Proceedings of the Estonian Academy of Sciences,
2013, **62**, 1, 39–46
doi: 10.3176/proc.2013.1.05
Available online at www.eap.ee/proceedings

COMPUTER SCIENCE

An approach to the inference of finite state machines based on a gravitationally-inspired search algorithm

Margarita Spichakova

Institute of Cybernetics at Tallinn University of Technology, Akadeemia tee 21, 12618 Tallinn, Estonia; margo@cs.ioc.ee

Received 31 August 2011, revised 9 April 2012, accepted 21 September 2012, available online 20 February 2013

Abstract. As the inference of a finite state machine from samples of its behaviour is NP-hard, heuristic search algorithms need to be applied. In this article we propose a methodology based on applying a new gravitationally-inspired heuristic search algorithm for the inference of Moore machines. Binary representation of a Moore machine, an evaluation function, and the required parameters of the algorithm are presented. The experimental results show that this method has a lot of potential.

Key words: finite state machine, gravitational search algorithm, system identification.

1. INTRODUCTION

Identification is an inference process, which deduces an internal representation of a system (named *internal model*) from samples of its functioning (named *external model*) [1]. The inference of finite state machines (FSMs) is widely applied in different fields, such as logical design, verification, and software systems.

The goal of identification is to find the 'best' FSM, which respects the dynamics of the external model. In practice, the 'best' FSM is the one that best describes the model behaviour given by input–output sequences. We are interested in finding a minimum size deterministic FSM consistent with the set of the given samples. This is an NP-hard problem [2]. Heuristic algorithms are an alternative that can reduce the complexity of the identification methods.

The paper is organized as follows. Section 2 provides an overview of the problem of FSM inference. Section 3 describes gravitationally-inspired search algorithms. Section 4 introduces our approach, and Section 5 shows experimental results of the work.

2. INFERENCE OF FINITE STATE MACHINES

2.1. Problem statement

We give a brief overview of our approach to FSM identification. There are several types of FSMs, but in this article we will discuss only one well-known representation of them, namely the Moore machines.

A *Moore machine* is a six-tuple Mo $= \langle Q, \Sigma, \Delta, \delta, \lambda, q_0 \rangle$, where

- Q is a finite set of states, where q_0 denotes the initial state,
- Σ is the input alphabet,
- Δ is the output alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $\lambda : Q \to \Delta$ is the output function represented by the output table that shows what character from Δ will be printed by each state that is entered [3].

The general structure of our approach to the inference of FSMs is presented in Fig. 1.

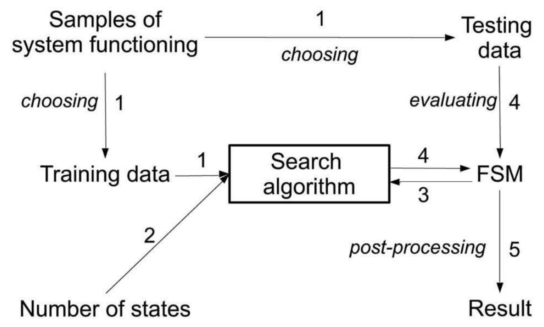

Fig. 1. Problem of system identification.

The outline of our approach is the following:

1. The system to be inferred is tested and samples of its functioning are generated. Some of the samples are chosen as *training data* and some as *testing data*.
2. The number of states in the FSM is received as an input.
3. The search algorithm is applied and a FSM M is outputted.
4. M is evaluated using the given training data and/or testing data. If M describes the given input–output data sufficiently well, it is considered as result. Otherwise the search process with other parameters or training data will be repeated.
5. If required, post-processing (e.g., minimization, reduction of unreachable states) is applied.

It is possible to specify several criteria for the required result. The first criterion is consistency of the FSM. Using this criterion, we can define two different types of solutions: the *generalized solution* (i.e., the solution that performs correctly for all positive input–output sequences) and the *consistent solution* (i.e., the solution that performs correctly for the input–output sequences used in the training set). Another criterion is the FSM size. We can search for the minimal FSM or a FSM with k or fewer states.

We formulate our goal as the inference of a *deterministic FSM with k or fewer states, consistent with input–output sequences at hand.*

2.2. Background

Heuristic techniques are widely applied to the inference of different types of FSMs. The most popular are the various types of Evolutionary Algorithms. In the early 1960s Fogel et al. [4] introduced Evolutionary Programming (EP). The simulated evolution was performed by modifying a population of FSM. Other authors also used EP for solving the problem of FSM identification. Chellapilla and Czarnecki [5] proposed the variation of EP to solve the problem of modular FSM synthesis. Benson [6] presented a model comprising an FSM with embedded genetic programs which co-evolve to perform the task of Automatic Target Detection.

Another approach to solve the problem of FSM identification is based on the Genetic Algorithm (GA). This method has been researched by several authors. Ngom et al. [7] used genetic simulation for Moore machine identification, Tongchim and Chongstitvatana [8] investigated parallel implementation of the GA to solve the problem of FSM synthesis. Lucas [9] paid more attention to finite state transducers and he and Reynolds [10] compared this method to 'Heuristic State Merging'. Niparnan and Chongstitvatana [11] improved GA by evolving only the state transition function. Chongstitvatana and Aporntewan [12] presented a method of FSM synthesis from multiple partial input/output sequences. Horihan and Lu [13] focused on improving the FSM evolution by using progressive fitness functions. Also Generated Simulated Annealing was used for the inference of FSM [14].

We apply a gravitationally-inspired search algorithm. The next section describes the general ideas of this new class of algorithms.

3. GRAVITATIONALLY-INSPIRED SEARCH ALGORITHM

3.1. Gravity as inspiration for heuristic search algorithms

Four main forces are acting in our universe: gravitational, electromagnetic, weak nuclear, and strong nuclear. These forces define the way our universe behaves and appears. The weakest force is gravitational; it defines how objects move depending on their mass. In physics three kinds of masses can be distinguished (active mass M_{a}, passive mass M_{p}, and inertial mass M_{i}), which have been shown experimentally to be equivalent (see [15]).

The gravitational force between two objects i and j is directly proportional to the product of their masses and inversely proportional to the square distance between them

$$F_{ij} = G\frac{M_{\mathrm{a}j} \cdot M_{\mathrm{p}i}}{R_{ij}^2}. \tag{1}$$

Knowing the force acting on a body we can compute acceleration as

$$a_i = \frac{F_i}{M_{\mathrm{i}i}}. \tag{2}$$

Our universe is growing, this yields an effect of decreasing gravity, so the gravitational 'constant' can be described as

$$G(t) = G(t_0) \cdot \left(\frac{t_0}{t}\right)^{\beta}, \beta < 1. \tag{3}$$

We can formulate the following basic ideas inspired by gravity:

- Each object in the universe has mass and position.
- There are some interactions between objects, which can be described using the law of gravity.
- Bigger objects create larger gravitational fields and attract smaller ones.

During the last decade some researchers have tried to adapt the idea of gravity to find out optimal search algorithms. Such algorithms have some general ideas in common:

- The system is modelled by objects with mass.
- The position of those objects describes the solution, and the mass of the objects depends on the objective function.
- The objects interact with one another using gravitational force.
- The objects with greater mass present the points in the search space with better solutions.

Using these characteristics, it is possible to define the family of optimization algorithms based on gravitational force. For example, *Central Force Optimization* (CFO) is a deterministic gravity-based search algorithm proposed and developed by Formato [16]. It simulates the group of probes that fly into search space and explore it. Another algorithm, *Space Gravitational Optimization* (SGO), was developed by Hsiao et al. [17] in 2005. It simulates asteroids flying through curved search space. A gravitationally-inspired variation of local search, *Gravitational Emulation Local Search Algorithm* (GELS), was proposed by Webster and Bernhard [18] and further elaborated by Webster [19]. The newest one, *Gravitational Search Algorithm* (GSA), was described by Rashedi et al. [20] as a stochastic variation of CFO.

The next subsection will give a more detailed overview of the GSA, which is used as a basis of our approach.

3.2. Gravitational search algorithms

The GSA was described by Rashedi et al. [20] as a stochastic variation of the CFO and used for different applications. It was successfully applied to optimize various continuous problems, such as filter modelling [21], the set covering problem [22], allocation of static var compensator [15], and synthesis of thinned scanned concentric ring array antenna [23].

The algorithm is constructed so that there is a system of N objects, each of which is described by a real-valued position vector, and each position vector codes candidate solution

$$X_i = \left(x_i^1, \dots, x_i^d, \dots, x_i^n\right), d \in [1 \dots n], \tag{4}$$

where x_i^d represents the position of the ith object in dimension d.

Masses of objects are computed based on the quality measure as follows:

$$M_{\mathrm{a}i} = M_{\mathrm{p}i} = M_{\mathrm{i}i} = M_i,\ i \in [1, 2, \dots N], \tag{5}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)},\ m_i = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)}, \tag{6}$$

where $worst(t)$ and $best(t)$ are defined for maximization problem as

$$best(t) = \underbrace{\max}_{j \in [1 \dots N]} fit_j(t),\ worst(t) = \underbrace{\min}_{j \in [1 \dots N]} fit_j(t),$$

and fit_i is the value of the objective function.

In other words, a heavier mass means that the quality of the object is better and it has greater attraction and inertia (i.e., moves slowly towards other objects).

At a specific time t we can recompute the force that is applied to the object i with mass M_i by some object j with mass M_j

$$F_{ij}^d(t) = G(t) \frac{M_{\mathrm{p}i}(t) \cdot M_{\mathrm{a}j}(t)}{R_{ij} + \varepsilon} (x_j^d - x_i^d), \tag{7}$$

where ε is a free parameter, required to avoid division by zero, and R_{ij} is the Euclidean distance between position vectors:

$$R_{ij} = \left\| X_i(t), X_j(t) \right\|. \tag{8}$$

According to Rashedi et al. [15], R_{ij} gives better experimental results than R_{ij}^2.

The gravitational constant G (Eq. (3)) is computed as

$$G(t) = G(G_0, t). \tag{9}$$

In physics, the general force acting on an object is computed as a vector sum of all acting forces. In the GSA, a stochastic characteristic is added to the algorithm, so the general force is computed as

$$F_i^d(t) = \sum_{j=1, i \neq j}^{N} rand_j \cdot F_{ij}^d(t),\ rand_j \in [0, 1]. \tag{10}$$

The acceleration of object i can be computed knowing its inertial mass $M_{\mathrm{i}i}$ and force $F_i^d(t)$ as

$$a_i^d(t) = \frac{F_i^d(t)}{M_{\mathrm{i}i}(t)}. \tag{11}$$

Knowing current acceleration, we can recompute velocity and position as follows:

$$v_i^d(t+1) = rand_i v_i^d(t) + a_i^d(t),\ rand_i \in [0 \dots 1]; \tag{12}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1). \tag{13}$$

The general procedure of the GSA is described in Algorithm 1. Firstly, the initial set of objects is generated randomly. Secondly, each object is evaluated. Based on evaluation results, the required parameters ($G(t)$, $worst(t)$, $best(t)$) are updated, and the forces and accelerations are computed. Thirdly, the agents' positions are changed according to acting forces and the updated positions are evaluated. The process continues until the best solution is found or the number of iterations is over.

Algorithm 1. General procedure of GSA

```
Generate initial positions
repeat
  Evaluate quality of each object
  Update $G(t)$, $worst(t)$, $best(t)$
  Calculate masses and accelerations
  Calculate velocities and positions
until meeting ending criterion
Return best solution
```

In the GSA, the position vector is real-valued. However, for some applications discrete or binary vectors are required. A discrete modification of the algorithm was proposed by Zibanezhad et al. [24] in a context of Web-Service composition. The binary GSA (BGSA) was introduced by Rashedi et al. [25] in 2010. In the next section we will focus on the BGSA.

3.3. Binary gravitational search algorithm

The key difference between the GSA and the BGSA is the binary search space, meaning that each dimension has only two possible values: '0' or '1'. The main laws of the BGSA may be defined as in real-valued case (see Eqs (7), (11), and (12)). But the positions' updating law (see Eq. (13)) must be modified so that each dimension changes between two values according to the velocity. A higher velocity gives a greater probability of changing the value.

To modify Eq. (13), in the BGSA a special probability function $S(v_i^d)$ was introduced, which transfers the value of v_i^d to $[0\dots1]$:

$$S(v_i^d) = \left|\tanh(v_i^d)\right|. \tag{14}$$

The law for updating the position can be defined as follows:

$$x_i^d(t+1) = \begin{cases} F(x_i^d(t)) & \text{if } rand < S(v_i^d(t+1)), \\ x_i^d(t) & \text{if } rand \geq S(v_i^d(t+1)), \end{cases} \tag{15}$$

where $F(x_i^d(t)) = complement(x_i^d(t))$. Some other modifications were made:

- Velocity v_i^d is bounded: $\left|v_i^d\right| < v_{\max}$.
- Distance R is computed as the Hamming distance.
- Gravitational constant G is considered as a linear decreasing function

$$G(t) = G_0(1 - t/T). \tag{16}$$

4. GRAVITATIONALLY-INSPIRED SEARCH ALGORITHM FOR THE INFERENCE OF FSMs

To apply the BGSA to the inference of FSMs we need to define an objective function and a process of encoding FSMs to a binary position vector. Also modifications of the original BGSA have to be made.

4.1. Representation of an FSM

We discuss only Moore machines with exactly n states. Consider a target machine Mo with n states, input alphabet $\Sigma = \{i_0, \dots, i_{l-1}\}$, output alphabet $\Delta = \{o_0, \dots, o_{m-1}\}$, and set of states $Q = \{q_0, \dots, q_{n-1}\}$.

To store the information about state q_j, we need to store the output value o^j of the state and corresponding transitions from the given state q_j to get some target state q^{i_k}, which are activated by reading symbol i_k. Each section represents one state (Fig. 2), where the first part is an output value of the state and the other part stores the corresponding transitions from that state. Initially, information is presented in a decimal way (decimal representation). To get binary representation we transform each integer number to the corresponding binary number.

The number of bits required for storing the whole binary position vector can be computed as follows:

$$Length = n \cdot (\lceil \log_2 m \rceil + l \lceil \log_2 n \rceil). \tag{17}$$

Each Mo has a unique binary representation, but not each binary string has a corresponding Mo.

Let us take a look at a Moore machine with the transition diagram presented in Fig. 3.

We have four states $Q = \{0, 1, 2, 3\}$, the input alphabet contains two symbols $\Sigma = \{a, b\}$, and the output alphabet two symbols $\Delta = \{0, 1\}$.

Thus we need 20 bits to store this FSM (Eq. (17)): $4 \cdot (\lceil \log_2 2 \rceil + \lceil \log_2 4 \rceil \cdot 2) = 20$ bits. The general structure of the position vector required to encode this FSM is presented in Fig. 4.

State q_j			
o^j	q^{i_0}	$q^{\cdots}$	$q^{i_{k-1}}$

Fig. 2. A section of the binary position vector for storing the Moore machine with a fixed number of states.

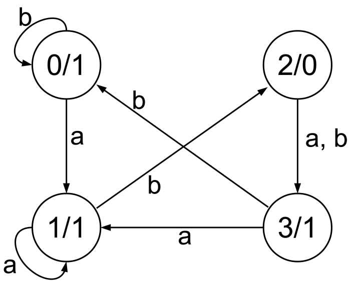

Fig. 3. A Moore machine represented as a transition diagram.

	a	b		a	b		a	b		a	b	
1	1	0	1	1	2	0	3	3	1	1	0	Dec. representation
1	01	00	1	01	10	0	11	11	1	01	00	Bin. representation

Fig. 4. Example. Binary position vector for storing the Moore machine.

4.2. An objective function

We propose an objective function defined on all input–output sequences (pairs {input, output}). The idea is to estimate the proximity between the current and the desired FSMs by finding the distance between strings.

4.2.1. Distance between strings

Consider a function $\Delta(a,b)$, where a,b are symbols in some alphabet, and define

$$\Delta(a,b) = \begin{cases} 0 & : a = b, \\ 1 & : a \neq b. \end{cases} \tag{18}$$

That is, if character a is not equal to character b, the function $\Delta(a,b)$ will return 1, otherwise the function will return 0.

We propose two distance functions between strings x and y. The first function is the *Hamming distance* d_{Ham}. To compute it, we need to count the number of different bits in the same positions

$$d_{\text{Ham}}(x,y) = \Sigma_{i=1}^{\min(|x|,|y|)} \Delta(x_i, y_i). \tag{19}$$

The second function evaluates the *length of maximal equal prefix* d_{LP} (i.e., the computation will be stopped at the first difference between strings)

$$d_{\text{LP}}(x,y) = \Sigma_{i=1}^{x=y} \Delta(x_i, y_i). \tag{20}$$

4.2.2. Evaluation of the objective function

We specify several objective functions for evaluating FSMs based on d_{Ham} and d_{LP}. Assume we have our training data represented as a collection of input–output sequences (the size of the collection is n). We also have output strings produced by an FSM (see Table 1).

Our task is to measure how ‘far’ the strings generated by the FSM are from the expected strings. The objective function based on the Hamming distance (d_{Ham}) defines the *objective function* as follows:

$$OF = \Sigma_{i=1}^{n} \left(l_i - d_{\text{Ham}} \left(\text{Out}_i^{\text{expected}}, \text{Out}_i^{\text{produced}} \right) \right), \tag{21}$$

where n is the number of the given data and l_i is the length of $\text{Out}_i^{\text{expected}}$.

Table 1. Measuring the value of the objective function

Input	Expected output	Produced output	Distance
In_0	$\text{Out}_0^{\text{expected}}$	$\text{Out}_0^{\text{produced}}$	dist._0
In_1	$\text{Out}_1^{\text{expected}}$	$\text{Out}_1^{\text{produced}}$	dist._1
...	...	...	...
In_n	$\text{Out}_n^{\text{expected}}$	$\text{Out}_n^{\text{produced}}$	dist._n

In the second case we use d_{LP} for measuring the distance. Thus, the *objective function* can be defined as the sum of the lengths of all sequences

$$OF = \Sigma_{i=1}^{n} \left(d_{\text{LP}} \left(\text{Out}_i^{\text{expected}}, \text{Out}_i^{\text{produced}} \right) \right). \tag{22}$$

4.3. Algorithm description

In this section we will focus on the properties of our algorithm. Search space is described by a set of binary position vectors, where each position vector corresponds to an FSM as described in Section 4.1.

First, we set
- the free parameter ε,
- the maximal speed $v_{\max}$,
- the number of iterations,
- the number of objects,
- the number of states n in the FSM,
- the initial value of gravitational constant G_0,
- the mass value minimum $M_{\min}$

according to the problem under consideration.

The initial positions are generated randomly from the feasible region, so that each position corresponds to an FSM. To do so, the FSM is generated in decimal form, a number of symbols in input and output alphabet are restored from the input data. After generating the FSM in decimal form it is encoded into binary representation (see Section 4.1).

The objective function of a candidate solution is computed as described in Subsection 4.2.2. Despite the fact that in physics the active, passive, and inertial masses are considered to be equivalent (see 3.1), we modified mass computation laws to improve the search algorithm. The active M_{a}, passive M_{p}, and inertial M_{i} masses are computed as follows

$$M_{\text{p}} = M_{\text{i}} = \frac{OF}{OF_{\max}}, \tag{23}$$

$$M_{\text{a}} = \begin{cases} M_{\text{i}} & \text{if } M_{\text{a}} > M_{\min}, \\ 0 & \text{if } M_{\text{a}} \leq M_{\min}. \end{cases} \tag{24}$$

If M_{a} is smaller than the minimum value $M_{\min}$ of the defined mass, then $M_{\text{a}} = 0$ (i.e., an object with a smaller mass does not create a gravitational field).

Forces acting on the object are computed via Eq. (7). The distance in one dimension can be computed as follows:

$$(x_j^d - x_i^d) = \begin{cases} 1 & \text{if } x_j^d \neq x_i^d, \\ -1 & \text{if } x_j^d = x_i^d. \end{cases} \tag{25}$$

The acceleration vector is computed via Eq. (11). The velocity vector is computed by Eq. (12). If the velocity is higher than $v_{\max}$, then its value will be set to $v_{\max}$.

The new position is computed using the old position and the velocity vector (see Eq. (15)). The probability function (i.e., the threshold function) $S(v_i^d)$ is taken as

$$S(v_i^d) = \left|\sin(v_i^d)\right|, \qquad (26)$$

in this case $v_{\max} = \pi/2$.

5. IMPLEMENTATION AND EXPERIMENTS

Our approach was implemented in Java (JDK 1.5) and tested on random machines and some 'toy' examples. Results are compared to the canonical Genetic Algorithm (more details about GA can be found in [10,26]).

5.1. Experiments I

Experiments were constructed so that general parameters, such as the number of iterations and the number of objects, the encoding of the Moore machine, and its initialization algorithm are the same (see Subsection 4.1). Evaluation of the machine is described in Subsection 4.2; the objective function is constructed on the Hamming similarity. The specific parameters of the algorithm are described for a concrete experiment in the corresponding table.

During the experiments, each algorithm was run 20 times with a different initial set of objects. Results are presented in Table 2 and Table 3, where the row 'Init. %' shows the mass value of the best solution at the initial step (randomly generated), the row 'Sol. %' shows the object value of the best found solution, and the row 'Iter.' shows how many iterations were required to find this solution ('–' means that the best possible solution was not found).

5.1.1. Pattern recognizer

The goal of this experiment was to reconstruct a pattern 'aab' recognizer (see Table 2) from the given input–output pairs. As input data we use six pairs with each input string having a length of 12. The number of states n is four. The number of iterations is taken 100, and the number of objects equals 200.

This experiment showed that the BGSA was more frequently able to find 100% solutions than the GA (10/20 compared to 7/20 for GA) and fewer iterations were required to find them.

5.1.2. Parity checker

The goal of this experiment was to reconstruct a parity checker (see Table 3) from the given input–output pairs. As input data we use seven pairs with length 8 of each input string. The number of states n equals two. The number of iterations is taken 20, and the number of objects equals five.

This experiment showed that the BGSA was more frequently able to find 100% solutions than the GA (14/20 compared to 10/20 for GA) and fewer iterations were required to find them. In three out of twenty cases the GA was not able to improve the maximal solution that was randomly generated in the initial population; for the GSA this happened only in one case out of twenty.

Table 2. Experiment I.1 'Pattern recognizer'

BGSA, $\varepsilon = 0.01$, $G_0 = 100$, $M_{\min} = 0.1$																	
Init. %	92	94	92	95	87	92	89	87	91	87	92	89	89	96	92	86	98
Sol. %	100	99	99	100	99	100	100	100	100	100	100	100	98	98	100	98	98
Iter.	66	–	–	2	–	32	70	14	2	38	2	75	–	–	1	–	–
GA, $P_{\text{mutation}} = 0.04$, roulette wheel selection, one point crossover																	
Init. %	98	87	87	95	87	87	87	87	92	96	87	87	92	93	87	87	92
Sol. %	100	100	100	95	100	92	92	91	100	96	100	96	94	99	91	92	96
Iter.	75	13	65	–	52	–	–	–	14	–	82	–	–	–	–	–	–

Table 3. Experiment I.2 'Parity checker'

BGSA, $\varepsilon = 0.01$, $G_0 = 10$, $M_{\min} = 0.1$																	
Init. %	49	55	65	65	55	55	55	55	55	55	65	55	61	55	57	55	55
Sol. %	100	65	100	65	65	100	100	65	100	100	100	65	100	100	65	100	100
Iter.	1	–	9	–	–	11	15	–	14	14	6	–	9	7	–	15	1
GA, $P_{\text{mutation}} = 0.04$, roulette wheel selection, one point crossover																	
Init. %	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55
Sol. %	100	65	100	55	100	65	100	100	100	65	100	65	65	100	100	55	55
Iter.	12	–	11	–	8	–	12	7	5	–	15	–	–	13	5	–	–

5.2. Experiments II

The goal of those experiments was to compare the BGSA and GA for the same random initial set of objects. Tasks were taken as in the previous experiments (i.e., 'pattern recognizer' and 'parity checker'). Those experiments were constructed in such a way that the initial population was taken the same for both algorithms, the parameters such as the number of iterations and the number of objects were also equal for both algorithms, and are described in Subsection 5.1. Each algorithm (BGSA and GA) was executed 10 times with the same initial set of objects as in Experiments I (5.1). The average best-so-far solutions are presented in Fig. 5. According to the results of this experiment, in the case of 'pattern recognizer' the BGSA solves the task better than the GA (Fig. 5a). For the second task, 'parity checker' (Fig. 5b), the BGSA behaves almost like the GA.

6. CONCLUSIONS AND FUTURE WORK

In this paper we presented a method for the inference of Moore machines based on a gravitationally-inspired search algorithm. Binary representation of FSMs and different types of objective functions were introduced. Parameters and variations of the proposed algorithm were discussed. The proposed approach was implemented and successfully tested using random data and different examples. During the first experiments, our approach gave promising results.

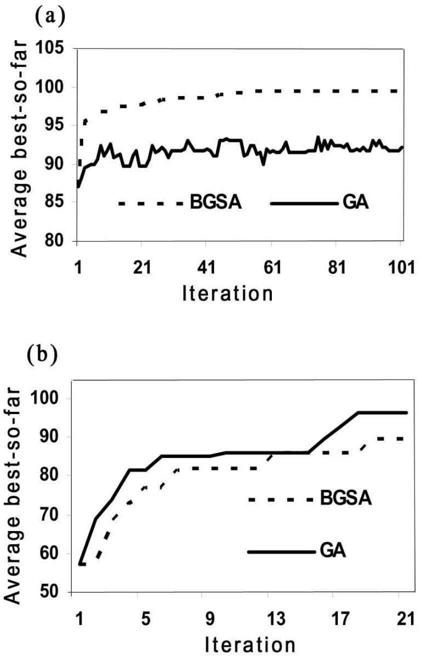

Fig. 5. Comparison between the BGSA and GA: (a) pattern recognizer, (b) parity checker.

To improve the quality of the proposed approach, parameters of the algorithm and their effect on the presented methods will be explored. The effect of using different aspects of laws will be investigated. During further developments the proposed method will be adjusted to take into account other types of FSM, for example the Mealy machines.

ACKNOWLEDGEMENTS

This research was supported by the Estonian Ministry of Education and Research target-financed research theme No. 0140007s12.

REFERENCES

1. Angluin, D. and Smith, C. H. Inductive inference: theory and methods. *ACM Comput. Surv.*, 1983, **15**, 237–269.
2. Gold, E. M. Complexity of automaton identification from given data. *Inform. Control*, 1978, **37**(3), 302–320.
3. Hopcroft, J. E., Motwani, R., and Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation*. International Edition (2nd edn). Addison-Wesley, 2003.
4. Fogel, L. J., Owens, A. J., and Walsh, M. J. *Artificial Intelligence Through Simulated Evolution*. Wiley, Chichester, UK, 1966.
5. Chellapilla, K. and Czarnecki, D. A preliminary investigation into evolving modular finite state machines. In *Proceedings of the 1999 Congress on Evolutionary Computation*. Vol. 2. IEEE Press, 1999, 1349–1356.
6. Benson, K. A. Evolving finite state machines with embedded genetic programming for automatic target detection within SAR imagery. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*. IEEE Press, 2000, 1543–1549.
7. Ngom, L., Baron, C., and Geffroy, J. Genetic simulation for finite state machine identification. In *SS '99: Proceedings of the Thirty-Second Annual Simulation Symposium*. IEEE Computer Society, Washington, DC, USA, 1999, 118.
8. Tongchim, S. and Chongstitvatana, P. Parallel genetic algorithm for finite state machine synthesis from input/output sequences. In *Evolutionary Computation and Parallel Processing* (Cantu-Paz, E. and Punch, B., eds). Las Vegas, Nevada, USA, 2000, 20–25.
9. Lucas, S. M. Evolving finite state transducers: some initial explorations. In *EuroGP*. 2003, 130–141.
10. Lucas, S. M. and Reynolds, T. J. Learning finite state transducers: evolution versus heuristic state merging. *IEEE T. Evolut. Comput.*, 2007, **7**, 308–325.
11. Niparnan, N. and Chongstitvatana, P. An improved genetic algorithm for the inference of finite state machine. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2002, 189.

12. Chongstitvatana, P. and Aporntewan, C. Improving correctness of finite-state machine synthesis from multiple partial input/output sequences. In *Proceedings of the 1st NASA/DoD Workshop on Evolvable Hardware*. 1999, 262–266.
13. Horihan, J. W. and Lu, Y.-H. Improving fsm evolution with progressive fitness functions. In *GLSVLSI '04: Proceedings of the 14th ACM Great Lakes Symposium on VLSI*. ACM Press, New York, NY, USA, 2004, 123–126.
14. Cerruti, U., Giacobini, M., and Liardet, P. Prediction of binary sequences by evolving finite state machines. In *Selected Papers from the 5th European Conference on Artificial Evolution*. Springer-Verlag, London, UK, 2002, 42–53.
15. Rashedi, E., Nezamabadi-pour, H., Saryazdi, S., and Farsangi, M. M. Allocation of static var compensator using gravitational search algorithm. In *First Joint Congress on Fuzzy and Intelligent Systems, Ferdowsi University of Mashhad, Iran, 29–31 August, 2007*, 29–31.
16. Formato, R. A. Central force optimization: a new metaheuristic with applications in applied electromagnetics. *PIER*, 2007, **77**, 425–491.
17. Hsiao, Y.-T., Chuang, C.-L., Jiang, J.-A., and Chien, C.-C. A novel optimization algorithm: space gravitational optimization. In *IEEE International Conference on Systems, Man and Cybernetics, 2005*, Vol. 3. 2005, 2323–2328.
18. Webster, B. and Bernhard, P. J. A local search optimization algorithm based on natural principles of gravitation. Technical Report CS-2003-10, Florida Institute of Technology, 2003.
19. Webster, B. *Solving Combinatorial Optimization Problems Using a New Algorithm Based on Gravitational Attraction*. PhD thesis, Florida Institute of Technology, Melbourne, FL, USA, 2004.
20. Rashedi, E., Nezamabadi-pour, H., and Saryazdi, S. GSA: a gravitational search algorithm. *Inform. Sciences*, 2009, **179**(13), 2232–2248.
21. Rashedi, E., Nezamabadi-pour, H., and Saryazdi, S. Filter modeling using gravitational search algorithm. *Eng. Appl. Artif. Intell.*, 2011, **24**, 117–122.
22. Balachandar, S. R. and Kannan, K. A meta-heuristic algorithm for set covering problem based on gravity. *International Journal of Computational and Mathematical Sciences*, 2010, **4**(5), 223–228.
23. Chatterjee, A., Mahanti, G. K., and Pathak, N. Comparative performance of gravitational search algorithm and modified particle swarm optimization algorithm for synthesis of thinned scanned concentric ring array antenna. *PIER B*, 2010, **25**, 331–348.
24. Zibanezhad, B., Zamanifar, K., Nematbakhsh, N., and Mardukhi, F. An approach for web services composition based on QoS and gravitational search algorithm. In *Proceedings of the 6th International Conference on Innovations in Information Technology, IIT'09*. IEEE Press, Piscataway, NJ, USA, 2009, 121–125.
25. Rashedi, E., Nezamabadi-pour, H., and Saryazdi, S. BGSA: binary gravitational search algorithm. *Nat. Comp.*, 2010, **9**, 727–745.
26. Fabera, V., Janes, V., and Janesova, M. Automata construct with genetic algorithm. In *DSD '06: Proceedings of the 9th EUROMICRO Conference on Digital System Design*. IEEE Computer Society, Washington, DC, USA, 2006, 460–463.

Meetod lõplike automaatide genereerimiseks gravitatsiooniseadusest inspireeritud otsimisalgoritmi abil

Margarita Spichakova

Kuna lõplike automaatide genereerimine sisend-väljundpaaride näidiste alusel on NP-keerukas ülesanne, tuleb selle lahendi leidmiseks kasutada heuristilisi algoritme. Artiklis on pakutud metoodika Moore'i masinate genereerimiseks, kasutades uut, gravitatsiooniseadusest inspireeritud otsimisalgoritmi. On esitatud algoritmi rakendamiseks vajalik Moore'i masina binaaresitus, sihifunktsioon ja algoritmi juhtimiseks kasutatavad parameetrid. Eksperimendid näitavad, et lähenemisel on arvestatav potentsiaal.