Regular inference as vertex coloring[☆]

Christophe Costa Florêncio[a], Sicco Verwer[b,1]

[a] *Institute for Computer Science, University of Amsterdam, the Netherlands*
[b] *Institute for Computing and Information Sciences, Radboud University Nijmegen, the Netherlands*

Abstract

This paper is concerned with the problem of supervised learning of deterministic finite state automata, in the technical sense of identification in the limit from complete data, by finding a minimal DFA consistent with the data (regular inference).

We solve this problem by translating it in its entirety to a vertex coloring problem. Essentially, such a problem consists of two types of constraints that restrict the hypothesis space: *inequality* and *equality* constraints. Inequality constraints translate to the vertex coloring problem in a very natural way. Equality constraints however greatly complicate the translation to vertex coloring. In previous coloring-based translations, these were therefore encoded either dynamically by modifying the vertex coloring instance on-the-fly, or by encoding them as satisfiability problems.

We provide the first translation that encodes both types of constraints together in a pure vertex coloring instance. We prove the correctness of the construction, and show that regular inference and vertex coloring are in some sense equally hard.

The coloring approach offers many opportunities for applying insights from combinatorial optimization and graph theory to regular inference. We immediately obtain new complexity bounds, as well as a family of new learning algorithms which can be used to obtain exact hypotheses as well as fast approximations.

Keywords: Grammar induction, automata induction, vertex coloring, satisfiability, identification in the limit

1. Introduction

The regular inference problem consists of learning (finding) a smallest deterministic finite state automaton (DFA) that is consistent with a given sample (set) of labeled strings, rejecting the negative strings and accepting the positive strings. The decision version of finding a DFA with a given upper bound on its size (number of states) was shown to be **NP**-complete in [5, 24], and an inapproximability result was

[☆] An abridged version of this paper appeared as [12].

Email addresses: C.CostaFlorencio@uva.nl (Christophe Costa Florêncio), S.Verwer@cs.ru.nl (Sicco Verwer)

[1] The second author is supported by STW project 11763 Integrating Testing And Learning of Interface Automata (ITALIA).

demonstrated in [32]. In spite of these hardness results, quite a few DFA identification algorithms have been developed over time, see [14] for an overview. In particular, encouraging results have been obtained with a recently proposed algorithm based on a *translation* of the regular inference problem into satisfiability (SAT) [22].

This translation is based on an earlier translation of regular inference to vertex coloring introduced in [13]. Vertex coloring is the problem of assigning a color to every vertex in a given graph such that vertices with the same color are not connected, and doing so using a minimal number of colors. Determining whether a coloring exists that uses at most $k \geq 3$ colors is a well-known **NP**-complete problem, see, e.g., [18].[2] The main idea of our reduction to vertex coloring is to use a distinct color for every state of the target DFA. The vertices in the coloring instance represent substrings of the labeled strings from the sample and are connected just if there is a string s that is accepted when starting from one of them and rejected when starting from the other. The vertex coloring problem thus ensures that the resulting DFA is consistent. The size of this DFA is determined by the amount of colors needed to solve the vertex coloring problem, a minimum size DFA is found by iterating over this amount.

The reduction from [13], however, is not based purely on vertex coloring. In addition to the *inequality constraints*, which stipulate that that two vertices cannot be assigned the same color, so-called *equality constraints* are needed to model regular inference. These constraints stipulate that two vertices should be be in the same color class if two other vertices are in the same color class. Together, the equality and inequality constraints can efficiently represent an instance of the regular inference problem. Until now it has not been clear exactly how to encode such constraints as a vertex coloring problem instance. In [13], these were encoded dynamically by generating new vertex coloring instances that satisfied the constraints on-the-fly. In [22], they were encoded directly into SAT instances instead of in the intermediate vertex coloring instance, thus benefiting as much as possible from the highly optimized modern SAT solving techniques. In this paper, we present a new construction that encodes them directly into vertex coloring.

In terms of complexity (size), our encoding of the equality constraints is comparable to the encoding to satisfiability described in [22]: they both require $O(|C|^2 \cdot |V|)$ additional clauses or vertices, where C is the set of colors and V is the size of the data set (the APTA, see Section 2). The inequality constraints, however, can be encoded in vertex coloring in a very natural way, requiring only a single edge for every constraint compared to the $O(|C|^2)$ (or $O(|C|)$ for some that can be encoded more efficiently, see [22]) clauses that are needed for every such constraint in a satisfiability instance. In addition, using our encoding we can make use of sophisticated solvers for vertex coloring, including techniques for symmetry-breaking, many local-search based approaches, cutting-plane algorithms, etc. See, e.g., [29].

The rest of the paper is structured as follows: Section 2 covers the required background. Section 3

[2] It remains **NP**-complete even when restricted to 3-colorability of planar graphs with maximum degree 4, see [17].

introduces and discusses a first version of our construction. It may lead to a quadratic blowup in size, but it is the more accessible of the two; Section 4 develops a more efficient variant. Section 5 discusses a learning algorithm, its time complexity is discussed in Section 6. Section 7 shows how coloring problems can be transformed into learning problems. Finally, Section 8 concludes the paper and suggests some topics for future research.

2. Background and notation

2.1. Regular inference

A *deterministic finite state automaton* (DFA) is one of the basic and most commonly used variety of finite state machines. We provide a concise description of DFAs below, the reader is referred to [33] for a more elaborate overview.

A DFA $\mathcal{A} = \langle Q, T, \Sigma, q_0, F \rangle$ is a directed graph consisting of a set of *states* Q (vertices) and labeled *transitions* T (directed edges). The *start state* $q_0 \in Q$ is a specific state of the DFA and any state can be an *accepting state* (final state) in $F \subseteq Q$. The labels of transitions are all members of a given *alphabet* Σ. A DFA $\mathcal{A}$ can be used to *generate* or *accept* sequences of symbols (strings) using a process called *DFA computation*. This process begins in q_0, and iteratively *activates* (or *fires*) an outgoing transition $t_i = \langle q_{i-1}, q_i, l_i \rangle \in T$ with label $l_i \in \Sigma$ from the *source state* it is in, q_{i-1}, moving the process to the *target state* q_i pointed to by t_i. A computation $q_0 t_1 q_1 t_2 q_2 \dots t_n q_n$ is *accepting* if the state it *ends* in (its last state) is an accepting state, i.e., $q_n \in F$, otherwise it is *rejecting*. The labels of the activated transitions form a string $l_1 \dots l_n$. A DFA accepts exactly those strings formed by the labels of accepting computations, and rejects all others. By definition, DFA is *deterministic*, which means that for every state q and every label l there exists at most one outgoing transition from q with label l. The set of all strings accepted by a DFA $\mathcal{A}$ is called the *language* $L(\mathcal{A})$ of $\mathcal{A}$.

Given a pair S of finite sets, one of positive example strings S_+ and one of negative example strings S_-, which is called the *input sample*, the goal of *regular inference* (or *DFA identification/learning*) is to find a (non-unique) *smallest* DFA $\mathcal{A}$ that is *consistent* with $S = \{S_+, S_-\}$. That is, every string in S_+ is accepted by $\mathcal{A}$, and every string in S_- is rejected by $\mathcal{A}$. In this context, the size of a DFA is defined as the number of states it contains. Developing efficient search algorithms for finding such DFAs is an active research topic in the grammatical inference community, see, e.g., [14].

For many years, the state-of-the-art in DFA identification has been the *evidence-driven state-merging* (EDSM) algorithm [27]. State-merging is a common technique from grammatical inference for learning a small language model by combining (merging) the states of a large initial DFA model, see, e.g., [14]. Essentially, EDSM is a *greedy method* that tries to find a good local optimum efficiently. In addition, an earlier state-merging method called RPNI has been shown to *converge efficiently* (from polynomial time and

data) to the global optimum in the limit [31]. EDSM participated in and won (in a tie) the Abbadingo DFA learning competition in 1997 [27].

Since our method is based on state-merging, we now briefly explain this simple yet effective approach. For more information the reader is referred to [14]. The key idea of state-merging is to first construct a tree-shaped DFA $\mathcal{A}$ from the input sample S, and then to merge (combine) the states of $\mathcal{A}$. This initial DFA is called an *augmented prefix tree acceptor* (APTA). An example is shown in Figure 1.

Definition 1 *The* APTA $\mathcal{A} = (\langle Q, T, \Sigma, q_0, F\rangle, R)$ *for an input sample* $\{S_+, S_-\}$ *consists of a DFA* $\langle Q, T, \Sigma, q_0, F\rangle$ *and a set of rejecting states* R*, where* Σ *is the alphabet of* $S_+ \cup S_-$*,* $q_0 = \epsilon$ *(the empty word),* $Q = \{a \in \Sigma^* \mid \exists b \in \Sigma^* : ab \in (S_+ \cup S_-)\}$*,* $T = \{\langle a, a', l\rangle \in Q \times Q \times \Sigma \mid a' = al\}$*,* $F = S_+$*, and* $R = S_-$*.*

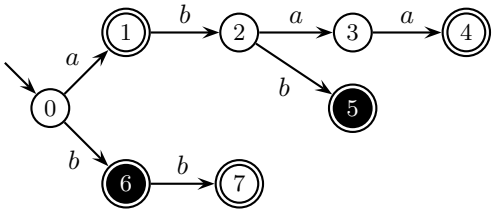

Figure 1: An augmented prefix tree acceptor for $S = (S_+ = \{a, abaa, bb\}, S_- = \{abb, b\})$. The start state is the state with an arrow pointing to it from nowhere.

A *merge* of two states q and q' combines the states into one: it creates a new state q^* that has the incoming and outgoing transitions of both q and q', which are subsequently removed from $\mathcal{A}$. Such a merge is only allowed if the states are *consistent*, i.e., it is not the case that one of the two is accepting while the other is rejecting

When a merge introduces a non-deterministic choice, i.e., q^* is the source of two transitions with the same label l, the target states of these transitions, q_1 and q_2, are merged as well. This is called the *merging for determinization* process and is repeated until there are no non-deterministic choices left. However, if this process at some point merges two inconsistent states, the original states q and q' are also considered inconsistent and the merge will fail. The result of a successful merge is a new DFA that is smaller than before, and still consistent with the input sample S. A state-merging algorithm iteratively applies this state merging process until no more consistent merges are possible.

Withinn the grammatical inference community, the development of advanced and efficient search techniques based on the EDSM heuristic has been an active research topic for the last decade or so. 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 [30],

using mutually (in)compatible merges [3], and searching most-constrained vertices first [26]. A comparison of different search techniques for EDSM can be found in [10]. Recently, instead of wrapping a search technique around EDSM, a *translation* of the regular inference problem into satisfiability (SAT) was proposed in order to use a state-of-the-art SAT-solver to search for an optimal solution [22]. The main advantage of such an approach is that it makes use directly of advanced search techniques such as conflict analysis, intelligent back-jumping, and clause learning, see, e.g., [7]. The winning contribution to the 2010 Stamina DFA learning competition was a combination of this SAT-based approach and EDSM with a modified heuristic [34]. Other recently proposed improvements to the algorithm are a parallelized version [4], and the use of ensembles of learned DFAs [16].

2.2. Translating regular inference

The idea of translating the regular inference problem to other computational problems for which dedicated solvers exist is by no means new. In fact, one of the earliest regular inference algorithms, due to Biermann [8], is of this type. Biermann proposed to solve the regular inference problem by mapping it to constraint satisfaction. In his translation, every state is represented by a natural number, and constraints on the possible values of states are added to enforce consistency. The range of these numbers is then minimized, this corresponds to minimizing the number of states in the resulting DFA. More recently, Grinchtein et al. [20] adapted this translation in order to map regular inference to satisfiability (SAT) instead of constraint satisfaction. The numeric constraints from the constraint satisfaction problem are encoded using either a unary or a binary scheme into clauses and literals.

Another type of translation, introduced in Coste [13], maps regular inference to vertex coloring based on the state-merging approach. The main idea of this translation is to use a distinct color for every state of the identified DFA. Every vertex in the graph which is to be colored corresponds to a distinct state in the APTA. Two vertices v and w in this graph are connected by an edge (and thus cannot be assigned the same color) just if merging v and w would lead to an inconsistency in the original regular inference problem:

Definition 2 *The* consistency graph $G_c = (V, E_c)$ *for an APTA* $\mathcal{A} = (\langle Q, T, \Sigma, q_0, F\rangle, R)$ *consists of a set of vertices* V *and edges* E_c *such that* $V = Q$ *and* $E_c = \{\{a, a'\} \in \Sigma^* \times \Sigma^* \mid \exists b \in \Sigma^* : ab \in F \text{ and } a'b \in R\}$.

The edges in this graph are called *inequality constraints*. Figure 2 shows an example of such a graph. This mapping is a very natural one, however it is not complete. In addition to these inequality constraints, *equality constraints* are required: if the parents of two states (in the APTA) with the same incoming transition label are merged, then these states must be merged as well, in order to obtain a deterministic automaton.

Definition 3 *The set of* equality constraints E_e *for an APTA* $\mathcal{A} = (\langle Q, T, \Sigma, q_0, F\rangle, R)$ *is the set of pairs of paired states* $\langle (a, b), (al, bl)\rangle \subset Q^2 \times Q^2$ *with* $a, b \in \Sigma^*$ *and* $l \in \Sigma$.

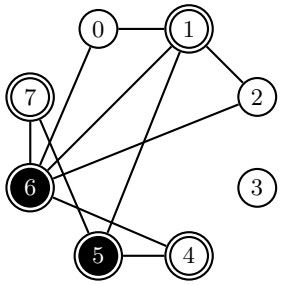

Figure 2: The consistency graph corresponding to the APTA of Figure 1. Some states in the consistency graph are not directly inconsistent, but inconsistent due to determinization. For instance states 2 and 6 are inconsistent because the strings *abb* (negative) and *bb* (positive) would end in the same state if these states were merged.

These equality constraints encode that two parent states a and b can be in the same color class only if their child states al and bl are in the same color class. Until now it has been unclear how to encode such constraints in a vertex coloring problem instance. In [13], these were encoded by modifying the graph according to the consequences of these constraints. This implies that a new vertex coloring instance has to be solved every time an equality constraint is used. Apart from being inefficient, this limits our choice of coloring algorithms to those that incrementally color a graph, vertex by vertex. Thirteen years later, this graph coloring encoding in [13] was used by Heule and Verwer as the basis for a more efficient translation to satisfiability [22], which encodes the equality constraints directly.

In the following, we develop a novel construction that encodes the equality constraints directly into vertex coloring.

2.3. Vertex coloring

We briefly discuss vertex coloring in this subsection, and assume that the reader is familiar with the more basic concepts from graph theory.

A *coloring* of a graph is a function from its vertices to *colors* (or *color classes*). The term colors is due to historical reasons; it was originally studied in the context of coloring maps. In the remainder, we will simply use natural numbers as names for these colors.

A coloring is called *proper* for graph G if no two connected vertices in G have the same color, and *optimal* for G if it is both proper and assigns the smallest possible number of colors to the vertices of G. This number is known as the *chromatic number* of G, denoted by $\chi(G)$. We will write $\text{color}(x) = c$ when vertex x is labeled with color c. We write $x =_c y$ to indicate that vertices x, y are members of the same color class.

When we call an optimal coloring *unique*, this is taken to mean unique *up to recoloring*. Recoloring can be understood as renaming, i.e., applying a substitution σ to the color labels of G such that, whenever for any two vertices x, y from G, $x =_c y$, then $\sigma[x] =_c \sigma[y]$, and when $x \neq_c y$, then $\sigma[x] \neq_c \sigma[y]$. Note that for every recoloring, its inverse exists.

3. Encoding equality constraints into the graph

In this section we will show how equality constraints can either be encoded into the graph, or can be reduced to simple checks after a coloring has been generated.

3.1. Graphs with chromatic number less than or equal to two

The 1-colorable graphs are obviously exactly the edgeless graphs. Since all vertices of the graph are members of the same color class, the issue of equality constraints is irrelevant in this case.[3]

Also note that a target automaton with just one state always generates Σ^*; for any sample for such a language, $S_- = \emptyset$.

It is a well-known fact that the 2-colorable graphs are exactly the bipartite graphs. For this class, a coloring can be found in polynomial time with a parity-based algorithm: pick an arbitrary vertex v and label all vertices in the graph with their distance to v (which can be done with depth-first search). We obtain a bipartite graph, one partition of which consists of all vertices at even distance from v, and the other partition of which consists of all vertices at odd distance from v. Each partition can then be regarded as a color class. Two vertices obviously have the same color if and only if they are members of the same partition.

Equality constraints can be ignored when both of the pairs of vertices involved in such a constraint are from the same connected component of the bipartite graph. It suffices to check that the constraint is not violated *after* the coloring has been assigned to the consistency graph.[4]

However, in the case that G_c consists of multiple connected components, equality constraints may block certain merges, resulting in $\chi(G_e) > 2$, so the coloring problem becomes NP-complete.[5]

3.2. Graphs with chromatic number greater than or equal to three

When the chromatic number of the consistency graph is three or more, equality constraints have to be taken into account. This requires the construction of a graph that, for each equality constraint, includes a gadget as seen in Figure 3.

This construction is formally defined as follows:

[3] As an aside, it should be noted that $\chi(G_c(S)) = 1$ does *not* in itself imply that the target automaton consists of just one state. This can be easily seen by considering any sample with $S_- = \emptyset$. This implies the complete absence of conflicts, but this may simply be due to a sample not being representative for the target language.

[4] Technically speaking, even this check is not necessary: a violation can only occur if the sample is inconsistent, and such a case is excluded by definition.

[5] A recent proof of NP-completeness of the problem for 2-state automata, based on a reduction from SAT, can be found in [15]. It is also shown there that this more restricted problem can be solved in polynomial time when the samples consists of just two words (Theorem 16).

It is pointed out in an extended version of that paper that an earlier, unpublished proof is due to Dana Angluin. This is mentioned in [2], and without reference in [14].

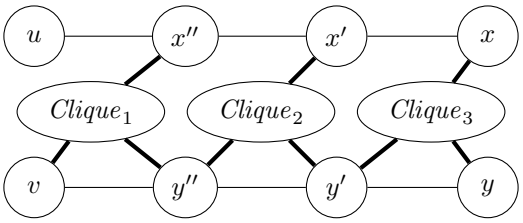

Figure 3: This gadget encodes equality constraints into a graph. A thick line represents a set of edges that connect a vertex (circle) to all vertices in a clique (ellipse).

Definition 4 *Given a consistency graph $G_c = G_c(S) = (V, E)$, let $\chi = \chi(G_c)$, and let $Clique_1$, $Clique_2$ and $Clique_3$ be three disjoint cliques of size $\chi - 2$.*

Let E_e be the set of equality constraints for APTA(S)*, and let $G_e = (V \cup V', E \cup E')$ be the smallest graph such that, for each equality constraint $e = \langle (u, v), (x, y) \rangle \in E_e$,*

1. *$Clique_1$, $Clique_2$ and $Clique_3$ are in the graph;*
2. *vertices x', x'', y', y'' are in the graph;*
3. *v, x'', y'' are connected to all vertices in $Clique_1$;*
4. *x', y'', y' are connected to all vertices in $Clique_2$;*
5. *y', x, y are connected to all vertices in $Clique_3$;*
6. *u is connected to x'', $x''$to x', v to y'', y'' to y', x to x', and y to y'.*

We are now in the position to state a lemma which will play a key role in the remainder of this paper:

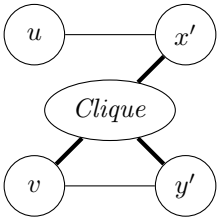

Figure 4: The subgraph discussed in Lemma 5.

Lemma 5 *Given a graph G, let $\chi = \chi(G)(\geq 3)$, and let G' be an induced subgraph of G which is isomorphic to $G''[u, v, x', y']$, where G'' is the graph shown in Figure 4. Note that its clique is of size $\chi - 2$.*

Then, given any optimal coloring for G':

1. *either $x' =_c v$, or $x' =_c y'$;*
2. *if it is the case that $u =_c v$, then we also have $x' =_c y'$.*

Proof. Let C be the set of all colors used in some optimal coloring of G, and let C_1 be the colors assigned to the subgraph *Clique* from G''. Since *Clique* is of size $\chi - 2$, $\chi(\mathit{Clique}) = \chi - 2$, thus $|C - C_1| = 2$.

Because v is connected to all vertices in *Clique*, it has to be assigned some color c_v from $C - C_1$. Since y' is connected to v and to all vertices in *Clique*, it has to be assigned the color $C - C_1 - c_v$.

Vertex x' is connected to all vertices in *Clique*, so it has a color from $C - C_1$. Since this set contains just 2 colors, either $x' =_c v$, or $x' =_c y'$, which proves 1.

Since x' is connected to u, it has a color from $C - C_1 - c_u$. If $u =_c v$, this set is a singleton and contains just the color assigned to y', which proves 2. □

We are now in a position to prove correctness of our construction.

Proposition 6 *Let $G_c = G_c(S)$, and $\chi(G_c) \geq 3$. Let $G_e = G_e(S)$ (as given in Definition 4).*

Then, given an optimal coloring for G_e, for any equality constraint $e = \langle (u, v), (x, y) \rangle \in E_e(S)$, if the vertices corresponding to u and v are in the same color class, then so are x and y.

Proof. Given an equality constraint which states that merging u and v requires merging x and y, we show the following:

1. in our construction, if vertices u and v are in the same color class, then x and y are members of the same color class, for any minimal coloring;
2. if vertices u and v are *not* in the same color class, then x and y can be in the same color class, but this is not necessarily the case;
3. we show that our construction is correct for any combination of 3 colors;
4. we show that it remains correct for any combination of more than 3 colors.

Demonstrating these four points together proves the proposition. First, let $C(G) = \{c_1, \ldots c_\chi\}$ be the set of all colors used in any optimal coloring of graph G ($\chi = \chi(G)$).

Point 1 can be demonstrated by applying Lemma 5 three times: if $u =_c v$, then $x'' =_c y''$; if $x'' =_c y''$, then $x' =_c y'$; if $x' =_c y'$, then $x =_c y$.

Thus $u =_c v$ implies $x =_c y$.

We now proceed to demonstrate point 2 using the same method: if $u \neq_c v$, then by Lemma 5 an optimal coloring exists with $x'' \neq_c y''$; similarly for $x' \neq_c y'$; and thus $x \neq_c y$. If $u \neq_c v$, then by Lemma 5 an optimal coloring exists with $x'' =_c y''$; therefore $x' =_c y'$; and thus $x =_c y$.

Point 3 can best be demonstrated by case analysis, i.e., simply enumerating all possible colorings (up to recoloring). As the reader may check, Figure 5 exhaustively enumerates all possible cases for $\chi = 3$. i.e. all unique colorings.

We conclude by demonstrating point 4:

Points 1 and 2 hold for any $\chi \geq 3$, so it suffices to generalize point 3 to cases where $\chi \geq 4$. Let G_χ be a gadget as in Definition 4, for some $\chi \geq 4$, and G_3 the same for $\chi = 3$. It is clear that G_3 is an induced subgraph of G_χ. To be more precise, G_χ can be obtained from G_3 by adding $i - 3$ distinct new vertices to each of its three central cliques and connecting them in the obvious way.

It is easy to see that, in the case that $x =_c y$ and $u =_c v$, we can obtain an optimal coloring for G_χ by picking (a recoloring of) one of the lowest three colorings from Figure 5. This colors a subgraph isomorphic to G_3, the colors for the vertices not in this subgraph are the 'new' ones added to each of the central cliques Clique_i are obtained simply by non-deterministically assigning them from $C - C(\mathit{Clique}_i) - C(N(\mathit{Clique}_i))$ (where $C(G)$ yields the colors assigned to vertices in G, and $N(G)$ yields the union of neighborhoods of all vertices in G).

In the case that $x =_c y$ and $u \neq_c v$, colorings can be obtained from the middle three colorings from Figure 5. The top left vertex, u, can be assigned any color as long as $u \neq_c v$, since it's not connected to any of the cliques. It is connected only to x'', and we have $x'' =_c v$. If v gets assigned a color such that $\text{color}(v) > 3$, we get $v \neq_c y''$, so we get a proper coloring when no vertex in Clique_1 is assigned $\text{color}(v)$ or $\text{color}(y'')$ (which is 1 or 2 in the figure). The same line of reasoning can be applied to x and y: If x and y get assigned a color such that $\text{color}(x) > 3$, we obtain an admissible coloring just when no vertex in Clique_3 is assigned $\text{color}(x)$ or $\text{color}(x'')$ (which is 1 or 2 in the figure).

For the case that $x \neq_c y$, consider the top three colorings from Figure 5. A complicating factor is that the lower right vertex, y, has multiple options for coloring; for a gadget for χ colors, there are $\chi - 1$ options. It is easy to see though that the only restriction on $\text{color}(y)$ is that $y \neq_c y'$, which implies $y \neq_c x$, since $y' =_c x$ for all three gadgets. So, if $\text{color}(y) > 3$, the other of the $\chi - 1$ options can be assigned to vertices in Clique_3 and an admissible coloring is obtained. For vertices x, u, v, reasoning from the previous paragraphs applies.

We have thus demonstrated the validity of all four points, which concludes the proof. □

4. More efficient equality constraints

The translation described above encodes the equality constraints from the regular inference problem, but unfortunately it is not very efficient: in the worst case it can require up to $O(\|S\|^2)$ cliques of size $\chi - 2$. Since S (the input sample) can get very large, this quadratic relation is highly undesirable. In [22], a similar problem was observed for a translation of regular inference to satisfiability. This was solved by introducing additional variables that encode the equality constraints globally, i.e., for the complete resulting automaton model instead of separately for each pair of APTA states. Below, we show that such a global encoding is also possible for our translation to vertex coloring, and that it reduces this quadratic relation to a linear one as well.

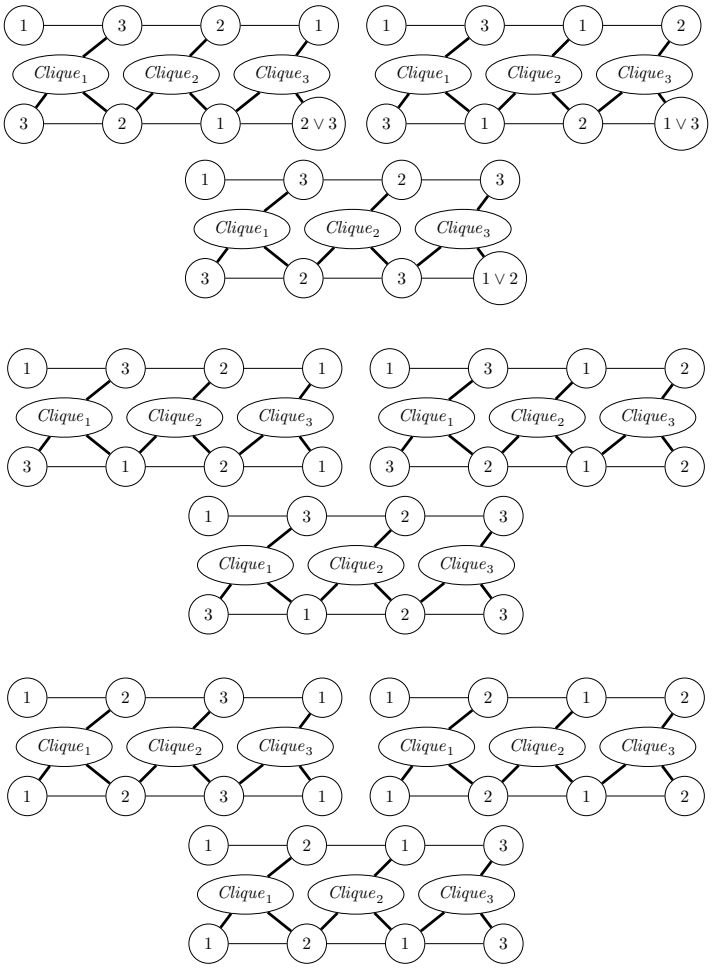

Figure 5: Possible colorings for the $\chi(G_c) = 3$ construction.

The key idea is to introduce two additional sets of vertices that encode the states of the resulting automaton model and the transitions between them. The first set contains a clique of χ vertices, one for every state of the automaton. The second set contains $\chi \cdot |\Sigma|$ pairwise non-connected vertices, one for every possible transition of the automaton. We denote the vertices from the first set using q_i (state i in the resulting automaton), and those from the second set using $t_{i,l}$ (the target of the transition from state i with label a). We now replace the v and y vertices from the gadget in Figure 3 by the q_i and $t_{i,l}$ vertices shown in Figure 6. This construction is identical to the previous one, except that it connects every pair of vertices (u, x) that is used in an equality constraint $\langle u, v, x, y\rangle \in E_e$ for a label l ($v = ul$) to $(q_i, t_{i,l})$ for all $0 \leq i < \chi$. If two pairs of vertices (u, x) and (v, y) were connected by the gadget in Figure 3 in the translation described in the previous section, they are now connected through the vertices $(q_i, t_{i,l})$ from the two gadgets in Figure 6.

As shown below, this is sufficient to correctly encode every equality constraint.

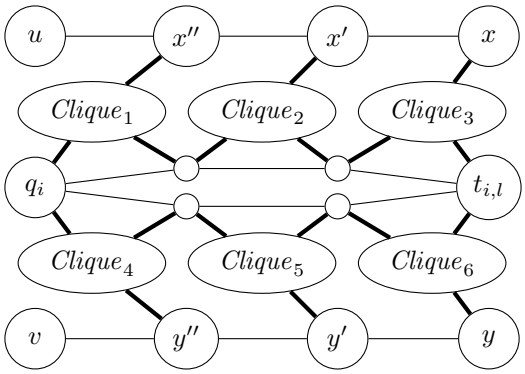

Figure 6: This gadget encodes equality constraints into a graph more efficiently than the gadget from Figure 3. There exists one vertex q_i for every color i, and one vertex $t_{i,l}$ for every color i and symbol l from the alphabet. The gadget is repeated for every possible color i, and the q_i vertices connected to each other in a clique of their own. Although the number of gadgets required per equality constraint is increased, the resulting encoding is more efficient due to the overlap in the created subgraphs: every pair of vertices (v, y) or (u, x) needs to be connected only once to every $(q_i, t_{i,l})$.

Proposition 7 *By replacing every occurrence of v by q_i and y by $t_{i,l}$ for all $0 \leq i < \chi$ in Definition 4, we obtain the construction in Figure 6. Let G_e be the graph resulting from this construction. Then, given a minimal coloring for G_e, for any equality constraint $\langle (u, v), (x, y) \rangle \in E_e$, if the vertices corresponding to u and v are in the same color class, then so are x and y. Furthermore, no other constraints are encoded by the gadget in Figure 6.*

Proof. Due to the clique connecting the vertices q_i, there exists an q_i in C for any color class C. Thus, if u and v are in the same color class C, then there exists a q_i that is in this class as well. By Proposition 6, there exists in G_e a gadget that forces $t_{i,l}$ to be in the same color class C' as x since u and q_i are both in C. Similarly, there exists a gadget that forces $t_{i,l}$ to be in the same color class as y. Clearly, this is only possible if y is also in C'.

It is also straightforward to see that this new gadget does not impose any constraints other than equality. If u and v are in a different class, then they are in the same class with different q_i and q_j, and thus x and y are in the same class as different $t_{i,l}$ and $t_{j,l}$, which can belong to different color classes. Furthermore, since the gadget connects (u, x) with $(q_i, t_{i,l})$ for all i only if there is a transition from u to x in the APTA with label l, no constraints are constructed for pairs of states (u, x) and (v, y) with differently labeled transitions between (u, x) and (v, y). □

The size of the resulting translation is significantly smaller than before since every pair of vertices (u, x) that occurs on one side of an equality constraint for label l now connects through the gadget from Figure 3 to all pairs of vertices $(q_i, t_{i,l})$ for $0 \leq i < \chi$, resulting in $O(\|S\| \cdot \chi)$ gadgets instead of $O(\|S\|^2)$.

5. Learning algorithm

Given the construction sketched in Proposition 7, it is straightforward to come up with a learning algorithm fsuch as the following:

Definition 8 *Let* LEARN(S) *be the following algorithm:*

Require: *Sample* $S = (S_+, S_-)$, CHROM_NR(), COLOR()
  $A :=$ APTA(S_+, S_-)
  $G_c(= (V_c, E_c)) := G_c(A)$ {*consistency graph for* A}
  $upp\_bound := |V_c|$
  $\chi :=$ CHROM_NR(G_c)
  for $i = \chi$ *to* upp_bound ***do***
    $G_e := G_e($APTA$(S), i)$ {*consistency graph with equality constraints for* A *assuming* i *colors*}
    $C :=$ COLOR(G_e, i) {*proper coloring for* G_e *with* i *colors*}
    if C *defined* ***then***
      BREAK
    end if
  end for
  for all c *in* C ***do***
    MERGE *all states in* A *that correspond to vertices in* c
  end for
  compute normal form A' *of* A {*only observable part, no 'reject' labels*}
  return A'

Here, CHROM_NR and COLOR are user-supplied algorithms for determining chromatic number and computing a vertex coloring, respectively. Note that $\chi(G_c)$ may be underestimated without affecting correctness, so even the constant 1 would be acceptable as CHROM_NR. However, for reasons of efficiency it would be wise in practice to use an estimate that is both accurate and easy to compute.

It was shown in [19] that an algorithm that enumerates all DFAs with monotonically increasing size until it finds one consistent with a given sample identifies all DFAs in the limit from complete data. Thus, if we assume CHROM_NR$(G_c) \leq \chi(G_e)$, and we choose the APTA-generating algorithm and COLOR() so that they yield the first automaton in such an enumeration consistent with the sample, we obtain:

Theorem 9 *The algorithm given in Definition 8 solves the regular inference problem, that is, it finds a minimal automaton consistent with given positive and negative data.*

Proof. It is clear that COLOR finds an optimal coloring for G_e. Since there is a one-to-one correspondence between color classes of G_e and states in the hypothesized automaton, the hypothesis is always an automaton of minimal size (w.r.t. the sample). Since G_c is an induced subgraph of G_e, the hypothesis does not violate any inequality constraints and thus accepts all of S_+ and rejects all of S_-. By Proposition 6, the resulting automaton also respects all equality constraints. Thus the hypothesis is always an automaton of minimal size consistent with given positive and negative data. □

Corollary 10 *The algorithm given in Definition 8 identifies in the limit from positive and negative data the class of all deterministic finite state automata.*

It should be clear that our algorithm is consistent, order-independent and set-driven. We leave open the questions of conservative learning and the possibility of an incremental learning algorithm.

6. Bounds

Recall that in Section 4, an upper bound was established on the number of equality constraints of $\|S\| \cdot \chi$. Since the gadget consists of $4 + 3(\chi - 2)$ vertices, in the case that $\chi \geq 3$, we obtain an upper bound of $s \cdot \chi \cdot (4 + 3(\chi - 2)) + s = s \cdot (3\chi^2 - 2\chi + 1)$ vertices in G_e, where $s = \|S\|$ and $\chi = \chi(G_c)$.[6]

Note that this does not necessarily imply that our learning algorithm has quadratic space requirements. Depending on the choice of algorithm for COLOR, it may not be necessary to explicitly represent G_e with, for example, an adjacency matrix. Instead, a representation of G_c could be used, and the additional edges and vertices necessary for representing equality constraints could be computed on the fly just when the coloring algorithm requires them. It will in general be necessary to keep track of the colors assigned to the additional vertices, but for the cliques in the equality subgraphs a representation can be used that requires, for every such clique, only as many bits as $\chi(G_e)$.

The fastest known (exact) vertex coloring algorithm has a time bound of $O(2^v v)$ ([9], v being the number of vertices in G_e), and, given graphs of chromatic number 3 or 4, the tighter bounds of $O(1.3289^v)$ ([6]) and $O(1.7504^v)$ ([11]), respectively. Combined with our bound for the size of G_e, assuming that the algorithm has to iterate from 1 to χ, and assuming CHROM_NR simply yields 1, we obtain the following time bounds ($s = \|S\|$ and $\chi = |A|$):

1. target automaton has 1 state: $f(s)$, with f some polynomial function;
2. target automaton has 2 or 3 states: $O(1.3289^{22s})$;[7]

[6] This bound was first derived in [12], unfortunately, however, its presentation was flawed. What we present here is the corrected version.

[7] It was claimed incorrectly in [12] that in the case of a 2-state automaton, $G - e(S)$ is 2-colorable. Recall that this is not necessarily the case, see the discussion in Subsection 3.1.

3. target automaton has exactly 4 states: $O(1.3289^{22s} + 1.7504^{41s})$;
4. target automaton has 5 or more states:
   $O((\chi - 2) \cdot 2^{s \cdot (3\chi^2 - 2\chi + 1)} \cdot (3\chi^2 - 2\chi + 1))$.

To the casual observer, these bounds may seem very loose. An obvious way to make them tighter would be to consider an algorithm that employs binary search instead of iteration starting from an estimated lower bound. For this to work well, tight estimates are required, but the complexity introduced by the equality constraints, as well as the non-approximability of chromatic number in general, very strongly indicate that these cannot be obtained in reasonable time.

In practice, it might well be that the complexity is overestimated, and good heuristics may lead to an algorithm that performs acceptably on all but a small number of 'difficult' samples.

For *upp_bound*, an obvious candidate would be $\Delta(G)(+1$ if G is a clique), the maximum degree of the vertices in G, which obviously is trivial to compute. An interesting option for the lower bound could be the *Lovász number* of G_c ($\vartheta(G_c)$). This is a real number which forms an upper bound on the Shannon capacity of G, see [28]. Lovász' so-called Sandwich Theorem (see [25] for a discussion), states that $\omega(G) \leq \vartheta(G) \leq \chi(G)$ (here $\omega(G)$ is the clique number of G, i.e. the size of its largest clique, which, like chromatic number, is **NP**-complete to compute). The value of $\vartheta(G)$ can be approximated by the ellipsoid method in time polynomial in $|V|$[21]. Thus it could be a natural candidate for CHROM_NR.

7. From coloring problem to consistent DFA problem

Since coloring is known to be such a hard problem, it could be argued that it may not be a good choice as range of a translation of problem instances. In other words, it cannot a priori be excluded that a heavily optimized coloring algorithm performs much worse than a good DFA learning algorithm applied directly to a sample.

In this section we demonstrate that any coloring can be transformed into a DFA learning problem in linear time. This demonstrates that in some sense, the problems are equally hard, which justifies our choice. Considering that far more research effort has been directed at optimization problems such as coloring than at automata induction, we would expect state-of-the-art coloring algorithms to perform better than state-of-the-art automata induction algorithms.

We define a two-part algorithm. The first part assigns a minimal number of positive and negative labels to the vertices in G so that for any pair $(v_i, v_j) \in E$, $j > i$, one element of the pair has a positive label a and the other a negative label a.

The second part of the algorithm builds a sample based on the vertex labels generated in the first part. This way, a sample is obtained where for each vertex i, there is a word "i l" in the positive (negative) sample if i has l as a positive (negative) label.

Require: Graph $G = (V, E)$, ordered alphabet Σ

Initialize matrices L^+, L^-

for $i = 1$ to $|V|$ **do**

$L_i^+ = \Sigma[i]$

$N := \{v_j | v_j \in N_G(v_i), j > i\}$

for all n in N **do**

push(L_n^-, $\Sigma[i]$)

end for

end for

$S_+ := \emptyset$

$S_- := \emptyset$

for all i in V **do**

for all l in L_i^+ **do**

push(S_+, 'i l')

end for

for all l in L_i^- **do**

push(S_-, 'i l')

end for

end for

$S := (S_+, S_-)$

return S

As an example, consider the graph G in Figure 7. Note that $\chi(G) = 3$.

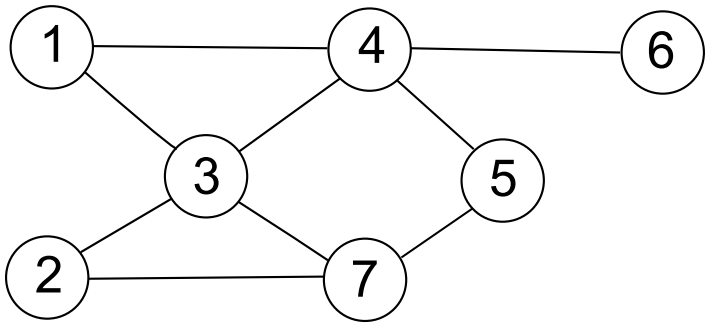

Figure 7: A graph with chromatic number 3.

The algorithm transforms G in a sample S with $\Sigma(S) = \{1, \ldots, 7\} \cup L$, where $L = \{a, b, c, d\}$ (note that $|L| \leq |V|$). We obtain $S_+ = \{4a, 1b, 5c, 2b, 2d, 7b\}$ and $S_- = \{1a, 3a, 3b, 5a, 6a, 7c, 7d\}$.

Consider now APTA(S). As can be seen in Figure 8, the states at level 2 of the APTA can be pairwise

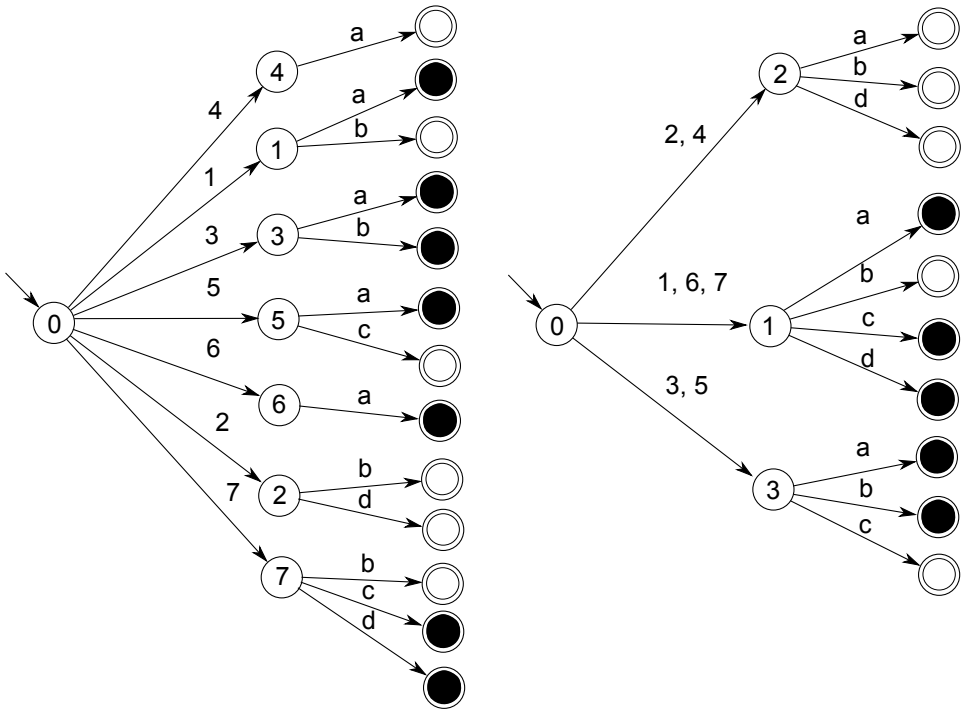

Figure 8: The APTA for the graph in Figure 7 (left), and a DFA obtained by (optimally) merging the states in this APTA at level 2 (right).

merged in such a way that we obtain exactly three states at level 2. Note that the starting state can be merged with any of the other states, and that the leaves can be pairwise merged so that we obtain just one accepting and one rejecting state as leaves. These can then be merged with two distinct states at level 2, and we obtain an automaton with three states. The reader can check that this is actually the minimal size for a consistent DFA, and that it corresponds to an optimal coloring for G: we obtain the three color classes $\{2, 4\}$, $\{3, 5\}$ and $\{1, 6, 7\}$.

It is worth noting that there are no pairs of equality constraints in the APTA such that $\langle (s, t), (u, v)\rangle \in E_e$ and $\langle (u, v), (x, y)\rangle \in E_e$. In other words, merging any pair of states in the APTA may require at most one other pair to be merged in order to obtain a minimal automaton.

Proposition 11 *Given a graph $G = (V, E)$, the instance of the coloring problem for G can be transformed into an instance of the minimal consistent DFA problem for some sample S in time linear in $|G|$, where $|\Sigma(S)| \leq 2|V|$, $\mathrm{len}(w) = 2$ for all $w \in S$, and $|S_+| + |S_-| \leq 2|E|$.*

Note that from the last two statements it follows that $\|S\| \leq 4|E|$.

The number of states of the resulting APTA is $1 + \chi(G) + 2|L|$. From this APTA, an automaton $\mathcal{A}$ can be obtained that has $\chi(G) + 2$ states. Note that the start state can be merged with any of the other states,

and all the accepting (rejecting) states can be merged to a single accepting (rejecting) state.

It can thus be concluded that the minimal consistent DFA problem and the vertex coloring problem have very similar complexity, and that it makes sense to transform instances of the former into instances of the latter.[8]

8. Conclusions

We have shown that instances of the regular inference problem can be transformed to (pure) vertex coloring problem instances of comparable size, and proved correctness of our construction. Furthermore, we have shown that coloring is a rational choice, in the sense that it is a problem of comparable complexity.

We have proposed a learning algorithm based on this new construction, and showed that it identifies in the limit DFAs from positive and negative data. From this, we obtained new complexity bounds and a family of new learning algorithms, some of which require a lot of computational resources in order to obtain exact solutions, while others can quickly yield approximations.

Our approach could be applied or extended in many different ways. We mention here two topics for future research:

semidefinite programming. Algorithms based on semidefinite programming techniques are known that find optimal colorings for perfect graphs in polynomial time. These can often also be used to find approximate colorings for non-perfect graphs in polynomial time. The algorithm discussed in [23], for example, has a hyperparameter which allows the user to obtain solutions anywhere on the spectrum between solutions that use few colors but are not necessarily proper, and proper colorings that may be far removed from a optimal coloring.

The former corresponds with an automaton inconsistent with the sample, the latter with an automaton with more states than the target automaton. This makes a learning algorithm based on such an approach flexible; the user can decide which trade-off is appropriate for the problem at hand by setting the value of this hyperparameter on a case-by-case basis.

[8] It may be worth noting that the coloring problem has appeared before in the machine learning literature. We quote from page 27 of [1]:

> We exhibit a polynomial time transformation from instances of the **NP**-complete Graph-k-colorability (GkC) to finite samples such that there exists a 1-dimensional k-fold semilinear set consistent with the resulting sample, if and only if the original graph is k-colorable. Furthermore, the transformation will be such that whenever there is a k-fold semilinear set consistent with the sample produced by it, there is one whose size is polynomially bounded. A similar transformation is obtained for any dimension d.'

Note that the setting there is efficient PAC-learnability from labeled data. Also, this reduction is in one direction only, from vertex coloring to learning, for the sole purpose of proving **NP**-completeness of a large family of learning problems. It is thus not clear how it could be used to obtain a learning algorithm.

Note that an automaton with too many states indicates a failure of generalization, i.e., $\mathrm{L}(\mathcal{A}_{Target}) - \mathrm{L}(\mathcal{A}_{Hypothesis}) \neq \emptyset$. Note however that the hypothesized automaton may be overly general due to wrong merges, i.e., it may be the case that $\mathrm{L}(\mathcal{A}_{Hypothesis}) - \mathrm{L}(\mathcal{A}_{Target}) \neq \emptyset$.

efficiently learnable classes. Our translations suggests a natural approach to identifying efficiently learnable classes of automata: choose a class of graphs that can be efficiently colored, identify a class of samples that correspond to such graphs, and then identify the class of automata that generate only such samples.

The coloring problem restricted to perfect graphs is known to be in **P** (recognition of this class, however, is **NP**-complete). Note that the gadget used in our reductions contains a number of induced 5-cycles, so no graph with our gadget as induced subgraph cannot be Berge, and thus cannot be perfect. Therefore, except for some trivial cases, the problem of coloring G_e may still be **NP**-complete, even when G_c is perfect. The authors were not able to come up with alternative gadgets without induced 5-cycles, we will leave this as an open problem.

Acknowledgements

The authors wish to thank Henning Fernau for stimulating discussions.

References

[1] N. Abe. Characterizing PAC-learnability of semilinear sets. *Information and Computation*, 116(1):81–102, 1995.

[2] N. Abe and M. K. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9:205–260, 1992.

[3] J. Abela, F. Coste, and S. 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.

[4] H. I. Akram, A. Batard, C. de la Higuera, and C. Eckert. PSMA: A parallel algorithm for learning regular languages. In *NIPS Workshop on Learning on Cores, Clusters and Clouds*, 2010.

[5] D. Angluin. On the complexity of minimum inference of regular sets. *Information and control*, 39(3):337–350, 1978.

[6] R. Beigel and D. Eppstein. 3-coloring in time $O(1.3289^n)$. *Journal of Algorithms*, 54(2):168–204, 2005.

[7] A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications, Handbook of Satisfiability*. IOS Press, February 2009.

[8] A. W. Biermann and J. A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Comput.*, 21(6):592–597, 1972.

[9] A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, July 2009.

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

[11] J. M. Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32(6):547 – 556, 2004.

[12] C. Costa Florêncio and S. Verwer. Regular inference as vertex coloring. In N. H. Bshouty, G. Stoltz, N. Vayatis, and T. Zeugmann, editors, *ALT*, volume 7568 of *Lecture Notes in Computer Science*, pages 81–95. Springer, 2012.

[13] F. Coste and J. Nicolas. Regular inference as a graph coloring problem. In *Workshop on Grammatical Inf., Automata Ind., and Language Acq. (ICML'97)*, 1997.

[14] C. de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.

[15] H. Fernau, P. Heggernes, and Y. Villanger. A multivariate analysis of some DFA problems. In *Proceedings of LATA 2013*, To appear.

[16] P. García, M. V. de Parga, D. López, and J. Ruiz. Learning automata teams. In *ICGI*, volume 6339 of *LNAI*, pages 52–65. Springer-Verlag, 2010.

[17] M. R. Garey and D. S. Johnson. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

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

[20] O. Grinchtein, M. Leucker, and N. Piterman. Inferring network invariants automatically. In *IJCAR*, volume 4130 of *LNCS*, pages 483–497. Springer, 2006.

[21] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981. 10.1007/BF02579273.

[22] M. J. H. Heule and S. Verwer. Exact DFA identification using SAT solvers. In *ICGI*, volume 6339 of *LNCS*, pages 66–79. Springer, 2010.

[23] D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *J. ACM*, 45:246–265, March 1998.

[24] M. J. Kearns and L. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *J. ACM*, 41:67–95, January 1994.

[25] D. E. Knuth. The sandwich theorem. *Electronic Journal of Combinatorics A1*, 1:48, 1994. arXiv:math/9312214.

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

[27] K. J. Lang, B. A. Pearlmutter, and R. 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.

[28] L. Lovász. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25(1):1–7, Jan. 1979.

[29] E. Malaguti and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17(1):1–34, 2010.

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

[31] 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.

[32] L. Pitt and M. K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. In *STOC*, pages 421–432, 1989.

[33] T. A. Sudkamp. *Languages and Machines: an introduction to the theory of computer science*. Addison-Wesley, third edition, 2006.

[34] N. Walkinshaw, B. Lambeau, C. Damas, K. Bogdanov, and P. Dupont. STAMINA: a competition to encourage the development and assessment of software model inference techniques. *Empirical Software Engineering*, pages 1–34, to appear.