retrieval, it "is the fraction of retrieved documents that are relevant". The notion of "perceived precision" provided here bases on the latter.



(a) The trend of the quality of the cumulative sum of constraints discovered, w.r.t. the assigned Support

(b) The trend of the quality of the cumulative sum of constraints discovered, scaled by their total amount, w.r.t. the assigned Support

Fig. 4: Evaluation of MINERful on a case study: trend of the quality of the process w.r.t. Support of constraints

soil amounts to less than 10%. On the other hand, the same graph depicts that more than 85% of errors are given a Support of 100%.

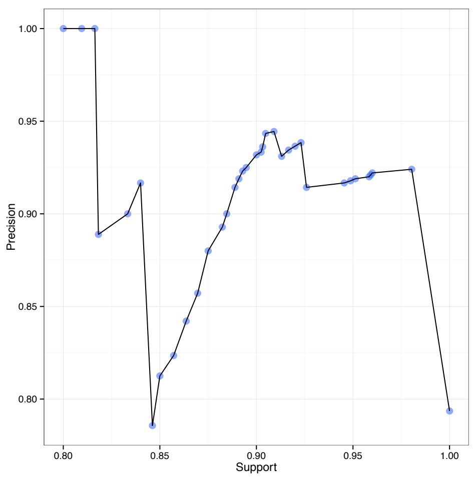
The trend of Precision is drawn on Figure 5, with respect to *(i)* Support (Figure 5a), *(ii)* Confidence Level (Figure 5b) and *(iii)* Interest Factor (Figure 5c). Support (Figure 5a) is proven not to be a good measure for filtering misinterpretations away. The Precision indeed tends to rapidly decrease near to the support level of 1.0. Perceived precision's curve tends to grow with Confidence (Figure 5b) and Interest Factor (Figure 5c). Hence, relevance metrics contribute to the improvement of quality of results, from the user's perspective.

Next Section will investigate the role of Support, Confidence Level and Interest Factor in enhancing the conformance of the discovered control flow with respect to the log. The reader interested in further details on the case study, and in particular to the Finite State Automaton (FSA) describing the discovered process, can refer to [Di Ciccio and Mecella 2013a].

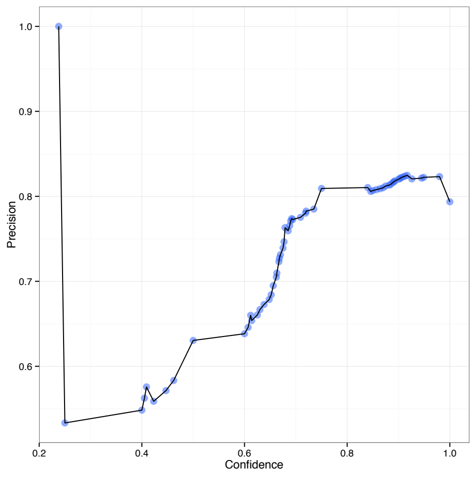
5.3. Fitness-based Analysis of Discovered Models' Conformance

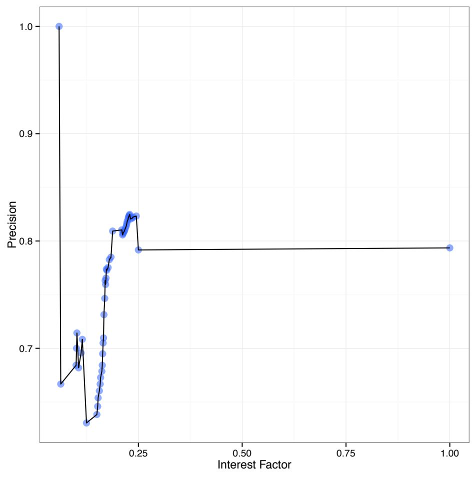
A model with good fitness allows for most of the behavior seen in the event log. A model has a perfect fitness if all traces can be replayed by the model from the beginning to the end. Fitness is one of the four quality dimensions (Fitness, Precision,[5] Generalization, Simplicity [Buijs et al. 2012]) for checking the conformance of event logs with respect to process models. [de Leoni et al. 2014] describes an analytical approach to measure them for declarative process models. Its implementation is available as a ProM plug-in: Declare Replayer[6] [de Leoni et al. 2012]. In order to take advantage of the ProM plug-in, we created a module for translating the output of MINERful into the ProM Declare XML format. At the time of writing, though, Declare Replayer measures only the level

[6]Available at https://svn.win.tue.nl/repos/prom/Packages/DeclareChecker/Trunk. The authors want to thank Fabrizio M. Maggi for his useful insight and advice.



(a) Precision, w.r.t. the assigned Support of discovered constraints





(b) Precision, w.r.t. the assigned Confidence Level of discovered constraints

(c) Precision, w.r.t. the assigned Interest Factor of discovered constraints

Fig. 5: Evaluation of MINERful on a case study: trend of Precision w.r.t. Support, Confidence Level and Interest Factor of discovered constraints

of Fitness for an input process. Due to this, the work presented in this Section considers only that metric. Nevertheless, it sets out a preliminary study on the conformance of mined declarative processes to the input logs, based on the systematic measurement of established metrics.

Two sets of discovered processes were analyzed. The first set was mined from a real event log extracted from email messages (see Section 5.2). The second one was mined from a synthetic log, comprising 8 activities, 250 traces and 5205 events, compliant to

	Support threshold									
	0.910	0.920	0.930	0.940	0.950	0.960	0.970	0.980	0.990	1.000
Email archives	0.822	0.904	0.965	0.965	0.967	0.991	0.995	0.995	1.000	1.000
Synth. log	0.926	0.947	0.964	0.979	0.988	0.991	0.996	0.999	0.999	1.000

Table XII: Fitness of the discovered processes, w.r.t. the threshold for Support

the example process described in Section 3. On the basis of the mined processes, we conducted two analyses:

(A) the first aimed at evaluating the Fitness of discovered processes, with respect to the threshold for Support;
(B) the second compared the control flows mined by MINERful with the ones of Declare Maps Miner [Maggi et al. 2013].

Declare Maps Miner is considered as the most suitable term of paragon, since it is the only technique so far in the literature that prunes out irrelevant constraints (see Section 2). Furthermore, MINERful and Declare Maps Miner share a common set of metrics, i.e., Support for the reliability of constraints, and Confidence and Interest Factor for their relevance. Although the concepts that such metrics are based upon are the same, the ways to calculate their values differ (see Section 4.4). For instance, MINERful computes Support on a per-event basis, whereas Declare Maps Miner adopts a per-trace measurement. Therefore, it was also considered of interest the different behavior of the algorithms with respect to equatable tunings of thresholds.

For what analysis (A) is concerned, we have run MINERful on both logs ten times. For each run, a different Support threshold was set, ranging from 0.91 to 1.00 with a step of 0.01. As shown in Table XII, the level of Fitness grows monotonically: the lowest level is obtained when the threshold is minimum, whereas Fitness amounts to 100% when the threshold is set to 1.0. Such result confirms that *(i)* as expected, the Fitness of the discovered model w.r.t. the analyzed event log tends to increase as the Support threshold is raised, and *(ii)* a Support threshold of 1.0 guarantees that only those constraints that are proven to hold true for every trace of the event log are presented to the user. Therefore, we have experimental evidence that MINERful returns discovered control flows which comply with the input. The Support threshold plays a crucial role in determining to what extent the mined process fits to the log. Therefore, lower levels for Support thresholds allow for noise in logs, whereas the maximum level is eligible when the log is deemed as unaffected by errors. A preliminary study on the effect that noise in logs has on mined declarative processes can be found in [Di Ciccio and Mecella 2013b].

As previously stated, we conducted a comparative analysis (B) on MINERful and the state-of-the-art algorithm, Declare Maps Miner. To this extent, we used the aforementioned logs as input for both miners.[7] We have sampled four levels for Support thresholds ranging between 0.91 and 1.00, namely

(1) the lower end (0.91),
(2) the rounded-up value for the first third of the range (0.94),
(3) the rounded-up value for the second third of the range (0.97), and
(4) the upper end (1.00).

[7]Declare Maps Miner requires the user to specify which constraint templates it has to look for within the log. Therefore, we set it up to search for the templates that MINERful is able to identify, on one hand. On the other hand, we excluded the *End* constraints from the output of MINERful.

For each level, we set it as the Support threshold and let both tools run. In addition, we have considered the relevance parameters that MINERful and Declare Maps Miner have in common, i.e., Confidence and Interest Factor, in order to investigate their influence on results. To this extent, we have set four different thresholds for Confidence and Interest Factor. Those values were chosen according to the same rationale adopted for Support, namely:

(1) the lower end (▯),
(2) the rounded-up value for the first third of the range (▮),
(3) the rounded-up value for the second third of the range (▮▮), and
(4) the upper end (▮▮▮).

The range was fixed on the basis of minimum and maximum possible values of Confidence (resp. Interest Factor) that actively altered the resulting discovered control flow.[8] Thereafter, we have launched MINERful and Declare Maps Miner for each combination consisting of *(i)* a Support threshold, and *(ii)* a Confidence (resp. Interest Factor) threshold, within the list of levels described above. For the sake of readability, Tables XIII and XIV present the results referring only to Support thresholds 0.91 and 0.97. The comprehensive list is transcribed in the on-line appendix.

For both MINERful and Declare Maps Miner, we counted the number of constraints in the discovered process and computed the average number of constraints per activity. Table XIII shows that Declare Maps Miner tends to shrink by default the number of returned constraints more than MINERful. However, MINERful is more sensitive to the variation of Confidence and Interest Factor. This is probably due to the different calculation that the techniques adopt for assessing the relevance metrics. Taking advantage of Declare Checker, we evaluated the Fitness of discovered control flows. The mined processes achieve close values for both counterparts. Declare Maps Miner accounts to slightly more fitting models, even without raising thresholds for Confidence and Interest Factor. The same data show that not only Support but also Confidence and Interest Factor lead MINERful-discovered processes to the maximum Fitness, when their threshold is increased.

In order to assess the similarity of mined processes, we considered them as sets of constraints, following the rationale of Section 4.4. Therefore, we could consider well-established similarity measures to this end. In particular, we considered Jaccard, Sørensen-Dice and Meet/min measures [Deza and Deza 2006]. Naming MINERful and Declare Maps Miner's discovered sets as, resp., $\mathcal{B}_M$ and $\mathcal{B}_D$, we have that:

— Jaccard coefficient is defined as $\frac{|\mathcal{B}_M \cap \mathcal{B}_D|}{|\mathcal{B}_M \cup \mathcal{B}_D|}$,

— Sørensen-Dice coefficient is defined as $\frac{2 \cdot |\mathcal{B}_M \cap \mathcal{B}_D|}{|\mathcal{B}_M| + |\mathcal{B}_D|}$,

— Meet/min coefficient is defined as $\frac{|\mathcal{B}_M \cap \mathcal{B}_D|}{\min\{|\mathcal{B}_M|, |\mathcal{B}_D|\}}$.

According to Jaccard and Sørensen-Dice coefficients, the pairs of processes turn out not to be matching. However, the Meet/min coefficient suggests that their difference resides in the respective sizes, rather than in their content. Indeed, they are mostly overlapping, as shown by the high values reached by the Meet/min coefficient. We

[8]In MINERful, the range is fixed from 0.0 to 1.0 for both Confidence and Interest Factor. For Declare Maps Miner, Confidence can vary from 0 to 100, whilst Interest Factor possibly ranges from 0 to a maximum value that changes according to the process, possibly exceeding 100. However, neither Confidence nor Interest Factor affect the result if their respective thresholds are lower than Support threshold in Declare Maps Miner. Therefore, their ranges have been rescaled from the given Support threshold to the maximum allowed. See the on-line appendix for further details.

recall here that the upper bound for Meet/min, 100%, indicates that one set entirely contains the other.

Table XIV shows similar characteristics. However, Declare Maps Miner returns either one of two control flows, depending on the thresholds' setup: one consists of 44 constraints, the other comprises 5 constraints. The Fitness of both is equal to 1.0 though. MINERful, instead, varies the mined process (and their Fitness) according to the parameters tuning. This is probably due to the trace-based assessment of metrics of the former, as opposed to the event-based calculation of the latter.

As a concluding remark, Declare Maps Miner turns out to return more fitting and concise process models by default, i.e., just by setting a proper Support threshold. MINERful is more sensitive to the parameter tuning in this regard. However, it outperforms Declare Maps Miner in terms of time of computation. Therefore, Declare Maps Miner seems more appropriate for an off-line execution, whilst MINERful is more suited for interactive sessions, where different setups can be tried in order to find the best combination. Such characteristic meets the requirement described in the end of Section 1: logs extracted out of semi-structured texts can be severely affected by noise. Therefore, the mining phase may need to be repeated several times with different combinations of parameters, so as to find the highest-quality result.

			MINERful			**Declare M. Miner**			**Similarity**		
Support %	Confidence	Interest Factor	Constraints	Cn's per activity	Fitness	Constraints	Cn's per activity	Fitness	Meet/min %	Jaccard %	Sørensen-Dice %
91	|	|	83	10.375	0.926	24	3.000	0.992	95.833	27.381	42.991
91	|	I	83	10.375	0.926	23	2.875	0.996	86.957	23.256	37.736
91	|	II	50	6.250	0.977	8	1.000	0.996	87.500	13.725	24.138
91	|	III	7	0.875	1.000	8	1.000	0.996	42.857	25.000	40.000
91	I	|	83	10.375	0.926	23	2.875	0.994	95.652	26.190	41.509
91	II	|	50	6.250	0.977	21	2.625	0.996	85.714	33.962	50.704
91	III	|	7	0.875	1.000	19	2.375	0.996	85.714	30.000	46.154
97	|	|	40	5.000	0.996	17	2.125	0.999	94.118	39.024	56.140
97	|	I	40	5.000	0.996	6	0.750	0.999	83.333	12.195	21.739
97	|	II	32	4.000	0.996	6	0.750	0.999	83.333	15.152	26.316
97	|	III	7	0.875	1.000	6	0.750	0.999	50.000	30.000	46.154
97	I	|	40	5.000	0.996	15	1.875	0.999	100.000	37.500	54.545
97	II	|	32	4.000	0.996	15	1.875	0.999	86.667	38.235	55.319
97	III	|	7	0.875	1.000	15	1.875	0.999	85.714	37.500	54.545

Table XIII: Discovered constraints and Fitness of processes mined out of a synthetic log by resp. MINERful and Declare Maps Miner.

6. CONCLUSIONS

In this paper, we have presented a novel technique, named MINERful, for discovering control-flow declarative constraints in artful processes. The peculiarities of the approach are: *(i)* modularity, i.e., it is based on two steps, where the first builds a knowl-

			MINERful			Declare Maps Miner			Similarity		
Support %	Confidence	Interest Factor	Constraints	Cn's per activity	Fitness	Constraints	Cn's per activity	Fitness	Meet/min %	Jaccard %	Sørensen-Dice %
91	|	|	188	14.462	0.822	44	3.385	1.000	100.000	23.404	37.931
91	|	▮	113	8.692	0.923	5	0.385	1.000	40.000	1.724	3.390
91	|	▮▮	41	3.154	0.967	5	0.385	1.000	40.000	4.545	8.696
91	|	▮▮▮	5	0.385	1.000	5	0.385	1.000	40.000	25.000	40.000
91	▮	|	116	8.923	0.859	44	3.385	1.000	68.182	23.077	37.500
91	▮▮	|	97	7.462	0.928	44	3.385	1.000	59.091	22.609	36.879
91	▮▮▮	|	7	0.538	1.000	44	3.385	1.000	57.143	8.511	15.686
97	|	|	163	12.538	0.995	44	3.385	1.000	100.000	26.994	42.512
97	|	▮	89	6.846	0.995	5	0.385	1.000	40.000	2.174	4.255
97	|	▮▮	29	2.231	0.995	5	0.385	1.000	40.000	6.250	11.765
97	|	▮▮▮	5	0.385	1.000	5	0.385	1.000	40.000	25.000	40.000
97	▮	|	92	7.077	0.995	44	3.385	1.000	68.182	28.302	44.118
97	▮▮	|	75	5.769	0.995	44	3.385	1.000	59.091	27.957	43.697
97	▮▮▮	|	7	0.538	1.000	44	3.385	1.000	57.143	8.511	15.686

Table XIV: Discovered constraints and Fitness of processes mined out of the email-based log by resp. MINERful and Declare Maps Miner.

edge base for the second, verifying the constraints as the results of specific queries; *(ii)* probabilistic approach to the inference of constraints; *(iii)* capability of eliminating the redundancy of subsumed constraints. The technique is part of a comprehensive approach, named MAILOFMINE [Di Ciccio et al. 2012; Di Ciccio and Mecella 2013a], for mining artful processes out of email archives. It has been designed to be reasonably fast. The main reason for keeping its computation time low also resides in its usage inside MAILOFMINE. Since it is run over logs which are built on top of uncertain information (the semi-structured text of email messages), the output strongly depends on the reliability of the log. The log can indeed contain several outliers or misinterpreted information. Thus, if users or experts considered the resulting process model as poorly compliant to the reality, the mining phase might need to be repeated over refined logs, so to improve the overall quality of the result. A slow algorithm may undermine such an iterative approach, making it impractical.

In the paper, the MINERful technique has been entirely described and extensively analyzed in its performances, both from a theoretical and an experimental perspective. Moreover, a user-driven evaluation of the quality of the achievable results, conducted on a real case study performed with the whole MAILOFMINE, has been reported. In order to discuss the fitness of the workflow-models mined by MINERful, a study has been performed, comparing our techniques with the state-of-the-art tool Declare Maps Miner. To the best of our knowledge, this is one of the first studies on the conformance of mined declarative processes to the input logs, based on the systematic measurement of established metrics.

In future investigations, we aim at addressing the complementary challenging issue of inferring the data flow in artful processes. Indeed, in order to comprehend (and apply) an artful process, a knowledge worker needs to understand the data flow in conjunction with the control flow. For instance, it is clearly beneficial to know that activity “submit report” is followed by “review report”. However, the usefulness would be greatly improved if the user could also understand what factors in a report can cause an “approval” vis-a-vis a “rejection”. In this regard, a novel approach has been recently proposed in [Bose et al. 2013]. In their work, Bose et al. focus not only on the control-flow perspective, but also on attributes of events in the log. Specifically, the possible correlations between attribute values are searched, in order to refine the discovery of related activities that are involved in the constraints of a Declare model. Our future work will address how the data flow can be mined and summarized along with the process flow.

REFERENCES

Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. 1998. Mining process models from workflow logs. In *Advances in Database Technology – EDBT'98*, Hans-Jrg Schek, Gustavo Alonso, Felix Saltor, and Isidro Ramos (Eds.). Lecture Notes in Computer Science, Vol. 1377. Springer Berlin / Heidelberg, 467–483. http://dx.doi.org/10.1007/BFb0101003 10.1007/BFb0101003.

Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In *VLDB*, Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo (Eds.). Morgan Kaufmann, 487–499. http://www.vldb.org/conf/1994/P487.PDF

Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. 2008. Verifiable agent interaction in abductive logic programming: The SCIFF framework. *ACM Trans. Comput. Log.* 9, 4 (August 2008), 29:1–29:43. DOI:http://dx.doi.org/10.1145/1380572.1380578

Gustavo Alonso, Peter Dadam, and Michael Rosemann (Eds.). 2007. *Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings*. Lecture Notes in Computer Science, Vol. 4714. Springer.

Elena Bellodi, Fabrizio Riguzzi, and Evelina Lamma. 2010a. Probabilistic Declarative Process Mining. In *KSEM (Lecture Notes in Computer Science)*, Yaxin Bi and Mary-Anne Williams (Eds.), Vol. 6291. Springer, 292–303. DOI:http://dx.doi.org/10.1007/978-3-642-15280-1_28

Elena Bellodi, Fabrizio Riguzzi, and Evelina Lamma. 2010b. Probabilistic Logic-Based Process Mining. In *CILC (CEUR Workshop Proceedings)*, Wolfgang Faber and Nicola Leone (Eds.), Vol. 598. CEUR-WS.org. http://ceur-ws.org/Vol-598/paper17.pdf

R. P. Jagadeesh Chandra Bose, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. 2013. Enhancing Declare Maps Based on Event Correlations. In *BPM (Lecture Notes in Computer Science)*, Florian Daniel, Jianmin Wang, and Barbara Weber (Eds.), Vol. 8094. Springer, 97–112. DOI:http://dx.doi.org/10.1007/978-3-642-40176-3_9

Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. 2012. On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In *CoopIS (Lecture Notes in Computer Science)*, Robert Meersman, Hervè Panetto, Tharam S. Dillon, Stefanie Rinderle-Ma, Peter Dadam, Xiaofang Zhou, Siani Pearson, Alois Ferscha, Sonia Bergamaschi, and Isabel F. Cruz (Eds.), Vol. 7565. Springer. On the Move to Meaningful Internet Systems (OTM 2012) Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012.

Federico Chesani, Evelina Lamma, Paola Mello, Marco Montali, Fabrizio Riguzzi, and Sergio Storari. 2009. Exploiting Inductive Logic Programming Techniques for Declarative Process Mining. *T. Petri Nets and Other Models of Concurrency* 2 (2009), 278–295. http://dx.doi.org/10.1007/978-3-642-00899-3_16

Edmund M. Clarke, Orna Grumberg, and Doron Peled. 2001. *Model Checking*. MIT Press. I–XIV, 1–314 pages.

Jonathan E. Cook and Alexander L. Wolf. 1998. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.* 7, 3 (1998), 215–249. DOI:http://dx.doi.org/10.1145/287000.287001

Jordi Cortadella, Michael Kishinevsky, Luciano Lavagno, and Alexandre Yakovlev. 1998. Deriving Petri nets from finite transition systems. *Computers, IEEE Transactions on* 47, 8 (aug. 1998), 859 –882. DOI:http://dx.doi.org/10.1109/12.707587

Thomas H. Davenport, Sirkka L. Jarvenpaa, and Michael C. Beers. 1996. Improving Knowledge Work Processes. *Sloan Management Review* 37, 4 (1996), 53–65. http://sloanreview.mit.edu/the-magazine/articles/1996/summer/3744/improving-knowledge-work-processes

Giuseppe De Giacomo and Moshe Y. Vardi. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*, Francesca Rossi (Ed.). IJCAI/AAAI. http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997

Massimiliano de Leoni, Fabrizio Maria Maggi, and Wil M.P. van der Aalst. 2014. An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Information Systems* (2014). DOI:http://dx.doi.org/10.1016/j.is.2013.12.005

Massimiliano de Leoni, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. 2012. Aligning Event Logs and Declarative Process Models for Conformance Checking. In *BPM (Lecture Notes in Computer Science)*, Alistair P. Barros, Avigdor Gal, and Ekkart Kindler (Eds.), Vol. 7481. Springer, 82–97. DOI:http://dx.doi.org/10.1007/978-3-642-32885-5_6

Ana Karla A. de Medeiros, A. J. M. M. Weijters, and Wil M. P. van der Aalst. 2007. Genetic process mining: an experimental evaluation. *Data Min. Knowl. Discov.* 14, 2 (2007), 245–304. DOI:http://dx.doi.org/10.1007/s10618-006-0061-7

Jörg Desel and Wolfgang Reisig. 1996. The synthesis problem of Petri nets. *Acta Informatica* 33 (1996), 297–315. Issue 4. http://dx.doi.org/10.1007/s002360050046 10.1007/s002360050046.

Elena Deza and Michel Deza. 2006. *Dictionary of Distances*. North-Holland. I–XV, 1–391 pages.

Claudio Di Ciccio, Andrea Marrella, and Alessandro Russo. 2012. Knowledge-intensive Processes: An Overview of Contemporary Approaches. In *1st International Workshop on Knowledge-intensive Business Processes, KiBP 2012, Rome, Italy, June 15, 2012*, Arthur H.M. ter Hofstede, Massimo Mecella, Sebastian Sardina, and Andrea Marrella (Eds.), Vol. 861. CEUR Workshop Proceedings, 33–47. http://ceur-ws.org/Vol-861/KiBP2012_paper_2.pdf

Claudio Di Ciccio, Andrea Marrella, and Alessandro Russo. 2014. Knowledge-Intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches. *Journal on Data Semantics* (2014). DOI:http://dx.doi.org/10.1007/s13740-014-0038-4

Claudio Di Ciccio and Massimo Mecella. 2012a. *MINERful, a Mining Algorithm for Declarative Process Constraints in MailOfMine*. Technical Report. Dipartimento di Ingegneria Informatica, Automatica e Gestionale "Antonio Ruberti" – SAPIENZA, Università di Roma. http://ojs.uniroma1.it/index.php/DIS_TechnicalReports/issue/view/416

Claudio Di Ciccio and Massimo Mecella. 2012b. Mining Constraints for Artful Processes. In *15th International Conference on Business Information Systems, BIS 2012, Vilnius, Lithuania, May 21-23, 2012 (Lecture Notes in Business Information Processing)*, Witold Abramowicz, Dalia Kriksciuniene, and Virgilijus Sakalauskas (Eds.), Vol. 117. Springer, 11–23. DOI:http://dx.doi.org/10.1007/978-3-642-30359-3_2

Claudio Di Ciccio and Massimo Mecella. 2013a. Mining Artful Processes from Knowledge Workers' Emails. *IEEE Internet Computing* 17, 5 (2013), 10–20. DOI:http://dx.doi.org/10.1109/MIC.2013.60

Claudio Di Ciccio and Massimo Mecella. 2013b. Studies on the Discovery of Declarative Control Flows from Error-prone Data. In *3rd International Symposium on Data-Driven Process Discovery and Analysis, SIMPDA 2013, Riva del Garda, Italy, August 30, 2013 (CEUR Workshop Proceedings)*, Rafael Accorsi, Paolo Ceravolo, and Philippe Cudre-Mauroux (Eds.), Vol. 1027. 31–45. http://ceur-ws.org/Vol-1027/paper3.pdf

Claudio Di Ciccio and Massimo Mecella. 2013c. A Two-Step Fast Algorithm for the Automated Discovery of Declarative Workflows. In *4th IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2013, Singapore, April 16-19, 2013*. IEEE, 135–142. DOI:http://dx.doi.org/10.1109/CIDM.2013.6597228

Claudio Di Ciccio, Massimo Mecella, Monica Scannapieco, Diego Zardetto, and Tiziana Catarci. 2012. MailOfMine – Analyzing Mail Messages for Mining Artful Collaborative Processes. In *Data-Driven Process Discovery and Analysis*, Karl Aberer, Ernesto Damiani, and Tharam Dillon (Eds.). Lecture Notes in Business Information Processing, Vol. 116. Springer, 55–81. DOI:http://dx.doi.org/10.1007/978-3-642-34044-4_4

Volker Diekert and Paul Gastin. 2008. First-order Definable Languages. In *Logic and Automata (Texts in Logic and Games)*, Jörg Flum, Erich Grädel, and Thomas Wilke (Eds.), Vol. 2. Amsterdam University Press, 261–306.

Marie-Christine Fauvet and Boudewijn F. van Dongen (Eds.). 2013. *Proceedings of the BPM Demo sessions 2013, Beijing, China, August 26-30, 2013*. CEUR Workshop Proceedings, Vol. 1021. CEUR-WS.org. http://ceur-ws.org/Vol-1021

Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. 1980. On the Temporal Analysis of Fairness. In *Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '80)*. ACM, New York, NY, USA, 163–173. DOI:http://dx.doi.org/10.1145/567446.567462

Norbert Gronau and Edzard Weber. 2004. Management of Knowledge Intensive Business Processes. In *Business Process Management (Lecture Notes in Computer Science)*, Jörg Desel, Barbara Pernici, and Mathias Weske (Eds.), Vol. 3080. Springer, 163–178. DOI:http://dx.doi.org/10.1007/978-3-540-25970-1_11

Christian W. Günther and Eric Verbeek. 2012. XES Standard Definition. (10 2012). http://www.xes-standard.org/_media/xes/xesstandarddefinition-1.4.pdf

Charles Hill, Robert Yates, Carol Jones, and Sandra L. Kogan. 2006. Beyond predictable workflows: Enhancing productivity in artful business processes. *IBM Systems Journal* 45, 4 (2006), 663–682. http://dx.doi.org/10.1147/sj.454.0663

Orna Kupferman and Moshe Y. Vardi. 2003. Vacuity detection in temporal model checking. *STTT* 4, 2 (2003), 224–233. DOI:http://dx.doi.org/10.1007/s100090100062

Evelina Lamma, Paola Mello, Marco Montali, Fabrizio Riguzzi, and Sergio Storari. 2007a. Inducing Declarative Logic-Based Models from Labeled Traces, See Alonso et al. [2007], 344–359. DOI:http://dx.doi.org/10.1007/978-3-540-75183-0_25

Evelina Lamma, Paola Mello, Fabrizio Riguzzi, and Sergio Storari. 2007b. Applying Inductive Logic Programming to Process Mining. In *ILP (Lecture Notes in Computer Science)*, Hendrik Blockeel, Jan Ramon, Jude W. Shavlik, and Prasad Tadepalli (Eds.), Vol. 4894. Springer, 132–146. DOI:http://dx.doi.org/10.1007/978-3-540-78469-2_16

Fabrizio Maria Maggi. 2013. Declarative Process Mining with the Declare Component of ProM, See Fauvet and van Dongen [2013]. http://ceur-ws.org/Vol-1021/paper_8.pdf

Fabrizio Maria Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. 2012. Efficient Discovery of Understandable Declarative Process Models from Event Logs. In *CAiSE (Lecture Notes in Computer Science)*, Jolita Ralyté, Xavier Franch, Sjaak Brinkkemper, and Stanislaw Wrycza (Eds.), Vol. 7328. Springer, 270–285. http://dx.doi.org/10.1007/978-3-642-31095-9_18

Fabrizio Maria Maggi, R. P. Jagadeesh Chandra Bose, and Wil M. P. van der Aalst. 2013. A Knowledge-Based Integrated Approach for Discovering and Repairing Declare Maps. In *CAiSE (Lecture Notes in Computer Science)*, Camille Salinesi, Moira C. Norrie, and Oscar Pastor (Eds.), Vol. 7908. Springer, 433–448. DOI:http://dx.doi.org/10.1007/978-3-642-38709-8_28

Fabrizio Maria Maggi, Arjan J. Mooij, and Wil M. P. van der Aalst. 2011. User-guided discovery of declarative process models. In *CIDM*. IEEE, 192–199. http://dx.doi.org/10.1109/CIDM.2011.5949297

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*. Cambridge University Press. I–XXI, 1–482 pages.

Jan Mendling, Gustaf Neumann, and Wil M. P. van der Aalst. 2007a. Understanding the Occurrence of Errors in Process Models Based on Metrics. In *OTM Conferences (1) (Lecture Notes in Computer Science)*, Robert Meersman and Zahir Tari (Eds.), Vol. 4803. Springer, 113–130. DOI:http://dx.doi.org/10.1007/978-3-540-76848-7_9

Jan Mendling, Hajo A. Reijers, and Jorge Cardoso. 2007b. What Makes Process Models Understandable?, See Alonso et al. [2007], 48–63. DOI:http://dx.doi.org/10.1007/978-3-540-75183-0_4

Marco Montali. 2010. *Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach*. Lecture Notes in Business Information Processing, Vol. 56. Springer. DOI:http://dx.doi.org/10.1007/978-3-642-14538-4

Maja Pesic, Dragan Bosnacki, and Wil M. P. van der Aalst. 2010. Enacting Declarative Languages Using LTL: Avoiding Errors and Improving Performance. In *SPIN (Lecture Notes in Computer Science)*, Jaco van de Pol and Michael Weber (Eds.), Vol. 6349. Springer, 146–161. DOI:http://dx.doi.org/10.1007/978-3-642-16164-3_11

Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. 2007. DECLARE: Full Support for Loosely-Structured Processes. In *EDOC*. IEEE Computer Society, 287–300. http://doi.ieeecomputersociety.org/10.1109/EDOC.2007.25

Maja Pesic and Wil M. P. van der Aalst. 2006. A Declarative Approach for Flexible Business Processes Management. In *Business Process Management Workshops (Lecture Notes in Computer Science)*, Johann Eder and Schahram Dustdar (Eds.), Vol. 4103. Springer, 169–180. http://dx.doi.org/10.1007/11837862_18

Paul Pichler, Barbara Weber, Stefan Zugal, Jakob Pinggera, Jan Mendling, and Hajo A. Reijers. 2011. Imperative versus Declarative Process Modeling Languages: An Empirical Investigation. In *Business Process Management Workshops (1) (Lecture Notes in Business Information Processing)*, Florian Daniel, Kamel Barkaoui, and Schahram Dustdar (Eds.), Vol. 99. Springer, 383–394. DOI:http://dx.doi.org/10.1007/978-3-642-28108-2_37

Matthew Richardson and Pedro Domingos. 2006. Markov logic networks. *Machine Learning* 62, 1-2 (2006), 107–136. DOI:http://dx.doi.org/10.1007/s10994-006-5833-1

Dennis M. M. Schunselaar, Fabrizio Maria Maggi, and Natalia Sidorova. 2012. Patterns for a Log-Based Strengthening of Declarative Compliance Models. In *IFM (Lecture Notes in Computer Science)*, John Derrick, Stefania Gnesi, Diego Latella, and Helen Treharne (Eds.), Vol. 7321. Springer, 327–342. DOI:http://dx.doi.org/10.1007/978-3-642-30729-4_23

Arthur H. M. ter Hofstede, Wil M. P. van der Aalst, Michael Adamns, and Nick Russell (Eds.). 2010. *Modern Business Process Automation: YAWL and its Support Environment*. Springer. http://www.springer.com/computer+science/database+management+\%26+information+retrieval/book/978-3-642-03120-5

Wil M. P. van der Aalst, , Vladimir Rubin, Eric Verbeek, Boudewijn F. van Dongen, Ekkart Kindler, and Christian W. Günther. 2010. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling* 9 (2010), 87–111. Issue 1. http://dx.doi.org/10.1007/s10270-008-0106-z 10.1007/s10270-008-0106-z.

Wil M. P. van der Aalst. 2011. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer. I–XVI, 1–352 pages. DOI:http://dx.doi.org/10.1007/978-3-642-19345-3

Wil M. P. van der Aalst. 2012. Process Mining: Overview and Opportunities. *ACM Trans. Manage. Inf. Syst.* 3, 2, Article 7 (July 2012), 17 pages. DOI:http://dx.doi.org/10.1145/2229156.2229157

Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. 2009a. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D* 23, 2 (2009), 99–113. DOI:http://dx.doi.org/10.1007/s00450-009-0057-9

Wil M. P. van der Aalst, Boudewijn F. van Dongen, Christian W. Günther, Anne Rozinat, Eric Verbeek, and Ton Weijters. 2009b. ProM: The Process Mining Toolkit. In *BPM (Demos) (CEUR Workshop Proceedings)*, Ana Karla A. de Medeiros and Barbara Weber (Eds.), Vol. 489. CEUR-WS.org. http://ceur-ws.org/Vol-489/paper3.pdf

Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. 2004. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Trans. Knowl. Data Eng.* 16, 9 (2004), 1128–1142. http://csdl.computer.org/comp/trans/tk/2004/09/k1143abs.htm

Boudewijn F. van Dongen. 2011. Real-life event logs – A hospital log. First International Business Process Intelligence Challenge (BPIC'11). (2011). DOI:http://dx.doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54

Boudewijn F. van Dongen. 2012. Real-life event logs – A loan application process. Second International Business Process Intelligence Challenge (BPIC'12). (2012). DOI:http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

Paul Warren, Nick Kings, Ian Thurlow, John Davies, Tobias Buerger, Elena Simperl, Carlos Ruiz, Jose Manuel Gomez-Perez, Vadim Ermolayev, Rayid Ghani, Marcel Tilly, Tom Bösser, and Ali Imtiaz. 2009. Improving Knowledge Worker Productivity - the Active integrated approach. *BT Technology Journal* 26, 2 (2009), 165–176.

A. J. M. M. Weijters and Wil M. P. van der Aalst. 2003. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering* 10, 2 (2003), 151–162. http://iospress.metapress.com/content/8puq22eumrva7vyp/

Lijie Wen, Wil M. P. van der Aalst, Jianmin Wang, and Jiaguang Sun. 2007. Mining process models with non-free-choice constructs. *Data Min. Knowl. Discov.* 15, 2 (2007), 145–180. http://dx.doi.org/10.1007/s10618-007-0065-y

Michael Westergaard. 2011. Better Algorithms for Analyzing and Enacting Declarative Workflow Languages Using LTL. In *BPM (Lecture Notes in Computer Science)*, Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf (Eds.), Vol. 6896. Springer, 83–98. http://dx.doi.org/10.1007/978-3-642-23059-2_10

Michael Westergaard and Christian Stahl. 2013. Leveraging Super-Scalarity and Parallelism to Provide Fast Declare Mining without Restrictions, See Fauvet and van Dongen [2013]. http://ceur-ws.org/Vol-1021/paper_10.pdf

Received 15/05/2013; revised 05/11/2013; accepted 24/04/2014

Online Appendix to: On the Discovery of Declarative Control Flows for Artful Processes

Claudio DI CICCIO, Wirtschaftsuniversität Wien
Massimo MECELLA, Sapienza Università di Roma

A. TECHNICAL DETAILS ON THE SUPPORT FUNCTIONS

Support functions compute a value assessing the reliability of a constraint, i.e., they estimate to what extent a constraint is likely to hold true in the discovered process. In order to do so, they apply a heuristic-driven analysis of the statistical information kept in MINERfulKB. In order to discuss the rationale behind them, we proceed as follows: *(i)* we provide formal semantics for the Declare constraint templates that MINERful discovers (see Section 3), in First Order Logic (FOL); *(ii)* we specify formal semantics for the functions of MINERfulKB (see Section 4.1); *(iii)* we show that the queries evaluated by MINERful on MINERfulKB for discovering constraints express the ratio of cases favorable to the constraint, to the number of all cases possible, w.r.t. the analyzed event log.

A.1. Declare Constraint Templates as FOL Formulae

Semantics of Declare constraint templates have been specified as Linear Temporal Logic (LTL [Clarke et al. 2001]) formulae, interpreted on finite traces (LTL$_f$) [Pesic et al. 2007; ter Hofstede et al. 2010]. LTL$_f$ is known to have the same expressive power of First Order Logic over finite ordered traces [Gabbay et al. 1980; Diekert and Gastin 2008]. Therefore, here we provide such semantics of Declare templates as FOL formulae. This approach is inspired by the LTL$_f$-to-FOL translation over finite linear ordered sequences, discussed in [De Giacomo and Vardi 2013].

Specifically, we consider a first-order language that consists of:

(1) variables (e.g., i, j, k, ρ, σ, etc.);
(2) constants *first* and *last*;
(3) binary predicates $Succ$, $<$, $=$;
(4) the ternary predicate $InTrace$.

The domain of interpretation is $\Delta^t = \langle \{1, \ldots, |t|\} \cup \Sigma \rangle$ with $|t| \in \mathbb{N}^+$ (where $\mathbb{N}^+$ is the set of natural numbers, excluding 0) and Σ is an alphabet of characters. We denote the uppermost value in the range of Δ^t as $|t|$, in order to stress on the fact that it is meant to be the length of a string t. Indeed, such a string is thought to be a representation of a trace, where each character identifies an event. Σ is the alphabet consisting of events' identifiers.

For the sake of comprehensibility, we specify here the interpretation function $\left(^{\Delta^t,t}\cdot\right)$, which we will adopt in the following. It interprets the aforementioned predicates and constants as follows. Let $t \in \Sigma^*$ be a string, $|t|$ the length of the string t and $t[i]$ the character occurring in t at position i. Then,

— $^{\Delta^t,t}Succ \doteq \left\{ \langle i, j \rangle \ : \ i \in 1, \ldots, |t| - 1 \quad \text{and} \quad j = i + 1 \right\}$
— $^{\Delta^t,t} < \doteq \left\{ \langle i, j \rangle \ : \ i \in 1, \ldots, |t| \quad \text{and} \quad i < j \right\}$

© YYYY ACM 2158-656X/YYYY/01-ARTA $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000

— $\Delta^{t},{}^{t} = \doteq \Big\{ \langle i, i \rangle \ : \ i \in 1, \ldots, |t| \Big\}$
— $\Delta^{t},{}^{t}\mathit{first} \ \doteq 1$ and $\Delta^{t},{}^{t}\mathit{last} \ \doteq |t|$
— $\Delta^{t},{}^{t}\mathit{InTrace} \ \doteq \Big\{ \langle i, \rho \rangle \ : \ t[i] = \rho \Big\}$

As an example, on a string t like aabac over the alphabet $\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, the interpretation function would be such that $\Delta^{\mathsf{aabac}} = \langle \{1, \ldots, 5\} \ \cup \{\mathsf{a}, \mathsf{b}, \mathsf{c}\} \rangle$. The predicates would be interpreted as follows:

— $\Delta^{\mathsf{aabac}},{}^{\mathsf{aabac}}\mathit{Succ} \ = \Big\{ \langle 1, 2 \rangle, \langle 2, 3 \rangle, \langle 3, 4 \rangle, \langle 4, 5 \rangle \Big\}$
— $\Delta^{\mathsf{aabac}},{}^{\mathsf{aabac}} < \doteq \Big\{ \langle 1, 2 \rangle, \langle 2, 3 \rangle, \langle 3, 4 \rangle, \langle 4, 5 \rangle, \langle 1, 3 \rangle, \langle 1, 4 \rangle, \langle 1, 5 \rangle, \langle 2, 4 \rangle, \langle 2, 5 \rangle, \langle 3, 5 \rangle \Big\}$
— $\Delta^{\mathsf{aabac}},{}^{\mathsf{aabac}} = \doteq \Big\{ \langle 1, 1 \rangle, \langle 2, 2 \rangle, \langle 3, 3 \rangle, \langle 4, 4 \rangle, \langle 5, 5 \rangle \Big\}$
— $\Delta^{\mathsf{aabac}},{}^{\mathsf{aabac}}\mathit{first} \ \doteq 1$ and $\Delta^{\mathsf{aabac}},{}^{\mathsf{aabac}}\mathit{last} \ \doteq 5$
— $\Delta^{\mathsf{aabac}},{}^{\mathsf{aabac}}\mathit{InTrace} \ \doteq \Big\{ \langle 1, \mathsf{a} \rangle, \langle 2, \mathsf{a} \rangle, \langle 3, \mathsf{b} \rangle, \langle 4, \mathsf{a} \rangle, \langle 5, \mathsf{c} \rangle \Big\}$

The specification of Declare constraint templates follows, in terms of predicates of the First-Order language that we have described. In the following, together with the quantifiers $\exists$ and $\forall$, we will make use of the usual abbreviation $\nexists$ for $\neg\exists$ and $\neq$ for $\neg =$. $<$, $=$ and $\neq$ will be used with an infix notation, for the sake of readability.

$$\mathit{Init}(\rho) \equiv \mathit{InTrace}(\mathit{first}, \rho) \tag{1a}$$
$$\mathit{End}(\rho) \equiv \mathit{InTrace}(\mathit{last}, \rho) \tag{1b}$$
$$\mathit{Participation}(\rho) \equiv \exists i. \ \mathit{InTrace}(i, \rho) \tag{1c}$$
$$\mathit{AtMostOne}(\rho) \equiv \exists i. \ \mathit{InTrace}(i, \rho) \ \rightarrow \ \nexists j. \ \mathit{InTrace}(j, \rho) \ \wedge \ j \neq i \tag{1d}$$

$$\mathit{RespondedExistence}(\rho, \sigma) \equiv \forall i. \ \mathit{InTrace}(i, \rho) \ \rightarrow \exists j. \ \mathit{InTrace}(j, \sigma) \ \wedge \ i \neq j \tag{1e}$$
$$\mathit{Response}(\rho, \sigma) \equiv \forall i. \ \mathit{InTrace}(i, \rho) \ \rightarrow \exists j. \ \mathit{InTrace}(j, \sigma) \ \wedge \ i < j \tag{1f}$$
$$\begin{aligned}\mathit{AlternateResponse}(\rho, \sigma) \equiv \forall i. \ \mathit{InTrace}(i, \rho) \ \rightarrow & \exists j. \ \mathit{InTrace}(j, \sigma) \ \wedge \ i < j \wedge \\ & \nexists l. \ \mathit{InTrace}(l, \sigma) \ \wedge \ i < l < j \ \rightarrow \\ & \nexists k. \ \mathit{InTrace}(k, \rho) \ \wedge \ i < k < j\end{aligned} \tag{1g}$$
$$\mathit{ChainResponse}(\rho, \sigma) \equiv \forall i. \ \mathit{InTrace}(i, \rho) \ \rightarrow \exists j. \ \mathit{InTrace}(j, \sigma) \ \wedge \ \mathit{Succ}(i, j) \tag{1h}$$
$$\mathit{Precedence}(\rho, \sigma) \equiv \forall j. \ \mathit{InTrace}(j, \sigma) \ \rightarrow \exists i. \ \mathit{InTrace}(i, \rho) \ \wedge \ i < j \tag{1i}$$
$$\begin{aligned}\mathit{AlternatePrecedence}(\rho, \sigma) \equiv \forall j. \ \mathit{InTrace}(j, \sigma) \ \rightarrow & \exists i. \ \mathit{InTrace}(i, \rho) \ \wedge \ i < j \wedge \\ & \nexists k. \ \mathit{InTrace}(k, \rho) \ \wedge \ i < k < j \ \rightarrow \\ & \nexists l. \ \mathit{InTrace}(l, \sigma) \ \wedge \ i < l < j\end{aligned} \tag{1j}$$
$$\mathit{ChainPrecedence}(\rho, \sigma) \equiv \forall j. \ \mathit{InTrace}(j, \sigma) \ \rightarrow \exists i. \ \mathit{InTrace}(i, \rho) \ \wedge \ \mathit{Succ}(i, j) \tag{1k}$$

$$\mathit{CoExistence}(\rho, \sigma) \equiv \mathit{RespondedExistence}(\rho, \sigma) \ \wedge \ \mathit{RespondedExistence}(\sigma, \rho) \tag{1l}$$
$$\mathit{Succession}(\rho, \sigma) \equiv \mathit{Response}(\rho, \sigma) \ \wedge \ \mathit{Precedence}(\rho, \sigma) \tag{1m}$$
$$\mathit{AlternateSuccession}(\rho, \sigma) \equiv \mathit{AlternateResponse}(\rho, \sigma) \ \wedge \ \mathit{AlternatePrecedence}(\rho, \sigma) \tag{1n}$$
$$\mathit{ChainSuccession}(\rho, \sigma) \equiv \mathit{ChainResponse}(\rho, \sigma) \ \wedge \ \mathit{ChainPrecedence}(\rho, \sigma) \tag{1o}$$

$$\begin{aligned}
\mathit{NotCoExistence}(\rho,\sigma) \equiv & (\forall i.\ \mathit{InTrace}(i,\rho) \ \rightarrow \nexists j.\ \mathit{InTrace}(j,\sigma) \ \wedge \ i \neq j) \wedge \\
& (\forall j.\ \mathit{InTrace}(j,\sigma) \ \rightarrow \nexists i.\ \mathit{InTrace}(i,\rho) \ \wedge \ i \neq j) && \text{(1p)} \\
\mathit{NotSuccession}(\rho,\sigma) \equiv & (\forall i.\ \mathit{InTrace}(i,\rho) \ \rightarrow \nexists j.\ \mathit{InTrace}(j,\sigma) \ \wedge \ i < j) \\
& (\forall j.\ \mathit{InTrace}(j,\sigma) \ \rightarrow \nexists i.\ \mathit{InTrace}(i,\rho) \ \wedge \ i < j) && \text{(1q)} \\
\mathit{NotChainSuccession}(\rho,\sigma) \equiv & (\forall i.\ \mathit{InTrace}(i,\rho) \ \rightarrow \nexists j.\ \mathit{InTrace}(j,\sigma) \ \wedge \ \mathit{Succ}(i,j)) \\
& (\forall j.\ \mathit{InTrace}(j,\sigma) \ \rightarrow \nexists i.\ \mathit{InTrace}(i,\rho) \ \wedge \ \mathit{Succ}(i,j)) && \text{(1r)}
\end{aligned}$$

A.2. Formal specification of MINERfulKB

According to the description provided in Section 4.1, we can describe the semantics of MINERful ownplay and MINERful interplay functions, according to the previously defined language. A textual description of each function, along with examples, can be found in Section 4.1. Let $t \in \Sigma^*$ be a string of a log T, ρ and σ two characters in the alphabet Σ and d an integer, ranging from $-|t|$ to $|t|$, where $|t|$ is the length of t. Thereupon, we have the following formulae.

$$\begin{aligned}
\delta(T,\rho,\sigma,d) &= \sum_{t \in T} \left| \left\{ i \ : \ t[i] = \rho \ \wedge \ \exists i \ . \ t[j] = \sigma \ \wedge \ j = i + d \right\} \right| && \text{(2a)} \\
\delta(T,\rho,\sigma,+\infty) &= \sum_{t \in T} \left| \left\{ i \ : \ t[i] = \rho \ \wedge \ \nexists j \ . \ t[j] = \sigma \ \wedge \ i < j \right\} \right| \\
\delta(T,\rho,\sigma,-\infty) &= \sum_{t \in T} \left| \left\{ i \ : \ t[i] = \rho \ \wedge \ \nexists j \ . \ t[j] = \sigma \ \wedge \ i < j \right\} \right| \\
\delta(T,\rho,\sigma,\pm\infty) &= \sum_{t \in T} \left| \left\{ i \ : \ t[i] = \rho \ \wedge \ \nexists j \ . \ t[j] = \sigma \ \wedge \ j \neq i \right\} \right|
\end{aligned}$$

$$\begin{aligned}
\beta^{\rightarrow}(T,\rho,\sigma) = \sum_{t \in T} \Big| \Big\{ k \ : t[i] = \rho \ \wedge \ t[k] = \rho \ \wedge \ t[j] = \sigma \ \wedge \\
\nexists l \ . \ t[l] = \sigma \ \wedge \ i < l < j \Big\} \Big| && \text{(2b)} \\
\beta^{\leftarrow}(T,\rho,\sigma) = \sum_{t \in T} \Big| \Big\{ l \ : \ t[i] = \rho \ \wedge \ t[l] = \sigma \ \wedge \ t[j] = \sigma \ \wedge \\
\nexists k \ . \ t[k] = \rho \ \wedge \ i < k < j \Big\} \Big| && \text{(2c)}
\end{aligned}$$

$$\begin{aligned}
\gamma(T,\rho,n) = \Big| \Big\{ t \in T \ : \exists i^1, i^2, \ldots, i^n \ . \ \bigwedge_{m \in [1,n]} t[i^m] = \rho \ \wedge \bigwedge_{\substack{m,p \in [1,n] \\ m \neq p}} t[i^m] \neq t[i^p] \ \wedge \\
\nexists i^{n+1} \ . \ t[i^{n+1}] = \rho \ \wedge \bigwedge_{m \in [1,n]} i^{n+1} \neq i^m \Big\} \Big| && \text{(2d)}
\end{aligned}$$

$$\begin{aligned}
\alpha(T,\rho) &= \Big| \Big\{ t \in T \ : t[1] = \rho \Big\} \Big| && \text{(2e)} \\
\omega(T,\rho) &= \Big| \Big\{ t \in T \ : t[|t|] = \rho \Big\} \Big| && \text{(2f)}
\end{aligned}$$

From $\gamma(T,\rho,n)$ we also derive the total number of occurrences of ρ in the log:

$$\Gamma(T,\rho) = \sum_{n > 0} \gamma(T,\rho,n) \cdot n$$

A.3. Support Functions for Computing the Probability of Constraints

Table VIII reports the functions computing the support for constraints in MINERful. In the following, we discuss their validity as suitable estimates for assessing whether constraints hold true in the process: we show that each function expresses the probability of a constraint to hold true, according to classical definition of Laplace:[9] the ratio of cases favorable to the constraint, to the number of all cases possible. The favorable and total cases are taken from the analysis of a log. As required from the definition, we assume that "none of the cases occurs more than any other", i.e., we consider them equally possible.

The cases favorable to the constraints derive from the semantics of constraints themselves, as described in Section A.1. Indeed, applying the interpretation function $\left({}^{\Delta^t,t}\cdot\right)$ explained in Section A.1 to the FOL predicates of expressions 1a–r, we obtain the following formulae determining whether a trace (string) t complies to a constraint.

$$t \models {}^{\Delta^t,t}\mathit{Init}(\rho) \quad \text{iff} \quad t[1] = \rho \tag{3a}$$

$$t \models {}^{\Delta^t,t}\mathit{End}(\rho) \quad \text{iff} \quad t[|t|] = \rho \tag{3b}$$

$$t \models {}^{\Delta^t,t}\mathit{Participation}(\rho) \quad \text{iff} \quad \exists i.\ t[i] = \rho \tag{3c}$$

$$t \models {}^{\Delta^t,t}\mathit{AtMostOne}(\rho) \quad \text{iff} \quad \exists i.\ t[i] = \rho \rightarrow \quad \nexists j.\ j \neq i \ \wedge \ t[j] = \rho \tag{3d}$$

$$t \models {}^{\Delta^t,t}\mathit{RespondedExistence}(\rho,\sigma) \quad \text{iff} \quad \forall i.\ t[i] = \rho \rightarrow \exists j.\ t[j] = \sigma \ \wedge \ i \neq j \tag{3e}$$

$$t \models {}^{\Delta^t,t}\mathit{Response}(\rho,\sigma) \quad \text{iff} \quad \forall i.\ t[i] = \rho \rightarrow \exists j.\ t[j] = \sigma \ \wedge \ i < j \tag{3f}$$

$$\begin{aligned} t \models {}^{\Delta^t,t}\mathit{AlternateResponse}(\rho,\sigma) \quad \text{iff} \quad & \forall i.\ t[i] = \rho \rightarrow \exists j.\ t[j] = \sigma \ \wedge \ i < j \ \wedge \\ & \nexists l.\ t[l] = \sigma \ \wedge \ i < l < j \ \rightarrow \\ & \nexists k.\ t[k] = \rho \ \wedge \ i < k < j \end{aligned} \tag{3g}$$

$$t \models {}^{\Delta^t,t}\mathit{ChainResponse}(\rho,\sigma) \quad \text{iff} \quad \forall i.\ t[i] = \rho \rightarrow \exists j.\ t[j] = \sigma \ \wedge \ j = i+1 \tag{3h}$$

$$t \models {}^{\Delta^t,t}\mathit{Precedence}(\rho,\sigma) \quad \text{iff} \quad \forall j.\ t[j] = \sigma \rightarrow \exists i.\ t[i] = \rho \ \wedge \ i < j \tag{3i}$$

$$\begin{aligned} t \models {}^{\Delta^t,t}\mathit{AlternatePrecedence}(\rho,\sigma) \quad \text{iff} \quad & \forall j.\ t[j] = \sigma \rightarrow \exists i.\ t[i] = \rho \ \wedge \ i < j \ \wedge \\ & \nexists k.\ t[k] = \rho \ \wedge \ i < k < j \ \rightarrow \\ & \nexists l.\ t[l] = \sigma \ \wedge \ i < l < j \end{aligned} \tag{3j}$$

$$t \models {}^{\Delta^t,t}\mathit{ChainPrecedence}(\rho,\sigma) \quad \text{iff} \quad \forall j.\ t[j] = \sigma \rightarrow \exists i.\ t[i] = \rho \ \wedge \ i = j+1 \tag{3k}$$

[9]cf. Pierre-Simon de Laplace: Théorie analytique des probabilités. Paris, 1812: *the probability of an event is the ratio of the number of cases favorable to it, to the number of all cases possible when nothing leads us to expect that any one of these cases should occur more than any other, which renders them, for us, equally possible*.

$$t \models {}^{\Delta^t,t}\mathit{CoExistence}(\rho,\sigma) \quad \text{iff} \quad {}^{\Delta^t,t}\mathit{RespondedExistence}(\rho,\sigma) \wedge {}^{\Delta^t,t}\mathit{RespondedExistence}(\sigma,\rho) \tag{3l}$$

$$t \models {}^{\Delta^t,t}\mathit{Succession}(\rho,\sigma) \quad \text{iff} \quad {}^{\Delta^t,t}\mathit{Response}(\rho,\sigma) \wedge {}^{\Delta^t,t}\mathit{Precedence}(\rho,\sigma) \tag{3m}$$

$$t \models {}^{\Delta^t,t}\mathit{AlternateSuccession}(\rho,\sigma) \quad \text{iff} \quad {}^{\Delta^t,t}\mathit{AlternateResponse}(\rho,\sigma) \wedge {}^{\Delta^t,t}\mathit{AlternatePrecedence}(\rho,\sigma) \tag{3n}$$

$$t \models {}^{\Delta^t,t}\mathit{ChainSuccession}(\rho,\sigma) \quad \text{iff} \quad {}^{\Delta^t,t}\mathit{ChainResponse}(\rho,\sigma) \wedge {}^{\Delta^t,t}\mathit{ChainPrecedence}(\rho,\sigma) \tag{3o}$$

$$t \models {}^{\Delta^t,t}\mathit{NotCoExistence}(\rho,\sigma) \quad \text{iff} \quad (\forall i.\ t[i]=\rho \rightarrow \nexists j.\ t[j]=\sigma \wedge i \neq j) \wedge (\forall i.\ t[j]=\sigma \rightarrow \nexists i.\ t[i]=\rho \wedge i \neq j) \tag{3p}$$

$$t \models {}^{\Delta^t,t}\mathit{NotSuccession}(\rho,\sigma) \quad \text{iff} \quad (\forall i.\ t[i]=\rho \rightarrow \nexists j.\ t[j]=\sigma \wedge i < j) \wedge (\forall i.\ t[j]=\sigma \rightarrow \nexists i.\ t[i]=\rho \wedge i < j) \tag{3q}$$

$$t \models {}^{\Delta^t,t}\mathit{NotChainSuccession}(\rho,\sigma) \quad \text{iff} \quad (\forall i.\ t[i]=\rho \rightarrow \nexists j.\ t[j]=\sigma \wedge j = i+1) \wedge (\forall j.\ t[j]=\sigma \rightarrow \nexists i.\ t[i]=\rho \wedge j = i+1) \tag{3r}$$

A.4. *Init* and *End*

$\alpha(T,\rho)$ (Formula 2e) and $\omega(T,\rho)$ (Formula 2f) count the number of strings t in log T complying to, resp., $\mathit{Init}(\rho)$ and $\mathit{End}(\rho)$. It comes immediately from the comparison of Formulae 3a (3b) and 2e (2f). Therefore, $\alpha(T,\rho)$ and, resp., $\omega(T,\rho)$ count the number of favorable cases to $\mathit{Init}(\rho)$ and $\mathit{End}(\rho)$. Given log T, the total number of cases is the size of the log itself. This leads to the Support functions of $\mathit{Init}(\rho)$ and $\mathit{End}(\rho)$, i.e., resp.,

$$\frac{\alpha(T,\rho)}{|T|} \text{ and } \frac{\omega(T,\rho)}{|T|}$$

(cf. Table VIII).

A.5. *Participation*

We recall here that the probability of an event A, $P(A)$, is equal to $1 - P(\bar{A})$ where $\bar{A}$ is the complement of A. Therefore, we may want to consider the following:

$$t \models {}^{\Delta^t,t}\neg\mathit{Participation}(\rho) \quad \text{iff} \quad \nexists i.\ t[i]=\rho$$

From the definition of $\gamma(T,\rho,n)$ (Formula 2d), we have that

$$\gamma(T,\rho,0) = \left|\left\{t \in T \ :\ \nexists i \ :\ t[i]=\rho\right\}\right|$$

Hence, $\gamma(T,\rho,0)$ counts the number of cases favorable to $\neg\mathit{Participation}(\rho)$. Thus,

$$1 - \frac{\gamma(\rho,0)}{|T|}$$

is a suitable Support function for $\mathit{Participation}(\rho)$.

A.6. *AtMostOne*

From the definition of $\gamma(T,\rho,n)$ (Formula 2d), we have that

$$\gamma(T,\rho,1) = \left|\left\{t \in T \;:\; \exists i^1 \;:\; t[i^1] = \rho \;\wedge\; \nexists i^2 \,.\, t[i^2] = \rho \;\wedge\; i^2 \neq i^1\right\}\right|$$

Given two FOL formulae ϕ and ψ, *(i)* the logic implication $\phi \rightarrow \psi$ is derived by disjunction and negation as $\neg\phi \vee \psi$, and *(ii)* $\phi \vee \psi \equiv \phi \vee \psi \wedge \neg\phi$, we can rewrite the Formula 3d as follows:

$$t \models^{\Delta^t,t} \mathit{AtMostOne}(\rho) \quad \text{iff} \quad \underbrace{\nexists i.\; t[i] = \rho}_{\text{cf. } \gamma(T,\rho,0)} \quad \vee \quad \underbrace{\exists i.\; t[i] = \rho \;\wedge\; \nexists j.\; j \neq i \;\wedge\; t[j] = \rho}_{\text{cf. } \gamma(T,\rho,1)}$$

As highlighted in the Formula, the first term of the disjunction is captured by $\gamma(T,\rho,0)$, counting the favorable cases. $\gamma(T,\rho,1)$ counts the cases that are favorable to the second term. The two terms represent mutually exclusive conditions. We recall that if A and B are mutually exclusive events, $P(A \cup B) = P(A) + P(B)$. Therefore,

$$\frac{\gamma(T,\rho,0)}{|T|} + \frac{\gamma(T,\rho,1)}{|T|} = \frac{\gamma(T,\rho,0) + \gamma(T,\rho,1)}{|T|}$$

properly computes Support of $\mathit{AtMostOne}(\rho)$.

A.7. *RespondedExistence*, *Response* **and** *Precedence*

Following the procedure of Section A.5, we negate Formulae 3e, 3f and 3i as a first step. Remembering that $\neg\forall\phi \equiv \exists\neg\phi$ and that $\neg(\phi \vee \psi) \equiv \neg\phi \wedge \neg\psi$, we have the following:

$$\begin{aligned}
t \models^{\Delta^t,t} \neg\mathit{RespondedExistence}(\rho,\sigma) \quad &\text{iff} \quad \exists i.\; t[i] = \rho \wedge \nexists j.\; t[j] = \sigma \;\wedge\; i \neq j \\
t \models^{\Delta^t,t} \neg\mathit{Response}(\rho,\sigma) \quad &\text{iff} \quad \exists i.\; t[i] = \rho \wedge \nexists j.\; t[j] = \sigma \;\wedge\; i < j \\
t \models^{\Delta^t,t} \neg\mathit{Precedence}(\rho,\sigma) \quad &\text{iff} \quad \exists j.\; t[j] = \sigma \wedge \nexists i.\; t[i] = \rho \;\wedge\; i < j
\end{aligned}$$

Given the four Formulae 2a, we see that, resp., $\delta(T,\rho,\sigma,\pm\infty))$, $\delta(T,\rho,\sigma,+\infty))$ and $\delta(T,\sigma,\rho,-\infty))$ count the favorable cases for $\mathit{RespondedExistence}(\rho,\sigma)$, $\mathit{Response}(\rho,\sigma)$ and $\mathit{Precedence}(\rho,\sigma)$ in the log.

We highlight here that the indexes, where ρ (for $\mathit{RespondedExistence}(\rho,\sigma)$ and $\mathit{Response}(\rho,\sigma)$) and σ (for $\mathit{Precedence}(\rho,\sigma)$) appear in the string, are counted, instead of the entire strings. This is a different approach with respect to the computation of Support for Existence Constraints, seen in Sections A.4, A.5 and A.6. This is due to the fact that we might have favorable and unfavorable cases within the same string. For instance, a string like aabac has two favorable cases and an unfavorable one for $\mathit{Response}(\mathsf{a},\mathsf{b})$. Therefore, as the total number of cases, we do not consider the number of strings in the log, but rather the number of occurrences of ρ (for $\mathit{RespondedExistence}(\rho,\sigma)$ and $\mathit{Response}(\rho,\sigma)$) and σ (for $\mathit{Precedence}(\rho,\sigma)$) in the log – i.e., $\Gamma(T,\rho)$ and, resp., $\Gamma(T,\sigma)$.

Consequently,

$$1 - \frac{\delta(T,\rho,\sigma,\pm\infty)}{\Gamma(T,\rho)},\; 1 - \frac{\delta(T,\rho,\sigma,+\infty)}{\Gamma(T,\rho)} \text{ and } 1 - \frac{\delta(T,\sigma,\rho,-\infty)}{\Gamma(T,\sigma)}$$

are proper Support functions for, resp., $\mathit{RespondedExistence}(\rho,\sigma)$, $\mathit{Response}(\rho,\sigma)$ and $\mathit{Precedence}(\rho,\sigma)$.

A.8. *AlternateResponse* and *AlternatePrecedence*

The negations of Formulae 3g and 3j can be reformulated and expanded as follows. Considering that $i < k < l < j \models i < j$, we have:

$$\begin{aligned}
t \models^{\Delta^t,t} \neg \mathit{AlternateResponse}(\rho,\sigma) \quad \text{iff} \quad & \exists i.\ t[i] = \rho \ \wedge\ (\ \nexists j.\ t[j] = \sigma \ \wedge\ i < j \vee \\
& \qquad \nexists l.\ t[l] = \sigma \ \wedge\ i < l < j \wedge \\
& \qquad \exists k.\ t[k] = \rho \ \wedge\ i < k < j\ ) \\
t \models^{\Delta^t,t} \neg \mathit{AlternatePrecedence}(\rho,\sigma) \quad \text{iff} \quad & \exists j.\ t[j] = \sigma \ \wedge\ (\nexists i.\ t[i] = \rho \ \wedge\ i < j \vee \\
& \qquad \nexists k.\ t[k] = \rho \ \wedge\ i < k < j \wedge \\
& \qquad \exists l.\ t[l] = \sigma \ \wedge\ i < l < j\ )
\end{aligned}$$

Applying the property according to which $\phi \vee \psi \ \equiv\ \phi \vee \psi \wedge \neg\phi$ and the distributivity of the logical conjunction over logical disjunction, we have the equivalent following Formulae:

$$\begin{aligned}
t \models^{\Delta^t,t} \neg \mathit{AlternateResponse}(\rho,\sigma) \quad \text{iff} \quad & \exists i.\ (t[i] = \rho \wedge\ \nexists j.\ t[j] = \sigma \ \wedge\ i < j\ )\ \vee \\
& (t[i] = \rho \wedge\ \exists j.\ t[j] = \sigma \ \wedge\ i < j \ \wedge \\
& \qquad \exists l.\ t[l] = \sigma \ \wedge\ i < l < j \wedge \\
& \qquad \exists k.\ t[k] = \rho \ \wedge\ i < k < j\ ) \\
t \models^{\Delta^t,t} \neg \mathit{AlternatePrecedence}(\rho,\sigma) \quad \text{iff} \quad & \exists j.\ (t[j] = \sigma \wedge \nexists i.\ t[i] = \rho \ \wedge\ i < j\ )\ \vee \\
& (t[j] = \sigma \wedge \exists i.\ t[i] = \rho \ \wedge\ i < j \wedge \\
& \qquad \nexists k.\ t[k] = \rho \ \wedge\ i < k < j \wedge \\
& \qquad \exists l.\ t[l] = \sigma \ \wedge\ i < l < j\ )
\end{aligned}$$

The preceding Formulae are thus both divided into a disjunction between two mutually exclusive conditions. The favorable cases for the first terms of the disjunctions are resp. counted by $\delta(T,\rho,\sigma,+\infty))$ and $\delta(T,\sigma,\rho,-\infty))$ (see Formulae 2a and Section A.7). The favorable cases for the second terms of the disjunctions are resp. counted by $\beta^{\rightarrow}(T,\rho,\sigma)$ and $\beta^{\leftarrow}(T,\rho,\sigma)$ – cf. Formulae 2b and 2c. Delving more into the detail, $\beta^{\rightarrow}(T,\rho,\sigma)$ and $\beta^{\leftarrow}(T,\rho,\sigma)$ do not count the number of times a ρ occurs before at least another ρ preceding the nearest σ (resp., the number of times a σ appears after at least another σ following the nearest ρ). It counts the occurrences of ρ's (σ's) between the two characters, instead. However, the reader can easily verify that the two approaches lead to the same result.

According to the given explanation,

$$1 - \frac{\beta^{\rightarrow}(T,\rho,\sigma) + \delta(T,\rho,\sigma,+\infty)}{\Gamma(T,\rho)} \text{ and } 1 - \frac{\beta^{\leftarrow}(T,\rho,\sigma) + \delta(T,\sigma,\rho,-\infty)}{\Gamma(T,\rho)}$$

are proper Support functions for, resp., $\mathit{AlternateResponse}(\rho,\sigma)$ and $\mathit{AlternatePrecedence}(\rho,\sigma)$.

A.9. *ChainResponse* and *ChainPrecedence*

Given the definition of $\delta(T,\rho,\sigma,d)$ (Formula 2a), we have the following, resp. assigning d the values 1 and -1:

$$\delta(T,\rho,\sigma,1) = \sum_{t \in T} \left| \left\{ i\ :\ t[i] = \rho \ \wedge\ \exists j\ .\ t[j] = \sigma \ \wedge\ j = i + 1 \right\} \right|$$

$$\delta(T,\sigma,\rho,-1) = \sum_{t \in T} \left| \left\{ i\ :\ t[i] = \sigma \ \wedge\ \exists j\ .\ t[j] = \rho \ \wedge\ j = i - 1 \right\} \right|$$

Therefore, comparing the preceding Formulae with Formulae 3h and 3k, it follows that they count the favorable cases for $\mathit{ChainResponse}(\rho, \sigma)$ and $\mathit{ChainPrecedence}(\rho, \sigma)$. The total number of cases for $\mathit{ChainResponse}(\rho, \sigma)$ is given by $\Gamma(T, \rho)$, and the total number of cases for $\mathit{ChainPrecedence}(\rho, \sigma)$ is given by $\Gamma(T, \sigma)$. Thus,

$$\frac{\delta(T, \rho, \sigma, 1)}{\Gamma(T, \rho)} \text{ and } \frac{\delta(T, \sigma, \rho, -1)}{\Gamma(T, \sigma)}$$

are proper Support functions for, resp., $\mathit{ChainResponse}(\rho, \sigma)$ and $\mathit{ChainPrecedence}(\rho, \sigma)$.

A.10. $\mathit{CoExistence}$**,** $\mathit{Succession}$**,** $\mathit{AlternateSuccession}$ **and** $\mathit{ChainSuccession}$

All the Mutual Relation constraints, such as $\mathit{CoExistence}(\rho, \sigma)$, are such that if either ρ or σ occur in the string, then the constraint has an effect on the string. When ρ or σ appear in the string, indeed, $\mathit{CoExistence}(\rho, \sigma)$ imposes that resp. σ or ρ appear at least once in the string as well. $\mathit{RespondedExistence}(\rho, \sigma)$, instead, does not constrain a string where no ρ's appear. This is the reason why the total number of cases for Mutual Relation constraints has to include the occurrences of both ρ and σ:

$$\Gamma(T, \rho) + \Gamma(T, \sigma)$$

Given this, we can consider the unfavorable cases of $\mathit{CoExistence}(\rho, \sigma)$ as the sum of unfavorable cases for $\mathit{RespondedExistence}(\rho, \sigma)$ and $\mathit{RespondedExistence}(\sigma, \rho)$ (see Section A.7), since, as stated in Formulae 1l and 3l, $\mathit{CoExistence}(\rho, \sigma)$ is the conjunction of the two:

$$\delta(T, \rho, \sigma, \pm\infty) + \delta(T, \sigma, \rho, \pm\infty)$$

Therefore,

$$1 - \frac{\delta(T, \rho, \sigma, \pm\infty) + \delta(T, \sigma, \rho, \pm\infty)}{\Gamma(T, \rho) + \Gamma(T, \sigma)}$$

is a proper Support function for $\mathit{CoExistence}(\rho, \sigma)$.

As stated in Formulae 1m and 3m, the unfavorable cases for $\mathit{Succession}(\rho, \sigma)$ are those which can be considered unfavorable to $\mathit{Response}(\rho, \sigma)$ and $\mathit{Precedence}(\rho, \sigma)$. Following the same rationale of the Support function for $\mathit{CoExistence}(\rho, \sigma)$ and referring to Section A.7, we have that

$$1 - \frac{\delta(T, \rho, \sigma, +\infty) + \delta(T, \sigma, \rho, -\infty)}{\Gamma(T, \rho) + \Gamma(T, \sigma)}$$

is a proper Support function for $\mathit{Succession}(\rho, \sigma)$.

For the same reason, though considering the unfavorable cases for $\mathit{AlternateResponse}(\rho, \sigma)$ and $\mathit{AlternatePrecedence}(\rho, \sigma)$, we have that

$$1 - \frac{\beta^{\rightarrow}(T, \rho, \sigma) + \delta(T, \rho, \sigma, +\infty) + \beta^{\leftarrow}(T, \rho, \sigma) + \delta(T, \sigma, \rho, -\infty)}{\Gamma(T, \rho) + \Gamma(T, \sigma)}$$

is a proper Support function for $\mathit{AlternateSuccession}(\rho, \sigma)$ (cf. Formulae 1n, 3n and Section A.8).

$$\frac{\delta(T, \rho, \sigma, 1) + \delta(T, \sigma, \rho, -1)}{\Gamma(T, \rho) + \Gamma(T, \sigma)}$$

is a proper Support function for $\mathit{ChainSuccession}(\rho, \sigma)$ since the numerator counts the favorable cases for $\mathit{ChainResponse}(\rho, \sigma)$ and $\mathit{ChainPrecedence}(\rho, \sigma)$ (cf. Formulae 1o, 3o and Section A.9), i.e., the constraints that $\mathit{ChainSuccession}(\rho, \sigma)$ is the conjunction of. Therefore, there is no need to subtract the result to 1.

A.11. *NotCoExistence*, *NotSuccession* and *NotChainSuccession*

Comparing Formula 3p to the specification of $\delta(T, \rho, \sigma, \pm\infty)$ (Formula 2a) and Formula 3q to the specifications of $\delta(T, \rho, \sigma, +\infty)$ and $\delta(T, \rho, \sigma, -\infty)$ (Formula 2a), it immediately comes to evidence that $\delta(T, \rho, \sigma, \pm\infty) + \delta(T, \sigma, \rho, \pm\infty)$ counts the favorable cases for $\mathit{NotCoExistence}(\rho, \sigma)$, as well as $\delta(T, \rho, \sigma, +\infty) + \delta(T, \sigma, \rho, -\infty)$ counts the favorable cases for $\mathit{NotSuccession}(\rho, \sigma)$. In Section A.9, we derived the specification of $\delta(T, \rho, \sigma, 1)$ and $\delta(T, \sigma, \rho, -1)$. Considering Formula 3r, we have that $\delta(T, \rho, \sigma, 1) + \delta(T, \sigma, \rho, -1)$ counts the unfavorable cases for $\mathit{NotChainSuccession}(\rho, \sigma)$, since:

$$t \models^{\Delta^t, t} \neg \mathit{NotChainSuccession}(\rho, \sigma) \quad \text{iff} \quad \begin{array}{l} (\exists i.\ t[i] = \rho \ \rightarrow\ \exists j.\ t[j] = \sigma \ \wedge\ j = i + 1) \vee \\ (\exists j.\ t[j] = \sigma \ \rightarrow \exists i.\ t[i] = \rho \ \wedge\ j = i + 1) \end{array}$$

Therefore,

$$\frac{\delta(T, \rho, \sigma, \pm\infty) + \delta(T, \sigma, \rho, \pm\infty)}{\Gamma(T, \rho) + \Gamma(T, \sigma)},$$

$$\frac{\delta(T, \rho, \sigma, +\infty) + \delta(T, \sigma, \rho, -\infty)}{\Gamma(T, \rho) + \Gamma(T, \sigma)} \text{ and}$$

$$1 - \frac{\delta(T, \rho, \sigma, 1) + \delta(T, \sigma, \rho, -1)}{\Gamma(T, \rho) + \Gamma(T, \sigma)}$$

are proper Support functions for, resp., $\mathit{NotCoExistence}(\rho, \sigma)$, $\mathit{NotSuccession}(\rho, \sigma)$ and $\mathit{NotChainSuccession}(\rho, \sigma)$.

B. PROOF OF COMPLEXITY RESULTS

Here we present some formal proofs about the complexity of the proposed technique (cf. Sections 4.3 and 4.4).

LEMMA B.1. *The procedure for building the knowledge base of MINERful is (i) linear time w.r.t. the number of strings in the log, (ii) quadratic time w.r.t. the size of strings in the log, (iii) quadratic time w.r.t. the size of the alphabet; therefore, the complexity is* $O(|T| \cdot |t_{max}|^2 \cdot |\Sigma|^2)$.

PROOF. The outermost cycle (line 4) is repeated exactly $|T|$ times, the following inner cycle (line 11) is executed $|t|$ times for each $t \in T$. In the worst case, i.e., assuming that each string is as long as the longest one, it loops $|t_{max}|$ times, where $|t_{max}| = \max_{t \in T} |t|$. Actually, the pair of loops let the instructions be repeated exactly $\sum_{t \in T} |t|$ times, where T is likely to be the most significant part of the input, in terms of size. At line 15, we have a cycle whose number of repetitions grows as new characters are found in the analyzed string. The number of loops depends on the size of the string $|t|$ and on the size of the alphabet Σ at the same time. In fact, we might assume that each character read was not found before in the string. Therefore, as soon as a new character is read, one more loop is made. We might argue that the instructions in the block are executed $1 + 2 + 3 + \cdots + |t|$ times as the cursor in the string moves on. If it were so, loops starting at line 11 and line 15 had run at most

$$\frac{|t| \times (|t| + 1)}{2}$$

times, as in the formula for counting the sum of the first $|t|$ natural numbers. However, the maximum amount of "new" characters is bounded by the characters we can actually have. Therefore, it runs at most $1 + 2 + 3 + \cdots + |\Sigma|$ times assuming the worst case, i.e., all the characters of Σ in every string. Therefore, we have to subtract the number of loops that are not executed due to the limitation of the alphabet size (if the alphabet size is smaller than the size of the strings, which is likely). Let:

$$\Delta_{|t|,|\Sigma|} = |t| - |\Sigma|$$

Thus, the number of loops is equal to:

$$\frac{|t| \times (|t| + 1)}{2} - \frac{\Delta_{|t|,|\Sigma|} \times (\Delta_{|t|,|\Sigma|} + 1)}{2} \cdot \Theta(\Delta_{|t|,|\Sigma|} - 1)$$

where $\Theta(x)$ is the Heaviside step function. The function returns either 1 or 0. Specifically, it returns 1 if $\Delta_{|t|,|\Sigma|} > 1$, i.e., if $|t| \geqslant |\Sigma| + 1$. Otherwise, its value is 0. Therefore, if $|t| < |\Sigma| + 1$, the second term of the above equation is removed. Thus, if $|t| > |\Sigma| + 1$ we can reduce the preceding equation to the following:

$$\frac{2|\Sigma||t| - |\Sigma|^2 + |\Sigma|}{2} = \frac{2\Delta_{|t|,|\Sigma|}|\Sigma| + |\Sigma|}{2} \leqslant |\Sigma||t|$$

Depending on the condition at line 16, the algorithm enters one of the two innermost loops, one starting at line 17, the other at line 26.

The first is executed exactly $|\Sigma| - 1$ times, regardless of the outer cycles. The second loop cycles as many times as a character is repeated along the string. If we had strings composed by concatenations of the same character (the worst case for the second cycle) after a prefix comprising the whole alphabet (the worst case for the first cycle), this would lead to $|t|$ loops, asymptotically.

The instructions in the bodies of the loops are readings and writings in memory.[10] Therefore, they are deemed as irrelevant, for what the complexity of the algorithm is concerned.

Summing up this computation analysis, we have that the worst-case complexity of the algorithm is

$$O\left(\underbrace{|T|}_{\text{loop at 4}}\ \underbrace{|t_{max}||\Sigma|}_{\text{loops at 11 and 15}}\left(\underbrace{|\Sigma|}_{\text{loop at 17}} + \underbrace{|t_{max}|}_{\text{loop at 26}}\right)\right)$$

□

LEMMA B.2. *The procedure for discovering the constraints of processes out of MINERful knowledge base, along with their Support, Confidence Level and Interest Factor is (i) quadratic in time w.r.t. the size of the alphabet, (ii) linear in time w.r.t the number of constraint templates, which is fixed and equal to* 18 *(thus, constant); therefore, the complexity is* $O(|\Sigma|^2)$.

PROOF. Algorithm 4 executes two nested cycles: the outer from line 3 to line 28, the inner, from line 11 to line 26. Inside them both, only mathematical operations are performed. The number of such operations is fixed and amounts to the quantity of constraint templates that MINERful discovers. In addition, calculi are performed on top of a fixed number of entries in MINERfulKB, depending on the parameters for functions in Table VIII. Both loops are executed for each character in the alphabet. The complexity of CALCMETRICSFORCONSTRAINTS is $O(|\Sigma|^2)$, then. The number of entries in bag $\mathcal{B}^+$ is $O(|\Sigma|^2)$ as well.

Algorithm 5 begins with the cloning of that bag (line 2). Therefore, it is a $O(|\Sigma|^2)$ procedure by itself. Afterwards, each element in the clone of $\mathcal{B}^+$ is subject to some checks and modifications. The cycle from line 3 to line 34 iterates $O(|\Sigma|^2)$ times. Remembering what stated in the explanation of the procedure, the functions invoked ($hasParent$, $getParent$, etc.) can be considered as invocations of an oracle, being based on the knowledge about semantics of the constraint templates and not on MINERfulKB. The loop from line 7 to line 9 is executed, at most, a limited number of times. Such limit does not depend on the input, but on the hierarchy of subsumptions in the constraint templates. By visual inspection of Figure 1 or Table IX, it is evident that such limit is fixed and less than or equal to 4. Thus, CLEANOUTPUT is $O(|\Sigma|^2)$ too.

Finally, Algorithm 6's complexity is affected by the iteration on the elements of $\mathcal{B}^+$. Hence, it is $O(|\Sigma|^2)$ like the other two procedures. Therefore, DISCOVERCONSTRAINTS is a $O(|\Sigma|^2)$ algorithm. □

THEOREM B.3. *The MINERful algorithm is (i) linear time w.r.t. the number of strings in log, (ii) quadratic time w.r.t. the size of strings in the log, (iii) quadratic time w.r.t. the size of the alphabet; therefore, the complexity is* $O(|T| \cdot |t_{max}|^2 \cdot |\Sigma_T|^2)$.

PROOF. Directly follows from Lemmata B.1 and B.2. □

[10]The computational effort for searching a datum in temporary data structures such as N can be disregarded. This is because we can take advantage of hashing functions, which make the read and write operations $0(1)$. The usage of hashing functions is doable as: *(i)* we can exploit the alphanumeric ordering function for sorting the pairs of characters, and *(ii)* the alphabet of characters is known a priori.

C. TECHNICAL DETAILS ON THE ANALYSIS OF DISCOVERED PROCESSES

In this Section of the Appendix, we provide the full tables reporting the results of comparative tests conducted on MINERful and Declare Maps Miner [Maggi 2013]. The input for experiments were a synthetic log and a real-life log extracted from email conversations [Di Ciccio and Mecella 2013a]. Results are listed in Table XVII and XVIII, respectively. In order to ease the reading of those Tables, here we report a legend.

Support %. The threshold for the Support parameter of MINERful and Declare Maps Miner, in percentage.
Confidence. The threshold for the Confidence parameter of MINERful and Declare Maps Miner.
Interest Factor. The threshold for the Interest Factor parameter of MINERful and Declare Maps Miner.
Constraints. The number of constraints in the mined process.
Cn's per activity. The average number of constraints per activity, in the mined process.
Fitness. The level of Fitness assessed for the discovered process, as computed by Declare Replayer [Buijs et al. 2012; de Leoni et al. 2012].
Meet/min %. The Meet/min coefficient for the set of MINERful and Declare Maps Miner's discovered constraints, in percentage [Deza and Deza 2006].
Jaccard %. The Jaccard coefficient for the sets of MINERful and Declare Maps Miner's discovered constraints, in percentage [Deza and Deza 2006].
Sørensen-Dice %. The Sørensen-Dice coefficient for the sets of MINERful and Declare Maps Miner's discovered constraints, in percentage [Deza and Deza 2006].

In particular, naming MINERful and Declare Maps Miner's discovered sets as, resp., $\mathcal{B}_M$ and $\mathcal{B}_D$, we have that:

— Jaccard coefficient is defined as $\frac{|\mathcal{B}_M \cap \mathcal{B}_D|}{|\mathcal{B}_M \cup \mathcal{B}_D|}$,

— Sørensen-Dice coefficient is defined as $\frac{2 \cdot |\mathcal{B}_M \cap \mathcal{B}_D|}{|\mathcal{B}_M| + |\mathcal{B}_D|}$,

— Meet/min coefficient is defined as $\frac{|\mathcal{B}_M \cap \mathcal{B}_D|}{\min \{|\mathcal{B}_M|, |\mathcal{B}_D|\}}$.

For Confidence and Interest Factor thresholds, four symbols denote their value, namely:

|. The lower end of the range.
I. The rounded-up value for the first third of the range.
II. The rounded-up value for the second third of the range.
III. The upper end of the range.

The range is fixed on the basis of minimum and maximum possible values of Confidence (resp. Interest Factor) that actively altered the resulting discovered control flow. The range changes according to *(i)* the tool (either MINERful or Declare Maps Miner), *(ii)* the related Support threshold set, and *(iii)* whether it refers to Confidence or Interest Factor. The range for both Confidence and Interest Factor is fixed in MINERful and equal to $[0.0, 1.0]$, regardless of Support threshold. For Declare Maps Miner instead, Confidence can vary from 0 to 100, whereas Interest Factor possibly ranges from 0 to a maximum value that changes according to the process. Furthermore, it may exceed 100. However, neither Confidence nor Interest Factor affect the result if their respective thresholds are lower than Support threshold. Thus, their range has been rescaled from the given Support threshold to the maximum allowed. For instance, when Support threshold is assigned 94 for Declare Maps Miner, the minimum value for Confidence and Interest Factor is considered 94, as 94 and every other lower value do not affect the result. Therefore, we considered a range for Confidence equal to [94, 100] and rescaled the rounded-up value for the first and second third of the range accordingly: respectively, 96 and 98. The maximum Interest Factor for the email log was 201. Then, we fixed the range to [94, 201] and assigned 130 and 166 to resp. the rounded-up first and second third of the range. Because of this, when Support threshold is equal to 100, Confidence can be only equal to 100, whereas Interest Factor ranges from 100 to 201. As said, the rescaling is not necessary for MINERful. See Tables XV and XVI for further details.

[redacted]		**Decl.M.M.**		**MINERful**	
Support %	Level	Confidence %	Interest Factor	Confidence %	Interest Factor %
91	|	91	91	0	0
	I	94	98	34	34
	II	97	105	37	37
	III	100	111	100	100
94	|	94	94	0	0
	I	96	100	34	34
	II	98	106	37	37
	III	100	111	100	100
97	|	97	97	0	0
	I	98	102	34	34
	II	99	107	37	37
	III	100	111	100	100
100	|	100	100	0	0
	I	100	104	34	34
	II	100	107	37	37
	III	100	111	100	100

Table XV: Confidence and Interest Factor values chosen for thresholds in the experiments, when mining the synthetic log.

		Decl.M.M.		**MINERful**	
Support %	Level	Confidence %	Interest Factor	Confidence %	Interest Factor %
91	|	91	91	0	0
	I	94	128	34	34
	II	97	165	37	37
	III	100	201	100	100
94	|	94	94	0	0
	I	96	130	34	34
	II	98	166	37	37
	III	100	201	100	100
97	|	97	97	0	0
	I	98	132	34	34
	II	99	167	37	37
	III	100	201	100	100
100	|	100	100	0	0
	I	100	134	34	34
	II	100	168	37	37
	III	100	201	100	100

Table XVI: Confidence and Interest Factor values chosen for thresholds in the experiments, when mining the email log.

			MINERful			Declare M. Miner			Similarity		
Support %	Confidence	Interest Factor	Constraints	Cn's per activity	Fitness	Constraints	Cn's per activity	Fitness	Meet/min %	Jaccard %	Sørensen-Dice %
91	ǀ	ǀ	83	10.375	0.926	24	3.000	0.992	95.833	27.381	42.991
91	ǀ	**I**	83	10.375	0.926	23	2.875	0.996	86.957	23.256	37.736
91	ǀ	**II**	50	6.250	0.977	8	1.000	0.996	87.500	13.725	24.138
91	ǀ	**III**	7	0.875	1.000	8	1.000	0.996	42.857	25.000	40.000
91	**I**	ǀ	83	10.375	0.926	23	2.875	0.994	95.652	26.190	41.509
91	**II**	ǀ	50	6.250	0.977	21	2.625	0.996	85.714	33.962	50.704
91	**III**	ǀ	7	0.875	1.000	19	2.375	0.996	85.714	30.000	46.154
94	ǀ	ǀ	52	6.500	0.979	19	2.375	0.996	94.737	33.962	50.704
94	ǀ	**I**	52	6.500	0.979	17	2.125	0.999	88.235	27.778	43.478
94	ǀ	**II**	37	4.625	0.987	6	0.750	0.999	83.333	13.158	23.256
94	ǀ	**III**	7	0.875	1.000	6	0.750	0.999	50.000	30.000	46.154
94	**I**	ǀ	52	6.500	0.979	17	2.125	0.999	94.118	30.189	46.377
94	**II**	ǀ	37	4.625	0.987	15	1.875	0.999	86.667	33.333	50.000
94	**III**	ǀ	7	0.875	1.000	15	1.875	0.999	85.714	37.500	54.545
97	ǀ	ǀ	40	5.000	0.996	17	2.125	0.999	94.118	39.024	56.140
97	ǀ	**I**	40	5.000	0.996	6	0.750	0.999	83.333	12.195	21.739
97	ǀ	**II**	32	4.000	0.996	6	0.750	0.999	83.333	15.152	26.316
97	ǀ	**III**	7	0.875	1.000	6	0.750	0.999	50.000	30.000	46.154
97	**I**	ǀ	40	5.000	0.996	15	1.875	0.999	100.000	37.500	54.545
97	**II**	ǀ	32	4.000	0.996	15	1.875	0.999	86.667	38.235	55.319
97	**III**	ǀ	7	0.875	1.000	15	1.875	0.999	85.714	37.500	54.545
100	ǀ	ǀ	27	3.375	1.000	13	1.625	1.000	100.000	48.148	65.000
100	ǀ	**I**	27	3.375	1.000	4	0.500	1.000	75.000	10.714	19.355
100	ǀ	**II**	21	2.625	1.000	4	0.500	1.000	75.000	13.636	24.000
100	ǀ	**III**	7	0.875	1.000	4	0.500	1.000	75.000	37.500	54.545
100	**I**	ǀ	27	3.375	1.000	13	1.625	1.000	100.000	48.148	65.000
100	**II**	ǀ	21	2.625	1.000	13	1.625	1.000	84.615	47.826	64.706
100	**III**	ǀ	7	0.875	1.000	13	1.625	1.000	85.714	42.857	60.000

Table XVII: Discovered constraints and Fitness of processes mined out of a synthetic log by resp. MINERful and Declare Maps Miner.

			MINERful			Declare Maps Miner			Similarity		
Support %	Confidence	Interest Factor	Constraints	Cn's per activity	Fitness	Constraints	Cn's per activity	Fitness	Meet/min %	Jaccard %	Sørensen-Dice %
91	ǀ	ǀ	188	14.462	0.822	44	3.385	1.000	100.000	23.404	37.931
91	ǀ	**I**	113	8.692	0.923	5	0.385	1.000	40.000	1.724	3.390
91	ǀ	**II**	41	3.154	0.967	5	0.385	1.000	40.000	4.545	8.696
91	ǀ	**III**	5	0.385	1.000	5	0.385	1.000	40.000	25.000	40.000
91	**I**	ǀ	116	8.923	0.859	44	3.385	1.000	68.182	23.077	37.500
91	**II**	ǀ	97	7.462	0.928	44	3.385	1.000	59.091	22.609	36.879
91	**III**	ǀ	7	0.538	1.000	44	3.385	1.000	57.143	8.511	15.686
94	ǀ	ǀ	171	13.154	0.965	44	3.385	1.000	100.000	25.731	40.930
94	ǀ	**I**	96	7.385	0.977	5	0.385	1.000	40.000	2.020	3.960
94	ǀ	**II**	32	2.462	0.992	5	0.385	1.000	40.000	5.714	10.811
94	ǀ	**III**	5	0.385	1.000	5	0.385	1.000	40.000	25.000	40.000
94	**I**	ǀ	99	7.615	0.977	44	3.385	1.000	68.182	26.549	41.958
94	**II**	ǀ	82	6.308	0.977	44	3.385	1.000	59.091	26.000	41.270
94	**III**	ǀ	7	0.538	1.000	44	3.385	1.000	57.143	8.511	15.686
97	ǀ	ǀ	163	12.538	0.995	44	3.385	1.000	100.000	26.994	42.512
97	ǀ	**I**	89	6.846	0.995	5	0.385	1.000	40.000	2.174	4.255
97	ǀ	**II**	29	2.231	0.995	5	0.385	1.000	40.000	6.250	11.765
97	ǀ	**III**	5	0.385	1.000	5	0.385	1.000	40.000	25.000	40.000
97	**I**	ǀ	92	7.077	0.995	44	3.385	1.000	68.182	28.302	44.118
97	**II**	ǀ	75	5.769	0.995	44	3.385	1.000	59.091	27.957	43.697
97	**III**	ǀ	7	0.538	1.000	44	3.385	1.000	57.143	8.511	15.686
100	ǀ	ǀ	161	12.385	1.000	44	3.385	1.000	100.000	27.329	42.927
100	ǀ	**I**	87	6.692	1.000	5	0.385	1.000	40.000	2.222	4.348
100	ǀ	**II**	27	2.077	1.000	5	0.385	1.000	40.000	6.667	12.500
100	ǀ	**III**	5	0.385	1.000	5	0.385	1.000	40.000	25.000	40.000
100	**I**	ǀ	90	6.923	1.000	44	3.385	1.000	68.182	28.846	44.776
100	**II**	ǀ	73	5.615	1.000	44	3.385	1.000	59.091	28.571	44.444
100	**III**	ǀ	7	0.538	1.000	44	3.385	1.000	57.143	8.511	15.686

Table XVIII: Discovered constraints and Fitness of processes mined out of the email-based log by resp. MINERful and Declare Maps Miner.

References

Di Ciccio, Claudio and Massimo Mecella (2015). "On the Discovery of Declarative Control Flows for Artful Processes". In: *ACM Trans. Manage. Inf. Syst.* 5.4, 24:1–24:37. ISSN: 2158-656X. DOI: 10.1145/2629447.

BibTeX

```
@Article{         DiCiccio.Mecella/ACMTMIS2015:DiscoveryDeclarativeControl,
  author        = {Di Ciccio, Claudio and Mecella, Massimo},
  title         = {On the Discovery of Declarative Control Flows for Artful
                  Processes},
  journal       = {ACM Trans. Manage. Inf. Syst.},
  year          = {2015},
  volume        = {5},
  number        = {4},
  pages         = {24:1--24:37},
  issn          = {2158-656X},
  doi           = {10.1145/2629447},
  keywords      = {MailOfMine, process mining, artful processes, control-flow
                  discovery, declarative process model},
  publisher     = {ACM}
}
```