Regular inference as vertex coloring

Christophe Costa Florêncio[1] and Sicco Verwer[2⋆]

[1] Department of Computer Science, University of Amsterdam, the Netherlands
[2] 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. This 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 both exact hypotheses, as well as fast approximations.

1 Introduction

The regular inference problem consists of learning (finding) a smallest deterministic finite state automaton (DFA) that is consistent with a given 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 [3, 18], and an inapproximability result was demonstrated in [24]. In spite of these hardness results, quite a few DFA identification algorithms exist, see [11]. In particular, a recently proposed algorithm based on a *translation* of the regular inference problem into satisfiability (SAT) has shown promising results [16].

The translation in [16] is based on an earlier translation of regular inference to graph coloring in [10]. Graph coloring is the problem of assigning a color to every node in a given graph such that nodes with the same color do not share an edge. Determining whether there exists a coloring that uses at most $k \geq 3$

⋆ The second authos is supported by STW project 11763 Integrating Testing And Learning of Interface Automata (ITALIA).

colors is a well-known **NP**-complete problem, see, e.g., [13]. The main idea of this translation into graph coloring is to use a distinct color for every state of the learned DFA. The nodes in the graph coloring instance represent the labeled strings and share an edge if one of them is positive and the other negative. The graph coloring problem thus ensures that pairs of positive and negative examples cannot obtain the same color, and therefore cannot end in the same state, making the resulting DFA consistent. The size of this DFA is determined by the amount of colors used in the graph coloring problem. Finding the minimum is done by iterating over this amount.

The above mentioned reduction from [10], however, was not purely based on graph coloring. In addition to the *inequality constraints*, denoting that two vertices cannot be assigned the same color, so-called *equality constraints* are needed to model regular inference. These constraints denote that two vertices should be assigned the same color if two other vertices are assigned the same color. Together, the equality and inequality constraints can efficiently encode the regular inference problem. Unfortunately, however, it has remained unknown how to encode such constraints in a graph coloring problem instance. In [10], they were encoded dynamically by creating new graph coloring instances that satisfied them on-the-fly. In [16], they were encoded directly into satisfiability instead of in the intermediary graph coloring instance. In this paper, we develop the first construction that encodes them directly into graph coloring.

In terms of complexity (size), our encoding of the equality constraints is comparable to the encoding to satisfiability described in [16]: 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, are much easier to encode in graph coloring, 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 [16]) 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 graph coloring, including techniques for symmetry-breaking, many local-search based approaches, cutting-plane algorithms, etc. see, e.g., [21].

2 Background and notation

2.1 Regular inference

A *deterministic finite state automaton* (DFA) is one of the basic and most commonly used finite state machines. Below, we provide a concise description of DFAs, the reader is referred to [25] for a more elaborate overview. A DFA $A = \langle Q, T, \Sigma, q_0, F \rangle$ is a directed graph consisting of a set of *states* Q (nodes) 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 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, it rejects all others. A 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 A is called the *language* $L(A)$ of A.

Given a pair of finite sets of positive example strings S_+ and negative example strings S_-, called the *input sample*, the goal of *regular inference* (or *DFA identification/learning*) is to find a (non-unique) *smallest* DFA A that is *consistent* with $S = \{S_+, S_-\}$, i.e., such that every string in S_+ is accepted by A, and every string in S_- is rejected by A. Typically, the size of a DFA is measured by the number of states it contains. Seeking this DFA is an active research topic in the grammatical inference community, see, e.g., [11].

For many years, the state-of-the-art in DFA identification has been the *evidence-driven state-merging* (EDSM) algorithm [20]. 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., [11]. 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 [23]. EDSM participated in and won (in a tie) the Abbadingo DFA learning competition in 1997 [20].

Since our method is based on the simple yet effective state-merging approach, we now briefly explain this approach. For more information, the reader is referred to [11]. The key idea of state-merging is to first construct a tree-shaped DFA A from the input sample S, and then to merge (combine) the states of A. This initial DFA A is called an *augmented prefix tree acceptor* (APTA). An example is shown in Figure 1.

Definition 1. *The* APTA $A = (\langle Q, T, \Sigma, q_0, F \rangle, R)$ *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* $R = S_-$*.*

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 A. 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. 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 continued 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'

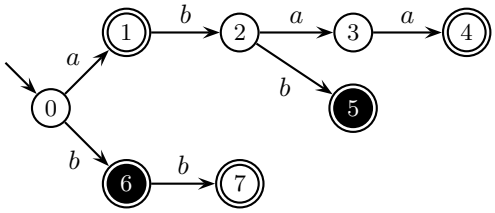

Fig. 1. An augmented prefix tree acceptor for $S = (S_+ = \{a, abaa, bb\}, S_- = \{abb, b\})$. The start state is the state with an arrow pointing to it from nowhere.

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.

In the grammatical inference community, there has been some research into developing advanced and efficient search techniques based on the EDSM heuristic. 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 [22], using mutually (in)compatible merges [1], and searching most-constrained nodes first [19]. A comparison of different search techniques for EDSM can be found in [8]. 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 [16]. 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., [5]. The winning contribution to the 2010 Stamina DFA learning competition was a combination of this SAT-based approach and EDSM with a modified heuristic [26]. Other recently proposed improvements are the parallelization of the algorithm [2], and the use of ensembles of learned DFAs [12].

2.2 Translating regular inference

The idea of translating the regular inference problem to other computational problems for which dedicated solvers exist is not new. In fact, one of the earliest regular inference algorithms due to Biermann [6] is of this type. Biermann proposed to solve the regular inference problem by mapping it to constraint satisfaction. In this translation, every state is represented by a natural number, constraints on the possible values of states are added that enforce consistency, and the aim is to minimize the range of these numbers, which translates back to minimizing the number of states in the resulting DFA. More recently, Grinchtein et al. [15] adapted this translation in order to map regular inference to satisfiability (SAT) instead of constraint satisfaction. The numeric constraints from

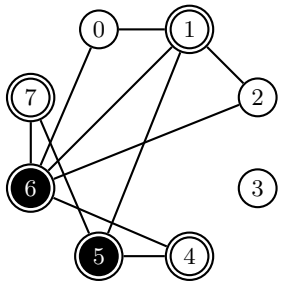

Fig. 2. The consistency graph corresponding to the APTA of Figure 1. Some states in the consistency graph are not directly inconsistent, but inconsistent due to determinization. For instance states 2 and 6 are inconsistent because the strings *abb* (negative) and *bb* (positive) would end in the same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

the constraint satisfaction problem are encoded using either a unary or a binary scheme into clauses and literals for the satisfiability problem.

Another type of translation is that of Coste [10], who maps regular inference to graph 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 node in the graph coloring problem corresponds to 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 in the original regular inference problem:

Definition 2. *The* consistency graph $G_c = (V, E_c)$ *for an APTA* $(\langle Q, T, \Sigma, q_0, F\rangle, R)$ *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. 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 too (encoding the merging for determinization procedure).

Definition 3. *The set of* equality constraints E_e *for an APTA* $A = (\langle Q, T, \Sigma, q_0, F\rangle, R)$ *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* $l \in \Sigma$.

For graph coloring problem, these equality constraints encode that two parent states a and b can get the same color only if their child states al and bl get the same color. Until now it has been unclear how to encode such constraints in a graph coloring problem instance. In [10], these were encoded 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. This is clearly not very efficient. Thirteen years later, this graph coloring

encoding in [10] was used by Heule and Verwer as a basis for a more efficient translation to satisfiability [16], which encodes the equality constraints directly.

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

2.3 Graph coloring

We briefly discuss graph 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 ≤ 2

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

[3] As an aside, it should be noted that $\chi(G_c(S)) = 1$ does *not* 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.

graph with their distance to v (this 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 $\chi(G_e) > 2$.

3.2 Graphs with chromatic number ≥ 3

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

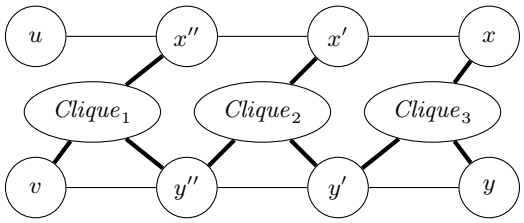

Fig. 3. This gadget encodes equality constraints into a graph. A thick line represents a set of edges that connect a vertex (circle) to all vertices in a clique (ellipse).

This construction is formally defined as follows:

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 $Clique_2$;*

[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. *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:

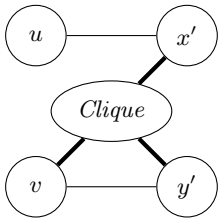

Fig. 4. The subgraph discussed in Lemma 1.

Lemma 1. *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 from Figure 4, with its clique 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*. 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 a color c_v from $C - C_1$. Since y' is connected to v and to all vertices in *Clique*, it has 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 $x' =_c y'$. 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'. □

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

Proposition 1. *Let $G_c = G_c(S)$, and $\chi(G_c) \geq 3$. Let $G_e = G_e(G_c)$ (as given in Definition 4).*

Then, given an optimal coloring for G_e, for any equality constraint $e = \langle (u, v), (x, y) \rangle \in E(G_c)$, 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 aree 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 not necessarily;
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 1 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 1 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 1 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. □

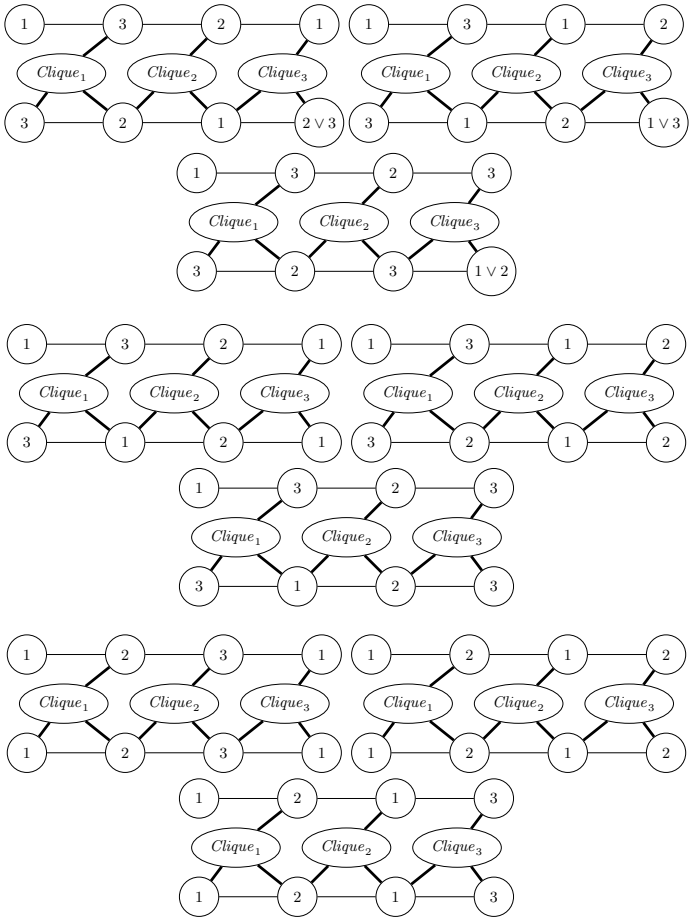

Fig. 5. Possible colorings for the $\chi(G_c) = 3$ construction.

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 [16], a similar problem was observed for a translation of regular inference to satisfiability. There, it was solved by introducing additional variables that encode the equality constraints globally, i.e., for the resulting automaton model instead of per pair of APTA states. Below, we show that such a global encoding is also possible for our translation to graph coloring and that it reduces this quadratic relation to a linear one.

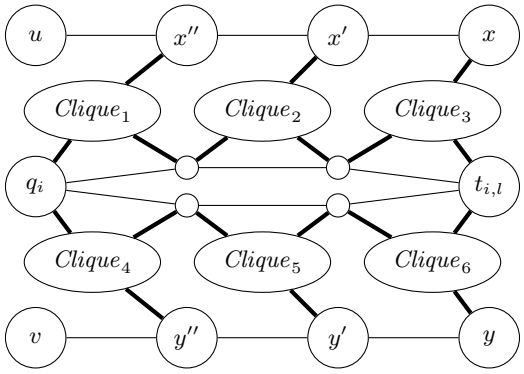

Fig. 6. This gadget encodes equality constraints into a graph more efficiently than the gadget from Figure 3. There exists one node q_i for every color i, and one node $t_{i,l}$ for every color i and symbol l from the alphabet. The gadget is repeated for every possible color i, and the q_i nodes connected 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 nodes (v, y) or (u, x) needs to be connected only once to every $(q_i, t_{i,l})$.

The key idea is to introduce two additional sets of nodes 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.

Proposition 2. *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 nodes 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 1, 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 nodes (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 nodes $(q_i, t_{i,l})$ for $0 \leq i < \chi$, resulting in $O(\|S\| \cdot \chi)$ gadgets instead of $O(\|S\|^2)$.

5 Learning algorithm

Definition 5. *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***
    $G_e := G_e(\text{APTA}(S), i)$ {*consistency graph with equality constraints for* A *assuming* i *colors*}
    $C = \text{COLOR}(G_e, i)$ {*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-specified algorithms for determining chromatic number and computing a vertex coloring, respectively. Note that $\chi(G_c)$ may be underestimated without affecting correctness so simply the constant 1 would be acceptable as CHROM_NR.

It was shown in [14] that an algorithm that enumerates all DFAs with monotonically increasing size until it finds one consistent with a sample, identifies all DFAs in the limit. Thus, if we assume $\text{CHROM\_NR}(G_c) \leq \chi(G_e)$, 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 1. *The algorithm given in Definition 5 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 a 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 1, 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 1. *The algorithm given in Definition 5 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 we established an upper bound on the number of equality constraints of $\|S\| \cdot \chi$ (Section 4). 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 $\chi = \chi(G_c)$.

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 $O(2^v v)$ ([7], v being the number of vertices in G_e), and, given graphs of chromatic number 3 or 4, the tighter bounds of $O(1.3289^v)$ ([4]) and $O(1.7504^v)$ ([9]), 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 or 2 states: $f(s)$, with f some polynomial function;
2. target automaton has exactly 3 states: $O(1.3289^{22s})$;
3. target automaton has exactly 4 states: $O(1.3289^{22s} + 1.7504^{41s})$;
4. target automaton has 5 or more states:
   $O((\chi - 2) \cdot 2^{s \cdot (3\chi^2 + 2\chi + 1)} \cdot (3\chi^2 + 2\chi + 1))$.

7 Discussion

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 [17], 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.

References

1. 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.
2. 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.
3. D. Angluin. On the complexity of minimum inference of regular sets. *Information and control*, 39(3):337–350, 1978.
4. R. Beigel and D. Eppstein. 3-coloring in time $O(1.3289^n)$. *Journal of Algorithms*, 54(2):168–204, 2005.

5. 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.
6. 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.
7. A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, July 2009.
8. M. Bugalho and A. L. Oliveira. Inference of regular languages using state merging algorithms with search. *Pattern Recognition*, 38:1457–1467, 2005.
9. J. M. Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32(6):547 – 556, 2004.
10. 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.
11. C. de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.
12. 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.
13. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.
14. E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, June 1978.
15. O. Grinchtein, M. Leucker, and N. Piterman. Inferring network invariants automatically. In *IJCAR*, volume 4130 of *LNCS*, pages 483–497. Springer, 2006.
16. M. J. H. Heule and S. Verwer. Exact DFA identification using SAT solvers. In *ICGI*, volume 6339 of *LNCS*, pages 66–79. Springer, 2010.
17. D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *J. ACM*, 45:246–265, March 1998.
18. M. J. Kearns and L. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *J. ACM*, 41:67–95, January 1994.
19. K. J. Lang. Faster algorithms for finding minimal consistent DFAs. Technical report, NEC Research Institute, 1999.
20. 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.
21. E. Malaguti and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17(1):1–34, 2010.
22. 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.
23. 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.
24. L. Pitt and M. K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. In *STOC*, pages 421–432, 1989.
25. T. A. Sudkamp. *Languages and Machines: an introduction to the theory of computer science*. Addison-Wesley, third edition, 2006.
26. 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.