Mutation Operators for the Evolution of Finite Automata

Dirk Wiesmann

FB Informatik, LS 11, Univ. Dortmund, 44221 Dortmund, Germany
wiesmann@ls11.cs.uni-dortmund.de

Abstract. Evolutionary programming has originally been proposed for the breeding of finite state automata. The mutation operator is working directly on the graph structure of the automata. In this paper we introduce variation operators based on the automatons input/output behavior rather than its structure. The operators are designed to make use of additional information based on a ranking of states as well as a problem-specific metric which enhances the search process.

1 Introduction

Within the scope of evolutionary programming (EP) the evolution of Mealy automata is studied. An automaton is represented as a directed graph. The nodes are representing the states of the automaton and the edges are representing the state transitions. Every edge is labeled with an input and an output symbol. In the original work of Fogel et al. [5] the mutation operator is working on the graph structure of the automaton. There are five random mutation operators that affect the graph structure in different ways. The effect of a mutation event on the input/output behavior of the automaton is not obvious. In this paper we propose two variation operators which are not motivated by a random variation of the graph structure, but by the effect of a variation on the input/output behavior of the automaton. For simplification we will regard only deterministic finite automata (DFA). In the following we will first give a short overview on the topic of finite automata and present some well known properties we will refer to later on. After that we discuss the traditional mutation operators used in EP. Then the two alternative approaches are presented and evaluated.

2 Deterministic Automata

Finite automata are a formal representation for the analysis of sequential logic systems. For each point in time a finite automaton is in a state q of a finite nonempty set of states Q. In every step the automaton reads a symbol $w_i \in \Sigma$, writes a symbol $y_i \in \Omega$ and changes its state according the mapping $\delta : Q \times \Sigma \rightarrow Q$. Automata of this kind are called deterministic Mealy automata and can be described by the system $(Q, \Sigma, \Omega, q_o, \delta, \gamma)$. Where Q is a finite nonempty set of states, Σ the finite input alphabet, Ω the finite output alphabet, $q_0 \in Q$ the initial state, $\delta : Q \times \Sigma \rightarrow Q$ the state transition mapping, and $\gamma : Q \times \Sigma \rightarrow \Omega$ the output function. Thus, a Mealy automaton computes a function $f : \Sigma^* \rightarrow \Omega^*$. Where Σ^* denotes the set of all finite strings of symbols from the alphabet Σ. In the following we will focus on decision problems. An input string $w \in \Sigma^*$ is said to be accepted, iff the automaton is in a final state after reading w. An automaton A of this kind is denoted as an DFA and can be described as a system $A = (Q, \Sigma, q_0, \delta, F)$. Where $F \subseteq Q$ is the set of final (accepting) states. The set of all strings accepted by A is denoted as the regular language $L(A)$. The language $L^n \subseteq \Sigma^n$ consists only of strings of a fixed length n. Thus, $L^n(A)$ is the set of all strings of fixed length n accepted by the DFA A.

We will propose a mutation operator which is based on the operations intersection, union, and negation of regular languages. All algorithms can work efficiently on DFAs [2]:

Theorem 1. *Given a DFA A accepting the language $L(A)$ a DFA A' for the complement $\overline{L(A)}$ can be computed in linear time $O(|Q|)$.*

Proof. The set F of the final states needs only to be interchanged with the set $Q \setminus F$. □

Theorem 2. *Given two DFAs A_1 and A_2 accepting the languages $L(A_1)$ and $L(A_2)$ a DFA A accepting the language $L(A_2) \cup L(A_2)$ can be computed in time $O(|Q_1||Q_2||\Sigma|)$.*

Proof. The DFA A is constructed as follows. Let $Q = Q_1 \times Q_2$, and q_0 the pair of the initial states from A_1 and A_2. Then let $F = \{(q_i, q_j) \mid q_i \in F_1 \vee q_j \in F_2\}$ and $\delta((q_i, q_j), a) = (\delta_1(q_i, a), \delta_2(q_j, a))$ with $0 \leq i < |Q_1|, 0 \leq j < |Q_2|$. □

Theorem 3. *Given two DFAs A_1 and A_2 accepting the languages $L(A_1)$ and $L(A_2)$ a DFA A accepting the language $L(A_2) \cap L(A_2)$ can be computed in time $O(|Q_1||Q_2||\Sigma|)$.*

Proof. $L(A_1) \cap L(A_2) = \overline{(\overline{L(A_1)} \cup \overline{L(A_2)})}$. □

In the following we will apply these operations to minimum state DFAs only, i.e. DFAs with the minimum number of states.

Theorem 4. *If at first the unreachable states are eliminated from a DFA A, and then the equivalence class automaton A' is constructed, then A' is equivalent to A and has the minimum number of states.*

The proof and further details can be found in [2]. The set of unreachable states of a DFA can be computed by a depth first search starting in the initial state, in time $O(|Q||\Sigma|)$. The non equivalent states can be computed in time $O(|Q|^2|\Sigma|)$.

3 Evolutionary Programming

In the scope of evolutionary programming (EP) the evolution of finite automata was studied since the 60s [5, 4]. The starting point of the research was the question, whether a simulated evolution on a population of contending algorithms is able to produce some kind of artificial intelligence. Intelligent behavior was viewed as the ability to predict an event in a given environment and to react on this event to meet a given goal. For the reason of simplification the environment was modeled as a sequence of symbols taken from a finite alphabet Σ. The algorithms were represented as Mealy automata, which are reading the sequence of symbols. Every symbol the automaton reads activates a state transition and and produces one output symbol from the finite alphabet Ω. The task of the EP system is to evolve an automaton that correctly predicts, i.e., produces, the next symbol to appear in the environment on the bases of the sequence of symbols it has previously observed. Thus, the number of wrong predictions is to be minimized.

An EP system is working on the graph representation of an automaton [5]. An automaton is represented as a directed graph. The nodes are representing the states of the automaton, and the edges correspond to the state transitions. Every edge is labeled with an input and an output symbol. Five different mutations have been derived from the graph description: change of an output symbol, change of a state transition, addition of a state, deletion of a state, and change of the initial state. The mutation operator selects with equal probability a certain mode of mutation and applies it to an individual. Depending on the mode of mutation the nodes and edges are selected with equal probability. The number of mutations per offspring is chosen with respect to a probability distribution [4]. A recombination operator was proposed but not implemented.

The graph representation of an automaton and the resulting five modes of mutation have two advantages. First, every single mode of mutation can be performed efficiently. Provided that the graph is stored as an adjacency list, every change of an output symbol and every mutation of a state transition needs only linear time in the number $|Q|$ of nodes. To add or to delete a state needs quadratic time. The change of the initial state can be done in constant time. Since the deletion of a state and the change of the initial state are only allowed when the parent automaton has more than one state, every mutation leads to a feasible representation of an automaton. But the resulting automaton is not necessarily minimal. In particular there can be nodes and even whole subgraphs that are not reachable from the initial state.

A potential drawback of the mutation may be, that every single mode of mutation is solely based on the structure of a automaton. The size of a mutation, e.g. the length of a mutation step is directly related to the complexity of the structural modification. A mutation which deletes a state and changes a state transition has greater influence on the structure as a mutation that only changes a symbol of the output alphabet. Thus, the impact on the input/output behavior is not considered here. Even the influence of two mutations of the same mode may vary significantly (see section 5.1). Moreover it is difficult to find a suitable distance measure (metric), which measures the structural difference of two automata. Especially for the gradual approximation of a solution in a large search space, it is important that mutation will prefer small steps in the search space (regarding a suitable distance measure). By using EP to evolve programs in form of symbolic expressions it was observed, that preferring mutations with a small effect has some advantages [1]. A formal approach is presented in [3]. By defining two related distance measures within the geno- and the phenotype space, so that neighboring elements have similar fitness, problem-specific knowledge was incorporated into the variation operators. The metric allows to reason about the distance of individuals and the size of mutation steps in a formal framework. The size of a mutation is correlated with the change in the fitness value and is not directly based on the structural modifications within the representation of an individual. The requirements on the mutation operator are described in section 5.2. By the example of a synthetic problem, where a Boolean function has to be found based on the complete training set, it was shown, that systems which fulfill the requirements have a significant advantage [3].

To simplify our consideration we will focus on DFAs in the following. In general this restriction to decision problems is not too strong [6].

4 Fitness function and distance of DFAs

Given a finite subset $S \subseteq \Sigma^*$ and a training set $T = \{(w, f(w)) | w \in S\}$. Based on the training set a DFA has to be found which accepts the language $L(A) := f^{-1}(1)$ for a function $f : \Sigma^* \rightarrow \{0, 1\}$. The EP system has to evolve DFAs which will generalize from the training set to $L(A)$.

For simplicity, we restrict the problem space in two ways. Firstly, we only consider languages with strings of fixed length n. Secondly, the search will be based on the complete training set T_v. For a function $f : \Sigma^n \rightarrow \{0, 1\}$ and the training set $T_v = \{(w, f(w)) \mid w \in \Sigma^n\}$ a DFA has to be found which accepts the language $L^n(A) := f^{-1}(1)$. Thus, A must achieve:

$$\forall(w, 1) \in T \text{ is } w \in L^n(A) \text{ and } \forall(w, 0) \in T \text{ is } w \notin L^n(A).$$

The effects of the restrictions will be discussed later. The fitness function $F(A) := |\{(w, f(w)) \in T_v \mid f(w) = 1 \Leftrightarrow w \in L^n(A)\}|$ counts the number of strings on which the DFA A will make the right decision.

Now, how can the similarity of two DFA A and B be measured? The distance d^n of A and B should be the number of strings on which A and B are not corresponding:

$$d^n(A, B) = |L^n(A)| + |L^n(B)| - 2|L^n(A) \cap L^n(B)|$$

The maximum difference in fitness values of two DFA is $d^n(A, B)$. Note, that the fitness calculation is based on T_v. The distance measure $d^n : \Sigma^n \times \Sigma^n \rightarrow \mathbb{N}$ is a metric. Imagine that all strings from Σ^n are sorted in lexicographical order. A language L^n can then be represented as a bit-string of length $|\Sigma^n|$. The i-th bit equals 1, if the i-th string from Σ^n is in the language L^n. Otherwise the i-th bit equals 0. Thus, d^n equals the hamming distance between the bit-strings belonging to the corresponding languages. Obviously, this distance measure won't distinguish between two structural different DFAs accepting the same language.

5 Proposals for EP mutation operators

5.1 Weighted mutation

The first proposal for a new mutation operator is motivated by the observation that the fitness calculation can provide more information than the pure fitness value only. To compute the fitness of a given DFA for every word in the training set a path beginning at the initial state has to be traversed.

In order to assess the influence of a mutation event every node (state) is assigned a weight index with initial value 0. Every time a node is visited during fitness calculation the weight index is incremented by 1. After fitness calculation on the complete training set, the weights give an upper bound for a change in the fitness value caused by mutation.

Lets consider the following example. Let $\Sigma^4 = \{0,1\}^4$ and $L^4 = \{0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111\}$ be the language of all strings with an even number of 0's and an even number of 1's of length 4. Figure 1 shows the graph representation (state diagram) of an automaton with the corresponding weights after fitness calculation on the complete training set T_v. In three cases the DFA draws the wrong decision on T_v.

Now, let us discuss the impact of different mutations: The state q_2 has a relative high weight of 15. If the state q_2 would be deleted by a mutation event, then the fitness can be changed by the value 15 at most. In comparison the deletion of state q_5 can change the fitness by the value of 3 at most. By ranking the states according to their weights, states with a lower weight can be mutated with higher probability than states with a higher weight. State transitions can be mutated likewise. Transitions beginning in a state with a lower weight will be mutated with higher probability than transitions beginning in a state with a higher weight. The insertion of new states will take place with higher probability between states with a lower weight.

This approach allows to define a reasonable probability distribution on mutation events on every single mode of mutation. But it is not obvious how the different modes of mutation should be weighted among each other. E.g., should transitions be mutated with higher probability than states? Should we mutate states with a low weight more often than transitions beginning in a state with a high weight? Additionally, even with respect to T_v the upper bound may turn out to be a bad estimate for the real change in fitness. E.g., an improvement and a decline of the fitness may cancel each other out. These observations show once again the problems of variation operators purely based on the structure, even when additional information is available.

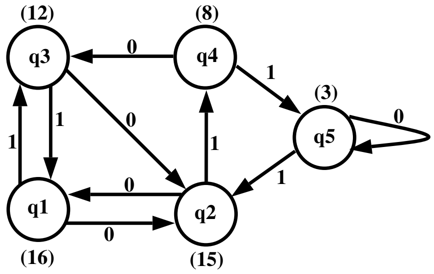

Fig. 1. DFA with weighted states (weights in parenthesis) after fitness evaluation on the complete training set T_4 for the language L^4 of all strings with an even number of 0's and an even number of 1's. State q_1 is initial and final state.

5.2 Metric Based Mutation

In order to overcome the deficiencies described above we first post some formal requirements on the mutation operator. Let $\mathcal{G}$ be the genotype space. Here $\mathcal{G}$ consists of all graph representations of DFAs A accepting a language $L^n(A) \subseteq \Sigma^n$. Since we consider minimum state automata only, $\mathcal{G}$ is finite. Let $d_{\mathcal{G}} : \mathcal{G} \times \mathcal{G} \to \mathbb{N}$ be a suitable metric on $\mathcal{G}$. Without loss of generality we restrict our discussion to the reduced mutation operator $m' : \mathcal{G} \times \Omega_{m'} \to \mathcal{G}$ with the finite probability space $(\Omega_{m'}, P_{m'})$. With probability $P_{m'}(m'(u) = v) := P_{m'}(\{\omega \in \Omega_{m'} \mid m'(u, \omega) = v\})$ the mutation operator m' changes an element $u \in \mathcal{G}$ to a new element $v \in \mathcal{G}$. The first rule assures that from each point $u \in \mathcal{G}$ any other point $v \in \mathcal{G}$ can be reached in one mutation step.

Guideline M 1 *The mutation m' should fulfill:*

$$\forall u, v \in \mathcal{G} : P_{m'}(m'(u) = v) > 0.$$

Moreover small mutations (with respect to $d_{\mathcal{G}}$) should occur more often than large mutations.

Guideline M 2 *The mutation m' should fulfill:* $\forall u, v, w \in \mathcal{G}$:

$$(d_{\mathcal{G}}(u, v) < d_{\mathcal{G}}(u, w)) \Rightarrow (P_{m'}(m'(u) = v) > P_{m'}(m'(u) = w))$$

The mutation should not prefer any search direction, e.g. should not induce a bias by itself.

Guideline M 3 *The mutation m' should fulfill:* $\forall u, v, w \in \mathcal{G}$:

$$(d_{\mathcal{G}}(u, v) = d_{\mathcal{G}}(u, w)) \Rightarrow (P_{m'}(m'(u) = v) = P_{m'}(m'(u) = w)) .$$

A motivation of the guidelines and a discussion of a suitable metric can be found in [3, 9]. We will now design a mutation operator in accordance to the guidelines which uses the metric d^n defined in section 4. The mutation will make use of the efficient synthesis operations for DFAs presented in section 2.

The first step in mutating the DFA A to a DFA B is to randomly choose a step size K with $0 \leq K \leq |\Sigma^n|$ using the following probability distribution:

$$P(K=k) = \begin{cases} \alpha + (1-\alpha)^{|\Sigma^n|+1} & \text{, if } k = 0 \\ \alpha \cdot (1-\alpha)^k & \text{, if } 1 \leq k \leq |\Sigma^n| \\ 0 & \text{, if } k > |\Sigma^n| \end{cases} .$$

This is a slight modification of the geometric distribution ($P(K=k) = \alpha \cdot (1-\alpha)^k$) with parameter $\alpha \in (0,1)$. Using an equally distributed random variable $R \in [0,1]$, the modified geometrical distribution can be created in constant time [7].

Then we choose a subset $M^n \subseteq \Sigma^n$ with $|M^n| = K$. All strings in Σ^n have an equal probability to be selected for the set M^n. The set M^n is split in two sets X^n and Y^n with:

$$\forall x \in X^n : x \in L^n(A), \quad \forall y \in Y^n : y \notin L^n(A) \text{ and } X^n \cup Y^n = M^n.$$

No $x \in X^n$ should be accepted by the DFA B. The DFA B should only accept all $y \in Y^n$. On every other input string A and B should agree. Thus, it is $d^n(A,B) = K$. For the partitioning in the sets X^n and Y^n the DFA A has to be tested K times (cost: $K \cdot n$). To obtain B two DFA A_X and A_Y are constructed with:

$$L^n(A_X) = \Sigma^n \setminus X^n \text{ and } L^n(A_Y) = Y^n.$$

With this we get B as $L^n(B) = (L^n(A) \cap L^n(A_X)) \cup L^n(A_Y)$.

A_X and A_Y are constructed as follows. For every $x^i \in X^n = \{x^1, \ldots, x^{|X^n|}\}$ we construct an automaton A_{x^i} which accepts only the string x^i, thus $L^n(A_{x^i}) = \{x^i\}$. This automaton has $n+2$ states. Figure 2 shows the structure of a DFA only accepting the string $a = a_1 \ldots a_n$. Thus, we have:

$$L^n(A_X) = \overline{L^n(A_{x^1}) \cup \ldots \cup L^n(A_{x^{|X^n|}})}.$$

For every $y^i \in Y^n = \{y^1, \ldots, y^{|Y^n|}\}$ we construct an automaton A_{y^i} only accepting the string y^i as well. With $L^n(A_{y^i}) = \{y^i\}$ we have:

$$L^n(A_Y) = L^n(A_{y^1}) \cup \ldots \cup L^n(A_{y^{|Y^n|}}).$$

After each synthesis operation the resulting DFA will be minimized. An EP system using this mutation operator is called a MBEP system.

Theorem 5. *The constructed mutation operator fulfills the guidelines M1, M2 and M3.*

Proof. The guideline M1 is fulfilled, because every step size $K \in \{0, \ldots, |\Sigma^n|\}$ and all subsets $M^n \subseteq \Sigma^n$ with $|M^n| = K$ have positive probability of being chosen. According to the design of the operator, every language $L^n \subseteq \Sigma^n$ can be generated. The guideline M2 is fulfilled, because for $k_1 < k_2$ it is guaranteed that $P(K = k_1) > P(K = k_2)$, and all subsets $M^n \subseteq \Sigma^n$ with $|M^n| = K$ have an equal probability of being chosen. This also implicates that guideline M3 is fulfilled too. □

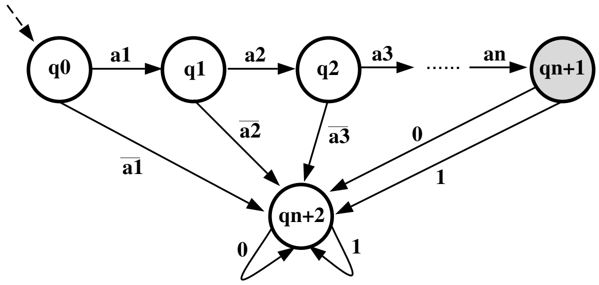

Fig. 2. The structure of a DFA on $\Sigma^n = \{0,1\}^n$. The DFA only accepts the string $a = a_1 \ldots a_n$. The final state is hatched and q_0 is the initial state.

6 Experiments

For reasons discussed in section 7 a direct comparison between EP and MBEP is not possible. Due to its design the MBEP system searches for languages with strings of fixed length n. An EP system can operate on strings of arbitrary length. A (1+1)-MBEP system was tested on two different languages. The first language L^n_{even} consists of all strings of length n with an even number of 0's and an even number of 1's. The second language L^n_{fel} consists of all strings of length n where the last symbol equals the first. The initial start point was chosen by random selection of an element from the set of all languages with strings of length n with equal probability. We used a constant setting $\alpha = 0.3$, but a dynamic adaptation of the parameter α would be possible, too. The number of generations (mutations) until the language was found the first time were averaged over 50 independent runs (Table 1). One has to keep in mind that the time needed for a mutation depends on the length n of the strings, the step size K, and on the size of the DFAs. The mutation operator is efficient in these sizes, but more time-consuming than standard EP mutation (see sections 3 and 5.2, and [3]). For 500 mutations the MBEP system needs for $n = 4$, $n = 6$, and $n = 8$, about 1, 4, and 16 seconds, respectively (on a Sparc Ultra 10/300). It is not surprising that the evolution process for both languages need similar time.

To explain this observation recall the bit-string representation from section 4. Given a bit-string of length $|\Sigma^n|$ for every DFA. At the i-th position the bit-string has a 1, if the DFA draws the right decision for the i-th string. Otherwise this position hold a 0. The fitness function is just counting the number of 1's in the bit-string. Thus, the fitness function equals the counting ones problem [8] on a string of length $|\Sigma|^n$. Since the MBEP mutation operator is based on the metric d^n, all languages L^n have the same difficulty to be found.

Language	Runs	Generations
L^4_{even}	50	143.86
L^6_{even}	50	794.84
L^8_{even}	50	4610.22

Language	Runs	Generations
L^4_{fel}	50	138.11
f^6_{fel}	50	884.82
f^8_{fel}	50	4743.54

Table 1. Number of generations averaged over 50 independent runs until the $(1+1)$-MBEP system found the language the first time.

7 Problems

The work towards applicable MBEP system is still in its in fancies. The MBEP system is subject to substantial restrictions. The system can only work on regular languages with strings of fixed length n. But this restriction could be weakened. Prior to a mutation step a string length could be chosen with respect to a probability distribution. Then the mutation operator works only on strings of the chosen length. The restriction that the MBEP system can only work on the complete training set is much stronger. In its current implementation the system has no generalization ability. Due to the construction of the DFAs (see Figure 2), there can not occur cycles over final states. Strings that are too long or too short are kept in a non accepting state. This problem may be solved by setting transitions starting in state q_{n+1} (see Figure 2) randomly to states in $\{q_0, \ldots, q_{n+1}\}$. Additionally states in $\{q_0, \ldots, q_{n+1}\}$ have to be final states with a certain probability. Unfortunately, first experiments have shown that under this condition the resulting DFAs may become very large. The size of a DFA depend on the size of the incomplete training set. If the training set is too small the DFAs may become too large.

8 Conclusion

In this work we have discussed the mutation operator in evolutionary programming. We proposed two alternative mutation operators for structure optimization. The operators are using additional information to improve the search process. The weighted mutation operator uses information that results from the fitness calculation. The metric based mutation operator uses a problem-specific distance measure. The weighted mutation operator was not analyzed in detail due to open questions. A MBEP system has shown its performance on a synthetic problem. A practical application of a MBEP system remains for future work. But some alternative starting points for the design of variation operators for structure optimization have been identified.

Acknowledgements

This research was supported by the Deutsche Forschungsgemeinschaft as part of the collaborative research center “Computational Intelligence” (531).

References

1. K. Chellapilla. Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation*, 1(3):209–216, 1997.
2. P.J. Denning, J.B. Dennis, and J.E. Qualitz. *Machines, Languages, and Computation*. Prentice-Hall, Englewood Cliffs, 1979.
3. S. Droste and D. Wiesmann. Metric Based Evolutionary Algorithms. In R. Poli, W. Banzhaf, W.B. Langdon, J.F. Miller, P. Nordin, and T.C. Fogarty, editors, *Genetic Programming, Proc. of EuroGP'2000, Edinburgh, 15.–16. April 2000*, LNCS. Springer, Berlin, 2000. (in print).
4. D.B. Fogel. *Evolutionary Computation:* Toward a New Philosophy of Machine Intelligence. IEEE Press, New York, 1995.
5. L.J Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.
6. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York, 1979.
7. G. Rudolph. An evolutionary algorithm for integer programming. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature - PPSN III, Int'l Conf. Evolutionary Computation*, pages 139–148, Jerusalem, October 9–14, 1994. Springer, Berlin.
8. G. Rudolph. *Convergence Properties of Evolutionary Algorithms*. Verlag Dr. Kovač, Hamburg, 1997.
9. G. Rudolph. Finite Markov chain results in evolutionary computation: A tour d'horizon. *Fundamenta Informaticae*, 35(1-4):67–89, 1998.