Efficient Search Techniques for the Inference of Minimum Size Finite Automata

Arlindo L. Oliveira
Cadence European Laboratories
IST-INESC
Lisbon, Portugal
aml@inesc.pt

João P. M. Silva
Cadence European Laboratories
IST-INESC
Lisbon, Portugal
jpms@inesc.pt

Abstract

We propose a new algorithm for the inference of the minimum size deterministic automaton consistent with a prespecified set of input/output strings. Our approach improves a well known search algorithm proposed by Bierman, by incorporating a set of techniques known as dependency directed backtracking. These techniques have already been used in other applications, but we are the first to apply them to this problem. The results show that the application of these techniques yields an algorithm that is, for the problems studied, orders of magnitude faster than existing approaches.

1. Introduction and Related Work

This work addresses the problem of inferring a finite automaton with minimal complexity that matches a given set of input/output pairs. This problem has been extensively studied in the literature, both from a practical and theoretical point of view and it has many applications, from machine learning [13] to logic synthesis and digital circuit design [14].

Selecting the minimum DFA consistent with a set of labeled strings is known to be NP-complete. Specifically, Gold [10] proved that given a finite alphabet Σ, two finite subsets $S, T \subseteq \Sigma^*$ and an integer k, determining if there is a k-state DFA that recognizes L such that $S \subset L$ and $T \subset \Sigma^* - L$ is NP-complete. Furthermore, it is known that even finding a DFA with a number of states polynomial on the number of states of the minimum solution is NP-complete [17].

If *all* strings of length n or less are given (a *uniform-complete* sample), then the problem can be solved in time polynomial on the input size [11, 18, 24]. Note, however, that the size of the input is in itself exponential on the number of states in the resulting DFA. Angluin has shown that even if an arbitrarily small fixed fraction $(|\Sigma^{(n)}|)^\epsilon$, $\epsilon > 0$ is missing, the problem remains NP-complete [1].

The problem becomes easier if the algorithm is allowed to make queries or experiment with the unknown automaton. Angluin [2] proposes an algorithm based on the approach described by Gold [9] that solves the problem in polynomial time by allowing the algorithm to ask membership queries. Schapire [20] proposes an interesting approach that does not require the availability of a reset signal to take the automaton to a known state.

All these algorithms address simpler versions of the problem discussed here. Here, we assume the learner is given a set of labeled strings and is not allowed to make queries or experiment with the machine. The basic search algorithm for this problem was proposed by Bierman [3]. Later, the same author proposed an improved search strategy that is much more efficient in the majority of the complex problems [4]. Section 3 describes these algorithms in detail. More recently, the applicability of implicit enumeration techniques to this problems was studied [15], but these techniques have not shown any significant advantages when compared with the basic search techniques proposed by Bierman.

The central contribution of this paper is the application of advanced search techniques to the inference problem stated above. More specifically, we show how the application of dependency-directed backtracking techniques improves significantly the search algorithm proposed by Bierman. These techniques, applied to date in other domains like truth maintenance systems and boolean satisfiability solvers [21, 23], allow the search algorithm to prune large sections of the search tree by diagnosing the ultimate causes of conflicts encountered during the search. In many cases, these conflicts are caused by assignments that were made several levels above and significant parts of the search tree can be removed from consideration.

These techniques were implemented in a program, BIC.

The approach was evaluated by comparing this program with alternative approaches in sets of problems with known solutions. The results presented in section 5 show that these techniques are indeed effective in extending the scope of applicability of the search techniques and make the algorithms able to handle many problems of non-trivial size.

A different approach is to view the problem of selecting the minimum automaton consistent with a set of strings as equivalent to the problem of reducing an incompletely specified finite automaton. This problem is more general than the one addressed here and was also proved to be NP-complete by Pfleeger [16], but previous work [15] has shown that these algorithms are extremely inefficient when applied to this problem, and present no advantages over the approach presented here.

2. Definitions

2.1. Basic definitions

We follow Gold's notation [10], and we use the following definition of finite state automata[1]:

Definition 1 *A deterministic finite automaton (DFA) is a Mealy model finite state automaton represented by the tuple $M = \big(\Sigma, \Delta, Q, q_0, \delta, \lambda\big)$ where $\Sigma \neq \emptyset$ is a finite set of input symbols, $\Delta \neq \emptyset$ is a finite set of output symbols, $Q \neq \emptyset$ is a finite set of states, $q_0 \in Q$ is the initial "reset" state, $\delta(q,a) : Q \times \Sigma \rightarrow Q \cup \{\phi\}$ is the transition function, and $\lambda(q,a) : Q \times \Sigma \rightarrow \Delta \cup \{\epsilon\}$ is the output function.*

We will assume that $Q = \{q_0, q_1, ...q_n\}$ and will use will use $q \in Q$ to denote a particular state, $a \in \Sigma$ a particular input symbol and $b \in \Delta$ a particular output symbol. An automaton is incompletely specified if the destination or the output of some transition is not specified. When referring to incompletely specified automata, we will use ϕ to denote an unspecified transition and ϵ to denote an unspecified output. The function $\delta(q,a)$ defines the structure of the state transition graph of the automaton while the function $\lambda(q,a)$ defines the labels present in each of the edges of that graph.

We say that an output b_i is compatible with an output b_j (and write $b_i \equiv b_j$) if $b_i = b_j$ or $b_i = \epsilon$ or $b_j = \epsilon$.

The domain of the second variable of functions λ and δ is extended to strings of any length in the usual way:

Definition 2 *The output of a sequence $s = (a_1, \ldots, a_k)$ applied to state q, denoted by $\lambda(q,s)$, represents the output of an automaton after a sequence of inputs $(a_1, \ldots, a_k)$, is applied in state q. The output of such a sequence is defined to be*

$$\lambda(q,s) = \lambda\big(\delta(\delta(\cdots\delta(q,a_1)\cdots),a_{k-1}),a_k\big) \qquad (1)$$

Definition 3 *The destination state of a sequence $s = (a_1, \ldots, a_k)$, denoted by $\delta(q,s)$, represents the final state reached by an automaton after a sequence of inputs $(a_1, \ldots, a_k)$, is applied in state q. This state is defined to be*

$$\delta(q,s) = \delta(\delta(\ldots\delta(\delta(q,a_1),a_2)\ldots),a_k) \qquad (2)$$

To avoid unnecessary notational complexities, $\lambda(\phi,a) = \epsilon$ and $\delta(\phi,a) = \phi$, by definition.

2.2. IO Mappings and Loop Free Automata

The objective is to infer an automaton with minimum number of states that is consistent with a given input/output mapping. An input/output mapping is specified by one or more sequences of input-output pairs:

Definition 4 *An input/output mapping is a set of pairs $T = \{(s_1,l_1), \ldots, (s_m,l_m)\}$ where each pair $(s,l) \in \Sigma \times \{\Delta \cup \epsilon\}$ represents one input string and the output observed for that string.*

If the output alphabet is the set $\{0,1\}$ the input/output mapping can be viewed as specifying a set of accepted strings (the ones that output 1) and a set of rejected strings (the ones that output 0). An example of a possible input/output mapping is given in figure 1.

Accepted:	1	11	1111	11111		
Rejected:	0	101	01	00	011	11110

Figure 1. IO mapping specified as a set of accepted and rejected strings.

Alternatively, the input/output mapping can be specified by one or more sequences where, at each time, the value of the input/output pair is known. Both forms of input/output mapping descriptions are equivalent and can be viewed as defining a particular type of incompletely specified automata, a *Loop Free Automata* (LFDFA). A LFDFA is a DFA that has a state transition graph without loops or re-convergent paths. Figure 2 shows the LFDFA that corresponds to the input/output mapping in figure 1.

There is a one to one correspondence between loop free automata and input/output mappings. A loop free automaton represents a input/output mapping if

[1]We use the less commonly used formalism of Mealy automata instead of Moore automata, since applications in digital circuit design always use Mealy implementations, for efficiency reasons.

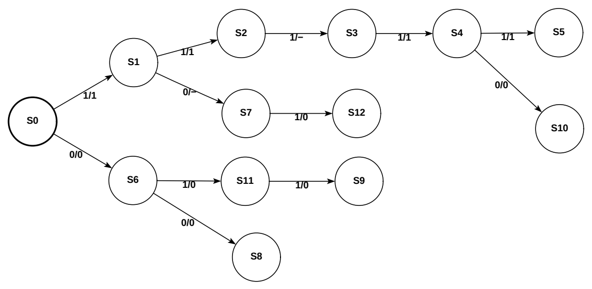

Figure 2. Loop free automaton for the input/output mapping in figure 1

.

1. Its output for each input sequence present in the input/output mapping agrees with the label in that input/output mapping.
2. The output for input sequences not present in the input/output mapping is undefined.

Formally,

Definition 5 *An automaton is said to be the loop free automaton representing a input/output mapping T if it satisfies definition 1 and the following additional requirements:*

$$\forall q \in Q \setminus q_0,\ \exists^1 (q_i, a) \in Q \times \Sigma\ s.t.\ \delta(q_i, a) = q$$

$$\forall q \in Q,\ \forall a \in \Sigma\ \delta(q, a) \neq q_0$$

$$\lambda(q_0, s_i) = l_i \text{ if } (s_i, l_i) \in T,$$

$$\lambda(q_0, s) = \epsilon \text{ if } (s, l) \notin T$$

Since the input/output mapping T defines uniquely the corresponding loop-free automaton, we will use T to denote both the input/output mapping itself and the LFDFA that it defines. We will in general use quoted symbols (Q', δ', etc) to refer to the loop free automata and unquoted symbols to refer to the resulting completely specified automata.

The aim is to construct an automaton M that exhibits a behavior equivalent to T, that is, an automaton M that outputs the same output as T every time this output is defined.

Definition 6 *An automaton $M = \left(\Sigma, \Delta, Q, q_0, \delta, \lambda\right)$ is equivalent with a LFDFA $T = \left(\Sigma, \Delta, Q', q'_0, \delta', \lambda'\right)$ if, for any input string $s = (a_1, \ldots, a_k)$ $\lambda(q_0, s) \equiv \lambda'(q'_0, s)$.*

Given a specific mapping function $F : Q' \to Q$ with $F(q'_0) = q_0$ from the states in T to the states in M, it defines a valid solution iff it satisfies the following two requirements:

Definition 7 *A function F satisfies the output and transition requirements iff:*

$$\forall q = F(q'),\ \lambda'(q', a) \equiv \lambda(q, a) \tag{3}$$

$$\forall q = F(q'),\ F(\delta'(q', a)) = \delta(q, a) \tag{4}$$

It is known that the minimum finite state automaton that satisfies the input/output mapping can be found by selecting an appropriate mapping function. That result is stated in a formal way in the following theorem:

Theorem 1 *Any automaton $M = (\Sigma, \Delta, Q, q_0, \delta, \lambda)$ consistent with the loop free finite state automaton $T = (\Sigma, \Delta, Q', q'_0, \delta', \lambda')$ can be obtained by selecting an appropriate mapping function $F : Q' \to Q$, $F(q'_0) = q_0$ that verifies $\lambda'(q', a) = \lambda(F(q'), a)$ and $F(\delta'(q', a)) = \delta(q, a)$.*

Proof: Let $F : Q' \to Q$ be defined by $F(\delta'(q'_0, s)) = \delta(q_0, s)$ for each string s such that $\delta'(q'_0, s)) \neq \phi$. Since M is equivalent to T, it gives an output compatible with T, for every string s applied at the reset state, i.e., $\lambda(q_0, s) \equiv \lambda'(q'_0, s)$. Consider two states $q' \in Q'$ and $q \in Q$ and assume that $F(q') = q$ because $\delta'(q'_0, s) = q'$ and $\delta(q_0, s) = q$. Now, $\lambda'(q', a) = \lambda'(q'_0, sa) \equiv \lambda(q_0, sa) = \lambda(\delta(q_0, s), a) = \lambda(q, a)$, and therefore equation 3 is respected. On the other hand, $F(\delta'(q', a)) = F(\delta'(q'_o, sa)) = \delta(q_0, sa) = \delta(\delta(q_0, s), a) = \delta(q, a) = \delta(F(q'), a)$ and therefore equation 4 is also respected. Therefore, F is a mapping function.
$\square$

This result has been implicitly used by Bierman [4] and a slightly different proof has been presented in [15]. Is is important to note that, in general, the minimum DFA equivalent to a given incompletely specified DFA can **not** be obtained by selecting a mapping function in this way. This result is only valid for LFDFAs.

Since any automaton that satisfies these requirements can be found by selecting a mapping function, the objective of selecting the minimum consistent DFA can be attained by selecting a mapping function that exhibits a range of minimum cardinality.

For the sake of simplicity, and following the notation of other authors [4] we will define S_i as the index of the state in the target automaton that state q'_i in the original LFDFA maps to, i.e., $q_{S_i} = F(q'_i)$.

An equivalent DFA with N states can therefore be found by selecting an assignment to the variables $S_0, \ldots S_n$, such that each S_i is assigned a value between 0 and $N - 1$, and this assignment defines a mapping function that satisfies (3) and (4).

2.3. Compatible and Incompatible States

Two states q'_i and q'_j in a finite state automaton T are incompatible if, for some input string s, $\lambda(q'_i, s) \not\equiv \lambda(q'_j, s)$. This information can be represented by a graph, the *incompatibility graph*. The nodes in this graph are the states in Q', and there is an edge between state q'_i and q'_j if these states are incompatible.

The incompatibility graph is represented by a function $I : Q' \times Q' \rightarrow \{1, 0\}$. $I(q'_i, q'_j)$ is 1 if and only if states q'_i and q'_j are incompatible.

A clique in the incompatibility graph gives a lower bound on the size of the minimum automaton. By definition, pairs of incompatible states cannot be mapped to the same state and therefore, a clique in this graph corresponds to a group of states that must map to different states in the resulting automaton. Identifying the largest clique in a graph is in itself an NP-complete problem [8]. A large clique (not necessarily the maximum one) can be identified using a slightly modified version of an exact algorithm proposed by Carraghan and Pardalos [5]. The size of the clique provides a lower bound on the number of states needed in the resulting automaton. This lower bound is used as the starting point for the search algorithms described in the next section. If the algorithms described in the next section fail to find an automaton with a number of states equal to the lower bound, we increase N by one and re-execute the algorithms.

3. The Search Algorithm

As shown in the previous sections, the objective is to select a mapping function F that has a range of minimum cardinality.

The constraints that need to be obeyed by the mapping function can be restated as follows:

1. If two states q'_i and q'_j in the original LFDFA are incompatible, then $S_i \neq S_j$.

2. If two states q'_i and q'_j have successor states q'_k and q'_l for some input u, respectively, then $S_i = S_j \Rightarrow S_k = S_l$.

These two conditions can be rewritten as:

$$I(q'_i, q'_j) = 1 \quad \Rightarrow \quad S_i \neq S_j \tag{5}$$

and

$$q'_k = \delta'(q'_i, a) \wedge q'_l = \delta'(q'_j, a) \quad \Rightarrow \quad S_i \neq S_j \ \vee S_k = S_l \tag{6}$$

Assume, for the moment, that a search is being performed for an automaton with N states. The basic search with backtrack procedure iterates through the following steps:

1. Select the next variable to be assigned, S, from among the unassigned variables S_i.

2. Extend the current assignment by selecting a value from the range $0 \ldots N-1$ and assigning it to S. If no more values exist, undo the assignment made to the last variable chosen.

3. If the current assignment leads to a contradiction, undo it and goto step 2. Else goto step 1.

This search process can be viewed as a search tree (or decision tree), and we define the decision level as the level in this tree where a given assignment was made. The assignment made at the root is at decision level 0, the second assignment at decision level 1, and so on.

To illustrate the fundamental differences between this and succeeding search techniques, consider an hypothetical example where a search is being performed by a automaton with 3 states. Under these conditions, each S_i can assume only the values 0, 1 or 2. Suppose that variables will be assigned in the order $S_1 \ldots S_9$ and that the following restrictions exist in this problem:

$$S_1 \neq S_2 \vee S_8 = S_9 \tag{7}$$

$$S_8 \neq S_9 \vee S_2 = S_3 \tag{8}$$

The section of the search tree depicted in figure 3 is obtained by the basic search algorithm described above. In every leaf of this tree a conflict was detected and backtracking took place.

Bierman noted [4] that a more effective search strategy can be applied if some bookkeeping information is kept and used to avoid assigning values to variables that will later prove to generate a conflict. This bookkeeping information can also be used to identify variables that have only one possible assignment left, and should therefore be chosen next. This procedure can be viewed as a generalization to the multi-valued domain of the unit clause resolution of the Davis-Putnam procedure [6], and can be very effective in the reduction of the search space that needs to be explored.

This can be done in the following way [2]

- For each node in $q'_j \in Q'$, a table is kept that lists the values that S_j can take.

[2]The original formulation is made in slightly different terms. We present here an adapted description of Bierman's algorithm, suited to follow our different notation. The interested reader is referred to the reference for the original formulation.

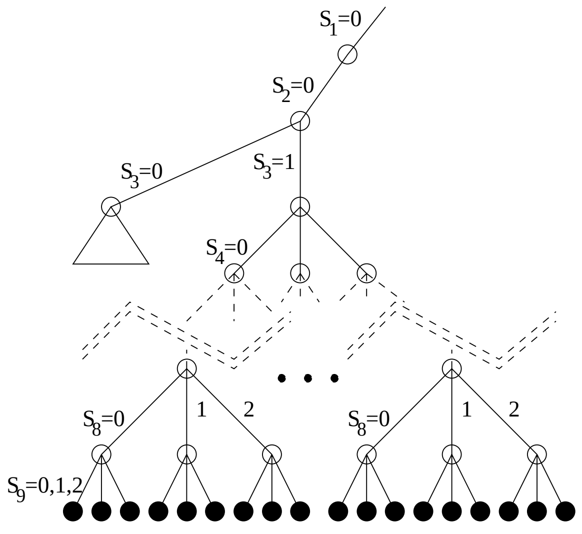

Figure 3. Search tree for simple backtrack algorithm

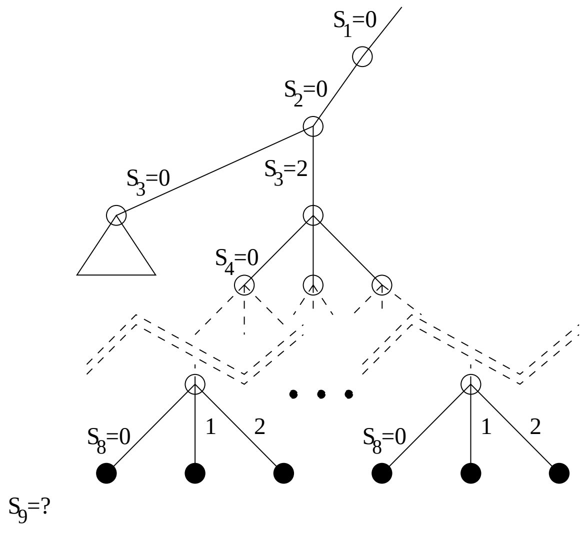

Figure 4. Reduced search tree for Bierman's improved algorithm

- Every time some S_i is assigned to the value z, the tables for all unassigned nodes are updated according to the following algorithm:
  - If $I(q'_i, q'_j) = 1$, then z is removed from the list of values that S_j can be assigned to. (Equation (5))
  - If the assignment of S_i forces some specific value z on some node q'_l (forced by equation (6)), then all values except z are removed from the table of possible values in node q'_l.
- When selecting the next variable S_i to assign, priority is given to the variables that correspond to nodes in Q' that can at that depth take only one possible value.

Clearly, the information on these tables needs to be updated after every assignment is made to some S_i and also after each backtrack takes place. Bierman has shown, and our experiments have confirmed, that for hard problems, this improved search strategy leads to considerable improvements in speed, and expands the size of the problems that can be addressed.

For the example considered above, the section of the search tree obtained is shown in figure 4. Note that as soon as a value is assigned to S_8, the algorithm automatically identifies that no solution exists because no value can possibly be assigned to S_9. This may lead to very considerable savings, specially if S_9 is not the next variable to be assigned.

4. Improving the Search Algorithm

The major contribution of this work is to show that the application of recently developed search techniques can improve considerably the efficiency of the search algorithm and extend the range of problems that can be effectively solved.

The improvement proposed is the application of *conflict diagnosis* techniques to allow for the use of *dependency directed backtracking*. In this paper, we use the term *dependency directed backtracking* to denote two techniques that can, in fact, be used independently. The first technique is based on the realization that, under certain conditions, it is possible to perform jumps in the decision tree that span one or more decision levels. These jumps are called *non-chronological backtracks* or *backjumps* [19]. The second technique is due to the fact that, under similar conditions, it is possible to assert a set of conditions that need to be obeyed in the future if a solution is to be found.

Before we describe these techniques, we will first reformulate slightly the problem at hand. For this, we point out that the set of restrictions (5) and (6) can be computed only once at the beginning of the algorithm execution. This means that a set of restrictions is generated, where each restriction is of the form:

$$(X_1 \,\mathrm{op}\, X_2) \vee (X_3 \,\mathrm{op}\, X_4) \vee \ldots \vee (X_{n-1} \,\mathrm{op}\, X_{i_n}) \quad (9)$$

Each restriction is a disjunction of one or more elements of the form $X_i \mathrm{op} X_{i+1}$ where each X_i is either a variable

S_j or a constant in the range 0 to $N-1$. Each operator op is either $=$ or $\neq$. Restrictions generated from equation (5) have one element, while restrictions generated from equation (6) have 2 elements. It is clear that the algorithm can be easily extended to satisfy any restrictions with more than 2 elements, but the problem formulation does not create them.

Conflict diagnosis [22] is a technique first proposed by Stallman and used in the context of truth-maintenance systems [7, 23] and constraint satisfiability solvers. To make clear how conflict diagnosis can be used to prune the search tree, consider again the example described above, and the section of the search tree shown in figure 5. In this problem,

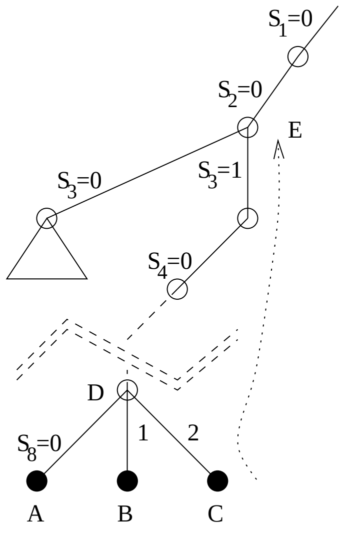

Figure 5. Example of non-chronological backtracking used by the improved search algorithm (BIC)

for each possible value assigned to S_8, a conflict is detected as soon as all the values of S_9 are considered. In this example the conflicts detected in the backtrack points A, B and C can be traced to the following conditions: a) $S_1 = S_2$ and b) $S_2 \neq S_3$. Under these conditions, all the possible assignments to S_8 and S_9 failed to yield a solution, and therefore no solution can be found until one of these assignments is removed. This fact has two immediate consequences:

1. No progress can be made by assigning other values to S_4, S_5 up to S_7, until at least one of specific assignments made to S_1, S_2 and S_3 are changed.

2. If the conditions that caused this conflict, namely the specific assignments made to S_1, S_2 and S_3, are again present in some other part of the search tree, they will generate a replica of this conflict.

Therefore, we can state that the current assignments to S_1, S_2 and S_3 can not be present in any assignments that lead to a solution. This means that we can perform a non-chronological backtrack to the decision level where S_3 was assigned and that a new restriction can be added to the restrictions in the database.

These facts lead to the following general procedure for handling conflicts and controlling the backtrack search procedure.

1. Every time a conflict is detected, diagnose the conflict and generate a restriction that expresses that conflict. The result of this diagnosis is the consensus of all the assignments that originated the conflict.

2. Identify the variable present in the conflict that is at the highest decision level, and perform a *non-chronological backtrack* to that level.

3. Store the restriction generated by the conflict diagnosis engine in the restriction database, and use it to restrict choices of variables in the future. This is sometimes described as *dependency directed backtracking*.

4.1. Conflict diagnosis

Every time a conflict arises, the assignments that are directly responsible for the conflict are identified. The conditions that need to be obeyed to resolve this conflict can be summarized in a restriction of the form of the general restriction (9). Consider again the example in figure 5, and the conflicts detected in nodes A, B and C. At each of these nodes, it is found that no possible assignments exist to variable S_9.

Analyzing in more detail the conflict in node A, we see that S_9 cannot take the value 1 because the second restriction in the example, restriction (8) is not satisfied, given the current assignments $S_2 = 0$ and $S_3 = 1$. The other values possible for S_9, 2 or 3, are barred because restriction (7) would not be satisfied, since $S_1 = 0$ and $S_2 = 0$. The condition that leads to this conflict can therefore be written as:

$$S_1 = 0 \wedge S_2 = 0 \wedge S_3 = 1 \wedge S_8 = 0 \tag{10}$$

This condition represents simply the consensus of all the restrictions that lead to the conflict in this node. The last element in the restriction, $S_8 = 1$ is also a cause of the conflict, and therefore should be listed in the condition.

For nodes B and C, a similar procedure could be followed and we would arrive at the following two conditions:

$$S_1 = 0 \wedge S_2 = 0 \wedge S_3 = 1 \wedge S_8 = 1 \tag{11}$$

$$S_1 = 0 \wedge S_2 = 0 \wedge S_3 = 1 \wedge S_8 = 2 \tag{12}$$

Note that, in this simplified example, the three conditions are very similar, but that needs not be the case in general. The cause of the conflict detected in node D can now be diagnosed as the consensus of the causes of all the conflicts in the children nodes, A, B and C. The consensus of a set of restrictions is simply the conjunction of all restriction elements that are in agreement in all the restrictions in the set. A variable that is present in all the values of its domain is removed. In this case, all the values for variable S_8 were tried, and therefore the conflict cannot be due to any specific choice of S_8. This is a general rule, and a non-chronological backtrack can only be made after all the possible choices for a given variable have been tried.

The conflict in node D is therefore diagnosed as being caused by

$$S_1 = 0 \wedge S_2 = 0 \wedge S_3 = 1 \qquad (13)$$

To solve this conflict, the negation of this condition has to be asserted, and therefore the restriction

$$S_1 \neq 0 \vee S_2 \neq 0 \vee S_3 \neq 1 \qquad (14)$$

can be added to the database, since this restriction will have to be satisfied in any assignments that lead to a solution.

Clearly, this restriction can only be satisfied if a non-chronological jump to the level of node E is performed. Note that the condition in (13) is propagated backwards as the cause of the conflict in node D, and will be used in the computation of the cause of the conflict in node E, if one exists.

This leads to the following general procedure for the diagnosis of conflicts and control of backtrack:

1. At each leaf in the search tree, compute the set of assignments involved in the conflict.

2. At each non-leaf node where a conflict is detected, compute the consensus of all the conditions involved in the conflicts of children nodes.

3. Complement the resulting condition, and add it to the restriction database. Also, store this condition as the cause of the conflict at this node.

4. Compute the highest decision level involved in this condition, and perform a non-chronological backtrack to that level.

5. Experimental Results

To test the performance of the algorithms, we randomly generated 115 finite state automata with binary inputs and outputs. These finite state automata were reduced and unreachable states were removed before the experiments were run. The size of the automata (after reduction) varied between 3 and 19 states. A total of 575 IO mappings were generated, with each IO mapping containing twenty strings of length 30. Each program was given 30 minutes of CPU time and 64 Megabytes of memory to find the minimum consistent automaton in a 133MHz Pentium running Linux.

Figure 6 shows the number of backtracks required for each problem. The first algorithm (labeled *Bierman*) is the approach proposed by Bierman [4] and described in section 3. This algorithm uses the improvements described in the final part of that section, but does not use any of the new techniques described in section 4. The second algorithm, BIC, uses the techniques proposed in this work and described in section 4. The problems were sorted in order of increasing CPU time taken by the algorithms. The improved version of the algorithm was able to solve 468 problems, while the original version, as proposed by Bierman, solved only 371 problems. It is clear that the number

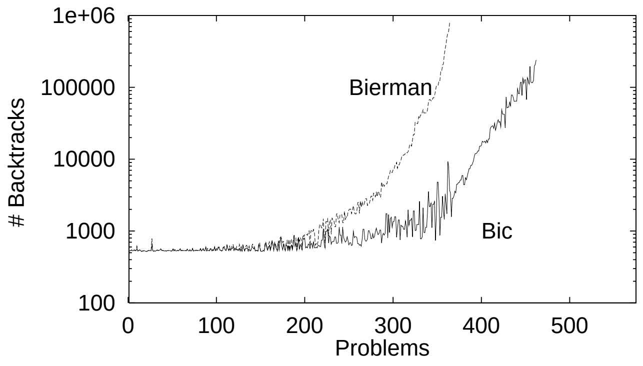

Figure 6. Number of backtracks executed by each of the algorithms.

of backtracks by the improved algorithm is much smaller than the number of backtracks used by the original algorithm proposed by Bierman. However, each backtrack takes longer to execute in the modified algorithm. Figure 7 shows the time **each** backtrack takes to execute, for all the problems that are solved in the alloted time. This graph shows that, in the interesting range, i.e., problems with index 250 to 360, each backtrack is on the order of 2 times more costly if the improved algorithm is used.[3] However, this extra cost is more than offset by the much smaller number of backtracks needed. Figure 8 shows the total CPU time spent by each of the two algorithms. From this graph, it is clear that the techniques proposed in this work improve the performance of the search algorithm by a large factor. The net

[3]For the easier problems, the ones with an index lower than 250 in these graphs, very little CPU time is spent overall and the statistic shown in figure 7 is not very significant.

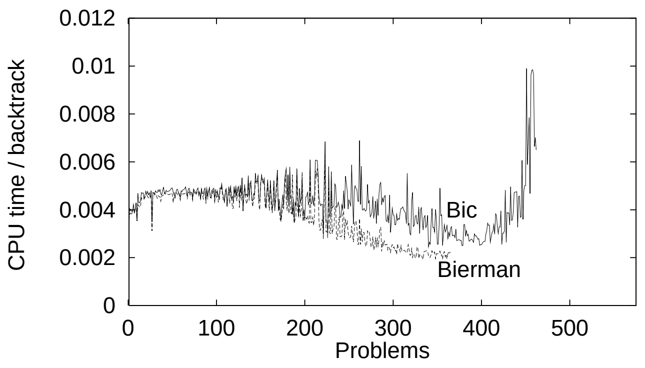

Figure 7. Cpu time by executed backtrack in the search tree.

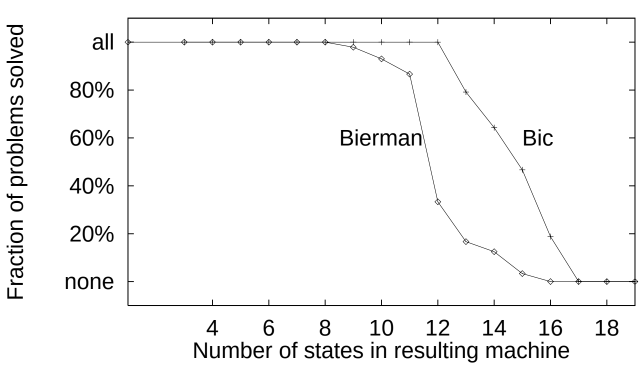

Figure 9. Fraction of the problems solved by the original and modified algorithms.

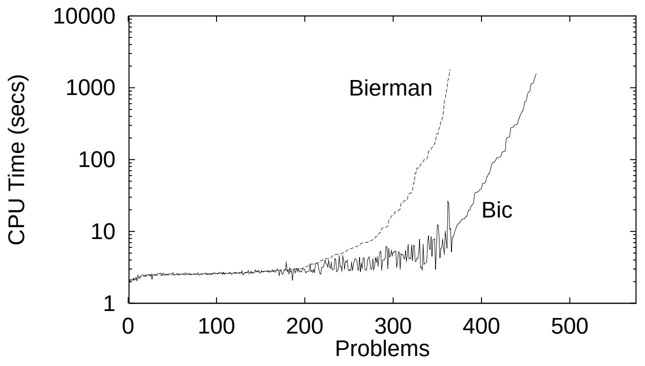

Figure 8. Comparison of the CPU time for the original and improved algorithms.

result of this improvement is that, given a fixed amount of time, many problems that are outside the reach of the existing approaches can be solved by BIC. Figure 9 shows the fraction of problems that can be solved, plotted as a function of the number of states in the finite state automaton that represents the solution. BIC manages to solve all problems that have a solution with no more than 12 states. It is clear from these results that, for the randomly generated automata created by the procedure outlined above, the use of dependency-directed backtracking provides a very significant advantage. In many of these cases, the application of these techniques reduces the CPU time required by many orders of magnitude.

6. Conclusions and Future Work

We presented a new approach for the problem of inferring the finite state automaton with minimum number of states that is consistent with a given input/output mapping. This approach is based on well known algorithms but uses a technique that provides a very significant improvement, dependency-directed backtracking. Although this technique has previously been applied in other domains, it has not been used on this specific problem.

The results show that, for the set of problems studied, the use of these techniques results in an algorithm that is many orders of magnitude faster that the alternative approaches. For this set of problems, the extra overhead incurred by the bookkeeping necessary to apply these techniques is recovered in the vast majority of the problems, with the exception of the smaller and simpler ones.

For the set of problems studied, the algorithms described in this paper find the exact solution in very little time in all problems that have solutions up to 11-12 states, and become progressively less effective in problems that have solutions between 13 and 15 states. Naturally, the dimension of the problems that can be solved depends strongly on the exact input/output mapping used, the number of possible inputs and outputs and the structure of the state transition graph. We believe, however, that the techniques described here will be extremely effective in a variety of other situations.

There are several open problems that are of interest for future research. One of these problems is related with the outer loop of the algorithm. The current version of BIC starts by looking for a automaton with N states, where N is the size of the largest clique found in the incompatibility graph. If this search fails, it increases the target size by one, and restarts the algorithm, therefore loosing all the information stored in the restriction database. We are currently looking at ways to change this procedure so that useful and still valid restrictions are kept in the database. Since some restrictions may be no longer valid when the target size is increased, this is an interesting topic for future research. Another topic for future research is the applicability of additional pruning techniques like recursive learning

[12] to speedup the search process. It may also be possible to improve the performance of the system by a considerable factor if a more strict control is imposed on restrictions added to the clause database. The current version of the algorithm poses no limits on the size of the restrictions and does not analyze whether redundant restrictions are added to the database, although a simple fingerprinting technique is used to avoid duplication of equivalent restrictions.

A distinct possibility for the solution of this type of problems is the use of heuristic algorithms, like, for instance, the ones described by Lang [13]. This particular algorithm collapses the large original LFDFA using local and possibly non-optimal choices. Clearly, these algorithms remain the option of choice if the exact algorithms cannot handle the problems under study. We believe, however, that the search techniques described in this work can be used in an heuristic approach in problems where the search for the optimal solution is too expensive.

Finally, it must be observed that the algorithms described here solve problems specified by a very general set of restrictions. The most interesting direction for future research will probably be the application of these techniques to other problems that can be formulated in a similar fashion.

7. Acknowledgments

The authors would like to thank Tiziano Villa, who helped greatly in the preparation of this work and parts of this manuscript, having contributed with many helpful suggestions. They also thank Timothy Kam and Stephan Edwards by helping with the experimental procedures. Finally, they thank the Portuguese PRAXIS XXI program and Prof. Sangiovanni-Vincentelli by their support to this line of research.

References

[1] D. Angluin. On the complexity of minimum inference of regular sets. *Inform. Control*, 39(3):337–350, 1978.

[2] D. Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75(2):87–106, Nov. 1987.

[3] A. W. Biermann and J. A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Transactions on Computers*, 21:592–597, June 1972.

[4] A. W. Biermann and F. E. Petry. Speeding up the synthesis of programs from traces. *IEEE Trans. on Computers*, C-24:122–136, 1975.

[5] R. Carraghan and P. M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375–382, November 1990.

[6] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7(3):201–215, 1960.

[7] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127–162, Mar. 1986.

[8] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[9] E. M. Gold. System identification via state characterization. *Automatica*, 8:621–636, 1972.

[10] E. M. Gold. Complexity of automaton identification from given data. *Inform. Control*, 37:302–320, 1978.

[11] J. N. Gray and M. A. Harrison. The theory of sequential relations. *Information and Control*, 9, 1966.

[12] M. H. Kunz and D. K. Pradhan. Recursive learning: an attractive alternative to the decision tree for test generation in digital circuits. In *Proceedings of the International Test Conference*, pages 816–825, 1992.

[13] K. J. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *Proc. 5th Annu. Workshop on Comput. Learning Theory*, pages 45–52. ACM Press, New York, NY, 1992.

[14] A. L. Oliveira. *Inductive Learning by Selection of Minimal Complexity Representations*. PhD thesis, UC Berkeley, December 1994. Also available as UCB/ERL Technical Report M94/97.

[15] A. L. Oliveira and S. Edwards. Limits of exact algorithms for inference of minimum size finite state machines. In *Proceedings of the Seventh Workshop on Algorithmic Learning Theory*, number 1160 in Lecture Notes in Artificial Intelligence, pages 59–66, Sydney, Australia, Oct. 1996. Springer-Verlag.

[16] C. F. Pfleeger. State reduction in incompletely specified finite state machines. *IEEE Trans. Computers*, C-22:1099–1102, 1973.

[17] L. Pitt and M. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *J. ACM*, 40(1):95–142, 1993.

[18] S. Porat and J. A. Feldman. Learning automata from ordered examples. In *Proc. 1st Annu. Workshop on Comput. Learning Theory*, pages 386–396, San Mateo, CA, 1988. Morgan Kaufmann.

[19] S. Russel and P. Norvig. *Artificial Intelligence : A Modern Approach*. Prentice-Hall, 1996.

[20] R. E. Schapire. *The Design and Analysis of Efficient Learning Algorithms*. MIT Press, Cambridge, MA, 1992.

[21] J. P. M. Silva. *Search Algorithms for Satisfiability Problems in Combinational Switching Circuits*. PhD thesis, University of Michigan, May 1995.

[22] J. P. M. Silva and K. Sakallah. Grasp - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227. IEEE Computer Society Press, 1996.

[23] R. M. Stallman and G. J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9:135 – 196, 1977.

[24] B. A. Trakhtenbrot and Y. M. Barzdin. *Finite Automata*. North-Holland, Amsterdam, 1973.