Learning Minimal DFA: Taking Inspiration from RPNI to Improve SAT Approach

Florent Avellaneda and Alexandre Petrenko

Computer Research Institute of Montreal
{florent.avellaneda, alexandre.petrenko}@crim.ca

Abstract. Inferring a minimal Deterministic Finite Automaton (DFA) from a learning sample that includes positive and negative examples is one of the fundamental problems in computer science. Although the problem is known to be NP-complete, it can be solved efficiently with a SAT solver especially when it is used incrementally. We propose an incremental SAT solving approach for DFA inference in which general heuristics of a solver for assigning free variables is replaced by that employed by the RPNI method for DFA inference. This heuristics reflects the knowledge of the problem that facilitates the choice of free variables. Since the performance of solvers significantly depends on the choices made in assigning free variables, the RPNI heuristics brings significant improvements, as our experiments with a modified solver indicate; they also demonstrate that the proposed approach is more effective than the previous SAT approaches and the RPNI method.

Keywords: Machine Inference · Machine Identification · Learning Automata · DFA · Grammatical Inference · SAT Solver.

1 Introduction

When we have an unknown system, re-engineering its model brings many advantages. A formal representation of the system allows us to understand how it works. The model can be used to check the properties of the system. Tests could be generated from the model using existing methods for model-based testing.

In this paper we are interested in the inference of a DFA model from observations. As is customary, we follow the principle of parsimony. This principle states that among competing hypotheses, the one with the fewest assumptions should be selected. Addressing the model inference problem, this principle suggests to infer the simplest model consistent with observations. Since the model to infer is an automaton, we generally use the number of states to measure the complexity.

There are two types of approaches for DFA inference, heuristic and exact approaches. Heuristic approaches merge states in an automaton representation of observations until a local minimum is reached. Exact approaches try to find a minimal automaton consistent with observations. The most known heuristic approach is probably the RPNI (Regular Positive and Negative Inference) method [12]. It performs a breadth first search by trying to merge a newly encountered

state with states already explored. Effective exact approaches generally formulate constraints and solve them using SAT solvers. Heule and Verwer have proposed an efficient SAT modeling [9]. We proposed an incremental SAT solving approach in the case of FSM inference [3]. The heuristic and exact approaches are generally quite distinct. In this paper we try to combine them together in order to achieve a better performance. The idea is as follows. We know that the efficiency of SAT solvers depends strongly on the order in which the Boolean variables are considered. To choose a "good" order among the Boolean variables SAT solvers use all kinds of generic heuristics which do not exploit the specifics of a particular problem, in our case it is DFA inference. In this paper, we use the RPNI heuristics to define the variable assignment order. Thus, the resulting approach can be viewed as an exact approach, though it uses RPNI to help finding a minimal automaton consistent with observations more quickly.

The paper is organized as follows. Section 2 contains definitions. Section 3 defines the inference problem and provides an overview of passive inference. Section 4 contains our contributions, namely, an incremental SAT solving approach for DFA, and modifications of a SAT solver incorporating the RPNI heuristics for determining the assignment order. Section 5 contains benchmarks. Finally Section 6 concludes.

2 Definitions

A *Deterministic Finite Automaton* (DFA) is a sextuplet $\mathcal{A} = (Q, \Sigma, \delta, q_\epsilon, F_A, F_R)$, where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, $q_\epsilon \in Q$ is the initial state, and $F_A \subseteq Q$ and $F_R \subseteq Q$ are disjoint sets of marked states, called the *accepting* and *rejecting* states, respectively [6]. We recursively extend the function δ to $Q \times \Sigma^* \to Q$ such that $\delta(q, \epsilon) = q$ and $\delta(q, a.w) = \delta(\delta(q, a), w)$. Also, for simplicity, we will write $(q, a, q') \in \delta$ if $\delta(q, a) = q'$.

A *learning sample* is a pair of finite disjoint sets of positive examples S^+ and negative examples S^-. We say that a DFA $\mathcal{A}$ is *consistent* with $S = (S^+, S^-)$ if $\forall w \in S^+ : \delta(q_\epsilon, w) \in F_A$ and $\forall w \in S^- : \delta(q_\epsilon, w) \in F_R$. If all DFAs with fewer states than $\mathcal{A}$ are not consistent with S, then we say that $\mathcal{A}$ is a *minimal* DFA consistent with S. We say that an example w is *inconsistent* with $\mathcal{A}$ if w is a positive example and $\delta(q_\epsilon, w) \notin F_A$ or w is a negative example and $\delta(q_\epsilon, w) \notin F_R$. We use $Pref(S)$ to denote the set of all prefixes of S^+ and S^-.

A *Prefix Tree Acceptor* (PTA) for a learning sample S, denoted $\mathcal{P}(S)$ is the tree-like DFA consistent with S such that all prefixes in $Pref(S)$ are the states of $\mathcal{P}(S)$ and only they. We denote by q_w the state reached by $\mathcal{P}(S)$ with the word w.

We say that two states $q, q' \in Q$ are *incompatible*, denoted $q \not\equiv q'$, if $q \in F_A \wedge q' \in F_R$ or $q \in F_R \wedge q' \in F_A$ or $\exists a \in \Sigma : \delta(q, a) \not\equiv \delta(q', a)$. Two states are *compatible* if they are not incompatible.

3 Inference Problem

Given a learning sample $S = (S^+, S^-)$ generated by an unknown DFA, we want to find a minimal DFA $\mathcal{A}$ consistent with S.

The existing approaches merge states in two different ways. The so-called RPNI approach [12] merges states incrementally. It is a heuristic approach, but we know that if the learning sample is large enough then it will find a minimal DFA consistent with S.

Another approach is based on a SAT solver and tries to determine a partition on the set of states of PTA $\mathcal{P}(S) = (Q, \Sigma, \delta, q_\epsilon, F_A, F_R)$ into compatible states such that the number of blocks does not exceed n. Clearly, n should be smaller than $|Q|$. If no partition can be found, it means that the bound n is too low. In this case we increase n and start again.

This approach has the advantage to guarantee that a minimal DFA consistent with S can be found independently of the size of the learning sample.

3.1 RPNI method

The algorithm RPNI is a popular method for inferring a DFA from a learning sample. A detailed explanation of the RPNI algorithm can be found in [6]. The idea consists in trying to merge states iteratively in a particular order. The algorithm attempts to merge first the states closest to the root state.

In particular, RPNI starts with the PTA determined from S. Then a breadth-first search is performed respecting the lexicographical order. Each time when a new state is found, the algorithm tries to merge it with already explored states (from the earliest to the most recently considered). The algorithm terminates when all states are considered and no more merge can be performed.

A remarkable property of this algorithm is that it identifies in the limit the generator of S. This means that with enough positive and negative examples, the DFA inferred by this algorithm will be the generator.

3.2 SAT Solving approach

The inference problem can be cast as a constraint satisfaction problem (CSP) [4]. For each state $q \in Q$ of the PTA we introduce an integer variable x_q such that

$$\begin{aligned} \forall q_i, q_j \in Q: \ & \text{if } q_i \in F_A \wedge q_j \in F_R \text{ then } x_{q_i} \neq x_{q_j} \\ & \text{if } \exists a \in \Sigma : (q_i, a, q_i'), (q_j, a, q_j') \in \delta \text{ then} \\ & (x_{q_i} = x_{q_j}) \Rightarrow (x_{q_i'} = x_{q_j'}) \end{aligned} \tag{1}$$

Let $B = \{0, ..., n-1\}$ be a set of integers representing blocks of a partition. The blocks are ordered following the order of natural numbers. Assuming that the value of x_q is in B for all $q \in Q$, we need to find a solution, i.e., an assignment of values of all variables such that (1) is satisfied. Each assignment implies a partition of n blocks and thus a DFA with at most n states consistent with S.

These CSP formulas can be translated to SAT using unary coding for each integer variable x_q where $q \in Q$: x_q is represented by n Boolean variables $v_{q,0}, v_{q,1}, ..., v_{q,n-1}$. Moreover, Heule and Verwer [9] propose to use auxiliary variables and redundant clauses in order to speed up the solving process. The SAT formulation they propose is as follows.

They define three kinds of variables:

- $v_{q,i}$, $q \in Q$ and $i \in B$. If $v_{q,i}$ is *true*, it means that state q is in block i.
- $y_{a,i,j}$, $i, j \in B$ and $a \in \Sigma$. If $y_{a,i,j}$ is *true*, it means that for any state in block i, the successor reached by symbol a is in block j.
- z_i, $i \in B$. If z_i is *true*, this means that block i becomes an accepting state.

For each state $q \in Q$, we have the clause:

$$v_{q,0} \vee v_{q,1} \vee ... \vee v_{q,n-1} \tag{2}$$

These clauses mean that each state should be in at least one block.

For each state q and every $i, j \in B$ such that $i \neq j$, we have the clauses:

$$\neg v_{q,i} \vee \neg v_{q,j} \tag{3}$$

These clauses mean that each state should be in at most one block.

The clauses 2 and 3 encode the constraint that each state should be in exactly one block.
For every states $q \in F_A, q' \in F_R$ and each $i \in B$, we have the clauses:

$$(\neg v_{q,i} \vee z_i) \wedge (\neg v_{q',i} \vee \neg z_i) \tag{4}$$

These clauses mean that an accepting state cannot be in the same block as a rejecting state.

For each transition $(q, a, q') \in \delta$ and for every $i, j \in B$:

$$y_{a,i,j} \vee \neg v_{q,i} \vee \neg v_{q',j} \tag{5}$$

This means that if state q is in the block i and its successor q' on symbol a is in the block j then blocks i and j are related for symbol a.
For each transition $(q, a, q') \in \sigma$ and for every $i, j \in B$:

$$\neg y_{a,i,j} \vee \neg v_{q,i} \vee v_{q',j} \tag{6}$$

This means that if blocks i and j are related for symbol a and a state q is in block i, then the successor of q with symbol a have to be in block j.
For each symbol $a \in \Sigma$, for every $i, j, h \in B$ such that $h < j$:

$$\neg y_{a,i,h} \vee \neg y_{a,i,j} \tag{7}$$

This means that each block relation can include at most one pair of blocks for each symbol to enforce determinism. Because of the commutative property of the operator $\vee$, we add the constraint $h < j$ to remove the equivalent clauses. For each symbol $a \in \Sigma$ and each $i \in B$:

$$y_{a,i,0} \vee y_{a,i,1} \vee ... \vee y_{a,i,n-1} \tag{8}$$

This means that each block relation must include at least one pair of blocks for each symbol.

We represent in Table 1 a summary of the formulas defined by Heule and Verwer.

Table 1. Summary for encoding (1) with clauses from PTA $\mathcal{P}(S) = (Q, \Sigma, \delta, q_\epsilon, F_A, F_R)$ into SAT. n is the maximal number of states in a DFA to infer, $B = \{0, ..., n-1\}$.

Ref	Clauses	Range
(2)	$v_{q,0} \vee v_{q,1} \vee ... \vee v_{q,n-1}$	$q \in \mathcal{P}(S)$
(3)	$\neg v_{q,i} \vee \neg v_{q,j}$	$q \in \mathcal{P}(S); 0 \leq i < j < n$
(4)	$\neg v_{q,i} \vee \neg v_{q',i}$	$q \in F_A, q' \in F_R; i \in B$
(5)	$y_{a,i,j} \vee \neg v_{q,i} \vee \neg v_{q',j}$	$(q, a, q') \in \delta; i, j \in B$
(6)	$\neg y_{a,i,j} \vee \neg v_{q,i} \vee v_{q',j}$	$(q, a, q') \in \delta; i, j \in B$
(7)	$\neg y_{a,i,h} \vee \neg y_{a,i,j}$	$a \in \Sigma; h, i, j \in B; h < j$
(8)	$y_{a,i,0} \vee y_{a,i,1} \vee ... \vee y_{a,i,n-1}$	$a \in \Sigma; i \in B$

It is possible that different assignments for a given SAT formula represents the same solution. In this case, we say that we have symmetry. A good practice is to break this symmetry [1, 2, 5] by adding constraints such that different assignments satisfying the formula represent different solutions. A formulation can result in a significant amount of symmetry if any permutation of the blocks is allowed. To eliminate this symmetry, Heule and Verwer use the state incompatibility graph which has $|Q|$ nodes and two nodes are connected iff the corresponding states of Q are incompatible. Clearly, each state of a clique (maximal or smaller) must be placed in a distinct block. Hence, they add to the SAT formula clauses for assigning initially each state from the clique to a separate block.

Experiments indicate that the proposed encoding of the constraints (1) is rather compact.

4 Incremental SAT Solving with Domain Specific Heuristics

4.1 Incremental SAT Solving

A disadvantage of the above SAT method is that the bigger a learning sample, the more complex the SAT formula. Thus, it can be expected that the solution time will increase significantly with the size of the learning sample. However,

in practice, this becomes detrimental, because we would like to use the largest possible learning sample to increase the chances of inferring a good model.

Addressing this problem, we proposed an iterative method for inferring FSMs [3]. Similar to this method, we propose to generate SAT constraints incrementally for DFAs as well. The idea is to iteratively infer a DFA from constraints generated for a subset (initially it is an empty set) of the learning sample. If the inferred DFA is inconsistent with the full learning sample, then we add more constraints considering an inconsistent example. This idea is in fact used by active inference methods, though active inference rely on a black box as an oracle capable of judging whether or not a word belongs to the model. In our method, the role of an oracle is assigned to a learning sample S. Even if this oracle is restricted since it cannot decide the acceptance for all possible examples, nevertheless, as we demonstrate, it leads to an efficient approach for passive inference from a learning sample.

Our incremental inference method works as follows. Let S be a learning sample (generated by a deterministic DFA). We want to find a minimal DFA consistent with S iteratively. To do that, we search for a DFA $\mathcal{A}$ with at most n states satisfying a growing set of constraints (initially we do not have any constraints). If no solution is found, it means that the bound n is too low. In this case we increase n and start again. If a solution is found and $\mathcal{A}$ is consistent with S, then we return this solution. Otherwise, we find the shortest example w in S inconsistent with $\mathcal{A}$. Then, we formulate a constraint that w has to be consistent with $\mathcal{A}$.

Note that Heule and Verwer's method of using a clique in the incompatibility graph is not applicable in an iterative approach context. Thus, we use an implicit and not explicit symmetry breaking method. In particular, we forbid block permutations by using a total order on the set of states. Let $<$ be a total order over the set of states $Q = \bigcup_{w \in Pref(S)} Q_w$ for all positive and negative examples. Based on a chosen order we add the following clauses excluding permutations. For each $q \in Q$ and each $i \in B$, we have a Boolean formula (which can be translated trivially into clauses):

$$(\bigwedge_{q'<q} \neg v_{q',i}) \Rightarrow \neg v_{q,i+1} \tag{9}$$

Intuitively, these clauses force to use blocks not already assigned when a state requires a new block.

The SAT formulation from Heule and Verwer is an efficient compact encoding, but determining that two states cannot be merged is a complex task. With our new heuristics, that we will present in Section 4.2, the solver will attempt to merge numerous not always compatible states. To reduce the number of such attempts we add more auxiliary (thus redundant) clauses that allow the solver to immediately detect that two states cannot be merged.

In particular, we add new auxiliary variables $E_{q,q'}$ for each pair of states $q, q' \in Q$.

First, we add clauses to encode the constraint that an accepting and a rejecting state cannot be merged. For every states q, q' such that $q \in F_A$ and $q' \in F_R$ we have a Boolean formula:

$$\neg E_{q,q'} \tag{10}$$

Notice that the clauses (4) express the same constraint, but in a less explicit way. In the same vein, we enforce the determinism of solutions by requiring that if two states merged together, their successors for any symbol also have to be merged together. We encode this property by the following formula (which can be translated trivially into clauses). For all $(q, a, p), (q', a, p') \in \delta$ we have a Boolean formula:

$$E_{q,q'} \Rightarrow E_{p,p'} \tag{11}$$

Finally, we encode the propagation of incompatibility to prohibit some mergers by the following formula (which can be translated trivially into clauses). For every states $q, q' \in Q$ and all $i \in \{0, ..., n-1\}$

$$(\neg E_{q,q'} \wedge v_{q,i}) \Rightarrow \neg v_{q',i} \tag{12}$$

It should be noted that we do not only propagate incompatibility here. The aim is to detect a conflict when a wrong merge is done without having to assign more free variables. Obviously the detection of such an error is not always possible without having to assign all free variables, but the above formulas increase the number of cases where this is possible.

Table 2. Summary for additional clauses from the PTA $\mathcal{P}(S) = (Q, \Sigma, \delta, q_\epsilon, F_A, F_R)$.

Ref	Clauses	Range
(9)	$(\bigwedge_{q'<q} \neg v_{q',i}) \Rightarrow \neg v_{q,i+1}$	$q \in Q, i \in \{0, ..., n-1\}$
(10)	$\neg E_{q,q'}$	$q \in F_A; q' \in F_R$
(11)	$\neg E_{q,q'} \vee E_{p,p'}$	$(q, a, p), (q', a, p') \in \delta$
(12)	$E_{q,q'} \vee \neg v_{q,i} \vee \neg v_{q',i}$	$q, q' \in Q; i \in \{0, ..., n-1\}$

The incremental SAT solving approach is formalized in Algorithm 1. The algorithm refers to Table 1 and 2 to encode the problem in SAT. Note that in practice we only add clauses not already added to exploit the ability of the SAT solver to operate incrementally.

Theorem 1. *Algorithm 1 returns a DFA consistent with S if it exists and $false$ otherwise.*

Proof. If the algorithm returns a DFA, it means that the condition in line 7 holds, $\mathcal{A}$ is consistent with S. If the algorithm returns $false$, it means that the formula C is unsatisfiable, and therefore there is no partition of size n for $\mathcal{P}(S')$; hence there is no solution for learning sample S.

Algorithm 1 Infer a DFA from a learning sample

Input: A learning sample S and an integer n
Output: A DFA with at most n states consistent with S if it exists

1: Let S' be an empty set
2: $C := v_{q_\epsilon,0}$
3: $C := C \wedge \bigwedge_{a\in\Sigma,\ 0\leq i<n} (y_{a,i,0} \vee ... y_{a,i,n-1})$ (See Formula 8)
4: $C := C \wedge \bigwedge_{a\in\Sigma,\ 0\leq i,j,h<n,\ h<j} (\neg y_{a,i,h} \vee \neg y_{a,i,j})$ (See Formula 7)
5: **while** C is satisfiable **do**
6: Let $\mathcal{A}$ be a DFA of a solution of C
7: **if** $\mathcal{A}$ is consistent with S **then**
8: **return** $\mathcal{A}$
9: **end if**
10: Let w be the shortest example in S inconsistent with $\mathcal{A}$
11: $S' := S' \cup \{w\}$
12: Let C be the clauses from the PTA $\mathcal{P}(S')$ using Table 1 and Table 2
13: **end while**
14: **return** $false$

The termination of the algorithm is guaranteed by the fact that in each execution of the loop, a new example of S is considered. Thus, when $S' = S$, we know that the condition in line 7 is true.

4.2 Domain Specific Heuristics

The performance of solvers depends strongly on the choices made when assigning free variables. A free variable is a variable not yet assigned to a value $true$ or $false$. Indeed, the resolution time can be significantly longer or shorter depending on these choices. In order to mitigate this problem, solvers use all kinds of heuristics [8, 10, 11]. These heuristics are generally intended to be comprehensive and try to reduce the resolution time whatever the formulas to solve are.

In this section, we propose to use, instead of the general heuristics, a heuristics specific to the DFA inference to decide which free variable should be assigned next. As the RPNI algorithm does exactly this and identifies in the limit the generator, we propose to use its heuristics to make the variable choices. This is motivated by the observation that RPNI makes state merge choices more and more relevant as the number of examples increases. Thus, we expect that the extra time required by a SAT solver to solve a problem when more examples are added will be compensated by the time saved by our heuristics and by making better choices of next free variables to assign. We know that eventually this will be the case, because all the merging choices made by RPNI are correct choices when the number of examples is large enough.

4.2.1 RPNI heuristics on decision variables

Most of the SAT solvers allow the user to distinguish two types of variables, decision and auxiliary variables. The decision variables are the variables for which we want to know a valid assignment, i.e., the assignment that satisfies the formula. Auxiliary variables are additional variables that can be used to factorize the encoding of a SAT formula or just help a solver find a solution faster. We do not seek generally to find an assignment for these auxiliary variables, since it can be deduced from a valid assignment of the decision variables. In our SAT formula, only variables $v_{q,i}$ will be decision variables. The other variables will be considered by the solver as auxiliary variables. Thus, the SAT solver will terminate when it finds an assignment for all variables $v_{q,i}$.

The RPNI heuristics will be used to decide which variable $v_{q,i}$ should be chosen among the free variables. To do that, each word $w.i$ such that $\delta(q_\epsilon, w) = q$ is assigned to the variable $v_{q,i}$. When the solver must decide which free variable to pick, one of variables $v_{q,i}$ will be chosen according to the lexicographical order on the words associated with variables. Then it will try to assign this variable to *true*.

In fact, this heuristic suggests selecting a state not already assigned to a block and trying to assign it to a block in the ascending order. The order in which the states are selected respects the RPNI strategy, i. e., selecting the state closest in the lexicographical order to the root.

4.2.2 Implementation

Adding the proposed heuristics to a solver, we have slightly modified the solver MiniSAT [7]. In MiniSAT, the variable *order_heap* of type $VarOrderLt$ associates a weight of type $Integer$ to each free variable. The heuristics used by the solver consists in modifying these weights during the resolution of the formula according to various criteria. Thus, when a free variable assignment must be done, the solver uses *order_heap* to select the free variable according to its weight.

Our modification consists in disabling the default solver heuristics and changing the $VarOrderLt$ structure of each variable to a word. Thus, each Boolean variable $v_{q_w,i}$ is associated with the weight $w.i$. As a result, when a free variable has to be assigned the solver returns a variable associated with the shortest word in the lexicographic order.

5 Experimental Evaluation

We have performed a set of benchmarks to evaluate our approach. All DFAs are generated randomly with $|\Sigma| = 4$ and n states. For each state s and each $a \in \Sigma$, we randomly choose a state s' such that $\delta(s, a) = s'$. If the DFA we obtain is not minimal, we start again until we obtain a minimal one. Since generating a random DFA is rather simple, it does not take much time, even if many attempts are required to find a minimal DFA. To generate examples from this DFA, we perform random walks of a random length between 0 and 50.

We compare five algorithms.

- *RPNI*: We use the implementation provided by Stamina competition [13].
- *H&V*: It is the SAT approach elaborated by Heule and Verwer. The method is summarized in Table 1.
- *Incremental SAT*: It is a SAT approach implemented in an incremental way recently proposed by us [3]. This approach corresponds to Algorithm 1, neither using Table 2 for clause generation nor changing the SAT solver.
- *Incremental SAT*2: It is *Incremental SAT* in which we add the additional clauses from Table 2.
- *New algo*: It is our approach described in Section 4.

The SAT solver used for this experimentation is MiniSat [7] and we use a VirtualBox with 12 GB of RAM and Intel® Core™ i7-2600K processor.

5.1 Inference varying the number of examples

In this section, we compare the five algorithms on DFAs with five states. We limit the number of states to 5 so that each algorithm is able to solve the problem. DFAs with more states will be considered in the next section. In this experiment, see Figure 1, we vary the number of examples and determine time it takes to infer a DFA.

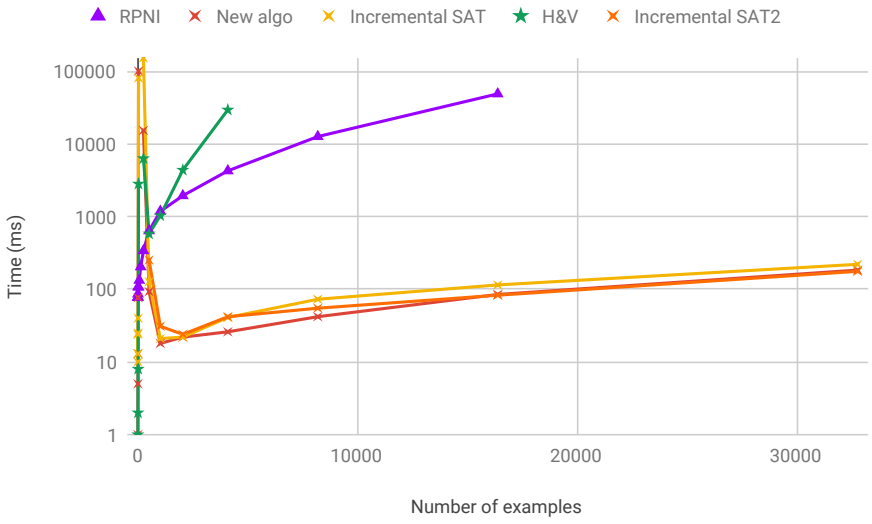

Fig. 1. Average time over 100 instances to infer a DFA with five states vs the number of examples.

We notice that the performances of the *New algo*, *Incremental SAT* and *Incremental SAT*2 algorithms are almost the same as well as that they behave best when the learning samples are large enough. The results show that our approach is faster than *H&V* and *RPNI* except in the case where the number of examples is very small.

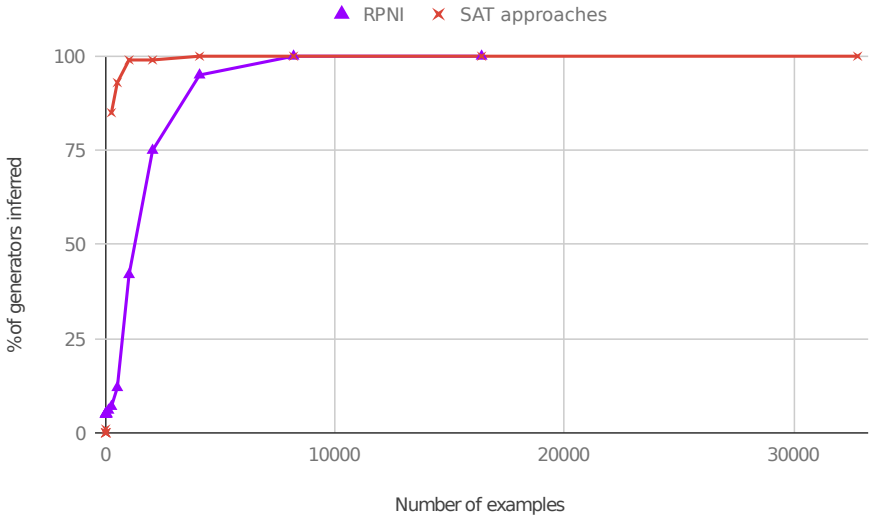

Fig. 2. Percentage of generators inferred correctly vs the number of examples.

To find the reason for that we determine the percentage of generators correctly inferred for SAT and RPNI approaches for learning samples of various sizes, see Figure 2. We have grouped all the SAT approaches into a single curve because the quality of the obtained solutions is almost the same. This is not surprising because all SAT approaches are focused on finding an optimal solution, i.e., a minimum DFA consistent with observations. The data indicate that longer solution time is the price to pay for a higher percentage of good models inferred by the SAT approaches. After all, RPNI is heuristic, while SAT approaches are deductive. The obtained results indicate that SAT approaches have an important advantage over RPNI. In particular, SAT approaches need about thousand examples to correctly infer almost all generators, while the RPNI approach needs more than ten thousand. In addition, when the RPNI approach does not correctly infer the DFA, the result is generally quite far from the generator and contains hundreds of states.

Thus, the fact that SAT approaches are rather slow when the size of learning samples is only a few dozen is not really important, because with so few examples we are hardly able to infer generators correctly.

5.2 Inference from learning samples of growing generators

In this section, we focus on experimental comparison of the *Incremental SAT*, *Incremental SAT*2 and *New algo* approaches. In the previous section, we saw that they perform similarly when the number of states in generators is fixed to five. Here we will push the algorithms to their limits. Thus we set the number of examples in learning samples at 100,000 and increase the number of generator's states. Comparison with *RPNI* and *H&V* is not possible here because these algorithms are unable to proceed with such a large number of examples.

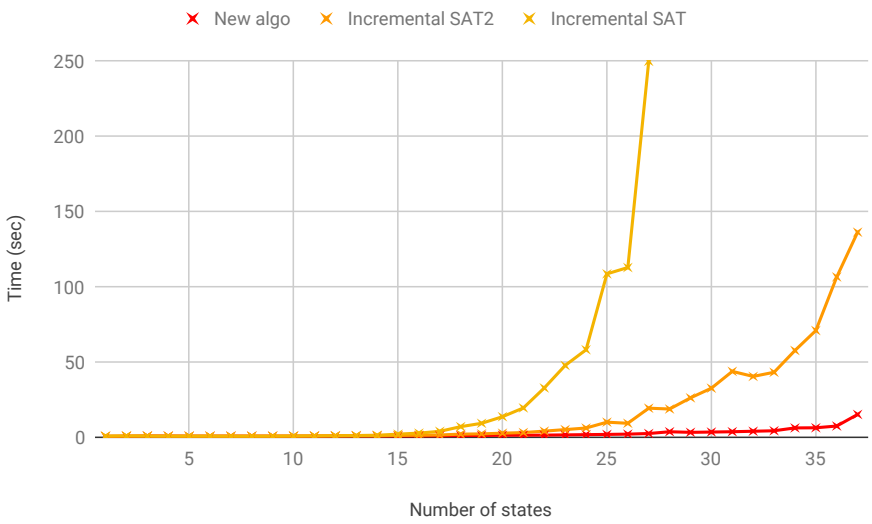

Fig. 3. Average time over 100 instances to infer a DFA vs the number of generator's states

Figure 3 indicates that the three algorithms equally perform when the generators have less than 15 states. However, for generators with more states, our new method has a clear advantage. As an examples, DFAs of 27 states are inferred on average in 2.5 seconds with our new method while it takes more than 4 minutes with *Incremental SAT*, and DFAs of 37 states are inferred on average in 15 seconds with our new method while it takes more than 2 minutes without the RPNI heuristics on decision variables.

6 Conclusion

In this paper we considered the problem of inferring a minimal DFA from a learning sample that includes positive and negative examples. Among the existing approaches, the heuristic approaches, like RPNI, merge states reaching a local minimum and the exact approaches solve constraints finding a minimal automaton consistent with observations.

In order to improve the scalability of exact inferring approaches, we made the following contributions.

First, we proposed to construct the SAT formula incrementally during the DFA inference, similar to our method for the FSM inference, thus avoiding to deal with a whole (large) learning sample.

Second, we found a way of combining the two approaches such that the result surpasses each of them. In particular, to improve the performance of a SAT solver we proposed to use the RPNI heuristics determining the order in which free variables are assigned.

Finally, we also suggested new auxiliary variables and additional clauses to be used in the traditional SAT encoding which accelerate the process of determining the state incompatibility.

The experimental evaluation of the proposed approach indicates that when a learning sample is large enough, it gives better results than the classical SAT solving approaches and the RPNI algorithm. The experimental results show that the proposed approach is somewhat slower that the latter, but only when the learning sample is too small to correctly infer the generator from it. These experiments seem to confirm that the scalability of the SAT solving approach for DFA inference improves when the SAT formula is built incrementally and a solver is enriched with a problem specific heuristics.

Acknowledgments This work was partially supported by MEI (Ministère de l'Économie et Innovation) of Gouvernement du Québec and NSERC of Canada.

References

1. Fadi A Aloul, Arathi Ramani, Igor L Markov, and Karem A Sakallah. Solving difficult SAT instances in the presence of symmetry. In *Proceedings of the 39th annual Design Automation Conference*, pages 731–736. ACM, 2002.
2. Fadi A Aloul, Karem A Sakallah, and Igor L Markov. Efficient symmetry breaking for boolean satisfiability. *IEEE Transactions on Computers*, 55(5):549–558, 2006.
3. Florent Avellaneda and Alexandre Petrenko. Fsm inference from long traces. In *International Symposium on Formal Methods*, pages 93–109. Springer, 2018.
4. Alan W Biermann and Jerome A Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE transactions on Computers*, 100(6):592–597, 1972.
5. Cynthia A Brown, Larry Finkelstein, and Paul Walton Purdom Jr. Backtrack searching in the presence of symmetry. In *International Conference on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 99–110. Springer, 1988.

6. Colin De la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.
7. Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *International conference on theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.
8. Jon William Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, Citeseer, 1995.
9. Marijn JH Heule and Sicco Verwer. Software model synthesis using satisfiability solvers. *Empirical Software Engineering*, 18(4):825–856, 2013.
10. Joao Marques-Silva. The impact of branching heuristics in propositional satisfiability algorithms. In *Portuguese Conference on Artificial Intelligence*, pages 62–74. Springer, 1999.
11. Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.
12. José Oncina and Pedro Garcia. Inferring regular languages in polynomial updated time. In *Pattern recognition and image analysis: selected papers from the IVth Spanish Symposium*, pages 49–61. World Scientific, 1992.
13. Neil Walkinshaw, Bernard Lambeau, Christophe Damas, Kirill Bogdanov, and Pierre Dupont. Stamina: a competition to encourage the development and assessment of software model inference techniques. *Empirical software engineering*, 18(4):791–824, 2013.