A Sampling-Based Heuristic for Tree Search Applied to Grammar Induction

Hugues Juillé and **Jordan B. Pollack**
Computer Science Department
Brandeis University
Waltham, Massachusetts 02254-9110
{hugues, pollack}@cs.brandeis.edu

Abstract

In the field of Operation Research and Artificial Intelligence, several stochastic search algorithms have been designed based on the theory of global random search (Zhigljavsky 1991). Basically, those techniques iteratively sample the search space with respect to a probability distribution which is updated according to the result of previous samples and some predefined strategy. Genetic Algorithms (GAs) (Goldberg 1989) or Greedy Randomized Adaptive Search Procedures (GRASP) (Feo & Resende 1995) are two particular instances of this paradigm. In this paper, we present SAGE, a search algorithm based on the same fundamental mechanisms as those techniques. However, it addresses a class of problems for which it is difficult to design transformation operators to perform local search because of intrinsic constraints in the definition of the problem itself. For those problems, a procedural approach is the natural way to construct solutions, resulting in a state space represented as a tree or a DAG. The aim of this paper is to describe the underlying heuristics used by SAGE to address problems belonging to that class. The performance of SAGE is analyzed on the problem of grammar induction and its successful application to problems from the recent Abbadingo DFA learning competition is presented.

Introduction

The design of massively parallel search algorithms requires the use of new strategies to take advantage of highly distributed architectures. Some examples of such algorithms are Genetic Algorithms (GAs) (Goldberg 1989), Genetic Programming (GP) (Koza 1992), Evolutionary Programming (EP) (Fogel 1962) or Evolutionary Strategies (ES) (Bäck, Hoffmeister, & Schwefel 1991) that have shown very good performance on large scale problems (Hillis 1992; Koza *et al.* 1996). Those algorithms admit an efficient parallel implementation and, basically, sample the search space in order to gather information about the regularities in the distribution of solutions. Then, the search is focused by keeping for the next stage the most promising states according to an objective function. Those algorithms are exploring the state space by applying search operators on a representation for states. Those operators can perform a "recombination" for which a new state is constructed by taking elements from two or more selected states or a "mutation" which alters randomly some elements of a selected state. As a consequence, those search algorithms rely heavily on a local search operator. They require the design of a representation for the state space and the definition of a neighborhood structure over this representation that presents regularities that can be exploited. Problems for which solutions are constructed using an imperative procedure, i.e. each step of the construction depending strongly on the previous steps, make difficult to design a representation and operators which capture the constraints that apply for the construction of valid solutions while at the same time exhibiting an appropriate neighborhood structure with respect to the search procedure. If the representation and the operators can't capture the construction constraints or if the neighborhood structure doesn't present some regularities that can be exploited by the search operators, the search becomes trapped in poor local optima. Such undesirable property of a representation is sometimes referred to as *epistasis*, meaning that the degree of interdependency between the components of the representation is very high and makes the problem very deceptive with respect to the search operators.

On the other hand, traditional artificial intelligence backtracking search algorithms which are designed to explore problem spaces would be appropriate to tackle such problems. However, to face the problem of combinatorial explosion, some heuristics have to be used to control the search. Such heuristics use problem-specific knowledge that has been identified for the problem at hand. However, those heuristics don't use any information about the distribution of solutions in the state space unless this information has been provided explicitly in the definition of the evaluation function (Baum 1992). So far, very little work has been done to explore

the idea of using the distribution of solutions when such an evaluation function is not known or difficult to design.

Our goal was to design an algorithm for tree search that would be able to extract and exploit information about the distribution of solutions and would admit an efficient parallel implementation in order to address large scale problems. The main idea for that algorithm is to collect information about the distribution for the value of the leaves of the search tree corresponding to the alternatives under consideration by performing a series of random sampling. Then, this information is used to focus the search on the most promising alternatives. This result is achieved by using a model of local competition between the elements of the distributed model which allows the algorithm to allocate more "resources" (i.e., processing elements) to the most promising alternatives in a self-adaptive manner. As described later in this paper, such a model provides this algorithm with the advantage of allowing a highly distributed implementation because of a loose central control. Because of this architecture, we named this algorithm *Self-Adaptive Greedy Estimate* (SAGE) search procedure. So far, random sampling techniques on search trees have been used essentially to predict the complexity of search algorithms (Knuth 1975; Chen 1992), but never as a heuristic to control the search. We believe our algorithm to be the first to exploit that knowledge.

This paper presents the application of this search algorithm to the *grammar induction* problem. This application originated from the Abbadingo DFA learning competition (Lang & Pearlmutter 1997) 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. The SAGE search algorithm ended up as one of the two winners of this competition. The second winner was an algorithm that exploits a new evidence-driven heuristic first discovered by Rod Price during the competition.

The paper is organized as follows: Section 2 presents in details the SAGE search algorithm. Section 3 describes the work performed for the Abbadingo competition. In the following section, an analysis compares the performance of SAGE to the Trakhtenbrot-Barzdin algorithm and to our own implementation of the evidence-driven heuristic.

Presentation of SAGE

Principle

We consider the class of problems for which any feasible solution can be generated by a construction procedure. Hence, a search algorithm will proceed by making an ordered sequence of decisions, each decision being selected among a list of choices with respect to a particular strategy. At each iteration, the construction procedure determines this list of choices from the current state of the partial solution. So, the construction procedure defines the search space as a tree. A solution is a path from the root of that tree to a leaf and 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 is a distributed model that implements multi-threaded search. That is, SAGE is composed of a population of *processing elements*. Each of them is playing the role of an elementary search algorithm and is seen as one alternative in the beam. In tree search algorithms, a *fitness* (or score) is assigned to internal nodes in order to determine which nodes will be explored further and which nodes will be ignored. In the case of beam search, heuristics are used to score the different alternatives. However, this approach assumes the existence of such an evaluation function to score nodes. The new idea proposed by SAGE is to estimate the score of internal nodes by performing a random sampling from this node. That is, a path is constructed incrementally and randomly 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.

More precisely, 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. 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. During each iteration, the following operations are performed:

- *construction phase*: each processing element calls the construction procedure designed for the current problem. This procedure starts the construction from the internal node which represents the alternative assigned to the calling processing element and thereafter makes each decision by randomly selecting one choice from the list of choices available at each step. Each random selection is performed with respect to a uniform probability distribution.

- *competition phase*: the purpose of this phase is to focus the search on most promising alternatives by assigning more representatives to them. The details of that phase are described below.

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. Then, the search space is explored 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 alternative by performing a random sampling. This is the construction phase.
3. The competition phase is operated. It results in the search focusing on most promising alternatives.
4. The meta-level heuristic 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 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. The next three sections describe the construction phase, the competition phase and the management of the commitment degree.

Construction Phase

The construction procedure used during this phase plays a very important role since it determines the distribution of solutions when sampling the search tree. The design of this procedure allows some flexibility. In particular, some problem-specific heuristics can be introduced to drive the search in a particular direction, the search space can be reduced by introducing some constraints or the distribution of solutions can be modified by using some strategies. The *multiple-sampling* technique is an example of such a strategy. This technique simply evaluates the score of an alternative by taking the best out of n samples. In practice, n cannot be too large since the computational resource required increases linearly with its value. However, increasing n can be an interesting alternative to a larger population size since the amount of memory required to store the members of the population can quickly become overwhelming for complex problems.

Competition Phase

The SAGE search algorithm uses a population of geographically distributed processing elements. This architecture makes it amenable to an easy implementation on several models of computer networks and parallel machines. For the sake of simplicity, we will consider a model in which the population of processing elements is mapped on a wrap-around mesh (i.e. a torus). A processing element is assigned to each node of the mesh. In such a setup, the competition phase can be performed by implementing a local competition among the processing elements. Each processing element compares the score of the alternative it represents to those in its neighborhood. This neighborhood is composed of the nodes in the mesh whose Manhattan distance from the current node is lesser than a given value. If there is a node with a higher score then the processing element becomes a representative of the same alternative as that neighbor. In case of a tie, one alternative is selected randomly. So, if a given alternative for the current level of search in the tree is more likely to lead to a high score solution then more copies of that alternative will be represented in the population of processing elements as a result of this competition phase (therefore focusing the beam).

Management of the Commitment Degree

Successive competition phases result in most promising alternatives being represented by a larger and larger number of processing elements. Ultimately, if one waits until all the population agrees on the same alternative before continuing the search on the next level, the beam would be reduced to the exploration of a single alternative. On the contrary, if the level of search is increased too frequently, the beam would become very wide, making the search unreliable. Clearly, for the search to be efficient, a balance must be maintained between those two extremes.

We experimented two different strategies to control the commitment degree. The first one simply increases the commitment degree every n iterations. The drawback of this strategy is that n has to be chosen astutely in order to maintain the right balance between exploration and exploitation. This makes this strategy very brittle. The second strategy performs a measure of the state of the population called *diversity measure* which is an estimate of the width of the beam. To compute this measure, each processing element in the population counts the number of neighbors that correspond to a different alternative than itself. Then, this number is summed over all the members of the population and the result is the diversity measure. This measure reflects the degree of convergence of the population. If only a few alternatives are represented in the population then this measure is small because most of the processing elements represent the same alternative as their neighbors. On the other hand, if many alternatives are explored the diversity measure is large. As the population focus on most promising alternatives because of the competition phase, the diversity measure decreases. When this measure reaches an arbitrarily fixed threshold (which is a parameter of the model), the meta-level

strategy considers that the width of the beam is small enough and the search can continue on the next level of the tree. The drawback of this strategy is that it can take a long time for the diversity measure to reach the given threshold if several alternatives lead to equivalent solutions. In that case, a combination of the two strategies offers a good compromise, the first strategy preventing such deadlocks.

Discussion

As discussed previously, the performance of SAGE relies heavily on the construction procedure. Indeed, the construction procedure determines the distribution of solutions when the search space is sampled. The underlying heuristic exploited by SAGE at each iteration is to focus the search on domains of the state space for which sampled solutions are of better quality. Thus, the assumption is that reliable information is collected by the sampling procedure to control the search. If the construction procedure is not compatible with this assumption, the performance of the search is very poor. In practice, a few tries are often required to discover an appropriate procedure which results in a good performance. As a rule of thumb, the construction procedure should be designed so that early choices result in a reliable discrimination between poor domains of the search space and domains that are worth being explored.

Induction of DFAs

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. (Angluin & Smith 1983) present an excellent survey of the field, covering in particular the issue of computational complexity and describing some inference methods for inductive learning.

The application of the SAGE search algorithm to this problem has been motivated by the Abbadingo competition organized by (Lang & Pearlmutter 1997). It 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 1978) 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 1992) 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. The competition has been setup as follows. A set of DFAs of various size have been randomly generated. 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 (Lang & Pearlmutter 1997) 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. The rule of the competition is that the first person to solve a problem owns it. Because of the two dimensions for the evaluation of the difficulty of problems (namely sparsity of training data and size of target DFA), a partial order is defined over problems resulting in the possibility of multiple winners. This partial order is represented in table 2. In this table, a given problem dominates all those that are in the top-left square whose bottom-right corner corresponds to the position of that problem. For instance, problem 6 dominates problems 1, 2, 3, 4 and 5. At the end, a winner is somebody who owns a problem which is not dominated by any other solved problem. The different naming conventions for problems come from the fact that problems R, S and T were added later in the competition. Table 2 also describes which algorithm was the first to solve each problem. In the competition, SAGE first took the lead by solving problems 1, 2, 4 and 5. Then, the evidence-driven heuristic (EDH) was discovered and defeated SAGE by solving problems 3, R, 6 and S. However, SAGE later solved problem 7, hereby becoming a co-winner. Problems 8, 9 and T were not solved by the end of the competition.

In the next sections, two algorithms are presented to address this grammar induction problem. First, the construction procedure to be used in the SAGE search algorithm is described. Then, our own implementation of an algorithm exploiting the evidence-driven heuristic is presented.

Implementation for SAGE

The construction procedure makes use of the state merging method described in (Trakhtenbrot & Barzdin 1973). 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. This method allows the design of a construction

Problem name	Target DFA states	Target DFA depth	Strings in training set
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

Table 1: Abbadingo data sets.

		Training set density		
		dense		sparse
Target DFA size	small	1: SAGE	4: SAGE	7: SAGE
		2: SAGE	5: SAGE	8: unsolved
		3: EDH	6: EDH	9: unsolved
	large	R: EDH	S: EDH	T: unsolved

Table 2: Partial order over problems and status of each problem.

```
Begin with a single state mapped to the root of the
    prefix tree
The list L of unprocessed states consists of that state
do
    Pick randomly a state S from L
    Compute the set T_0 of valid transitions on "0" from
        state S
    Pick randomly a transition t_0 from 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 L
    endif

    Compute the set T_1 of valid transitions on "1" from
        state S
    Pick randomly a transition t_1 from 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 L
    endif
until (L is empty)
/* The output is a DFA consistent with the training
data */
```

Figure 1: Randomized construction procedure for DFA learning used by SAGE.

procedure which satisfies the rule of thumb discussed in the description of SAGE. Indeed, an incorrect merge performed early in the construction of a DFA results in the propagation of several incorrect constraints and has much more effect on the following steps of the search compared to an incorrect merge performed a few iterations later. Therefore, the sampling technique gives a good idea about which merges are more likely to be correct, provided the training data is not too sparse.

The construction procedure begins with a single state (the initial state of the DFA) which is attached to the root of the prefix tree. Then, at each step of the construction, one of the states that has no outgoing transition is selected at random and the "0" and "1" transitions are created for that state. Two cases are possible when considering a new 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 new 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 is finished. 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. Indeed, if there is an inconsistency in the labeling when trying to merge two nodes, this means that the corresponding transition is not valid. The merging is then undone in order to restore the state of the prefix tree before the operation was performed. The pseudo-code describing this construction procedure is given in Figure 1.

The Evidence-Driven Heuristic

The state merging method implemented in (Trakhtenbrot & Barzdin 1973) 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 S of states in the DFA consists of that state
The list T of unprocessed transitions consists of the
      outgoing transitions from that state
For each t ∈ T, compute:
    . the subset S_dest(t) from S of valid destinations
    . the merge count for each destination in S_dest(t)
do
   Construct T_0 = {t ∈ T s.t. S_dest(t) = ∅}
   /* A new state is created if T_0 is not empty */
   if (T_0 ≠ ∅) then
       Select t_0 ∈ T_0 outgoing from the shallowest node
            (break ties at random)
       Remove t_0 from T
       Create a new state S_0 mapped to the destination
            node for t_0 in the prefix tree
       S ← S ⋃ S_0
       T ← T ⋃ {outgoing transitions from S_0}
       For each outgoing transition t' from S_0:
           . compute S_dest(t')
           . evaluate merge count for each destination
                in S_dest(t')
       For each transition t ∈ T, add S_0 to S_dest(t) if
            S_0 is a valid destination for t and compute
            its merge count
   else
       /* Operate a merge */
       Select t_0 ∈ T with the highest merge count
            (break ties at random)
       Merge the destination node for t_0 in the prefix
            tree with the destination node correspond-
            ing to this highest merge count
       Remove t_0 from T
       For each t ∈ T, update S_dest(t) and the merge
            counts
   endif
until (T = ∅)
```

Figure 2: Pseudo-code of our DFA construction algorithm exploiting the evidence-driven heuristic.

Experimental Results

Results for Problems in the Competition.

In a first stage, we have been using a sequential implementation of SAGE since small populations were enough to solve the smallest instances of the Abbadingo competition. Later, a network of workstations was used 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 uses 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 3 presents the experimental setup along with the results for some of the DFAs that passed the "Oracle" test. We decided to report in this table the value for parameters that had been used when each problem was solved for the first time. 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. Our implementation of the evidence-driven heuristic can solve all the problems in the first and the second column in Table 2 except problem 5. Since this algorithm implements a greedy strategy, the time performance is much better than for SAGE. In particular, it takes a few seconds to solve problems 1 and 4 and about 8 minutes to solve problem S on a Pentium PC 200MHz.

Procedure for Generation of Problems.

For the experimental analysis of our algorithms, we constructed a set of problem instances, using the same procedure as for the generation of Abbadingo challenges.

- *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

Problem name	1	2	4
Population size	64	64	256 (+ best of 2 samples)
Neighborhood radius for competition phase	1	1	1
Value of the threshold for commitment degree increase (radius neighborhood = 1)	200	200	800
Number of generations	200	1600	100
Results (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
Neighborhood radius for competition phase	1	1	1
Value of the threshold for commitment degree increase (radius neighborhood = 1)	2400	2000	10000
Number of generations	250	20	100
Results (size of DFA model)	131 states	103 states	65 states
Execution time	40 hours (parallel)	2 hours (parallel)	4 hours (parallel)

Table 3: Experimental results for the SAGE search algorithm applied to problems 1, 2, 4, 5 and 7 of the Abbadingo competition.

same procedure is used to construct the testing set but strings already in the training set are excluded.

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 while the other is a greedy algorithm using a 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.

Conclusion

This paper presents SAGE, a new algorithm for search in trees and directed graphs based on a random sampling strategy to evaluate the score of internal nodes and on the management of a population to adaptively focus the search on most promising alternatives. This stochastic algorithm admits a parallel implementation on both fine-grained and coarse grained distributed systems because of the loose central control.

The performance of SAGE has been analyzed on the problem of DFA learning, inspired from the recent Abbadingo competition. Those experiments have shown that for average size target DFAs (on the order of 64 to 128 states) SAGE compares favorably to the well-known

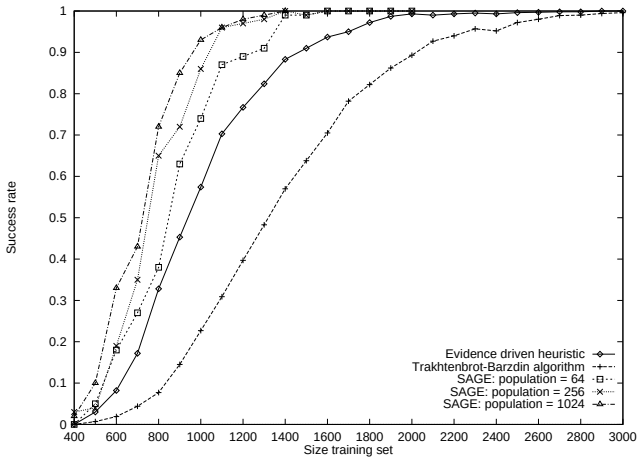

Figure 3: Comparison of performance for target DFAs of nominal size 32.

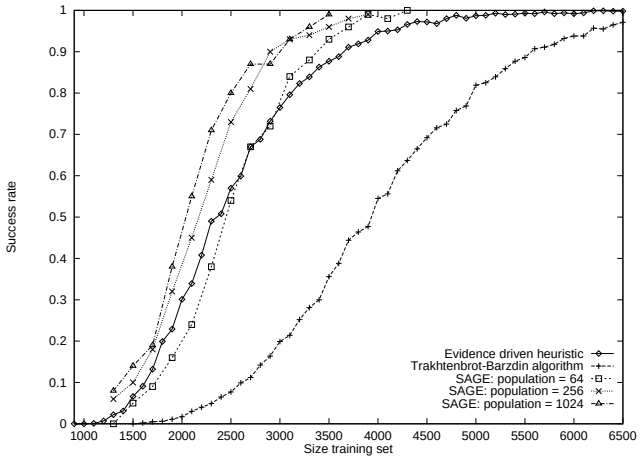

Figure 4: Comparison of performance for target DFAs of nominal size 64.

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.

One important contribution of this paper is to provide some insights concerning the domain of stochastic search algorithms. The general idea for search algorithms is to exploit some information about the state space. This information can be provided either explicitly (hand-coded knowledge) or extracted by performing some form of statistical analysis. Then, this knowledge is exploited to decide how to focus the search. However, the actual distribution of solutions in the search space is a source of information that has rarely been exploited in the context of tree exploration. We believe that SAGE is a first example of a tree search algorithm able to exploit this information efficiently.

References

Angluin, D., and Smith, C. H. 1983. Inductive inference: Theory and methods. *Computing Surveys* 15:237–269.

Bäck, T.; Hoffmeister, F.; and Schwefel, H.-P. 1991. A survey of evolution strategies. In Belew, R. K., and Booker, L. B., eds., *Proceedings of the Fourth International Conference on Genetic Algorithms*, 2–9. San Mateo, California: Morgan Kaufmann.

Baum, E. B. 1992. On optimal game tree propagation for imperfect players. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 507–512.

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

Feo, T. A., and Resende, M. G. 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6:109–133.

Fogel, L. J. 1962. Autonomous automata. *Industrial Research* 4:14–19.

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

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Hillis, W. D. 1992. Co-evolving parasites improve simulated evolution as an optimization procedure. In Langton, C., et al., eds., *Artificial Life II*, 313–324. Addison Wesley.

Knuth, D. E. 1975. Estimating the efficiency of backtracking programs. *Math. Comp.* 29:121–136.

Koza, J. R.; Bennett, F. H.; Andre, D.; and Keane, M. A. 1996. Four problems for which a computer program evolved by genetic programming is competitive with human performance. In *Proceedings of the Third IEEE International Conference on Evolutionary Computation*, 1–10. IEEE Press.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Lang, K. J., and Pearlmutter, B. A. 1997. Abbadingo one: Dfa learning competition. http://abbadingo.cs.unm.edu.

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

Trakhtenbrot, B. A., and Barzdin, Y. M. 1973. *Finite Automata: Behavior and Synthesis*. North Holland Publishing Company.

Zhigljavsky, A. A. 1991. *Theory of Global Random Search*. Kluwer academic. volume 65 of Mathematics and Its Applications (Soviet Series).