PSI-Miner 2020
Mining Temporal Causal Sequences from Time-Series

António Anastasio Bruto da Costa
Pallab Dasgupta
Formal Methods Laboratory, Dept. of Computer Science and Engineering

Indian Institute of Technology, Khrargpur

Contents

1 Philosophy and Architecture

PSI-Miner takes multiple time-value traces each as a *comma-separated value* (csv) file. The tool requires a configuration file (*config*) to be written. The config contains meta-parameters of the tool; parameters for the construction of the decision tree, n (number of sub-expressions in the sequence-expressions), k (delay resolution) and parameters for controlling the process of mining predicates using a simulated annealing local search. The config must specify domain knowledge in the form of the predicate alphabet to be used.

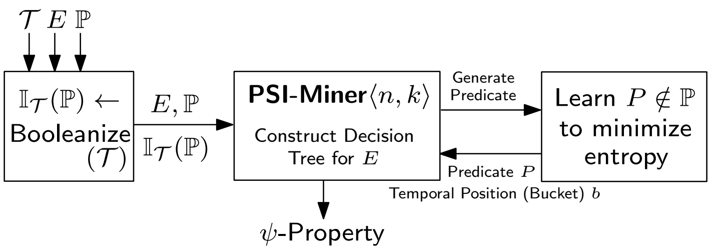
$\mathcal{T}$ E $\mathbb{P}$
$\mathbb{I}_{\mathcal{T}}(\mathbb{P}) \leftarrow$ Booleanize ($\mathcal{T}$)
$E, \mathbb{P}$
$\mathbb{I}_{\mathcal{T}}(\mathbb{P})$
PSI-Miner$\langle n, k\rangle$
Construct Decision Tree for E
Generate Predicate
Learn $P \notin \mathbb{P}$ to minimize entropy
Predicate P
Temporal Position (Bucket) b
ψ-Property

Figure 1: PSI-Miner: Outline

Under the constraints set by meta-parameters n and k, the form of the sequence generated is as follows:

$$s_n \ \tau_n \ s_{n-1} \ \tau_{n-1} \ \dots \ \tau_2 \ s_1 \ \tau_1 \ s_0 \texttt{ => } \tau_0 \ E$$

where, $\tau_i = [a : b]$, $a \geq 0$ and $b \leq k$. There can be at most n delay terms in the sequence defining the antecedent of the property. A sub-expression s_i may be void of constraints, which causes adjacent delay terms to merge.

Given a time-series $\mathcal{T}$, a predicate alphabet $\mathbb{P}$ derived from knowledge of the domain, and a target event E (the observation for which a PSI-property is to be mined), the PSI-Miner broadly executes the following three steps:

1. *Booleanize* the trace $\mathcal{T}$, obtaining dense Boolean truth intervals for the truth of predicates in $\mathbb{P}$.

2. *Construct the decision tree* by evaluating the predicates in the given alphabet $\mathbb{P}$, and using predicate templates to *learn new predicates*, with the aim of finding a predicate and its position in the prefix sequence that best explain the truth of target E.

3. *Report PSI-Properties* for decision tree nodes where the entropy is zero. During this process, delay terms are scrutinized and the strongest delay terms are computed.

The steps carried out by PSI-Miner are summarized in Figure 1. In the figure *PSI-Miner*$\langle n, k\rangle$ reflects that, concretizing the values of meta-parameters n and k, an instance of the PSI-Miner is created. For different values of n and k PSI-Miner would behave differently. In the remainder of this Chapter, we do not specifically reference the meta-parameters and assume that they are set when using the miner.

The PSI-Miner consists of three core modules:

1. *The Booleanizer*: Booleanizes the trace $\mathcal{T}$ into dense time truth intervals for a given predicate set.

2. *The Decision Tree Builder and Constraint Set Evaluator*: Builds the decision tree and uses the methodology in [1] to evaluate predicates and sequence them appropriately. It also decides if new predicates must be learned.

3. *The predicate learning framework*: Learns new predicates over a given variable set, using preset predicate templates and returns the best predicate it has learned for placement in sequence position b.

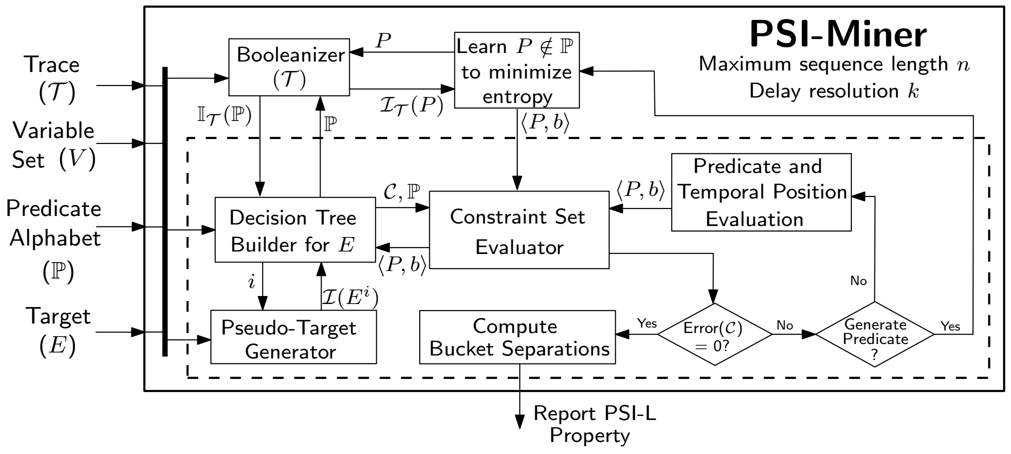
PSI-Miner
Maximum sequence length n
Delay resolution k
Trace ($\mathcal{T}$)
Variable Set (V)
Predicate Alphabet ($\mathbb{P}$)
Target (E)
Booleanizer ($\mathcal{T}$)
Learn $P \notin \mathbb{P}$ to minimize entropy
P
$\mathcal{I}_{\mathcal{T}}(P)$
$\mathbb{I}_{\mathcal{T}}(\mathbb{P})$
$\mathbb{P}$
$\langle P, b\rangle$
Decision Tree Builder for E
$\mathcal{C}, \mathbb{P}$
Constraint Set Evaluator
Predicate and Temporal Position Evaluation
i
$\mathcal{I}(E^i)$
Pseudo-Target Generator
Compute Bucket Separations
Error($\mathcal{C}$) = 0?
Generate Predicate ?
Yes
No
Report PSI-L Property

Figure 2: PSI-Miner: Implementation Modules

2 Tool Output

We express explanations in the form of a sequence of events or predicates (possibly over real-valued variables). In addition to an ordering between events, the timing between adjacent events is key. An explanation for a target event is observed as a prefix to the target. The language used for describing an explanation for an event is therefore called the *Prefix Sequence Inferencing* language (PSI-L).

The language used to describe prefix sequences inferred from time-series data has the following general syntax:

$$\mathcal{S} \texttt{ => } E \quad \text{or} \quad \mathcal{S} \texttt{ => } \tau_1\ E$$

where, $\mathcal{S}$ is a prefix sequence, also known as a sequence expression, of the form $s_n\ \tau_n\ s_{n-1}\ \tau_{n-1}$ $\dots \tau_1\ s_0$. A delay τ_i is a time interval of the form $[a : b]$, $a \leq b$ and each s_i is a Boolean expression of PORVs and events. The length of the sequence expression is n (having n non-temporal sub-expressions). The predicate or event E in a PSI-L formula is assumed to be known. It is in the context of E that $\mathcal{S}$ is learned. E is called the *target* of the PSI-L property. The notation S_i^j is used to denote the expression $s_j\ \tau_j \dots \tau_{i+1}\ s_i$, $0 \leq i \leq j \leq n$. Hence $S \equiv S_0^n$. Note that here, an increase in the index indicates going backward in time, whereas for traces increasing the index indicates a movement forward in time.

For variable set V, the set $\mathbb{D} = \mathbb{R}^{\geq 0} \times \mathbb{R}^{|V|}$ is the domain of valuations of timestamps and variables in V. A data point is a tuple $(t, \eta) \in \mathbb{D}$, $t \in \mathbb{R}^{\geq 0}$ and $\eta \in \mathbb{R}^{|V|}$. For variable $x \in V$, the value of x in the data point (t, η) is given as $\eta[x]$. Boolean and real-valued variables are treated the same in view of the implementation. A Boolean value at a data point is either `1` for *true* or `0` for *false*, and $\{0, 1\} \subset \mathbb{R}$.

Definition 2.1 ***Hybrid System Trace:*** *A trace $\mathcal{T}$ of a hybrid system is an ordered list of tuples (t_1, η_1), (t_2, η_2), $(t_3, \eta_3) \dots (t_d, \eta_d)$, $\forall_{i \in \mathbb{N}_{d-1}} t_i < t_{i+1}$. The length of $\mathcal{T}$, the number of tuples in $\mathcal{T}$, expressed as $|\mathcal{T}|$, is d. The temporal length of $\mathcal{T}$, denoted $||\mathcal{T}||$, is $t_d - t_1$.*

$\mathcal{T}(i)$ denotes the i^{th} data point (t_i, η_i) in trace $\mathcal{T}$.

The time domain for trace $\mathcal{T}$, denoted, $\mathbb{D}(\mathcal{T})$, is the interval $[t_1, t_d]$.

A sub-trace $\mathcal{T}_i^j$ of $\mathcal{T}$ is defined as the ordered list (t_i, η_i), (t_{i+1}, η_{i+1}), $\dots (t_j, \eta_j)$; $i, j \in \mathbb{N}_d$ and $i \leq j$. ■

Definition 2.2 ***Match of a Sequence Expression and a PSI-L formula:*** *The sub-trace $\mathcal{T}_i^j$, $i \leq j$, of $\mathcal{T}$ models the sequence expression $S_l^m ::= s_m\ \tau_{m-1}\ s_{m-1}\ \tau_{m-2} \ldots \tau_l\ s_l$, denoted $\mathcal{T}_i^j \vDash S_l^m$ iff:*

- $\eta_i \vDash s_m$,
- $\exists_{i \leq k \leq j}\ \mathcal{T}_k^j \vDash S_l^{m-1}$ *and* $t_k - t_i \in \tau_{m-1}$.

An PSI-L formula can match at zero or more data points in $\mathcal{T}$. A PSI-L property $s_m\ \tau_{m-1}\ s_{m-1}\ \tau_{m-2} \ldots \tau_1\ s_0 \Rightarrow \tau_0\ E$ matches in $\mathcal{T}$ at $\mathcal{T}(k)$ iff, $\exists_j\ i \leq j \leq k$, $\mathcal{T}_i^j \models S_0^m$, $t_k - t_j \in \tau_0$, and $\mathcal{T}(k) \models E$. The sub-trace $\mathcal{T}_i^j$ is then a witness to the PSI-L property in trace $\mathcal{T}$. ■

3 Tool Directory Structure

This section discusses the files containing the logic of PSI-Miner. The PSI-Miner is composed of compiler/translators and logic that's distributed across multiple C and Python code-bases.

The directory structure of PSI-Miner is as follows:

- **[code]:** PSI-Miner logic distributed across C code-bases *psiMiner.c*, *psiMinerStructs.c/h*, *structs.c/s* and Python code *bpg_constrained.py* or *bpg_multiple.py.*
- **[parsers]:** Lex and Yacc parsers for parsing time-series inputs, configurations, and the Python-C file interface.
- **[build]:** Code and Binaries copied during the build process.
- **[examples]:** Data-sets, configurations and internal experimental data.

The critical code structures are as follows:

- **psiMinerStructs.[c/h])** Contains custom data-structures and methods used by PSI-Miner. The logic contained in the code in psisMinerStructs is core to the operation of PSI-Miner.
- **struct.[c/h]:** Contains common fundamental data structures and methods used by the PSI-Miner and parsers.
- **psiMiner.c:** Contains the supervisory code that uses data structures and functions in the other files.
- **predToPy.[l/y]:** These files contains the grammar/parser for the configuration file. The parser reads the configuration file and translates it into an in-memory configuration structure that can be searched and used as the decision tree is constructed.
- **intervalList.[l/y]:** When the time-series is Booleanized, the truth set of intervals generated by the Booleanizer is parsed into internal data-structures for fast access. These files contain the parser that reads the interval sets generated by the Booleanizer into memory.
- **learnedOP.[l/y]:** The output of the predicate learning process is a list of predicates, their gains and truth intervals. These are parsed by "learnedOP" and stored in custom data structures of the tool.
- **bpg_constrained.py/bpg_multiple.py:** This is the python script that generates (learns) new predicates for use in the construction of the decision. The predicates are generated based on an input configuration for the script. This configuration informs the script of the present state of the decision node, that is the current end match truth intervals for each sub-sequence expression, and the target's truth intervals.

4 Installation Instruction

The tool requires the following libraries and packages to run:

- C/C++ compilers: gcc was used to build the tool
- Lexical Analyzer *lex* or *flex*
- Bison
- Python 3 Libraries for running python scripts: *pandas*, *csv*, *re.*

To build the tool run the *build script,* `"build.sh"`. The build scripts is written in Linux Bash, and when executed copies the code-base into the *[build]* directory and compiles the parsers and the property miner into a single executable binary `"psiMiner"`.

Note*: For the branch "MultiTraceSupport" (see Sec. 5.2), if copying the binary of psiMiner for use outside the directory [build], ensure that you also copy the binary "Booleanize".*

5 Branches, and Tool Usage

PSI-Miner is maintained on GitHub at `https://github.com/antoniobruto/PSIMiner`. The tool has 3 core branches, each representing different versions of the tool. Each branch contains an enhancement to PSI-Miner and are treated independently in this section. The master branch is the fundamental branch. All other branches mentioned in this document inherit learning abilities from the master, other branches can do whatever the master can do.

5.1 Branch: Master

The master branch contains code for mining properties using a given knowledge base, that is, the predicates used to learn properties are provided to PSI-Miner as an input. The learning problem here will identify the *best* predicates, amongst those provided, their truths, and temporal position in the sequence-expression $s_n \ \tau_n \ s_{n-1} \ \tau_{n-1} \ \ldots \ \tau_1 \ s_0$. The predicates provided to PSI-Miner are in general predicates over the variables in the time-series.

5.2 Branch: MultiTrace Predicate Learning

In addition to providing known predicates as an input to PSI-Miner, PSI-Miner's "multiTrace-Support" branch supports learning the predicates using Simulated Annealing, from the data-set provided. It is to be noted, that while this is supported, learning predicates incurs an computation time overhead that depends on the simulated annealing parametes used for predicate learning.

5.3 Branch: Mining Properties to Differentiate Events/Trace Classes

While most studies for mining properties have focussed on trace classification, PSI-Miner, in general, tries to learn *causes* for chosen events. While it is also possible to use PSI-Miner to classify traces, we wish to additionally allow PSI-Miner to learn properties to differentiate between *events*, *predicates* and *traces.* For predicates P and Q, we wish to identify why P occurs and why Q occurs with the understanding that there is a different action sequence that causes P than that which causes Q. Treating P and Q as trace labels, we are also able to distinguish between classes of time-series.

5.4 Configuration Parameters

PSI-Miner parameters are set through a configuration file. The configuration file for the *master*, and *multiTraceSupport* branches is identical, while additional configuration parameters are added for the branch *diffMiner*. We first deal with the common configuration parameters and then introduce the parameters for *diffMiner*.

The tool's configuration file uses a number of parameters that control its execution. An example configuration is provided in Figure 3. The configuration parameters are as follows:

```
seqLength=10
delayRes=1
traceFile=trafficExpt.csv
depth=5
bestPredCount=10
tmin=1
tmax=10
learnMode=2

start 2

x 2 1
y 3 1
v 4 1
a 5 1

begin
        x>=10
        x<=15
        y>=5
        y<=10
        v>=100
        a<=3.8
        a>=4
end

RES = (1) && (2) && (3) && (4)
HIGHSPEED = (17)
(6)
(7)
```

Figure 3: PSI-Miner configuration file structure.

- ***verbosity:*** The tool presently supports two verbosity levels, 0 and 1. When verbosity is set to 1, all internal computations, function calls, variable instantiations, and the state of the decision tree as mining progresses, are printed. This helps to debug and verify computations. When the value is 0 the tool runs without producing expert level information and only produces the mined PSI properties.

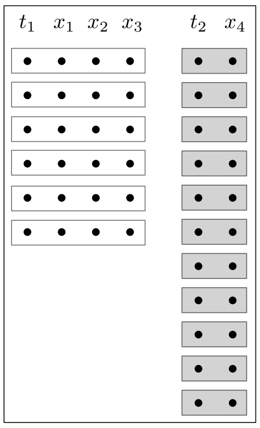

Figure 4: Trace file format: each variable has values associated with a time reference, multiple time references may exist

- ***traceFile:*** This allows the user to specify the path to files containing the time-series. Each time-series must be provided as a csv file with a column for each variable. There must be a time-reference column. The time-reference may be common for the all variables, or custom time-reference columns may be used for variables that are sampled at a different frequency.
- ***delayRes:*** The time separation between sub-expression is given as k. This parameter specifies the resolution with which the sequence expression is built, signifying the temporal separation between two buckets in the prefix sequence.
- ***seqLength:*** The length of the prefix in terms of the number of sub-expressions, the number of steps of k, the number of bucket position to be considered by the tool is indicated as n. The effective window of time over which the prefix sequence is generated is then $n \times k$.
- ***depth:*** The maximum depth of the decision tree is given by this parameter.
- ***learnMode:*** PSI-Miner operates in one of the following four modes:
  1. *Knowledge Only*: No new predicates are learned. The miner only uses predicates in the predicate alphabet.
  2. *Learn On Zero Gain*: A new predicate is only learned if no improvements in Gain are found using the predicates in $\mathbb{P}$.
  3. *Ignorance is Bliss*: The miner will ignore the knowledge it has about the predicate alphabet and always us the predicate learning framework to guide its decisions.
  4. *Best of Both Worlds*: At each decision node, the miner evaluates the predicates it is provided in $\mathbb{P}$ and predicates learned by the predicate learning framework to inform its decision.

  The index is used to specify the mode in which PSI-Miner operates.
- ***bestPredCount:*** When using local search for generating predicates, m specifies the number of predicates that are returned to PSI-Miner by the value search algorithm.
- ***tmax:*** The value of *tmax* specifies the maximum value of temperature used in the simulated annealing for the initialization.

- ***tmin:*** The value of *tmin* specifies the minimum value of the temperature used in the simulated annealing algorithm.
- ***start:*** The keyword ”start” specifies that what comes thereafter is the set of symbols representing variables in the time-series. Along with ”start” an index is specified that indicates which row of the ”csv” file is the first valid row. Only rows with this index and greater are used by the tool.

  The symbols are specified, one on each new line, after the ”start” keyword in the following format:

  $$\langle identifier\rangle\ \langle value\text{-}column\rangle\ \langle time\text{-}ref\text{-}column\rangle$$

  For instance the following line,

  ```
  x 5 2
  ```

  indicates that the identifier `x` is a signal of interest, with a value located in column 5 (counting starts at 1), and the time reference for a value in any entry is located in column 2.
- ***begin:*** After the symbol set is specified, the keyword ”begin” is used to specify the list of predicates, with one predicate specified thereafter on each new line.

  A predicate is specified as follows:

  $$\langle identifier\rangle\ \langle inequality\rangle\ \langle arithmetic\text{-}expression\rangle$$

  Examples of valid predicates: ”`x<=10`”, ”`y>=3.2`”, ”`x>=-3.433`”, ”`x<=y+3*z`”.

  Each predicate is assigned an index based on its position in the list. The first predicate is assigned index ”1”, with the index incremented for each predicate thereafter. These indices are used to develop more complex Boolean expressions over the predicate list.
- ***end:*** The list of predicates is terminated using the ”end” keyword. A list of Boolean expressions is listed after this keyword.

  The list of expressions is used by the tool as its predicate alphabet. To use the predicates listed earlier as is, the indexes of the predicates that should be used by the tool should be listed, with one index per line.

  Each Boolean expression is a combination of indexes from the list of predicates, written in conjunctive normal form.

  A Boolean expression may also be aliased, giving it a name that is meaningful to the domain, by prefixing it as follows:

  $$\langle alias\rangle = \langle CNF\text{-}Boolean\text{-}expressions\text{-}over\text{-}indices\rangle$$

  A Boolean expression that does not have an alias is given an internal name by the tool. The name for the Boolean expression is `KP`¡index¿, where the index is the index of the Boolean expression in the list of expressions, with an index of 1 referring to the first Boolean expression after the ”end” keyword.

  For example the following are two Boolean expressions over some indices, the first is aliased, while the second is not.

  ```
  LOWVOLTAGE = (1) && (2)
  (4)
  ```

The two additional parameters/inputs for *DiffMiner* are as follows:

- ***exDepth:*** Given a reason S_P that explains a predicate or time-series class P, DiffMiner learns a property S_P^Q to explain the series of actions that cause Q in relation to the sequence-expression S_P. To construct S_P^Q PSI-Miner is allowed to extend the decision tree depth bound by *exDepth.*
- ***targets begin:*** A set of numbered targets is chosen from the listed Boolean expressions in the last section of the configuration in Figure 3. These targets are specified within "*targets begin*" and "*end*" markers.

5.5 Usage Instructions

Using the tool involves executing a single command:

```
./psiMiner <config> [<bias>]
```

The bias parameter is experimental and optional. It is used to control the generation of the decision tree with bias towards explaining one of the two truths of the target. For a target E, if the bias is 1 then the tool tries to generate properties that explain E as true, while if it is 0 the generation of properties is biased towards explaining $\neg E$.

6 Data Structures and Methods

*** contact authors ***

7 Disclaimer

Applying this tool to learn temporal causal rules in any domain requires expert knowledge in the domain in which it is used. We do not consider this document to be complete and not all functions and parameters of the tool have been explained in this document. This is due to the fact that some aspects of the tool are experimental, are still in development, or are targetted towards special applications and limited audiences.

For reporting any bugs or for contacting the authors, please visit `cse.iitkgp.ac.in` or e-mail:

- **Pallab Dasgupta:** pallab@cse.iitkgp.ac.in
- **António A. B. da Costa:** antonio@iitkgp.ac.in

Acknowledgements

The authors gratefully acknowledge Intel corporation CAD SRS funding for the partial financial support of this research.

References

[1] Bruto da Costa, A. A., Frehse, G., and Dasgupta, P. Flexible mining of prefix sequences from time-series traces. *CoRR abs/1905.12262* (2019).