A Stochastic Search Approach to Grammar Induction

Hugues Juillé and Jordan B. Pollack

Computer Science Department, Brandeis University
Waltham, Massachusetts 02254-9110, USA
hugues,pollack@cs.brandeis.edu

Abstract. This paper describes a new sampling-based heuristic for tree search named SAGE and presents an analysis of its performance on the problem of grammar induction. This last work has been inspired by the Abbadingo DFA learning competition [14] which took place between Mars and November 1997. SAGE ended up as one of the two winners in that competition. The second winning algorithm, first proposed by Rodney Price, implements a new evidence-driven heuristic for state merging. Our own version of this heuristic is also described in this paper and compared to SAGE.

1 Introduction

In the field of Artificial Intelligence, important efforts are devoted to the design of efficient search algorithms. Among those approaches, stochastic search algorithms benefit from several interesting properties. First, they often admit an efficient implementation on distributed architectures because they use a limited central control strategy. Secondly, they offer a general purpose procedure when little knowledge is available about the intrinsic properties of the problem or when this knowledge is difficult to introduce in a search procedure. In particular, techniques like Genetic Algorithms (GAs) [8], Genetic Programming (GP) [12], Evolutionary Programming (EP) [5] or Evolutionary Strategies (ES) [2] have had some recent success to tackle some problems with ill-defined search space. Basically, those algorithms sample the state space in order to gather information about the distribution of solutions. Then, this information is used to control the focus of the search.

This paper describes a new algorithm named Self-Adaptive Greedy Estimate (SAGE) search procedure whose underlying heuristics are similar to the ones implemented in those evolutionary algorithms. However, this algorithm has been designed for the search of trees or directed graphs (DAG). Indeed, problems for which the natural representation of the state space is a tree of a DAG are usually not amenable to evolutionary search. So far, random sampling techniques on search trees have been used essentially to predict the complexity of search algorithms [11, 3], but never as a heuristic to control the search. We believe our algorithm to be the first to exploit that knowledge. The work presented in this paper tests this search algorithm on a *grammar induction* problem. This application originated from the Abbadingo DFA learning competition [14] which took place between March and November 1997. This competition proposed a set of difficult instances for the problem of DFA learning as a challenge to the machine learning community and to encourage the development of new algorithms for grammar induction. Our motivation for using a search intensive approach came from a first insight that there might be no simple heuristic to address the problems proposed in that competition. The outcome of the competition proved us wrong on that point since an evidence-driven heuristic discovered by Rodney Price appears to be very adapted for that task, improving very significantly over the Trakhtenbrot-Barzdin algorithm. Because of the requirement in computing resource, SAGE was not able to address problems with large target DFA. For that reason, it was defeated by the evidence-driven heuristic on the problems with average density training sets. However, by solving one problem from the set of challenges with sparse training data, SAGE ended up as a co-winner of that competition.

The paper is organized as follows: Section 2 presents an overview of the SAGE search algorithm. Section 3 describes the DFA learning task and the Abbadingo competition. Then, the implementation of SAGE is described in section 4 along with an implementation of the evidence-driven heuristic. Experimental results are presented in section 5. In this last section, an analysis also compares the performance of SAGE to the Trakhtenbrot-Barzdin algorithm and to the evidence-driven heuristic.

2 Description of the SAGE Search Algorithm

The central component for SAGE is a problem-specific *construction procedure.* This construction procedure is designed such that it always terminates and outputs a feasible solution for the problem under consideration. The construction procedure is an iterative algorithm which makes an ordered sequence of decisions, each decision being selected among a list of valid extensions given the current partial solution. Therefore, this procedure defines the search space as a tree. Solutions correspond to the leaves of that tree while internal nodes represent partial solutions. Eventually, the search space can be a directed acyclic graph (DAG) if there are equivalent nodes in the tree.

There is a well-known AI algorithm for search in DAGs and trees called *Beam search.* Beam search examines in parallel a number of nearly optimal alternatives (the *beam*). This search algorithm progresses level by level in the tree of states and it moves downward only from the best w nodes at each level. Consequently, the number of nodes explored remains manageable: if the branching factor in the tree is at most b then there will be at most wb nodes under consideration at any depth. SAGE is similar to this algorithm in the sense that it implements a multi-threaded search. More precisely, SAGE is composed of a population of *processing elements.* Each of them plays the role of an elementary search algorithm and is seen as one alternative in the beam. However, there are two important differences between SAGE and Beam search. First, Beam search uses an evaluation function to score the different alternatives in order to select alternatives that are most promising. The design of such an evaluation function involves some problem-specific knowledge in order to exploit some properties about the problem under consideration that have been identified. When such knowledge is not available, this approach might not be appropriate. The heuristic exploited by SAGE to address this issue consists in estimating the score of internal nodes by performing a random sampling. That is, the construction procedure selects at random a valid extension at each step until a leaf (or valid solution) is reached. Then, the score of this solution is directly computed with respect to the problem objective function and it is assigned to the initial node. Secondly, the control strategy implemented by Beam search to focus the search simply selects the best w alternatives among the current set of partial solutions. In the case of SAGE, the control strategy involves two mechanisms. The first one implements a model of local competition among the processing elements, which allows the algorithm to allocate more "resources" (i.e., processing elements) to the most promising alternatives in a self-adaptive manner. The motivation for this approach is to allow a scalable system in which the number of elements can be increased easily in order to augment (hopefully) the performance of the algorithm. The second one is a greedy strategy which controls the depth in the search tree of the current alternatives represented by the population of processing elements. This control strategy is discussed in little more details in the following paragraphs.

Put in another way, SAGE is an iterative search procedure for which each iteration is composed of two phases, a *construction* phase and a *competition* phase. SAGE implements a population of elementary randomized search algorithms and a *meta-level heuristic* which controls the search procedure by distributing the alternatives under consideration among this population of processing elements. For any iteration, all the alternatives represented in the population have the same depth in the search tree and they are represented by a number of processing elements which depends on their relative performance. At the beginning of the search, this depth is null and it increases with the number of iterations according to a strategy implemented by the meta-level heuristic. At each iteration, the construction phase is performed, followed by the competition phase. The following operations are performed during those two phases:

- *construction phase*: each processing element calls the construction procedure. This procedure starts the construction from the partial solution represented by the calling processing element and thereafter makes each decision by randomly selecting one choice from the list of valid extensions available at each step. Each random selection is performed with respect to a uniform probability distribution. This phase ends when all the processing elements have constructed a complete solution.
- *competition phase*: the purpose of this phase is to focus the search on most promising alternatives by assigning more representatives to them. This result is achieved by assigning better alternatives to the processing elements that are representative of poor alternatives. A model of local competition between

the different processing elements implements this mechanism. Such a model allows a very efficient implementation on different distributed architectures.

In summary, SAGE is a population-based model in which each processing element is the representative of one alternative for the current level of search in the tree. That alternative determines the initial node from which the random sampling is performed by that processing element during the construction phase. Then, SAGE controls the exploration of the search space according to the following strategy:

1. Initially, the search is restricted to the first level of the tree and each processing element in the population randomly selects one of the first-level nodes.
2. Each processing element scores its associated node (or alternative) by performing a random sampling. This is the construction phase.
3. Then, the competition phase is operated. The purpose of this phase is to focus the search on most promising alternatives.
4. A test is performed by the meta-level heuristic and the result determines whether the level of search is increased by one or not. In the affirmative, each processing element selects uniformly randomly one of the children of its associated node in the search tree and this node becomes the new alternative assigned to the processing element.
5. The search stops if no new node can be explored (because the search reached the leaves of the tree); otherwise it continues with step 2.

In the SAGE model, the level of search in the tree is called the *commitment degree* since it corresponds to a commitment to the first choices of the incremental construction of the current best solution.
A complete description of the search algorithm can be found in [10].

3 Induction of DFAs

3.1 Presentation

The aim of inductive inference is to discover an abstract model which captures the underlying rules of a system from the observation of its behavior and thus to become able to give some prediction for the future behavior of that system. In the field of grammar induction, observations are strings that are labeled "accepted" or "rejected" and the goal is to determine the language that generates those strings. An excellent survey of the field is presented in [1], covering in particular the issue of computational complexity and describing some inference methods for inductive learning. Several representations have been proposed to describe the abstract models used for grammar induction like deterministic finite state automata, boolean formula or propositional logic. More recently, Pollack [16] proposed dynamical recognizers as an interesting alternative to those symbolic approaches, leading to a wide range of Recurrent Neural Network (RNN) architectures [18, 19, 4, 6] that have been employed for similar tasks. However, none of them could compete in the Abbadingo competition because of the proposed problems size.

3.2 The Abbadingo Competition

The Abbadingo competition (organized by Lang and Pearlmutter [14]) is a challenge proposed to the machine learning community in which a set of increasingly difficult DFA induction problems have been designed. Those problems are supposed to be just beyond the current state of the art for today's DFA learning algorithms and their difficulty increases along two dimensions: the size of the underlying DFA and the sparsity of the training data. Gold [7] has shown that inferring a minimum finite state automaton compatible with given data consisting of a finite number of labeled strings is NP-complete. However, Lang [13] empirically found out that the average case is tractable. That is, randomly generated target DFAs are approximately learnable even from sparse data when this training data is also generated at random. One of the aims of the Abbadingo competition is to estimate an empirical lower bound for the sparsity of the training data for which DFA learning is still tractable on average.

Competition Setup A set of DFAs of various size has been randomly constructed. Then, a training set and a test set have been generated from each of those DFAs. Only the labeling for the training sets have been released. Training sets are composed of a number of strings which varies with the size of the target DFA and the level of sparsity desired. The goal of the competition is to discover a model for the training data that has a predictive error rate smaller than one percent on the test set. Since the labeling for the test sets has not been released, the validity of a model can be tested only by submitting a candidate labeling to an "Oracle" implemented on a server at the University of New Mexico [14] which returns a "pass/fail" answer. Table 1 presents the different problems that compose this competition. The size and the depth of the target DFA are provided as a piece of information to estimate how close a DFA hypothesis is from the target.

Procedure for Generation of Problems

- *Generation of target DFAs*: To construct a random DFA of nominal size n, a random digraph with $\frac{5}{4}n$ nodes is constructed, each vertex having two outgoing edges. Then, each node is labeled "accept" or "reject" with equal probability, a starting node is picked, nodes that can't be reached from that starting node are discarded and, finally, the Moore minimization algorithm is run. If the resulting DFA's depth isn't $\lfloor (2\log_2 n) - 2 \rfloor$, the procedure is repeated. This condition for the depth of DFAs corresponds to the average case of the distribution. It is a design constraint which allows the generation of a set of problems whose relative complexity remains consistent along the dimension of target size.
- *Generation of training and testing sets*: A training set for a n-state target DFA is a set drawn without replacement from a uniform distribution over the set of all strings of length at most $\lfloor (2\log_2 n) + 3 \rfloor$. The same procedure is used to construct the testing set but strings already in the training set are excluded.

Table 1. Abbadingo data sets.

Problem name	Target DFA size	Target DFA depth	Training set size
1	63	10	3478
2	138	12	10723
3	260	14	28413
R	499	16	87500
4	68	10	2499
5	130	12	7553
6	262	14	19834
S	506	16	60000
7	65	10	1521
8	125	12	4382
9	267	14	11255
T	519	16	32500

Results of the Competition The description of the development of the competition is presented in details in [15]. The two-dimensional ranking of problems with respect to target size and training data density allowed multiple winners. In fact, two algorithms ended up as co-winners in the competition. The first one used an evidence driven heuristic discovered by Rodney Price. This algorithm outperformed SAGE by being able to solve the largest problems from the first and second group in table 1 (i.e. problems R and S). However, SAGE has later been able to solve problem 7 (the smallest of the problems with sparse training data) and ended up as the second co-winner in the competition.

4 Implementation

4.1 Construction Procedure for SAGE

The construction procedure makes use of the state merging method described in [17]. It takes as input the prefix tree acceptor constructed from the training data. Then, a finite state automaton is iteratively constructed, one transition at a time until a valid DFA is generated (i.e., until every state has a "0" and a "1" outgoing transition).

In the construction procedure two cases are possible when considering a transition: either it goes to an existing state or it goes to a newly created state. As the hypothesis DFA is constructed, states are mapped with nodes in the prefix tree and transitions between states are mapped with edges. When a transition is created going to an existing state, corresponding nodes in the prefix tree are merged. When two nodes in the prefix tree are merged, the labels in the tree are updated accordingly and the merging of more nodes can be recursively triggered so that the prefix tree reflects the union of the labeled string suffixes that are attached to those nodes. Thus, as the DFA is constructed, the prefix tree is collapsed into a graph which is an image of the final DFA when the construction procedure terminates. This merging procedure provides the mechanism to test whether a transition between two existing states is consistent with the labeling and should be considered as a potential choice in the construction procedure. The pseudo-code describing this construction procedure is given in figure 1.

```
Begin with a single state mapped to the root of the prefix tree
The list $\mathcal{L}$ of unprocessed states consists of that state
do
    Pick randomly a state $S$ from $\mathcal{L}$
    Compute the set $\mathcal{T}_0$ of valid transitions on "0" from state $S$
    Pick randomly a transition $t_0$ from $\mathcal{T}_0$
    if ($t_0$ goes to an existing state) then
        Merge corresponding nodes in the prefix tree
    else
        Create a new state, map it to the corresponding
            node in the prefix tree and add it to $\mathcal{L}$
    endif

    Compute the set $\mathcal{T}_1$ of valid transitions on "1" from state $S$
    Pick randomly a transition $t_1$ from $\mathcal{T}_1$
    if ($t_1$ goes to an existing state) then
        Merge corresponding nodes in the prefix tree
    else
        Create a new state, map it to the corresponding
            node in the prefix tree and add it to $\mathcal{L}$
    endif
until ($\mathcal{L}$ is empty)
/* The output is a DFA consistent with the training data */
```

Fig. 1. Randomized construction procedure for DFA learning.

4.2 The Evidence-Driven Heuristic

The state merging method implemented in [17] considers a breadth-first order for merging nodes, with the idea that a valid merge involving the largest sub-trees in the prefix tree has a higher probability of being correct than other merges. The evidence-driven heuristic doesn't follow that intuition and considers instead the number of labeled nodes that are mapped over each other and match when merging sub-trees in the prefix tree. Different control strategies can be designed to explore the space of DFA constructions exploiting this heuristic. Our implementation maintains a list of valid destinations for each undecided transition for the current partial DFA and, as a policy, always gives priority to "forced" creation of new states over merge operations. The pseudo-code for this algorithm is presented in figure 2.

```
Begin with a single state mapped to the root of the prefix tree
The list $\mathcal{S}$ of existing states in the DFA construction consists of that state
The list $\mathcal{T}$ of unprocessed transitions consists of the two outgoing transitions
        from that state, on "0" and "1"
For each $t \in \mathcal{T}$, compute:
        . the subset $\mathcal{S}_{dest}(t)$ from $\mathcal{S}$ of valid destinations for $t$
        . the merge count for each destination in $\mathcal{S}_{dest}(t)$
do
    Construct the subset $\mathcal{T}_0$ of transitions $t \in \mathcal{T}$ for which $\mathcal{S}_{dest}(t) = \emptyset$
    /* Transitions in $\mathcal{T}_0$ cannot go to any existing state */
    if ($\mathcal{T}_0$ is not empty) then
        Select $t_0 \in \mathcal{T}_0$ outgoing from the shallowest node (break ties at random)
        Remove $t_0$ from $\mathcal{T}$
        Create a new state $S_0$ mapped to the destination node for $t_0$ in the prefix tree
        Add $S_0$ to $\mathcal{S}$
        Add the two outgoing transitions from $S_0$, $t'_0$ and $t'_1$, to $\mathcal{T}$
        Compute $\mathcal{S}_{dest}(t'_0)$ and $\mathcal{S}_{dest}(t'_1)$ along with the corresponding merge counts
        For each transition $t \in \mathcal{T}$, add $S_0$ to $\mathcal{S}_{dest}(t)$ if it is a valid destination for $t$
                and compute its merge count
    else
        /* Operate a merge */
        Select $t_0 \in \mathcal{T}$ with the highest merge count (break ties at random)
        Merge the destination node for $t_0$ in the prefix tree with the destination
                state corresponding to this highest merge count
        Remove $t_0$ from $\mathcal{T}$
        For each $t \in \mathcal{T}$, update $\mathcal{S}_{dest}(t)$ and the merge counts
    endif
until ($\mathcal{T}$ is empty)
```

Fig. 2. A construction procedure using the evidence-driven heuristic.

5 Experimental Results

5.1 Problems in the Abbadingo Competition

In a first stage, we used a sequential implementation of SAGE since small populations were enough to solve the smallest instances of the Abbadingo competition. Then, we used a network of workstations to scale the population size and address the most difficult problem instances in the competition. In particular, the solution to problems 5 and 7 involved around 16 workstations on average. This parallel implementation is composed of a server that manages the population of partial solutions and distributes the work load among several clients. This architecture presents the advantage that clients can be added or removed at any time.

SAGE has been able to solve problems 1, 2, 4, 5 and 7 from table 1. To solve problem 7, we proceeded in two steps. First, the construction procedure described previously has been extended with the evidence-driven heuristic in order to prune the search tree. The construction procedure switches to this heuristic when the number of states in the current DFA has reached a given size. Before that threshold size is reached, the construction procedure remains unchanged. After about 10 runs, a DFA with 103 states has been discovered very early. Then, in a second step, more experiments were performed using the original construction procedure but starting with the same first few choices as those that had been made for the 103-state DFA. This resulted in a DFA of size 65. This second step uses SAGE for local search, starting from a good prefix for the DFA construction. The appropriate size for the prefix has been determined experimentally. It is clear from those experiments that the density for the training data available for problem 7 is at the edge of what SAGE can manage. This observation is confirmed by the analysis presented in the following section.

Table 2 presents the population size, the size of the DFA hypothesis and an estimate of the computation time for the different problems that SAGE has been able to solve. We decided to report in that table the values when each problem was solved for the first time. Parameters could be tune to improve the execution time (on average up to a fourfold factor). Experiments for problems 1, 2 and 4 have been performed on a Pentium PC 200MHz. For problems 5 and 7, a network of Pentium PCs and SGI workstations has been used. The evidence-driven heuristic can solve all the problems in the first and the second group in table 1 except problem 5 on which it fails.

Table 2. Experimental results for the SAGE search algorithm applied to problems 1, 2, 4, 5 and 7 of the Abbadingo competition.

Problem name	1	2	4
Population size	64	64	256 (+ best of 2 samples)
Size of DFA model	63 states	150 states	71 states
Execution time	1 hour (sequential)	40 hours (sequential)	4 hours (sequential)

Problem name	5	7 (step 1)	7 (step 2)
Population size	576 (+ best of 8 samples)	1024	4096
Size of DFA model	131 states	103 states	65 states
Execution time	40 hours (parallel)	2 hours (parallel)	4 hours (parallel)

Comparative Performance Analysis. In a comparative study, the performance of the three approaches: Trakhtenbrot-Barzdin (T-B) algorithm, evidence-driven heuristic and SAGE has been evaluated against a set of random problem instances generated using the procedure described in the previous section. For each target DFA, the three algorithms were evaluated across a range of density for the training data in order to observe the evolution of each approach when working with sparser data. For the first two algorithms, 1000 problems were used while only 100 problems were used to evaluate SAGE because of the requirement in computational resources. This comparison has been performed for three values of the population size for SAGE: 64, 256 and 1024 and for two values of the targets nominal size: 32 and 64 states (figures 3 and 4 respectively).

In those experiments, the performance is the ratio of problems for which the predictive ability of the model constructed by the algorithm is at least 99% accurate. This threshold is the same as the success criterion for solving problems in the Abbadingo competition. Figures 3 and 4 show the dependence of SAGE on the population size for its performance. Indeed, a larger population results in a better reliability for the control of the focus of the search because of a larger sample. For the set of problems generated for the purpose of this analysis, SAGE and the evidence-driven heuristic clearly outperform the T-B algorithm. With a population size large enough, SAGE also exhibits a performance consistently better than the evidence-driven heuristic. However, it is difficult to compare those two approaches since SAGE is a general purpose search algorithm using very little knowledge about the problem (i.e., the one introduced in the construction procedure) while the other is a greedy algorithm using a strong problem-specific heuristic. For this reason, SAGE doesn't scale up as well as the evidence-driven heuristic (or the T-B algorithm) for larger target DFAs. The introduction of problem-specific heuristics in the construction procedure becomes necessary for SAGE to address this scaling issue.

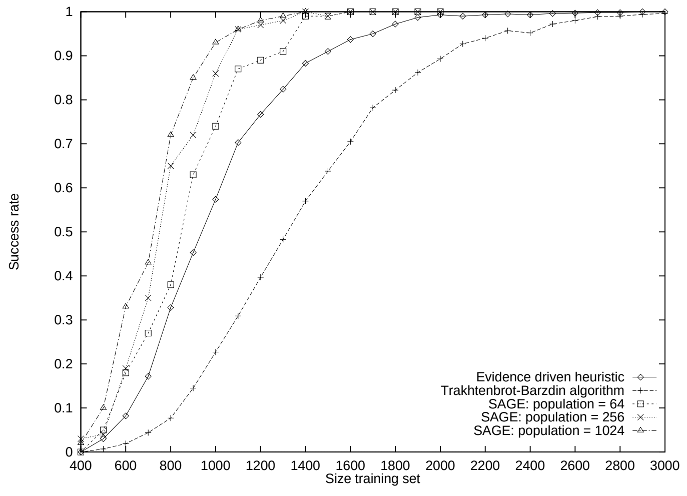

Fig. 3. Comparison of performance for target DFAs of nominal size 32.

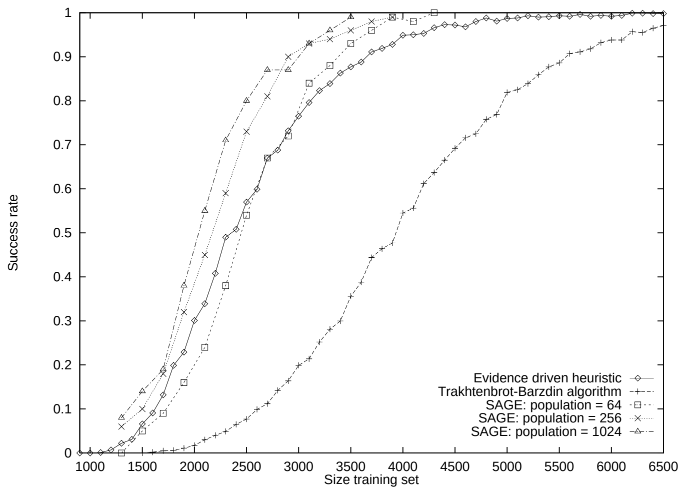

Fig. 4. Comparison of performance for target DFAs of nominal size 64.

6 Conclusion

One property of the state merging approach for DFA learning is that early merges play a very important role since they propagate constraints that control the future merges. This property is exploited by the underlying heuristics implemented in SAGE. Indeed, the random sampling strategy used to evaluate partial solutions is likely to return a better score (that is a smaller DFA hypothesis) on average if the early merges implemented in those partial solutions are correct.

The comparative analysis presented in this paper shows that for average size target DFAs (on the order of 64 to 128 states) SAGE compares favorably to the well-known Trakhtenbrot-Barzdin algorithm and to a new evidence-driven heuristic. However, as the size of the target DFA increases, SAGE doesn't scale up and requires a prohibitive amount of computer resource. To search such a large state space, the introduction of problem-specific heuristics becomes necessary.

DFA induction is not the only field of application for SAGE. In a previous work, we applied SAGE to the construction of sorting networks with a minimum number of comparators [9].

References

[1] Dana Angluin and Carl H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15:237–269, september 1983.

[2] Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9, San Mateo, California, 1991. Morgan Kaufmann.

[3] Pang C. Chen. Heuristic sampling: a method for predicting the performance of tree searching programs. *SIAM Journal on Computing*, 21:295–315, april 1992.

[4] S. Das and M. C. Mozer. A unified gradient-descent/clustering architecture for finite state machine induction. In *Neural Information Processing Systems*, volume 6, pages 19–26, 1994.

[5] Lawrence J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.

[6] M. L. Forcada and R. C. Carrasco. Learning the initial state of a second-order recurrent neural network during regular-language inference. *Neural Computation*, 7(5):923–930, 1995.

[7] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[8] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[9] Hugues Juillé. Evolution of non-deterministic incremental algorithms as a new approach for search in state spaces. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, San Mateo, California, 1995. Morgan Kaufmann.

[10] Hugues Juillé and Jordan B. Pollack. Sage: a sampling-based heuristic for tree search. 1998. Submitted to Machine Learning.

[11] Donald E. Knuth. Estimating the efficiency of backtracking programs. *Math. Comp.*, 29:121–136, 1975.

[12] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[13] Kevin J. Lang. Random dfa's can be approximately learned from sparse uniform examples. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 45–52, 1992.

[14] Kevin J. Lang and Barak A. Pearlmutter. Abbadingo one: Dfa learning competition. http://abba–dingo.cs.unm.edu, 1997.

[15] Kevin J. Lang, Barak A. Pearlmutter, and Rodney Price. Results of the abbadingo one dfa learning competition and a new evidence driven state merging algorithm. 1998. Submitted to Machine Learning.

[16] Jordan B. Pollack. The induction of dynamical recognizers. *Machine Learning*, 7:227–252, 1991.

[17] B. A. Trakhtenbrot and Ya M. Barzdin. *Finite Automata: Behavior and Synthesis*. North Holland Publishing Company, 1973.

[18] R. L. Watrous and G. M. Kuhn. Induction of finite state languages using second-order recurrent networks. *Neural Computation*, 4(3):406–414, 1992.

[19] Z. Zeng, R. M. Goodman, and P. Smyth. Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5(6):976–990, 1994.