COMPUTATIONAL SCIENCE AND TECHNIQUES
Volume 5, Number 1, 2017, 560-573

Publisher: Klaipėda University
http://journals.ku.lt/index.php/CST
Online ISSN: 2029-9966
DOI: 10.15181/csat.v5i1.1277

INVESTIGATING SOME STRATEGIES FOR CONSTRUCTION OF INITIAL POPULATIONS IN GENETIC ALGORITHMS

Alfonsas Misevičius, Dovilė Kuznecovaitė
Kaunas University of Technology, Department of Multimedia Engineering

Abstract. Population initialization is one of the important tasks in evolutionary and genetic algorithms (GAs). It can affect considerably the speed of convergence and the quality of the obtained results. In this paper, some heuristic strategies (procedures) for construction of the initial populations in genetic algorithms are investigated. The purpose is to try to see how the different population initialization strategies (procedures) can influence the quality of the final solutions of GAs. Several simple procedures were algorithmically implemented and tested on one of the hard combinatorial optimization problems, the quadratic assignment problem (QAP). The results of the computational experiments demonstrate the usefulness of the proposed strategies. In addition, these strategies are of quite general character and may be easily transferred to other population-based metaheuristics (like particle swarm or bee colony optimization methods).

Keywords: combinatorial optimization, artificial intelligence, metaheuristics, genetic algorithms, quadratic assignment problem.

Introduction

Genetic algorithms (GAs) follow an analogy with the behaviour of genes in populations of live organisms and are based on Darwinian notion of natural selection (Holland, 1975). In the context of combinatorial optimization[1], the basic attributes of GAs are the "population" of solutions and the special operators called "selection", "crossover" ("recombination"), "mutation", and "replacement".

The solutions of an optimization problem correspond to (chromosomes of) individuals of a biological system and the cost of a solution (the value of the objective function) is associated with the fitness of an individual. Traditionally, the GAs manipulate with solutions which are represented by strings of bits, but also other encodings (for example, the permutations of integers) can be used for particular optimization problems. The goal is to produce better and better solutions by applying iterative process of repeating virtual generations which consist of the above mentioned operators. The global optimality of the obtained solutions can not be guaranteed, however the near-optimal solutions of tolerable quality are usually found within

[1] Recall that a combinatorial optimization problem can be formally described by a pair (S, f), where $S = \{s_1, s_2, \ldots, s_i, \ldots\}$ is a finite set of feasible solutions (a search space) and $f: S \to R$ is a real-valued objective (cost) function. We suppose that f seeks a global minimum, then the goal is to find a solution $s^* \in S$ such that $s^* \in S^* = \left\{ s^* \mid s^* = \arg\min_{s \in S} f(s) \right\}$.

reasonable computation time. (More thorough description of the principles of GAs can be found in (Goldberg, 1989; Reeves, Rowe, 2001; Sivanandam, Deepa, 2008).

One of the distinctive aspects of GAs is that these algorithms always operate on a whole population of solutions (rather than a single solution), which largely improves the chance of seeking the high quality results. The other important advantages of GAs include: parallelism, liability, ability to explore wider search spaces and handle complex fitness landscapes, as well as potential of employment for a broad variety of optimization problems. Despite of this, the GAs may also suffer some limitations, first of all, the loss of genetic variance (genetic drift), premature convergence, stalled evolution. To try to overcome these barriers, various conceptual enhancements have been proposed, for example, messy GAs (Goldberg et al., 1989), parallel GAs (Cantú-Paz, 2001), hybrid GAs (memetic algorithms) (Krasnogor, Smith, 2005; El-Mihoub et al., 2006), affinity GAs (Zhao, Gao, 2007). The researchers have also considered the fine-tuning of the parameter settings of GAs (Schaffer, 1989; Misevičius et al., 2009), the amendment of the genetic operators (Fox, McMahon, 1991), and introduction of other new modifications (among them, special replacement strategies (Wu, Ji, 2007), GAs with random immigrants (Cheng, Yang, 2010), diversification mechanisms to increase diversity (Misevicius, 2008), using micro-populations (Kazarlis, 2001), extra/differential improvement (Drezner, Misevičius, 2013)).

It is commonly known that the efficiency of the traditional GAs depends mostly on the standard genetic operators (selection, crossover/mutation, replacement), which are of the explorative nature; in modern, hybrid GAs, however, more attention is to be paid rather to the exploitative, improving operations (Misevičius, Rubliauskas, 2008; Drezner, Misevičius, 2013), including the enhanced improvement procedures used to create the high quality populations. In this paper, we propose, in particular, several simple heuristic strategies for construction of the initials populations to see how the structure and quality of these populations affect the resulting performance and the final results of the genetic algorithm.

The rest of this paper is structured as follows. In Section 1, some simple strategies for creation of the initial population are discussed. Section 2 describes the implementation of the proposed strategies for the quadratic assignment problem (QAP), along with the results of the computational experiments. The paper is completed with concluding remarks.

1. Construction of initial populations in genetic algorithms: some simple strategies

1.1. Random generation

In this particular case, we do not use any improvement at all. So, the genetic algorithms starts from a pure random population.

1.2. Uniform improvement

1. 2. 1. Uniform improvement (variant 1)

In this case, the initial population is constructed in two main steps: 1) random generation (see above), 2) improvement of the individuals. During the second step, all the existing members of the population created in the first step undergo an improvement process. Any local-search-methodology-based algorithm can be used for this task, for example, descent/greedy local search,

simulated annealing, tabu search, etc. The time spent for the improvement is essentially the same for every population member, so the resultant population may be thought of as uniform like (with respect to the fitness of the individuals, i.e. the values of the objective function), given that, during the first step, the population is generated randomly, uniformly.

The only control parameter needed for the uniform improvement procedure is the number of iterations (Q) of the improvement algorithm.

1. 2. 2. Uniform improvement (variant 2)

This variant of improvement is very similar to that described in previous section. The basic distinction is that the number of improvement iterations is more or less increased to obtain even higher quality population. We can easily regulate the improvement time/extensity and, consequently, the quality of the population by means of a pre-defined parameter (coefficient), C. The resulting number of improvement iterations (during the initialization phase only) is thus equal to the product of C and Q, where Q is the standard number of improvement iterations (see above). Note that if the time-expensive improvement heuristics are applied, then the overall GA's execution time (or the corresponding total number of generations) should be accordingly decreased to keep the run time fixed and also allow a fair comparison with other variants (strategies).

This strategy seems to slightly resemble a compounded approach proposed in (Drezner, 2005), where several starting populations (sub-populations) are maintained and the individuals of every distinct population are independently pre-improved.

1.3. Non-uniform improvement

The remaining strategies differ from the above ones in the sense that they try to construct (at least virtually) the non-uniform like populations (in terms of the fitness of the individuals) instead of the uniform like populations. Four simple heuristic construction rules are briefly described in the subsequent sections.

1. 3. 1. Non-uniform improvement (variant 1)

Our first quite spontaneous idea was to vary the computation time, i.e. the number of improvement iterations at the population initialization stage according to an elegant rule so that the produced individuals of the obtained population are distributed (in terms of the individuals' fitness) according to some law (for example, similar in some sense to a normal (Gaussian) like distribution law). With our approach, we, of course, are far from the theoretical schemes and, instead of producing a normally-distributed population in a straightforward way, we are speaking at most of the heuristic procedure for a conceptual simulation of a pseudo normal distribution only. The outline of our procedure is as follows:

(a) generate the starting (preliminary) population randomly, uniformly;
$i \leftarrow 1$;

(b) choose the ith member of the existing population and apply $i*K$ improving iterations (here, K is an a priori parameter). Note: for the improvement, any local search-based algorithm can be applied;
$i \leftarrow i + 1$;

(c) if i is less than or equal to the pre-specified size of the population, P, then go to

(b) and continue the process of the next iteration;
otherwise, the procedure is finished.

Obviously, for the (lexicographically) last (Pth) population member, $P*K$ improving iterations are applied and we may think of this member as some hero (leader) of the population.

1. 3. 2. Non-uniform improvement (variant 2)

The modified non-uniform improvement procedure is designed with the intension of a more flexible control of the improvement process. In addition, the non-uniformity of the population is allowed to be achieved in a more gentle way. The procedure is as follows:

(a) generate the starting (preliminary) population randomly, uniformly;
$i \leftarrow 1$;
(b) choose the ith member of the existing population and apply $(i + L)*K$ improving iterations (here, L is a parameter);
repeat the same for the $(i + 1)$th population member;
$i \leftarrow i + 2$;
(c) if i is less than the pre-specified size of the population, P, then go to (b) and continue the process;
otherwise, the procedure is finished.

Note: it assumed that the procedure operates with a population whose size is an even integer.

1. 3. 3. Non-uniform improvement (variant 3 – differential improvement)

The concept of "differential improvement" comes from (Drezner, Misevičius, 2013) and it is initially proposed by Prof. Z. Drezner during the collaborative work on the genetic algorithms with the authors of the current paper. The heart of the concept is to perform more extensive improvement on some selected solutions. In this paper, we have slightly adapted the basic idea so that some pre-determined solutions are improved more than the others. The heuristic process to achieve this effect is very simple:

(a) generate the starting (preliminary) population randomly, uniformly;
$i \leftarrow 1$;
(b) choose the ith member of the existing population;
if the index i is an odd integer, then apply M improving iterations (M is a parameter);
otherwise, apply $M*2$ improving iterations;
$i \leftarrow i + 1$;
(c) if i is less than or equal to the pre-specified size of the population, P, then go to (b) and continue the process;
otherwise, the procedure is finished.

The population members with odd indices ($i = 1, 3, ...$) may be associated with virtual females, the members with even indices ($i = 2, 4, ...$) — with virtual males (or vice versa).

1. 3. 4. Non-uniform improvement (variant 4 – extra improvement)

In this variant, the random population is firstly created, then the population is improved uniformly. After this, an extra improvement step takes place. The specific point here is that only one member of the population is selected for the additional extensive improvement (extra improvement). We follow a special rule of selection proposed by Z. Drezner in (Drezner, Misevičius, 2013). In particular, R population members are selected at random (with possible repetition) from the existing population. The improvement algorithm with extra N iterations is then applied to the best selected solution. Here, R ($R \leq P$), N are parameters of the user's choice.

All the proposed procedures, except the random generation, may be seen as some kind of "initial burst" ("initial invasion"), where the goal is to obtain an elite population of the outstanding quality. Such a constructive, positive invasion occurs only once before starting the genetic process and this allows good starting conditions for the convergence speed of GA. Remind that, in this case, local-search-based improvement heuristics are required, which might be rather time-consuming. The compensation would be based on maintaining compact, small-sized populations, where the population size is sacrificed for the extensive improvement.

In Fig. 1, we have depicted the graphical representations of the fitness landscapes of the populations obtained using different population initialization procedures. The corresponding different character of created populations can be observed quite clearly.

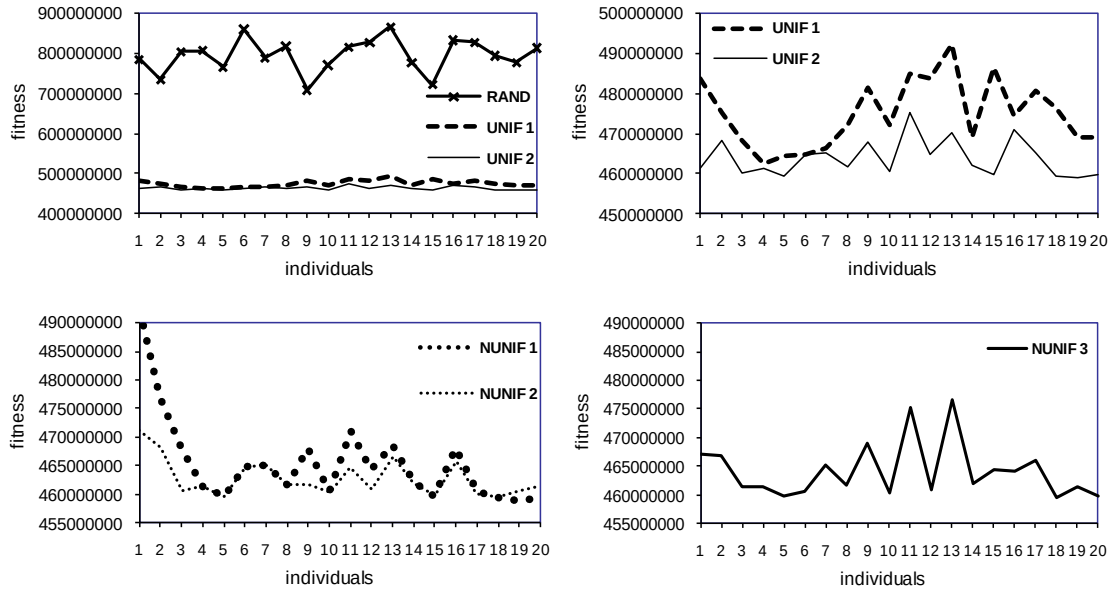

Fig. 1. **Graphical illustrations of the population fitness landscapes: (a, b) random population (RAND) and populations after uniform improvement (variant 1, variant 2) (UNIF 1, UNIF 2); (c, d) populations after non-uniform improvement (variant 1, variant 2, variant 3) (NUNIF 1, NUNIF 2, NUNIF 3). (The landscapes are for the quadratic assignment problem instance tai50b from the library of the QAP instances QAPLIB (Burkard et al., 1997) (also see Section 2.1). The population size is equal to 20)**

2. Computational experiments with the quadratic assignment problem

2.1. The quadratic assignment problem

In order to evaluate the behaviour of the proposed procedures more thoroughly, the computational experiments have been carried out on the quadratic assignment problem (Koopmans, Beckmann, 1957; Çela, 1998). The QAP is formulated as follows. Given two matrices $A = (a_{ij})_{n\times n}$ and $B = (b_{kl})_{n\times n}$ and the set Π of permutations of the integers from 1 to n, find a permutation $\pi \in \Pi$ that minimizes

$$z(\pi) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)} \,. \tag{1}$$

The QAP is a typical example of combinatorial optimization problems, where solutions are represented by permutations (i.e. $S \equiv \Pi = \{\pi \mid \pi = (\pi(1), \pi(2), \ldots, \pi(n)), \pi(i) \in \{1, \ldots, n\}, i = 1, \ldots, n, \pi(i) \neq \pi(j), i, j = 1, \ldots, n, i \neq j\}$)[2] and the objective function is described according to the above formula. The QAP is used in many applications (including computer-aided design, factory/office layout design, network design). It also serves as a suitable platform for testing various optimization methods and the hybridized genetic algorithms are among the most successful heuristic techniques for this problem (Drezner, 2003; Misevicius, 2004).

In our experiments, we have used the benchmark QAP instances from the QAP library QAPLIB (Burkard et al., 1997) (also see the web site http://www.seas.upenn.edu/qaplib). In particular, the following representative types of the instances were used:

a) random instances (in QAPLIB, they are denoted by tai20a, tai25a, tai30a, tai35a, tai40a, tai50a, tai60a, tai80a, and tai100a);

b) real-life like instances (these instances are denoted by tai20b, tai25b, tai30b, tai35b, tai40b, tai50b, tai60b, tai80b, tai100b, and tai150b).

These types of instances have been proposed by E. Taillard in (Taillard, 1991) and (Taillard, 1995). The instances tai*a are generated randomly according to the uniform distribution, while the instances tai*b are artificially designed in such a way that they look like the real-world problems from practical applications.

2.2. The genetic algorithm used in the experiments

The high-level description of the genetic algorithm is presented in Fig. 2.

[2] In this case, the solution (permutation) π can be directly associated with a chromosome so that the single solution element $\pi(j)$ corresponds to a gene occupying the jth locus of the chromosome.

```
Genetic Algorithm
 0: create the initial population of fixed size P, depending on the population
    initialization option (variant);
 1: randomly choose R population members and (try to) extra improve
    the best one out of R selected members;
 2: select parents π′, π″ from the existing population;
 3: apply crossover to π′ and π″, get the offspring π‴;
 4: improve the offspring π‴, get the improved offspring π⁗;
 5: if the offspring π⁗ is worse than both its parents (π′ and π″),
    then it is extra improved;
 6: update (replace) the population;
 7: if the current generation number is less than the
    pre-defined maximum number of generations, G, then go to 1,
    otherwise the algorithm is stopped.
```

Fig. 2. **High-level description of the genetic algorithm**

The algorithm components and parameters used in the experiments are as follows.

i) The population initialization option (variant) can take on seven different values: 0 (random population); 1 (uniform improvement (variant 1)); 2 (uniform improvement (variant 2)); 3 (non-uniform improvement (variant 1)); 4 (non-uniform improvement (variant 2)); 5 (non-uniform improvement (variant 3 – differential improvement)); 6 (non-uniform improvement (variant 4 – extra improvement)). In all the cases, the population size (P) is very compact and it is equal to 20.

ii) The improvement algorithm is the fast iterated tabu search (ITS) based on the enhanced tabu search (Misevicius, 2005). The run time of the ITS algorithm is controlled by the parameters Q, τ, where Q is the number of iterations (search extensity) and τ is the depth of the search (search intensity). The value of τ is kept fixed in our experiments; it is equal to n^2 for the tai*a instances and n for the tai*b instances (this is due to more apparent hardness of the random instances).

The ITS algorithm uses random mutations (perturbations), where mutations are applied every τ iterations. The variable tabu tenure is applied; more precisely, the tabu tenure varies in an oscillating ($\vee\vee\vee\vee\vee$) manner. The amplitude of the oscillations is equal to $h_{high}*n - h_{low}*n$, where $h_{high}*n$, $h_{low}*n$ are the maximum and minimum values of the tabu tenure (here, h_{high}, h_{low} ($0 < h_{low} < h_{high} \leq 1$) are the coefficients and n denotes the problem size).

iii) The number of generations, G, of the genetic algorithm is equal to $\max\{30, n/2\}$. (This is true for the case of the random initial population. For the remaining variants, the decreased value of G is applied so that the total run time is kept approximately the same.)

iv) For parent selection, a rank-based selection rule (Tate, Smith, 1995) is applied. The cohesive like crossover operator similar to the one in (Drezner, 2003) is used. One offspring is generated at each generation.

v) The extra improvement is applied in two cases. Firstly (also see (Drezner, Misevičius, 2013)), R population members are randomly chosen from the current population and the best one out of R selected members is (attempted to be) extra improved (in our case, we use the extra number of $5Q$ improvement iterations). Secondly (also see (Misevičius, D.Rubliauskas, 2008)), after producing and improving the offspring, it is checked if the new offspring is better than its parents; if this is not the case, then the offspring is again tried to

be additionally improved by allowing an increased number of improvement iterations (in this case, we use the extra number of $10Q$ iterations). For determining the values of the parameter R, see Section 2.3.

vi) At the population update (replacement) phase, we apply a steady state, elitist strategy, where the best offspring replaces the worst population member. In addition, if the diversity of the obtained population is lost, then a special restart mechanism is used (for more details, see (Misevicius, 2008)).

2.3. The results of the experiments

We have experimented with the following algorithm variants: RAND — random generation (a variant without using any improvement of the initial population); UNIF 1 — uniform improvement (variant 1 (standard variant)); UNIF 2 — uniform improvement (variant 2 (extended variant)); NUNIF 1 — non-uniform improvement (variant 1); NUNIF 2 — non-uniform improvement (variant 2); NUNIF 3 — non-uniform improvement (variant 3 (differential improvement)); NUNIF 4 — non-uniform improvement (variant 4 (extra improvement)). The algorithms were programmed in Pascal (using Free Pascal compiler). The experiments were performed on a personal computer with an Intel Pentium 3 GHz single-core processor.

As a performance criterion for the algorithms, we use the average relative deviation ($\bar{\delta}$) of the obtained solutions from the best known (pseudo-optimal) solution (BKS). It is defined by the formula: $\bar{\delta} = 100(\bar{z} - z^{\diamond})/z^{\diamond}$ [%], where $\bar{z}$ is the average objective function value over 10 runs of the algorithm and $z^{\diamond}$ denotes the best known value (BKV) of the objective function. (BKVs are from QAPLIB.)

Firstly, we experimented with the different values of the parameter R to determine the most preferable values for the further experimentation. We used the following values of R: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 15, 20. We used the extra improvement variant for the population initialization with the corresponding values. The values of the parameters Q, N were as follows: $Q = 5$, $N = 10Q$. The results of the experiments are presented in Tables 1, 2 (only results for $R = 1, 2, 3, 5, 10, 15, 20$ are represented).

In Fig. 3, we graphically summarize the results from Tables 1, 2 and illustrate the influence of the values of the parameter R on the quality of solutions for the random and real-life like instances. It can be seen that the trend curve is parabolic shaped, but, on the whole, the quality of the obtained solutions is quite insensitive to the values of R. However, we have observed that, for the random instances, the higher values of R ($R = 10, 15, 20$) are preferable to the lower ones. (We used $R = 15$ in the further experiments.) At the same time, we could not reveal a preferable region of good values of R for the real-life like instances. We have decided to use $R = 2$ in the further experimentation. The motivation was that, firstly, we have obtained the best results with this particular value during the preliminary experiments; secondly, our intension was to pay more attention to the explorative capabilities of the algorithm by allowing less "discriminative" selection rule using smaller value of R.

Table 1. Results of the experiments with different values of the parameter R (I)

Instance‡	BKV	$\bar{\delta}$							Time (sec.)‡‡
		R=1	R=2	R=3	R=5	R=10	R=15	R=20	
Tai20a	703482	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*3*
Tai25a	1167256	**0.000**	**0.000**	0.037	0.073	0.037	0.037	0.052	*6*
Tai30a	1818146	0.041	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*12*
Tai35a	2422002	0.087	0.051	0.073	0.033	0.018	0.018	**0.000**	*23*
Tai40a	3139370	**0.187**	0.263	0.283	0.248	0.248	0.229	0.247	*50*
Tai50a	4938796	0.463	0.519	0.523	**0.457**	0.520	0.505	0.485	*125*
Tai60a	7205962	0.519	0.501	**0.422**	0.518	0.451	0.492	0.516	*260*
Tai80a	13499184	0.544	**0.494**	0.584	0.577	0.565	0.540	0.559	*900*
Tai100a	21052466	0.430	0.421	0.410	0.441	0.398	**0.394**	0.413	*2500*
	Average:	0.252	0.250	0.259	0.261	0.249	**0.246**	0.252	

‡ the numeral in the instance name indicates the size of the problem; ‡‡ average CPU time per run is given.

In the further experiments, we have compared the different strategies for the population initialization (RAND, UNIF 1, UNIF 2, NUNIF 1, NUNIF 2, NUNIF 3, NUNIF 4). The values of the parameters (Q, C, K, L, M, N) were as follows: $Q = 5$, $C = 4$, $K = 2$, $L = 5$, $M = 15$, $N = 10CQ$. The results of the comparison are presented in Tables 3, 4.

Table 2. Results of the experiments with different values of the parameter R (II)

Instance	BKV	$\bar{\delta}$							Time (sec.)
		R=1	R=2	R=3	R=5	R=10	R=15	R=20	
Tai20b	122455319	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*1.1*
Tai25b	344355646	**0.000**	**0.000**	**0.000**	**0.000**	0.007	**0.000**	**0.000**	*1.5*
Tai30b	637117113	**0.000**	0.011	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*2.6*
Tai35b	283315445	**0.000**	0.018	0.010	0.010	0.019	**0.000**	0.048	*4*
Tai40b	637250948	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*6*
Tai50b	458821517	0.037	**0.007**	0.078	0.076	0.035	0.034	0.079	*11*
Tai60b	608215054	0.024	0.023	0.027	0.018	**0.013**	0.020	0.015	*21*
Tai80b	818415043	0.314	0.230	0.410	0.336	0.263	**0.192**	0.216	*42*
Tai100b	1185996137	0.092	0.095	0.157	0.129	0.103	**0.087**	0.121	*100*
Tai150b	498896643	0.192	0.201	**0.178**	0.258	0.275	0.355	0.352	*480*
	Average:	0.066	**0.059**	0.086	0.083	0.072	0.069	0.083	

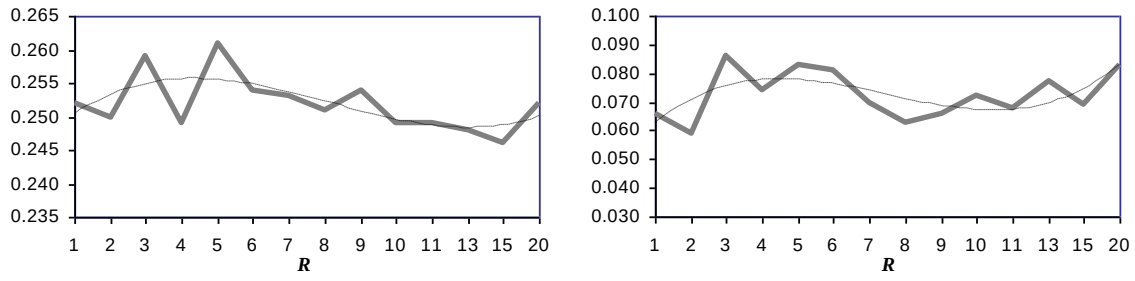

Fig. 3. **Graphical illustrations of the behaviour of the influence of the parameter R on the quality of solutions: (a) random instances (tai*a), (b) real-life like instances (tai*b).**
(Deviation ($\bar{\delta}$) is averaged over all the random and real-life like instances, respectively)

We were also interested in the long-time behaviour of the most promising algorithm variants (UNIF 2, NUNIF 2), so we have performed more extensive experiments, where the run time is roughly doubled. We experimented with the procedure NUNIF 2 for the both types of instances and the procedure UNIF 2 for the tai*b instances. In particular, we have examined the following variants: $Q = 10$ (UNIF 2^1, NUNIF 2^1), $Q = 11$ (UNIF 2^2, NUNIF 2^2), $Q = 12$ (UNIF 2^3, NUNIF 2^3), $Q = 15$ (NUNIF 2^4), $Q = 20$ (NUNIF 2^5), $Q = 25$ (NUNIF 2^6), $Q = 30$ (NUNIF 2^7). The values of the parameters K, L were also increased (we used $K = 4$, $L = 10$) and the number of generations was accordingly decreased. In Tables 5, 6, we present the obtained results, which demonstrate a very satisfactory effect of using the non-uniform improvement (variant NUNIF 2) and the extensive uniform improvement (variant UNIF 2).

Table 3. Results of the experiments with different initial populations (I)

Instance	BKV	$\bar{\delta}$							Time (sec.)
		RAND	UNIF 1	UNIF 2	NUNIF 1	NUNIF 2	NUNIF 3	NUNIF 4	
Tai20a	703482	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*5*
Tai25a	1167256	0.073	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*8*
Tai30a	1818146	0.002	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*18*
Tai35a	2422002	**0.000**	**0.000**	**0.000**	0.048	**0.000**	**0.000**	**0.000**	*34*
Tai40a	3139370	0.300	0.299	0.201	0.201	**0.196**	0.256	0.260	*75*
Tai50a	4938796	0.507	0.369	0.407	0.378	0.415	**0.366**	0.411	*190*
Tai60a	7205962	0.426	**0.406**	0.448	0.412	0.425	0.424	0.422	*390*
Tai80a	13499184	0.539	0.519	0.488	0.524	0.484	**0.483**	0.514	*1300*
Tai100a	21052466	0.421	0.348	0.373	0.374	0.377	0.385	**0.343**	*3700*
	Average:	0.252	0.216	0.213	0.215	**0.211**	0.213	0.217	

Table 4. Results of the experiments with different initial populations (II)

Instance	BKV	$\bar{\delta}$							Time (sec.)
		RAND	UNIF 1	UNIF 2	NUNIF 1	NUNIF 2	NUNIF 3	NUNIF 4	
Tai20b	122455319	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*1.6*
Tai25b	344355646	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*2.2*
Tai30b	637117113	**0.000**	0.001	**0.000**	**0.000**	**0.000**	**0.000**	0.012	*4.0*
Tai35b	283315445	**0.000**	**0.000**	0.029	0.033	**0.000**	0.010	0.037	*6*
Tai40b	637250948	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*9*
Tai50b	458821517	0.109	**0.001**	0.003	0.002	**0.001**	0.002	0.003	*16*
Tai60b	608215054	0.005	**0.003**	0.005	0.007	0.010	0.005	0.008	*32*
Tai80b	818415043	**0.088**	0.293	0.107	0.169	0.108	0.187	0.276	*62*
Tai100b	1185996137	**0.035**	0.045	0.095	0.065	0.063	0.072	0.061	*150*
Tai150b	498896643	0.241	0.231	0.206	0.250	0.254	**0.224**	0.235	*700*
	Average:	0.048	0.057	0.045	0.053	**0.044**	0.050	0.063	

3. Concluding remarks

In this paper, some heuristic strategies (procedures) for construction of initial populations in genetic algorithms within the context of combinatorial optimization are proposed. The intension was to examine the behaviour of the genetic algorithm, depending on the different ways of creation of an initial population. Three main strategies have been tested: random generation, uniform improvement, and non-uniform improvement.

These strategies were implemented and tested on one of the hard combinatorial optimization problems, the quadratic assignment problem. The results obtained from the experiments with the Taillard instances from the QAP library QAPLIB demonstrate promising potential of using the extensified search at the early phase of initialization of population. It is also observed (especially, for the random Taillard problems) that it is of high importance to operate with diversified, "asymmetric" initial populations, where some members are worse and others better.

Our proposed strategies are quite simple and universal, so they can be easily replicated and transferred to other types of (combinatorial) optimization problems.

It might also be worthy to further investigate the other possible variations of the proposed strategies for construction of initial populations.

Table 5. Results of the extensive experiments with the non-uniform initial populations

Instance	BKV	$\bar{\delta}$							Time (sec.)
		NUNIF 2^1	NUNIF 2^2	NUNIF 2^3	NUNIF 2^4	NUNIF 2^5	NUNIF 2^6	NUNIF 2^7	
Tai20a	703482	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*10*
Tai25a	1167256	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*16*
Tai30a	1818146	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*37*
Tai35a	2422002	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*70*
Tai40a	3139370	0.099	0.105	0.117	**0.096**	0.155	0.098	0.106	*155*
Tai50a	4938796	0.324	0.332	0.303	0.281	0.333	0.316	**0.239**	*390*
Tai60a	7205962	**0.233**	0.299	0.306	0.258	0.282	0.345	0.340	*800*
Tai80a	13499184	**0.403**	0.403	0.458	0.433	0.439	0.418	0.404	*2650*
Tai100a	21052466	0.292	0.308	0.280	0.285	0.285	0.280	**0.277**	*7500*
	Average:	0.153	0.161	0.163	**0.150**	0.166	0.162	0.152	

Table 6. Results of the extensive experiments with the uniform and non-uniform initial populations

Instance	BKV	$\bar{\delta}$							Time (sec.)
		UNIF 2^1	UNIF 2^2	UNIF 2^3	NUNIF 2^1	NUNIF 2^2	NUNIF 2^3	NUNIF 2^4	
Tai20b	122455319	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*3.2*
Tai25b	344355646	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*4.5*
Tai30b	637117113	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*8.0*
Tai35b	283315445	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*12*
Tai40b	637250948	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*20*
Tai50b	458821517	0.001	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*35*
Tai60b	608215054	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	*68*
Tai80b	818415043	0.007	0.003	0.001	0.001	**0.000**	0.001	**0.000**	*130*
Tai100b	1185996137	0.012	0.009	**0.000**	0.011	0.002	**0.000**	**0.000**	*310*
Tai150b	498896643	0.046	0.020	**0.017**	0.038	0.026	0.037	0.025	*1450*
	Average:	0.007	0.003	**0.002**	0.005	0.003	0.004	0.003	

References

Burkard, R.; Karisch, S.; Rendl, F. (1997). "QAPLIB — a quadratic assignment problem library," Journal of Global Optimization 10: 391–403. [Žr. taip pat prieigą per internetą: <http://www.seas.upenn.edu/qaplib/>.]

Cantú-Paz, E. (2001). "Migration policies, selection pressure, and parallel evolutionary algorithms," Journal of Heuristics 7: 311–334.

Çela, E. (1998). The Quadratic Assignment Problem: Theory and Algorithms. Dordrecht: Kluwer.

Cheng, H.; Yang, S. (2010). "Genetic algorithms with immigrants schemes for dynamic multicast problems in mobile ad hoc networks," Engineering Applications of Artificial Intelligence 23: 806–819.

Drezner, Z. (2003). "A new genetic algorithm for the quadratic assignment problem," INFORMS Journal on Computing 15: 320–330.

Drezner, Z. (2005). "Compounded genetic algorithms for the quadratic assignment problem," Operations Research Letters 33: 475–480.

Drezner, Z.; Misevičius, A. (2013). "Enhancing the performance of hybrid genetic algorithms by differential improvement," Computers & Operations Research 40: 1038–1046.

El-Mihoub, T.A.; Hopgood, A.A.; Nolle, L.; Battersby, A. (2006). "Hybrid genetic algorithms: a review," Engineering Letters 13: 124–137.

Fox, B.R.; McMahon, M.B. (1991). "Genetic operators for sequencing problems," In Rawlins, G. (ed.), Foundations of Genetic Algorithms. San Mateo: Morgan-Kaufmann, 284–300.

Goldberg, D.E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Reading: Addison-Wesley.

Goldberg, D.E.; Deb, K.; Korb, B. (1989). "Messy genetic algorithms: Motivation, analysis and first results," Complex Systems 3: 493–530.

Holland, J.H. (1975). Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press.

Kazarlis, S.A.; Papadakis, S.E.; Theocharis, J.B. (2001). "Microgenetic algorithms as generalized hill-climbing operators for GA optimization," IEEE Transactions on Evolutionary Computation 5: 204–217.

Koopmans, T.; Beckmann, M. (1957). "Assignment problems and the location of economic activities," Econometrica 25: 53–76.

Krasnogor, N.; Smith, J. (2005). "A tutorial for competent memetic algorithms: model, taxonomy, and design issues," IEEE Transactions on Evolutionary Computation 9: 474–488.

Misevicius, A. (2004). "An improved hybrid genetic algorithm: new results for the quadratic assignment problem," Knowledge-Based Systems 17: 65–73.

Misevicius, A. (2005). "A tabu search algorithm for the quadratic assignment problem," Computational Optimization and Applications 30: 95–111.

Misevicius, A. (2008). "Restart-based genetic algorithm for the quadratic assignment problem," In Bramer, M.; Coenen, F.; Petridis, M. (eds.), Research and Development in Intelligent Systems, Proceedings of AI-2008, the Twenty-eighth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence. London: Springer, 91–104.

Misevičius, A.; Rubliauskas, D. (2008). "Enhanced improvement of individuals in genetic algorithms," Information Technology and Control 37: 179–186.

Misevičius, A.; Rubliauskas, D.; Barkauskas, V. (2009). "Some further experiments with the genetic algorithm for the quadratic assignment problem," Information Technology and Control 38: 325–332.

Reeves, C.R.; Rowe, J.E. (2001). Genetic Algorithms: Principles and Perspectives. Norwell: Kluwer.

Schaffer, J.D.; Caruana, R.A.; Eshelman, L.J. (1989). "A study of control parameters affecting online performance of genetic algorithms," In Schaffer, J.D. (ed.), Proceedings of the 3rd International Conference on Genetic Algorithms. San Mateo: Morgan Kaufmann, 51–60.

Sivanandam, S.N.; Deepa, S.N. (2008). Introduction to Genetic Algorithms. Berlin-Heidelberg-New York: Springer.

Taillard, É.D. (1991). "Robust taboo search for the QAP," Parallel Computing 17: 443–455.

Taillard, É.D. (1995). "Comparison of iterative searches for the quadratic assignment problem," Location Science 3: 87–105.

Tate, D.M.; Smith, A.E. (1995). "A genetic approach to the quadratic assignment problem," Computers & Operations Research 1: 73–83.

Wu, Y.; Ji, P. (2007). "Solving the quadratic assignment problems by a genetic algorithm with a new replacement strategy," Proceedings of World Academy of Science, Engineering and Technology (WASET) 24: 310–314.

Zhao, X.; Gao, X.-S. (2007). "Affinity genetic algorithm," Journal of Heuristics 13: 133–150.

Alfonsas Misevičius gimė 1962 m. Lietuvoje, Marijampolės rajone. 1986 m. su pagyrimu baigė tuometinį Kauno politechnikos institutą. 1996 m. Kauno technologijos universitete (KTU) apgynė techn. m. daktaro disertaciją. Nuo 1998 m. – docentas, nuo 2011 m. – KTU Informatikos fakulteto Multimedijos inžinerijos katedros profesorius. Yra publikavęs arti 100 mokslo ir mokymo metodikos darbų kompiuterių mokslo, optimizavimo algoritmų temomis. Moksliniai interesai: dirbtinis intelektas; optimizavimas ir euristiniai optimizavimo algoritmai; genetiniai/evoliuciniai algoritmai.

Dovilė Kuznecovaitė gimė 1989 m. Lietuvoje, Kaune. 2014 m. baigė KTU Informatikos fakultetą ir įgijo informatikos magistro laipsnį. Šiuo metu yra KTU Informatikos fakulteto doktorantė. Moksliniai interesai: euristiniai algoritmai, kombinatorinis optimizavimas.

KAI KURIŲ PRADINIŲ POPULIACIJŲ KONSTRAVIMO STRATEGIJŲ (PROCEDŪRŲ) GENETINIUOSE ALGORITMUOSE TYRIMAS

Alfonsas Misevičius, Dovilė Kuznecovaitė

Santrauka

Pradinių populiacijų formavimas yra svarbus etapas evoliuciniuose ir genetiniuose algoritmuose. Nuo suformuotos populiacijos kokybės priklauso algoritmų konvergavimo greitis ir gaunami rezultatai. Šiame straipsnyje ir nagrinėjamos kai kurios galimos pradinių populiacijų konstravimo strategijos (procedūros), siekiant išsiaiškinti, kaip gali būti įtakojamas genetinių algoritmų (GA) efektyvumas, priklausomai nuo pradinių populiacijų konstravimo procedūrų ypatumų. Yra pasiūlytos kelios euristinės nesudėtingos populiacijų formavimo procedūros, tai: atsitiktinis generavimas, tolygusis pagerinimas, netolygusis

(diferencijuotasis) pagerinimas. Šios procedūros išbandytos, sprendžiant vieną iš sudėtingų kombinatorinio optimizavimo uždavinių — kvadratinio paskirstymo uždavinį. Gautieji eksperimentų rezultatai liudija, jog GA rezultatų kokybė gali būti padidinta, panaudojant (vietoje atsitiktinio generavimo) tolygiojo ir/arba netolygiojo (diferencijuotojo) populiacijos pagerinimo procedūras. Pasiūlytos strategijos yra gana universalaus pobūdžio ir galėtų būti pritaikytos kituose populiacijų panaudojimu besiremiančiuose euristiniuose metoduose (pvz., dalelių ar bičių spiečių optimizavimo metoduose).