Mining Constraints for Artful Processes*

Claudio Di Ciccio and Massimo Mecella

Sapienza – Università di Roma
Dipartimento di Ingegneria Informatica, Automatica e Gestionale Antonio Ruberti
{cdc,mecella}@dis.uniroma1.it

Abstract. Artful processes are informal processes typically carried out by those people whose work is mental rather than physical (managers, professors, researchers, engineers, etc.), the so called "knowledge workers". MailOfMine is a tool, the aim of which is to automatically build, on top of a collection of email messages, a set of workflow models that represent the artful processes laying behind the knowledge workers activities. After an outline of the approach and the tool, this paper focuses on the mining algorithm, able to efficiently compute the set of constraints describing the artful process. Finally, an experimental evaluation of it is reported.

Key words: process mining, artful process, declarative workflow, email

1 Introduction

For a long time, formal business processes (e.g., the ones of public administrations, of insurance/financial institutions, etc.) have been the main subject of workflow related research. Informal processes, a.k.a. "artful processes", are conversely carried out by those people whose work is mental rather than physical (managers, professors, researchers, engineers, etc.), the so called "knowledge workers" [30]. In contrast to business processes that are formal and standardized, often informal processes are not even written down, let alone defined formally, and can vary from person to person even when those involved are pursuing the same objective. Knowledge workers create informal processes "on the fly" to cope with many of the situations that arise in their daily work. Though informal processes are frequently repeated, they are not exactly reproducible even by their originators – since they are not written down – and can not be easily shared either. Their outcomes and their information exchanges are done very often by means of email conversations, which are a fast, reliable, permanent way of keeping track of the activities that they fulfill.

Understanding artful processes involving knowledge workers is becoming crucial in many scenarios. Here we mention some of them:

– *personal information management (PIM)*, i.e., how to organize one's own activities, contacts, etc. through the use of software on laptops and smart

* This work has been partly supported by Sapienza – Università di Roma through the grants FARI 2010 and TESTMED, and by the EU Commission through the FP7 project Smart Vortex. The authors would like also to thank Monica Scannapieco and Diego Zardetto for useful insights and discussions.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by doi: 10.1007/978-3-642-30359-3_2

devices (iPhones/iPads, smartphones, tablets). Here, inferring artful processes in which a person is involved allows the system to be proactive and thus drive the user through its own tasks (on the basis of the past) [11, 30];
- *information warfare*, especially in supporting anti-crime intelligence agencies: let us suppose that a government bureau is able to access the email account of a suspected person. People planning a crime or an act out of law are used to speak a language of their own to express duties and next moves, where meanings may not match with the common sense. Thus, a system should build the processes that lay behind their communications anyway, exposing the activities and the role of the actors. At that point, translating the sense of misused words becomes an easier task for investigators, and allows inferring the criminal activities of the suspected person(s);
- *enterprise engineering*: in design and engineering, it is important to preserve more than just the actual documents making up the product data. Preserving the "soft knowledge" of the overall process (the so-called product life-cycle) is of critical importance for knowledge-heavy industries. Hence, the idea here is to take to the future not only the designs, but also the knowledge about processes, decision making, and people involved [20, 28].

The objective of the MailOfMine approach, firstly introduced in [16], is to automatically discover, on top of a collection of email messages, a set of workflow models that represent the artful processes laying behind the knowledge workers' activities. In [14], we describe our ideas on how to effectively show the users such models. In this paper, we outline the general approach of the mining algorithm, able to infer the constraints that specify the workflow altogether, out of the given execution traces. Then we present some experiments, showing the validity and efficiency of the technique.

The work presented here is related to the so called *process mining*, a.k.a. *workflow mining* [3], that is the set of techniques allowing the extraction of structured process descriptions from a set of recorded real executions (stored in the *event log*s). ProM [4] is one of the most used plug-in based software environment for implementing workflow mining techniques. Most of the mainstream process mining tools model processes as Workflow Nets (WFNs – see [2]), explicitly designed to represent the control-flow dimension of a workflow. From [7] onwards, many techniques have been proposed, in order to address specific issues: pure algorithmic (e.g., α algorithm [6] and its evolution α^{++} [32]), heuristic (e.g., [31]), genetic (e.g., [22]). Indeed, heuristic and genetic algorithms have been introduced to cope with noise, that the pure algorithmic techniques were not able to manage. A very smart extension to the previous research work has been recently achieved by the two-steps algorithm proposed in [1].

The need for flexibility in the definition of processes leads to an alternative to the classical "imperative": the "declarative" approach. Rather than using a procedural language for expressing the allowed sequences of activities, 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 them. Such constraints, in DecSerFlow [5] and ConDec [24] (now named Declare [25, 26]), are formulations of Linear Temporal Logic and have a graphical representation

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-642-30359-3_2

as well. [21] outlines an algorithm for mining Declare processes, implemented in ProM. The technique is based on [18, 19, 26], for the translation of Declare constraints into automata, and [33], for the optimization of such task.

[12] described the usage of inductive logic programming techniques to mine models expressed as a $\mathcal{S}$CIFF [9] theory, finally translated to the ConDec notation.

The technique introduced in this paper differs from both [12] and [21] in that it does not directly verify the candidate constraints over the whole set of traces in input. It prepares an ad-hoc knowledge base of its own, so to further analyze the response to specific queries.

We believe that the declaration of collaborative workflows constraints can be expressed by means of regular expressions, rather than LTL formulae: regular expressions express finite languages (i.e., processes with finite traces, where the number of enacted tasks is limited). LTL formulae are thought to be used for verifying properties over semi-infinite runs instead. On the contrary, human processes have an end, other than a starting point. We envision the process schemes like grammars describing the language spoken by collaborative entities in terms of activities, thus being more related to formal languages rather than temporal logic. Finally, we can exploit the scientific results from the literature in the pattern matching (e.g., [17, 8, 29]).

The remainder of this paper is organized as follows: Section 2 outlines the overall approach of MailOfMine, then Section 3 describes the process model we adopt in our mining approach. Section 4 describes the relevant features of the mining technique; the interested reader can find more details, formal definitions, proofs and technicalities in [15]. Section 5 presents an extensive validation of the technique, and finally Section 6 draws some concluding remarks, outlining future activities.

2 The MailOfMine Approach

The MailOfMine approach (and the tool we are currently developing) adopts a modular architecture, the components of which allow to incrementally refine the mining process; before briefly presenting it, we need to introduce some basic concepts needed in the following. The details of the architecture, as well as the formal definitions of concepts, are presented in [16].

An *actor* is the subject directly or indirectly taking part in the progress of a work. A *task* is an elementary unit of work. Each task is connected to *(i)* its expected *duration*, *(ii)* zero or more *outcome*s, *(iii)* one or more *actor*s. An *activity* is a collection of tasks or (recursively) other activities. A *key part* is each *unique* piece of text belonging to the email messages exchanged in a communication trace. For instance, signatures put in the footer by the senders of the email messages, quotations of previous email messages used in replies, etc., are all examples of redundant information that may appear more than once in a thread: they have to be filtered out by means of the key part concept, then. On the other hand, any piece of text, previously unread during the automated analysis of the thread, is interpreted as a key part, since it is supposed to add some

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-642-30359-3_2

information to the discussion (and thus, to the underlying enacted process). An *indicium* is any communication trace, or part of it, attesting the likely execution of a task (task indicium) or an activity (activity indicium). In other words, it is any text where you can find an evidence that a task (or an activity) has been performed. A *process scheme* (or *process* for short) is a semi-structured set of activities, where the semi-structuring connective tissue is represented by the set of constraints stating the interleaving rules among activities or tasks. Constraints do not force the process instance to follow a tight sequence, but rather leave it the flexibility to follow different paths, while performing the execution, though respecting a set of rules that avoid illegal or non-consistent states. We argue that each constraint is expressible through regular grammars. Regular grammars are recognizable through Finite State Automata (FSA) [13] (either deterministic or non-deterministic [27]). The FSA recognizing the correct traces for processes (i.e., accepting the valid strings) is the intersection of all the FSAs composing the set of constraints.

Initially, we need to extract email messages from the given archive(s), so to analyze their raw data, including senders, recipients, headers, base-64-encoded attachments, etc., and extract the relevant information, in order to further elaborate and store it into a database. On top of the latter, all the subsequent steps are carried out. The first of them is the clustering of retrieved messages into extended communication threads, i.e., flows of messages which are related to each other. The considered technique, which has been described in [16], is based not only on the Subject field (e.g., looking at "Fwd:" or "Re:" prefixes) or SMTP Header fields (e.g., reading the "In-Reply-To" field), but on the application of a more complex object matching decision method. Once the communication threads are recognized, we can assume them all as possible proofs (namely, indicia) of an enacted activity.

Afterwards, email messages are cleaned up from signatures and quotations citing some text already written in another message within the thread, by the usage of a combination of the techniques described in [10] and [23]. The key parts are the text remaining in bodies and subjects once such filtering operation has been performed. MailOfMine can thus build the activity indicia as the concatenation of all the key parts belonging to the messages of a communication thread. Then, the clustering algorithm is used again, this time to identify the matches between activity indicia. By taking into account the activities set and the key parts, the clustering algorithm checks for matching key parts: those are considered task indicia. At this point, MailOfMine starts searching for execution constraints between tasks. Presenting how this technique works, and validating it, is the aim of this paper; it will be detailed in Section 4.

3 The Process Model

As previously introduced, a *process scheme* (or *process* for short) is a semi-structured set of activities, where the semi-structuring connective tissue is represented by the set of constraints stating the interleaving rules among activities or tasks. Constraints do not force the tasks to follow a tight sequence, but rather

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-642-30359-3_2

leave them the flexibility to follow different paths, though respecting a set of rules that avoid illegal or non-consistent states in the execution.

Here, we adopt the Declare [21] taxonomy of constraints as the basic language for defining artful processes in a declarative way. But whereas in Declare constraints are translated into LTL formulas, we express each constraint through regular expressions, as discussed in Section 3 of [15].

The $Existence(m, a)$ constraint imposes the a character to appear at least m times in the trace. The $Absence(n, a)$ constraint holds if a occurs at most $n - 1$ times in the trace. $Init(a)$ makes each trace start with a. $RespondedExistence(a, b)$ holds if, whenever a is read, b was already read or is going to be read (i.e., no matter if before or afterwards). Instead, $Reponse(a, b)$ enforces it by forcing a b to appear after a, if a was read. $Precedence(a, b)$ forces b to occur after a as well, but the condition to be verified is that b was read - namely, you can not have any b if you did not read an a before. $AlternateResponse(a, b)$ and $AlternatePrecedence(a, b)$ both strengthen respectively $Response(a, b)$ and $Precedence(a, b)$ by stating that *each* a (b) must be followed (preceded) by at least one occurrence of b (a). The "alternation" is in that you can not have two a (b) in a row before b (a). $ChainResponse(a, b)$ and $ChainPrecedence(a, b)$, in turn, specialize $AlternateResponse(a, b)$ and $AlternatePrecedence(a, b)$, both declaring that no other character can occur between a and b. The difference between the two is in that the former is verified for each occurrence of a, the latter for each occurrence of b. $CoExistence(a, b)$ holds if both $RespondedExistence(a, b)$ and $RespondedExistence(b, a)$ hold. $Succession(a, b)$ is valid if $Response(a, b)$ and $Precedence(a, b)$ are verified. The same holds with $AlternateSuccession(a, b)$, equivalent to the conjunction of $AlternateResponse(a, b)$ and $AlternatePrecedence(a, b)$, and with $ChainSuccession(a, b)$, with respect to $ChainResponse(a, b)$ and $ChainPrecedence(a, b)$. $NotChainSuccession(a, b)$ expresses the impossibility for b to occur immediately after a. $NotSuccession(a, b)$ generalizes the previous by imposing that, if a is read, no other b can be read until the end of the trace. $NotCoExistence(a, b)$ is even more restrictive: if a appears, not any b can be in the same trace.

In addition to the above constraints, we introduced some new ones (cf. Section 3 of [15]). Taking inspiration from the relational data model cardinality constraints, we call *(i)* $Participation(a)$ the $Existence(m, a)$ constraint for $m = 1$ and *(ii)* $Unique(a)$ the $Absence(n, a)$ constraint for $n = 2$, since the former states that a must appear at least once in the trace, whereas the latter causes a to occur no more than once. $End(a)$ is the dual of $Init(a)$, in the sense that it constrain each trace to end with a. Beware that $End(a)$ would be clueless in a LTL interpretation, since LTL is thought to express temporal logic formulae over infinite traces. On the contrary, it makes perfectly sense to have a concluding task for a finite process, expressed by means of a regular automaton like the one underlying any regular expression.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-642-30359-3_2

3.1 An Example

Here we outline a brief example. Let us suppose to have an email archive, containing various process instances indicia, and to focus specifically on the planning of a new meeting for a research project. We suppose to execute the overall MailOfMine technique, and we report the possible result, starting from the list of tasks in activities (Process Description 1). Then we consider the tasks of the "Agenda" activity only.

Process Description 1 Activities and tasks list

Activity: ⟨*Agenda*⟩
  Task: p ("*p*roposeAgenda"): Productive
    Actors: {*You*: Contributor, *Community*: Spectator}
    Duration: 4 dd.
  Task: r ("*r*equestAgenda"): Clarifying
    Actors: {*Participant*: Contributor, *Community*: Spectator}
    Duration: ⊥
  Task: c ("*c*ommentAgenda"): Clarifying
    Actors: {*Participant*: Contributor, *Community*: Spectator}
    Duration: ⊥
  Task: n ("*co*nfirmAgenda"): Productive
    Actors: {*You*: Contributor, *Community*: Spectator}
    Duration: 2 dd.

We suppose that a final agenda will be committed ("confirmAgenda" – n) after that requests for a new proposal ("requestAgenda" – r), proposals themselves ("proposeAgenda" – p) and comments ("commentAgenda" – c) have been circulated.

Some terms used in the example, e.g., *You*, Contributor, Community, Productive, Clarifying, etc. refer to the classification MailOfMine operates on messages during the mining phases, and are explained in [16]. For the purposes of this paper, the reader can skip them. The aforementioned tasks and activities are bound to the following constraints. We start with the *existence* constraints of Process Description 2, each focusing on a single task.

Process Description 2 Existence constraints on the example tasks and activities

Activity: ⟨*Agenda*⟩
  Task: "*co*nfirmAgenda", n: *Participation*(n), *Unique*(n), *End*(n)

In Process Description 3 we report the *relation* constraints, namely holding between couple of tasks.

Process Description 3 Relation constraints on the example tasks and activities

Activity: ⟨*Agenda*⟩
  response(r, p)
  respondedExistence(c, p)
  succession(p, n)

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-642-30359-3_2

4 The MINERful Algorithm

MINERful is the algorithm for mining declarative constraints out of activities' traces. The previous stages of the MailOfMine approach allow tasks to be abstracted like characters appearing over finite strings, which, in turn, represent process traces. Thus, MINERful works on collections of finite strings, actually. Therefore, it solves the more general problem of finding a specific set of regular patterns out of a number of strings. We will interchangeably use the terms "task" and "character", as well as "trace" and "string", then.

MINERful is based on the concept of *MINERfulKB*: it holds all of the useful information extracted from the given traces and tailored to the further discovery of constraints that possibly lay behind. The first step of MINERful is to synthesize such a matrix, actually, in order to easily mine the declarative model afterwards. The details of the structure of MINERfulKB, as well as the formal definitions related to it, are omitted for sake of space. They can be found in [15]. The MINERfulKB is composed by MINERful interplays and MINERful ownplays.

The *MINERful interplay* is referred to a couple of characters, the first considered as the *pivot*, ρ, and latter considered as the *searched*, σ. For sake of simplicity, examples will consider a as the pivot ρ and b as the searched σ, over an alphabet $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. The MINERful interplay consists of *(i)* a function $\delta_{\rho,\sigma}$, mapping a distance between ρ and σ to the number of cases they appeared at that distance in a string [2], *(ii)* a scalar $b^{\rightarrow}_{\rho,\sigma}$, the *in-between onwards appearances* counter, and *(iii)* a scalar $b^{\leftarrow}_{\rho,\sigma}$, the *in-between backwards appearances* counter. As examples, $\delta_{\rho,\sigma}(2) = 10$ means that we have the evidence of a searched σ appearing 2 characters after the pivot ρ (like in cacbcc) over 10 cases; $b^{\rightarrow}_{\rho,\sigma} = 2$ means that the pivot ρ appeared 2 times between the preceding occurrence of ρ and the following first occurrence of the searched σ (as in the substring accaacb); and $b^{\leftarrow}_{\rho,\sigma} = 3$ means that the pivot ρ appeared 3 times between the following occurrence of ρ and the preceding first occurrence of the searched σ, as in the substring bcacaaca. The MINERful interplay expresses a local view on two characters, measuring the *distances* and the *alternations* between the first and the second one. Thus, one should focus on substrings, starting from the pivot ρ and ending in the searched σ (or viceversa, in case of negative distances). Such substrings are not necessarily related to the *first* occurrence of ρ in the string: *any* ρ is the initial (final) character for a following (preceding) substring ending in (starting from) σ, each separately analyzed.

The *MINERful ownplay* refers to one character at a time (i.e., the pivot ρ over itself). It is composed by *(i)* a function γ_ρ, denoting the total amount of appearances per string, *(ii)* a scalar g^i_ρ, namely the number of strings where the

[2] A positive distance represents the number of characters between ρ and the *following* σ, whereas a negative one is for a *preceding* σ. With a slight abuse of notation, we consider $\delta_{\rho,\sigma}(+\infty)$ and $\delta_{\rho,\sigma}(-\infty)$ to denote the number of cases in which the searched σ, respectively, did not appear in a string *after* the pivot ρ, or did not appear in a string *before* ρ. $\delta_{\rho,\sigma}(0)$ counts the number of cases in which the searched σ did not appear nor before neither after ρ.

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-642-30359-3_2

pivot ρ appeared as the *initial* one, and *(iii)* a scalar g^l_ρ, namely the number of strings where the pivot ρ appeared as the *last* one.

For instance, suppose to have the string aabbac, being a the pivot. Then we have for $\delta_{\mathsf{a},\cdot}$, $b^{\rightarrow}_{\mathsf{a},\cdot}$ and $b^{\leftarrow}_{\mathsf{a},\cdot}$ the values reported in Table 1, whereas $\gamma_{\mathsf{a}} = \left\{ \begin{matrix} \langle 3,1\rangle \\ \langle x,0\rangle \ \forall x \in \mathbb{N} \setminus \{3\} \end{matrix} \right\}$, $g^i_{\mathsf{a}} = 1$, and $g^l_{\mathsf{a}} = 0$.

	$-\infty$	$\cdots$	-4	-3	-2	-1	0	$+1$	$+2$	$+3$	$+4$	$+5$	$\cdots$	$+\infty$	
$\delta_{\mathsf{a},\mathsf{a}}$	0	0	1	1	0	1	0	1	0	1	1	0	0	0	$b^{\rightarrow}_{\mathsf{a},\mathsf{a}} = 0;\ b^{\leftarrow}_{\mathsf{a},\mathsf{a}} = 0$
$\delta_{\mathsf{a},\mathsf{b}}$	2	0	0	0	1	1	0	1	2	1	0	0	0	0	$b^{\rightarrow}_{\mathsf{a},\mathsf{b}} = 1;\ b^{\leftarrow}_{\mathsf{a},\mathsf{b}} = 0$
$\delta_{\mathsf{a},\mathsf{c}}$	3	0	0	0	0	0	0	1	0	0	1	1	0	0	$b^{\rightarrow}_{\mathsf{a},\mathsf{c}} = 2;\ b^{\leftarrow}_{\mathsf{a},\mathsf{c}} = 0$

Table 1. Examples of statistical values stored in MINERfulKB

The initial step of our technique is the construction of the MINERfulKB, having as the input a bag of strings T and an alphabet Σ_T. The algorithm in charge of it is called twice, one onwards, one backwards, i.e., reading the string from left to the right and viceversa (according to the Western Latin standard). The algorithm is designed to be completely on-line, i.e., it updates the MINERfulKB as new strings occur and as new characters in the string are read, with no need to go back on previous data in the end. The MINERfulKB is designed to be tailored to the further reasoning for constraints discovery. Thus this latter step becomes easier and faster, than analyzing it directly from the raw data (the bag of strings). At the same time, building the MINERfulKB had to be fast as well: moving the whole complexity to this step would take no advantage. The pseudo-code of the algorithm, as well as the details and running examples of its execution are presented in in [15].

The next and final step of the technique is the identification of the constraints verified over the set of strings. These are directly expressible as predicates over the MINERfulKB, and therefore easily transposable into instructions for a verification algorithm. The details, as well as the definitions of all the constraints as predicates over the MINERfulKB, are presented in [15]. As an example, the *RespondedExistence*(a, b) constraint is expressed like the following:

$$\mathit{RespondedExistence}(\mathsf{a},\mathsf{b}) \equiv \neg(\delta_{\mathsf{a},\mathsf{b}}(0) > 0)$$

i.e., there is no string such that b was not read in if a was. A high-level description of the technique can be represented as in Algorithm 1.

Algorithm 1 The MINERful pseudo-code algorithm (bird-eye watching)

$\mathcal{K}_T \leftarrow \mathsf{computeKBOnwards}(T, \Sigma_T)$
$\mathcal{K}_T \leftarrow \mathsf{computeKBBackwards}(T, \Sigma_T)$
$\mathcal{B} \leftarrow \mathsf{discoverConstraints}(\mathcal{K}_T, \Sigma_T, |T|)$

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-642-30359-3_2

5 Validation and Experiments

A validation of the technique has been performed by using the example outlined in Section 3.1 as the starting point. Its aim is to show the efficiency of the mining technique, being the underlying algorithm polynomial wrt. the dimension of the input.

We tested the algorithm by varying the input in terms of alphabet size (different characters appearing in the strings), number of constraints valid over the strings, range of the number of characters per string. Starting with two tasks, n and p, and the related constraints, we made various experiments making the alphabet grow up to the original, thus including also r and c, plus an unconstrained task, e. The constraints ranged from the minimal set of four ($Unique(n)$, $Participation(n)$, $End(n)$, $Succession(p, n)$) to the maximal set of seven (including $Response(r, p)$, $RespondedExistence(c, p)$, $AlternatePrecedence(r, c)$). The lengths of the strings ranged through intervals of $\{[2 \dots 8], [3 \dots 12], [4 \dots 16], [5 \dots 20]\}$. The number of strings ranged as the power of 10, from 100 up to 1000000 with an exponential step. For each of the preceding combination, 10 runs were performed, for a total amount of 4480 executions. The random strings were created by Xeger[3], a Java open-source library for generating random text from regular expressions. All of the parameters, and not only constraints, were expressed in terms of regular expressions indeed, and their conjunction passed to the Xeger engine.

The machine was a Sony VAIO VGN-FE11H (an 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.

As the reader can see in Figure 1(a), the number of constraints does not affect the time taken by the algorithm to run – indeed, you are not able to distinguish between the curve designed by a group of points and another, where each group is related to a given number of constraints.

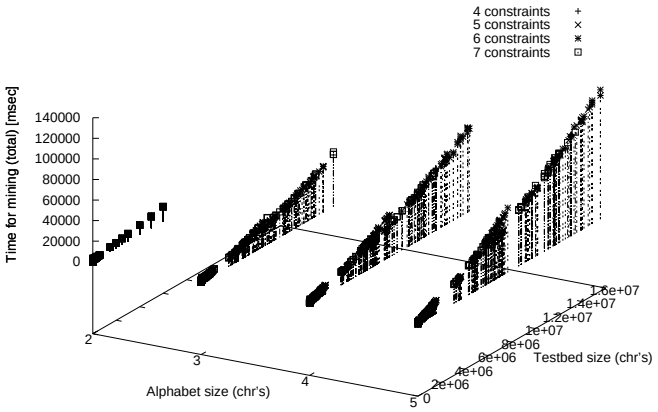
Figure 1(b) shows the fitting curves of the time taken by the algorithm to run, in comparison with the total amount of characters in input. It is a section of a parabola, confirming that the algorithm is quadratic w.r.t. the size of the strings.

The algorithm is linear in the size of the collection of traces. In order to test this, we made a slightly different set-up: we fixed the number of constraints (7), the number of characters per string (10), and the alphabet length (5), whereas the number of strings ranged from 1000 to 12000 with a step of 1000. The result is depicted in Figure 1(c).

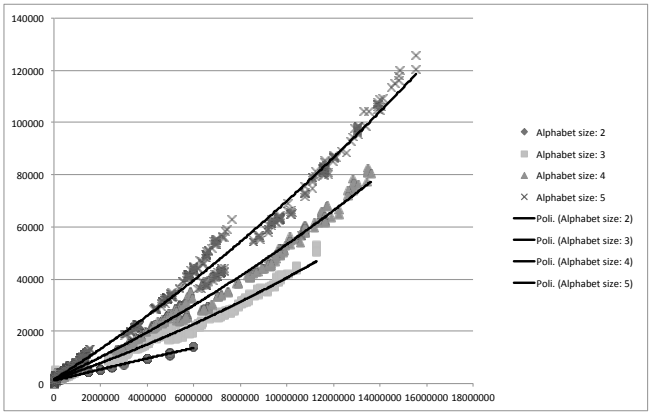
Furthermore, we report here by evidence that the time to build the MINERfulKB is the hardest task of the algorithm, with respect to the mining of constraints: in the worst case (1000000 strings, 7 constraints, 5 characters, length ranging from 5 to 20) the MINERfulKB was computed in 125.78 seconds whereas constraints were discovered in less than half a second (47 msec).

[3] `http://code.google.com/p/xeger/`

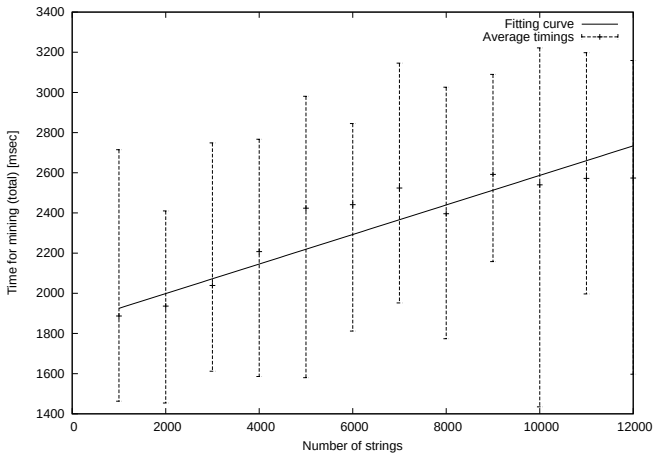
Pre-print copy of the manuscript published by Springer (available at `link.springer.com`) identified by DOI: `10.1007/978-3-642-30359-3_2`



(a) Time needed for the execution, with respect to the testbed size and the alphabet size, setting the number of imposed constraints as parameter



(b) Execution time (ordinates), with respect to the testbed size (abscissae), setting the alphabet size as parameter



(c) Execution time, with respect to the number of strings, keeping fixed the size of strings, the alphabet length and the number of constraints

Fig. 1. Experimental results

Pre-print copy of the manuscript published by Springer (available at `link.springer.com`) identified by DOI: `10.1007/978-3-642-30359-3_2`

6 Conclusions

As a concluding remark, we would like to highlight how the technique presented in this paper is only the last step of a complex approach, aimed at inferring artful processes from email messages; once that other techniques, out of the scope of this paper, allow us to consider email messages as strings over an alphabet of characters, the MINERful technique presented in this paper is able to infer which constraints are valid over such strings, thus inferring the process (described in a declarative way) that may lay behind them.

Further research activities are needed in order to refine and solve all of the used techniques and raised issues in MailOfMine. An extensive validation of the overall approach is planned as well. In this paper, we have shown that MINERful is a very efficient algorithm for mining constraints over strings. Throughout the experiments, we will be able to assess how much overfitting or underfitting MINERful is. We aim at validating it on a corpus of about 10 Gigabytes of email messages, derived from the activity of one of the authors in about 10 years of works in research projects, in order to infer common processes that partners adopted during software/deliverables' production. Then we will apply the approach to the field of collaborative activities of Open Source software development, reported by publicly available mailing lists.

References

1. van der Aalst, W., Rubin, V., Verbeek, H., van Dongen, B., Kindler, E., Gnther, C.: Process mining: a two-step approach to balance between underfitting and overfitting. Software and Systems Modeling 9, 87–111 (2010)
2. van der Aalst, W.M.P.: Verification of workflow nets. In: ICATPN 1997, LNCS 1248, pp. 407–426
3. van der Aalst, W.M.P.: Process mining: Discovery, conformance and enhancement of business processes. Springer (2011)
4. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Rozinat, A., Verbeek, E., Weijters, T.: ProM: The process mining toolkit. In: BPM 2009 Demos. CEUR Workshop Proceedings, vol. 489
5. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: WS-FM 2006, LNCS 4184, pp. 1–23
6. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. 16(9), 1128–1142 (2004)
7. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: EDBT'98, LNCS 1377, pp. 467–483
8. Agrawal, R., Srikant, R.: Mining sequential patterns. In: ICDE 1995, pp. 3–14
9. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: The SCIFF framework. ACM Trans. Comput. Log. 9(4) (2008)
10. de Carvalho, V.R., Cohen, W.W.: Learning to extract signature and reply lines from email. In: CEAS 2004
11. Catarci, T., Dix, A., Katifori, A., Lepouras, G., Poggi, A.: Task-centred information management. In: DELOS Conference 2007, LNCS 4877, pp. 197–206

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-642-30359-3_2

12. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. T. Petri Nets and Other Models of Concurrency 2, 278–295 (2009)
13. Chomsky, N., Miller, G.A.: Finite state languages. Information and Control 1(2), 91–112 (1958)
14. Di Ciccio, C., Catarci, T., Mecella, M.: Representing and visualizing mined artful processes in MailOfMine. In: Workshop on Human-Computer Interaction & Knowledge Discovery and Data Mining (HCI-KDD), LNCS 7058, 2011
15. Di Ciccio, C., Mecella, M.: MINERful, a mining algorithm for declarative process constraints in MailOfMine. Tech. rep. SAPIENZA Università di Roma (2012), http://ojs.uniroma1.it/index.php/DIS_TechnicalReports/issue/view/416
16. Di Ciccio, C., Mecella, M., Scannapieco, M., Zardetto, D., Catarci, T.: MailOfMine – Analyzing mail messages for mining artful collaborative processes. In: SIMPDA 2011, pp. 45–59
17. Garofalakis, M.N., Rastogi, R., Shim, K.: SPIRIT: Sequential pattern mining with regular expression constraints. In: VLDB 1999, pp. 223–234
18. Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: PSTV 1995, pp. 3–18
19. Giannakopoulou, D., Havelund, K.: Automata-based verification of temporal properties on running programs. In: ASE 2001, pp. 412–416
20. Heutelbeck, D.: Preservation of enterprise engineering processes by social collaboration software (2011), personal communication
21. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: CIDM 2011, pp. 192–199
22. Medeiros, A.K., Weijters, A.J., Aalst, W.M.: Genetic process mining: an experimental evaluation. Data Min. Knowl. Discov. 14(2), 245–304 (2007)
23. Myers, E.W.: An O(ND) difference algorithm and its variations. Algorithmica 1(2), 251–266 (1986)
24. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: BPM 2006 Workshops, LNCS 4103, pp. 169–180
25. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. In: EDOC 2007, pp. 287–300
26. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-based workflow models: Change made easy. In: OTM Conferences 2007, LNCS 4803, pp. 77–94
27. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. 3, 114–125 (April 1959)
28. Smart Vortex Consortium: Smart Vortex – Management and analysis of massive data streams to support large-scale collaborative engineering projects. FP7 IP Project, http://www.smartvortex.eu/
29. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: EDBT 1996, LNCS 1057, pp. 3–17
30. Warren, P., Kings, N., Thurlow, I., Davies, J., Buerger, T., Simperl, E., Ruiz, C., Gomez-Perez, J.M., Ermolayev, V., Ghani, R., Tilly, M., Bösser, T., Imtiaz, A.: Improving knowledge worker productivity - the Active integrated approach. BT Technology Journal 26(2), 165–176 (2009)
31. Weijters, A., van der Aalst, W.: Rediscovering workflow models from event-based data using little thumb. Integrated Computer-Aided Engineering 10, 2003 (2001)
32. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Min. Knowl. Discov. 15(2), 145–180 (2007)
33. Westergaard, M.: Better algorithms for analyzing and enacting declarative workflow languages using ltl. In: BPM 2011, LNCS 6896, pp. 83–98

Pre-print copy of the manuscript published by Springer (available at link.springer.com) identified by DOI: 10.1007/978-3-642-30359-3_2

This document is a pre-print copy of the manuscript
(Di Ciccio and Mecella 2012)
published by Springer (available at link.springer.com).

The final version of the paper is identified by DOI: 10.1007/978-3-642-30359-3_2

References

Di Ciccio, Claudio and Massimo Mecella (2012). "Mining Constraints for Artful Processes". In: *BIS*. Ed. by Witold Abramowicz, Dalia Kriksciuniene, and Virgilijus Sakalauskas. Vol. 117. Lecture Notes in Business Information Processing. Springer, pp. 11–23. ISBN: 978-3-642-30358-6. DOI: 10.1007/978-3-642-30359-3_2.

BibTeX

```
@InProceedings{   DiCiccio.Mecella/BIS2012:MiningConstraintsArtfulProcesses,
  author        = {Di Ciccio, Claudio and Mecella, Massimo},
  title         = {Mining Constraints for Artful Processes},
  booktitle     = {BIS},
  year          = {2012},
  pages         = {11-23},
  publisher     = {Springer},
  crossref      = {BIS2012},
  doi           = {10.1007/978-3-642-30359-3_2},
  keywords      = {process mining, artful process, declarative workflow,
                   email}
}
@Proceedings{     BIS2012,
  title         = {Business Information Systems - 15th International
                   Conference, {BIS} 2012, Vilnius, Lithuania, May 21-23,
                   2012. Proceedings},
  year          = {2012},
  editor        = {Witold Abramowicz and Dalia Kriksciuniene and Virgilijus
                   Sakalauskas},
  volume        = {117},
  series        = {Lecture Notes in Business Information Processing},
  publisher     = {Springer},
  isbn          = {978-3-642-30358-6},
  doi           = {10.1007/978-3-642-30359-3}
}
```