A Two-Step Fast Algorithm for the Automated Discovery of Declarative Workflows

Claudio Di Cicco and Massimo Mecella

Claudio Di Cicco (cdc@dis.uniroma1.it)
IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2013)
Wednesday, April the 17th, Singapore
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.
  • www.processmining.org
A different context for process mining
Artful processes and knowledge workers

• **Artful processes** [HillEtAl06]
  - informal processes typically carried out by those people whose work is mental rather than physical (managers, professors, researchers, engineers, etc.)
  
  • **“knowledge workers”** [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
  - **Loosely structured**
  - **Highly flexible**
MINERful++

Our mining algorithm

- MINERful++ is the workflow discovery algorithm of MailOfMine
- 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 modeling 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 modeling 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 those which do not respect such constraints
  - this technique fits for processes that are highly flexible and subject to changes, such as artful processes

The notation here is based on [Pesic08, MaggiEtAl11] (ConDec, Declare)
### 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] * (a[^a]<em>)([^a]</em>[^a]<em>[^a]</em>)</td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>Unique((a) \equiv Absence(2, a))</td>
<td>[^a] * (a[^a]<em>)?[^a]</em></td>
<td>bcacc</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 constraints</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RespondedExistence((a, b))</td>
<td>[^a] * (a[^a]*b</td>
<td>b[^a]<em>)[^a]</em></td>
</tr>
<tr>
<td>Response((a, b))</td>
<td>[^a] * (a[^a]<em>b)[^a]</em></td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>AlternateResponse((a, b))</td>
<td>[^a] * (a[^a]<em>b)</em>[^a]*</td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>ChainResponse((a, b))</td>
<td>[^a] * (ab[^a]<em>b)</em>[^a]*</td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>Precedence((a, b))</td>
<td>[^b] * (a[^b]<em>b)</em>[^b]*</td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>AlternatePrecedence((a, b))</td>
<td>[^b] * (a[^b]<em>b)</em>[^b]*</td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>ChainPrecedence((a, b))</td>
<td>[^b] * (ab[^a]<em>b)</em>[^b]*</td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>CoExistence((a, b))</td>
<td>[^a^b] * (a[^b]*b</td>
<td>b[^a]<em>)[^a]</em></td>
</tr>
<tr>
<td>Succession((a, b))</td>
<td>[^a^b] * (a[^a]<em>b)</em>[^a]*</td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>AlternateSuccession((a, b))</td>
<td>[^a^b] * (a[^a]<em>b)</em>[^a]*</td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>ChainSuccession((a, b))</td>
<td>[^a^b] * (ab[^a]<em>b)</em>[^a]*</td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td><strong>Negative relation constraints</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NotChainSuccession((a, b))</td>
<td>[^a] * (a[^a]<em>b)[^a]</em>[^a]*</td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>NotSuccession((a, b))</td>
<td>[^a] * (a[^b]<em>b)</em>[^a]*</td>
<td>bcaaccbbbaba</td>
</tr>
<tr>
<td>NotCoExistence((a, b))</td>
<td>[^a^b] * (a[^b]*b)^{1}</td>
<td>(b[^a]*b)^{2}</td>
</tr>
</tbody>
</table>
On the modeling of processes

Imperative vs. declarative

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

“Spaghetti process” [Aalst2011.book]
The declarative specification of an artful process (e.g.)

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")
The declarative specification of an artful process (e.g.)

Constraints on activities

- **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++

Workflow discovery by constraints inference

- MINERful++ is a **two-step** algorithm
  1. Construction of a **Knowledge Base**
  2. Constraints inference by means of **queries** evaluated on the KB
- This allows the discovery of constraints through a **faster** procedure on **data** which are **smaller** in size than the whole input
- Returned constraints are weighted with their **support**
  - the normalized fraction of cases in which the constraint is verified over the set of input traces
MINERful++ by example
Workflow discovery by constraints inference

- In order to see **how it works now**
  - we see a run of MINERful++ over a string, compliant with the previous example:

  \[
  r r p c r p c r c p c n
  \]

- We start with the construction of the algorithm’s Knowledge Base
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. Repetitions in-between:
    
    i. onwards: 2 times
        $b_{p,n}^{-} = 2$
    
    ii. backwards: never
        $b_{p,n}^{+} = 0$

- Looking at the string
  
  i. $rrppcrpcrep$ $p $ $n$
  
  ii. $rrppcrpcrep$ $p $ $n$
  
  iii. $rrppcrpcrep$ $p $ $n$
  
  iv. $rrppcrpcrep$ $p $ $n$
  
  v. $rrppcrpcrep$ $p $ $n$
MINERful++ by example
Building the “interplay” of $r$ and $p$

<table>
<thead>
<tr>
<th></th>
<th>-∞</th>
<th>-5</th>
<th>-2</th>
<th>+1</th>
<th>+2</th>
<th>+4</th>
<th>+5</th>
<th>+8</th>
<th>+9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta_{r,p}$</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$b^{r \rightarrow}_{r,p} = 1$

$b^{r \leftarrow}_{r,p} = 0$

$r r p c r p c r c p c n$
MINERful++

Computing the support of constraints

- Let $\Gamma_r$ and $\Gamma_p$ be the number of times in which $r$ and $p$ respectively appear in the log

$$\Gamma_x = \sum_{n>0} \gamma_x(n) \cdot n$$

we have, e.g.,

- support for $Response(r, p)$

$$1 - \frac{\delta_{r,p}(+\infty)}{\Gamma_r}$$

- hint: how many times $p$ was not read in the traces after $r$ occurred?

  In those cases, $Response(r, p)$ does not hold

- support for $Succession(r, p)$

$$1 - \frac{\delta_{r,p}(+\infty) + \delta_{p,r}(-\infty)}{\Gamma_r + \Gamma_p}$$

- how many times $p$ was not read in the traces after $r$ occurred, nor $r$ was read before $p$ occurred?

  In those cases, $Succession(r, p)$ does not hold

- ...
## MINERful++
Computing the support of constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Support function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existence(n, a)</td>
<td>$1 - \frac{\sum_{i=0}^{n-1} Y_a(i)}{</td>
</tr>
<tr>
<td>Absence(m, a)</td>
<td>$\frac{q_a}{</td>
</tr>
<tr>
<td>Init(a)</td>
<td></td>
</tr>
<tr>
<td>RespondedExistence(a, b)</td>
<td>$1 - \frac{\delta_{a,b}(0)}{r_b}$</td>
</tr>
<tr>
<td>Response(a, b)</td>
<td>$1 - \frac{\delta_{a,b}(+\infty)}{r_a}$</td>
</tr>
<tr>
<td>AlternateResponse(a, b)</td>
<td>$1 - \frac{b_{a,b}^{-} + \delta_{a,b}(+\infty)}{r_a}$</td>
</tr>
<tr>
<td>ChainResponse(a, b)</td>
<td>$\frac{\delta_{a,b}(1)}{r_a}$</td>
</tr>
<tr>
<td>CoExistence(a, b)</td>
<td>$1 - \frac{\delta_{a,b}(0) + \delta_{b,a}(0)}{r_a + r_b}$</td>
</tr>
<tr>
<td>Succession(a, b)</td>
<td>$1 - \frac{\delta_{a,b}(+\infty) + \delta_{b,a}(-\infty)}{r_a + r_b}$</td>
</tr>
<tr>
<td>AlternateSuccession(a, b)</td>
<td>$1 - \frac{b_{a,b}^{-} + \delta_{a,b}(+\infty) + b_{b,a} + \delta_{b,a}(-\infty)}{r_a + r_b}$</td>
</tr>
<tr>
<td>ChainSuccession(a, b)</td>
<td>$\frac{\delta_{a,b}(1) + \delta_{b,a}(-1)}{r_a + r_b}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Support function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participation(a)</td>
<td>$1 - \frac{y_a(0)}{</td>
</tr>
<tr>
<td>Uniqueness(a)</td>
<td>$\frac{y_a(0) + y_a(1)}{</td>
</tr>
<tr>
<td>End(a)</td>
<td>$\frac{q_a}{</td>
</tr>
<tr>
<td>Precedence(a, b)</td>
<td></td>
</tr>
<tr>
<td>AlternatePrecedence(a, b)</td>
<td>$1 - \frac{\delta_{b,a}(-\infty)}{r_b}$</td>
</tr>
<tr>
<td>ChainPrecedence(a, b)</td>
<td>$1 - \frac{\delta_{b,a}(-\infty) + b_{a,b}^{-}}{r_b}$</td>
</tr>
<tr>
<td>NotCoExistence(a, b)</td>
<td>$\frac{\delta_{a,b}(0) + \delta_{b,a}(0)}{r_a + r_b}$</td>
</tr>
<tr>
<td>NotSuccession(a, b)</td>
<td>$\frac{\delta_{a,b}(+\infty) + \delta_{b,a}(-\infty)}{r_a + r_b}$</td>
</tr>
<tr>
<td>NotChainSuccession(a, b)</td>
<td>$1 - \frac{\delta_{a,b}(1) + \delta_{b,a}(-1)}{r_a + r_b}$</td>
</tr>
</tbody>
</table>
MINERful++ by example
Computing the support of constraints

1. r p c r p c r c p n
2. r c c r p p c p c r c p p n
3. c c c c c c r p c r r p r p r p p p n
4. c p r p n
5. r p p n
6. r r r c r c p n
7. r p c n
8. c r p r r c c p p n
9. p c c n
10. c r r e c c r r p c p c c p n

- Support for...
  - Response(r, p)
    - 1.0
  - Succession(r, p)
    - 0.96364
  - Precedence(c, n)
    - 0.9

- The support can be used to prune out those constraints falling under a given threshold
  - E.g., 0.95
- A threshold equal to 1.0 selects those constraints which are always valid on the log
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 have to show the most strict constraint only ($\text{ChainSuccession}(a, b)$)

- \textbf{MINERful}++ faces this issue, by pruning the returned constraints on the basis of the \textit{subsumption hierarchy of constraints}
Relation constraint templates subsumption

Constraint templates are not independent of each other
Relation constraint templates subsumption

A hint on the pruning procedure
Relation constraint templates subsumption

A hint on the pruning procedure
Evaluation

Characteristics of MINERful++

- **MINERful++** is
  - Independent on the formalism used for expressing constraints
  - Modular (two-phase)
  - Capable of eliminating redundancy in the process model
  - Fast

<table>
<thead>
<tr>
<th>Source</th>
<th>Tasks</th>
<th>Traces</th>
<th>Events processed</th>
<th>Total comp. time</th>
<th>Engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synth. log, Setup 1</td>
<td>5</td>
<td>100000</td>
<td>167647 (avg. 16.764)</td>
<td>00:00:15.036</td>
<td>MINERful++</td>
</tr>
<tr>
<td>Synth. log, Setup 2</td>
<td>52</td>
<td>16000</td>
<td>296277 (avg. 18.517)</td>
<td>00:11:20.000</td>
<td>Declare Miner 2011</td>
</tr>
<tr>
<td>Financial log [BPIC12]</td>
<td>24</td>
<td>13087</td>
<td>262200 (avg. 20.035)</td>
<td>00:21:24.000</td>
<td>MINERful++</td>
</tr>
<tr>
<td>Hospital log [BPIC11]</td>
<td>624</td>
<td>1143</td>
<td>150291 (avg. 131.488)</td>
<td>00:00:08.997</td>
<td>Declare Miner 2011</td>
</tr>
</tbody>
</table>

Sony VAIO VGN-FE11H
Intel Core Duo T2300 1.66 GHz
2 MB L2 cache
2 GB of DDR2 RAM @ 667 Mhz

<table>
<thead>
<tr>
<th>Setup</th>
<th>Min. length</th>
<th>Max. length</th>
<th>Number of traces</th>
<th>Alphabet size</th>
<th>Total tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0,8]</td>
<td>[5,20]</td>
<td>$10^2, 10^6$</td>
<td>[2,5]</td>
<td>29000</td>
</tr>
<tr>
<td>2</td>
<td>[0,2]</td>
<td>[10,25]</td>
<td>$10^3, 16 \cdot 10^3$</td>
<td>[5,50]</td>
<td>13536</td>
</tr>
</tbody>
</table>
Evaluation

On the complexity of MINERful++

- Linear w.r.t. the number of traces in the log $|T|$
- Quadratic w.r.t. the size of traces in the log $|t_{max}|$
- Quadratic w.r.t. the size of the alphabet $|\Sigma_T|$
- Hence, polynomial in the size of the input $O(|T| \cdot |t_{max}|^2 \cdot |\Sigma_T|^2)$
Preliminary results
MINERful++ on real data

- Logs extracted from 2 mailboxes
  - [DiCiccioMecella2012]
- 5 traces
  - 34.75 events each on average
  - 139 events read in total
- Evaluation of inferred constraints conducted with an expert domain
- Precision ≈ 0.794
Future work

Research in progress

- Integrate MINERful++ with ProM
  - MINERful++ is already capable of reading/writing XES logs
- Study the effects of errors in logs on the inferred workflow
  - Error-injected synthetic logs can help us conduct an automated analysis on the quality of results
- Refine the estimation calculi for support
- Auto-tune the support threshold
  - Depending on the constraint template, which can be more or less “robust”
- Enlarge the set of discovered constraint templates to the branching Declare constraints
References

Cited articles and resources, in order of appearance


