Unsupervised Learning of Finite Automata: A Practical Approach

D. Carmel and S. Markovitch

CIS Report #9504
May 1995

For the elimination of any doubt, it is hereby stressed that the staff member and/or the Technion and/or the Technion Research and Development Foundation Ltd. will not be liable for any property damage and/or corporeal damage and/or expense and/or loss of any kind or sort that will be caused or may be caused to you or to anyone acting on your behalf, in consequence of this statement of opinion or this report, or in any connection to it.
Unsupervised Learning of Finite Automata: A Practical Approach

David Carmel and Shaul Markovitch
Computer Science Department
Technion, Haifa 32000
Israel
Email: carmel@cs.technion.ac.il, shaulm@cs.technion.ac.il

Abstract

Unsupervised learning of finite automata has been proven to be NP-hard. However, there are many real situations when such learning is necessary. The work presented here studies the possibilities of using heuristic methods for unsupervised learning of finite automata. An algorithm for learning a finite-state machine based on a sample of its behavior is presented. The algorithm is based on Angluin’s $L^*$ algorithm. When $L^*$ receives a counterexample it extends its current model to agree with the example. It then consults a teacher for resolving uncertainties in the extended model. Our algorithm, instead of using a teacher, resolves the uncertainties by consulting its own model that was generated during the previous learning step. A set of experiments that show the potential merit of the algorithm is presented.

1 Introduction

The problem of inferring finite automata (DFA) from its input/output behavior is a well known study in the machine learning literature, and continues to generate new interest. Finding the smallest finite automata consistent with a given sample of machine’s behavior, has been shown to be NP-Hard [Gol78, Ang78]. It has also been shown that the minimal consistent automata can not be approximated within any polynomial time algorithm [Pit89]. Furthermore, Kearns and Valiant have shown that, even without restrictions on the model representation of the learner, modeling automata from input/output behavior is as hard as factoring Blum integers, inverting RSA, or deciding quadratic residues [KV89].

Thus, passive modeling of a given automaton from arbitrary sample seems to be infeasible. Angluin [Ang87] describes an algorithm that efficiently infers an automata model using a ‘minimal adequate teacher’, an oracle that answers membership and equivalence queries. This algorithm, named $L^*$, is an efficient inference procedure that constructs a minimal DFA consistent with the learned machine. The computational time is polynomial in the number of states of the machine, and the longest counterexample supplied by the teacher.

Another alternative was studied by Rivest and Schapire [RS89]. Their procedure simulates iterated interactions of a robot with an unknown environment, and is based on $L^*$. Instead of an adequate teacher that answers queries, the learner is permitted to experiment with the environment (machine). When the learner is requested by $L^*$ to simulate a membership query for a specific input, it operates the machine and observes the result. If the model’s prediction is different from the actual behavior of the machine, the learner treats it as a counterexample. To experiment with the machine, the learner is required to have an ability to reset it. In the case of an unresetable
machine, Rivest and Schapire’s algorithm uses ‘homing sequences’ instead, a special sequence of actions that according to the machine’s outputs, determines the final state of the machine. Given an input parameter $\delta > 0$, their procedure approximately identifies a minimal consistent DFA, with a probability of at least $1 - \delta$, in time polynomial in the machine’s size, the length of the longest counterexample, and in the logarithm of $\frac{1}{\delta}$.

In this work, we investigate the ability to model an unknown machine from its input/output behavior, without the presence of a teacher and with a very limited ability to experiment with the machine. Our framework simulates a situation where an agent is required to interact with a given machine, and the machine’s behavior has to be modeled to discover an efficient interactive strategy. In contrast to Rivest and Schapire’s procedure, experiments with the machine might be too expensive or even destructive for the learner. The agent has a sample of the machine’s behavior in the past. It can also choose its actions during the interaction process to learn the machine’s behavior. Actions selection can be effected from the requirement to act optimally at the current stage of interaction, but also from the intention to explore the machine.

Our inference procedure is incremental. We assume that the machine can be modeled as a DFA. At any stage of interaction, the learner exploits the machine’s current model to predict the machine’s behavior, and chooses its own action according to the model’s prediction. When model’s prediction is wrong, the agent updates the machine’s model to become consistent with the new counterexample. The learning procedure performs a hill climbing search in the hypothesis space for the minimal model that is consistent with the machine’s behavior in the past, and also with the new counterexample. Following the ‘Occam razor’ principle, we search for the minimal DFA consistent with the machine’s behavior, using heuristic methods that look for the smallest consistent model.

The remainder of the paper is organized as follows: Section 2 defines a framework for the research and describes the related theoretical results that our work is based on. In section 3 we describe and analyze US-L*, an unsupervised version of L* that interactively and unsupervisely infers a model of an unknown machine, consistent with its behavior. Instead of a teacher, US-L* answers membership queries by using its own heuristics, and by treating wrong predictions as counterexamples. We also outline preliminary results of various experiments with the algorithm. It seems that the success of US-L* to construct a reasonable model, depends especially on the sample type. For prefix-closed samples, results are quiet promising. For arbitrary samples, model size can grow up to be the same as the sample size. Section 4 concludes and discusses future work.

2 Background

2.1 Definitions

A deterministic finite automata, DFA, sometimes called a finite state machine FSM, is a 6-tuple $M = (Q, \Sigma_{in}, \delta, q_0, \Sigma_{out}, F)$, where:

- $Q$ is a finite non-empty set of states.
- $\Sigma_{in}$ is the input alphabet. In our notation is the action set of the agent.
- $\delta : Q \times \Sigma_{in} \rightarrow Q$ is a transition function. We extend $\delta$ to the domain $Q \times \Sigma_{in}^*$ in the usual way:
  \begin{align*}
  \delta(q, \lambda) &= q \\
  \delta(q, s \sigma) &= \delta(\delta(q, s), \sigma)
  \end{align*}
  
  $\lambda$ denotes the null string, $s \in \Sigma_{in}^*$ and $\sigma \in \Sigma_{in}$.
- $q_0$ is the initial state of the DFA.
- $F : Q \rightarrow \Sigma_{out}$ is an output function. The DFA is interpreted as a Moore machine. In our notation $F$ returns the output of the environment, given it is found in state $q$.

We define the output of a DFA $M$ on a string of input actions $s \in \Sigma_{in}^*$ as $M(s) = F(\delta(q_0, s))$.

For two strings $s_1, s_2$ we denote the concatenation of them as $s_1s_2$.

An example of the machine’s behavior is a pair $(s, \sigma)$ where $s \in \Sigma_{in}^*$, $\sigma \in \Sigma_{out}$. $\sigma$ annotates the output of the machine for an input sequence $s$. A Sample $D$ is a finite set of examples of the machine behavior. For any example $(s, \sigma) \in D$, we mark $\sigma$ as $D(s)$. We say that a model $M$ is consistent with a sample $D$ iff for any example $(s, \sigma) \in D$, $M(s) = D(s)$.

We say that the learner has a perfect model of the machine if it can predict perfectly its output given any sequence of input actions. Our goal is to construct a perfect model of an unknown machine given a sample of its behavior.

2.2 Identifying a DFA from a Given Sample

Gold [Gol72, Gol78] studied the problem of identifying a DFA from a given sample by representing a DFA by an observation table $(S, E, T)$. $S$ is a prefix-closed set of strings. Each element $s$ of $S$ represents the state $\delta(q_0, s)$. The transition function is represented by $S\Sigma_{in} = \{s\sigma | s \in S, \sigma \in \Sigma_{in}\}$. It is possible that two elements of $S$ represent the same state. $E$ is a suffix-closed set of strings, called tests, that distinguish between states of the machine. An answer for test $e$ on string $s$ is defined as $F(se)$. Two strings $s_1, s_2 \in S$ represent the same state iff their answers are equal for all tests in $E$. $T$ is a two dimensional table, with one row for each element of $S \cup S\Sigma_{in}$, and one column for each element of $E$. An entry of the table, $T(s, e)$, records the output of the machine for the string $se$, $F(se)$.

A table is closed iff for any given $s \in S\Sigma_{in}$ there is a row $s' \in S$ such that $row(s) = row(s')$. A table is consistent iff for any two equal rows in $S$, $s_1, s_2$, and for any $\sigma \in \Sigma_{in}$, $row(s_1, \sigma) = row(s_2, \sigma)$. We say that a DFA $M$ is consistent with an observation table $(S, E, T)$ iff for any entry $(s, e)$ in $T$, $M(se) = T(s, e)$.

A DFA $M$ that is consistent with a closed and consistent table $M = M(S, E, T)$ can be constructed as follows [Ang87]:

- $Q = \{row(s) : s \in S\}$
- $q_0 = row(\lambda)$
- $\delta(row(s), \sigma) = row(s\sigma)$
- $F(row(s)) = T(s, \lambda)$

Theorem 1 (Angluin) If $(S, E, T)$ is a closed and consistent observation table, then the DFA $M(S, E, T)$ is consistent with the table $T$, and any other DFA consistent with $T$ but inequivalent to $M(S, E, T)$, must have more states.

We say that a table $(S, E, T)$ covers a sample $D$, if for any $(d, \sigma) \in D$, there is an entry $(s, e) \in T$ such that $d = se$ and $T(s, e) = D(d)$. We say that a table entry $(s, e)$, is supported by a sample $D$, if there is an example $d \in D$ such that $d = (se, \sigma)$, and $T(s, e) = \sigma$. 

4
Given a closed and consistent table \((S, E, T)\) that covers \(D\), we can construct \(M(S, E, T)\) consistent with \(D\). Thus, the problem of finding a consistent DFA with \(D\) is reduced to the problem of finding a closed and consistent observation table that covers \(D\).

Given a sample \(D\), assume that we try to construct a table \((S, E, T)\) that covers \(D\). It is easy to find \(S\) and \(E\) such that for any \(d \in D\), there are \(s \in S, e \in E\) and \(d = se\). Table identification demands the learner to fill all table entries. Table entries that are supported by \(D\) must be filled by \(T(s, e) = D(se)\). The main question left is how to fill entries not supported by \(D\).

**Definition 1** An entry \((s, e)\) of the table \((S, E, T)\) will be called a permanent entry if \(T(s, e) = D(se)\). \((s, e)\) will be called a hole entry if \(T(s, e)\) has to be guessed by the learner.

Two table entries, \((s_1, e_1)\) and \((s_2, e_2)\) will be called tied if \(s_1e_1 = s_2e_2\).

**Definition 2** An assignment is a vector of output values, that assigns an output value into each hole of a table. An assignment will be called a legal assignment iff tied holes get the same value.

Finding a legal assignment for a given table is easy. For example, the assignment that inserts the same output value for all the holes must be legal. We will call it the *trivial assignment*. The problem becomes much harder if we look for a legal assignment that causes minimal extensions of the table for bringing it to become closed and consistent. We will call such assignment an *optimal assignment*.

**Theorem 2** (Gold) The problem of finding an optimal assignment for an observation table that covers a given sample, is NP-hard.

### 2.3 Supervised Learning of a DFA

Following Gold’s results, an exhaustive search for an optimal assignment for a given table is too hard. Angluin [Ang87] shows how to infer a structure of any machine, in the presence of a ‘minimal adequate teacher’. The teacher directs the learner to fill hole values optimally by answering membership queries. The learner is allowed to ask the teacher membership and equivalence queries. On a membership query, the algorithm asks for the machine’s output for a given string of input actions \(s\). On an equivalence query, the algorithm conjectures that the model and the machine are equivalent. The teacher replies by ‘Yes’ for a right conjecture, or provides a counterexample \(m\), a string which the model and the machine disagree.

The \(L^*\) algorithm maintains an observation table \((S, E, T)\). Initially, \(S = E = \{\lambda\}\) and all table entries are filled by membership queries. In the main loop, \(L^*\) tests the current table for closeness and consistency. If not, \(L^*\) extends the table to become closed and consistent, where new table entries are filled by membership queries. When the table becomes closed and consistent, \(L^*\) constructs \(M(S, E, T)\) and asks for equivalence. If a counterexample is provided by the teacher, the table is extended to include the new example. The algorithm continues until ‘YES’ is replied by the teacher for the conjectured model.

**Theorem 3** (Angluin) \(L^*\) eventually terminates and outputs a minimal equivalent DFA to the Teacher’s DFA. Moreover, if \(n\) is the number of states of the teacher’s DFA, and \(m\) is an upper bound on the length of the longest counterexample provided by the teacher, then the total running time of \(L^*\) is bounded by a polynomial in \(n\) and \(m\).
Another version of $L^*$ is also used by a learning procedure described in Rivest and Schapire [RS89]. The learner holds a model of the machine’s behavior, based on a closed and consistent observation table, and compares the model’s prediction to the actual behavior of the machine. When model’s prediction is wrong, the learner treats it as a counterexample and extends the table to cover the new example exactly as $L^*$ does. Instead of a teacher, their algorithm optimally fills new holes by conducting experiments with the machine. For filling an entry $(s, e)$, the action sequence $se$ is provided to the machine and the machine’s output is inserted into the hole.

3 Unsupervised learning of a DFA: a heuristic approach

An adequate teacher, or an unlimited ability to experiment with the machine, do not exist in most practical situations. What should an agent do in the case when a model of the environment’s behavior is essential for its success, and when it does not have a teacher or experimentation ability? The theoretic results covered in the previous section are disappointing in the sense that inferring a reasonable model for the environment’s behavior seems to be too computationally hard. Construction of a naive model that covers the data is possible but does not seem to bring much benefit. For example, using the trivial assignment for filling an observation table that covers the sample would probably cause the model to be in size of the given data, and therefore, no generalization be made.

We propose to deal with this problem by considering heuristic approaches that search for an optimal assignment for a given observation table. According to the ‘Occam Razor’ assumption, we should look for the smaller model consistent with the given data, in order to receive a model with high prediction power. By using heuristics that try to control the growth of the model during extension, and by taking some limiting assumptions on the given data, we show that a reasonable solution might be found.

This method is analogous to the famous classification problem of constructing a decision tree from a set of pre-classified examples. Finding the smallest decision tree consistent with a given data is known to be NP-Hard [QR89]. However, using heuristic methods such those described in [Qui86], a ‘reasonable’ consistent decision tree can be constructed, in time polynomial in the size of the given data. This method has been shown to be efficient in many practical applications.

The following subsection describes US-L*, an unsupervised version of $L^*$ that interactively constructs a model of an unknown machine from its input/output behavior. At any stage of the learning session, when a counterexample arrives and the table is extended, holes are filled by the current model in hand, but attempt is made to change hole values in order to control the growth of the table.

3.1 Description of the Learning Algorithm

During encounters with the environment, the learning agent holds a consistent model with the environment’s behavior in the past, and exploits the model to predict the environment’s behavior in the future. The model $M$ is based on a closed and consistent observation table $(S, E, T)$ that covers past examples, and $M = M(S, E, T)$. When a new example arrives, it can be a supporting example or a counterexample. For the first case, the algorithm changes hole entries in the observation table, supported by the new example, to become permanent entries. This operation does not change table values so the current model is not changed either. For a counterexample, two cases have to be considered. When the new example contradicts hole values of the table, the algorithm changes these values to become consistent with the example, and marks these entries as permanent entries.
When a counterexample is not covered by the table, the algorithm extends the table to cover it in the same way as \( L^* \) does. Following that, the algorithm arranges the updated table to become closed and consistent again, and constructs a new model consistent with the new table.

New entries of the extended table are filled as follows: When an entry is supported by a past example, it gets the example's output value and it is marked as a permanent entry. When a table entry is not supported by a past example, it gets an output value predicted by the current model, and it is marked as a hole entry. This approach guarantees that at any stage of the learning process, the observation table covers the past examples of the environment's behavior, and the assignment for hole entries of the table is always legal. This claim can be proved by the fact that for any pair of tied entries, if both are permanent entries, both are supported by the same example. If both are hole entries and both were added to the table at the same stage, then they both got their output value from the same model. If one of them is an old entry and one was added later, the new one will get the same value as the second. The current model is consistent with the table, so it must assign the same value for the new entry as the old entry's value.

In spite of the consistency of the accepted model with past behavior of the environment, its size grows without limitations and its success to predict the environment's behavior in future is disputable. The guiding principle of the algorithm is to limit the growth of the model as much as possible. When the algorithm extends the table to cover a new counterexample, it attempts to change holes assignment such that the new example will be covered with minimal needed extensions. If two rows, \( s_1 \) and \( s_2 \), were equal before the new example arrived, and are not equal after table's extension, the algorithm tries to change the assignment to keep the rows equal, and to prevent the necessity to add new states for the current model. The rule for changing the assignment is quiet simple. If two equal rows, \( s_1 \) and \( s_2 \), are not equal after extension, there is a a test \( e \) such that \( T(s_1,e) \neq T(s_2,e) \). If one entry is permanent and the second is a hole, the hole entry get the output value of the permanent. When both are hole entries, the longer one get the output value of the shorter one. Changing a value of a hole entry, causes all its tied entries to get the same value, for keeping the legality of the assignment, but it might also cause other rows to become unequal.

The same heuristic is operated when consistency of the table is checked. In the case that the table is not consistent after extension, there are two equal S-rows, \( s_1 \) and \( s_2 \), and \( \sigma \in \Sigma_{in} \), such that \( row(s_1\sigma) \neq row(s_2\sigma) \). It means that there is a test \( e \) such that \( T(s_1\sigma,e) \neq T(s_2\sigma,e) \). \( L^* \) solves this inconsistency by adding a new test \( \sigma e \) into \( E \), an extension that separates \( row(s_1) \) and \( row(s_2) \) and causes an addition of a new state to the current model. Before adding a new test and separating the two rows, US-L* tries to solve the inconsistency by changing hole assignment. When \( T(s_1\sigma,e) \neq T(s_2\sigma,e) \), hole values are changed according to the same rules as above. Adding the new test is done only after the failure to solve inconsistency by changing hole values. Figure 1 shows a pseudo code of the algorithm.

The suggested heuristic is implemented easily but it is not the only one possible. For further research, our intention is to continue and look for other ones, their attitudes, and the situations in which they succeed to control the size growth of the learned model.

### 3.2 Correctness of US-L*

**Theorem 4** If \((S,E,T)\) is a closed and consistent observation table that covers \( D \), and \( t \) is a new example. Then US-L* eventually terminates, and outputs a closed and consistent table that covers \( D \cup \{ t \} \).

To prove the theorem we need the two following lemmas:
Algorithm: US-L*

Given:
- \( D \): a set of past examples of the machine’s input/output behavior.
- \((S, E, T)\): A closed and consistent observation table that covers \( D \).
- \( M \): The current model. \( M = M(S, E, T) \).

Repeat

let \( t \) be a new example of the machine’s behavior

\[ D \leftarrow D \cup \{ t \} \]

if \( t \) is a supporting example

for each table entry \((s, e)\) that is covered by \( t \), (e.g., \( t = se \))

mark \((s, e)\) as a permanent entry

if \( t \) is a counterexample

for each prefix \( p \) of \( t \)

if \( p \not\subseteq S \cup S\Sigma \)

Add \( p \) into \( S \) and Extend\((S, E, T)\)

else

let \( e \) be a suffix of \( t \) such that \( t = pe \)

if \((p, e)\) \( \in T \)

mark \((p, e)\) as permanent and change its value to \( D(t) \)

\[ \forall s \in S \cup S\Sigma, \text{ such that } row(s) = row(p) \]

if \((s, e)\) is a hole, \((s, e)\) (and its tied entries) \( \leftarrow D(t) \)

move \( p \) into \( S \) and Extend\((S, E, T)\)

mark all entries as not changed

To prevent infinite consistent loop

While not Consistent\((S, E, T)\)

find two equal rows \( s_1, s_2 \in S, \sigma \in \Sigma_{in}, e \in E \), such that \( T(s_1\sigma, e) \neq T(s_2\sigma, e) \)

if both \((s_1\sigma, e)\) and \((s_2\sigma, e)\) are permanent

or both have been changed before \{we must distinguish between rows \( s_1 \) and \( s_2 \}\}

Add \( \sigma e \) into \( E \) and Extend\((S, E, T)\)

else

if one entry is not a hole that was not changed before \{assume \((s_1\sigma, e)\)\)

or both entries are holes that was not changed before and assume \( s_1 \leq s_2 \)

\( T(s_2\sigma, e) \) (and its tied entries) \( \leftarrow T(s_1\sigma, e) \)

mark \((s_2\sigma, e)\) (and its tied entries) as changed

While not Closed\((S, E, T)\)

find \( s \in S\Sigma\) such that \( \forall s' \in S \), \( row(s') \neq row(s) \)

move \( s \) into \( S \) and Extend\((S, E, T)\)

\[ M \leftarrow M(S, E, T) \]

Until forever

Figure 1: US-L*: Unsupervised learning algorithm of a DFA
Lemma 1 The closeness loop at the end of the algorithm, terminates after finite number of iterations, does not change the consistency of the table, and outputs a closed table.

Proof: At each iteration, \( S \) is extended with one row that is distinct from all other rows in \( S \). \( E \) is not extended during the loop. The number of different rows in \( S \) is finite and bounded by \( |\Sigma_{\text{out}}| \). Therefore, the number of iterations must also be finite. Termination occurs only when the table is closed so the loop is terminated and output a closed table. Consistency of the table is not changed during the closeness loop because any row that is added to \( S \) is distinct from any other rows in \( S \). Therefore, inconsistency can not be added to the table. □

Lemma 2 The consistency loop is finite and outputs a consistent table.

Proof: Let’s define an inconsistent point as a tuple \( (s_1, s_2, \sigma, e) \), where \( s_1, s_2 \in S \) and \( \text{row}(s_1) = \text{row}(s_2) \), \( \sigma \in \Sigma_{\text{in}} \), \( e \in E \), and \( T(s_1, \sigma, e) \neq T(s_2, \sigma, e) \). At each iteration, at least one inconsistent point is eliminated, but others might be added to the table. The proof will show that this process must be finite.

First, an extension of \( E \) might cause an addition of inconsistent points to the table. We show that the number of extensions of \( E \) is finite. At each iteration, \( E \) is extended with at most one element. Assume that \( \sigma e \) is added into \( E \) to eliminate an inconsistent point \( (s_1, s_2, \sigma, e) \), \( T(s_1, \sigma e) \neq T(s_2, \sigma e) \) and for any other \( e \in E \), \( T(s_1, e) = T(s_2, e) \). Therefore, the added column is distinct from all other columns of \( T \). \( S \) is not extended during the loop, so the number of different columns is bounded by \( |\Sigma_{\text{out}}| \) and the number of extensions of \( E \) must also be finite.

Second, eliminating an inconsistent point may be done by changing hole values. This process might cause consistent points to become inconsistent because changing a table entry causes all its tied entries to be changed either. To prevent an infinite loop, any entry that is changed, is marked and can not be changed again. At each iteration at least one inconsistent point is eliminated and any other point can become inconsistent only once, so, we can conclude that the loop must terminates after finite number of iterations, and outputs a consistent table. □

Now we can prove the correctness of the algorithm.

Proof: (of theorem 4) If \( t \) is a supporting example the proof is trivial. If \( t \) is a counterexample, the algorithm extends \( (S, E, T) \) to cover \( t \). During table extension, any permanent entry is filled with the value of its supporting example, and any hole entry is filled with the current model. This filling strategy assures that the extended table covers \( D \cup \{t\} \), and the legality of the assignment to the holes. The two previous lemmas show that the consistency and the closeness loops at the end of the algorithm, terminate and output a closed and consistent table, so the proof is complete. □

3.3 Example Runs of the Algorithm

Assume that \( \Sigma_{\text{in}} = \{a, b\}, \Sigma_{\text{out}} = \{0, 1\} \), and the learned machine has only one state \( q_0 \). Figure 2 describes the learned machine, and figure 3 describes a learning session of the algorithm.

The initial model has one state and it predicts the opposite from the machine. Following the first example, \((\lambda, 0)\), the algorithm changes the table entry \((\lambda, \lambda)\) to zero and marks it as a permanent entry. Other entries of the table are also changed by the algorithm to preserve equal rows in order to prevent extension of the model. The accepted model, \( M_1 \), is equivalent to the learned machine and any following example must be a supporting example and does not change the model. This
example can point out the importance of the heuristics used by US-L*. If other entries of the table would have not been changed, the accepted model would have two states and any other example would have caused an addition of at least one more state to the model.

Figure 4 shows another DFA and Figure 5 describes an example of a learning session of the machine.

The first example \((\lambda, 1)\) is a supporting example and only changes the entry \((\lambda, \lambda)\) to become permanent. The second example, \((a, 0)\) is a counterexample, table entries are changed and the new model has two states. The third example, \((ab, 0)\), is also a counter example. The algorithm adds \(ab\) into \(S\) and \(aba\) and \(abb\) into \(S\Sigma\). An inconsistent point is added, \((a, ab, b, \lambda)\). In contradiction to the original \(L^*\), by changing the entry \(T(abb, \lambda)\) to be equal to \(T(ab, \lambda)\), the table becomes consistent without further extension. As a result, the third model is equivalent to the learned machine.

The examples, and the order they arrive, mostly effect the size of the learned model. The correctness proof also shows that the observation table can grow to become exponential in size of the sample. Figure 6 shows the model constructed from DFA1, when an arbitrary stream of examples is provided to the algorithm. After four examples, \((bbab, 0), (abab, 1), (bababa, 0), (aaabbb, 0)\), the model has eight states and is still not equivalent to the learned machine.
Figure 5: A learning session of DFA1. After three examples, the learned model is equivalent to the machine.

Figure 6: The accepted model for DFA1 from an arbitrary sample (bbab,0), (abbab,1), (bababa,0), (aaabbb,0)
We hypothesize that the algorithm is best suited for prefix-closed samples. We conducted an experiment where random machines of various sizes were created and were modeled by US-L^*, using various sizes of random prefix-closed samples of their behavior. A random machine with a given number of states was constructed by choosing a random transition function and by choosing a random output function. 100 experiments were conducted for each pair of sample size and machine size. It is important to clarify that the randomly built machines are not necessarily minimal.

Figure 7 shows the average size of the learned models as a function of the sample size, and as a function of the machine size. It is quiet clear that the average size of the learned models is similar to the size of the machines, and is not effected from the sample size. These results suggest that US-L^* has a strong ability to explore the common pattern of the sample.

![Figure 7: Average model size accepted from a random prefix-closed samples of various size, and as a function of the size of the learned machine](image)

4 Discussion

Unsupervised learning of finite automata has been proven to be NP-hard. However, there are many real situations when we must perform such learning. The work presented here studies the possibility of using heuristic methods for unsupervised learning of finite automata.

When a learning algorithm gets an example that is not consistent with its current model, it changes the model to agree with the new example. In the case of learning a DFA, this step leads to the creation of “holes” in the table representing the automata. Filling these holes in an optimal way is NP-hard. The $L^*$ algorithm overcomes the complexity problem by using a teacher for filling the holes. Rivest and Schapire’s algorithm overcomes the complexity problem by experimenting with the machine. The method presented in this paper is targeted at situations when a teacher is not available and experimentation is not possible.

At each learning step, our method fills the holes by consulting the model that was generated
in the previous step. The filled holes have tentative status and the algorithm readily changes their value if the changes may lead to a more compact model.

We conducted a set of experiments where random automata were generated and the algorithm tried to learn them based on prefix-closed samples of their behavior. The algorithm managed to learn very compact models that agree with the samples. The size of the sample had very little effect on the size of the model. Obviously our algorithm does not solve the complexity issue and it is always possible to build an adversary sample. However, the experimental results suggest that there might be classes of problems for which the algorithm behaves well.

The work presented here is only a first step. We are currently working on the following issues:

- Analyze the parameters that effect the algorithm’s complexity.
- Find conditions that identify classes under which the algorithm behaves well.
- Identify alternative heuristics for filling holes.
- Testing alternative search methods in the space of assignments.

References


CIS 9001
Guy Foux, Michael Heymann, and Alfred M. Bruckstein: “Two-Dimensional Polygon Navigation Among Unknown Stationary Polygonal Obstacles”.

CIS 9002

CIS 9003

CIS 9004

CIS 9005

CIS 9006

CIS 9007

CIS 9008

CIS 9009

CIS 9010

ANNUAL REPORT, 1990.

CIS 9011

CIS 9012

CIS 9101

CIS 9102

CIS 9103

CIS 9104


CIS 9201

CIS 9202

CIS 9202 (Revised, June 1992)

CIS 9202 (2nd Revision, April 1993)

CIS 9203

CIS 9204

CIS 9205

CIS 9206

CIS 9207

CIS 9208

CIS 9209

CIS 9210

CIS 9211

CIS 9212
CIS 9213

CIS 9214

CIS 9215

CIS 9216

CIS 9217

CIS 9218

CIS 9219

CIS 9220

CIS 9221

CIS 9222

CIS 9301

CIS 9302
<table>
<thead>
<tr>
<th>Report Number</th>
<th>Title</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIS 9303</td>
<td>G. Sapiro, A.M. Bruckstein: “A B-Spline Based Affine Invariant Multiscale Shape Representation”</td>
<td>January 1993</td>
</tr>
<tr>
<td>CIS 9304</td>
<td>G. Sapiro and A.M. Bruckstein: “The Ubiquitous Ellipse”</td>
<td>January 1993</td>
</tr>
<tr>
<td>CIS 9307</td>
<td>D. Geiger and D. Heckerman: “Knowledge Representation and Inference in Similarity Networks and Bayesian Multinets”</td>
<td>February 1993</td>
</tr>
<tr>
<td>CIS 9312</td>
<td>G. Elber: “Accessibility in 5-Axis Milling Environment”</td>
<td>March 1993</td>
</tr>
<tr>
<td>CIS 9313</td>
<td>Y. Baram: “Pattern Correction and Association and Sequence Regeneration by Correlative Memory”</td>
<td>March 1993</td>
</tr>
<tr>
<td>CIS 9314</td>
<td>R. Kimmel and A.M. Bruckstein: “Sub-Pixel Distance Maps and Weighted Distance Transforms”</td>
<td>April 1993</td>
</tr>
</tbody>
</table>


CIS 9328

CIS 9329

CIS 9330

CIS 9331

CIS 9332

CIS 9401

CIS 9402

CIS 9403

CIS 9404

CIS 9405

CIS 9406

CIS 9407

CIS 9408
CIS 9409

CIS 9410

CIS 9411

CIS 9412

CIS 9413

CIS 9414

CIS 9415

CIS 9416

CIS 9417

CIS 9418

CIS 9419

CIS 9420
Y. Baram and Z. Roth: “Multi-Dimensional Density Shaping by Sigmoidal Networks with Application to Classification, Estimation and Forecasting”, December 1994.

CIS 9501
CIS 9502

CIS 9503

CIS 9505

CIS 9506

CIS 9411

CIS 9507

CIS 9508

CIS 9509

CIS 9510

CIS 9511

CIS 9504