Journal of Siberian Federal University. Mathematics & Physics 2017, 10(4), 463-473

УДК 519.87

Self-configuring Nature Inspired Algorithms for Combinatorial Optimization Problems

Olga Ev. Semenkina*
Eugene A. Popov†
Olga Er. Semenkina‡
Siberian State Aerospace University
Krasnoyarsky rabochy, 31, Krasnoyarsk, 660037
Russia

Received 10.03.2017, received in revised form 10.06.2017, accepted 20.08.2017

In this work authors introduce and study the self-configuring Genetic Algorithm (GA) and the self-configuring Ant Colony Optimization (ACO) algorithm and apply them to one of the most known combinatorial optimization task – Travelling Salesman Problem (TSP). The estimation of suggested algorithms performance is fulfilled on well-known benchmark TSP and then compared with other heuristics such as Lin-Kernigan (3-opt local search) and Intelligent Water Drops algorithm (IWDs). Numerical experiments show that suggested approach demonstrates the competitive performance. Both adaptive algorithms show good results on these problems as they outperform other algorithms with their settings with average performance.

Keywords: travelling Salesman problem, genetic algorithm, ant colony optimization, intelligent water drops algorithm, self-configuration.
DOI: 10.17516/1997-1397-2017-10-4-463-473.

Introduction

Decision-making problems often produce a task of choice one of possible options for an action. Formalized statement of this problem is an optimization problem. Optimization, including combinatorial one, is an actual area of modern science. Real word problems are rather complicated, that is why standard techniques, such as mathematical programming, cannot solve them efficiently. Therefore, evolutionary methods and biology inspired algorithms are used to solve complex optimization problems.

In this paper we consider one of the most known combinatorial optimization problem named Travelling Salesman problem (TSP) [1], which has many practical applications. The traveling salesman problem is a generalization of the problem of Hamiltonian cycles in graphs and belongs to the class of NP-complete problems. TSP is often used to test the newly created algorithms of combinatorial optimization and it has a lot of applications in routing, scheduling and many other fields. It is formulated as follows: *Let there be a given set of n cities. The task is to find closed shortest path included all cities under condition that each city must be visited only once.*

*oleese@mail.ru
†epopov@bmail.ru
‡semenkina.olga@mail.ru

This paper considers several algorithms and their modifications, to be exact an adaptation of the algorithm parameters to a particular task. Nature inspired algorithms usually have many adjustable parameters and this is their significant disadvantage as the parameters tuning is a difficult task even for the specialists. Nowadays, for the elimination of this defect one applies the various approaches to algorithms' self-adaptation, i.e. parameters tuning during algorithms execution. Another approach is the algorithm self-configuration, i.e. a choice of appropriate algorithm's configuration (set of operators, etc.) "on-the-fly" [2].

Considered way of GA self-configuration was introduced in [3] where its usefulness was demonstrated on benchmark problems and in applied problems of artificial neural networks weights adjustment. This approach was then successfully used in solving real world optimization problems with algorithmically given functions of mixed variables [4]. This made the approach to be a candidate for the development of adaptive algorithms of the combinatorial optimization. Performance of self-configuring Genetic Algorithm (GA) [5] and the self-configuring Ant Colony Optimization (ACO) [6] algorithm is compared with other heuristics, namely Lin-Kernigan heuristic [7] and Intelligent Water Drops algorithm [8]. Results of numerical experiments on benchmark problems show that suggested approach demonstrates competitive effectiveness.

1. Algorithms description

1.1. Lin-Kernighan heuristic

One of the classical methods for solving the traveling salesman problem is a local search [9], in particular so called k-opt algorithm (Lin-Kernighan heuristic [7]). TSP solution is presented by cyclic graph f. The k-opt neighborhood $N_k(f)$ includes all the tours which can be obtained by removing k edges from the original tour f and adding k different edges such that the resulting tour is feasible. The essence of the algorithm is to consider neighborhood of the current solution. If there exists a graph g in this neighborhood with better objective function value, then g becomes current solution. The procedure is repeated as long as the current solution can be improved.

Exploring the whole $N_k(f)$ takes $O(n^k)$ operations and, thus, 2-opt and rarely 3-opt are used in practice. This paper deals with 3-opt because it is more efficient [9].

1.2. Intelligent water drops algorithm

Intelligent water drops algorithm (IWDs) possesses a few properties of a natural water drops. The paths that a river follows have been created by a swarm of water drops. Thus, any swarm of water drops will influence the rivers path. On the other side, for a swarm of water drops, the river is the part of the environment that has an influence over it. A large influence on the movement of the river shows which type of soil and how resistant it is to the flow, as it determines the drops speed. Thus, the path of the water drops swarm depends on path of the river, type of soil and its resistance. So the formation of a natural river is the result of a competition between the water drops and the environment that resists its movement. Notice that all natural rivers are full of twists and turns. This is due to the influence of gravity which pulls the water through the path of least resistance to the lowest point.

It is assumed that each drop of water is able to transfer an amount of soil from one place to another. Furthermore, the soil is transferred from the fast parts of the river to the slow parts. This makes the fast parts deeper, allowing them to hold a greater volume of water. The quantity of soil a water drop is able to transfer depends on its velocity. Furthermore, the velocity of a

water drop depends on the amount of soil in its way. The velocity of a water drop grows faster on a path with less soil. Water drops prefer a path with the least amount of soil.

On the basis of the above properties Shah-Hosseini in 2007 proposed the intelligent water drops algorithm [8]. Every intelligent water drop (IWD) has two important properties: the amount of soil that it carries and its velocity. For each IWD, the values of both properties, soil and velocity, may change as the IWD flows in its environment. From the mathematical point of view, the environment is a problem for river sand swarm of water drops looking for the optimal path.

Velocity of IWD, that moves from its location i to the location j, is increased by an amount

$$\Delta vel = \frac{a_v}{b_v + c_v soil^2(i,j),} \tag{1}$$

where parameters a_v, b_v and c_v should be chosen as positive numbers.

IWD's soil is increased by removing some soil of the path ij. The amount of soil added to the IWD is calculated by

$$\Delta vel(i,j) = \frac{a_z}{b_z + c_z time^2(i,j;vel^{IWD}),} \tag{2}$$

where a_z, b_z and c_z are positive parameters. Time is calculated by the simple laws of physics for linear motion.

$$time(i,j,vel^{IWD}(t+1)) = \frac{HUD(i,j)}{vel^{IWD}.} \tag{3}$$

Local heuristic function HUD(.,.) has been defined for a given problem to measure the undesirability of an IWD to move from one location to the next. For TSP it is calculated as follow:

$$HUD(i,j) = ||\overline{e}(i) - \overline{e}(j)||, \tag{4}$$

where $\overline{e}(i)$ is vector of coordinates, $||\cdot||$ is Euclidean metric.

Soil amount between i and j is updated by the amount of soil removed by the IWD by formula:

$$soil(i,j) = (1-\rho_n)\cdot soil(i,j) - \rho_n \cdot \Delta soil(i,j). \tag{5}$$

The soil of the IWD is increased by the amount of soil as shown below:

$$soil^{IWD} = soil^{IWD} + \Delta soil(i,j). \tag{6}$$

The probability of choosing location j after i is proportional to the amount of the soil on the path between locations i and j and can be calculated by formula:

$$p_i^{IWD}(j) = \frac{f(soil(i,j))}{\sum\limits_{k \notin V} f(soil(i,k))}, \tag{7}$$

where

$$f(soil(i,j)) = \frac{1}{\varepsilon_z + g(soul(i,j))} \tag{8}$$

and

$$g(soil(i,j)) = \begin{cases} soil(i,j), & \text{if} \quad \min\limits_{l \in V_c}(soil(i,l)) \geqslant 0 \\ soil(i,j) - \min\limits_{l \in V_c}(soil(i,l)) & \text{otherwise} \end{cases} \tag{9}$$

The constant parameter ε_z is a small positive number to prevent division by zero. V_c is a list of visited nodes.

Effectiveness of the algorithm depends on many parameters. Some of them (velocity updating parameters a_v, b_v and c_v, soil updating parameters a_v, b_v and c_v, global soil updating parameter ρ_{IWD}) were fixed in our experiments according to the recommendations of the algorithm's author (1, 0.01, 1, 1, 0.01, 1 and 0.9 accordingly). However, the remaining parameters required settings for a specific task (significance of the best solution in upgrading of soil matrix α, local soil updating parameter ρ_n, initial soil on each edge of the graph *InitSoil*, initial velocity of each drop (*InitVelocity*). Each parameter can take a large number of values, but in this study we did not aimed to fine-tuning the algorithm for a specific task and therefore considered only 24 variants: α = 0.1, 0.3 or 0.5, ρ_n= 0.9 or 0.7, *InitSoil* = 1000 or 10000 and *InitVelocity* = 20 or 200.

1.3. Ant colony optimization algorithm

Ant colony optimization algorithm (ACO) [6] is a nature-inspired optimization metaheuristic based on the behavior and organization of ant colonies in their search for a food. Being almost blind animals, ants anyway can find shortest path from the nest to the food. For information exchange, ants use a ferment, or more exactly pheromone, that they leave on the traversed path. The probability that the ant will choose a certain path is proportional to the amount of pheromone on it.

Solutions in ACO are represented as permutation of n cities and ants chose next city using taboo-list (list of visited cities) and pheromone matrix at every stage. ACO has some adjustable parameters: the evaporation rate (ρ), the relative importance of previous search experience (α) and the relative importance of the distance between cities (β).

Pheromone trails are updated after each ant has completed a tour by formula:

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}, \tag{10}$$

where ρ is parameter such that $(1-\rho)$ is the evaporation and

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^k. \tag{11}$$

Here $\Delta\tau_{ij}^k$ is an amount of the pheromone that the ant k leaves on the edge ij and can be calculated by formula:

$$\Delta\tau_{ij}^k = \begin{cases} \dfrac{Q}{L_k} & \text{if } k \text{ uses edge } ij \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}, \tag{12}$$

where Q is a constant, L_k-length of the k-th ant tour.

Let $\eta_{ij} = 1/d_{ij}$ (d_{ij} is distance between i and j) be called a visibility. The probability of choosing the city j after i is a function of the distance between cities and amount of pheromone on the edge ij and can be expressed as follows

$$p_{ij}^k = \begin{cases} \dfrac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum\limits_{k \in allowed_k} [\tau_{ik}(t)]^\alpha \cdot [\eta_{ik}]^\beta} & \text{if} \quad j \in allowed_k \\ 0 & \text{if} \quad j \notin allowed_k \end{cases}, \tag{13}$$

where $allowed_k$ is a list of unvisited by k-th ant cities.

Some parameters, such as Q and ρ, does not significantly affect the efficiency of the algorithm, so in this paper we will consider only the parameters α and β.

1.4. Genetic algorithm

A well-known genetic algorithm (GA) [5] is based on some principles of evolution, but in the GA for the TSP a chromosome is represented as permutation of the n digits (number of cities). That is why some standard operations have a few changes, but many adjustable parameters remain such as mutation probability, the type of selection, etc.

There are three type of selection in genetic algorithm:

1. Tournament selection (parameter is size of the tournament)

First, we select a random subset of k individuals from the population and then select the best solution out of this subset.

2. Fitness proportional selection

The probability of the i-th individual to be selected is proportional to its fitness function value fit_i and is calculated as follows:

$$p_i = \frac{fit_i}{\sum\limits_{j=1}^{m} fit_j}, \tag{14}$$

where m is a number of individuals.

3. Rank selection (linear or exponential ranking)

Individual i has rank less than individual j ($R_i < R_j$) if i-th fitness function value is less than j-th fitness function value ($fit_i < fit_j$).

3.1. Linear ranking. Probability of the i-th individual to be selected is calculated by formula:

$$p_i = \frac{R_i}{\sum\limits_{k=1}^{m} R_k}, \tag{15}$$

where

$$\sum_{k=1}^{m} p_k = 1, \tag{16}$$

3.2. Exponential ranking. All individuals are assigned a weight according to the value of fitness function so that the best individual has weight $\omega_1 = 1$, $(k+1)$-th individual has weight

$$\omega_{k+1} = \begin{cases} \omega_k, & \text{if} \quad R_{k+1} = R_k \\ \omega_k \cdot \lambda, & \text{otherwise} \end{cases}, \tag{17}$$

where $\lambda \in [0, 1]$.

In this case the probability of the i-th individual to be selected is

$$p_i = \frac{\omega_i}{\sum\limits_{j=1}^{m} \omega_j}, \tag{18}$$

At the stage of recombination in problems on permutations, selected individuals (called parents) devolve part of their chromosomes by using certain rule. Firstly, part of the first parent chromosome is randomly selected and become a part of the offspring. Then the rest of the chromosome is filled with genes in the order in which they appear in the second parent. At the stage of mutation, two genes of the chromosome are randomly selected and swapped.

2. Self-configuration method

All bionic algorithms have many adjustable parameters, for the elimination of this defect one applies the tuning algorithm parameters during its work or, it can be said, an adaptation [2]. In this study we use the self-configuration technique borrowed from [3]. We have investigated the adaptive GA (AGA), which had 8 different selection variants – the tournament selection with the size of the tournament equals 2, 4 or 8, the rank selection with a linear ranking, the rank selection with exponential ranking with parameter λ equal to 0.95, 0.8 or 0.5, and the fitness proportional selection. Also it had 5 different mutation variants - very low, low, medium, high and very high. Adaptive ACO had 4 different variants of parameter α and also 4 variances of the parameter β, in both cases 1, 2, 5 or 10.

Automated choice of algorithms' operators is based on operator probabilistic rates computed during algorithm execution according to the operator's usefulness. Variant of each operator shall be determined separately, let z be a number of variants of operator. In our situation, for example, z=8 in case of selection operator of adaptive GA. If we take a parameter of the algorithm instead of the operator, and different values of this parameter instead of operator variants, then z=4 in case of parameter α of ACO. So here the essence of the adaptation will be described in terms of operators and their variants.

At the beginning of the algorithm execution the probability of selecting all the options of the operator are the same: $p = 1/z$. At each generation, the effectiveness of each operator variant is estimated as the average fitness of off-springs obtained with this operator:

$$averagefitness_i = \frac{\sum_{j=1}^{n_i} f_{ij}}{n_i}, \qquad i = 1, 2, \ldots, z, \tag{19}$$

where n_i is a number of individuals obtained by the i-th variant of the operator; f_{ij} is the value of the fitness function of the j-th individual obtained by the i-th variant of the operator; $averagefitness_i$ is the average value of the fitness function of individuals that were generated by the i-th variant of the operator.

Probability of the operator variant with the largest value of average fitness (i.e. the most effective) increases by $((z-1)\cdot K)/(z\cdot N)$, while the probability of all other operator variants decreases by $K/(z\cdot N)$), where N is the number of past generations of the algorithm, K is a constant, usually equals to 2. In addition, there must be a lower bound of the probability of the operator variant as no one of its probabilities can be equal to zero. If some probability reaches lower bound, this variant stops to give out its share in the benefit of the best variant. The sum of probabilities of all variants of the same operator is always equal to 1. Thus, the probability distribution of the operator variant selection is gradually displaced towards the most effective operator variant from less effective ones.

Thereby, when the algorithm has to create the next offspring from the current population, it firstly must configure settings, i.e. form the list of operators with the use of operator probability distributions. Then the algorithm selects parents with the chosen selection operator, produces an offspring with the chosen crossover operator, mutates this offspring with the chosen mutation probability and puts it into an intermediate population. When the intermediate population is complete, the fitness evaluation is executed and the operator rates (probabilities to be chosen) are updated according to the operator's productivity, i.e. the ratio of the average offspring's fitness obtained with this operator and the offspring population average fitness.

3. Experimental results

Algorithms performance was compared on two well-known benchmark problems Oliver30 and Eil51. To solve these problems all heuristics have got as much resource as algorithm 3-opt requires on average (52800 objective function calculations in case of Oliver30 and 342210 in case of Eil51). Results of numerical experiments averaged over 100 runs are presented in Tab. 1. Also there are results for test problem — grid 5 in 5 cities (15700 objective function calculations), which is very simple but also has a huge number of different optimal routes.

Table 1. Adaptive algorithms comparison with other algorithms on the tasks Oliver30, Eil51

		Grid 5x5		
		Best run	Average run	Standart deviation
3-opt		254,142	255,219	2,78603
IWD	best	254,142	258,648	5,70864
GA	best	254,142	260,576	5,30205
Adaptive GA		254,142	260,954	4,92872
ACO	best	254,142	254,142	0
Adaptive ACO		254,142	254,308	1,1598
		Oliver30		
		Best run	Average run	Standart deviation
3-opt		423,741	428,610	7,4740
IWD	best	254,142	425,500	3,7022
GA	best	254,142	432,356	13,2365
Adaptive GA		423,741	434,239	13,6875
ACO	best	423,741	423,782	0,1675
Adaptive ACO		423,741	426,428	3,7653
		Eil51		
		Best run	Average run	Standart deviation
3-opt		428,872	438,598	5,0160
IWD	best	428,872	437,279	5,57137
GA	best	431,953	447,943	7,5848
Adaptive GA		429,118	449,938	9,94143
ACO	best	429,484	432,732	3,11622
Adaptive ACO		429,484	433,936	2,88955

Fig. 1 shows one example of algorithms work on Oliver30 problem in the case where the better solution (423.741) was found. Here we can see some advantage in the effectiveness of ACO that may be due to the fact that it begins its work with a solution oriented on the distances between cities, but not completely randomly as other algorithms.

Tab. 1 shows that adaptive methods lose conventional with the best settings not so much. At the same time, they do not require testing many variants of parameters. Effectiveness of biology inspired algorithms on a specific task varies considerably depending on the settings. If we solve the problem only once (that is what we actually do with real problems), the effectiveness of the algorithms will be approximately equal to the mean value on this task and not to the best value. Such a comparison on the tasks is shown in Tab. 2, where two lines for each algorithm contain the values of the objective function, found in the best and average settings through all the settings and columns contain averaging over runs, and the standard deviation.

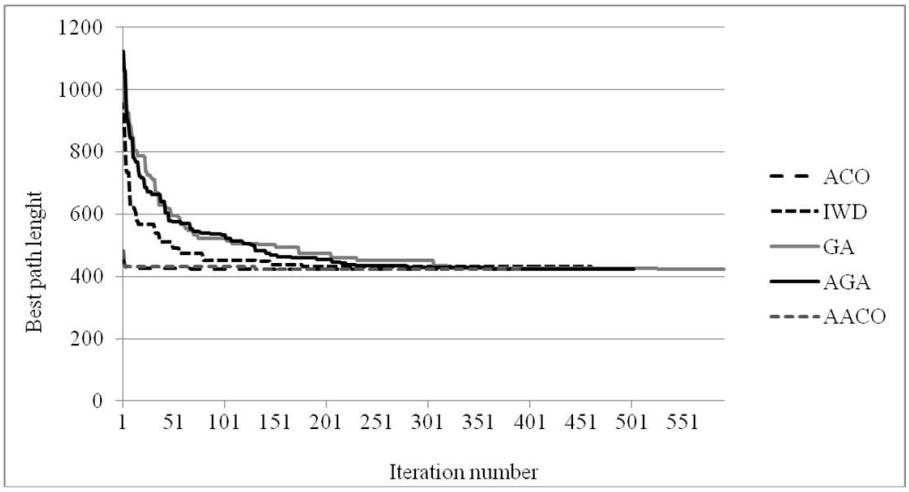

Fig. 1. Examples of the ACO, IWDs, GA and adaptive GA behavior on task Oliver30

Table 2. Adaptive algorithms comparison with other algorithms with different parameters variants on the tasks Oliver30 and Eil51

		Oliver30			Eil51		
		Best run	Average run	Standart deviation	Best run	Average run	Standart deviation
3-opt		423,741	434,61	11,931	435,58	450,346	8,59725
IWD	best	423,741	426,413	3,19925	433,101	442,05	433,101
	average	434,75	434,75	10,1741	450,775	468,64	450,775
GA	best	423,741	431,647	11,3784	442,672	450,913	442,672
	average	424,139	442,982	12,8611	444,923	460,762	444,923
AdaptiveGA		423,741	434,485	12,7539	440,817	457,286	12,0611
ACO	best	423,741	424,04	0,42986	428,872	429,866	428,872
	average	428,684	443,771	11,7151	478,636	496,615	478,636
AdaptiveACO		423,741	426,428	3,7653	429,118	434,634	3,38365

Both adaptive algorithms show good results on these problems as they outperform other algorithms with their settings giving average performance. Although on average adaptive GA cannot outperform other algorithms with their best settings and adaptive ACO cannot outperform conventional ACO with the best settings, one has to realize that it is unknown beforehand which settings of the algorithm on the given task will be the best. There were 16 settings variants of ACO and 24 settings variants of GA and IWDs that means much extra efforts for the determination of these "best" algorithms before they could win adaptive GA. One can use the part of these efforts to improve results of the adaptive GA or adaptive ACO, e.g. by adding computational resources.

Figs. 2–5 show typical examples of operator probabilities interplay in GA and ACO one run. If we deal with real-world problems then very possible situation is the absence of unique best settings. It means that there are different best settings on the different steps of problem solving and this is illustrated clearly on the Fig. 2. In such cases self-configuring algorithms bring much more advantages.

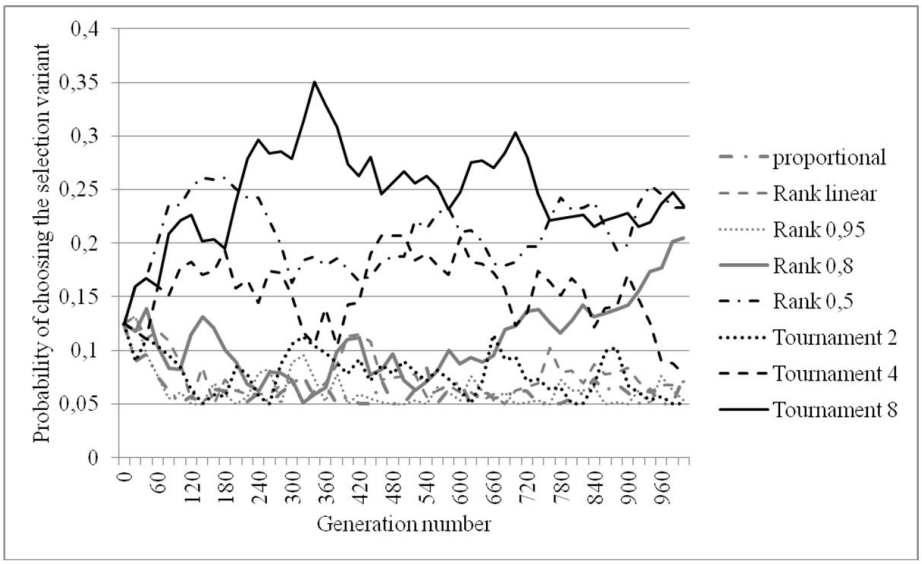

Fig. 2. Probabilities of selection variants through the one run of GA

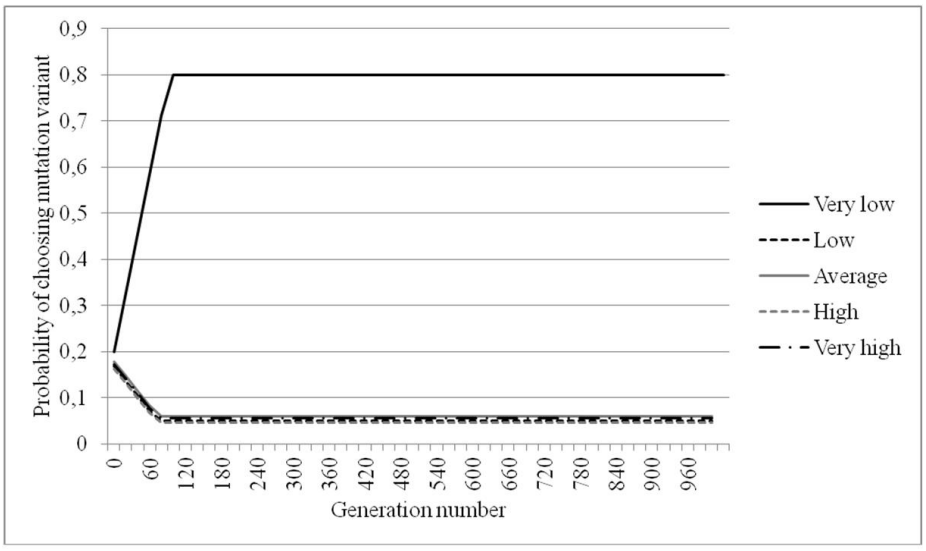

Fig. 3. Probabilities of mutation variants through the one run of GA

4. Conclusion

In this paper we compared performance of some heuristic algorithms of the combinatorial optimization, such as 3-opt algorithm, intelligent water drops algorithm, conventional genetic algorithm and conventional ant colony algorithm with self-configuring (adaptive) genetic and ant colony algorithms.

Our investigations demonstrate that adaptive algorithms are the effective methods of opti-

mization with the remarkable property, which consists in the fact that the user does not have to adjust parameters but can have competitive results in solution quality.

As a future work plans, it can be the comparison with other algorithms, the development of adaptive versions of other algorithms and the use of suggested approach for solving real world problems.

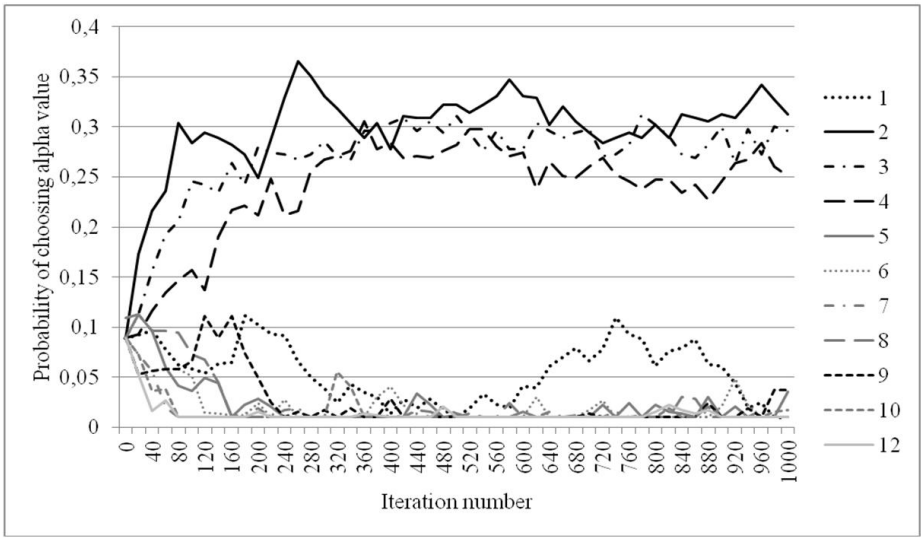

Fig. 4. Probabilities of alpha values through the one run of ACO

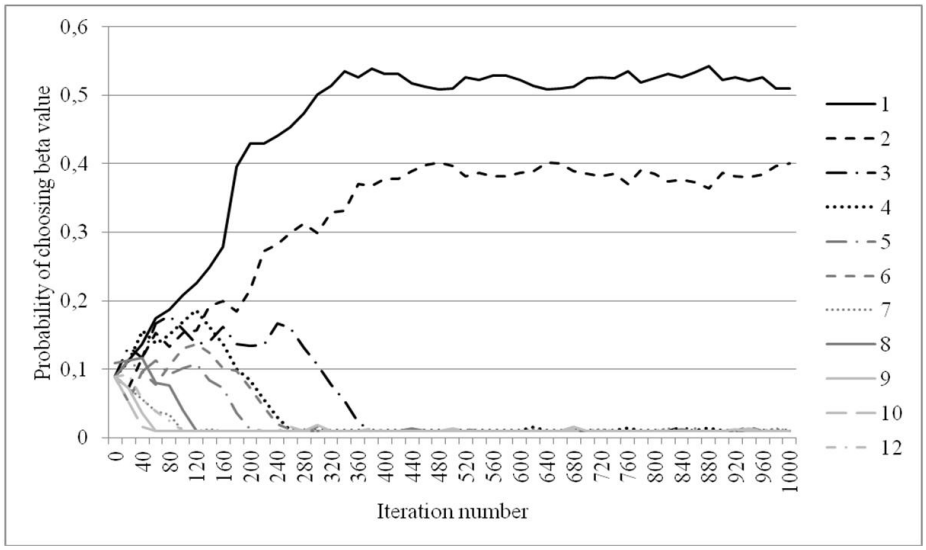

Fig. 5. Probabilities of beta values through the one run of ACO

Research is performed with the support of the Ministry of Education and Science of Russian Federation within State Assignment project no. 2.1680.2017/ПЧ.

References

[1] D.Davendra (Ed), Traveling Salesman Problem, Theory and Applications, In. Tech. Publishing, 2010.

[2] R.Schaefer, C.Cotta, J.Kolodziej, G.Rudolph, Parallel Problem Solving from Nature, PPSN 11th International Conference, Krakow, Poland, 2010, 11–15.

[3] E.Semenkin, M.Semenkina, Self-configuring genetic algorithm with modified uniform crossover operator, Advances in swarm intelligence, ICSI, part 1, LNCS 7331, Springer, Heidelberg, 2012, 414–421.

[4] E.Semenkin, M.Semenkina, Spacecrafts' control systems effective variants choice with self-configuring genetic algorithm, Proceedings of the 9th international conference on informatics in control, automation and robotics, vol. 1, 2012, 84–93.

[5] A.E.Eiben, J.E.Smith, Introduction to evolutionary computing, Springer-Verlag, Berlin, Heidelberg, 2003.

[6] M.Dorigo, L.M.Gambardella, Ant colony system: a cooperative learning approach to the Traveling Salesman Problem, IEEE transactions on evolutionary computation, 1997, 53–66.

[7] S.Lin, B.W.Kernigan, An effective heuristic algorithm for the Traveling-Salesman Problem, *Operations research*, **21**(1973), no. 2, 498–516.

[8] H.Shah-Hosseini, Problem solving by intelligent water drops, Proceedings of IEEE congresson evolutionary computation, Swissotel the Stamford, Singapore, 2007, 3226–3231.

[9] C.H.Papadimitriou, K.Steiglitz, Combinatorial optimization, Algorithms and complexity. Englewood Cliffs, NJ, Prentice-Hall, 1982.

Самоконфигурируемые алгоритмы для задач комбинаторной оптимизации

Ольга Е. Семенкина
Евгений А. Попов
Ольга Э. Семенкина
Сибирский государственный аэрокосмический университет
Красноярский рабочий, 31, Красноярск, 660014
Россия

В данной работе авторы предлагают и исследуют самоконфигурируемые генетический алгоритм (GA) и алгоритм муравьиных колоний (ACO) и применяют их к одной из наиболее известных задач комбинаторной оптимизации — задаче коммивояжера (TSP). Оценка работоспособности предложенных алгоритмов проводится на известных тестовых вариантах TSP, а затем сравнивается с другими эвристиками, а именно с эвристикой Лина-Карнигана (локальный поиск с 3-заменой) и алгоритмом "умных капель воды". Численные эксперименты показывают, что предложенный подход демонстрирует сопоставимую работоспособность. Оба адаптивных алгоритма показывают хорошие результаты на данных задачах, т.к. они превосходят другие алгоритмы с настройками, дающими среднюю эффективность.

Ключевые слова: задача коммивояжера, генетический алгоритм, муравьиный алгоритм, алгоритм "умных капель воды", самоконфигурация.