A SURVEY ON SOME POSSIBILITIES FOR APPLICATIONS OF GENETIC ALGORITHMS IN THE AUTOMATA THEORY

Abstract: One thing is for sure - the origin of life today, nature is very well aware of what’s he is doing. Genetic algorithms (GA) are a family of algorithms that use some kind of genetic principles that are present in nature, in order to solve specific computational problems. Genetic algorithms are robust and adaptive methods in addition to other fields of application can be used for solving combinatorial optimization problems. The reason is that genetic algorithms can be relatively easily adapted to the specific problem. Automata have already been constructed by evolutional techniques in 60’s of last century by Fogel. He created an “evolutional programming” that was the first attempt to construct an automaton by evolutionary approach. The aim of this paper was to further demonstrate the possibility of application of GA in the Automata theory.

Keywords: genetic algorithm, combinatorial optimization, NP problems, finite automata, reset word, trapped word.

1. Introduction

Genetic algorithm (GA) is one of the most interesting classes of optimization algorithms, which with its new applications and very good results in recent years constantly surprising how people from the world of technology, as well as ordinary users who are beneficiaries of products resulting from its use. From unusual ideas, to more unusual applications, mimicking the process of evolution GA demonstrate the power of the mechanisms of nature. Genetic algorithms (GA) are a family of algorithms that use some kind of genetic principles that are present in nature, in order to solve specific computational problems. These natural principles are: inheritance, crossover, mutations, survive of the best custom (or survival of the fittest), migration and so on. These algorithms can be used for solving various classes of problems because they are fairly general nature. By mode of action, genetic algorithms are among the methods directed random search space solutions are looking for a global optimum. Genetic algorithms simulate the natural evolutionary process.

The idea is based on the GA processes imitate natural selection. The basis of the selection process that takes place in nature are the following facts: individuals better adapted to survive the environment and have a stronger influence on
the formation of the next generation, a generation individuals in the population form the next generation, thus so what are the new features of specimens receive a combination of genetic content of the parents, from time to time there is a mutation, ie. accidental changes to the genetic content of one individual.

Elements of genetic algorithms are:
- search space, ie. set of all possible solutions,
- populations, ie. set of actual candidates for the resolution,
- elements of the population are individuals (nodes results, ie. point in search space),
- low space (strings), ie. space representation of individuals,
- as well as functions to map the search engine space in the area are low and vice versa,
- a set of genetic operators to generate new strings,
- and thus new individuals,
- evalulative function (fitness function), which determines the advanta ge (usefulness) certain individuals,
- stochastic control genetic operators.

In a narrow sense, the term genetic algorithm applies only to the model that was introduced by John Holland in his book “Adaption in natural and artificial systems”, 1975 (Holland, 1975). Holland is considered the creator of this meta-heuristic and basic settings of his earliest works are valid even today. More generally, genetic algorithm is any algorithm that is based on a population and operators of selection, crossing and mutation, which are used for production of new points in the space search.

The big problem of genetic algorithms is theoretical – the lack of rigorous mathematical proof that genetic algorithms generate an optimal solution. However, in practice they give satisfactory results, and often there is no need for a theoretical explanation. Usually the quality of a new type of genetic algorithm evaluates empirically, by experimentally tested on a problem of optimization.

Genetic algorithm is applied to the final set of individuals called the population. Each individual in the population is represented by a series of characters (genetic code) and corresponds to a solution in search space. Coding can be binary or of some other higher cardinality alphabet. Encoding solutions is an important step genetic algorithm for inappropriate choice of code can lead to bad results regardless of the rest of the structure of the algorithm.

Instead of working with objects (individuals), genetic algorithms work with parametric descriptions of objects (individuals). Mathematical model is based on the play, and the algorithm process control, reproduction and survival of objects (individuals) that can “compete” in the search for a solution. The algorithm, therefore, is a set of parametric descriptions of objects (individuals), with the population. Genetic operators are applied to individuals and shall be repeated their assessment. Genetic operators provide the offspring are similar but not identical with their parents, which allows the population to evolve to solutions that were not present in the initial set of objects (individuals).
Theory of GAs is based on schemata theory and probability, i.e. mathematical foundations of GA are:
- schema definition,
- hypothesis building blocks,
- theorem on implicit parallelism,
and GA to increase popularity was largely due to the existence of mathematical apparatus, whose use of some results can be predicted and explained.

The strength of the genetic algorithms is the fact that they are able to determine the global optimum position in space with multiple local extremes, the so-called multimodal space. Applying the classical deterministic methods to solve optimization problems, we get results that are always moving towards the local minimum or maximum, where it can be global, but this can not be determined from the results. On the other hand, stochastic methods, including genetic algorithm, not dependent on a possible starting point and can process their results, with some probability, to locate the global optimum of some objective function (Bogdanović, 2011).

Genetic algorithm shows its power at the most complex requirements, because almost sure to converge to the best solution (global minimum or global maximum). Its strength lies in the way in which the variables are varied and searching for a solution. The essence of the optimization procedure is as follows:
- To specify the problem formed by a set of possible solutions (orderly n-tuple variables). This set may contain solutions that are close to the best („works“), and were obtained in some other way, in which case the task of the genetic algorithm to improve them. However, this set is often obtained pseudo-random selection of n-tuples.
- Possible solutions (n-tuples) in some way are encoded in the sequence of binary digits, which we call the candidates, and formed the fitness function, whose role is to perform the inverse action, or to evaluate (assess) the candidates on the basis of specified criteria (objectives optimization). Fitness function actually shows us how close a candidate solution, and based on it shall be the best choice.
- Fitness function is strict. Candidates who are furthest from the solution, it is deleted, while those who show any movement toward the solution, it allows intertwine and give offspring (selection process). Besides selection process, and there are random changes (mutations). In this way, looking for the best solution, the best candidates survive, as in the process of evolution.

Automata Theory is an exciting, theoretical branch of computer science. Automata theory deals with the logic of computation with respect to simple machines, referred to as automata. Automata Theory is the study of self-operating virtual machines to help in logical understanding of input and output process, without or with intermediate stage(s) of computation (or any function/process). Automatons are abstract models of machines that perform computations on an input by moving through a series of states or configurations. The automaton reads
the symbols of the input word one after another and transitions from state to state according to the transition function, until the word is read completely. Once the input word has been read, the automaton is said to have stopped and the state at which automaton has stopped is called the final state. Depending on the final state, it's said that the automaton either accepts or rejects an input word. An automaton is a mathematical object that takes a word as input and decides either to accept it or reject it. Since all computational problems are reducible into the accept/reject question on words (all problem instances can be represented in a finite length of symbols), automata theory plays a crucial role in computational theory.

There are four major families of automaton:

- Finite-state machine
- Pushdown automata
- Linear-bounded automata
- Turing machine

The families of automata above can be interpreted in a hierarchal form, where the finite-state machine is the simplest automata and the Turing machine is the most complex.

The theories of computability and complexity are closely related.

In complexity theory, the objective is to classify problems as easy ones and hard ones.

In computability theory, the classification of problems is by those that are solvable and those that are not. Computability theory introduces several of the concepts used in complexity theory. Automata theory deals with the definitions and properties of mathematical models of computation. The finite automaton is used in text processing, compilers, and hardware design. The context – free grammar, is used in programming languages and artificial intelligence. Turing machines represent computable functions.

2. Genetic algorithms and finite automata

Finite automata are computing devices that accept/recognize regular languages and are used to model operations of many systems we find in practice. Their operations can be simulated by a very simple computer program.

Fogel in (Fogel et all, 1966) created automata with mutation operator, and Lucas (Lucas et al., 2005) designed automata without output function using GA. Collins and Jefferson solved “An Artificial Ant” problem by searching a Meally automaton with constant number of internal states, $n=4$. They used classical GA with stationary length of linear chromosome (Collins et al., 1991).

The paper (Fábera et al., 2006) summarizes the first experience with automata constructed with classical genetic algorithm. It compares classical algorithm with mutation affected by probability matrix that seems to be more successful especially for “chain-like” graphs. Very difficult is to create automaton with tree-based transition graph than a cyclic graph.

In the paper (Niño-Ruiz, 2012) the Metaheuristic Of Deterministic Swapping (MODS) is a local search strategy that explores the feasible solution space of combinatorial problems based on a data structure named Multi Objective Deterministic Finite Automata (MDFA). A MDFA is a Deterministic Finite Automata
that allows the representation of the feasible solution space of combinatorial problems. A MODS explores the feasible solution space represented through a MDFA using a search direction given by an elitist set of solutions.

Simulated Annealing & Metaheuristic Of Deterministic Swapping (Niño et al., 10) (SAMODS) is a hybrid local search strategy based on the MODS theory and Simulated Annealing algorithm for the multi-objective optimization of combinatorial problems. Its main propose consists in optimizing a combinatorial problem using a Search Direction and an Angle Improvement.

Simulated Annealing, Genetic Algorithm & Metaheuristic Of Deterministic Swapping (Niño, 2012) (SAGAMODS) is a hybrid search strategy based on the Automata Theory, Simulated Annealing and Genetics Algorithms. SAGAMODS is an extension of the SAMODS theory. SAGAMODS proposes crossover step before SAMODS template is executed. Due to this, SAGAMODS supports to SAMODS for exploring distant regions of the solution space.

In the same paper (Niño-Ruiz, 2012) Evolutionary Metaheuristic of Deterministic Swapping (EMODS) is a novel framework that allows the Multi-objective Optimization of Combinatorial Problems. Its framework is based on MODS template therefore its steps are the same: create initial solutions, improve the solutions (optional) and execute the core algorithm. Unlike SAMODS and SAGAMODS, EMODS avoids the slowly convergence of Simulated Annealing’s method. EMODS explores different regions from the feasible solution space and search for non-dominated solution using Tabu Search. The experimental results shows that for instances of two, three and four objectives, the proposed algorithm has the best performance as the metrics values corroborate. For the last instance worked, quaint-objective, the behavior of MODS, SAMODS and SAGAMODS tend to be the same, them have similar error rate but, EMODS has a best performance. In all the cases, EMODS shows the best performance. However, for the last test, all the algorithms have different solutions sets of non-dominated solutions, and those form the optimal solution set.

In the paper (Yang et al., 2012), a new method LAGA is used in high level synthesis to deal with scheduling and allocation simultaneously. It can produce area and performance optimized designs. This algorithm utilizes Genetic Algorithm and Learning Automata methods sequentially to search for the mode space. It can find the Solutions quickly by using Genetic Algorithm and Learning Automata sequentially in search process. The experimental results indicate that LAGA algorithm is very effective in high-level synthesis, especially under the area constraint.

In the paper (Lai et al., 2006) on developed a dynamic approach for automatically generating formal models from real systems, whose internal structure is totally unknown. That learning process is under the adaptive model checking framework, and the core learning algorithm is based on genetic algorithm. This method combines techniques from model checking, conformance testing and model learning theory. This approach can either build a model from scratch or iteratively refine it each time model checking requests are submitted, rather than learning a perfect model in one trial. Experimental results show that it has the potential to handle large systems as to the memory consumption aspect.
The paper (Takkar, 2014) describes and analyzes the behavior of an implementation of a non-deterministic finite-state acceptor using a genetic algorithm. There are at least three different algorithms that decide if and how a given regular expression matches a string. The oldest and fastest relies on a result in formal language theory that allows every nondeterministic finite automaton (NFA) to be transformed into a deterministic finite automaton (DFA). The DFA can be constructed explicitly and then run on the resulting input string one symbol at a time. An alternative approach is to simulate the NFA directly, essentially building each DFA state on demand and then discarding it at the next step. This keeps the DFA implicit and avoids the exponential construction cost, but running cost rises to $O(m \cdot n)$. Third algorithm is to match the pattern against the input string by backtracking. Its running time can be exponential.

3. Possibilities for applications of genetic algorithms for the directable automata

The automata considered throughout the paper are automata without outputs, in the sense of the definition given by F. Gécseg and I. Peák in (Gécseg and Peák, 1972), and we call them simply automata. Therefore, an automata is defined as a triple $(A, X, \delta)$, where $A$ and $X$ are non-empty sets, not necessarily finite. $A$ is called the set of states and $X$ is called the input alphabet, and $\delta: A \times X \rightarrow A$ is the transition function of this automata. All automata that will be considered in the paper will have the same input alphabet $X$ with $|X| \geq 2$.

The free monoid on the free semigroup over $X$ are denoted by $X^*$ and $X^+$, respectively. The length of a word $u \in X^*$ denoted by $|u|$. For any $k \in N^0$, the subsets $X^k, X^{\leq k}$ and $X^{\geq k}$ of $X^*$ are defined by $X^k = \{u \in X^*||u| = k\}$, $X^{\leq k} = \{u \in X^*||u| \leq k\}$ and $X^{\geq k} = \{u \in X^*||u| \geq k\}$.

Under the action of an input word $u \in X^*$, the automata $A$ goes from a state $a$ into the state denoted by $au$.

An automata $A$ is called $u$-directable for a given word $u \in X^*$ if $au = bu$, for all $a, b \in A$. In that case the word $u$ is called directing word of $A$. Furthermore, for automata $A$ is said to be directable if there exists a word $u \in X^*$ such taht $A$ is $u$-directable. We may say that the directing word $u$ directs the states of the automata $A$ into a single state that will be called a $u$-neck of $A$ and denoted by $d_u$. A state $d \in A$ is called a neck of $A$ if there exists $u \in X^*$ such that $d$ is a $u$-neck of $A$.

Now, we will give two ways to specialize notation of a directable automata. First, for a given number $k \in N_0$, an automata $A$ is called $k$-definite if each word from $X^{\leq k}$ is a directing word of $A$. An automata $A$ is said to be definite if there exists $k \in N_0$ such that $A$ is $k$-definite. The smallest number $k \in N_0$ for which $A$ is $k$-definite is called the degree of definiteness of $A$. The $1$-definite automata is called reset automata.

Furthermore, if $u \in X^*$ such that $A$ is $u$-directable and as a trap $a_0$, then $a_0$ is both the unique trap and the unique neck of $A$. In this case the word $u$ directs the states of $A$ into to trap $a_0$ and the automata $A$ is called trap- $u$-directable,
since the word \( u \) is called a \textit{trap-directing} word of \( A \). Similarly, an automata \( A \) is said to be \textit{trap-directable} if there exists a word \( u \in X^* \) such that \( A \) is trap- \( u \)-directable. For a given word \( u \in X^* \) an automata is called \textit{\( u \)-trapped} if \( au \in Tr(A) \), for all \( a \in A \). In this case the word \( u \) is called a \textit{trapping word} of \( A \). An automata \( A \) is called a \textit{trapped automata} if there exists a word \( u \in X^* \) such that \( A \) is \( u \)-trapped.

In the paper (Olschewski et al., 2010), the authors study several problems related to finding reset words in deterministic finite automata. They have investigated several decision problems and search problems about finding reset words in finite automata. In particular, deciding whether for a given automaton a shortest reset word has length \( k \) is DP-complete, and computing the length of a shortest reset word is \( FP^{NP[\log]} \)-complete, i.e. as hard as calculating the maximum number of simultaneously satisfiable clauses of a Boolean formula. The most fundamental decision problem concerning reset words is to decide whether a given deterministic finite automaton is synchronising. Computing a shortest reset word, however, cannot be done in polynomial time unless the following decision problems are in \( P \). In the paper (Olschewski et al., 2010) the authors reviewed the following decision problems:

\textbf{Short-reset-word}: Given a DFA \( A \) and a positive integer \( k \), decide whether there exists a reset word for \( A \) of length \( k \).

\textbf{Shortest-reset-word}: Given a DFA \( A \) and a positive integer \( k \), decide whether the minimum length of a reset word for \( A \) equals \( k \).

Eppstein (Eppstein et al., 1990) gave a matching lower bound by proving that short-reset-word is also NP-hard. Regarding shortest-reset-word, Samotij (Samotij, 2007) showed that the problem is NP-hard. The authors of the paper (Olschewski et al., 2010) showed that shortest-reset-word is DP-hard even over a binary alphabet. Therefore, these two decision problems could be solved by designing a genetic algorithm as metaheuristics for optimization.

The authors of the paper (Olschewski et al., 2010) also talk about search problems and prove that these problems are due to its complexity, popular for solving using optimization methods. More precisely, they deal with the following search problems:

Given a DFA \( A \) and a positive integer \( k \) in unary, \textit{compute a reset word} for \( A \) of length \( k \).

Given a DFA \( A \), \textit{compute the length of a shortest reset word} for \( A \).

Given a DFA \( A \), \textit{compute a shortest reset word} for \( A \).

\textbf{Proposition 3.1}. (Olschewski et al., 2010) \textit{The problem of computing a reset word of a given length is FNP-complete.}

\textbf{Theorem 3.2}. (Olschewski et al., 2010) \textit{The problem of computing the length of a shortest reset word is }\textit{FP}^{NP[\log]}\textit{-complete.}

\textbf{Theorem 3.3}. (Olschewski et al., 2010) \textit{The problem of computing a shortest reset word is in }\textit{FP}^{NP}\textit{ and hard for }\textit{FP}^{NP[\log]}.
And when we consider the trapped automata, as a special class of the directable automata, we can define a similar decision problems and search problems. These problems can be formulated as follows:

1. Finding trapping word for the trapped automata $A$;
2. Finding a shortest trapping word for the trapped automata $A$;
3. Given a DFA $A$ and a positive integer $k$ in unary compute a trapping word for $A$ of length $k$;
4. Given a DFA $A$ compute the length of a shortest trapping word for $A$;
5. Given a DFA $A$ compute a shortest trapping word for $A$.

In the paper (Popović et al., 2002) Popović and other gave the algorithms for testing the membership of a finite automata to the pseudovarieties of finite trapped, locally trap-directable and trap-directable automata. One of these algorithms determines the minimum length of trapping word for $A$. For a directable automata $A$ the number $d(A)$ is defined by $d(A) = \min\{|u||u \in DW(A)\}$, and for $n \in N$, the number $d(n)$ is defined as

$$d(n) = \max\{d(A)|A \text{ is directable automata with } |A| = n\}.$$

The Černý’s conjecture says that $d(n) \leq (n - 1)^2$, but in the case of the trapping automata Popović and other have proved the following estimation:

**Lemma 3.4.** (Popović et al., 2002) Let $A$ be a trapped automata with $n$ states and $t$ traps. Then the minimal length of trapping words of $A$ is less or equal than $d(n - t + 1)$.

Their results can serve as a good basis for creating a genetic algorithm that will resolve decision problems and search problems for the trapping automata. This will be the direction in which we move into the future work.

In the paper (Kisielewicz et al., 2013) on present a new fast algorithm finding minimal reset words for finite synchronizing automata. The problem is known to be computationally hard, and their algorithm is exponential. Yet, it is faster than the algorithms used so far and it works well in practice. The main idea is to use a bidirectional BFS and radix (Patricia) tries to store and compare resulted subsets. Its time performance depends mainly on the length of the minimum word, but one has to observe that the known slowly synchronizing automata with the longest reset words are not the hardest ones for the algorithm. Thus, there is a hope to discover new examples of classes of slowly synchronizing automata and to check how the situation changes in case of automata with more than 2 input letters. There are still possibilities to optimize the algorithm, in particular, by designing a faster data structure for subset checking and minimizing subset lists.

5. Conclusions

Genetic algorithms are a family of algorithms that use some kind of genetic principles that are present in nature, in order to solve specific computational problems. During more than four decades, especially in recent years, genetic algorithms (GA) have proved very powerful and also very general tool for solving problems in the field of learning in neural networks, in finding the shortest path, the traveling salesman problem, in strategy games, in problems similar to the
transport problem, the problem of the deployment process, the problem of determining the parameters of the system, the optimization of queries on the database and so on. These algorithms can be used for solving various classes of problems because they are fairly general nature. Initially, the genetic algorithms used mainly for solving optimization problems, but now they use for solving various problems, which are basically problems of optimization or search problems. The reason is that genetic algorithms can be relatively easily adapted to the specific problem. These algorithms find their application in the organization, chemistry, biology, neuroscience, technology, automata theory.

6. Future Work

One interesting, relatively new area of application of genetic algorithms is the theory of automata. Some of these applications are listed in this paper. We especially were interesting opportunities provided by genetic algorithms in solving problems of the directable automata. Future work will focus on problem solving and decision-search problems in various types of directable automata using the techniques of genetic algorithm.

References


Доц. др Милена Богдановић

ПРЕГЛЕД НЕКИХ МОГУЋНОСТИ КОРИШЋЕЊА ГЕНЕТСКИХ АЛГОРИТАМА У ТЕОРИЈИ АУТОМАТА

Резиме: Једно је сигурно – порекло живота је у природи, која то најбоље зна. Генетски алгоритми (ГА) су породица алгоритама који користе неку врсту генетских принципа који се налазе у природи, у циљу решавања конкретних рачунарских проблема. Генетски алгоритми су робусне и прилагодљиве методе које се поједина областима примене могу користити за решавање комбинаторних оптимизационих проблема. Разлог је у томе што се генетски
алгоритми могу релативно лако прилагодити специфичним проблемима. Автомати су већ изграђивани еволутивним техникама 60-их година прошлог века од стране Фогела. Он је створио „еволутивно програмирање“, што је био први покушај да се изгради један аутомат еволутивним приступом. Циљ овог рада је био да се додатно демонстрира и прикаже могућност примене ГА у теорији аутомата.

Кључне речи: генетски алгоритам, комбинаторијална оптимизација, НП проблеми, коначни аутомат, решетујућа реч, управљива реч.

Примљено: 8. 8. 2015. године.