Exact DFA Identification Using SAT Solvers*

Marijn J.H. Heule[1,**] and Sicco Verwer[2]

[1] Delft University of Technology
marijn@heule.nl
[2] Eindhoven University of Technology
s.verwer@tue.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 is competitive with alternative techniques. Our contributions are fourfold: 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. Fourth, we show how to use the flexibility of our translation in order to apply it to very hard problems. Experiments on a well-known suite of random DFA identification problems show that SAT solvers can efficiently tackle all instances. Moreover, our 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., [1]. 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, speech processing, and verification.

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,

* This is an extended version of: Marijn Heule and Sicco Verwer. Using a Satisfiability Solver to Identify Deterministic Finite State Automata. In BNAIC 2009, pp. 91-98.

** Supported by the Dutch Organization for Scientific Research (NWO) under grant 617.023.611.

J.M. Sempere and P. García (Eds.): ICGI 2010, LNAI 6339, pp. 66–79, 2010.

which has been shown to be NP-complete [2]. In spite of this hardness results, quite a few DFA identification algorithms exist, see, e.g., [1]. The current state-of-the-art in DFA identification is the evidence driven state-merging (EDSM) algorithm [3]. 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 [4]. However, wrapping a specialized search procedure around the EDSM heuristic method will typically lead to better results, see, e.g., [5,6,7,8].

Although the different search techniques improve the performance of EDSM, they are still less advanced than 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 solvers can be used in other problems by translating these problems into SAT instances, and subsequently running a SAT solver on these translated problems. This approach is very competitive for several problems, see, e.g., [9,10,11]. We adopt this approach for DFA identification.

In [12], 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. The nodes in the graph coloring instance represent the labeled examples. Two nodes are connected if the examples they represent have different labels, i.e., if they cannot end in the same state in the DFA. Dynamic constraints are used to guarantee that examples with different labels cannot end in the same state. The amount of colors used in the graph coloring problem is equal to the size of the identified DFA, and hence this should be as small as possible. Finding this minimum can be done by iterating over this amount.

An alternative approach [13] uses the well-known translation of DFA identification to an integer constraint satisfaction problem (CSP) from [14]. It translates this CSP into SAT in two ways: a unary and a binary encoding of the integers. Again, the minimum can be found by iterating over the number of states.

We propose a different method inspired by the encoding by [12]. The main problem we solve is how to efficiently encode the graph coloring constraints of [12] into SAT. A naive *direct encoding* [15] of these constraints would lead to $\mathcal{O}(k^2|V|^2)$ clauses, where k is the size of the identified DFA, and V is the set of labeled examples. Such a direct encoding is in fact identical to the unary encoding from [13], which can be considered the current state-of-the-art in translations of DFA identification to SAT. Our encoding, however, requires only $\mathcal{O}(k^2|V|)$ clauses. The crucial part of our translation is the use *auxiliary variables* to represent the problem more efficiently. In addition, we apply *symmetry breaking* [16] to prevent overlapping searches with different colors 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.

A nice feature of our encoding is that it is flexible in the sense that it can also be applied to partially identified DFAs. Starting with a partially identified DFA reduces the size of the SAT instance significantly. Thus, one could use our encoding as a subprocess in a larger DFA identification algorithm as follows:

first identify a small part of a DFA, and then run our encoding to determine how many additional states are required. In this way, our encoding can also be applied in cases where the number of clauses resulting from our initial encoding is too large for the current state-of-the-art SAT solvers.

The contributions of this paper are thus fourfold:

- We introduce a simple and efficient encoding of DFA identification to SAT.
- We suggest max-clique symmetry breaking to reduce the search space.
- We add redundant clauses to improve the performance of the SAT solver.
- We show how the flexibility of our encoding can be used in order to apply it to very hard problems.

We compare the performance of our SAT approach with the naive direct encoding, and two state-of-the-art search procedures for EDSM. We first tested these algorithms on a set of well-known benchmark problem instances. These results show that our approach is competitive with the state-of-the-art in DFA identification, and significantly outperforms the current state-of-the-art in translations of DFA identification to SAT. In addition, we tested our approach on a very challenging suite of very hard instances. For the second experiment we applied our encoding to a DFA that was partially identified by EDSM. This second experiment shows that the flexibility of our encoding allows it to be applied to very difficult DFA identification problems. In a few of these instances we could determine the exact solution starting from a short initial run of EDSM. In addition, during these experiments we discovered that our max-clique symmetry breaking technique can potentially be used to reduce the search space of the EDSM algorithm. Adapting EDSM to make use of this technique is left as future work.

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). Next, we explain the application of our encoding to partially identified DFA (Section 5). We present our experimental results (Section 6), and end with some conclusions and some ideas for future work (Section 7).

2 The State-of-the-Art in DFA Identification

We assume the reader to be familiar with the theory of languages and automata. 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 $S_- \subseteq \Sigma^* \setminus L(\mathcal{A})$ (where Σ^* is the set of all strings). The size of a DFA is measured by the usual measure, i.e., by the number of states it contains.

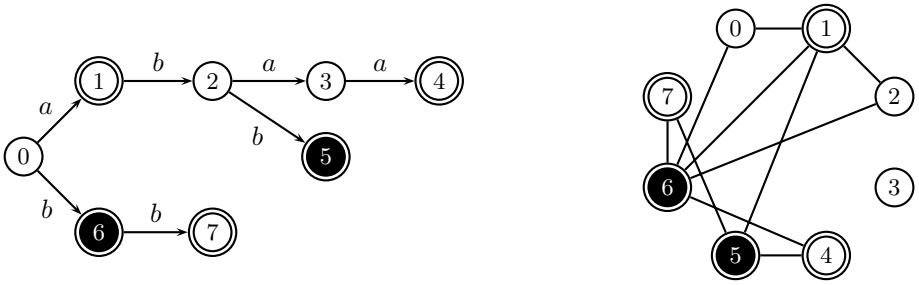

Fig. 1. An augmented prefix tree acceptor for $S = (S_+ = \{a, abaa, bb\}, S_- = \{abb, b\})$ (left) and the corresponding consistency graph (right). Some vertices in the consistency graph are not directly inconsistent, but inconsistent due to determinization. For instance state 2 and 6 are inconsistent 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.

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 such 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. 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 in the red-blue framework [3]. EDSM is a greedy procedure that uses a simple heuristic to determine which merge to perform. 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 [5], using mutually (in)compatible

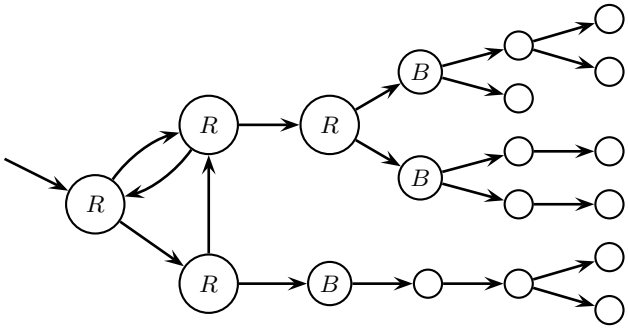

Fig. 2. The red-blue framework. The red states (labeled R) are the identified parts of the automaton. The blue states (labeled B) are the current candidates for merging. The uncolored states are pieces of the APTA.

merges [6], and searching most-constrained nodes first [7]. A comparison of different search techniques for EDSM can be found in [8].

Typically, a time bound is set and the algorithm is 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. In total EDSM tries $|V|^2$ possible merges in every iteration, and since V can be very large, this can take a very large amount of time. In order to avoid this, EDSM is often applied within the *red-blue framework*. The red-blue framework maintains a core of red states with a fringe of blue states, see Figure 2. A red-blue state-merging algorithm performs merges only between blue and red states. If no red-blue merge is possible the algorithm changes the color of a blue state into red. This framework reduces the amount of possible merges significantly without reducing the number of possible solutions, i.e., the algorithm is still complete. Since the algorithm is guaranteed not to change any of the transitions between red states, the red core of the DFA can be viewed as a part of the DFA that is already identified. Within the red-blue framework, EDSM is a polynomial time (greedy) algorithm that converges quickly to a local optimum. Despite its simplicity, EDSM participated in and won (in a tie) the Abbadingo DFA learning competition in 1997 [3]. The evidence measure that is used by EDSM is based on the idea that bad merges can often be avoided by performing those merges that have passed the most tests for consistency, and are hence most likely to be correct. Using this evidence measure EDSM participated in and won (in a tie) the Abbadingo DFA learning competition in 1997 [3]. The data set in this competition consisted of sparse data-sets. In the competition EDSM was capable of approximately (with 99% accuracy) learning a DFA with 500 states with a training set consisting of 60.000 strings.

The current state-of-the-art techniques are two simple search strategies called ed-beam and exbar [7]. The ed-beam 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. This solution then serves as an upper bound of the DFA size for the breadth-first search. The exbar procedure iteratively runs EDSM with an increasing upper bound on the number of DFA states. It continues this procedure until a solution is found. To reduce the size of the search space, exbar searches the most-constrained nodes first.

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 [12], 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 the parents $p(v)$ and $p(w)$ of two vertices v and w with the same incoming label are merged, then 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 our 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 $p(w)$ can get the same color only if v and w get the same color. Such a constraint is difficult to implement in graph coloring. In [12], 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 [9], equivalence checking [10] and rewriting termination problems [11]. Below we present such an approach to DFA identification.

4.1 Direct Encoding

Our translation reduces DFA identification into a graph coloring problem [12] which in turn is translated into SAT. A widely used translation of graph coloring problems into SAT is known as the *direct encoding* [15]. 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 the parents $p(v)$ and $p(w)$ of such a pair $(v, w) \in EL$ have the same color, then v and w must have the same color as well. This corresponds to the equality constraints in [12]. A straight-forward translation of these constraints into CNF is:

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

This encoding is identical to the CSP-based translation given in [13], and can be considered as the current state-of-the-art in translations of DFA identification to SAT. 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 has 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]. 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 preprocessing 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. This variant often provides 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

Table 1. Encoding of DFA identification into SAT. C = set of colors, L = set of labels (alphabet), V = vertices, E = conflict edges.

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**
$(x_{v,1} \vee x_{v,2} \vee \cdots \vee x_{v,\lvert C\rvert})$	$v \in V$	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
$(y_{a,i,1} \vee y_{a,i,2} \vee \cdots \vee y_{a,i,\lvert C\rvert})$	$a \in L; i \in C$	each parent relation must target at least one color
$(\neg y_{l(v),i,j} \vee \neg x_{p(v),i} \vee x_{v,j})$	$v \in V; i,j \in C$	a parent relation forces a vertex once the parent is colored
$(\neg x_{v,i} \vee \neg x_{w,i})$	$i \in C; (v,w) \in E$	all determinization conflicts explicitly added as clauses

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 $i \in C$. Also, once a parent relation is set, and some vertices have the source color, then all child nodes should have the target color $(\neg y_{l(v),i,j} \vee \neg x_{p(v),i} \vee x_{v,j})$ for $v \in V; i,j \in C$.

All three types of clauses are known as *blocked clauses* [17]. These clauses have at least one literal that cannot be removed by resolution. 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. We will show that these blocked clauses improve the performance of DFA identification. However, for other problems, removal of blocked clauses results in a speed-up [18].

Other types of redundant clauses consists of a second constraint on the parent relation and adding all edges that are not covered by the *accepting color clauses*. Although these clauses are redundant, they provide some additional knowledge about the problem to the SAT solver. However, since the largest number of clauses are created by the first parent relation, and since there can be an even larger number of conflicts, the addition of these clauses could potentially blow up the size of the encoding.

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:
S_1: find a large clique L (set of vertices) in the graph representing the APTA.
S_2: initialize the set of colors C in such a way that $|C| = |L|$.
S_3: construct a CNF by translating the APTA based on C and SBPs on L.
S_4: solve the formula of step S_3.
S_5: if the formula is unsatisfiable then add a color to C and goto step S_3.
S_6: return the DFA found in step S_4.

5 Translating Partially Identified DFAs

In spite of the efficiency of our translation, there can still be cases where the above procedure leads to a formula that is too large for the current state-of-the-art SAT solvers. For instance, the Abbadingo problem set [3] contains some very difficult problems that require hundreds of colors, resulting in over 100.000.000 clauses. Since the current state-of-the-art SAT solvers are known to work well up to 5.000.000 clauses, this is much too large.

In such cases another nice feature of our encoding can be used, which is that it also works when the input is a (partially identified) DFA instead of an APTA. Thus, a simple method that can be used to reduce the size of the problem is to:

1. apply a few steps of the EDSM algorithm, and then
2. apply our translation to SAT.

Every merge that is performed before applying the translation reduces the size of V significantly. Therefore also the encoding becomes much smaller. The price to pay is of course that the solution provided by the SAT solver will no longer be exact. The first few merges are performed by a greedy procedure, and hence they can lead to a larger DFA size. These first few merges will however be based on a lot of evidence. Consequently, they are likely to be correct, i.e., they are likely to lead to the optimal solution. Intuitively, this approach should therefore work well in practice, the main problem is to know how many merges to perform. For more information on the EDSM evidence value and its rationale, the reader is referred to [3].

An additional benefit of first applying the EDSM algorithm is that we automatically obtain a clique of conflicting states: no red state can be merged with another red states. Hence, the red states in a partially identified DFA resulting from a few steps of the EDSM algorithm can be used to construct symmetry breaking predicates. These predicates can be used instead of the ones resulting from the greedy max-clique algorithm.

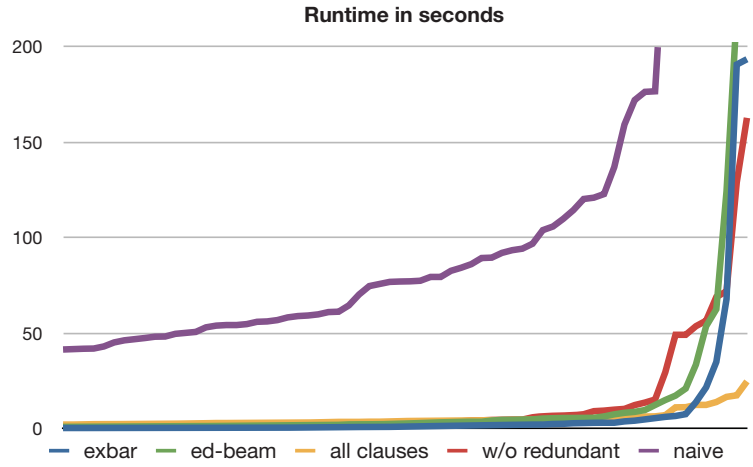

Fig. 3. Results on the set from [19,7]. The graph shows run-times in seconds of exbar, ed-beam, our encoding with and without redundant clauses, and the naive direct encoding. The horizontal axis shows the instances of DFA size 16 or more sorted by run-time.

6 Results

Our experiments are based on a suite of 810 instances[1] that were also used to evaluate exact DFA identification algorithms in [19,7]. 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. In addition, we performed tests on some instances from the challenging Abbadingo problem set [3]. All tests were performed on a Intel Pentium 4, 3.0 GHz with 1 Gb of memory running on Fedora Core 8.

We compare two implementations of our SAT encoding with the current state-of-the-art in exact DFA identification: ed-beam and exbar (see Section 2 for a description). In addition, we include the naive encoding described in Section 4. This encoding can be considered as the current state-of-the art translation to SAT. All SAT algorithms follow the iterative SAT solving procedure presented in Section 4.5. We used picosat [20] 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.

Figure 3 shows the run-times of all algorithms. All algorithms except the naive encoding solved the full suite within 200 seconds per instance. Most problems require almost no search at all and are solved by all algorithms except the naive encoding in a few seconds. Some of the larger problems, however, do require some search-time and there one clearly sees the strength of our approach: it outperforms the state-of-the-art on these instances. An interesting observation is the effect of the redundant clauses, without these the SAT solver no longer

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

outperforms the state-of-the-art. The huge difference between the naive and our encoding clearly shows the benefit of using the auxiliary variables we introduced in our encoding.

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.

We also experimented with instances of the Abbadingo challenge [3].[2] Initially, these benchmarks appeared too hard for our exact SAT approach. The smallest problem has an APTA with 12,796 states, resulting in 77,730,715 clauses. Therefore, we first ran a few iterations of EDSM and applied our translation as soon as the size of the partial DFA is small enough. Throughout our experiment, we observed that a partial DFA of about 5000 states is currently the limit of what state-of-the-art SAT solvers can manage. This means that for the smallest problems (#1, #4, and #7) about a dozen merge steps are required. Using this combined approach we were able to solve the first four challenge problems.

7 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. The result is simple, efficient, and advanced algorithm for solving the DFA identification problem. In experimental results, we show that our approach is very competitive with the current state-of-the-art in DFA identification. It even outperformed the state-of-the-art on several hard instances. In addition, we show that the flexibility of our transformation can be used to apply it to very challenging DFA identification instances.

The use of auxiliary variables by this transformation results in a significant improvement in the number of required clauses with respect to the current state-of-the-art in translating DFA identification to SAT [12] and [13]. Our transformation only requires $\mathcal{O}(k^2|V|)$ 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 current state-of-the-art, we would have required $\mathcal{O}(k^2|V|^2)$ clauses. Since $|V|$ is typically large, this is a big improvement.

We plan to experiment with alternative translations. Most of the graph color to SAT translations presented in [21] can be used for DFA identification too. In order to determine the usefulness of these alternatives, we will construct variants that use auxiliary variables to reduce the size of theses translations as well.

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,

[2] Available at `http://www-bcl.cs.may.ie/data-sets.html`

i.e., during the satisfiability solving. This is a new technique for graph coloring based satisfiability solving proposed in [22] that shows promising improvements.

In our experiments, the greedy max-clique algorithm often discovered a larger clique than the one induced by the red states in a partially identified DFA. This opens up a very interesting path for future work in DFA identification, namely to replace the red states in EDSM by the states represented by this clique. Since this clique poses more constraints on the partially identified DFA, we believe this will improve the performance of the EDSM algorithm.

References

1. de la Higuera, C.: A bibliographical study of grammatical inference. Pattern Recognition 38(9), 1332–1348 (2005)
2. Gold, E.M.: Complexity of automaton identification from given data. Information and Control 37(3), 302–320 (1978)
3. Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, p. 1. Springer, Heidelberg (1998)
4. Oncina, J., Garcia, P.: Inferring regular languages in polynomial update time. In: Pattern Recognition and Image Analysis. Series in Machine Perception and Artificial Intelligence, vol. 1, pp. 49–61. World Scientific, Singapore (1992)
5. Oliveira, A.L., Marques-Silva, J.P.: Efficient search techniques for the inference of minimum sized finite state machines. In: SPIRE, pp. 81–89 (1998)
6. Abela, J., Coste, F., Spina, S.: Mutually compatible and incompatible merges for the search of the smallest consistent DFA. In: Paliouras, G., Sakakibara, Y. (eds.) ICGI 2004. LNCS (LNAI), vol. 3264, pp. 28–39. Springer, Heidelberg (2004)
7. Lang, K.J.: Faster algorithms for finding minimal consistent DFAs. Technical report, NEC Research Institute (1999)
8. Bugalho, M., Oliveira, A.L.: Inference of regular languages using state merging algorithms with search. Pattern Recognition 38, 1457–1467 (2005)
9. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
10. Marques-Silva, J.P., Glass, T.: Combinational equivalence checking using satisfiability and recursive learning. In: DATE 1999, p. 33. ACM, New York (1999)
11. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of term rewriting. J. Autom. Reason. 40(2-3), 195–220 (2008)
12. Coste, F., Nicolas, J.: Regular inference as a graph coloring problem. In: Workshop on Grammatical Inference, Automata Induction, and Language Acquisition, ICML 1997 (1997)
13. Grinchtein, O., Leucker, M., Piterman, N.: Inferring network invariants automatically. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 483–497. Springer, Heidelberg (2006)
14. Biermann, A.W., Feldman, J.A.: On the synthesis of finite-state machines from samples of their behavior. IEEE Trans. Comput. 21(6), 592–597 (1972)
15. Walsh, T.: SAT v CSP. In: Dechter, R. (ed.) CP 2000. LNCS, vol. 1894, pp. 441–456. Springer, Heidelberg (2000)

16. Sakallah, K.A.: Symmetry and Satisfiability. In: Handbook of Satisfiability, ch. 10, pp. 289–338. IOS Press, Amsterdam (2009)
17. Kullmann, O.: On a generalization of extended resolution. Discrete Applied Mathematics 96-97(1), 149–176 (1999)
18. Jarvisalo, M., Biere, A., Heule, M.J.H.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 129–144. Springer, Heidelberg (2010)
19. Oliveira, A.L., Marques-Silva, J.P.: Efficient search techniques for the inference of minimum size finite automata. In: South American Symposium on String Processing and Information Retrieval, pp. 81–89. IEEE Computer Society Press, Los Alamitos (1998)
20. Biere, A.: Picosat essentials. Journal on Satisfiability, Boolean Modeling and Computation 4, 75–97 (2008)
21. Velev, M.N.: Exploiting hierarchy and structure to efficiently solve graph coloring as sat. In: ICCAD 2007: International conference on Computer-aided design, Piscataway, NJ, USA, pp. 135–142. IEEE Press, Los Alamitos (2007)
22. Schaafsma, B., Heule, M.J.H., van Maaren, H.: Dynamic symmetry breaking by simulating Zykov contraction. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 223–236. Springer, Heidelberg (2009)