BFS-Based Symmetry Breaking Predicates for DFA Identification

Vladimir Ulyantsev(✉), Ilya Zakirzyanov, and Anatoly Shalyto

ITMO University, Saint-Petersburg, Russia
{ulyantsev,zakirzyanov}@rain.ifmo.ru, shalyto@mail.ifmo.ru

Abstract. It was shown before that the NP-hard problem of deterministic finite automata (DFA) identification can be translated to Boolean satisfiability (SAT). Modern SAT-solvers can efficiently tackle hard DFA identification instances. We present a technique to reduce SAT search space by enforcing an enumeration of DFA states in breadth-first search (BFS) order. We propose symmetry breaking predicates, which can be added to Boolean formulae representing various DFA identification problems. We show how to apply this technique to DFA identification from both noiseless and noisy data. The main advantage of the proposed approach is that it allows to exactly determine the existence or non-existence of a solution of the noisy DFA identification problem.

Keywords: Grammatical inference · Boolean satisfiability · Learning automata · Symmetry breaking techniques

1 Introduction

Deterministic finite automata (DFA) are models that recognize regular languages [1], therefore the problem of DFA identification (induction, learning) is one of the best studied [2] in grammatical inference. The identification problem consists of finding a DFA with minimal number of states that is consistent with a given set of strings with language attribution labels. This means that such a DFA rejects the negative example strings and accepts the positive example strings. It was shown in [3] that finding a DFA with a given upper bound on its size (number of states) is an NP-complete problem. Besides, in [4] it was shown that this problem cannot be approximated within any polynomial.

Despite this theoretical difficulty, several efficient DFA identification algorithms exist [2]. The most common approach is the evidence driven state-merging (EDSM) algorithm [5]. The key idea of this algorithm is to first construct an augmented prefix tree acceptor (APTA), a tree-shaped automaton, from the given labeled strings, and then to iteratively apply a state-merging procedure until no valid merges are left. Thus EDSM is a polynomial-time greedy method that tries to find a good local optimum. EDSM participated in the Abbadingo DFA learning competition [5] and won it (in a tie). To improve the EDSM algorithm several specialized search procedures were proposed, see, e.g., [6,7]. One of the

A.-H. Dediu et al. (Eds.): LATA 2015, LNCS 8977, pp. 611–622, 2015.
DOI: 10.1007/978-3-319-15579-1_48

most successful approaches is the EDSM algorithm in the red-blue framework [5], also called the Blue-fringe algorithm.

The second approach for DFA learning is based on evolutionary computation; early work includes [8,9]. Later the authors of [10] presented an effective scheme for evolving DFA with a multi-start random hill climber, which was used to optimize the transition matrix of the identified DFA. A so-called smart state labeling scheme was applied to choose the state labels optimally, given the transition matrix and the training set. Authors emphasized that smart selection of state labels gives the evolutionary method a significant boost which allowed it to compete with the EDSM. Authors find that the proposed evolutionary algorithm (EA) outperforms the EDSM algorithm on small target DFAs when the training set is sparse. For larger DFAs with 32 states, the hill climber fails and EDSM then clearly outperforms it.

The challenge of the GECCO 2004 Noisy DFA competition [11] was to learn the target DFA when 10 percent of the given training string labels had been randomly flipped. In [12] Lucas and Reynolds show that within limited time EA with smart state labeling is able to identify the target DFA even at such high noise level. Authors compared their algorithm with the results of the GECCO competition and found that EA clearly outperformed all the entries. Thereby it is the state-of-the-art technique for learning DFA from noisy training data.

In several cases the best solution for noiseless DFA identification is the *translation-to-SAT* technique [13], which was altered to suit the *StaMInA* (State Machine Inference Approaches) competition [14] and ultimately won. The main idea of that algorithm is to translate the DFA identification problem to Boolean satisfiability (SAT). Thus we are able to use highly optimized modern DPLL-style SAT solving techniques [15]. The translation-to-SAT approach was also used to efficiently tackle problems such as bounded model checking [16], solving SQL constraints by incremental translation [17], analysis of JML-annotated Java sequential programs [18], extended finite-state machine induction [19].

Many optimization problems exhibit symmetries – groups of solutions which can be obtained from each other via some simple transformations. To speed up the solution search process we can reduce the problem search space by performing *symmetry breaking*. In DFA identification problems the most straightforward symmetries are groups of isomorphic automata. The idea of avoiding isomorphic DFAs by fixing state numbers in breadth-first search (BFS) order was used in the state-merging approach [20] (function `NatOrder`) and in the genetic algorithm from [21] (*Move To Front* reorganization). Besides, in [13] symmetry breaking was performed by fixing some colors of the APTA vertices from a clique provided by a greedy *max-clique* algorithm was applied in a preprocessing step of translation-to-SAT technique.

In this paper we propose new symmetry breaking predicates [15] which can be added to Boolean formulae representing various DFA identification problems. These predicates enforce DFA states to be enumerated in BFS order. Proposed predicates cannot be applied with the max-clique technique [13] at the same time, but our approach is more flexible. To show the flexibility of the approach, we

draw our attention to the case of noisy DFA identfication. Therefore we propose a modification of the noiseless translation-to-SAT for the noisy case (Section 3). We show that the previously proposed max-clique technique is not applicable in this case while our BFS-based approach is. The main advantage of our approach is that we can determine existence or non-existence of a solution in this case. Experiments showed that using BFS-based symmetry breaking predicates can significantly reduce the time of algorithm execution. Also we show that our strategy outperforms the current state-of-the-art EA from [12] if the number of the target DFA states, noise level and number of strings are small.

2 Encoding DFA Identfication into SAT

The goal of DFA identification is to find a smallest DFA A such that every string from S_+, a set of positive examples, is accepted by A, and every string from S_-, a set of negative examples, is rejected. The size of A is defined as the number of states C it contains. The alphabet $\Sigma = \{l_1, \ldots, l_L\}$ of the sought DFA A is the set of all L symbols from S_+ and S_-. The example of the smallest DFA for $S_+ = \{ab, b, ba, bbb\}$ and $S_- = \{abbb, baba\}$ is shown in Fig. 1. In this work we assume that DFA states are numbered from 1 to C and the start state has number 1.

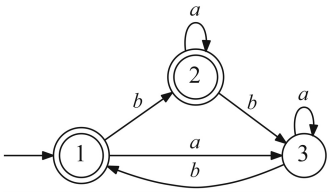
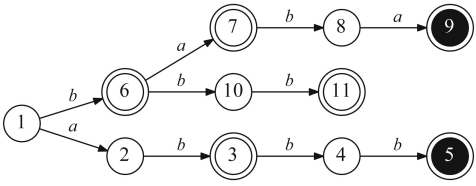


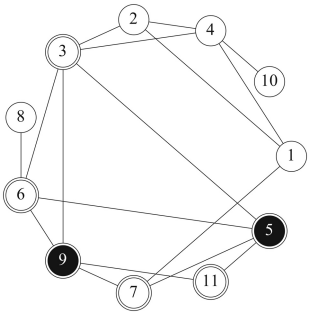
Fig. 1. An example of a DFA

In [13] Heule and Verwer proposed a compact translation of DFA identification problem into SAT. Here we briefly review the proposed technique, since our symmetry breaking predicates supplement it. The first step of both state-merging and translation-to-SAT techniques is augmented prefix tree acceptor (APTA) construction from the given examples S_+ and S_-. APTA is a tree-shaped automaton such that paths corresponding to two strings reach the same state v if and only if these strings share the same prefix in which the last symbol corresponds to v. We denote by V the set of all APTA states; by v_r – the APTA root; by V_+ – the set of accepting states; and by V_- – the set of rejecting states. Moreover, for state v (except v_r) we denote its incoming symbol as $l(v)$ and its parent as $p(v)$. The APTA for S_+ and S_- mentioned above is shown in Fig. 2a.

The second step of the technique proposed in [13] is the construction of the *consistency graph* (CG) for the obtained APTA. The set of nodes of the CG is identical to the set of APTA states. Two CG nodes v and w are connected



(a) An example of an APTA for $S_+ = \{ab, b, ba, bbb\}$ and $S_- = \{abbb, baba\}$



(b) The consistency graph for APTA from Fig. 2a

Fig. 2. An example of APTA and its consistency graph

with an edge (and called inconsistent) if merging v and w in APTA results in an inconsistency: an accepting state is merged with a rejecting state. Let E denote the set of CG edges. The CG for APTA of Fig. 2a is shown in Fig. 2b.

The key part of the algorithm is translating the DFA identification problem into a Boolean folmula in conjunctive normal form (CNF) and using a SAT solver to find a satisfying assignment. For a given set of examples and fixed DFA size C the solver returns a satisfying assignment (that defines a DFA with C states that is compliant with S_+ and S_-) or a message that it does not exist. The main idea of this translation is to use a distinct color for every state of the identified DFA and to find a consistent mapping of APTA states to colors. Three types of variables were used in the proposed compact translation:

1. *color* variables $x_{v,i} \equiv 1$ ($v \in V$; $1 \leqslant i \leqslant C$) iff APTA state v has color i;
2. *parent relation* variables $y_{l,i,j} \equiv 1$ ($l \in \Sigma$; $1 \leqslant i, j \leqslant C$) iff DFA transition with symbol l from state i ends in state j;
3. *accepting color* variables $z_i \equiv 1$ ($1 \leqslant i \leqslant C$) iff DFA state i is accepting.

Direct encoding, described in [13], uses only variables $x_{v,i}$; variables $y_{l,i,j}$ and z_i are auxiliary and are used in compact encoding predicates, which are described below.

The compact translation proposed in [13] uses nine types of clauses:

1. $x_{v,i} \Rightarrow z_i$ ($v \in V_+$; $1 \leqslant i \leqslant C$) – definitions of z_i values for accepting states ($\neg x_{v,i} \vee z_i$);
2. $x_{v,i} \Rightarrow \neg z_i$ ($v \in V_-$; $1 \leqslant i \leqslant C$) – definitions of z_i values for rejecting states ($\neg x_{v,i} \vee \neg z_i$);
3. $x_{v,1} \vee x_{v,2} \vee \ldots \vee x_{v,C}$ ($v \in V$) – each state v has at least one color;
4. $x_{p(v),i} \wedge x_{v,j} \Rightarrow y_{l(v),i,j}$ ($v \in V \setminus \{v_r\}$; $1 \leqslant i, j \leqslant C$) – a DFA transition is set when a state and its parent are colored ($y_{l(v),i,j} \vee \neg x_{p(v),i} \vee \neg x_{v,j}$);
5. $y_{l,i,j} \Rightarrow \neg y_{l,i,k}$ ($l \in \Sigma$; $1 \leqslant i, j, k \leqslant C$; $j < k$) – each DFA transition can target at most one state ($\neg y_{l,i,j} \vee \neg y_{l,i,k}$);

6. $\neg x_{v,i} \vee \neg x_{v,j}$ $(v \in V;\ 1 \leqslant i < j \leqslant C)$ – each state has at most one color;
7. $y_{l,i,1} \vee y_{l,i,2} \vee \ldots \vee y_{l,i,C}$ $(l \in \Sigma;\ 1 \leqslant i \leqslant C)$ – each DFA transition must target at least one state;
8. $y_{l(v),i,j} \wedge x_{p(v),i} \Rightarrow x_{v,j}$ $(v \in V \setminus \{v_r\};\ 1 \leqslant i, j \leqslant C)$ – state color is set when DFA transition and parent color are set $(\neg y_{l(v),i,j} \vee \neg x_{p(v),i} \vee x_{v,j})$;
9. $x_{v,i} \Rightarrow \neg x_{w,i}$ $((v, w) \in E;\ 1 \leqslant i \leqslant C)$ – the colors of two states connected with an edge in the consistency graph must be different $(\neg x_{v,i} \vee \neg x_{w,i})$.

Thus, the constructed formula consists of $\mathcal{O}(C^2|V|)$ clauses and, if the SAT solver finds a solution, we can identify the DFA.

To find a minimal DFA, authors use iterative SAT solving. Initial DFA size C is equal to the size of a *large clique* found in the CG. To find that clique, a greedy algorithm proposed in [13] can be applied. Then the minimal DFA is found by iterating over the DFA size C until the formula is satisfied.

The found clique was also used to perform symmetry breaking: in any valid coloring of a graph, all states in a clique must have a different color. Thus, we can fix the state colors in the clique in a preprocessing step. Later we will see that the max-clique symmetry breaking is not compatible with the one proposed in this paper.

To significantly reduce the SAT search space, the authors applied several EDSM steps before translation to SAT. Since EDSM cannot guarantee the minimality of solution, we will omit the consideration of this step in our paper.

3 Learning DFA from Noisy Samples

The translation described in the previous section deals with exact DFA identification. In this section we show how to modify the translation in order to apply it to noisy examples. We assume that not more than K attribution labels of the given training strings were randomly flipped. Solving this problem was the goal of the GECCO 2004 Noisy DFA competition [11] (with K equal to 10 percent of the number of the given training strings). An EA with smart state labeling was later proposed in [12], and since that time it is, to the best of our knowledge, the state-of-the-art technique for learning DFA from noisy training data.

In noisy case we cannot use APTA node consistency: we cannot determine whether an accepting state is merged with a rejecting state because correct string labels are unknown. Thus we cannot use CG and the max-clique symmetry breaking.

The idea of our modification is rather simple: for each labeled state of APTA we define a variable which states whether the label can be flipped. The number of flips is limited by K. Formally, for each $v \in V_{\pm} = V_+ \cup V_-$ we define f_v which is true if and only if the label of state v can (but does not have to) be incorrect (**f**lipped). Using these variables, we can modify the translation proposed in [13] to take into account mistakes in string labels. To do this, we change the z_i definition clauses (items 1 and 2 from list in Section 2): because of mistake possibility they hold in case f_v is false. Thus, new z_i value definitions are expressed in the following way: $\neg f_v \Rightarrow (x_{v,i} \Rightarrow z_i)$ for $v \in V_+$; $\neg f_v \Rightarrow (x_{v,i} \Rightarrow \neg z_i)$ for $v \in V_-$.

To limit the number of corrections to K we use an auxiliary array of K integer variables. This array stores the numbers of the APTA states for which labels can be flipped. Thus, f_v is true if and only if the array contains v. To avoid consideration of isomorphic permutations we enforce the array to be sorted in the increasing order.

To represent the auxiliary array as a Boolean formula we define variables $r_{i,v}$ for $1 \leqslant i \leqslant K$ and $v \in V_{\pm} = \{v_1, \ldots, v_W\}$. $r_{i,v}$ is true if and only if v is stored in the i-th position of the array. To connect variables f_v with $r_{i,v}$ we add so-called channeling constrains: $f_v \Leftrightarrow (r_{1,v} \vee \ldots \vee r_{K,v})$ for each $v \in V_{\pm}$.

We have to state that exactly one $r_{i,v}$ is true for each position i in the auxiliary array. To achieve that we use the order encoding method [22]. We add auxiliary **o**rder variables $o_{i,v}$ for $1 \leqslant i \leqslant K$ and $v \in V_{\pm} = \{v_1, \ldots, v_W\}$. We assume that $o_{i,v}$ for $v \in \{v_1, \ldots, v_j\}$ and $\neg o_{i,v}$ for $v \in \{v_{j+1}, \ldots, v_W\}$ for some j. This can be expressed by the following constraint: $o_{i,v_{j+1}} \Rightarrow o_{i,v_j}$ for $1 \leqslant j < W$. Now we define that $r_{i,v_j} \Leftrightarrow o_{i,v_j} \wedge \neg o_{i,v_{j+1}}$. Also we add clauses $o_{i,v_j} \Rightarrow o_{i+1,v_{j+1}}$ (for $1 \leqslant i < K$ and $1 \leqslant j < W$) to store corrections in increasing order.

The proposed constraints in CNF are listed in Table 1; there are $\mathcal{O}(C|V_{\pm}| + K|V_{\pm}|)$ clauses. Thus, to modify the translation for the noiseless case to deal with noise we can replace the z_i value definition and inconsistency clauses (items 1, 2 and 9 from list in Section 2) with the ones listed in Table 1.

Table 1. Clauses for noisy DFA identification

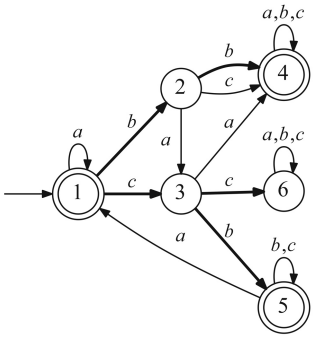
Clauses	CNF representation	Range
$\neg f_v \Rightarrow (x_{v,i} \Rightarrow z_i)$	$\neg x_{v,j} \vee z_j \vee f_v$	$1 \leqslant j \leqslant C$; $v \in V_+$
$\neg f_v \Rightarrow (x_{v,i} \Rightarrow \neg z_i)$	$\neg x_{v,j} \vee \neg z_j \vee f_v$	$1 \leqslant j \leqslant C$; $v \in V_-$
$f_v \Rightarrow (r_{1,v} \vee \ldots \vee r_{K,v})$	$\neg f_v \vee r_{1,v} \vee \ldots \vee r_{K,v}$	$v \in V_{\pm}$
$r_{i,v} \Rightarrow f_v$	$\neg r_{i,v} \vee f_v$	$1 \leqslant i \leqslant K$; $v \in V_{\pm}$
$r_{i,v_j} \Rightarrow o_{i,v_j}$	$\neg r_{i,v_j} \vee o_{i,v_j}$	$1 \leqslant i \leqslant K$; $1 \leqslant j \leqslant W$
$r_{i,v_j} \Rightarrow \neg o_{i,v_{j+1}}$	$\neg r_{i,v_j} \vee \neg o_{i,v_{j+1}}$	$1 \leqslant i \leqslant K$; $1 \leqslant j < W$
$o_{i,v_j} \wedge \neg o_{i,v_{j+1}} \Rightarrow r_{i,v_j}$	$\neg o_{i,v_j} \vee o_{i,b_{j+1}} \vee r_{i,v_j}$	$1 \leqslant i \leqslant K$; $1 \leqslant j < W$
$o_{K,v_W} \Rightarrow r_{K,v_W}$	$\neg o_{K,v_W} \vee r_{K,v_W}$	
$o_{i,v_{j+1}} \Rightarrow o_{i,v_j}$	$\neg o_{i,v_{j+1}} \vee o_{i,v_j}$	$1 \leqslant i \leqslant K$; $1 \leqslant j < W$
$o_{i,v_j} \Rightarrow o_{i+1,v_{j+1}}$	$\neg o_{i,v_j} \vee o_{i+1,v_{j+1}}$	$1 \leqslant i < K$; $1 \leqslant j < W$

4 Symmetry Breaking Predicates

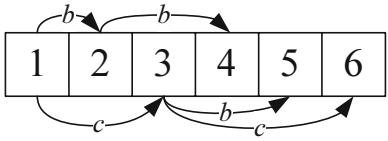
In this section we propose a way to fix automata state enumeration to avoid consideration of isomorphic DFAs during SAT solving. The main idea of our symmetry breaking is to enforce DFA states to be enumerated in breadth-first search (BFS) order. That idea was also used in function `NatOrder` in the state-merging approach described in [20] and the Move To Front reorganization algorithm used in the genetic algorithm [21].

BFS uses the *queue* data structure to store intermediate results as it traverses the graph. First we enqueue the initial DFA state (in this paper state number 1). While the queue is not empty we deque a state i and enqueue any direct child

states j that have not yet been discovered (enqueued before). Since our transitions are labeled with symbols from Σ, we enqueue child states in alphabetical order of symbols l on transitions $i \xrightarrow{l} j$. We call DFA *BFS-enumerated* if its states are enumerated in dequeuing (equals to enqueuing) order. An example of a BFS-enumerated DFA with six states shown in Fig. 3a (BFS-tree transitions that were used to enqueue states are marked bold); BFS enqueues are shown in Fig. 3b. The DFA shown in Fig. 1 is not BFS-enumerated – BFS first dequeues state 3 rather than state 2 (we consider $1 \xrightarrow{a} 3$ before $1 \xrightarrow{b} 2$).



(a) BFS-enumerated DFA with bolded BFS-tree edges



(b) BFS queue. Cells correspond to DFA states, transitions correspond to enqueues

Fig. 3. An example of BFS-numerated DFA and its BFS queue

We propose constraints that enforce DFA to be BFS-enumerated. We assume that translation of a given DFA identification problem to SAT deals with Boolean variables $y_{l,i,j}$ ($l \in \Sigma$; $1 \leqslant i, j \leqslant C$) to set the DFA transition function: $y_{l,i,j} \equiv 1$ iff transition with symbol l from state i ends in state j.

The main idea is to determine each state's parent in the BFS-tree and set constrains between states' parents. We store **p**arents in values $p_{j,i}$ (for each $1 \leqslant i < j \leqslant C$). $p_{j,i}$ is true if and only if state i is the parent of j in the BFS-tree. Each state except the initial one must have a parent with a smaller number, thus

$$\bigwedge_{2 \leqslant j \leqslant C} (p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1}).$$

Moreover, in BFS-enumeration states' parents must be ordered. State j must be enqueued before the next state $j+1$, thus the next state's parent k cannot be less than current state's parent i (see Fig. 4):

$$\bigwedge_{1 \leqslant k < i < j < C} (p_{j,i} \Rightarrow \neg p_{j+1,k}).$$

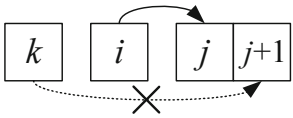


Fig. 4. Part of the queue illustrating the parent ordering predicates. Transitions show parent relations. The dotted transition is not allowed due to BFS-enumeration.

We set parents variables $p_{j,i}$ through $y_{l,i,j}$ using auxiliary variables $t_{i,j}$. In BFS-enumeration state j was enqueued while processing the state with minimal number i among states that have a transition to j:

$$\bigwedge_{1 \leqslant i < j \leqslant C} (p_{j,i} \Leftrightarrow t_{i,j} \wedge \neg t_{i-1,j} \wedge \ldots \wedge \neg t_{1,j}),$$

where $t_{i,j} \equiv 1$ iff there is a **t**ransition between i and j; we define these auxiliary variables using $y_{l,i,j}$:

$$\bigwedge_{1 \leqslant i < j \leqslant C} (t_{i,j} \Leftrightarrow y_{l_1,i,j} \vee \ldots \vee y_{l_L,i,j}).$$

Now to enforce DFA to be BFS-enumerated we have to order children in alphabetical order of symbols on transitions. We consider two cases: alphabet Σ consists of two symbols $\{a, b\}$ and more than two symbols $\{l_1, \ldots, l_L\}$. In the case of two symbols only two states can have the same parent i and they are forced by ordering constraints to have consecutive numbers j and $j + 1$. In this case we force the transition that starts in state i labeled with symbol a to end in state j instead of $j + 1$:

$$\bigwedge_{1 \leqslant i < j < C} (p_{j,i} \wedge p_{j+1,i} \Rightarrow y_{a,i,j}).$$

In the second case we have to introduce a third type of variables in our symmetry breaking predicates. We store the alphabetically **m**inimal symbol on transitions between states: $m_{l,i,j}$ is true if and only if there is a transition $i \xrightarrow{l} j$ and there is no such transition with an alphabetically smaller symbol. We connect these variables with DFA transitions by adding the following channeling predicates:

$$\bigwedge_{1 \leqslant i < j \leqslant C} \bigwedge_{1 \leqslant n \leqslant L} (m_{l_n,i,j} \Leftrightarrow y_{l_n,i,j} \wedge \neg y_{l_{n-1},i,j} \wedge \ldots \wedge \neg y_{l_1,i,j}).$$

Now it remains to arrange consecutive states j and $j+1$ with the same parent i in the alphabetically order of minimal symbols on transitions between them and i (see Fig. 5):

$$\bigwedge_{1 \leqslant i < j < C} \bigwedge_{1 \leqslant k < n \leqslant L} (p_{j,i} \wedge p_{j+1,i} \wedge m_{l_n,i,j} \Rightarrow \neg m_{l_k,i,j+1}).$$

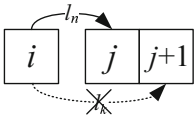


Fig. 5. Illustration of alphabetical ordering predicates. If i is the parent of j and $j+1$, l_n (l_k) is the alphabetically minimal symbol on transitions between i and j (i and $j+1$) then l_k cannot be alphabetically smaller than l_n

Thus we propose symmetry breaking predicates that are composed by listed constraints. Predicates (for three or more symbols case) translated into $\mathcal{O}(C^3 + C^2L^2)$ CNF clauses are listed in Table 2. Our implementation of proposed predicates and all algorithms can be found on the our labaratory github repository (https://github.com/ctlab/DFA-Inductor).

Table 2. BFS-based symmetry breaking clauses

Clauses	CNF representation	Range
$t_{i,j} \Rightarrow (y_{l_1,i,j} \vee \ldots \vee y_{l_L,i,j})$	$\neg t_{i,j} \vee y_{l_1,i,j} \vee \ldots \vee y_{l_L,i,j}$	$1 \leqslant i < j \leqslant C$
$y_{i,j,l} \Rightarrow t_{i,j}$	$\neg y_{l,i,j} \vee t_{i,j}$	$1 \leqslant i < j \leqslant C$; $l \in \Sigma$
$m_{l,i,j} \Rightarrow y_{l,i,j}$	$\neg m_{l,i,j} \vee y_{l,i,j}$	$1 \leqslant i < j \leqslant C$; $l \in \Sigma$
$m_{l_n,i,j} \Rightarrow \neg y_{l_k,i,j}$	$\neg m_{l_n,i,j} \vee \neg y_{l_k,i,j}$	$1 \leqslant i < j \leqslant C$; $1 \leqslant k < n \leqslant L$
$(y_{l_n,i,j} \wedge \neg y_{l_{n-1},i,j} \wedge \ldots \neg y_{l_1,i,j}) \Rightarrow m_{l_n,i,j}$	$\neg y_{l_n,i,j} \vee y_{l_{n-1},i,j} \vee \ldots \vee y_{l_1,i,j} \vee m_{l_n,i,j}$	$1 \leqslant i < j \leqslant C$; $1 \leqslant n \leqslant L$
$p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1}$	$p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1}$	$2 \leqslant j \leqslant C$
$p_{j,i} \Rightarrow t_{i,j}$	$\neg p_{j,i} \vee t_{i,j}$	$1 \leqslant i < j \leqslant C$
$p_{j,i} \Rightarrow \neg t_{k,j}$	$\neg p_{j,i} \vee \neg t_{k,j}$	$1 \leqslant k < i < j \leqslant C$
$(t_{i,j} \wedge \neg t_{i-1,j} \wedge \ldots \wedge \neg t_{1,j}) \Rightarrow p_{j,i}$	$\neg t_{i,j} \vee t_{i-1,j} \vee \ldots \vee t_{1,j} \vee p_{j,i}$	$1 \leqslant i < j \leqslant C$
$p_{j,i} \Rightarrow \neg p_{j+1,k}$	$\neg p_{j,i} \vee \neg p_{j+1,k}$	$1 \leqslant k < i < j < C$
$(p_{j,i} \wedge p_{j+1,i} \wedge m_{l_n,i,j}) \Rightarrow \neg m_{l_k,i,j+1}$	$\neg p_{j,i} \vee \neg p_{j+1,i} \vee \neg m_{l_n,i,j} \vee \neg m_{l_k,i,j+1}$	$1 \leqslant i < j < C$; $1 \leqslant k < n \leqslant L$

5 Experiments

All experiments were performed using a machine with an AMD Opteron 6378 2.4 GHz processor running on Ubuntu 14.04. All algorithms were implemented in Java, the *lingeling* SAT-solver was used. Our own algorithm was used for generating problem instances for all experiments based on randomly generated data sets. This algorithm builds a set of strings with the following parameters: size of DFA N which has to be generated, alphabet size A, number of strings S which have to be generated, noise level K (percent of attribution labels of generated strings which have to be randomly flipped).

In the exact case the max-clique method clearly outperforms BFS-based strategy.

For noisy DFA identification we used randomly generated instances. First we considered the case when the target DFA exists and the Boolean formula is satisfiable. We used following parameters: $N \in [5; 10]$, $A = 2$, $S \in \{10N, 25N, 50N\}$. We compared SAT approach without any symmetry breaking predicates, our

solution using BFS-based symmetry breaking predicates and the current state-of-the-art EA from [12]. Each experiment was repeated 100 times. The time limit was set to 1800 seconds. Initial experiments showed that EA clearly outperforms our method when $K > 4\%$. Therefore we set this parameter to $1\% - 4\%$. Results are listed in Table 3. We left only instances which were solved within the time limit. These results indicate that the BFS-based strategy finds the solution faster than the current state-of-the-art EA only when N is small (< 7), noise level is small ($1\% - 4\%$) and the number of strings is also small ($< 50N$). But BFS-based strategy finds the solution extremely faster than SAT approach without symmetry breaking strategy.

Table 3. Mean times of solving noisy DFA identification with count of strings in the each instance set to $10N$, $25N$ and $50N$ respectively

N	K	BFS, sec	SAT, sec	EA, sec
5	2	0.22	0.38	1.22
5	4	0.59	0.9	1.1
6	2	1.05	2.44	2.94
6	4	3.34	7.82	2.85
7	1	4.34	10.83	21.36
7	3	17.22	143.66	19.16
8	1	17.89	31.58	30.29
8	2	163.92	225.31	19.8

N	K	BFS, sec	SAT, sec	EA, sec
5	1	0.54	0.64	2.77
5	2	2.42	4.33	1.80
6	1	6.3	11.95	11.65
6	2	13.3	43.54	4.80
7	1	31.01	114.95	17.24
7	2	286.76	TL	13.11
8	1	239.46	404.32	21.73

N	K	BFS, sec	SAT, sec	EA, sec
5	1	4.2	7.59	6.07
5	2	12.87	22.36	3.05
6	1	20.76	52.5	20.39
6	2	107.94	309.22	11.28

The last experiment considered the case when the target DFA does not exist and the Boolean formula is unsatisfiable. Random dataset was also used here. We tried to find the target DFA using the following parameters: $N \in [5; 7]$, $A = 2$, $S = 50N$, $K \in [1; 2]$ percent. The input set of strings was generated from a $(N+1)$-sized DFA. It should be noted that the EA from [12] cannot determine that an automaton consistent with a given set of strings does not exist. On the other hand, all SAT-based methods are capable of that. Therefore we compared our implementation of compact SAT encoding without using symmetry breaking predicates and the same with BFS-based predicates. Each experiment was repeated 100 times and the time limit was set to 1800 seconds again. Results are listed in Table 4. It can be seen from the table that BFS-based strategy significantly reduces the mean time of determination that an automaton does not exist.

Table 4. Mean times and percent of passed solutions of solving noisy DFA identification when the target DFA does not exist

N	K	BFS, sec	WO, sec	passed BFS, %	passed WO, %
5	1	11.57	257.13	100	100
5	2	46.42	1296.71	100	30
6	1	110.05	TL	100	0
6	2	581.73	TL	100	0
7	1	995.27	TL	89	0
7	2	TL	TL	0	0

6 Conclusions and Future Work

We proposed symmetry breaking predicates which can be added to the Boolean formula representing various DFA identification problems. By adding the predicates we can reduce SAT search space through enforcing DFA states to be enumerated in breadth-first search (BFS) order.

We drew our attention to the case of noisy DFA identfication. We proposed a modification of the noiseless translation-to-SAT [13] for the noisy case. To achieve compact encoding for that case we used the order encoding method. We showed that the previously proposed max-clique technique for symmetry breaking is not applicable in the noisy case while our BFS-based approach is. We showed that the BFS-based strategy can be applied in the noisy case when an automaton which is consistent with a given set of strings does not exists. The current state-of-the-art EA from [12] cannot determine that. In experimental results, we showed that our approach with BFS-based symmetry breaking predicates clearly outperforms algorithm without any predicates. Also we showed that our strategy outperform EA if the number of the target DFA states is small, noise level is small and number of strings is small either.

We plan to translate noisy DFA identification to Max-SAT in order to limit the number of corrections without using an auxiliary array of integer variables. Also we plan to experiment with alternative integer encoding methods. In the future we would like to solve a problem of finding all solutions (instead of a single DFA) using our predicates.

Acknowledgements. Authors would like to thank Daniil Chivilikhin, Igor Buzhinsky and Andrey Filchenkov for useful comments. This work was financially supported by the Government of Russian Federation, Grant 074-U01, and also partially supported by RFBR, research project No. 14-07-31337 mol_a.

References

1. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (2006)
2. De La Higuera, C.: A bibliographical study of grammatical inference. Pattern Recognition **38**(9), 1332–1348 (2005)
3. Gold, E.M.: Complexity of automaton identification from given data. Information and Control **37**(3), 302–320 (1978)
4. Pitt, L., Warmuth, M.K.: The minimum consistent DFA problem cannot be approximated within any polynomial. Journal of the ACM **40**(1), 95–142 (1993)
5. 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, pp. 1–112. Springer, Heidelberg (1998)
6. Lang, K.J.: Faster Algorithms for Finding Minimal Consistent DFAs. Technical report (1999)
7. Bugalho, M., Oliveira, A.L.: Inference of regular languages using state merging algorithms with search. Pattern Recognition **38**(9), 1457–1467 (2005)

8. Dupont, P.: Regular grammatical inference from positive and negative samples by genetic search: the GIG method. In: Carrasco, R.C., Oncina, J. (eds.) ICGI 1994. LNCS, vol. 862, pp. 236–2445. Springer, Heidelberg (1994)
9. Luke, S., Hamahashi, S., Kitano, H.: Genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, pp. 1098–1105 (1999)
10. Lucas, S.M., Reynolds, T.J.: Learning DFA: evolution versus evidence driven state merging. In: The 2003 Congress on Evolutionary Computation, CEC 2003, vol. 1, pp. 351–358. IEEE (2003)
11. Lucas, S.: GECCO 2004 noisy DFA results. In: GECCO Proc. (2004)
12. Lucas, S.M., Reynolds, T.J.: Learning deterministic finite automata with a smart state labeling evolutionary algorithm. IEEE Transactions on Pattern Analysis and Machine Intelligence **27**(7), 1063–1074 (2005)
13. Heule, M.J.H., Verwer, S.: Exact DFA identification using SAT solvers. In: Sempere, J.M., García, P. (eds.) ICGI 2010. LNCS, vol. 6339, pp. 66–79. Springer, Heidelberg (2010)
14. Walkinshaw, N., Lambeau, B., Damas, C., Bogdanov, K., Dupont, P.: STAMINA: a competition to encourage the development and assessment of software model inference techniques. Empirical Software Engineering **18**(4), 791–824 (2013)
15. Biere, A., Heule, M., van Maaren, H.: Handbook of satisfiability, vol. 185. IOS Press (2009)
16. Amla, N., Du, X., Kuehlmann, A., Kurshan, R.P., McMillan, K.L.: An analysis of SAT-based model checking techniques in an industrial environment. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 254–268. Springer, Heidelberg (2005)
17. Lohfert, R., Lu, J.J., Zhao, D.: Solving SQL constraints by incremental translation to SAT. In: Nguyen, N.T., Borzemski, L., Grzech, A., Ali, M. (eds.) IEA/AIE 2008. LNCS (LNAI), vol. 5027, pp. 669–676. Springer, Heidelberg (2008)
18. Galeotti, J.P., Rosner, N., Lopez Pombo, C.G., Frias, M.F.: TACO: Efficient SAT-Based Bounded Verification Using Symmetry Breaking and Tight Bounds. IEEE Transactions on Software Engineering **39**(9), 1283–1307 (2013)
19. Ulyantsev, V., Tsarev, F.: Extended finite-state machine induction using SAT-solver. In: Proc. of ICMLA 2011, vol. 2, pp. 346–349. IEEE (2011)
20. Lambeau, B., Damas, C., Dupont, P.E.: State-merging DFA induction algorithms with mandatory merge constraints. In: Clark, A., Coste, F., Miclet, L. (eds.) ICGI 2008. LNCS (LNAI), vol. 5278, pp. 139–153. Springer, Heidelberg (2008)
21. Chambers, L.D.: Practical handbook of genetic algorithms: complex coding systems, vol. 3. CRC Press (2010)
22. Barahona, P., Hölldobler, S., Nguyen, V.: Efficient SAT-encoding of linear csp constraints. In: 13th International Symposium on Artificial Intelligence and Mathematics-ISAIM, Fort Lauderdale, Florida, USA (2014)