Adaptive Finite State Automata and Genetic Algorithms: Merging Individual Adaptation and Population Evolution

H. Pistori, P. S. Martins, A. A. de Castro Jr.
Department of Computer Engineering, Dom Bosco Catholic University, Brazil
E-mail: {pistori,martins,amaury}@ec.ucdb.br

Abstract

This paper presents adaptive finite state automata as an alternative formalism to model individuals in a genetic algorithm environment. Adaptive finite automata, which are basically finite state automata that can change their internal structures during operation, have proven to be an attractive way to represent simple learning strategies. We argue that the merging of adaptive finite state automata and GA results in an elegant and appropriate environment to explore the impact of individual adaptation, during lifetime, on population evolution.

1 Introduction

Most of the early works on genetic algorithms were based on the simplifying assumption that individuals do not change during lifetime. In contrast to this "rigid individuals" approach, some important trends on computational evolution, tackling the challenge of incorporating some kind of plasticity, into individuals modeling, started to emerge, in the last decade. Baldwinian [1] and Lamarckian [2] Evolution, are two of the major approaches, in which, individual phenotype is not a direct map from its genotype, but instead, the phenotype presents some flexibility in responding to the environment input.

Plasticity refers to the flexibility, or capacity for change [3], of a subject, and includes both reasonably simple traits, like the malleability of an amoeba cell membrane, and complex phenomena, like human learnable behaviour. It has been pointed out that individual plasticity is not always a benefi cial trait. Turney, for instance, presented ten dimensions of trade-offs related to the balance of phenotype rigidity and plasticity, like energy consumption, length of learning period and smoothness of the fi tness landscape [4].

Some simulation experiments involving plastic and learning individuals have been proposed. Most of these works are based on the use of artifi cial neural-networks with backpropagation [5] or some simpler hill-climbing strategy [2], to model the ability of an individual to perform a local search on the fi tness space, before a new population is produced by some standard genetic algorithm approach.

The concept of a plastic automaton, that can change its initial structure, during execution, have been initially proposed by Neto to address some problems on compiler construction design and implementation [6]. This formal device was named Adaptive Automaton. Independently, Shutt and Rubstein introduced a similar device, called Self-Modifying Automata [7]. More recently, Klein and Kutrib presented the Self-Assembling Automata [8], which share the same basic concepts created by Neto, Shutt and Rubstein, but with a new formalization. This paper presents the Adaptive Finite State Automata, or $\mathscr{A}$-FSA , as an alternative way to represent plastic individuals in a GA population. The $\mathscr{A}$-FSA formalism is heavily based on Adaptive Automata, but without some features that, despite being important in compiler construction design and some other application areas, would just add superflous complexity to the problem of represeting a GA genotype as a plastic automaton.

The three main reasons why $\mathscr{A}$-FSA is an interesting formalism to represent plastic individuals are: (1) plasticity is an inherent, but easily disabled, characteristic of $\mathscr{A}$-FSA ; (2) tradicional automata, like fi nite state [9], pushdown [10] and automata with multiplicities [11], are already being explored as individual modelling tool, in the context of "standard" genetic algorithms; (3) Adaptive fi nite state automata are related to two well stablished fi elds: formal language theory (FLT) and grammatical inference (GI) [12, 13]. The integration of $\mathscr{A}$-FSA and GA would bring these three fi elds (FLT, GI and GA) closer, facilitating information exchange, and tools reutilization.

The next section presents adaptive fi nite state automata and a graphical notation for their representation. Section 3 discuss the integration of Baldwinian and Lamarckian computational evolution with adaptive automata theory. Finally, conclusion and sugestions for future works are presented.

2 Adaptive Finite State Automata

An $\mathscr{A}$-FSA is a kind of finite state automaton that can change its transition relation during input reading. It can be seen as a simplified version of an adaptive automaton [6] and as a generalized self-assembling automata [8]. Each transition of an $\mathscr{A}$-FSA , besides operating as in a conventional FSA, can be attached to an *adaptive function*, which is executed just before the transition, removing or inserting new elements to the automaton's transition set.

Formally, an $\mathscr{A}$-FSA is a 10-uple $M = \langle Q, \Sigma, q_0, F, \delta, \kappa, \Gamma, \Psi, \Phi, \Delta \rangle$. The first five elements define the *subjacent mechanism*, where:

Q is the state set.

Σ is the input alphabet, finite and non-empty.

$q_0 \in Q$ is the initial state of the automaton.

$F \subseteq Q$ is the final state set.

$\delta \subseteq Q \times \{\Sigma \cup \{\epsilon\}\} \times Q \times \{\kappa \cup \{\epsilon\}\} \times 2^{\Gamma \mapsto \{Q \cup \Sigma\}}$ is the non-deterministic transition relation.

The transition relation differs from the usual one by two new elements: an adaptive function label, taken from κ, and a set $P \in 2^{\Gamma \mapsto \{Q \cup \Sigma\}}$, of parameter assignments, where $\Gamma \mapsto \{Q \cup \Sigma\}$ is a partial function that maps formal parameters to states or input symbols. The adaptive function label may also be an epsilon symbol (ϵ), indicating that the transition is a regular one. The others five elements of M define the *adaptive mechanism*, where:

κ is the set of adaptive functions labels.

Γ is a set of formal parameters and variables.

Ψ is a set of generators.

$\Phi : \{\Gamma \cup \Psi\} \mapsto \kappa$ is a partial function, mapping formal parameters, variables and generators, to adaptive function labels.

$\Delta \subseteq \kappa \times \{?, +, -\} \times \{Q \cup \Gamma \cup \Psi\} \times \{\Sigma \cup \{\epsilon\} \cup \Gamma\} \times \{Q \cup \Gamma \cup \Psi\} \times \{\kappa \cup \{\epsilon\} \cup \Gamma\} \times 2^{\Gamma \mapsto \{Q \cup \Sigma \cup \Gamma \cup \Psi\}}$ is the set of adaptive actions.

The first element of each adaptive action just groups adaptive actions into adaptive functions. The second element defines the type of the adaptive action, which can be a query (?), a remove ($-$) or an insert action ($+$). The remaining elements represent the transitions to be queried, removed or inserted. These elements can be replaced by a formal parameter, a variable or a generator. Formal parameters are mapped to the values defined in δ, during the adaptive function execution. Variables are used to indicate general transition patterns and generators indicate a state that is to be used for the first time inside the dynamically changing δ function. A detailed description of an $\mathscr{A}$-FSA is out of the scope of this paper, mainly for space restriction, but the following graphical representation proposal, can express, more intuitively, the operation of an adaptive function.

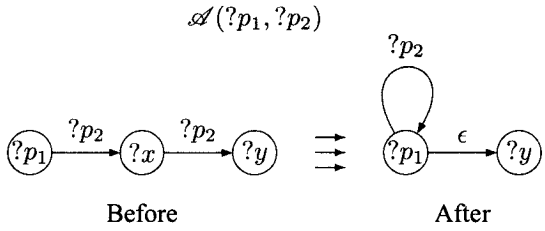

Fig. 1. Adaptive function Graphical Representation

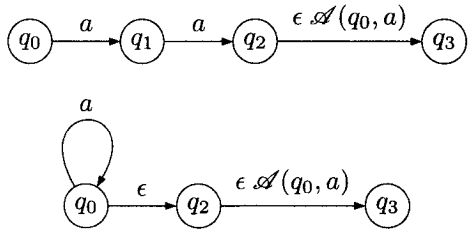

Fig. 2. Automaton structure before and after reading *aa*

In the graphical representation, adaptive functions are illustrated by two prototypical automata (represented by the usual graphical notation, with circles and arrows), separated by a triple arrow. The triple arrow direction indicates an automaton sub-structure *before* and *after* the adaptive function execution. Variables and formal parameters are marked with the prefix "?", while generators are prefixed by "*". The adaptive function label and its formal parameters appear above the graphics, using standard notation for functions: the function label followed by a comma delimited, bracket enclosed, sequence of parameters.

Figure 1 shows a 2-parameter adaptive function that removes any two adjacent transitions, departing from state $?p_1$, that reads the same input symbol, $?p_2$; and inserts a loop, on state $?p_1$, reading $?p_2$. An empty transition is also created, from $?p_1$ to the state previously reached by the adjacent transitions. This adaptive function, if properly used, generalizes the language accept by the automaton, creating a loop, after reading two con-

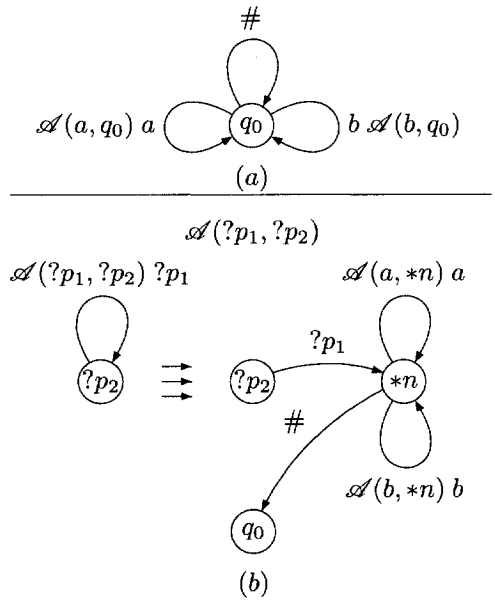

Fig. 3. Adaptive FSA for building Prefi x-Tree Acceptor

(a) Subjacent Mechanism (b) Adaptive Mechanism

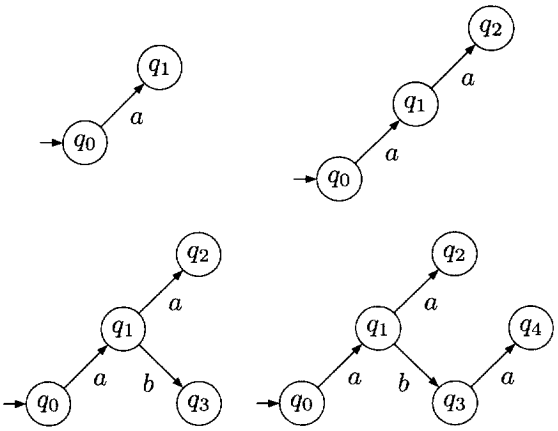

Fig. 4. Automaton structure as it reads *aa#aba*

secutive symbols. Figure 2 illustrates how a subjacent mechanism should look before and after the execution of this adaptive function.

Finally, its worth noting that an $\mathscr{A}$-FSA can be easily specialized to a self-assembling fi nite automata of degree k [8], just by: (1) restricting to k, the number of adaptive function parameters, (2) not accepting input symbols (just states) to be passed as parameters, (3) not allowing variables and (4) not allowing transitions to be removed. A formal proof of this result, which will imply that the class of languages accepted by self-assembling fi nite automata, is a subset of the one accepted by an $\mathscr{A}$-FSA , is under development.

2.1 An $\mathscr{A}$-FSA that builds a prefix-tree acceptor

Many grammar induction algorithms start from building a special kind of fi nite state automaton, called prefi x-tree acceptor, or simply, PTA [12]. Algorithms to build PTAs, from a set of positive strings, are straightforward. However, in the following example, both the PTA, and the algorithm that builds a PTA, from sample strings, will be modeled as an $\mathscr{A}$-FSA .

The automaton's subjacent layer, with alphabet $\Sigma = \{a, b, \#\}$, has one state, and is shown in fi gure 3.(a). The number sign, #, is just a string delimiter. Transitions (q_0, a, q_0) and (q_0, b, q_0) are both associated with the 2-parameters adaptive function, $\mathscr{A}$, presented in fi gure 3.(b). The adaptive function just breaks the loop (parameterized by $?p_1$ and $?p_2$), creating a new prefi x-tree edge, and two new loops, similar to the initial ones, that will allow the prefi x-tree to continue growing, if necessary. A string delimiter transition is also inserted to lead the automaton to its initial state whenever a number sign is read. Figure 4 illustrates the automaton's plasticity, as it reads the sample input *aa#aba*. Loops and delimiter transitions are omitted in the sake of clarity.

This automaton can be formalized as an $\mathscr{A}$-FSA $M = \langle Q, \Sigma, q_0, F, \delta, \kappa, \Gamma, \Psi, \Phi, \Delta \rangle$, where the subjacent mechanism has $Q = \{q_0\}$, $\Sigma = \{a, b, \#\}$, $q_0 = q_0$, $F = \{q_0\}$, $\delta = \{(q_0, a, q_0, \mathscr{A}(a, q_0)), (q_0, b, q_0, \mathscr{A}(b, q_0)), (q_0, \#, q_0, \epsilon)\}$ and the adaptive layer, containing just one adaptive function, $\mathscr{A}$, has $\kappa = \{\mathscr{A}\}$, $\Gamma = \{p_1, p_2\}$, $\Psi = \{n\}$, $\Phi = \{(p_1, \mathscr{A}), (p_2, \mathscr{A}), (n, \mathscr{A})\}$. The Δ relation, represented using a notation that emphasizes the adaptive action type (shown outside the brackets) and the association of adaptive function labels and parameters, is:

$$\begin{aligned}\mathscr{A}(?p_1, ?p_2) = \{ \quad & ?(?p_2, ?p_1, ?p_2, \mathscr{A}(?p_1, ?p_2)), \\ & -(?p_2, ?p_1, ?p_2, \mathscr{A}(?p_1, ?p_2)), \\ & +(?p_2, ?p_1, *n, \epsilon), \\ & +(*n, \#, q_0, \epsilon), \\ & +(*n, a, *n, \mathscr{A}(a, *n)), \\ & +(*n, b, *n, \mathscr{A}(b, *n))\}\end{aligned}$$

3 Baldwinian and Lamarckian Evolution with Adaptive Finite State Automata

Genetic Algorithms are very effi cient at exploring the entire search space; however, they are relatively poor at fi nding the precise local optimal solution in the region at which the algorithm converges. For make things better, there are hybrid algorithms which are the combination of improvement procedures, usually working as evaluation functions, and genetic algorithms. In or-

der to improve the algorithms performances, local improvement procedures have been incorporated into GAs, through what could be called "learning" or "individual plasticity". There are two basic strategies in using hybrid GAs: Lamarckian and Baldwinian evolution. The Baldwin Effect, as utilized in genetic algorithms, was first investigated by Hinton and Nolan [14] by allowing an individual's fitness (phenotype) to be determined based on learning. Like in natural evolution, the result of the improvement does not change the genetic structure (genotype) of the individual. Although Lamarckian evolution has been universally rejected as a viable theory of genetic evolution in nature, using ideas inspired on it, in genetic algorithms, can improve their convergence speed [2]. In Lamarckian computational evolution, the genetic structure of an individual can be changed to reflect the results of learning.

In a GA environment where genotype are represented as an $\mathscr{A}$-FSA , the Baldwin effect could be explored by the appropriate utilization of adaptive functions to model plasticity or learning. The plasticity level could be controled by designing, or admitting the evolution, of different adaptive functions. A kind of "Lamarckian effect" could also be easily acchieved by retaining the structural changes suffered by the automaton, between generations.

4 Conclusion

Adaptive finite state automata have been described and an example illustrating their inherent ability to model plasticity, in a formal language and automata theory framework, has been presented. Using adaptive finite state automata to model individuals in GA opens a new path for investigations on the interaction of individual plasticity and evolution, which is an important topic in current computational evolution research. It may also strengths the links between grammar inference and computational evolution areas, providing a new environment for theory and results exchange. Some suggestions for future work include the integration of AdapTools [1], an environment for adaptive automata development, with some GA computational library. In-depth studies on genotype representation, performance and limits of hybrid GA-$\mathscr{A}$-FSA algorithms should also be conducted in the near future.

References

[1] Jones, M., Konstam, A. (1999) The use of genetic algorithms and neural networks to investigate the Baldwin effect. In: Proc. of the ACM symposium on Applied computing, pp. 275–279

[2] Wellock, C., Ross, B. J. (2001) An examination of Lamarckian genetic algorithms. In: Erik D. Goodman (ed.) Genetic and Evolutionary Computation Conference Late Breaking Papers 2001, San Francisco, California, USA, pp. 474–481

[3] Belew, R., Mitchell, M. (1996) Adaptive Individuals in Evolving Populations: Models and Algorithms - SFI Studies in the Sciences of Complexity - Vol.XXIII. Addison Wesley, Boston

[4] Turney, P. (1996) Myths and legends of the Baldwin effect. In: Proc. of the 13th Int. Conf. on Machine Learning, pp. 135–142

[5] Nolfi , N. (1999) How learning and evolution interact: The case of a learning task which differs from the evolutionary task. Adaptive Bahavior 7(2):231–236

[6] Neto, J. J. (1994) Adaptive automata for context-sensitive languages. SIGPLAN Notices 29(9):115–124

[7] Rubinstein, R., Shutt, J. N. (1994) Self-modifying finite automata. In: Proc. of the 13th IFIP World Computer Congress, pp. 493–498

[8] Klein, A., Kutrib, M. (2002) Self-assembling finite automata. In: Proc. of the 8th Annual Int. Conf. on Computing and Combinatorics, pp. 310-319

[9] Belz, A., Eskikaya, B. (1998) A genetic algorithm for finite state automata induction with an application to phonotactics. In: ESSLLI-98 Workshop on Automated Acquisition of Syntax and Parsing, pp. 9–17

[10] Lankhorst, M. M. (1995) A genetic algorithm for the induction of nondeterministic pushdown automata. In: Computing Science Report CS-R 9502, University of Groningen.

[11] Bertelle, C., Flouret, M., Jay, V., Olivier, D., Ponty, J. (2001) Genetic algorithms on automata with multiplicities for adaptive agent behaviour in emergenet organizations. In: Proc. of World Multiconference on Systemics, Cybernetics and Informatics 2001, pp. 22–25

[12] Cicchello, O., Kremer, S. C. (2003) Inducing grammars from sparse data sets: A survey of algorithms and results. Journal of Machine Learning Research 4:603–632

[13] Higuera, C. De La (2001) Current trends in grammatical inference. Lecture Notes in Computer Science 1876:28–30

[14] Hinton, G., Nolan, S. (1987) How learning can guide evolution. Complex Systems 1:495–502

[1] Freely available at http://www.ucdbnet.com.br/adaptools/