Baltic J. Modern Computing, Vol. 5 (2017), No. 1, pp. 87–106
http://dx.doi.org/10.22364/bjmc.2017.5.1.06

Gravitationally Inspired Search Algorithm for Solving Agent Tasks

Margarita SPICHAKOVA

Institute of Software Science at Tallinn University of Technology
Tallinn, Estonia

margarita.spitsakova@ttu.ee

Abstract. Artificial ant problem is defined as constructing the agent that models the behavior of the ant on trail with food. The goal of the ant is to eat all food on trail with limited number of steps. Traditionally, the recurrent neural network or state machines are used for modeling ant and heuristic optimization methods for example Genetic Programming as search algorithm. In this article we use Mealy machines as ant model in combination with Particle Swarm Optimization method and heuristic algorithms inspired by gravity. We propose new gravitationally inspired search algorithm and its application to artificial ant problem. Proposed search algorithm requires discrete search space, so the specific string representation of Mealy machine is introduced. The simulation results and analysis of the search space complexity show that the proposed method can reduce the size of the search space and effectively solve the problem.

Keywords: Soft computing, swarm intelligence, agent, artificial ant, gravitational search, Particle Swarm Optimization.

Introduction

Agent games on grid world are used to understand the behavior of agents, especially their controllers in real world tasks. To behave correctly on the grid, the controller requires some memory, so, for example, *finite state machine* can be used to model agent.

The *artificial ant* task can be described as designing trail tracker, which acts as *artificial ant* and follows some trail, which contains food (see Fig. 1). The goal of ant is to collect maximal amount of food for limited number of steps, traditionally 200 steps. In our case, ant is modeled by *finite state machine* (Mealy type) like agent, but it is possible to use other models, for example recurrent neural networks. The input of such machine is only one variable: is there food in the next cell, with values `FOOD` and `EMPTY`, the outputs are defined as actions of ant: `WAIT`, `TURN LEFT`, `TURN RIGHT`, `MOVE`.

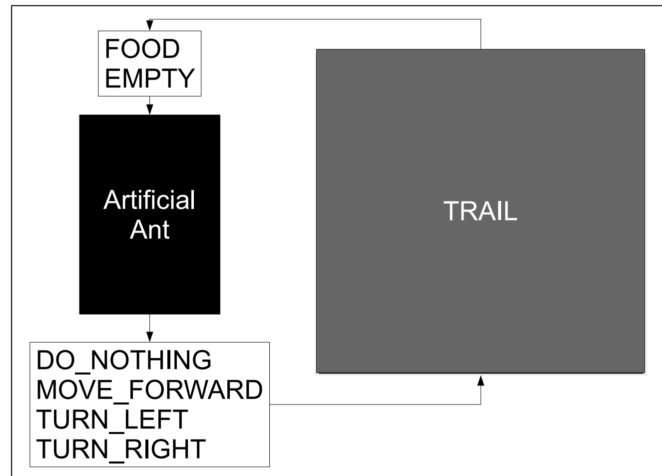

Fig. 1. General model of *artificial ant*

There is no deterministic solution for this problem, so stochastic optimization methods can be used for solving *artificial ant* problem. In most of the cases the population based heuristic methods, such as *Evolutionary Algorithm*s or simulated annealing, are used.

Initially, this task was proposed by Jefferson (1991) to benchmark *Evolutionary Algorithm*s and was researched afterward by many authors: Koza (1992), Angeline (1993), (1994), Kuscu (1998). We use *artificial ant* task for benchmarking our proposed method based on gravitationally inspired search algorithm.

This task is applicable not only in case of *FSM* inference, but also in a case of *Genetic Programming* by Koza (1992) and artificial neural network learning by Jefferson (1991). Chellapilla (1999) uses modular *Mealy machine* as *artificial ant* and *Evolutionary Programming* procedure for the optimization. We focus only on traditional *Mealy machine*s.

In this article we propose a new stochastic optimization method, which can be applied for the problem of *FSM* identification. The proposed method is inspired by combination of *Particle Swarm Optimization* (*PSO*) method and *Gravitational Search Algorithm* (*GSA*).

The article is constructed as follows: Section 1 describes the problem statement in details, Section 3 presents the general idea of proposed stochastic optimization method, Section 4 covers the problem of *FSM* representation, Section 5 gives the main ideas of the method with application to *FSM* identification and Section 6 presents the simulation results and analysis.

1 Artificial ant problem

The problem in hand is known as '*Artificial ant* problem' or 'Trail tracker problem'. It was originally proposed by Jefferson et al. (1991). The goal is construct the agent, which takes information about the food in the next cell and moves on the grid, so that it finds the optimal trail to collect all food on the grid with pre-given amount of steps.

1.1 Trail

The grid is presented by 32×32 toroidal structure, so that if ant is at the bottom of such grid and the next move is `MOVE DOWN`, the next cell will be at the top.

The food on the grid is located in a special way, constructed to make the task more complex. The are two well known trails *John Muir Trail* (Fig. 2 (left)) developed by Jefferson et al. (1991) and *Santa Fe Trail* (Fig. 2 (right)). Both of them contain 89 cell of food.

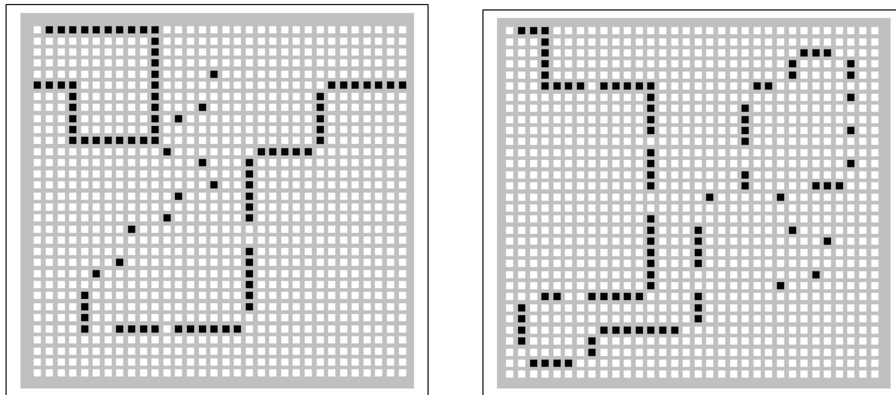

Fig. 2. *John Muir Trail* (left) and *Santa Fe Trail* (right)

1.2 *Finite state machine* as ant model

The ant is modeled by Mealy type machine with n states. The ant sees only one cell directly in front of him, so there is only one input variable with two values (events). So, the input alphabet contains only two chars: `F (FOOD)` – there is food in the next cell, `E (EMPTY)` – the next food is empty.

The allowed actions of ant can be coded by 4 letter alphabet: `W (WAIT)` – the ant will do nothing at this step, `M (MOVE)` – the ant will move to the next cell, if there is a food it will eat it (the eaten food is removed from the grid), `R (TURN RIGHT)` – the ant will stay in the same cell, but turn the right, `L (TURN LEFT)` – the ant will stay in

the same cell, but turn the left. `WAIT` action can be omitted, in that case the size of the alphabet is 3.

The ant can see only the cell in front of it, the address of the cell is defined by orientation of the ant: `N (NORTH)` – ant is looking the top cell, `E (EAST)` – ant is looking the right cell, `S (SOUTH)` – ant is looking the bottom cell, `W (WEST)` – ant is looking the left cell.

The start position of ant $(0, 0)$ is top left cell. The initial orientation is `EAST`. So, we define $\Sigma = \{E,\ F\}$ and $\Delta = \{W,\ M,\ L,\ R\}$. Fig. 3 (left) demonstrates the defined alphabets (actions and events) for the situation, where ant is oriented `EAST`.

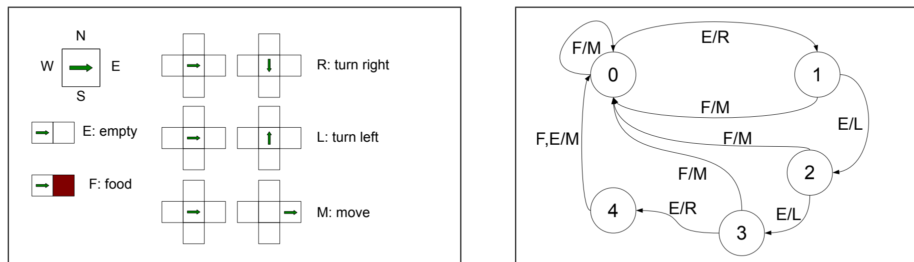

Fig. 3. Ant actions (left) and *Mealy machine* as *artificial ant* (right)

The *FSM*, which is able to fulfill the task is quite complex (we measure complexity by number of states). Jefferson (1991), the author of original task, constructed the *FSM* with 5 states (Fig. 3 (right)), which was able to eat 81 pieces of food out of 89 on *John Muir Trail* by 200 steps and eat all food by 314 steps. As you can see, this ant with 5 steps was not able to eat all the food by 200 steps, to do so, the number of states (memory) must be greater. In the literature ant with up to 13 states are used.

1.3 Ant evaluation

If we want to answer the question 'How well the given ant solves the problem?', we need to construct so called *objective function*. The score of the ant can be found only by modeling situation. The process of *objective function* computation is the most time consuming part of the search algorithm.

There are several possible function. In the easiest case, we can take into account only amount of food that was eaten during given number of steps. We modify this function so that it returns value $\in [0 \ldots 1]$ (Equation 1): we divide the number of eaten food by the total number of food on grid.

$$ObjValue = \frac{eaten\ food}{total\ food\ on\ trail} \tag{1}$$

As you have noticed, presented function does not take into account the number of states of the model and the number of steps that was required to eat all the food, if it is less

than given number of steps. The *objective function* can be modified to minimize the number of states, but we use the simplified version.

2 Inference of finite state machines

To define our problem in the context of stochastic optimization methods we need first of all define several concepts. We have the *problem statement* – 'find the *Mealy machine* with n states, which models the *artificial ant* that eats all food on given trail in given number of steps'. In fact we have our *search space* defined as set of all possible *Mealy machine*s with n states. Each point of such space is *candidate solution*. For each *candidate solution* we can assign *objective value* using the above defined *objective function* (Equation 1). Now, we can redefine the problem as **find the candidate solution with maximal objective value**, which is exactly the definition of the optimization task.

Such tasks are solvable by population based heuristic algorithms.

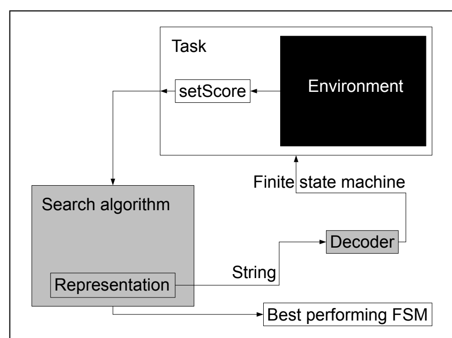

Fig. 4. The outline of the *FSM* search process

The proposed system implements the ideas of unified methods for *FSM* identification consists of three main modules (see Fig. 4). Each module of the system presents the independent part of the system and can be replaced by another implementation or definition:

- *'Task' module* contains the implementation of basic concepts and definitions that are used to describe the statement of the problem. To describe the problem, we need to define the type of the *FSM*, choose alphabets, number of states and construct the *FSM* evaluation algorithm. These parameters (number of states, type of the machine and alphabets) can be derived from problem specification (see Section 1).
- *'Search algorithm' module* contains the implementation of different stochastic optimization algorithms, such as *GA* or *PSO* (see Section 5).
- *'Representation+Decoder' module.* This subsystem contains algorithms for definition of *FSM* and string representation of *FSM* (see Section 4).

3 Heuristic search

There are many different heuristics, for example *Genetic Algorithm*s (*GA*) or *Particle Swarm Optimization* (*PSO*). Some of them are inspired by natural mechanisms, such as evolutionary theory, physics or social behavior.

But still, they have several mechanisms in common. First of all, initial set of the *candidate solution*s are generated randomly. Each of them is evaluated and the *score value* is assigned. Using those values some of the *candidate solution*s are chosen for modification. There are several algorithm specific modification methods. Using them we can improve the *candidate solution*s. After the modifications all *candidate solution*s are evaluated again. The process continues until the optimal solution is found or the *number of generation* is reached. The important fact is that in most of the cases *candidate solution*s are usually presented indirectly, encoded to some structure: Boolean vector, vector of integer or real values. The choice of *representation* and search algorithm plays important role in resolvability of the problem.

3.1 *Evolutionary Algorithm*s

Evolutionary Algorithm (*EA*) is a search algorithm based on simulation of biological evolution. In *EA* some mechanisms inspired by evolution are used: reproduction, mutation, recombination, selection. *Candidate solution* presents the individuals of the evolution.

Some differences in implementation of those principles characterize the instances of *EA* (Bäck 1997):

- *Genetic Algorithm*s (originally described by Holland in 1962) (*GA*) — the solution represented as an array of numbers (usually binary numbers). A recombination operator is also used to produce new solutions.
- *Evolutionary strategies* (developed by Rechenberg and Schwefel in 1965) — use the real number vectors for representing solution. Mutation and crossover are essential operators for searching in search space.
- *Evolutionary Programming*s (developed by Lawrence J. Fogel in 1962) — the algorithm was originally developed to evolve *FSM*, but most applications are for search spaces involving real-valued vectors. Does not incorporate the recombination of individuals and emphasizes the mutation.

*EA*s are often used to solve *artificial ant* task: Jefferson (1991), Koza (1992), Angeline (1993), (1994), Kuscu (1998).

3.2 Standard *Particle Swarm Optimization* algorithm

Particle Swarm Optimization algorithm is inspired by social behavior of some set of objects, for example bird flock or fish school. Initially, it is designed by Kennedy at al. (1995) for the real-valued vector.

There is the set of the particles – *swarm*. Each of them is characterized by *position vector*, *velocity vector* and the best known position for this object. Also, there is the *global best known position* for the whole *swarm*.

The *position* vector $p_d, d \in [0 \dots n]$ presents the *candidate solution*. Dimensionality n of the vector depends on the problem size.

We assign value for each *candidate solution* using *evaluation function*. According to those values we can choose the global best known position $Gbest$, which is the point with optimal value found so far by the whole swarm, and the local best known position $Pbest$, which is the best position that was found by this exact particle.

The *velocity* vector $v_d, d \in [0 \dots n]$ represents the trend of movement of the particle. It is computed by using (2),

$$v_d(t) = \alpha \cdot v_d(t-1) + \beta \cdot r_1 \cdot (Pbest_d - p_d(t-1)) + \gamma \cdot r_2 \cdot (Gbest_d - p_d(t-1)) \quad (2)$$

where:

- $\alpha,\ \beta,\ \gamma$ are learning coefficients and α represents the inertia, β - the cognitive memory, γ - the social memory. Those coefficients must be defined by user.
- r_1 and r_2 are random values in range $[0 \dots 1]$.
- $Pbest_d$ is a local best known position for this particle, $Gbest_d$ is global best known position of the *swarm*.
- $v_d(t)$ is the new value of the velocity vector at dimension d, $v_d(t-1)$ is the previous value of the velocity.

The new position $p_d(t)$ is simply defined as sum of the previous position $p_d(t-1)$ and new velocity $v_d(t)$ (3).

$$p_d(t) = p_d(t-1) + v_d(t) \quad (3)$$

3.2.1 *PSO* algorithm work-flow The initialization part consists of defining required learning parameters, setting up boundaries of the search space and generating the swarm with random position and velocity. The search process is iterative update of the positions and velocities. The process ends when the ending criteria are met: either the number of iterations is exceeded or the optimal solution is found.

3.3 Gravity as inspiration for optimization algorithms

There are four main forces 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 masses.

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

$$F_{ij} = G \frac{M_j \cdot M_i}{R_{ij}^2}. \quad (4)$$

Knowing the force acting on body we can compute acceleration as

$$a_i = \frac{F_i}{M_i}. \quad (5)$$

To construct the search algorithm based on gravity, we can adapt the following ideas:

- Each object in the universe has mass and position.
- There are interactions between objects, which can be described by using law of gravity.
- Bigger objects (with greater mass) create larger gravitational field 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 modeled by objects with mass.
- The position of those objects describes the solution and the mass of the objects depends on the *evaluation function*.
- The objects interact with each other using gravitational force.
- The objects with greater mass present the points in search space with better solution.

Using these characteristics, it is possible to define family of optimization algorithms based on gravitational force. For example, *Central Force Optimization (CFO)* is deterministic gravity based search algorithm proposed and developed by Formato (2007). It simulates the group of probes which fly into search space and explore it. Another algorithm, *Space Gravitational Optimization (SGO)* was developed by Hsiao and Chuang (2005). It simulates asteroids flying through curved search space. A gravitationally-inspired variation of local search algorithm, *Gravitational Emulation Local Search Algorithm (GELS)* was proposed by Webster (2003), (2004). Another one, *Gravitational Search Algorithm* (*GSA*) was proposed by Rashedi (2009)as a stochastic variation of *CFO*.

Basically, the gravitationally inspired algorithms are quite similar to *PSO* algorithms. Instead of particle swarm we have set of bodies with masses, ideas of position and velocity vectors are the same, the movement laws are similar.

The idea of hybridization of *PSO* algorithm and gravitationally inspired search algorithms is not new. There are several existing algorithms that adapt both ideas (*PSO* algorithm and gravity) to construct heuristic search algorithm: *PSOGSA* - *PSO*algorithm and *Gravitational Search Algorithm* developed by Mirjalili (2010), Extended *Particle Swarm Optimization* Algorithm Based On Self - Organization Topology Driven By Fitness - *PSO* algorithm and Artificial Physics, created by Mo et al. (2011), Gravitational Particle Swarm, proposed by Tsai (2013), Self-organizing *Particle Swarm Optimization* based on Gravitation Field Model, created by Qi et al. (2007).

The traditional way of hybridization of *PSO* algorithm with gravitationally-inspired search algorithms is to add gravitational component to the velocity computation. (2) has additional component, which is computed by using gravitational interactions. Unfortunately, this makes the behavior of the search algorithm even more complex and unpredictable. Additionally, the user defined parameters still need to be found.

The choice of the optimization method and its effectiveness strictly depends on type of the search space. In next Section we discuss the definition of our search space.

4 Representation of finite state machines

Let's have a target *Mealy machine* with n states.

- Input alphabet: $\Sigma = \{i_0, \dots, i_{k-1}\}$;
- Output alphabet: $\Delta = \{o_0, \dots, o_{m-1}\}$;
- Set of states: $Q = \{q_0, \dots, q_{n-1}\}$;

One row of the transition table for such *Mealy machine* can be described by structure presented on Fig. 5, where we store the information for corresponding state q_j: for all transition from this state with respect to input symbol $i_{0 \dots k-1}$ we encode output value $o^{i_{\dots}}$ and the label of target state $q^{i_{\dots}}$.

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

Fig. 5. One row of *Mealy machine* transition table

The structure required to code whole transition table is constructed as concatenation of such sections in the order of state labels (see Fig. 6).

State q_0					...	State q_{n-1}				
$o_0^{i_0}$	$q_0^{i_0}$	...	$o_0^{i_{k-1}}$	$q_0^{i_{k-1}}$	...	$o_{n-1}^{i_0}$	$q_{n-1}^{i_0}$	...	$o_{n-1}^{i_{k-1}}$	$q_{n-1}^{i_{k-1}}$

Fig. 6. $\mathrm{SR(MeFST)}$ as concatenation of transition table rows

Definition 1 ($\mathrm{SR(MeFST)}$**: String representation of *Mealy machine*).** The string representation of *Mealy machine* is a structure in form:
$o_0^{i_0} q_0^{i_0} \dots o_0^{i_{k-1}} q_0^{i_{k-1}} \dots o_{n-1}^{i_0} q_{n-1}^{i_0} \dots o_{n-1}^{i_{k-1}} q_{n-1}^{i_{k-1}}$, where $[o_0^{i_0} \dots o_{n-1}^{i_{k-1}}] \in [0 \dots m-1]$ presenting codes for output values of the transitions and $[q_0^{i_0} \dots q_{n-1}^{i_{k-1}}] \in [0 \dots n-1]$ presenting target states of the transitions.

Theorem 1 (The space complexity of $\mathrm{SR(MeFST)}$**).** *The length of* $\mathrm{SR(MeFST)}$ *representing* Mealy machine *with n states and over input alphabet k with output alphabet with m symbols is*

$$((1+1) \times k) \times n$$

The number of corresponding $\mathrm{SR(MeFST)}$ *strings is*

$$((m \times n)^k)^n$$

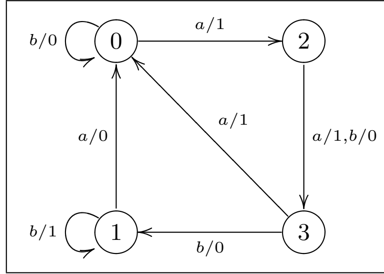

Fig. 7. Transition diagram of *Mealy machine* M_{me1}

Example 1. Let's take a look at a *Mealy machine* M_{me1} with transition diagram represented on Fig. 7. $Q = \{0, 1, 2, 3\}$, $\Sigma = \{a, b\}$, $\Delta = \{0, 1\}$.

The string representation for such machine will be
$SR(M_{me1}) = [1, 2, 0, 0, 0, 0, 1, 1, 1, 3, 0, 3, 1, 0, 0, 1]$ (see Fig. 8).

State 0				State 1				State 2				State 3			
a		b		a		b		a		b		a		b	
1	2	0	0	0	0	1	1	1	3	0	3	1	0	0	1

Fig. 8. String representation $SR(M_{me1})$

The separation of the the transition and output functions can be easily done by constructing two strings from a $\mathrm{SR(MeFST)}$ string. Fig. 9 describes separating the transition and output functions transformation for *Mealy machine*.

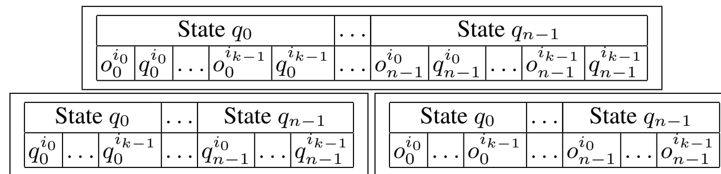

Fig. 9. $\mathrm{SR_S(MeFST)}$: Separating transition $SR_S(MeFST.transition)$ (right) and output functions $SR_S(MeFST.output)$ (left) for *Mealy machine* $\mathrm{SR(MeFST)}$ (top)

Based on this transformation process the $\mathrm{SR(MeFST)}$ is defined (Definition 2).

Definition 2 ($\mathrm{SR_S(MeFST)}$**: Separated string representation of *Mealy machine*).** The separated string representation of *Mealy machine* is a structure formed from

SR(MeFST):

$$SR_S(MeFST) = \{SR_S(MeFST.transition), SR_S(MeFST.output)\},$$

where

- $SR_S(MeFST.transition)$ presents transition function $q_0^{i_0} \dots q_0^{i_{k-1}} \dots q_{n-1}^{i_0} \dots q_{n-1}^{i_{k-1}}$, with $[q^{i_0} \dots q_{n-1}^{i_{k-1}}] \in [0 \dots n-1]$ presenting target states of the transitions and
- $SR_S(MeFST.output)$ presents output function $o_0^{i_0} \dots o_0^{i_{k-1}} \dots o_{n-1}^{i_0} \dots o_{n-1}^{i_{k-1}}$, where $[o^0 \dots o^{n-1}] \in [0 \dots m-1]$ is presenting codes for output values.

Example 2. Let's return to M_{me1} (Example 1). The original code is
$SR(M_{me1}) = [1, 2, 0, 0, 0, 0, 1, 1, 1, 3, 0, 3, 1, 0, 0, 1]$,
so if we transform it to separated string representation the result will be:
$SR_S(M_{me1}) = \{[2, 0, 0, 1, 3, 0, 3, 0, 0, 1], [1, 0, 0, 1, 1, 0, 1, 0,]\}$

Example 3. The ant on Fig. 3 (right) can be presented by
$cSR_S(M_{ant81t314}) = \{[1,\ 0,\ 2,\ 0,\ 3,\ 0,\ 4,\ 0,\ 0,\ 0],\ [3,\ 1,\ 2,\ 1,\ 2,\ 1,\ 3,\ 1,\ 1,\ 1]\}$.

The $SR_S(FSM.transition)$ code depends on the labels of the states and their ordering, renaming states leads to isomorphisms. We can solve this problem by determining the way the state labels will be named. To do so, we adapt a method known as *normal form string*.

4.1 Normal form strings

We describe the basic theory we use in further algorithms. The used methods were proposed by Almeida, Moreira and Reis (2005), (2007), (2009) in a context of enumeration of deterministic finite state acceptors (*DFA*). We present only some definitions and algorithms we require for the research, so more information, proofs and other algorithms can be found in original sources.

The main goal of their approach is to find unique string representation of initially connected (all states are reachable from the initial one) *DFA* (*ICDFA*), this is done by construction of state label ordering. This representation is unique.

Suppose we have $ICDFA_\emptyset = (Q, \Sigma, \delta, q_0)$, where Q is a set of states $|Q| = n$, q_0 is initial states, Σ is the input alphabet with k symbols and δ is a transition function. As you can notice, the set of final states is omitted.

The *canonical string representation* (based on canonical order of states of the *ICDFA*) is constructed by exploring set of states of given *ICDFA* using bread-first search by choosing outgoing edges in the order of symbols in Σ.

So, first of all, the ordering of input alphabet must be defined, for given $\Sigma = \{i_0, i_1, \dots, i_{k-1}\}$, there is order $i_0 < i_1 < \dots < i_{k-1}$, for example the lexicographical ordering can be used.

For given $ICDFA_\emptyset$, the representing string will be in form $(s_j)_{j \in [0 \dots kn-1]}$ with $s_j \in [0 \dots n-1]$ and $s_j = \delta(\lfloor j/k \rfloor, i_{j \bmod k})$. There is a one-to-one mapping between

string $(s_j)_{j\in[0\ldots kn-1]}$ with $s_j \in [0\ldots n-1]$ satisfying rules (more details in Almeida (2009) work):

$$(\forall m \in [2\ldots n-1])(\forall j \in [0\ldots kn-1])$$
$$(s_j = m \Rightarrow (\exists l \in [0\ldots j-1])s_l = m-1) \quad (6)$$
$$(\forall m \in [1\ldots n-1])(\exists l \in [0\ldots km-1])s_l = m \quad (7)$$

and non-isomorphic $ICDFA_\emptyset$ with n states and input alphabet Σ with k symbols.

In the canonical string representation $(s_j)_{j\in[0\ldots kn-1]}$ we can define *flags* $(f_j)_{j\in[1\ldots n-1]}$ that will be sequence of indexes of first occurrence of state label j. The initial sequence of flag is $(ki-1)_{i\in[1\ldots n-1]}$. The rules described before can be reformulated as

$$(\forall j \in [2\ldots n-1])(f_j > f_j - 1) \quad (8)$$
$$(\forall m \in [1\ldots n-1])(f_m < km) \quad (9)$$

For given k and n, the number of sequences $(f_j)_{j\in[1,n-1]}$, $F_{n,k}$ can be computed by

$$F_{k,n} = \binom{kn}{n}\frac{1}{(k-1)n+1} = C_n^{(k)}, \quad (10)$$

where $C_n^{(k)}$ are the Fuss-Catalan numbers. The proof can be found in Almeida (2009).

The process of enumeration of all possible *ICDFA* presented by canonical string representation consists from two parts: generating flags and generating all sequences inside the flag.

We define $\mathrm{cSR_S(FST)}$ as a special case of $\mathrm{SR_S(FST)}$ system, where $SR_S(FSM.transition)$ is represented by *canonical string representation*. This approach ensures that there are no isomorphisms and machines with unreachable states in the enumeration list.

4.2 Search space structure

The *normal form string* representation method allows us to define non-intersecting subspaces in the search space (Fig. 10). This gives the possibility to handle those subspaces separately in space or in time. Those subspaces are characterized by flags.

Definition 3. The universe is a set of all possible *FST*s, represented by canonical string representation $\mathrm{cSR_S(FST)}$, where canonical string representations of transition functions belong to one flag.

Definition 4. The multiverse is a set of all possible *Universes* defined by flags.

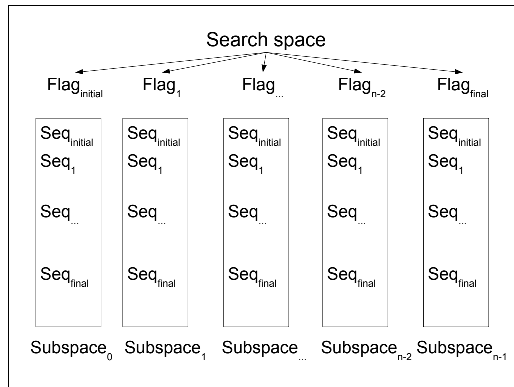

Fig. 10. Search space structure

5 Search algorithm

The task of the search algorithm is to find a point in a search space, which corresponds to the *FST* with the optimal behavior. For each point in a search space *evaluation function* assigns a score value from $[0 \ldots 1]$, which describes how well corresponding *FST* behaves. So, the task is to find the point with maximal value (1.0) or at least the point with maximal score value.

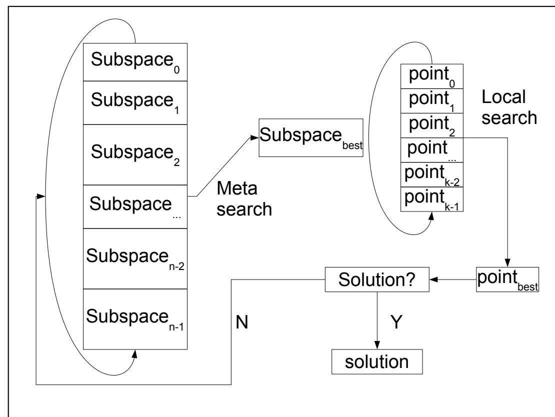

Fig. 11. Search algorithm

Search space (Multiverse) (see Definition 4) consists from several *Universes* (non-intersecting subsets) (see Definition 3) and each *Universe* contains points (see Subsection 4.2). Some of the *Universes* have higher probability to contain the solution than

others. The idea is to subdivide the search algorithm into two phases, the first one chooses the *Universe* (which is 'better' than others) and second phase searches this *Universe* locally, to find the best point (Fig. 11):

1. Phase 1. 'Multiverse search'. During meta search we try to evaluate the *Universes* (subspaces), so that it is possible to choose the *Universe*, which includes solutions with higher probability.
2. Phase 2. '*Universe* local search'. The task of the local search is to find the maximal point inside pre-given *Universe*. If solution was not found the algorithm will return to the meta search phase. This cycle is done until solution (with maximal value) is found or all Universes are explored. If all subspaces are explored and solution with value 1.0 was not found, then algorithm will return the solution with maximal value (< 1.0).

5.1 Multiverse search

The task of meta search algorithm is to assign value to each subspace (described by *flag*) and to choose 'the best' subspace, which contains the solution with higher probability.

```
Size of search space: 9487698075
  0 Universe: (1, 3, 5, 7): * 1 * 2 * 3 * 4 * *  Size: 600 * 59049  From: 0 to: 599
 1 Universe: (1, 3, 5, 6): * 1 * 2 * 3 4 * * *  Size: 750 * 59049  From: 600 to: 1349
 2 Universe: (1, 3, 4, 7): * 1 * 2 3 * * 4 * *  Size: 800 * 59049  From: 1350 to: 2149
 3 Universe: (1, 3, 4, 6): * 1 * 2 3 * 4 * * *  Size: 1000 * 59049  From: 2150 to: 3149
 4 Universe: (1, 3, 4, 5): * 1 * 2 3 4 * * * *  Size: 1250 * 59049  From: 3150 to: 4399
...

 37 Universe: (0, 1, 2, 7): 1 2 3 * * * * 4 * *  Size: 6400 * 59049  From: 108150 to: 114549
 38 Universe: (0, 1, 2, 6): 1 2 3 * * * 4 * * *  Size: 8000 * 59049  From: 114550 to: 122549
 39 Universe: (0, 1, 2, 5): 1 2 3 * * 4 * * * *  Size: 10000 * 59049  From: 122550 to: 132549
 40 Universe: (0, 1, 2, 4): 1 2 3 * 4 * * * * *  Size: 12500 * 59049  From: 132550 to: 145049
 41 Universe: (0, 1, 2, 3): 1 2 3 4 * * * * * *  Size: 15625 * 59049  From: 145050 to: 160674
```

Fig. 12. The structure of the search space: subspaces and their flags

Fig. 12 illustrates the structure of the search space for the case of *Mealy machine* with 5 states. It consists of 42 *Universes* characterized by *flags*: from $(1, 3, 5, 7)$ to $(0, 1, 2, 3)$. The only useful information we know about subspace is its *flag*. *Flag* gives the pattern for sequences and basically defines how states are connected (some part of transition function). The idealistic meta search algorithm evaluates subspaces without generating points in any of them. The realistic meta search algorithm evaluates subspaces based on generated points.

The simplest idea is to evaluate the *Universe* subspace based on average value of all points. The idea of this method is to generate randomly some amount of points (defined by per cent from the size of the search space) and based on values of those points construct the subspace score. This method uses the average function to get the score. Current method uses the maximal value of the points to get the score function $Universe.score = best(Points[k].value)$ for *Universe*.

5.2 Universe local search. Gravitationally inspired search algorithm

For the *Universe* local search we propose to apply heuristic search method. We reuse the ideas presented in *PSO* algorithm (Section 3.2) and in gravitationally inspired algorithm (Section 3.3), although both of them are for continuous search space we adapt their main ideas to discrete search space. The proposed method is called 'Discrete Gravitational Swarm Optimization' (*DGSO*).

Each point in a search space characterized by:

- $position[]$ – which represents the solution,
- $velocity[]$ –which stores the information about the change of the $position[]$ vector,
- $mass$ – corresponds to score value of the solution,
- $position[]_{best}$ – the best known position for this point,
- $position[]_{global}$ – the best known position in explored search space.

First of all, we will define the distance between points. Each point, in our case, is the n-dimensional vector of integers. To calculate the distance between vectors, we use distance in one dimension

$$distanceInD(valueD_1, valueD_2) = \begin{cases} 0 : valueD_1 == valueD_2 \\ 1 : valueD_1 \neq valueD_2 \end{cases}, \quad (11)$$

which returns '1', if values in corresponding dimension are not equal and otherwise it returns '0'.

The distance between vectors is defined as sum of distances for all dimensions:

$$distance(p_1[], p_2[]) = \Sigma_{d=0}^{p.length-1} distanceInD(p_1[d], p_2[d]) \quad (12)$$

Now we need to redefine two operations: movement (change of the position) and acceleration (change of the velocity). In *PSO* and *GSA* algorithms those operations are defined as sums, in our case we will use algorithms similar to 'crossover' operator in *EA*.

Algorithm 1 $Accelerate(Force_p, Force_g, velocity[], position[]_{pBest}, position[]_{gBest})$

for $dimension = 0 \rightarrow velocity.length - 1$ **do**
  if $Random() < Force_p$ **then**
    $newVelocity[dimension] = position_{pBest}[dimension]$
  else
    $newVelocity[dimension] = velocity[dimension]$
  end if
end for
for $dimension = 0 \rightarrow velocity.length - 1$ **do**
  if $Random() < Force_g$ **then**
    $newVelocity[dimension] = position_{gBest}[dimension]$
  else
    $newVelocity[dimension] = velocity[dimension]$
  end if
end for
return $newVelocity[]$

The 'move' operator (Algorithm 2) is the procedure, which changes the current position to the new one according to the tendency, which described by $velocity[]$ vector. The ability of changing is described by $mass_{inertial}$, the bigger $mass_{inertial}$ means, that this point tends to save it's current position. The 'move' operator is similar to uniform 'crossover' operator between $position[]$ and $velocity[]$ vectors.

Algorithm 2 $Move(position[], velocity[], mass_{inertial})$

```
for dimension = 0 → position.length − 1 do
    if Random() < 1 − mass_inertial then
        newPosition[dimension] = velocity[dimension]
    else
        newPosition[dimension] = position[dimension]
    end if
end for
return newPosition[]
```

The proposed gravitationally inspired search algorithm was evaluated in a context of Diophantine Equation Solver by Spichakova (2016) and showed good results.

6 Simulations and analysis

Before discussing the experiments and algorithm effectiveness, we describe the complexity of the search space.

6.1 Size of the search space

Search space complexity shows how many points (which can be considered as *FST*) there are in a search space for given number of states n, size of input alphabet $k = 2$ and size of the output alphabet $m = 3$ (we leave out `W (WAIT)` action, see Subsection 1.2).

Table 1. Search space complexity

n	$cSR_S(Ant)$	$SR_S(Ant)$
2	972	1 296
3	157 464	531 441
4	34 432 128	429 981 696
5	9 487 698 075	576 650 390 625
6	3 152 263 549 140	1 156 831 381 426 180
7	1 225 311 951 419 010	3 243 919 932 521 510 000

Table 1 shows how search space grows with change of number of states n. Columns $cSR(Ant)$ and $SR(Ant)$ show the size of search space (the number of all possible *FST*) with respect to corresponding string representation: $\mathrm{cSR_S(FST)}$ and $\mathrm{SR_S(FST)}$. As you can see, proposed $\mathrm{cSR_S(FST)}$ string representation significantly reduces the size of the search space.

6.2 How to compare the effectiveness of solutions

Due to different methods and algorithms for *artificial ant* representation (grammars (Sugiura 2016), trees, *FSM*s, neural nets etc.), different heuristic search methods (*GP*, *GA*, *Ant Colony Optimization* (Chivilikhin et al. 2013)) and different possible *evaluation function* the proposed method can not be directly compared to already existing solutions. Also, we minimized and structured the search space, so it is hard to separate the effect of the proposed gravitationally inspired search algorithm from the effect of the search space minimization.

One of the most time consuming process during the search algorithm is evaluation of the ant. To optimize the search process we need to minimize the number of evaluations. *The number of fitness evaluations* ($\mathrm{N_{eval}}$) shows how many times the evaluation function was computed. $\mathrm{N_{eval}}$ is also applicable for different search methods, so it can be used as metric for algorithm comparison.

6.3 Experimental results

During experiments we have tried to construct *MeFST* with 5 . . . 7 states, which models the *artificial ant*. The ant is simulated on *Santa Fe Trail*. The input and output alphabets are defined as $\Sigma = \{E,\ F\}$ and $\Delta = \{M,\ L,\ R\}$, the objective function counts the number of food eaten during 400 steps using ((1)).

The parameters for the proposed *DGSO* method are the following:

- number of steps in each local search phase $e = 50$,
- at the initial stage of the *Multiverse* search some amount of point for *Universe* must be generated. Coefficients C_t (coefficient for transition function) and C_o (coefficient for output function) (Table 2) show how many points are generated:
  - $|Universe(MeFST.transition)| = |cSR_s(MeFST.transition)| \cdot C_t$
  - $|Universe(MeFST.output)| = |cSR_s(MeFST.output)| \cdot C_o$

Table 2. Initialization parameters

n	C_t	C_o
5	0.002	0.002
6	0.00015	0.00015
7	0.00005	0.00005

We use $\mathrm{N}_{\mathrm{eval}}$ (see Subsection 6.2) for measuring the quality of proposed *DGSO* method. Christensen, S. and Oppacher, F. (2007) proposed a method with $\mathrm{N}_{\mathrm{eval}}$ = 20696 fitness evaluations for *Santa Fe Trail* based on *GP* + small tree analysis. In Table 3 the third column contains the results of the method proposed by Chivilikhin et al.(2013) with respect to number of states in the *FSM*.

Table 3. Fitness evaluations

Christensen (2007)	n	*Chivilikhin (2013)*	DGSO (avg)	DGSO (min)
20696	5	*10975*	114912	**4642**
20696	6	*9313*	233508	**5205**
20696	7	*9221*	423208	93290

Table 3 contains the result for a proposed *DGSO* method: the mean number of evaluations and the minimal number of evaluations in 100 runs. The arithmetic mean of $\mathrm{N}_{\mathrm{eval}}$ for *DGSO* method is bigger than for the Chivilikhin (2013) and Christensen (2007) methods, but the minimal $\mathrm{N}_{\mathrm{eval}}$ is better for the case of 5–6 states.

6.4 Analysis

Despite the fact that the mean number of evaluations $\mathrm{N}_{\mathrm{eval}}$ is bigger than for the other existing methods, the proposed *DGSO* method has lot of potential and for some cases was able to find the solution with smaller $\mathrm{N}_{\mathrm{eval}}$. The results with smaller $\mathrm{N}_{\mathrm{eval}}$ were produced, when the method was able to find the correct *Universe* (the *Universe*, which contains the solution) during meta-search stage in the first step. The results with bigger $\mathrm{N}_{\mathrm{eval}}$ were produced during runs, where several *Universes* (25 for the case of 5 states and 34 for the case of 6 states) were searched before the solution was found.

There are two main problems with proposed method that lead to such big $\mathrm{N}_{\mathrm{eval}}$:

- During the initialization process some amount of points in each *Universe* must be generated (for test cases these parameters presented in Table 2). Currently this amount is defined with respect to size of the *Universe*. If the search space is big, as for $n = 7$, huge amount of points are generated and evaluated already at initial stage.
- The worst case scenario for the search method is the situation, when the all *Universe*s are searched and the last one contains the solution (of cause if there is solution). The best case scenario is the situation, when the solution is found in the first *Universe*. Currently, the *Universe*s are evaluated by the best point found during the initialization phase, which is not optimal.

One of the possible solutions for these two problems is to change the initialization process and modify the function for *Universe* evaluation. The ideal situation would be

the possibility to evaluate the *Universe* without point generation. Such optimization can be researched in future.

7 Conclusion and future work

We presented a method for identification of *FSM*s in a context of *artificial ant* problem, which is based on heuristic optimization algorithm.

The special coding system *'Canonical String Representation'* is used to represent search space. Proposed string representation of *FSM*s is adaptation of existing method for enumeration of *FA*s, it was updated to take into account output function of *FST*. First of all, such representation allows to *minimize search space*, and, secondly, search space can be partitioned and non-intersecting partitions can be handled in *parallel* or separately in time.

The specifics of the search space representation gives the possibility to create *two stages of the search*: the search of 'best' partition (meta-search) and search inside the partition.

Due to complex structure of search space, standard *PSO* is not working well, so we propose to combine *PSO* and modern gravitational algorithm and present new hybrid *Discrete Gravitational Swarm Optimization* method. Also, the search method is modified to be able to work in discrete search space, because our search space is presented by decimal strings. So, all algorithm operators were redesigned.

The experimental results show that proposed method is efficient in some cases, but the 'meta-search phase' must be improved to make results more stable.

There are several ways to extend current research. First of all, we can adapt the proposed method to other structures, for example neural nets. Secondly, we can try the method to other similar tasks. Thirdly, the family of heuristic optimization techniques is constantly growing, so some of them can be modified to match the search space representation.

Acknowledgments

This research was supported by the Estonian Ministry of Research and Education institutional research grant no. IUT33-13.

References

Almeida, M., Moreira, M., Reis, R. (2007). Enumeration and generation with a string automata representation, *Theoretical Computer Science*, 93–102.

Almeida, M., Moreira, N., Reis, R. (2009). Aspects of enumeration and generation with a string automata representation, *CoRR*.

Angeline, P. J., Pollack, J. B. (1993). Evolutionary Module Acquisition, *Proceedings of the Second Annual Conference on Evolutionary Programming* , 154–163.

Angeline, P. J., Pollack, J. B. (1994). Coevolving High-Level Representations, *Addison-Wesley C. Langton (Eds.), Artificial Life III*, 55–71.

Bäck, T., Fogel, D.B., Michalewicz Z. (1997). Handbook of Evolutionary Computation, *IOP Publishing Ltd.*.

Chellapilla, K., Czarnecki, D. (1999). A preliminary investigation into evolving modular finite state machines *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99*, 1349–1356 Vol. 2.

Chivilikhin, D., Ulyantsev, V., Shalyto, A. (2013). Solving Five Instances of the Artificial Ant Problem with Ant Colony Optimization *Proceedings of the 7th IFAC Conference on Manufacturing Modelling, Management, and Control*, 1043–1048.

Christensen, S., Oppacher, F. (2007). Solving the artificial ant on the santa fe trail problem in 20,696 fitness evaluations. *Proceedings of the 9th annual conference on Genetic and evolutionary computation, GECCO 07*, 15741579.

Formato, R.A. (2007). Central force optimization: a new metaheuristic with applications in applied electromagnetics, *Progress in Electromagnetics Research*, 425–491.

Hsiao, Y.-T., Chuang, C.-L., Jiang J.-A., Chien C.-C.(2005). A novel optimization algorithm: space gravitational optimization, *IEEE International Conference on Systems, Man and Cybernetics*, 2323–2328, Vol. 3.

Jefferson, D., Collins, R., Cooper, C., Dyer, M., Flowers, M., Korf, R., Taylor, C., Wang, A. (1991). Evolution as a theme in artificial life: The Genesys/Tracker system, *C. G. Langten, C. Taylor, J. D. Farmer, and S. Rasmussen (Eds.), Artificial life II*, 549–578.

Kennedy, J., Eberhart, R. (1995). Particle swarm optimization, *IEEE International Conference on Neural Networks*, 1942–1948, Vol.4.

Koza, J. R. (1992). Genetic programming: on the programming of computers by means of natural selection, *MIT Press*.

Kuscu, I. (1998). Evolving a Generalised Behavior: Artificial Ant Problem Revisited, *Proceedings of Seventh Annual Conference on Evolutionary Programming, LNCS*, 799–808.

Mirjalili, S., Hashim, S. Z M. (2010). A new hybrid PSOGSA algorithm for function optimization *Proceedings of International Conference on Computer and Information Application (ICCIA)*, 374–377.

Mo, S., Zeng, J., Xu, W. (2011). An Extended Particle Swarm Optimization Algorithm Based On Self-Organization Topology Driven By Fitness, *Journal of Computational Information Systems, 7:12*, 4441–4454.

Qi, K., Lei, W., Qidi, W. (2007). A Novel Self-organizing Particle Swarm Optimization based on Gravitation Field Model *American Control Conference, 2007. ACC '07*, 528–533.

Rashedi, E., Nezamabadi-pour, H., Saryazdi, S. (2009). GSA: A Gravitational Search Algorithm, *Inform. Sciences, Nr.13, Vol. 179*, 2232–2248.

Reis, R., Moreira, N., Almeida, M. (2005). On the representation of finite automata, *Proc. of DCFS05*, 269–276.

Spichakova, M. (2016). Modified Particle Swarm Optimization Algorithm Based on Gravitational Field Interactions, *Proceedings of the Estonian Academy of Sciences, Vol. 65, Issue 1*, 15–27.

Sugiura, H., Mizuno, T., Kita, E. (2012). Santa Fe Trail Problem Solution Using Grammatical Evolution , *2012 International Conference on Industrial and Intelligent Information (ICIII 2012)*, 36–40.

Tsai, H.-C., Tyan, Y.-Y., Wu, Y.-W., Lin, Y.-H.(2013). Gravitational Particle Swarm, *Applied Mathematics and Computation, Vol. 219, Num. 17* , 9106–9117.

Webster, B., Bernhard, P. J. (2003). A Local search optimization algorithm based on natural principles of gravitation, *CS-2003-10, Florida Institute of Technology*.

Webster, B. (2004). Solving combinatorial optimization problems using a new algorithm based on gravitational attraction, *Ph.D. thesis, Florida Institute of Technology*.

Received September 9, 2016 , revised December 12, 2016, accepted February 21, 2017