Efficiently learning timed system models from observations

Sicco Verwer S.E.VERWER@TUDELFT.NL
Delft University of Technology, P.O. Box 5031, 2600 GA, Delft, the Netherlands

1. Learning timed models efficiently

We describe an efficient algorithm for learning a timed system model from observations. The algorithm is based on the state merging method for learning a deterministic finite state automaton (DFA). This method and its problem have been the subject of many studies within the grammatical inference field, see e.g. (de la Higuera, 2005). Consequently, it enjoys a sound theoretical basis which can be used to prove properties of our algorithm. For example, while it has long been known that learning DFAs is NP-complete, it has been shown that DFAs can be learned in the limit from polynomial time and data (efficiently in the limit) using a state merging method.

A DFA is a language model. A language is a set of finite sequences of symbols $\sigma = s_1 s_2 \dots s_n$ known as strings (or words). Adding time to a string can be done by giving every symbol a time value $t \in \mathbb{N}$[1], resulting in a sequence of symbol-time value pairs $\tau = (s_1, t_1)(s_2, t_2) \dots (s_n, t_n)$. Every time value in such a *timed string* represents the time that has elapsed since occurrence of the previous symbol. A set consisting of timed strings is called a *timed language*. There are two main reason why we want to learn a timed system model instead of an untimed model. First, in many applications the data that is obtained by observing a system contains timestamps. For example, this is the cse when time series data is obtained from sensor signals. Second, we believe that learning a timed model from such data is more efficient than learning an untimed model from this data. The reason for this belief is that, while it is possible to construct for any timed model an equivalent untimed model by sampling the time values, this untimed model is of size exponential in the size of the timed model. Thus, an efficient algorithm that learns a timed system using an untimed model is by definition an inefficient algorithm since it is exponential in time and data in the size of the timed model. In contrast, we show it is actually possible to learn certain types of timed models efficiently. Naturally, we want to focus on learning models that are efficiently learnable.

We assume familiarity with the theory of languages and automata. The timed models we consider are known as timed automata (TA) (Alur & Dill, 1994). In these models, time is represented using a finite number of *clocks*. A clock can be thought of as a stopwatch that measures the time since it was last reset. When a transition of a TA executes (fires) it can optionally reset the value of a clock. This clock then measures the time since the last execution of this transition. The value of clocks are used to constrain the execution of transitions in a TA. A transition in a TA can contain a boolean constraint, known as a *clock guard*, that specifies when this transition is allowed to be executed depending on the values of clocks. If the clock values satisfy the constraint, the transition can be executed. Otherwise, it cannot be executed. Thus a transition is executed only if the TA is currently in its source (or parent) state, the current symbol is equal to the transition label, and the current clock values satisfy the transitions clock guard. In this way, the execution of a TA depends not just on the symbols, but also on the time values occurring in a timed string.

We focus on learning algorithms for a simple timed model known as a deterministic real-time automaton (DRTA) (Dima, 2001) based on the state merging method. A DRTA is a TA that contains a single clock that is reset by every transition. Hence, the execution of a DRTA depends on the symbols, and on the time between two consecutive events of a timed string. The reason for restricting ourselves to these simple TAs is that the beforementioned learnability results for DFAs can quite easily be adapted to the case of DRTAs. In other words, we show that DRTAs are efficiently learnable using a state merging method. Due to the expressive power of clocks, we also show this does not hold for deterministic TAs (DTAs) in general: any algorithm that tries to learn a DTA $\mathcal{A}$ will sometimes require an input set of

[1] It is more common to use the set of reals $\mathbb{R}$ for time values. In practice, there always is a finite precision of time and hence this only makes the timed models unnecessarily complex.

size exponential in the size of $\mathcal{A}$. Other methods we know of for learning timed models try to learn a subclass of DTAs known as event recording automata, see e.g. (Grinchtein et al., 2005). These automata are more powerful than DRTAs. However, due to this extra power the algorithm is inefficient in the amount of data it requires to learn a model.

2. Learning from observations

In previous work, we constructed an algorithm for learning DRTAs from labeled data (Verwer et al., 2007). A very high-level view of this algorithm is given in Algorithm 1. The algorithm starts with a prefix tree $\mathcal{A}$ that is constructed from the input set S. This is a DRTA that is such that: all the clock guards are equal to true (it disregards time values), there exists exactly one execution one path to any state (it is a tree), and it is such that the execution of $\mathcal{A}$ on any timed string from S ends in a state in $\mathcal{A}$. Starting from this prefix tree, our algorithm performs the most consistent merge or split as long as consistent merges or splits are possible. A merge of two states a and b replaces a and b with a new state c that contains all the input and output transitions of both a and b. Afterwards, it is possible that $\mathcal{A}$ contains some non-deterministic transitions. These are removed by continuously merging the target states of these transitions until none are left. This merge process is identical to a merge in the original state merging algorithm.

A transition splitting process is used to learn the clock contraints of the DRTA. A *split* of a transition d, at time t removes d from $\mathcal{A}$ and adds a transition e that is satisfied by all the clock values that satisfy d up to time t, and a transition f that is satisfied by all the clock values that satisfy d starting at t. The timed strings from S that have an execution path over e and f are subsets of the timed strings from S that had an execution path over d. Therefore, the prefix tree is recalculated starting from the new transitions e and f.

Algorithm 1 State merging and splitting DRTAs

Require: A set of timed strings S.
Ensure: The result is a small consistent DRTA $\mathcal{A}$.
  Construct a timed prefix $\mathcal{A}$ tree from S.
  while States can be merged or split consistently **do**
    Evaluate all possible merges and splits.
    Perform the most consistent merge or split.
  end while
  return The constructed DRTA $\mathcal{A}$

There is one important part of Algorithm 1 that we left undefined: What does it mean to be consistent? In the original setting of our algorithm the answer to this question was simple: The algorithm got a labeled data set as input and consistenct was defined using these labels. However, for many applications this setting is unrealistic: Usually, only positive data is available. We adapted our algorithm to this setting by using statistics as a consistency measure for our models.

This seems a very natural thing to do and it has already been done for the problem of learning probabilistic DFAs (or just DFAs if you drop the probabilities) (Kermorvant & Dupont, 2002). The main problem to overcome is to come up with a good statistic for the (dis)similarity of two states based on the prefix trees of their suffixes. We treat two nodes in such a tree as being similar if the distributions with which they generate (next) symbols are not significantly different. In addition, since we deal with a timed model, we require that the distributions generating the time values of these symbols are not significantly different. There properties are tested for every node in the prefix tree using Chi-square and Kolmogorov-Smirnov hypothesis tests. The result is a large amount of p-values, which we combine into a single test using a standard method for multiple hypothesis testing. The orginal method that was used to learn DFAs does not make use of a multiple hypothesis testing method: it simply rejects if any of the tests fails.

References

Alur, R., & Dill, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, *126*, 183–235.

de la Higuera, C. (1997). Characteristic sets for polynomial grammatical inference. *Machine Learning*, *27*, 125–138.

de la Higuera, C. (2005). A bibliographical study of grammatical inference. *Pattern Recognition*, *38*, 1332–1348.

Dima, C. (2001). Real-time automata. *Journal of Automata, Languages and Combinatorics*, *6*, 2–23.

Grinchtein, O., Jonsson, B., & Leucker, M. (2005). Inference of timed transition systems. *Electronic Notes in Theoretical Computer Science*.

Kermorvant, C., & Dupont, P. (2002). Stochastic grammatical inference with multinomial tests. *ICGI '02* (pp. 149–160). Springer-Verlag.

Verwer, S., de Weerdt, M., & Witteveen, C. (2007). An algorithm for learning real-time automata. *Benelearn'07* (pp. 128–135).