Mining Constraints for Artful Processes

Claudio Di Ciccio and Massimo Mecella

Claudio Di Ciccio (cdc@dis.uniroma1.it)
15th International Conference on Business Information Systems (BIS 2012)
Tuesday, May the 22nd, Vilnius, Lithuania
Process Mining

Definition

• Process Mining [Aalst2011.book], also referred to as Workflow Mining, is the set of techniques that allow the extraction of process descriptions, stemming from a set of recorded real executions (logs).

• ProM [AalstEtAl2009] is one of the most used plug-in based software environment for implementing workflow mining (and more) techniques.
  • The new version 6.0 is available for download at www.processmining.org
Process Mining

Definition

- Process Mining involves:
  - Process discovery
    - Control flow mining, organizational mining, decision mining;
  - Conformance checking
  - Operational support
- We will focus on the **control flow mining**
- Many control flow mining algorithms proposed
  - $\alpha$ [AalstEtAl2004] and $\alpha^{++}$ [WenEtAl2007]
  - Fuzzy [GüntherAalst2007]
  - Heuristic [WeijtersEtAl2001]
  - Genetic [MedeirosEtAl2007]
  - Two-step [AalstEtAl2010]
  - ...

BIS 2012, Vilnius  
Mining Constraints for Artful Processes (Di Ciccio, C. - DIAG, SAPIENZA Università di Roma)
A different context

Artful processes and knowledge workers

- Artful processes \[\text{HillEtAl06}\]
  - informal processes typically carried out by those people whose work is mental rather than physical (managers, professors, researchers, engineers, etc.)
  - “knowledge workers” \[\text{ACTIVE09}\]
- Knowledge workers create artful processes “on the fly”
- Though artful processes are frequently repeated, they are not exactly reproducible, even by their originators, nor can they be easily shared.
A different context

Email conversations

- In collaborative contexts, knowledge workers **share** their information and outcomes with other knowledge workers
  - E.g., a software development mgr.
- Typically, by means of several **email conversations**
  - Email conversations are actual **traces** of running processes that knowledge workers adhere to
A different context
Processes from email conversations

• From the collection of email messages, you can **extract the processes** that lay behind
  ▪ Related e-mail conversations are traces of their runs

• Valuable advantages for users
  ▪ Automated discovery of **formal representations**
    • with **no effort** for knowledge workers
  ▪ **Tidy organization** for naïve best practices kept only in mind
  ▪ Opportunity to **share and compare** the knowledge on methodologies
  ▪ Automated discovery of **bottlenecks, delays, structural defects**
    • from the analysis of previous runs

• Email conversations are a kind of **semi-structured text**
  ▪ this approach is not tailored to the electronic mail
    • it can be extended to the analysis of other semi-structured texts
MailOfMine

What is MailOfMine?

- **MailOfMine** is the approach and the implementation of a collection of techniques, 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.

- [DiCiccioEtAl11](#)
- [DiCiccioMecella/TR12](#)
The MailOfMine approach
From the e-mail archive to key parts

Mail archive

Mail Database

Conversations

Key Parts

Multi-format mail storage plug-in based crawlers

[ZardettoEtAl10]-based clustering algorithm

[CarvalhoEtAl04]-based filter

BIS 2012, Vilnius
Mining Constraints for Artful Processes (Di Ciccio, C. - DIAG, SAPIENZA Università di Roma)
From key parts to processes

Tasks

Processes

Key Parts Concatenation

Activity indicium

[BernardosEtAl04, SakuraiEtAl05]-based task extractor

[BZardoetoEtAl10]-based
MINERful

Processes

BIS 2012, Vilnius
Mining Constraints for Artful Processes (Di Ciccio, C. - DIAG, SAPIENZA Università di Roma)
MINERful
The mining algorithm in MailOfMine

- MINERful is a workflow mining algorithm
- Its input is a collection of strings $T$ and an alphabet $\Sigma_T$
  - Each string $t$ is a trace
  - Each character is an event (enacted task)
  - The collection represents the log
- Its output is a declarative process model
  - What is a declarative process model?
On the visualization of processes

The imperative model

- Represents the whole process at once
- The most used notation is based on a subclass of Petri Nets (namely, the Workflow Nets)
On the visualization of processes

The declarative model

- Rather than using a **procedural language** for expressing the allowed sequence 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 the ones which do not respect such constraints
  - this technique fits with processes that are highly flexible and subject to changes, such as artful processes

The notation here is based on [AalstEtAl06, MaggiEtAl11] (DecSerFlow, Declare)
On the visualization of processes
Imperative vS declarative

Declerative models work better in presence of a partial specification of the process scheme
A real discovered process model

“Spaghetti process” [Aalst2011.book]
## Declare constraint templates

**Constraint templates as Regular Expressions (REs)**

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Regular expression</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Existence</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Participation(a) \equiv Existence(1, a)</td>
<td>[^a]<em>(a[^a]</em>+[^a]*)</td>
<td>bcaac</td>
</tr>
<tr>
<td>Unique(a) \equiv Absence(2, a)</td>
<td>[^a]<em>(a)[^a]</em>[^a]*</td>
<td>bcac</td>
</tr>
<tr>
<td>Init(a)</td>
<td>a.*</td>
<td>accbbbababa</td>
</tr>
<tr>
<td>End(a)</td>
<td>.*a</td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td><strong>Relation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RespondedExistence(a, b)</td>
<td>[^a]*((a.<em>b)\ (b.<em>a))</em>[^a]</em></td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>Response(a, b)</td>
<td>[^a]<em>((a.<em>b))</em>[^a]</em></td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>AlternateResponse(a, b)</td>
<td>[^a]<em>((a[^a]<em>b)</em>[^a])</em></td>
<td>bcaaccbbbab</td>
</tr>
<tr>
<td>ChainResponse(a, b)</td>
<td>[^a]<em>((ab[^a]<em>b)</em>[^a]&gt;</em>[^a])*</td>
<td>bcabbbab</td>
</tr>
<tr>
<td>Precedence(a, b)</td>
<td>[^b]*((a.<em>b))</em>[^b]*</td>
<td>caaccbbaba</td>
</tr>
<tr>
<td>AlternatePrecedence(a, b)</td>
<td>[^b]*((a[^b]<em>b)</em>[^b])*</td>
<td>caaccbbaba</td>
</tr>
<tr>
<td>ChainPrecedence(a, b)</td>
<td>[^b]*((ab[^a]<em>b)</em>[^b])*</td>
<td>caaccbbaba</td>
</tr>
<tr>
<td>CoExistence(a, b)</td>
<td>[^a]*((a.*b)\ (b.<em>a))</em>[^a]*</td>
<td>bcaccbbaba</td>
</tr>
<tr>
<td>Succession(a, b)</td>
<td>[^a]*((a.<em>b))</em>[^a]*</td>
<td>caaccbbab</td>
</tr>
<tr>
<td>AlternateSuccession(a, b)</td>
<td>[^a]*((a[^a]<em>b)</em>[^b])*</td>
<td>caccbab</td>
</tr>
<tr>
<td>ChainSuccession(a, b)</td>
<td>[^a]*((ab[^a]<em>b)</em>[^a])*</td>
<td>cabb</td>
</tr>
<tr>
<td><strong>Negative relation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NotSuccession(a, b)</td>
<td>[^a]*((a[^b]<em>b)</em>[^a])*</td>
<td>bcaaccbbba</td>
</tr>
<tr>
<td>NotCoExistence(a, b)</td>
<td>[^a]*((a[^b]<em>b)</em>\ (b[^a])*)?</td>
<td>bcaacca</td>
</tr>
<tr>
<td>NotChainSuccession(a, b)</td>
<td>[^a]*((a[^a]<em>b)</em>[^a])*</td>
<td>caacca</td>
</tr>
</tbody>
</table>
### Declare constraint templates

**Constraint templates as Regular Expressions (REs)**

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Regular expression</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Existence constraints</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Participation(a) (\equiv) Existence(1, a)</td>
<td>[^a]<em>(a[^a]</em>)+[^[a]*</td>
<td>bcaaac</td>
</tr>
<tr>
<td>Unique(a) (\equiv) Absence(2, a)</td>
<td>[^[a]*</td>
<td>(a)?^[^[a]*</td>
</tr>
<tr>
<td>Init(a)</td>
<td>a.*</td>
<td>accbbbaba</td>
</tr>
<tr>
<td>End(a)</td>
<td>.*a</td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td><strong>Relation constraints</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RespondedExistence(a, b)</td>
<td>[^[a]*((a.*b)</td>
<td>(b.<em>a))</em>^[^[a]*</td>
</tr>
<tr>
<td>Response(a, b)</td>
<td>[^[a]<em>((a.<em>b)</em>^[^[a]</em></td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>AlternateResponse(a, b)</td>
<td>[^[a]<em>((a[^a]<em>b)</em>^[^[a]</em></td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>ChainResponse(a, b)</td>
<td>[^[a]<em>((ab[^a]<em>b)</em>^[^[a]</em></td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>Precedence(a, b)</td>
<td>[^[b]<em>((a.<em>b)</em>^[^[b]</em></td>
<td>caaccbbbaba</td>
</tr>
<tr>
<td>AlternatePrecedence(a, b)</td>
<td>[^[b]<em>((a[^b]<em>b)</em>^[^[b]</em></td>
<td>caaccbbbaba</td>
</tr>
<tr>
<td>ChainPrecedence(a, b)</td>
<td>[^[b]<em>((ab[^a]<em>b)</em>^[^[b]</em></td>
<td>caaccbbbaba</td>
</tr>
<tr>
<td>CoExistence(a, b)</td>
<td>[^[a]*((a.*b)</td>
<td>(b.<em>a))</em>^[^[a]*^b)</td>
</tr>
<tr>
<td>Succession(a, b)</td>
<td>[^[a]<em>((a.<em>b)</em>^[^[a]</em>^b)</td>
<td>caaccbbbaba</td>
</tr>
<tr>
<td>AlternateSuccession(a, b)</td>
<td>[^[a]<em>((a[^a]<em>b)</em>^[^[a]</em>^b)</td>
<td>caaccbbbaba</td>
</tr>
<tr>
<td>ChainSuccession(a, b)</td>
<td>[^[a]<em>((ab[^a]<em>b)</em>^[^[a]</em>^b)</td>
<td>caaccbbbaba</td>
</tr>
<tr>
<td><strong>Negative relation constraints</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NotChainSuccession(a, b)</td>
<td>[^[a]<em>((a[^a]<em>b)</em>^[^[a]</em>^a)</td>
<td>bcaaccbbbba</td>
</tr>
<tr>
<td>NotSuccession(a, b)</td>
<td>[^[a]<em>((a[^b]<em>b)</em>^[^[a]</em>^b)</td>
<td>bcaaccacca</td>
</tr>
<tr>
<td>NotCoExistence(a, b)</td>
<td>[^[a]*((a[^b]*b)</td>
<td>(b[^a]*b))</td>
</tr>
</tbody>
</table>
MINERful by example

Scenario

- A project meeting is scheduled
- We suppose that a final agenda will be committed ("confirmAgenda") after that requests for a new proposal ("requestAgenda"), proposals themselves ("proposeAgenda") and comments ("commentAgenda") have been circulated.
- Shortcuts for tasks (**process alphabet**):
  - `p` ("proposeAgenda")
  - `r` ("requestAgenda")
  - `c` ("commentAgenda")
  - `n` ("confirmAgenda")
MINERful by example
Constraints on tasks

- **Existence constraints**
  1. Participation($n$)
  2. Uniqueness($n$)
  3. End($n$)

- **Relation constraints**
  4. Response($r,p$)
  5. RespondedExistence($c,p$)
  6. Succession($p,n$)

- **The agenda**
  1. *must* be confirmed,
  2. only *once*:
  3. it is the *last* thing to do.

- **During the compilation:**
  4. the proposal follows a request;
  5. if a comment circulates, there has been / will be a proposal;
  6. after the proposal, there will be a confirmation, and there can be no confirmation without a preceding proposal.
MINERful by example

Testing by replay

- In order to **validate the algorithm**
  - We translate **constraints into REs**
    - The overall process is expressed by the intersection of REs
  - We use a RE-driven random string builder [Xeger](https://xeger.com) for creating a test-and-validation set
  - We analyze the result and evaluate the performances

- In order to see **how it works now**
  - We follow a run of MINERful over a string built by Xeger:

```
rrp crp crp c
```
MINERful by example
Building the “ownplay” of $p$ and $n$

- $p$
  - $p$ occurred 3 times in 1 string
    - $\gamma_p(3) = 1$
      - For each $m \neq 3$
        - $\gamma_p(m) = 0$
  - $p$ did not occur as the first nor as the last character
    - $g^i(p) = 0$
    - $g^l(p) = 0$

- $n$
  - $\gamma_n(1) = 1$
  - For each $m \neq 1$, $\gamma_n(m) = 0$
  - $n$ occurred as the last character in 1 string
    - $g^i(n) = 0$
    - $g^l(n) = 1$
MINERful by example
Building the “interplay” of $p$ and $n$

- With respect to the occurrence of $p, n$ occurred...
  i. Never before: 3 times
     $\delta_{p,n}(-\infty) = 3$
  ii. 2 char’s after: 1 time
     $\delta_{p,n}(2) = 1$
  iii. 6 char’s after: 1 time
     $\delta_{p,n}(6) = 1$
  iv. 9 char’s after: 1 time
     $\delta_{p,n}(9) = 1$
  v. Alternating:
    i. Onwards: 2 times
       $b_{p,n} = 2$
    ii. Backwards: never
       $b_{p,n} = 0$

- Looking at the string
  i. $rrpcrpcrpcp cn$
  ii. $rrpcrpcrpcp cn$
  iii. $rrpcrpcrpcp cn$
  iv. $rrpcrpcrpcp cn$
  v. $rrpcrpcrpcp cn$

BIS 2012, Vilnius
Mining Constraints for Artful Processes (Di Ciccio, C. - DIAG, SAPIENZA Università di Roma)
MINERful by example
Building the “interplay” of \( r \) and \( p \)

\[
\begin{array}{cccccccc}
-\infty & -5 & -2 & +1 & +2 & +4 & +5 & +8 & +9 \\
\delta_{r,p} & 2 & 1 & 2 & 2 & 2 & 1 & 2 & 1 \\
\end{array}
\]

\[
b_{r,p}^\rightarrow = 1 \\
b_{r,p}^\leftarrow = 0
\]

\[
\mathbf{r r p c r p c r c p c n}
\]
MINERful by example
Workflow discovery by constraints inference

- Interplay and ownplay constitute the Knowledge Base of MINERful
  - The KB construction is such that each new string adds information
  - The algorithm does not need to read the strings more than once each
- Constraints are determined by the evaluation of boolean queries on the KB
- This allows the discovery of constraints with a faster procedure on a smaller set than the whole input
- MINERful is a two-step algorithm
MINERful by example

Some queries for inferring constraints

- \textit{RespondedExistence}(c,p)

  \hspace{1cm} \checkmark \hspace{1cm} \neg(\delta_{r,p}(0) > 0)

  There is no string where \( p \) does not occur, if \( r \) is read

- \textit{Response}(r,p)

  \hspace{1cm} \checkmark \hspace{1cm} \textit{RespondedExistence}(r,p) \land \neg(\delta_{r,p}(+\infty) > 0)

  \textit{RespondedExistence}(r,p) \) holds and there is no string where \( p \) does not follow \( r \)

- \textit{Precedence}(r,p)

  \hspace{1cm} \times \hspace{1cm} \textit{RespondedExistence}(p,r) \land \neg(\delta_{r,p}(-\infty) > 0)

  \textit{RespondedExistence}(p,r) \) holds and there is no string where \( p \) does not precede \( r \)

- \textit{Succession}(p,n)

  \hspace{1cm} \checkmark \hspace{1cm} \textit{Response}(p,n) \land \textit{Precedence}(p,n)

- ...
MINERful by example

Other inferred constraints

- Response(c, n)
- RespondedExistence(c, p)
- NotSuccession(n, c), NotSuccession(n, p), NotSuccession(n, r)
- Participation(p)
- AlternatePrecedence(p, n)
- Succession(p, n)
- AlternatePrecedence(r, c)
- Response(r, p)
Relation constraint templates subsumption
Constraint templates are not independent of each other

- E.g.,
  - A trace like \( a \ b \ a \ b \ c \ a \ b \ c \ c \) satisfies (w.r.t. \( b \) and \( a \)):
    - \( \text{RespondedExistence}(a, b), \text{RespondedExistence}(b, a), \text{CoExistence}(a, b), \text{CoExistence}(b, a), \text{Response}(a, b), \text{AlternateResponse}(a, b), \text{ChainResponse}(a, b), \text{Precedence}(a, b), \text{AlternatePrecedence}(a, b), \text{ChainPrecedence}(a, b), \text{Succession}(a, b), \text{AlternateSuccession}(a, b), \text{ChainSuccession}(a, b) \)
  
- The mining algorithm would show the most strict constraint only (\( \text{ChainSuccession}(a, b) \))

- MINERful faces and solves this issue, by refining queries on the basis of the subsumption hierarchy of constraints
Relation constraint templates subsumption
Constraint templates are not independent of each other
Conclusions

Recap

- MailOfMine is a system designed for mining artful processes out of email collections
- MINERful is the workflow mining algorithm designed for MailOfMine
- MINERful is
  - Independent on the formalism used for expressing constraints
  - Modular (two-phase)
  - Capable of eliminating redundancy in the process model
Conclusions

On the asymptotic complexity of MINERful

- Linear w.r.t. the number of strings in the testbed $|T|$
- Quadratic w.r.t. the size of strings in the testbed $|t_{\text{max}}|$
- Quadratic w.r.t. the size of the alphabet $|\Sigma_T|$
- Hence, polynomial in the size of the input $O(|T| \cdot |t_{\text{max}}|^2 \cdot |\Sigma_T|^2)$
References

Cited articles and resources, in order of appearance

References

Cited articles and resources, in order of appearance