Mutually Compatible and Incompatible Merges for the Search of the Smallest Consistent DFA

John Abela[1], François Coste[2], and Sandro Spina[1]

[1] Department of Computer Science & AI, University of Malta
{jabel,sandro}@cs.um.edu.mt
[2] INRIA/IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
francois.coste@irisa.fr

Abstract. State Merging algorithms, such as Rodney Price's EDSM (Evidence-Driven State Merging) algorithm, have been reasonably successful at solving DFA-learning problems. EDSM, however, often does not converge to the target DFA and, in the case of sparse training data, does not converge at all. In this paper we argue that is partially due to the particular heuristic used in EDSM and also to the greedy search strategy employed in EDSM. We then propose a new heuristic that is based on minimising the risk involved in making merges. In other words, the heuristic gives preference to merges, whose evidence is supported by high compatibility with other merges. Incompatible merges can be trivially detected during the computation of the heuristic. We also propose a new heuristic limitation of the set of candidates after a backtrack to these incompatible merges, allowing to introduce diversity in the search.

1 Introduction

Most real world phenomena can be represented as syntactically structured sequences. Examples of such sequences include DNA, natural language sentences, electrocardiograms, speech signals, chain codes, etc. Grammatical inference addresses the problem of extracting/learning finite descriptions/representations, from examples of these syntactically structured sequences. Deterministic finite state automata (DFA) are an example of a finite representation, used to learn these sequences. Section 2 presents to the reader some preliminary definitions. Section 3 describes the current leading DFA-learning algorithm EDSM, whereas sections 4 and 5 introduce a novel heuristic for EDSM, namely S-EDSM, and the backtracking heuristic. Finally, sections 6 and 7 document the initial results of this new heuristic.

2 Preliminary Definitions

This section introduces the reader with the terms and definitions used throughout this paper. It is being assumed that the reader is already familiar with the definitions and results in set theory and formal languages, as well as the area of DFA learning in particular state merging DFA learning algorithms.

G. Paliouras and Y. Sakakibara (Eds.): ICGI 2004, LNAI 3264, pp. 28–39, 2004.

2.1 Transition Trees

Transition trees represent the set of string suffixes of a language L from a particular state. Transition trees are *mapped* onto each other by taking the state partitions of the two transition trees and joining them into new blocks to form a new state set partition. The mapping operation is recursively defined as follows:

Definition 1 (Map) *A transition tree t_1 is **mapped** onto transition tree t_2 by recursively joining together blocks of the set partitions of t_1 and t_2, for each common string prefix present in t_1 and t_2. The result of this mapping operation is a set partition π, consisting of a number of blocks b. Each block in π, is a set of states which have been merged together.*

2.2 State Compatibility and Merges

States in a hypothesis DFA are either unlabeled or labeled as accepting or rejecting. Two state labels A,B are **compatible** in all cases except when, A is accepting and B is rejecting, or, A is rejecting and B is accepting. Two states are **state compatible** if they have compatible labels. The set of all possible merges is divided between the set of **valid** merges, $\mathcal{M}_\mathcal{V}$, and that of **invalid** merges, $\mathcal{M}_\mathcal{I}$. A valid merge is defined in terms of the transition trees mapping operation as follows:

Definition 2 (Valid Merge) *A **valid merge** M_V in a hypothesis DFA H is defined as (q, q'), where q and q' are the states being merged, such that, the* mapping *of q' onto q results in a state partition π of H, with a number of blocks b, such that for each block $b \in \pi$, all states in b are **state compatible** with each other.*

3 State Merging Algorithms

The first state merging algorithm is due to Trakhtenbrot and Barzdin [1]. In their algorithm all the states of the APTA are labeled, hence the algorithm does not make any labeling decisions. We then see Gold's algorithm, in which the algorithm determines the label of unlabeled states in the APTA. Clearly, the order in which merges occur, determines the effectiveness of the learning process. In this algorithm only compatibility is considered, and evidence is not taken into account. EDSM improves on this algorithm by ordering merges on the amount of evidence of each *individual* merge. [2] describes the search space of the regular inference.

3.1 EDSM

The Evidence Driven State Merging (EDSM) algorithm developed by Price [3] emerged from the Abbadingo One DFA learning competition organised by Lang and Pearlmutter [3] in 1998. EDSM searches for a target DFA within a lattice

of hypotheses (automata) enclosed between the augmented prefix tree acceptor (APTA) and the Universal Acceptor Automaton (UA) [2]. EDSM only considers DFAs that are consistent with the training examples. It is assumed that the target DFA lies in the search space of EDSM. It therefore follows, that at least one sequence of merges exists that will lead to the target DFA. The algorithm starts by constructing an augmented prefix tree acceptor (APTA) and then progressively performing merges. The search is guided by an evidence heuristic which determines which pair of states are to be merged [3]. The heuristic is based upon a score, that computes the number of compatible states found in the transition trees of the two states under consideration. At each iteration, all possible pairs of states in the current hypothesis are evaluated[1]. The pair with the highest evidence score is chosen for the next merge. This procedure is repeated until no more states can be merged.

3.2 Problems with EDSM

Although EDSM was one of the winners of Abbadingo One, it still could not solve the hardest four problems. These problems were characterised by very sparse training sets. If EDSM is to find the target DFA (or some other DFA that is close to it) it must, at each iteration of the algorithm, make a 'correct' merge. The scoring function is therefore critical in determining the direction to be taken within the set of possible merges. Since EDSM is a greedy depth-first search it is very sensitive to mistakes made in early merges. The algorithm does not backtrack to undo a 'bad' merge. In general, it is not possible to determine when a 'bad' merge had been made. Very often, EDSM converges to a DFA that is of much larger size than the target DFA. This is evidently because, EDSM makes some 'bad' merges in the beginning.

4 Shared Evidence

In order to improve on what EDSM does, we need to somehow gather more evidence from what is available. We propose that evidence can be augmented by gathering and combining together, the information derived from the interactions between all these valid merges. This combination of individual merge evidence, referred to as *shared evidence*, results in an improvement on the heuristic score. These sets of compatible merges, empirically prove to be valuable in two aspects. Initially they minimise the risk of making mistakes with the initial merges, thus decreasing the size of the hypothesis DFA from that generated by EDSM. Secondly, they prune the search tree (when a search strategy is applied) in such a way that equivalent merges are grouped together and need not be traversed individually. The strategy can also be seen as a kind of lookahead, computing an expectation of the score that you can expect in the next choices.

Shared evidence driven state merging (S-EDSM), is an attempt at a heuristic that tries to minimise the risk of a merge. Undoubtedly there exist no risk

[1] W-EDSM, also presented in [3], takes only a subset of all the possible merges.

free merges, however as opposed to EDSM, S-EDSM tries to share this risk across multiple merges. In this paper we are proposing a subset of a *preliminary* calculus, which specifically deals with how merges interact. A heuristic is then developed based on some of the properties derived from this analysis.

4.1 Pairwise and Mutual Compatible Merges

The most basic interaction between two merges is compatibility. Merge M is said to be *pairwise compatible* to merge M' if after performing merge M, M' remains a valid merge in the hypothesis automaton as changed by M. From this point onwards we will refer to merges which are elements of $\mathcal{M}_\mathcal{V}$. More formally two merges are pairwise compatible, if the following property holds:

Definition 3 (Pairwise Compatible) *Let π_1 and π_2 be the state partitions resulting from the application of the map operator to the two merges M_1 and M_2 on hypothesis H. Let H_1 and H_2 be the hypotheses resulting from π_1, π_2 respectively. M_1 and M_2 are **pairwise compatible** if for each state $s \in H$, $s \in H_1$ is state compatible with $s \in H_2$.*

Table 1. Score Calculation for Pairwise Compatible States

Current Merge	EDSM Score	S-EDSM Score	Pairwise Compatible Merges
$M1$	7	7+2	$\{M8\}$
$M2$	6	6+5+2+1	$\{M3, M8, M10\}$
$M3$	5	5+6+5+4+2	$\{M2, M4, M6, M8\}$
$M4$	5	5+5+4+2	$\{M3, M5, M8\}$
$M5$	4	4+5+3+2+1	$\{M4, M7, M8, M10\}$
$M6$	4	4+5+2	$\{M3, M8\}$
$M7$	3	3+4+2	$\{M5, M8\}$
$M8$	2	2+7+6+5+5	$\{M1, M2, M3, M4, M5,$
		4+4+3+1+1	$M6, M7, M9, M10\}$
$M9$	1	1+2+1	$\{M8, M10\}$
$M10$	1	1+6+4+2+1	$\{M2, M5, M8, M9\}$

Consider the set $V \subseteq \mathcal{M}_\mathcal{V}$, consisting of the 10 merges $\{M1 \,..\, M10\}$. Table 1 lists merges $M1$ to $M10$, together with the set of merges which are pairwise compatible to each merge. For instance, merges $M3$ and $M8$ are both pairwise compatible with $M2$. However, note that this does not necessarily imply that $M3$ is pairwise compatible with $M8$. Moreover from the definition of pairwise compatibility it follows that, if $M3$ is pairwise compatible with $M4$ then $M4$ is pairwise compatible with $M3$. This means that the order in which the two merges are executed does not change the resulting state partition.

Let pairwise compatibility between two states be denoted by the symbol $\uparrow$. Pairwise compatibility induces the binary relation $\uparrow \subseteq \mathcal{M}_\mathcal{V} \times \mathcal{M}_\mathcal{V}$. Thus, $M1 \uparrow M2$ denotes that $M1$ is pairwise compatible with $M2$. Moreover, $M1 \uparrow \{M2, M3, M4\}$ denotes that, $M1$ is pairwise compatible with $M2$, $M3$, and $M4$. Note that $\uparrow$ is a symmetric relation.

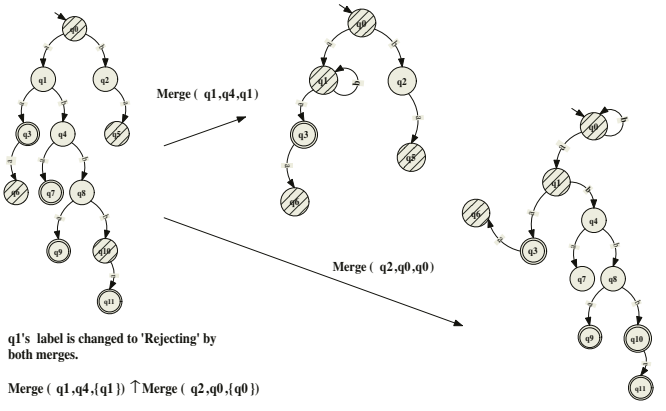

Fig. 1. Simple example of two pairwise compatible merges

$$M1 \uparrow M2 \rightarrow M2 \uparrow M1$$

Let us now consider the possibility that more that just two merges are compatible with each other. This means that for a particular hypothesis H, there *might exist* n valid merges with $n >$ two such that the execution of merges M_1 to M_{n-1} does not effect the validity of merge M_n. In this case we need to extend the definition of pairwise compatibility to include more than two merges. *Mutual compatibility* is defined as follows:

Definition 4 (Mutual Compatibility) *Let n be equal to an arbitrary number of valid merges. Let π_1,π_2, ..., π_n be the state partitions created when applying the map operator to the n merges M_1, M_2, ..., M_n on hypothesis H. Let H_1, H_2, ..., H_n be the hypotheses resulting from π_1, π_2, ..., π_n respectively. M_1, ..., M_n are* ***mutually compatible*** *if, for each state $s \in H$, s is state compatible with $s \in H_1 \wedge ... \wedge s \in H_{n-1} \wedge s \in H_n$.*

5 Merge Heuristic

Shared Evidence Driven State Merging (S-EDSM) is the algorithm based on the definitions presented in the previous section. Only pairwise compatibility is used in the heuristic. There are plans, however, to extend the heuristic to also incorporate other ideas such as merge mutual compatibility, merge dominance, merge coverage and merge intersection. The term **shared** was used to underline the basic notion that, the heuristic works by gathering and combining, thus sharing, the information of individual merges.

5.1 Increasing Evidence

Consider a scenario, where a number of valid merges can be performed. Table 1 shows ten potential merge candidates, with $M1$'s heuristic score being the

highest and $M10$'s heuristic score the lowest. While EDSM just executes the merge with the highest heuristic score (in this case $M1$), S-EDSM first checks for pairwise compatibility between all the merges and creates a list of *pairwise compatible merges* for each merge. Table 1 also shows an example of how these merges can be grouped through pairwise compatibility. The second column in this table indicates the EDSM score for the merge listed in the first column. Hence, the first row in the column is read as $M1 \uparrow \{M8\}$ and the second row as $M2 \uparrow \{M3, M8, M10\}$. Note that the sets are not checked for mutual compatibility. When EDSM merges $M1$ (the merge with the highest evidence score), merges $M2$, $M3$, $M4$, $M5$, $M6$, $M7$, $M9$ and $M10$ become invalid. This means that for the next merge selection procedure only merge $M8$ is possible, since this is the only merge which is pairwise compatible with $M1$. On the other hand, S-EDSM proceeds by first re-ordering the merges according to the pairwise compatibility of the merges. Column 3 of table 1 shows how the scores are re-calculated for all the merges. For the time being scores are simply added together. For instance, in the case of $M2$, the new score is added to $M3$'s score (5), $M8$'s score (2), and $M10$'s score (1). Thus, the score of the pairwise compatible set of $M2$ is set to 13. Simply adding the scores might however not be the best way of re-calculating evidence.

$M8$ is executed first by S-EDSM, since $M8$ is supported by evidence from all the other valid merges. $M4$ is the second merge to be executed, followed by $M3$. Once $M3$ is done, no more merges are possible. The merge sequence created by EDSM consists of two merges, $M1$ and $M8$. S-EDSM creates a merge sequence of three merges, $M8$, $M4$, then $M3$. The overall EDSM heuristic score for this sequence is 9, whereas for S-EDSM it is 12. Thus, it seems that overall S-EDSM has performed a sequence of merges which is better than the one chosen by EDSM. However, one should note that for the time being only pairwise compatibility is being considered. Pairwise compatibility alone *does not* give sufficient knowledge of how many states are actually giving **distinct evidence**. By distinct evidence we mean, the evidence that is given uniquely by a state through a state compatibility check. With pairwise compatibility sets, the same state compatibility check may account (and in practice it usually does) for increasing the evidence score of the pairwise compatible set when summing the individual EDSM scores. Consider for example figure 2. In this simple example, we have two possible merges $(q3, q2, \{q2\})$ and $(q2, q1, \{q1\})$ with EDSM evidence scores of one and two respectively. Clearly these two merges are pairwise compatible. Note however that when re-calculating the scores for S-EDSM, the evidence given by $q2$ is counted twice when calculating the S-EDSM score for the pairwise compatible set of $q1$.

5.2 Pairwise Compatibility – A Lookahead Strategy

Recall that EDSM's main problem is that when only a few states are labeled in the APTA, it is very difficult for the heuristic to determine which merge to perform. S-EDSM, by using pairwise compatibility for single merges, tries to

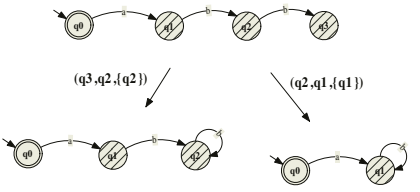

Fig. 2. Two simple merges

alleviate this problem by making use of a *lookahead strategy*. Lookahead occurs because of the following two factors:

- By gathering evidence for a single merge M from multiple merges pairwise compatible with M, and
- By increasing the number of labeled states (labeled by M) before recalculating the EDSM scores of the pairwise compatible states of M.

The execution of each valid merge M, before calculating the pairwise compatibility set of M, actually accounts for more evidence in terms of state labels which can now be used. S-EDSM works by always calculating these sets for each step in the search. Consider the DFA illustrated in figure 1. We know that for merge M1 = $(q0, q2, \{q0\})$ and merge M2 = $(q1, q4, \{q1\})$, $M1 \uparrow M2$. $M1$'s EDSM score is equal to zero simply because there are no states which are state compatible, which are both accepting or rejecting in the respective transition trees of $q0$ and $q2$. However, when calculating the pairwise compatible set for merge $M2$, EDSM's score for merge $M1$ becomes one, since now $q1$ has been labeled as rejecting by $M2$. This new evidence supports the execution of $M2$.

5.3 Calculating the Evidence Score

Computing the set of pairwise compatible merges for such a large number of merges is not feasible. The strategy adopted by S-EDSM is to include a parameter which determines the set $\mathcal{M}_\mathcal{S}$, of valid merges which are taken in consideration. Two possibilities have been implemented, with the first option used for the experiments.

- Identify the merge M_H with the highest evidence score. Include in $\mathcal{M}_\mathcal{S}$, all those merges whose EDSM score falls into a percentage from this score (typically 70% of the best score in the experiments).
- Include in $\mathcal{M}_\mathcal{S}$, a percentage of all the valid merges ordered by EDSM score in descending order.

5.4 Using Incompatible Merges in Backtrack

How to backtrack efficiently in the state merging framework is still an open question when the search space is too big for complete algorithms. Since the

Algorithm 1 Merge Score Calculation for S-EDSM

Require: A hypothesis H
Require: A set $\mathcal{V} \subseteq \mathcal{M}_{\mathcal{V}}$ on H
  for $j = 1$ to size($\mathcal{V}$) **do**
    enableTrackingOfMergeChangesOnStack
    $H \leftarrow executeMerge(V_j)$
    for $k = 1$ to size($\mathcal{V}$) **do**
      if V_k is still a valid merge **then**
        V_k is pairwise compatible to V_j
        include V_k in set of pairwise compatible merges of V_j
        score V_j = *score* V_j + *score* V_k
      else
        V_k is not pairwise compatible to V_j
        score V_j = *score* V_j - *score* V_k
      end if
    end for
    restoreMergeChangesFromStack
  end for
  $H \leftarrow executeMerge(highestScore(\mathcal{V}))$

Abbadingo competition, it is admitted that the first choices of EDSM are less informed when learning data become sparse and thus that the earlier merges are the most critical ones. After the winning but cpu demanding approach of Juillé [4], only few proposals have been made, focusing essentially on this first choices by a wrapper technique [5] or even by choosing randomly the first merge [6]. We believe that these methods have been moderately fruitful because they fail escaping from the neighbourhood of the first solution, visiting always the same area of the search space.

We propose here to introduce diversity in the exploration of the search space by limiting the choice of candidate merges after a backtrack to the set of incompatible merges with the undone merge. The set of incompatible merges with the merge with the highest score can be easily memorised with a small modification of algorithm 1: when V_k is not pairwise compatible to V_j, V_k can be added to a set of incompatible merges of V_j (Let us remark that only two sets of incompatible merges are needed: the current one and the one of the best candidate, denoted hereafter $\mathcal{I}$).

A simple implementation of the backtrack scheme may then consist in replacing the last line of algorithm 1 by:

enableTrackingOfMergeChangesOnStack
$H \leftarrow executeMerge(highestScore(\mathcal{V}))$
restoreMergeChangesFromStack
$H \leftarrow executeMerge(highestScore(\mathcal{V} \cap \mathcal{I}))$

More subtle implementations around this scheme can be developed by choosing to propagate the limitation of the candidates to the next choices, but these have to be chosen carefully according to the search strategy and the heuristic.

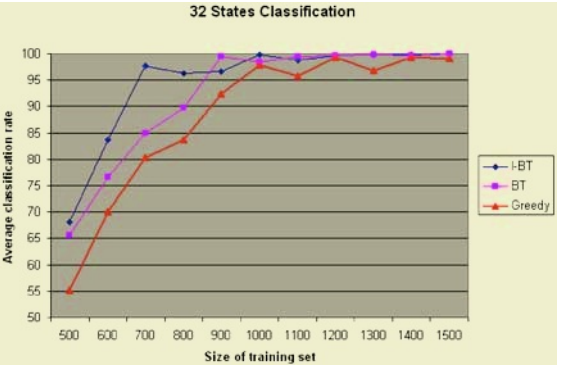

Fig. 3. Ten 32 state Gowachin Problems – Average Classification Rate

We show only here some preliminary results without propagation on small target automata. In these experiments, the backtrack has been limited to the three first merges. Figure 3 shows that using the incompatible merges backtrack scheme (I-BT) allows to outperform the reference algorithm with the same S-EDSM score calculation (BT) on sparser training samples.

6 S-EDSM Classification Rate

Classification of testing data is the best indicator of how well a learning algorithm performs. A comparison on classification rate between EDSM and S-EDSM is not a straightforward task. This is because there are numerous examples, when sparse training sets are used, were both algorithms perform poorly. Suppose EDSM achieves a classification rate of 0.51 and S-EDSM achieves a classification rate of 0.55. In these situations, there is really no difference between the two hypotheses. For this reason, classification rate experiments were carried out as follows. For a given problem, the target DFA was inferred by using a large training set. Portions of this training set are then systematically removed and the two learning algorithms are then applied on the new test sets. With this method we can check which algorithm requires the least amount of training strings in order to infer a good DFA. Since the target DFA is not known, training sets with 30,000 strings are initially used to exactly infer the target DFAs. The graphs of figure 4 show the average classification rates for ten DFAs with target sizes of 256, 192 and 128 states respectively. The three sets of ten problems each, have been sequentially downloaded from Gowachin.

With 128 states, the main difference between S-EDSM and EDSM occurs when the training set is reduced below 6,000 strings. The average classification rate for EDSM degenerates to 1,581 correctly classified strings (i.e. 87%). S-EDSM maintains (although still decreasing) an average classification rate of 1,737 (i.e. 96%). This indicates that, S-EDSM is somewhat less sensitive to sparse training sets. For the 256 states problems, S-EDSM starts to better classify

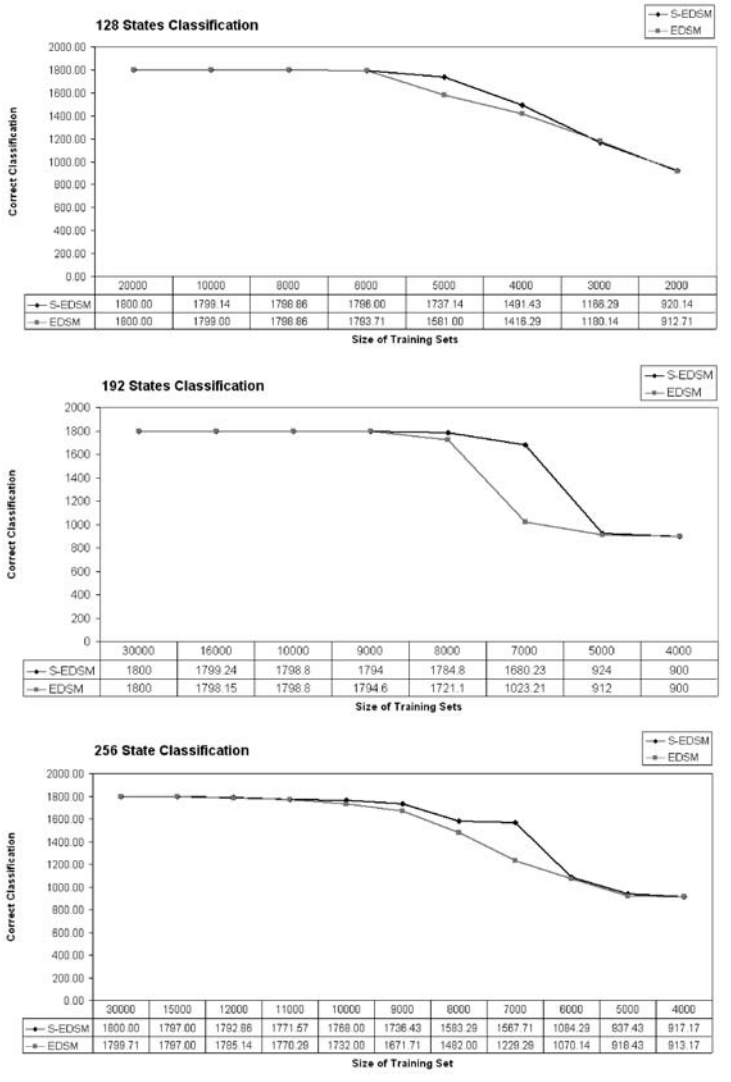

Fig. 4. Ten 128, 192 and 256 state Gowachin Problems - Average Classification Rate

the testing set with 10,000 strings. With 9,000 strings in the training set, the distance in classification rate between EDSM and S-EDSM increases. S-EDSM on average classifies correctly 1,736 strings (96%), while EDSM classifies 1,671 (92%). The biggest difference in classification rate occurs when the training set contains 7,000 strings. The average classification rate for S-EDSM remains high, while EDSM's classification rate drops considerably. Finally, for the set of ten 192 state problems, a similar behaviour is observed.

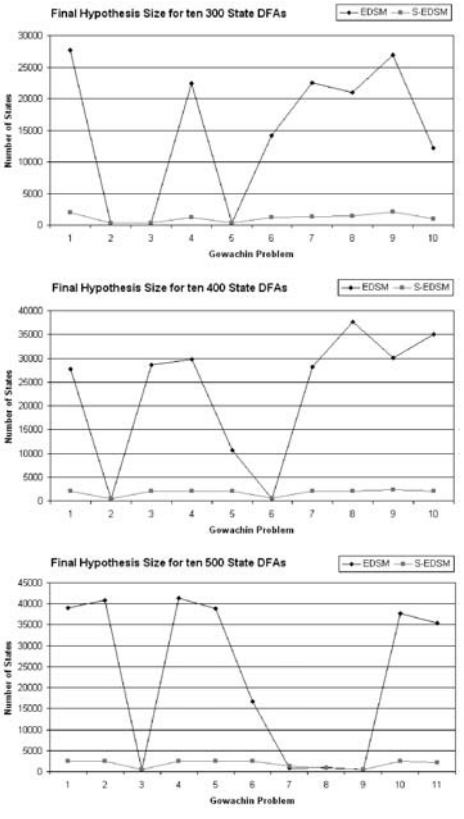

Fig. 5. Ten 300, 400 and 500 state Gowachin Problems - Final Hypothesis Size Comparison

7 Final Hypothesis Size

The final hypothesis size, gives an indication of how well a learning algorithm is searching the lattice of automata compatible with the training set. However, if using Occam's razor, when both algorithms give a classification rate of 0.5, we can argue that the size of the final hypothesis constitutes a measure which can be used to discriminate between the two algorithms. Essentially, the smaller the number of states used, to generalise a finite set of examples, the better. Figure 5 shows thirty consecutively downloaded Gowachin problems with a target size of 500 (24000 strings), 400 (20000) and 300 (12000) states. Although thirty problems certainly do not constitute an exhaustive sample of problems, the sample is enough to demonstrate a trend in the target sizes of the final hypothesis.

It is clear from these thirty problems that, when the percentage of the labeled nodes goes under 20%, S-EDSM outperforms EDSM heavily in target size convergence. S-EDSM's heuristic seems to be paying back in terms of target size convergence.

8 Conclusion and Perspectives

Initial results on S-EDSM's heuristic show that there is an improvement in both classification rate and final hypothesis size. The reason for this can be attributed to the fact that, S-EDSM augments its evidence score by combining the information of multiple valid merges. By doing so, S-EDSM avoids very 'bad' merges in the beginning, when the information is sparse. Considering both compatible merges and incompatible merges seems also promising and we think that S-EDSM coupled with a backtrack heuristic based on incompatible merges is worth being studied more systematically on artificial and also real data.

References

1. B. Trakhtenbrot and Ya. Barzdin. *Finite Automata: Behavior and Synthesis*. North Holland Pub. Comp., Amsterdam, 1973.
2. P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference ? In *Grammatical Inference and Applications, ICGI'94*, number 862 in Lecture Notes in Artificial Intelligence, pages 25–37. Springer Verlag, 1994.
3. K. Lang, B. Pearlmutter, and R. Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. *Grammatical Inference. ICGI 1998.*, LNAI 1433:1–12, 1998.
4. H. Juillé and J.B. Pollack. A stochastic search approach to grammar induction. In *Grammatical Inference*, number 1433 in Lecture Notes in Artificial Intelligence, pages 126–137. Springer-Verlag, 1998.
5. K. Lang. Evidence driven state merging with search. Technical report, NEC Research Institute, October 1998.
6. Orlando Cicchello and Stefan C. Kremer. Beyond edsm. *Grammatical Inference:Algorithms and Applications. ICGI 2002.*, 2002.