Using a satisfiability solver to identify deterministic finite state automata

Marijn J.H. Heule Sicco Verwer

Delft University of Technology, Algorithmics group
marijn@heule.nl, s.e.verwer@tudelft.nl

Abstract

We present an exact algorithm for identification of deterministic finite automata (DFA) which is based on satisfiability (SAT) solvers. Despite the size of the low level SAT representation, our approach seems to be competitive with alternative techniques. Our contributions are threefold: First, we propose a compact translation of DFA identification into SAT. Second, we reduce the SAT search space by adding lower bound information using a fast max-clique approximation algorithm. Third, we include many redundant clauses to provide the SAT solver with some additional knowledge about the problem. Experiments on a well-known suite of random DFA identification problems show that SAT solvers can efficiently tackle all instances. Moreover, our exact algorithm outperforms state-of-the-art techniques on several hard problems.

1 Introduction

The problem of identifying (learning) a deterministic finite state automaton (DFA) is one of the best studied problems in grammatical inference, see, e.g., [6]. A DFA is a well-known language model that can be used to recognize a regular language. The goal of DFA identification is to find a (non-unique) smallest DFA that is consistent with a set of given labeled examples. The size of a DFA is measured by the amount of states it contains. An identified DFA has to be as small as possible because of an important principle known as Occam's razor, which states that among all possible explanations for a phenomenon, the simplest is to be preferred. A smaller DFA is simpler, and therefore a better explanation for the observed examples. DFA identification thus consists of finding the regular language that is most likely to have generated a set of labeled examples. This problem has many applications in for example computational linguistics, bioinformatics, and speech processing.

The problem of finding a smallest consistent DFA can be very difficult. It is the optimization variant of the problem of finding a consistent DFA of fixed size, which has been shown to be NP-complete [8]. In spite of this hardness result, quite a few DFA identification algorithms exist, see, e.g., [6]. The current state-of-the-art in DFA identification is the evidence driven state-merging (EDSM) algorithm [11]. Essentially, EDSM is a heuristic method that tries to find a good local optimum efficiently. It has been shown using a version of EDSM called RPNI, that it is guaranteed to efficiently converge to the global optimum in the limit [15]. Since DFA identification is still an NP-complete problem, however, wrapping a specialized search procedure around the EDSM heuristic method will almost always lead to better results, see, e.g., [14, 1, 10, 4].

Although the different search techniques improve the performance of EDSM, they are still less advanced and optimized than the search techniques in solvers for well-studied problems such as graph coloring and satisfiability (SAT). Especially SAT solvers have become very powerful in the last decade. The power of these SAT solvers can be used in other problems by translating these problems into a SAT instances, and subsequently running a SAT solver on these translated problems. This approach is very competitive for several problems, see, e.g., [3, 12, 7]. We adopt this approach for DFA identification.

In [5], such a translation is introduced from DFA identification into graph coloring. We propose a different method inspired by this encoding that translates DFA identification into SAT. Our translation is efficient and very easy to implement. The crucial part of our translation is that we use auxiliary variables. Without these additional variables, a *direct encoding* [19] of the graph coloring constraints leads to

$\mathcal{O}(k|V|^2 + k^2|V|^2)$ clauses, where k is the size of the identified DFA, and V is the set of labeled examples. Our encoding however requires only $\mathcal{O}(k|V| + k^2|V|)$ clauses. In addition, we make use of *symmetry breaking predicates* [16] in order to prevent overlapping searches with different colors. We produce these predicates by preprocessing the result of our translation with a fast max-clique approximation algorithm. Furthermore, we add many redundant clauses to our translation that provide the SAT solver with some additional knowledge about the DFA identification instance.

We compare the performance of our SAT approach with the EDSM algorithm with a good search technique on a set of well-known benchmark problem instances. The results of these experiments show that our approach is competitive with the state-of-the-art in DFA identification.

This paper is organized as follows. We start with a short description of the EDSM algorithm (Section 2) and the translation into graph coloring (Section 3). We then give our translation into SAT, including symmetry breaking and redundant clauses (Section 4). We present our experimental results (Section 5), and end with some conclusions and some ideas for future work (Section 6).

2 The state-of-the-art in DFA identification

We assume the reader to be familiar with the theory of languages and automata, for an introduction see, e.g., [18]. A *deterministic finite state automaton* (DFA) $\mathcal{A}$ is a automaton model consisting of states and labeled transitions. It recognizes those symbol sequences formed by the labels of transitions on paths from a specific start start to a final state. In this way, DFAs can be used to recognize any regular language. We use $L(\mathcal{A})$ to denote the language of a DFA $\mathcal{A}$. Given a pair of finite sets of positive sample strings S_+ and negative sample strings S_-, called the *input sample*, the goal of DFA identification is to find a *smallest* DFA $\mathcal{A}$ that is *consistent* with $S = \{S_+, S_-\}$, i.e., such that $S_+ \subseteq L(\mathcal{A})$ and $S_- \subseteq L(\mathcal{A})^{\mathrm{C}}$. The size of a DFA is measured by the usual measure, i.e., by the number of states it contains.

The idea of a state-merging algorithm is to first construct a tree-shaped DFA from this input, and then to *merge* the states of this DFA. Such a tree-shaped DFA is called an *augmented prefix tree acceptor* (APTA), see Figure 1. An APTA is a DFA that is consistent with the input sample S. Moreover, it is such that there exists only one path from the start state to any other state. This implies that the computations of two strings s and s' reach the same state q if and only if s and s' share the same prefix until they reach q, hence the name prefix tree. An APTA is called *augmented* because it contains (is augmented with) states for which it is yet unknown whether they are accepting or rejecting. No execution of any sample string from S ends in such a state. We use V, V_+, and V_- to denote all states, the accepting states, and the rejecting states in the APTA, respectively.

A *merge* of two states q and q' combines the states into one: it creates a new state q'' that has the same incoming and outgoing transitions of both q and q'. Such a merge is only allowed if the states are *consistent*, i.e., it is not the case that q is accepting while q' is rejecting, or vice versa. Whenever a merge introduces a non-deterministic choice, i.e., q'' is the source of two transitions with the same symbol, the target states of these transitions are merged as well. This is called the *determinization* process, and is continued until there are no non-deterministic choices left. Of course, all of these merges should be consistent too. The result of a merge is a new DFA that is smaller than before, and still consistent with the input sample S. A state-merging algorithm iteratively applies the state-merging process until no more consistent merges are possible.

Currently, the most successful method for solving the DFA identification problem is the evidence driven state-merging (EDSM) algorithm [11]. In EDSM each possible merge is given a score based on the amount of *evidence* in the merges that are performed by the merge and the determinization processes. A possible merge gets an evidence score equal to the number of accepting states that are merged with accepting states plus the number of rejecting states that are merged with rejecting states. At each iteration of the EDSM algorithm, the merge with the highest evidence value is performed. EDSM is a polynomial time (greedy) algorithm that converges quickly to a local optimum.

In grammatical inference, there is a lot of research into developing advanced and efficient search techniques for ESDM. The idea is to increase the quality of a solution by searching other paths in addition to the path determined by the greedy EDSM heuristic. Examples of such advanced techniques are dependency directed backtracking [14], using mutually (in)compatible merges [1], and searching most-constrained nodes first [10]. A comparison of different search techniques for EDSM can be found in [4]. The best-performing search procedure in this comparison is a simple beam-search called ed-beam. This procedure calculates one greedy EDSM path starting from every node in the search tree in breadth-first order. The smallest DFA found by these EDSM paths is returned as a solution. Typically, a time bound is set and the algorithm is

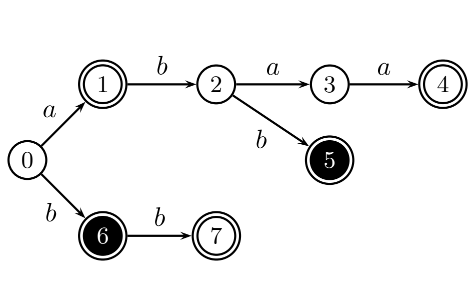

Figure 1: An augmented prefix tree acceptor for $S = (S_+ = \{a, abaa, bb\}, S_- = \{abb, b\})$ (left) and the corresponding consistency graph (right). Some vertices in the consistency graph are not directly inconsistent, but inconsistent due to determinization. For instance state 2 and 6 are inconsistent because the strings abb and bb will end in the same state if these states are merged. Also state 1 and 2 are inconsistent because the strings a and abb will end in the same state if these states are merged.

stopped when its running-time exceeds this bound. However, it can guarantee that it has found the optimal solution (a smallest DFA) if all smaller solutions have been visited by its breadth-first search.

3 From DFA identification to graph coloring

The EDSM search techniques are usually based on successful techniques for other more actively studied problems, such as satisfiability and graph coloring. There have been many competitions for algorithms that solve these problems and these solvers are therefore highly optimized. Although the different search techniques improve the performance of EDSM, and the implementations use efficient data structures, still a lot of work has to be done before the EDSM implementations are as efficient and advanced as the solvers for these problems. Since the decision version of DFA identification is NP-complete, it is also possible to translate the DFA identification problem into a more actively studied problem, and thus make use of the optimized search techniques immediately.

In [5], such a translation is introduced from DFA identification into graph coloring. The main idea of this translation is to use a distinct color for every state of the identified DFA. Every vertex in the graph of the graph coloring problem represents a distinct state in the APTA. Two vertices v and w in this graph are connected by an edge (cannot be assigned the same color), if merging v and w results in an inconsistency (i.e., an accepting state is merged with a rejecting state). These edges are called *inequality constraints*. Figure 1 shows an example of such a graph.

In addition to these inequality constraints, *equality constraints* are required: if two vertices v and w with the same incoming label are merged, then the parents $p(v)$ and $p(w)$ of v and w must be merged too. With the addition of these constraints, some of the inequality constraints become *redundant*: only the directly inconsistent edges (between accepting and rejecting states) are actually necessary, the other edges (resulting from the determinization process) are no longer needed because they logically follow from combining the direct constraints and the equality constraints. These redundant constraints are kept in the translation in order to help the search process.

In the graph coloring problem, the equality constraints imply that the two parent nodes $p(v)$ and $p(w)$ should get the same color if and only if v and w get the same color. Such a constraint is difficult to implement in graph coloring. In [5], this is dealt with by modifying the graph according to the consequences of these constraints. This implies that a new graph coloring instance has to be solved every time an equality constraint is used. We propose a different method to encode these inequality constraints, that is by encoding them using satisfiability. In addition, using auxiliary variables, we reduce the number of constraints that are required by the encoding.

4 Translating DFA identification into SAT

The *satisfiability* problem (SAT) deals with the question whether there exists an assignment to Boolean variables such that a given formula evaluates to true. Such a formula in conjunctive normal form (CNF) is a conjunction ($\wedge$) of clauses, each clause being a disjunction ($\vee$) of literals. Literals refer either to a Boolean variables x_i or to its negation $\neg x_i$.

In the last decade, SAT solvers have become very powerful. This can be exploited by translating a problem into CNF and solve it by a SAT solver. Despite the low level representation, such an approach is very competitive for several problems. Examples are bounded model checking [3], equivalence checking [12] and rewriting termination problems [7]. Below we present such an approach to DFA identification.

4.1 Direct encoding

Our translation reduces DFA identification into a graph coloring problem [5] which in turn is translated into SAT. A widely used translation of graph coloring problems into SAT is known as the *direct encoding* [19]. Given a graph $G = (V, E)$ and a set of colors C, the direct encoding uses (Boolean) *color variables* $x_{v,i}$ with $v \in V$ and $i \in C$. If $x_{v,i}$ is assigned to true, it means that vertex v has color i. The constraints on these variables are as follows (see Table 1 for details): For each vertex, *at-least-one* color clauses make sure that each vertex is colored, while *at-most-one* color clauses forbid that a vertex can have multiple colors. The latter clauses are redundant.

Additionally, we have to translate that adjacent vertices cannot have the same color. The direct encoding uses the following clauses:

$$\bigwedge_{i \in C} \bigwedge_{(v,w) \in E} (\neg x_{v,i} \vee \neg x_{w,i})$$

Finally, let EL be the set consisting of pairs of vertices that have the same incoming label in the APTA. In case a pair $(v, w) \in EL$ has the same color, then the parents of v and w (denoted by $p(v)$ and $p(w)$) must have the same color too. This corresponds to the equality constraints in [5]. A straight-forward translation of these constraints into CNF is:

$$\bigwedge_{i \in C} \bigwedge_{j \in C} \bigwedge_{(v,w) \in EL} \begin{array}{l} (\neg x_{v,i} \vee \neg x_{w,i} \vee \neg x_{p(v),j} \vee x_{p(w),j}) \wedge (\neg x_{v,i} \vee x_{w,i} \vee \neg x_{p(v),j} \vee \neg x_{p(w),j}) \wedge \\ (\neg x_{v,i} \vee \neg x_{w,i} \vee x_{p(v),j} \vee \neg x_{p(w),j}) \wedge (x_{v,i} \vee \neg x_{w,i} \vee \neg x_{p(v),j} \vee \neg x_{p(w),j}) \end{array}$$

Notice that the size of the direct encoding is $\mathcal{O}(|C|^2|V|^2)$. For interesting DFA identification problems this will result in a formula which will be too large for the current state-of-the-art SAT solvers. Therefore we will propose a more compact encoding below.

4.2 Compact encoding

The majority of clauses in the direct encoding originate from translating the equality constraints into SAT. We propose a more efficient encoding based on auxiliary variables $y_{a,i,j}$, which we refer to as *parent relation variables*. If set to true, $y_{a,i,j}$ means that for any vertex with color i, the child reached by label a must have color j. Let $l(v)$ denote the incoming label of vertex v, and let $c(v)$ denote the color of vertex v. As soon as both a child v_i and its parent $p(v_i)$ are colored, we force the corresponding parent relation variable to true by the clause $y_{l(v_i),c(p(v_i)),c(v_i)} \vee \neg x_{p(v_i),c(p(v_i))} \vee \neg x_{v_i,c(v_i)}$. This leads to $\mathcal{O}(|C|^2|V|)$ clauses. Additionally, we require *at most one* parent relation clauses to guarantee that each relation is unique – see Table 1 for details.

This new encoding reduces the number of clauses significantly. To further reduce this size, we introduce an additional set of auxiliary variables z_i with $i \in C$. If z_i is true, color i is only used for accepting vertices. Therefore, we refer to them as *accepting color variables*. They are used for the constraint that requires all accepting vertices to be colored differently from the rejecting states. Without auxiliary variables, this can be encoded as $(\neg x_{v,i} \vee \neg x_{w,i})$ for $v \in V_+$, $w \in V_-$, $i \in C$, resulting in $|V_+| \cdot |V_-| \cdot |C|$ clauses. Using the auxiliary variables z_i, the same constraints can be encoded as $(\neg x_{v,i} \vee z_i) \wedge (\neg x_{w,i} \vee \neg z_i)$, requiring only $(|V_+| + |V_-|)|C|$ clauses.

4.3 Symmetry breaking

In case a graph cannot be colored with k colors, the corresponding (unsatisfiable) SAT instance will solve the problem $k!$ times: once for each permutation of the colors. Therefore, when dealing with CNF formulas representing graph coloring problems, it is good practice to add *symmetry breaking predicates* (SBPs) [16].

Table 1: Encoding of DFA identification into SAT.

Variables	Range	Meaning
$x_{v,i}$	$v \in V; i \in C$	$x_{v,i} \equiv 1$ iff vertex v has color i
$y_{a,i,j}$	$a \in L; i, j \in C$	$y_{a,i,j} \equiv 1$ iff parents of vertices with color j and incoming label a must have color i
z_i	$i \in C$	$z_i \equiv 1$ iff an accepting state has color i
Clauses	**Range**	**Meaning**
$(x_{v,1} \vee x_{v,2} \vee \cdots \vee x_{v,\|C\|})$	$v \in V$	each vertex has at least one color
$(\neg x_{v,i} \vee z_i) \wedge (\neg x_{w,i} \vee \neg z_i)$	$v \in V_+; w \in V_-; i \in C$	accepting vertices cannot have the same color as rejecting vertices
$(y_{l(v),i,j} \vee \neg x_{p(v),i} \vee \neg x_{v,j})$	$v \in V; i, j \in C$	a parent relation is set when a vertex and its parent are colored
$(\neg y_{a,i,h} \vee \neg y_{a,i,j})$	$a \in L; h, i, j \in C; h < j$	each parent relation can target at most one color
Redundant Clauses	**Range**	**Meaning**
$(\neg x_{v,i} \vee \neg x_{v,j})$	$v \in V; i, j \in C; i < j$	each vertex has at most one color
$(y_{a,i,1} \vee y_{a,i,2} \vee \cdots \vee y_{a,i,\|C\|})$	$a \in L; i \in C$	each parent relation must target at least one color
$(\neg x_{v,i} \vee \neg x_{w,i})$	$i \in C; (v, w) \in E$	all determinization conflicts explicitly added as clauses

Notice that in any valid coloring of a graph, all vertices in a clique must have a different color. So, one can fix vertices in a large clique to a color in a pre-processing step.

Although finding the largest clique in a graph is NP-complete, a large clique can be computed cheaply using a greedy algorithm. Start with the vertex v_0 with the highest degree. In each step i, add the vertex v_i that is connected to all vertices v_0 to v_{i-1}, again with the highest degree.

Because the corresponding graph of an APTA can be huge (many edges), we propose a variant of this algorithm. First, compute the induced subgraph of accepting vertices ($v \in V_+$) and determine a large clique in this subgraph. Second, in a similar way find a large clique among rejecting vertices ($v \in V_-$). Because all accepting vertices are connected to all rejecting vertices, the union of both cliques is also a clique. On average, this variant appears to provide a clique that is larger than the clique found using the entire APTA. In addition, the computation costs are very low.

4.4 Adding redundant clauses

The compact encoding discussed above can be extended with several types of redundant clauses. First, we can explicitly state that each vertex must be colored with exactly one color by adding the redundant *at most one color clauses* $(\neg x_{v,i} \vee \neg x_{v,j})$ with $v \in V$ and $i < j \in C$. Similarly, we can explicitly state that for each combination of a color and a label exactly one parent relation variable must be true. This is achieved by adding the *at least one parent relation clauses* $(\bigwedge_{j \in C} y_{a,i,j})$ for all $a \in L$ and $i \in C$.

Both types of clauses are known as *blocked clauses* [9]. These clauses have either zero or two clashing literals with the other clauses. Therefore, blocked clauses cannot be used to derive the empty clause (i.e. show that the formula is unsatisfiable). So, formulas with and without blocked clauses are equisatisfiable.

A third type of redundant clauses consists of adding all edges that are not covered by the *accepting color clauses*. Although these edges are redundant, they provide some additional knowledge about the problem to the SAT solver. However, the addition of these clauses could potentially blow up the size of the encoding.

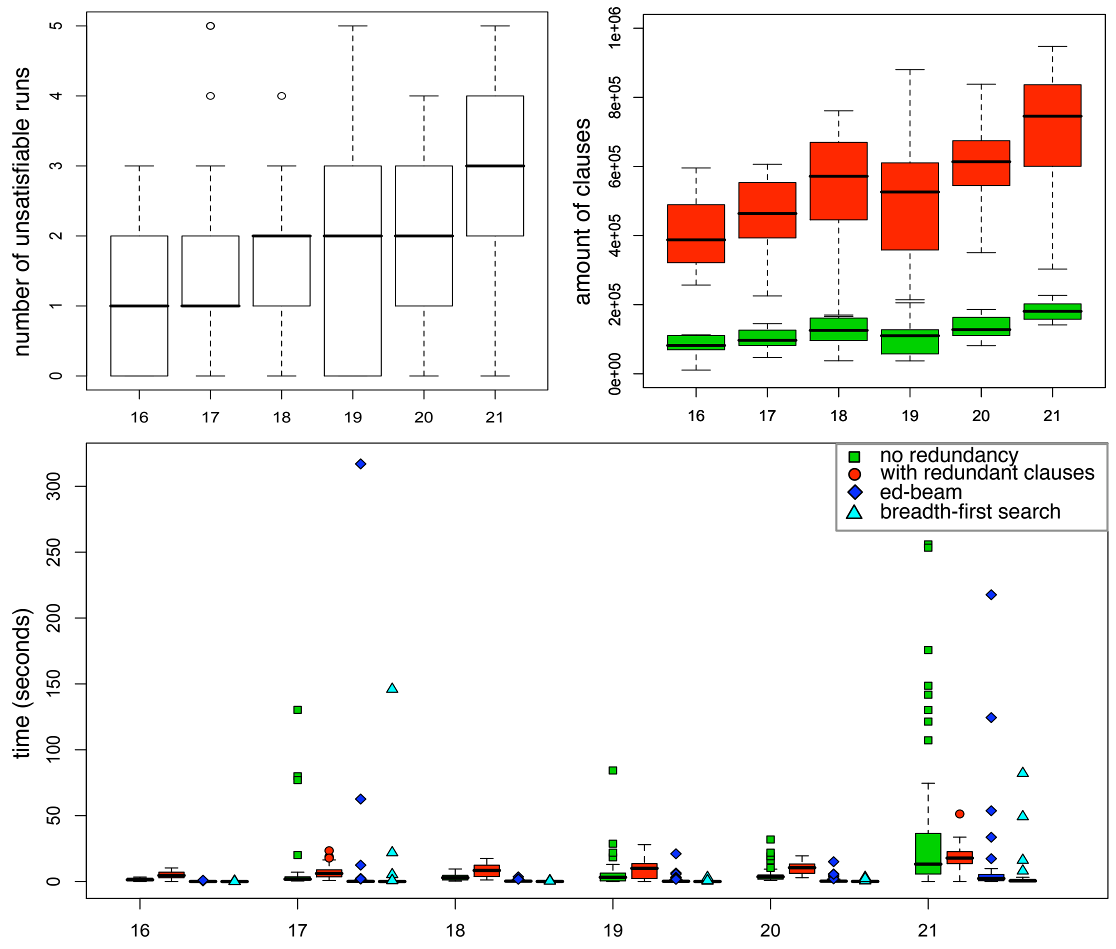

Figure 2: Results on the random set using boxplots. Top left shows the number of unsatisfiable runs (or the minimal DFA size minus the lower bound). Top right shows the number of clauses of the encoding with and without redundant clauses. The bottom image shows the runtime comparison of SAT and EDSM techniques.

4.5 Iterative SAT solving

The translation of DFA identification into SAT (direct encoding, compact encoding with and without redundant clauses) uses a fixed set of colors. To prove that the minimal size of a DFA equals k, we have to show that the translation with k colors is satisfiable and that the translation with $k - 1$ colors is unsatisfiable. The following procedure is used to determine the minimal size:

step 1: Compute a large clique L (a set of vertices) in the graph representing the APTA.
step 2: Initialize the set of colors C in such a way that $|C| = |L|$.
step 3: Construct a CNF based on the APTA with color set C and SBPs based on L.
step 4: Solve the formula of step 3.
step 5: If the formula is unsatisfiable then add a color to C and goto step 3.
step 6: Return the DFA found in step 4.

5 Results

Our experiments are based on a suite of 810 moore instances[1] that were also used to evaluate exact DFA identification algorithms in [13, 10]. The suite is partitioned into sizes ranging from 4 to 21. Since the larger ones are more difficult, we focus on the sizes 16 to 21. All tests were performed on a Intel Pentium 4, 3.0 GHz with 1 Gb of memory running on Fedora Core 8.

[1]available at `http://algos.inesc.pt/~aml/tar_files/moore_dfas.tar.gz`

We compare two implementations of our SAT encoding with the current state-of-the-art in DFA identification. One encoding uses the compact encoding, the other one uses all redundant clauses. These algorithms follow the iterative SAT solving procedure presented in Section 4.5. We used picosat [2] to solve the CNF instances. The performance is measured by cumulating all computational costs of the unsatisfiable runs together with the time to solve the smallest satisfiable instance. The top left of Figure 2 shows the number of unsatisfiable runs, while the top right shows the number of clauses of both encodings. Notice that the redundant clauses increase the size of the encoding by a factor four. The bottom plot shows the running-time of all four algorithms: first the two SAT approaches, then the ed-beam search variant of EDSM (see Section 2 for a description), and finally a breadth-first search. The breadth-first search computes only a little more than those parts of the search tree that ed-beam requires to guarantee finding an optimum.

All four algorithms solved the full suite within a timeout of about 5 minutes per instance. On average the ed-beam and breath-first search implementations are faster. However, the SAT translation with all redundant clauses performed best on the hardest problems. These results are promising, since they show that the search techniques used by SAT can be a lot more efficient than the state-of-the-art search variants of EDSM.

After analyzing the runtimes of solving the formulas with redundancies, we observed that on the unsatisfiable runs about 90% of the computation was consumed in the pre-processing phase (mostly on parsing the CNF). Therefore, the time spend on actual search is very low. This indicates that a further improvement of the performance can be obtained by creating a tighter coupling between the translation program and the SAT solver.

6 Conclusions and Future Work

We presented an efficient translation from DFA identification into satisfiability. By performing this transformation, we are able to make direct use of the advanced search techniques that are currently used in satisfiability solving. In addition, the transformation is very easy to implement. Thus, we have presented a simple, efficient, and advanced algorithm for solving the DFA identification problem. In experimental results, we show that our approach is competitive with the current state-of-the-art in DFA identification.

The use of auxiliary variables by this transformation results in a significant improvement in the number of required clauses with respect to a straightforward transformation that directly makes use of the transformation to graph coloring in [5]. The transformation only requires $\mathcal{O}(k|V| + k^2|V|)$ clauses for a DFA identification problem, where k is the size of the sought DFA and V are the states of the APTA constructed from the input sample. Using the straightforward transformation, we would have required $\mathcal{O}(k|V|^2 + k^2|V|^2)$ clauses. Since $|V|$ is typically large, this is a big improvement.

We make use of symmetry breaking predicates in order to prevent overlapping searches with different colors. These predicates are produced by preprocessing the result of the transformation with a fast max-clique approximation algorithm. In the future, we would like to perform this symmetry breaking also dynamically, i.e., during the satisfiability solving. This is a new technique for graph coloring based satisfiability solving proposed in [17] that shows promising improvements in performance.

Another approach we want to try in the near future is to combine the greedy EDSM algorithm with our satisfiability approach. The main idea is to use EDSM to compute a partial solution, and then use satisfiability in order to compute the optimum in the remaining search tree. Applying EDSM first will merge states in the APTA, and hence decrease the number of clauses of the satisfiability problem. In this way, we hope to solve large problem instances such as the ones from the Abbadingo competition [11], which currently result in too large satisfiability instances.

We would also like to test our approach on probabilistic DFA identification. In this problem, there are no negative examples. In this case, consistency constraints between vertices can be determined using statistical tests. Using our transformation, these consistency constraints can again be transformed to satisfiability. In this way, we hope to compute the optimal probabilistic DFA or hidden Markov model for event prediction. This problem has much more applications than DFA identification because negative data is often hard to obtain in practice.

Also, since many problems in machine learning are NP-complete, we hope that our encoding inspires more researchers to translate these problems into SAT in order to find optimal solutions to machine learning problems.

Acknowledgements The first author is supported by the Dutch Organization for Scientific Research (NWO) under grant 617.023.611.

References

[1] John Abela, François Coste, and Sandro Spina. Mutually compatible and incompatible merges for the search of the smallest consistent DFA. In *ICGI*, volume 3264 of *LNCS*, pages 28–39. Springer, 2004.

[2] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.

[3] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *TACAS '99*, pages 193–207, London, UK, 1999. Springer-Verlag.

[4] Miguel Bugalho and Arlindo L. Oliveira. Inference of regular languages using state merging algorithms with search. *Pattern Recognition*, 38:1457–1467, 2005.

[5] François Coste and Jacques Nicolas. Regular inference as a graph coloring problem. In *Workshop on Grammatical Inference, Automata Induction, and Language Acquisition (ICML'97)*, 1997.

[6] Colin de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, 2005.

[7] Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix interpretations for proving termination of term rewriting. *J. Autom. Reason.*, 40(2-3):195–220, 2008.

[8] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

[9] Oliver Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, 96-97(1):149–176, 1999.

[10] Kevin J. Lang. Faster algorithms for finding minimal consistent DFAs. Technical report, NEC Research Institute, 1999.

[11] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *ICGI*, volume 1433 of *LNCS*. Springer, 1998.

[12] Joao P. Marques-Silva and Thomas Glass. Combinational equivalence checking using satisfiability and recursive learning. In *Proceedings of DATE '99*, page 33, New York, NY, USA, 1999. ACM.

[13] Arlindo L. Oliveira and Joao P. Marques-Silva. Efficient search techniques for the inference of minimum size finite automata. In *In Proceedings of the 1998 South American Symposium on String Processing and Information Retrieval, Santa Cruz de La Sierra*, pages 81–89. IEEE Computer Society Press, 1998.

[14] Arlindo L. Oliveira and Joao P. Marques-Silva. Efficient search techniques for the inference of minimum sized finite state machines. *String Processing and Information Retrieval*, pages 81–89, 1998.

[15] J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, volume 1 of *Series in Machine Perception and Artificial Intelligence*, pages 49–61. World Scientific, 1992.

[16] Karem A. Sakallah. *Symmetry and Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications, Handbook of Satisfiability*, chapter 10, pages 289–338. IOS Press, February 2009.

[17] Bas Schaafsma, Marijn J.H. Heule, and Hans van Maaren. Dynamic symmetry breaking by simulating Zykov contraction. In *SAT 2009*, volume 5584 of *LNCS*, pages 223–236. Springer, 2009.

[18] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997.

[19] Toby Walsh. SAT v CSP. In *CP '02: Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming*, pages 441–456, London, UK, 2000. Springer-Verlag.