Incremental Measurement of Model Similarities in Probabilistic Timed Automata Learning

Salvador Fernandez Covarrubias

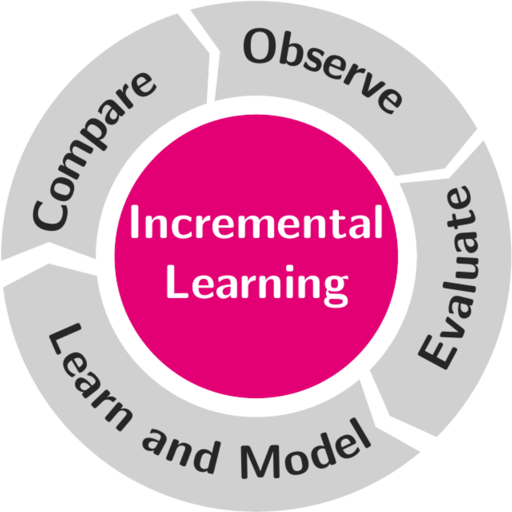
Institute for Software Systems

August 1, 2019

Objective

- Learning, measuring and modelling observations of probabilistic timed automata *on-the-fly*, applying the following concepts:
  - Passive learning of observations, by fitting the curve of the data to mathematical equations.
  - Incremental measurement and modelling of observations, based on Euclidean distances and cost functions.
  - Model similarity evaluations, utilizing graph matching techniques.

Objective



Observe
Evaluate
Learn and Model
Compare
Incremental Learning

Contents

Timed Automata

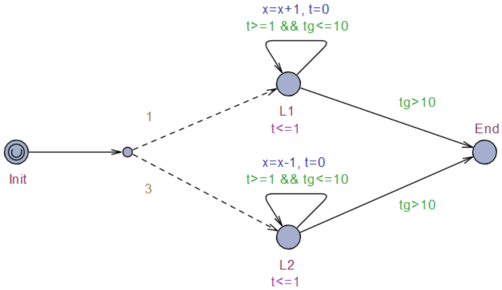
Definition

A timed automaton A is a tuple $< L, L_0, \Sigma, X, I, E >$ where,

- L is a finite set of locations.
- L_0 is the initial location, $L_0 \in L$.
- Σ is a finite set called the alphabet or also actions from A.
- X is a finite set of clocks.
- I labels each location with some clock constraint $\Phi(X)$, which represents a mapping from X to the set $\mathbb{R}$ of non-negative real numbers.
- $E \subseteq L \times \Sigma \times 2^X \times \Phi(x) \times L$ is a set of transitions. A transition $\langle l, a, \varphi, \lambda, l' \rangle$ represents an edge from a location l to location l' given an action $a \in \Sigma$, $\varphi \in \Phi(x)$ is a clock constraint over X that specifies when a transition is enabled, and $\lambda \subseteq X$ represent a clock reset function.

UPPAAL - Models

- UPPAAL is a tool for modeling, simulating and verifying real-time systems. Nevertheless, we only use it to model and simulate probabilistic timed automata.
- Locations are identified by labels and can be bound to invariants, which may allow time to elapse while a location remains idle.
- Edges represent transitions and may be composed of guards, actions or assignments; where guards represent a condition that an edge must satisfy.
- Branch points allow automata to choose transitions non-deterministically, by assigning weighted-probability values to edges.



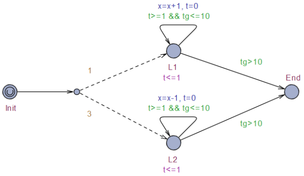
UPPAAL - Simulations

- UPPAAL provides a query language that allows to visualize the values of expressions along simulated runs. The syntax of the queries is as follows:

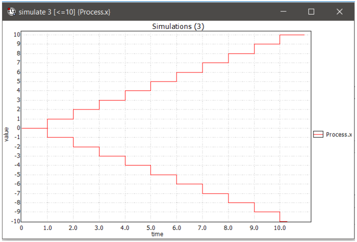
$$simulate\ N\ [<= bound]\ \{E1, \ldots, Ek\}$$

- where N is a natural number indicating the number of simulations to be performed, *bound* is the time bound of the simulations, and $\{E1, \ldots, Ek\}$ are the expressions that are to be monitored.

Example



Observed Model



Simulation: *simulate 3* $[\leq 10]$ *Process.x*

Data Fitting

- We consider an observation as the value of a variable in an exact time, obtained from a simulation run.
- Gathering more than one observation and fitting the curve of the data, allows us to derive equations that express the behavior of a variable given a period of time.
- A simulation run is a set of observations that are transformed to a set of equations.

Observation	Time	Value
1	0	-1
2	1	-2
3	2	-3
4	3	-4
5	4	-5
6	5	-6
7	6	-7
8	7	-8
9	8	-9
10	9	-10

OBS1	OBS2	OBS3	OBS4	OBS5	OBS6	OBS7	OBS8	OBS9	OBS10

- $x = x - 1$ from times 0-3

Data Fitting

- We consider an observation as the value of a variable in an exact time, obtained from a simulation run.
- Gathering more than one observation and fitting the curve of the data, allows us to derive equations that express the behavior of a variable given a period of time.
- A simulation run is a set of observations that are transformed to a set of equations.

Observation	Time	Value
1	0	-1
2	1	-2
3	2	-3
4	3	-4
5	4	-5
6	5	-6
7	6	-7
8	7	-8
9	8	-9
10	9	-10

OBS1	OBS2	OBS3	OBS4	OBS5	OBS6	OBS7	OBS8	OBS9	OBS10

- $x = x - 1$ from times 0-4

Data Fitting

- We consider an observation as the value of a variable in an exact time, obtained from a simulation run.
- Gathering more than one observation and fitting the curve of the data, allows us to derive equations that express the behavior of a variable given a period of time.
- A simulation run is a set of observations that are transformed to a set of equations.

Observation	Time	Value
1	0	-1
2	1	-2
3	2	-3
4	3	-4
5	4	-5
6	5	-6
7	6	-7
8	7	-8
9	8	-9
10	9	-10

OBS1	OBS2	OBS3	OBS4	OBS5	OBS6	OBS7	OBS8	OBS9	OBS10

- $x = x - 1$ from times 0-5

Data Fitting

- We consider an observation as the value of a variable in an exact time, obtained from a simulation run.
- Gathering more than one observation and fitting the curve of the data, allows us to derive equations that express the behavior of a variable given a period of time.
- A simulation run is a set of observations that are transformed to a set of equations.

Observation	Time	Value
1	0	-1
2	1	-2
3	2	-3
4	3	-4
5	4	-5
6	5	-6
7	6	-7
8	7	-8
9	8	-9
10	9	-10

OBS1	OBS2	OBS3	OBS4	OBS5	OBS6	OBS7	OBS8	OBS9	OBS10

- $x = x - 1$ from times 0-6

Data Fitting

- We consider an observation as the value of a variable in an exact time, obtained from a simulation run.
- Gathering more than one observation and fitting the curve of the data, allows us to derive equations that express the behavior of a variable given a period of time.
- A simulation run is a set of observations that are transformed to a set of equations.

Observation	Time	Value
1	0	-1
2	1	-2
3	2	-3
4	3	-4
5	4	-5
6	5	-6
7	6	-7
8	7	-8
9	8	-9
10	9	-10

OBS1	OBS2	OBS3	OBS4	OBS5	OBS6	OBS7	OBS8	OBS9	OBS10

- $x = x - 1$ from times 0-7

Data Fitting

- We consider an observation as the value of a variable in an exact time, obtained from a simulation run.
- Gathering more than one observation and fitting the curve of the data, allows us to derive equations that express the behavior of a variable given a period of time.
- A simulation run is a set of observations that are transformed to a set of equations.

Observation	Time	Value
1	0	-1
2	1	-2
3	2	-3
4	3	-4
5	4	-5
6	5	-6
7	6	-7
8	7	-8
9	8	-9
10	9	-10

OBS1	OBS2	OBS3	OBS4	OBS5	OBS6	OBS7	OBS8	OBS9	OBS10

- $x = x - 1$ from times 0-8

Data Fitting

- We consider an observation as the value of a variable in an exact time, obtained from a simulation run.
- Gathering more than one observation and fitting the curve of the data, allows us to derive equations that express the behavior of a variable given a period of time.
- A simulation run is a set of observations that are transformed to a set of equations.

Observation	Time	Value
1	0	-1
2	1	-2
3	2	-3
4	3	-4
5	4	-5
6	5	-6
7	6	-7
8	7	-8
9	8	-9
10	9	-10

OBS1	OBS2	OBS3	OBS4	OBS5	OBS6	OBS7	OBS8	OBS9	OBS10

- $x = x - 1$ from times 0-9

Data Learning and Measurement

Data Learning

- An equation trace is a list of mathematical equations organized in chronological order, that represent the behavior of a simulation run from an observed automaton.
- The learned model *lm* represents the learned automaton, which is created based on the equations derived from the observations.

Algorithm 2 Learner

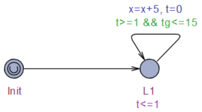
```
procedure LEARNER(equationTrace)
    parent ← lm.getInitialLocation()
    for each equation ∈ equationTrace do
        directSuccessors ← lm.getDirectSuccessors(parent)
        if directSuccessors.length > 0 then
            distances ← measureDistances(directSuccessors,equation)
            lastModifiedNode ← incrementalLearning(distances)
        else
            lastModifiedNode ← addNodeToLearnedModel(equation)
        end if
        parent ← lastModifiedNode
    end for
end procedure
```

Data Modeling

- Assuming that we derived the following equation list from the data of a simulation run:
    - $EQ1 \rightarrow x = x + 5,\ time \rightarrow (0s - 15s)$
    - $EQ2 \rightarrow x = x + 6,\ time \rightarrow (15s - 30s)$
    - $EQ3 \rightarrow x = x + 10,\ time \rightarrow (30s - 45s)$
- Learned Model (lm):

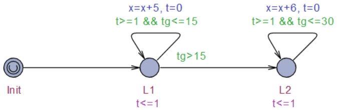
Data Modeling

- Assuming that we derived the following equation list from the data of a simulation run:
  - $EQ1 \rightarrow x = x + 5,\ time \rightarrow (0s - 15s)$
  - $EQ2 \rightarrow x = x + 6,\ time \rightarrow (15s - 30s)$
  - $EQ3 \rightarrow x = x + 10,\ time \rightarrow (30s - 45s)$
- Learned Model (lm):



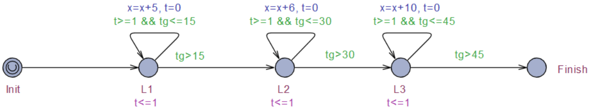
Data Modeling

- Assuming that we derived the following equation list from the data of a simulation run:
  - $EQ1 \rightarrow x = x + 5, \; time \rightarrow (0s - 15s)$
  - $EQ2 \rightarrow x = x + 6, \; time \rightarrow (15s - 30s)$
  - $EQ3 \rightarrow x = x + 10, \; time \rightarrow (30s - 45s)$
- Learned Model (lm):



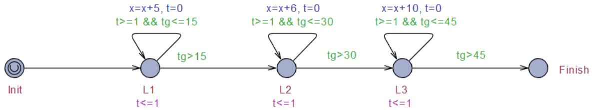
Data Modeling

- Assuming that we derived the following equation list from the data of a simulation run:
  - $EQ1 \rightarrow x = x + 5,\ time \rightarrow (0s - 15s)$
  - $EQ2 \rightarrow x = x + 6,\ time \rightarrow (15s - 30s)$
  - $EQ3 \rightarrow x = x + 10,\ time \rightarrow (30s - 45s)$
- Learned Model (lm):



Data Modeling

- Assuming that we derived the following equation list from the data of a simulation run:
  - $EQ1 \rightarrow x = x + 5,\ time \rightarrow (0s - 15s)$
  - $EQ2 \rightarrow x = x + 6,\ time \rightarrow (15s - 30s)$
  - $EQ3 \rightarrow x = x + 10,\ time \rightarrow (30s - 45s)$
- Learned Model (lm):



What happens when we derive different equations in other simulation runs?
For example: $EQ1 \rightarrow x = x + 1,\ time \rightarrow (0s - 10s)$

Data Learning and Measurement

Data Measurement

Algorithm 3 Measure-Distances

```
procedure MEASURE-DISTANCES(directSuccessors, observedEquation)
    timeSteps ← observedEquation.timeSteps
    observedFunction ← observedEquation.fittedFunction
    for each neighbor ∈ directSuccessors do
        neighborFunction ← neighbor.fittedFunction
        neighborPoints ← evaluatefunction(neighborFunction, timeSteps)
        observedPoints ← evaluatefunction(observedFunction, timeSteps)
        distance ← getEuclideanDistance(observedPoints, neighborPoints)
        distance ← 1/(1 + distance)
        distanceList.push(distance)
    end for
    return distanceList
end procedure
```

The Cost Function

- The closest distance among all direct successors (if any) is chosen.

$$Cost = nodeCount * nodeCost + propagation_f * functionalityCost \\ + propagation_t * timeCost$$

- Important to notice that every cost may or may not be the same, and that we always consider the lowest cost as the best option.

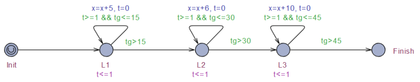
Algorithm 4 Incremental-Learning. An observed equation is added to the learned model as a new node or as a merged node, based on the closeness that it has among direct successors.

Input: similarity threshold *sim_th*, given by the user

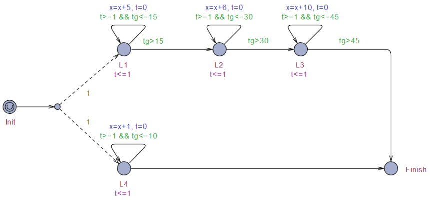
```
procedure INCREMENTAL-LEARNING(distances, observedEquation)
    closestNode ← getClosestDistanceAboveThreshold(distances, sim_th)
    farNode ← getClosestDistanceBelowThreshold(distances, sim_th)
    if closestNode.isNotEmpty() then
        nextNode ← addLeastExpensiveChange(closestNode, observedEquation)
        return nextNodeToTraverse
    end if
    if farNode.isNotEmpty() then
        nextNode ← addNodeToLearnedModel(farNode, observedEquation)
        return nextNode
    end if
end procedure
```

Node Addition Cost

- Assuming the following observation: $EQ1 \rightarrow x = x + 1$, $time \rightarrow (0s - 10s)$
- No functionality from the model is modified, but an extra location is added.
- $Cost = nodeCount * nodeCost$.



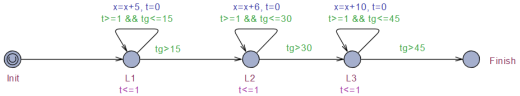
Learned Model



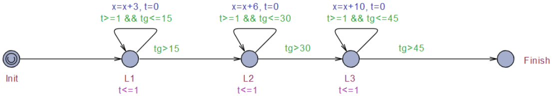
Learned Model After Addition

Node Replacement Cost

- Assuming the following observation: $EQ1 \rightarrow x = x + 1$, $time \rightarrow (0s - 10s)$
- Functionalities and time constraints of locations may be modified.
- $Cost = propagation_f * functionalityCost + propagation_t * timeCost$, where $propagation_f$ is the distance of the old functionality $x = x + 5$ and the new functionality $x = x + 3$, and $propagation_t$ the difference of time constrains in terms of second (e.g. 5 seconds).



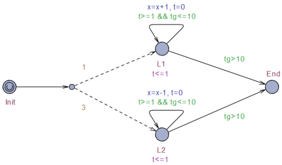
Learned Model



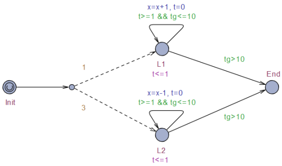
Learned Model After Replacement

Model Similarities Comparison

The goal is to match the nodes and edges of two automata.



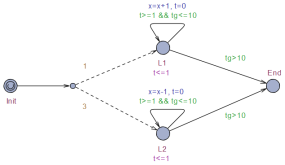
Observed Model



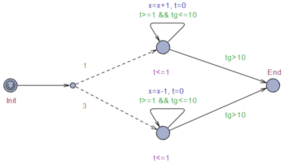
Learned Model

Model Similarities Comparison

We first strip the labels of the learned model.



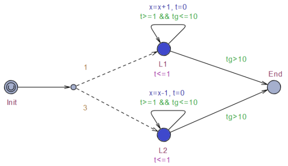
Observed Model



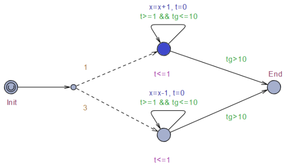
Learned Model

Model Similarities Comparison

Then we perform a simultaneous *breadth-first-search* and compare the functionalities and time constraints of locations against each other.



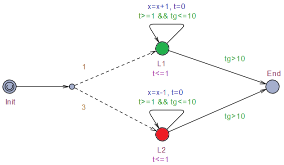
Observed Model



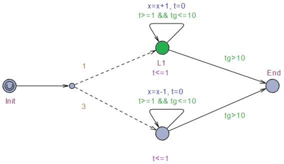
Learned Model

Model Similarities Comparison

Finally, we match the closest locations, based on their functionalities and time constraints.

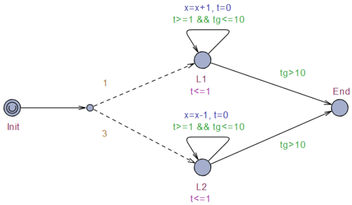


Observed Model

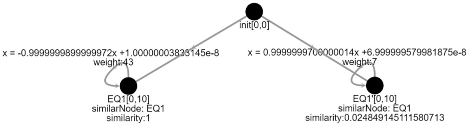


Learned Model

Automata Representation as Graphs

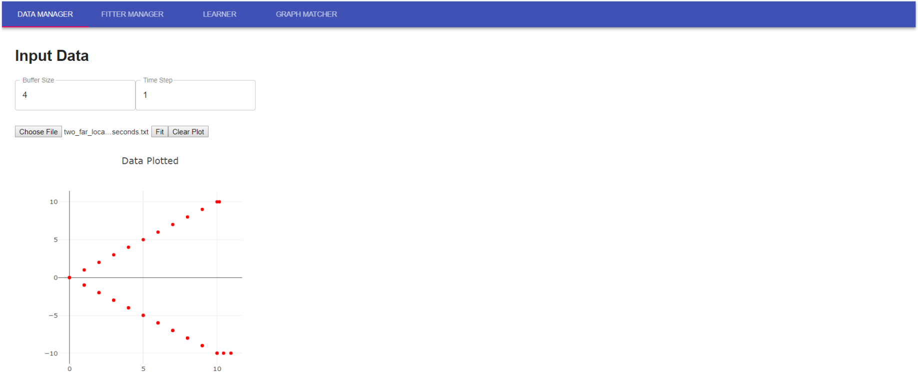


UPPAAL Automaton



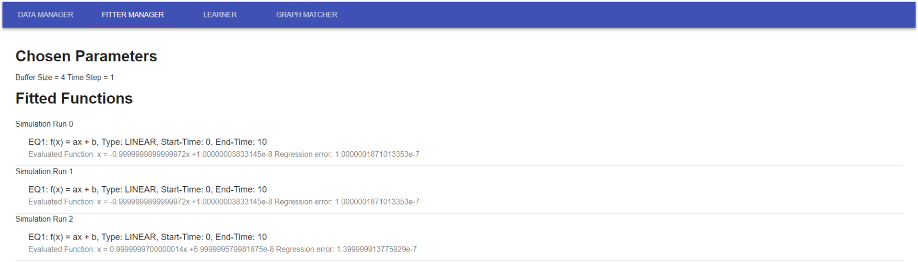
Graph Automaton

Implementation



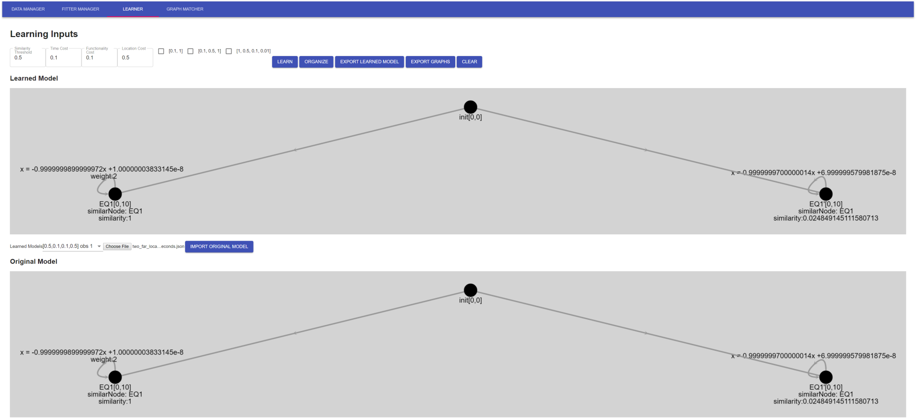
Data Manager

Implementation



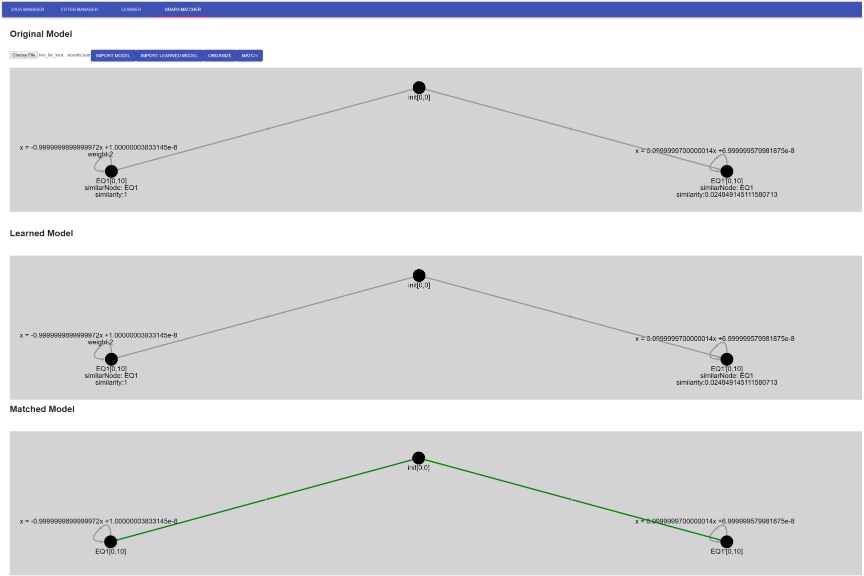
Fitter Manager

Implementation



Learner

Implementation

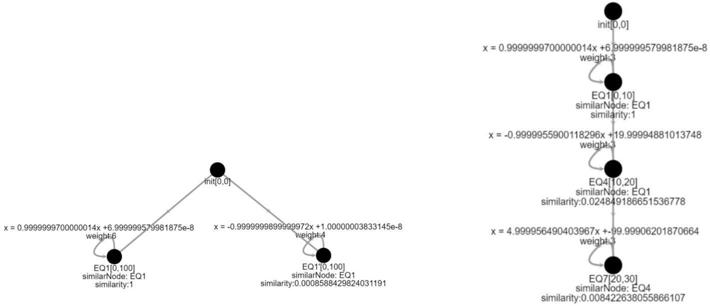


Experiments

- In the following experiments, we chose the values of 0.1, 0.5 and 1 as the possible values for the parameters of the incremental learning algorithm (*similarity threshold*, *functionality cost*, *time cost*, *addition cost*).
- Each model was learned with 81 different combinations, restricted by the previously mentioned set of values.

Experiment 1

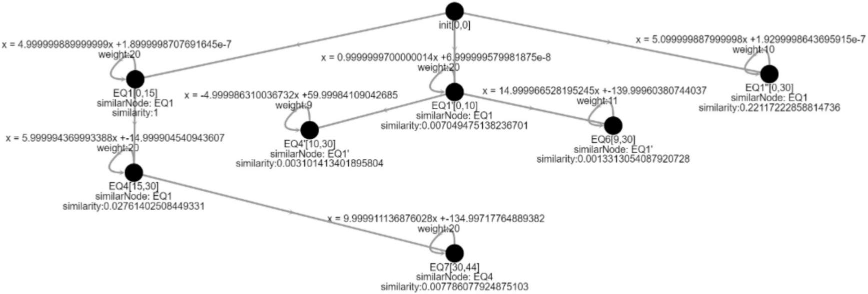
- For these experiments, all of the combinations allowed the algorithm to successfully learned the two observed models.
- Every location has a very different functionality.
- Perfect learning was possible because the *similarity threshold* could always distinguish the functionality of each location and because the models are not complex or ambiguous.



Observed Models

Experiment 2

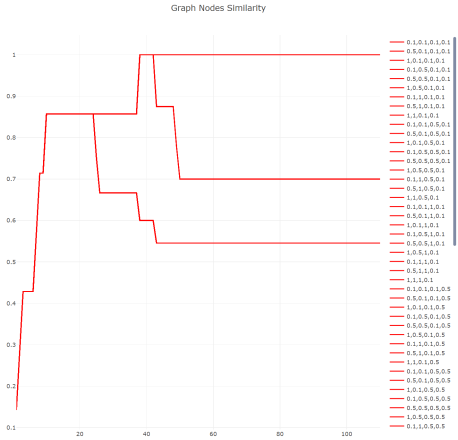
- The complexity of this model was increased by combining the previous models in one, along with extra locations with very close functionalities.
- At the end, 63 out of 81 combinations could match 54% of the observed model, 6 matched 70%, and only 12 combinations could match it by 100%.



Observed Model

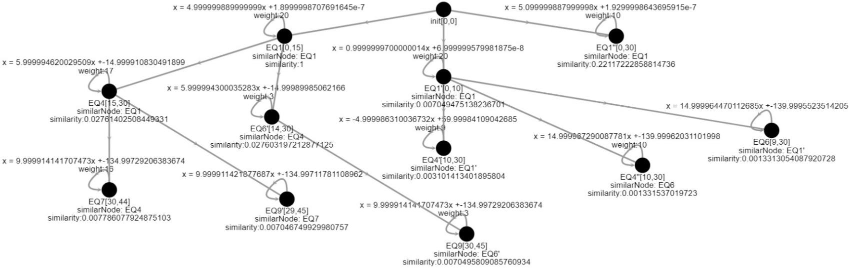
Experiment 2 - Incremental Learning Progress

- The model was simulated 50 times.
- The x-axis represents the number of observations.
- The y-axis represents the similarity between the observed model and the learned model.
- The legends represent the set of combinations.



Experiment 2 - Analysis

- Some locations have very similar functionalities and time constraints.
- Sometimes the simulation data was not sufficient for the tool to fit the exactly same equation.
- Very extreme cases of the cost function forced the learner to identify any miscalculation or noise from the observations as new functionalities, which sometimes was never the case.



Learned Model With Extra Locations

Conclusion

- Learning and recreation of models was successful, by analyzing and measuring observations in an incremental fashion, using Euclidean distances and cost functions.
- Our approach is able to detect noisy observations, but it was not possible to track the origin of the noise, as there is no interaction with the observed automaton like in active learning.
- Unfortunately, the implementation and concept of this approach is not able to detect optimal combinations on the fly.

Future Work

- Automata minimization. Working with a fixed number of locations would ensure having a canonical representation of automata, thus reducing complexity and ambiguities while learning and matching models.
- We could extend the concept of incremental learning to create graph reduction routines which would mainly consist of evaluating the cost of eliminating nodes by merging their functionality and time constraints with other similar nodes.

References

[1] Rajeev Alur. Timed automata. In *International Conference on Computer Aided Verification*, pages 8–22. Springer, 1999.

[2] Endika Bengoetxea, Pedro Larranaga, Isabelle Bloch, Aymeric Perchant, and Claudia Boeres. Inexact graph matching by means of estimation of distribution algorithms. *Pattern Recognition*, 35(12):2867–2880, 2002.

[3] Jean Berstel, Luc Boasson, Olivier Carton, and Isabelle Fagnot. Minimization of automata. *arXiv preprint arXiv:1010.5318*, 2010.

[4] Vincenzo Carletti. *Exact and Inexact Methods for Graph Similarity in Structural Pattern Recognition PhD thesis of Vincenzo Carletti.* PhD thesis, Université de Caen; Universita degli studi di Salerno, 2016.

[5] Jan Daciuk. *Optimization of automata*. Gdańsk University of Technology Publishing House, 2014.

[6] Javier Esparza. Automata theory: An algorithmic approach (lecture notes), 2012.

[7] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-match: an algorithm and an implementation of semantic matching. In *European semantic web symposium*, pages 61–75. Springer, 2004.

[8] Marta Kwiatkowska, Gethin Norman, David Parker, and Jeremy Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29(1):33–78, 2006.

[9] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.

[10] Viktor Losing, Barbara Hammer, and Heiko Wersing. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, 275:1261–1274, 2018.

[11] Alexander Maier. Online passive learning of timed automata for cyber-physical production systems. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, pages 60–66. IEEE, 2014.

[12] Jan Ramon, Maurice Bruynooghe, and Wim Van Laer. Distance measures between atoms. In *CompulogNet Area Meeting on Computational Logic and Machine Learning*, pages 35–41, 1998.

[13] Laura A Zager and George C Verghese. Graph similarity scoring and matching. *Applied mathematics letters*.

Thank you!