Computing and Informatics, Vol. 31, 2012, 827–845

A GENETIC ALGORITHM APPROACH FOR SOLVING THE MACHINE-JOB ASSIGNMENT WITH CONTROLLABLE PROCESSING TIMES

Aleksandar Savić

Faculty of Mathematics, University of Belgrade
Studentski trg 16/IV
11 000 Belgrade, Serbia
e-mail: aleks3rd@gmail.com

Communicated by José C. Cunha

Abstract. This paper considers a genetic algorithm (GA) for a machine-job assignment with controllable processing times (MJACPT). Integer representation with standard genetic operators is used. In an objective function, a job assignment is obtained from genetic code and for this, fixed assignment processing times are calculated by solving a constrained nonlinear convex optimization problem. Additionally, the job assignment of each individual is improved by local search. Computational results are presented for the instances from literature and modified large-scale instances for the generalized assignment problem (GAP). It can be seen that the proposed GA approach reaches almost all optimal solutions, which are known in advance, except in one case. For large-scale instances, GA obtained reasonably good solutions in relatively short computational time.

Keywords: Evolutionary approach, genetic algorithms, constrained convex optimization, computer numerically controlled (CNC) machines, flexible manufacturing systems

1 INTRODUCTION

From the beginning of the industrial era, and especially from the time when Henry Ford proposed the scheduling of works and machines, the problem of sequential and parallel scheduling of jobs and machines arose in both theory and practice. The assignment of jobs, machines, and workers started in the 60's with the first

assignment problems and their solving with modified simplex algorithms. As years passed, more and more complicated and time and resource consuming problems began to surface. With the era of computers and numerically controlled (CNC) machines, these problems became more numerous and sophisticated.

Numerous papers about job assignment have been considered in literature so far and they were mostly derived from practice. Therefore, a detailed presentation of the work concerning job assignment is out of this paper's scope, but some of the new and successful ones are [32, 33].

Because of high cost of flexible manufacturing systems and their maintenance, there arose a need for careful planning and scheduling of jobs [9]. In the early 80's, the consideration of processing times was first introduced. Since that period, there has been a growing interest in these problems. Flexible manufacturing systems are discussed in [28].

This paper explores the machine-job assignment problem with controllable processing times represented by nonlinear functions ([1]). This problem has surfaced in flexible manufacturing systems, where these processing times are numerically controlled. These systems are constituted of groups (even large ones) of non-identical machines, and they all have different working levels and different levels and modes of power and control. Processing times on computer numerically controlled (CNC) machines can be compressed by increasing the cutting speed and the feed rate at a convex increasing cost for compression. Thus, when processing time becomes a decision variable, one is faced with a trade-off between increasing yield and cost of machining.

The problem of machine-job assignment with controllable processing times can be modelled as a nonlinear mixed 0-1 maximization problem. If processing times are disallowed, then MJACPT is reduced to a well-known and NP-hard generalized assignment problem (GAP). Variables that represent controllable times are included in the nonlinear part of the objective function and this part only makes the problem considerably harder to solve.

This paper proposes to apply genetic algorithms on the linear part of the problem, which is very similar to GAP, the only difference being that in GAP, every job must be assigned to some machine, while here some jobs can remain unassigned. This was motivated by recent developments in genetic algorithms, especially [3, 6, 27]. Although the linear part of MJACPT problem has similarities with GAP, novel extensions of GA for solving GAP are not applicable to MJACPT.

For example, in [6] 3 new extensions of GA for solving GAP are reported: initialization heuristic, selection and replacement of infeasible solutions, and heuristic mutation operator. Constraint-ratio initialization heuristic is based on the fact that in GAP, the job must be assigned without exceeding resource capacities, which is not the case in MJACPT. Selection and replacement of infeasible solutions is based on penalty functions of infeasible solutions. In the GA approach to MJACPT, presented in Section 4, such solutions are feasible and there is no need for any penalty functions. Unfortunately, heuristic mutation operator, which is quite successful for GAP (see [6]), is not directly applicable to MJACPT.

Therefore, the proposed genetic algorithm for solving the linear part of MJACPT is based, in general, on well documented GA approaches presented in [21, 29, 30, 31]. Modification of GA for solving MJACPT is explained in detail in Section 4. After solving the linear part of MJACPT, there remained to solve the nonlinear part, which was now reduced to solving a classical convex nonlinear optimization problem.

Solving a classical convex nonlinear optimization problem with the quadratic objective function is a known problem and here it is solved by finding appropriate Lagrangean multipliers. This can be found in many works and the reader can refer to [24].

The content of this paper is organized in the following sections. In Section 2 the mathematical model of MJACPT from [1] is presented. A description of solving the nonlinear part of the problem for fixed 0-1 variables is available in Section 3. The proposed genetic algorithm (GA) method is discussed in Section 4. Finally, in Section 5 computational and experimental results are presented.

2 MATHEMATICAL FORMULATION

The problem of machine-job assignment with controllable processing time consists of choosing some of the jobs from n jobs and m machines and assigning them to machines with a view to deriving the maximum profit from this assignment. The notation will be as follows: let c_i be available time for the work on the machine $i = 1, \ldots, m$, let p_{ij} be processing time on the machine i to which job j is assigned, and let h_{ij} be profit from this assignment.

Processing time on CNC machines can be reduced by a specific setting of the machine parameters. Consequently, this also leads to shortening the useful life of these machines, which results in increased cost of exploitation. The function for modelling this increased machine cost is given as

$$f(y) = ky^{a/b}, \tag{1}$$

where $y \geq 0$ represents time compression. Obviously, f is a convex and increasing function of time compression. The value k is constant for a given machine, but varies from machine to machine, therefore technical characteristics of machines are contained in the matrix of coefficients k. Let x_{ij} be an assignment variable defined by

$$x_{ij} = \begin{cases} 1, & \text{if job } j \text{ is assigned to machine } i \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

and let y_{ij} be a variable that depicts compression time for the same job-machine pair. Then the machine-job assignment problem with controllable times (MJACPT) can be formulated as a nonlinear mixed 0-1 program:

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} \left(h_{ij} x_{ij} - f_{ij}(y_{ij})\right) \tag{3}$$

subject to

$$\sum_{j=1}^{n}(p_{ij}x_{ij} - y_{ij}) \leq c_i, \quad i = 1, \ldots, m, \tag{4}$$

$$y_{ij} \leq x_{ij}u_{ij}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n, \tag{5}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \quad j = 1, \ldots, n, \tag{6}$$

$$x_{ij} \in \{0, 1\}, \quad y_{ij} \geq 0, i = 1, \ldots, m, \quad j = 1, \ldots, n. \tag{7}$$

Constraint (4) ensures that jobs assigned to machine i do not exceed its available time c_i; constraint (5) ensures that compression of job j on machine i is not larger than the allowed maximum u_{ij} ($u_{ij} < p_{ij}$); constraint (6) ensures that one job is assigned to at most one machine.

Note that, for fixed values of x_{ij}, part of objective function $-\sum_{i=1}^{m}\sum_{j=1}^{n} k_{ij}y_{ij}^2$, which deals with time compression, given in (3), can be modelled as a classical convex nonlinear optimization problem. Let us call this problem FixedMJACPT.

This means that we must find, for a fixed set of variables x_{ij}, the minimum of function $\sum_{i=1}^{m}\sum_{j=1}^{n} k_{ij}y_{ij}^2$, which is a known problem in mathematical programming and, in this paper, it has polynomial time complexity. Genetic algorithm is used for finding an adequate set of variables x_{ij}.

Example 1. A little example is given here. Let there be $m = 3$ machines and $n = 5$ jobs. Let coefficients h_{ij} be given in matrix H, coefficients p_{ij} in matrix P, coefficients u_{ij} in matrix U, and coefficients c_i in vector c

$$H = (h_{ij}) = \begin{pmatrix} 17 & 21 & 22 & 18 & 24 \\ 23 & 16 & 21 & 16 & 17 \\ 16 & 20 & 16 & 25 & 24 \end{pmatrix}$$

$$P = (p_{ij}) = \begin{pmatrix} 8 & 15 & 14 & 23 & 8 \\ 15 & 7 & 23 & 22 & 11 \\ 21 & 20 & 6 & 22 & 24 \end{pmatrix}$$

$$U = (u_{ij}) = \begin{pmatrix} 5.5 & 12 & 11 & 7.5 & 5.6 \\ 11 & 3.5 & 19 & 17 & 8.6 \\ 14 & 15 & 2.3 & 18 & 11 \end{pmatrix}$$

$$c = (c_i) = \begin{pmatrix} 36 \\ 34 \\ 38 \end{pmatrix}.$$

Let nonlinear functions be

$$f_{ij}(y_{ij}) = k_{ij} y_{ij}^2$$

where all coefficients k_{ij} are equal to 0.1. Then the optimal objective function equals ≈ 114.97 and the assignment of jobs is as follows. On the first machine jobs 2, 3, and 5 are executed; on the second machine job 1 is executed, and job 4 on the third machine. Non-zero job compressions are $y_{12} = y_{13} = y_{15} = \frac{1}{3}$.

3 SOLUTION OF FIXEDMJACPT BY LAGRANGEAN MULTIPLIERS

In this section, let us demonstrate how the nonlinear part of problem can be solved. For a fixed set of variables x_{ij}, FixedMJACPT is reduced to the maximization of a concave function with linear constraints. The solution of FixedMJACPT is later used in GA implementation for solving MJACPT. Let the denotation of variables and coefficients be the same as in Section 2. Then, the problem is reduced to

$$\max H - \sum_{i=1}^{m} \sum_{j=1}^{n} k_{ij} y_{ij}^2 \tag{8}$$

with the following constraints

$$A_i - \sum_{j=1}^{n} y_{ij} \leq c_i, \quad i = 1, \ldots, m \tag{9}$$

$$0 \leq y_{ij} \leq x_{ij} u_{ij}, \quad i = 1, \ldots, m, j = 1, \ldots, n \tag{10}$$

where the last set of constraints is directly dependent on fixation of variables x_{ij}. Coefficients H and A_i are calculated values of the following equalities

$$\begin{array}{l} H = \sum_{i=1}^{m} \sum_{j=1}^{n} h_{ij} x_{ij} \\ A_i = \sum_{j=1}^{n} p_{ij} x_{ij}, \quad i = 1, \ldots, m \end{array} \tag{11}$$

for fixed values of variables x_{ij}. Now, when the theory of Lagrangean multipliers is applied to this problem, the Lagrangean function is as follows

$$\Phi = -\sum_{i=1}^{m} \sum_{j=1}^{n} k_{ij} y_{ij}^2 + \sum_{i=1}^{m} \lambda_i (A_i - c_i - \sum_{j=1}^{n} y_{ij}) \tag{12}$$

and the solution of the optimization problem is between the solutions of the next system of equations:

$$\begin{array}{l} \Phi'_{y_{ij}} = -2 k_{ij} y_{ij} + \lambda_i Ind_i = 0, \\ \lambda_i (A_i - c_i - \sum_{j=1}^{n} y_{ij}) = 0, \quad i = 1, \ldots, m \end{array}, \tag{13}$$

where $Ind_i = \begin{cases} 1, & A_i - c_i > 0 \\ 0, & \text{otherwise} \end{cases}$. Equations (13) are linear and can be solved independently for every y_{ij}. It is easy to see, with consideration of the upper bounds, that the solution is

$$y_{ij} = \begin{cases} \min\left(u_{ij}, \frac{\lambda_i}{2k_{ij}}\right), & Ind_i \neq 0 \\ 0, & Ind_i = 0 \end{cases}, \tag{14}$$

where Lagrangean coefficients are calculated from the following formulas

$$\lambda_i = \begin{cases} \frac{2(A_i - c_i)}{\sum_{j=1}^{n} \frac{1}{k_{ij}}}, & Ind_i \neq 0 \\ 0, & Ind_i = 0 \end{cases}. \tag{15}$$

For more information on solving optimization problems through Lagrangean multipliers see [24].

4 PROPOSED GA METHOD

Genetic algorithms are stochastic methods for searching and finding best solutions to problems. They are motivated by processes in the natural world and to a great extent try to emulate them. Like nature, GA works with individuals that constitute a population. Each individual represents some solution to a problem. As in nature we have individuals that are better suited to survival, here we have individuals that are better at accomplishing the optimum solution and are favored for passing on to the next generation. This passing of good qualities is accomplished with the genetic operators of crossover and mutation. The decision as to which individual has better qualities to be passed on to the next generation is attained by evaluating the fitness function. This process of betterment of individuals in a population is iteratively continued until optimum or some other stopping criterion is achieved. A detailed description of GAs is out of this paper's scope and it can be found in [8, 26, 30]. Extensive computational experience of various optimization problems shows that GA often produces high quality solutions in a reasonable time. Some of the recent applications are:

- hub location [8, 16, 17, 18, 30, 31];
- facility location [5, 12, 13, 25];
- generalized assignment [3, 6, 27];
- metric dimension of graphs [19, 20];
- biconnectivity augmentation of graphs [22, 23];
- maximally balanced connected partition of graphs [4];
- network design [14];
- discrete ordered median [29];

- index selection [15];
- routing and carrier selection [21].
- binary sequencing [10].

Furthermore, GA is quite robust in respect of parameter choice within reasonable bounds for a lot of different problems [4, 5, 7, 18, 19, 20, 29, 30, 31].

Detailed GA algorithm is given in the following scheme.

```
Input_Data();
Population_Init();
while not Finish() do
    for index:= (N_elite + 1) to N_pop do
      if (Exists_in_Cache(index)) then
          objvalue_index:= Get_Value_From_Cache(index);
      else
          objvalue_ind:= Objective_Function(index);
          Put_Into_the_Cache_Memory(index, objvalue_index);
          if (Full_Cache_Memory)) then
            Remove_LRU_Block_From_Cache_Memory();
          endif
      endif
    endfor
    Fitness_Function();
    Selection();
    Crossover();
    Mutation();
endwhile
Output_Data();
```

Fig. 1. The basic scheme of this GA implementation

In this paper, encoding of individuals is an integer. Every gene is partitioned into two parts. The first part consists of one bit, which is designated to represent the assignment of a specified job to any of the machines. This means that a bit is equal to 1 if $\sum_{j=1}^{n} x_{ij} = 1$, corresponding to a situation where only one of x_{ij} is 1 and all others are 0, and is equal to 0 if $\sum_{j=1}^{n} x_{ij} = 0$, which corresponds to a situation where all x_{ij} are 0. In the latter case, the second part of the genetic code is ignored. In the former case, the second part of the genetic code represents the ordinal numeral of the machine on which the job is executed.

When, for example, the r^{th} gene is considered, this means the evaluation of the sum

$$\sum_{j=1}^{r} (p_{ij} - u_{ij}) x_{ij}, r < n. \tag{16}$$

If this sum is greater than c_i (capacity of the i^{th} machine), constraint (4) is obviously unsatisfied, so that this gene is passed over and instead, one of the successive ones is taken into account, where the sum (16) is smaller than c_i. If there are no such genes, the job is transferred to the previous machines. If this condition is true for every machine, then that individual was incorrect. For finding solutions of nonlinear variables $y_{ij}, i = 1, \ldots, m, j = 1, \ldots, n$ the method explained in the previous section is used.

The objective function is evaluated following consecutive procedures.

1. In the first procedure, the values of variables x_{ij} are obtained from the genetic code. The running time complexity of this procedure is $O(n)$.
2. In the second procedure, the values of variables y_{ij} are calculated by solving FixedMJACPT by using the Lagrangean method from Section 3. The running time complexity of this procedure is $O(n \log n)$.
3. In the third procedure, LocalSearch is applied to variables obtained in procedure 1, in which some of x_{ij} are changed with some other variable from the same set, with the intention of improving the value of the objective function. LocalSearch is repeated until there are no more improvements. The running time complexity of any single improvement is $O(n^2 \log n)$.
4. Finally, the running time complexity for calculating the objective value is $O(n)$.

As can be seen, procedure 3 is the most time consuming and its execution dominates the whole solving process.

The very important parts of proposed GA are: FitnessFunction, Selection, Crossover and Mutation. In the following paragraphs, their utilization and importance to the whole GA will be explained.

The function that decides the best suitability f_{ind} of an individual to pass on to the next generation is the fitness function. Values of this function are computed by scaling objective values obj_{ind} of all individuals into the interval $[0, 1]$, so that the best suited individual ind_{max} gets value 1 and the worst ind_{min} gets 0. Explicitly, $f_{ind} = \frac{obj_{ind_{min}} - obj_{ind}}{obj_{ind_{min}} - obj_{ind_{max}}}$. Now, individuals are arranged in a non-increasing order by their best fitness: $f_1 \geq f_2 \geq \ldots \geq f_{N_{pop}}$, where N_{pop} is the number of individuals in a population.

When GAs are used, there is a possibility of domination of elite individuals. Elite individuals directly pass into the next generation without their substitution by offspring in order to preserve good solutions through generations of GA. So, elite individuals can pass into the next generation in two ways: first, because they are elite, and second, because of the selection operator. If GAs are to give quality results, it is necessary to have enough elite individuals for exploitation of their good qualities and to have enough non-elite individuals so that the genetic pool from which individuals can be chosen does not become too small. A small genetic pool could not guarantee a population's growth in the right direction. To prevent too large a number of elite individuals (N_{elite}) and their domination, the fitness of these

individuals is decreased as follows:

$$f_{ind} = \begin{cases} f_{ind} - \overline{f}, & f_{ind} > \overline{f} \\ 0, & f_{ind} \leq \overline{f} \end{cases}; \quad 1 \leq ind \leq N_{elite}; \quad \overline{f} = \frac{1}{N_{pop}} \sum_{ind=1}^{N_{pop}} f_{ind}. \tag{17}$$

All elite individuals, their number being N_{elite}, are automatically passed on to the next generation. All non-elite individuals, their number being $N_{nnel} = N_{pop} - N_{elite}$, are subject to genetic operators. This reduces computational time, because the objective function of elite individuals is the same in the next generation and needs to be calculated only once, in the first generation.

Individuals with the same genetic code in a population must be avoided, so their fitness is set to 0 in all occurrences, except the first one. Also, the number of individuals with the same objective function, but different genetic code must be limited by some constant N_{rv}. This is important, because too many of them in a population can inhibit the occurrence of individuals with good genetic material and point the algorithm to convergence toward local and not global optimum. To avoid this problem, the fitness of all individuals with the same value of objective function but different genetic material will be set to 0 except the first N_{rv} of them. For a detailed explanation of N_{rv} constant and its use, reduction of elite individuals' fitness and elitism ratio, see [18, 19].

Selection operators are applied to all non-elite individuals and they choose which of these individuals will have offspring in the next generation. This is done through tournaments. From a whole population, a predetermined number of individuals is chosen to participate in the tournament. The number of participants is called tournament size. The individuals are chosen randomly. The winner of the tournament is the individual with the highest value of objective function. The number of tournaments is equal to the number of non-elite individuals N_{nnel}, so that exactly N_{nnel} parents can be chosen for crossover. The same individual from a current generation can participate in more than one tournament. In a standard tournament selection, tournament size is an integer, which can hinder the algorithm efficiency.

Because of this, an improved tournament selection operator, fine-grained tournament selection – FGTS [5] is implemented here for selection purposes. Here, tournament size is a real parameter F_{tour}, which represents a preferable average tournament size. In this procedure, there are two types of tournaments. One is held k_1 times, with tournament size $\lfloor F_{tour} \rfloor$, and the other type is held k_2 times, with tournament size $\lceil F_{tour} \rceil$. From here $F_{tour} \approx \frac{k_1 \cdot \lfloor F_{tour} \rfloor + k_2 \cdot \lceil F_{tour} \rceil}{N_{nnel}}$.

For satisfactory results of GA, it is necessary to have a good ratio between the number of elite individuals and non-elite individuals. For example, for $N_{pop} = 150$ the adequate proportion is $N_{elite} = 100$ and $N_{nnel} = 50$. Corresponding k_1 and k_2 parameters in deciding tournament size are 30 and 20, respectively. Typically, for $N_{pop} = 150$ the adequate maximum of individuals with the same fitness, which means with the same value of objective function, is $N_{rv} = 40$.

As seen in [5, 7, 8, 29], quite numerous numerical experiments were performed for different optimization problems, FGTS performs best with the value of F_{tour} set on 5.4. Leaning on experience presented in the cited works the same value is used in this paper. For detailed information about FGTS see [7].

In a crossover operator, chosen non-elite individuals are now randomly paired in $\lfloor N_{nnel}/2 \rfloor$ pairs for an exchange of genes with the intention of producing offspring with potentially better suitability. Application of a crossover operator on a chosen pair of parents produces two offsprings. In this paper, standard one point crossover operator is used. This operator exchanges whole genes between the genetic codes of parents to produce an offspring. The probability of realization of the crossover operator is 85 %. This means that approximately 85 % pairs of individuals will exchange genes.

In genetic algorithm, a simple mutation operator is used. This operator changes a randomly selected gene in the genetic code of an individual at a certain mutation rate. For improvement of GA, a modification which deals with so-called frozen genes is also included. Sometimes it happens that all individuals in a population have the same gene in a certain position. This kind of gene is called a frozen gene. The problem with frozen genes is that they reduce search space and increase the possibility of premature convergence. For example, if there are q frozen genes in a population, then search space will be 2^q times smaller. The selection and crossover operators cannot change frozen genes, because all individuals in the population have them in the same position. The basic mutation rate is too small to ensure that frozen genes are changed within a reasonable time interval and also it is too small to restore regions of search space that were bypassed because of the frozen genes. Furthermore, an increase in mutation rate can reduce genetic algorithm to a pure random search. For a better understanding of mutation with frozen genes see [16, 17].

The above-mentioned improvement of GA is that the mutation rate is increased for frozen genes only. Which genes are frozen is determined for each generation. Then, the mutation rate for these genes is increased. In the proposed GA, the increase is 2.5 greater than the mutation rate of the non-frozen genes.

The initialization of GA is random, which gives the population in the first generation the most heterogeneous and diversified genetic pool.

The performance of the proposed GA is improved by using the caching technique. The main idea behind this technique is to avoid the evaluation of objective functions for individuals with the same genetic code. The values of individuals for which the objective functions were already computed are stored by the least recently used (LRU) caching technique into the hash-queue data structure. Because of this, whenever an individual with the same genetic code is generated, the value of its objective function is not computed, but is found in cache memory, which can result in significant time saving. The number of calculated values of objective functions in this implementation is limited to 5 000. If cache memory is full, then we remove the least recently used cache memory block. Detailed information about caching GA can be found in [11].

5 EXPERIMENTAL RESULTS

All computations were executed on a Dual Double Core 2.0 GHz MacPro computer with 3 Gb RAM. The genetic algorithm was coded in C programming language. For the first part of the experiments, instances described in [1], which can be obtained from `http://www.ieor.berkeley.edu/~atamturk/data/conic.sch/` were used. These instances include different numbers of jobs ($n = 50, 100, 150, 200$) and different numbers of machines ($m = 1, 2, 3$). For each pair of machine-job (m, n), there is a set of five instances with various h_{ij}, k_{ij}, p_{ij} and u_{ij}.

The finishing criterion of GA is the maximum number of generations $N_{gen} = 200$. The algorithm also stops if the best individual or best objective value remains unchanged through $N_{rep} = 100$ successive generations. Since the results of GA are nondeterministic, the GA was run 20 times on each of the instances.

Tables 1 and 2 summarize the GA results in these instances. In the first column the names of instances are given. An instance name carries information about the number of jobs n, the number of machines m, and the number of generated cases with same n and m. For example, instance CMJ2_N100_M3_ins3 is an instance which has $n = 100$ jobs on $m = 3$ machines and it is the fourth case generated for this n and m).

The second column provides the optimum solutions obtained by using ILOG CPLEX Version 10.1, but these solutions could have numerical instability, as will be explained later and in detail. The best GA values GA_{best} are given in the next column.

Average times needed to detect the best GA values are given in the t column, while t_{tot} represents the total running times (in seconds), needed for finishing GA. On the average, GA finished after *gen* generations. The solution quality in all 20 executions is evaluated as a percentage gap named *agap*, with respect to the optimal solution Opt_{sol}, with standard deviation σ of the average gap. A percentage gap *agap* is defined as $agap = \frac{1}{20}\sum_{i=1}^{20} gap_i$, where $gap_i = 100 * \frac{GA_i - GA_{best}}{GA_{best}}$ and GA_i represents the GA solution obtained in the i-th run, while σ is the standard deviation of gap_i, $i = 1, 2, \ldots, 20$, obtained by the formula $\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(gap_i - agap)^2}$. The last two columns are related to the caching: *eval* represents the average number of evaluations, while *cache* displays savings (in percent), achieved by using caching technique.

As can be seen in Tables 1 and 2, running time for all instances is reasonably small. The average execution on the biggest instance is a little more than 2 minutes.

It can be seen from Tables 1 and 2 that the results of CPLEX and GA algorithms are not identical, but vary slightly from instance to instance. Sometimes, GA has a greater value, sometimes the CPLEX result is the greater. Because CPLEX has implemented a lot of numerical algorithms for solving a wide range of problems, there is a small numerical instability, which is greater than the numerical instability executed by GA. This is due to universality of CPLEX vs. GA algorithm code, which was written strictly for this problem.

Instance name	$CPLEX_{sol}$	GA_{best}	t (sec)	t_{tot} (sec)	*gen*	*agap* (%)	σ (%)	*eval*	*cache* (%)
CMJ2_N50_M1_ins0	49.61845643	49.618457	0.060	2.0766	101	0.000	0.000	4 341	16.5
CMJ2_N50_M1_ins1	50.98362713	50.983628	0.060	1.9721	101	0.000	0.000	4 321	16.9
CMJ2_N50_M1_ins2	49.40102156	49.401021	0.055	1.8007	101	0.000	0.000	4 331	16.7
CMJ2_N50_M1_ins3	57.27832993	57.278331	0.047	1.4339	101	0.000	0.000	4 322	16.9
CMJ2_N50_M1_ins4	48.66271533	48.662717	0.056	1.4145	101	0.000	0.000	4 398	15.4
CMJ2_N50_M2_ins0	96.00463722	96.004638	0.162	2.3909	104	0.038	0.013	4 455	17.2
CMJ2_N50_M2_ins1	102.0868546	102.086855	0.088	2.5776	101	0.000	0.000	4 256	18.1
CMJ2_N50_M2_ins2	93.50524103	93.505242	0.820	3.742	122	0.032	0.039	5 227	16.7
CMJ2_N50_M2_ins3	99.57028503	99.570309	0.429	3.144	112	0.006	0.015	4 804	16.7
CMJ2_N50_M2_ins4	104.4961934	104.496326	0.081	2.452	101	0.000	0.000	4 323	16.8
CMJ2_N50_M3_ins0	150.5204773	150.520478	0.265	3.773	105	0.000	0.000	4 952	8.5
CMJ2_N50_M3_ins1	146.0709755	146.070978	0.447	3.957	110	0.000	0.000	5 150	8.9
CMJ2_N50_M3_ins2	134.3199589	134.320059	1.749	5.633	143	0.047	0.044	6 733	8.2
CMJ2_N50_M3_ins3	143.6944143	143.694416	0.263	3.707	105	0.000	0.000	4 926	9.0
CMJ2_N50_M3_ins4	142.8815256	142.881526	1.227	4.670	132	0.040	0.051	6 240	8.1
CMJ2_N100_M1_ins0	102.5891335	102.589135	0.443	14.059	101	0.000	0.000	4 509	13.3
CMJ2_N100_M1_ins1	103.9195384	103.919539	0.379	12.936	101	0.000	0.000	4 469	14.0
CMJ2_N100_M1_ins2	96.69656271	96.696563	0.547	15.792	101	0.000	0.000	4 463	14.2
CMJ2_N100_M1_ins3	105.1304131	105.130415	0.416	16.991	101	0.000	0.000	4 539	12.7
CMJ2_N100_M1_ins4	106.4550908	106.455094	0.374	11.690	101	0.000	0.000	4 579	11.9
CMJ2_N100_M2_ins0	201.2384673	201.238468	1.303	13.737	107	0.003	0.008	4 775	13.3
CMJ2_N100_M2_ins1	205.7837429	205.783802	0.651	19.941	101	0.000	0.000	4 436	14.7
CMJ2_N100_M2_ins2	196.1458415	196.145843	1.027	17.288	103	0.000	0.000	4 621	13.3
CMJ2_N100_M2_ins3	197.1896476	197.189654	1.182	20.752	103	0.000	0.000	4 626	13.1
CMJ2_N100_M2_ins4	201.8046931	201.804752	0.568	15.690	101	0.000	0.000	4 510	13.2
CMJ2_N100_M3_ins0	296.5320918	296.532094	7.553	28.480	132	0.013	0.050	6 365	5.9
CMJ2_N100_M3_ins1	291.6845401	291.685019	9.047	29.886	139	0.004	0.016	6 697	5.9
CMJ2_N100_M3_ins2	294.2433068	294.243765	14.333	36.253	156	0.025	0.025	7 505	5.9
CMJ2_N100_M3_ins3	285.7255766	285.729674	13.638	34.624	153	0.004	0.005	7 388	5.6
CMJ2_N100_M3_ins4	287.5521985	287.553312	15.842	35.680	161	0.043	0.063	7 737	5.7

Table 1. GA results on smaller instances

Because of these discrepancies, some additional tests and checks were run. With the changing of parameters in CPLEX, some results were improved, but the error was in a similar range with previous parameters. In some cases, solutions were even worse. Because of this, solutions with the default set of CPLEX parameters were presented in all cases. In most cases, values for x_{ij} in CPLEX and GA were identical. For those instances, y_{ij} were obtained with the method for solving FixedMJACPT, which is exact for any predetermined decimal, and for these sets of x_{ij} and y_{ij} values of objective function were calculated. The result was identical with GA solutions, so the conclusion is that discrepancies were due to a rounding off error in CPLEX.

In cases where solution sets of x_{ij} in CPLEX and GA algorithms were not identical, it was concluded that the variations were due to a rounding off error which disallowed some x_{ij} in the optimal CPLEX solution. Those x_{ij} could be put in equations of type (4) and boundary c_i might be reached only if rounding was allowed with a greater number of decimals. This is particularly the case in instance CMJ2_N200_M3_ins2, where GA result is greater than the maximum obtained by CPLEX for the amount of approximately 0.04. This instance was especially checked and the GA result was exact up to the 13^{th} decimal, whereas the CPLEX result deviates already on the 2^{nd} decimal.

Instance name	$CPLEX_{sol}$	GA_{best}	t (sec)	t_{tot} (sec)	*gen*	*agap* (%)	σ (%)	*eval*	*cache* (%)
CMJ2_N150_M1_ins0	158.498732	158.498733	1.133	37.556	101	0.000	0.000	4 630	10.9
CMJ2_N150_M1_ins1	156.6469826	156.646985	1.208	29.210	101	0.000	0.000	4 640	10.7
CMJ2_N150_M1_ins2	145.304105	145.304127	1.471	48.870	101	0.000	0.000	4 590	11.7
CMJ2_N150_M1_ins3	155.0724027	155.072406	1.116	39.004	101	0.000	0.000	4 678	10.0
CMJ2_N150_M1_ins4	159.1373384	159.138165	1.398	51.548	101	0.000	0.000	4 605	11.4
CMJ2_N150_M2_ins0	308.380354	308.380356	6.716	48.632	112	0.000	0.000	5 113	11.5
CMJ2_N150_M2_ins1	306.4666165	306.466620	1.923	59.845	101	0.000	0.000	4 528	12.9
CMJ2_N150_M2_ins2	306.4769149	306.477038	2.037	64.436	101	0.000	0.000	4 515	13.1
CMJ2_N150_M2_ins3	295.108654	295.108662	4.533	56.072	106	0.000	0.000	4 815	11.6
CMJ2_N150_M2_ins4	306.7194959	306.719503	4.199	54.772	105	0.000	0.000	4 770	11.9
CMJ2_N150_M3_ins0	449.6789004	449.682666	10.239	66.315	116	0.000	0.000	5 684	4.6
CMJ2_N150_M3_ins1	439.922495	439.922499	45.548	93.546	168	0.015	0.019	8 170	4.7
CMJ2_N150_M3_ins2	449.126137	449.126141	62.689	112.319	175	0.067	0.077	8 481	4.9
CMJ2_N150_M3_ins3	424.1957387	424.196692	27.467	77.111	150	0.011	0.023	7 299	4.9
CMJ2_N150_M3_ins4	433.6412309	433.653701	56.574	110.268	172	0.039	0.032	8 360	4.7
CMJ2_N200_M1_ins0	208.5897235	208.589725	2.106	63.542	101	0.000	0.000	4 734	8.9
CMJ2_N200_M1_ins1	210.5568331	210.556835	2.817	78.042	101	0.000	0.000	4 732	9.0
CMJ2_N200_M1_ins2	200.2334487	200.233450	3.680	84.405	101	0.000	0.000	4 725	9.1
CMJ2_N200_M1_ins3	204.1614891	204.161493	3.511	93.942	101	0.000	0.000	4 708	9.4
CMJ2_N200_M2_ins0	413.876559	413.876565	4.973	113.092	102	0.000	0.000	4 680	11.2
CMJ2_N200_M2_ins1	406.5458725	406.545878	18.031	141.491	111	0.000	0.000	5 140	10.1
CMJ2_N200_M2_ins2	419.1611158	419.161786	19.694	128.483	115	0.002	0.006	5 323	10.5
CMJ2_N200_M2_ins3	394.4577078	394.457714	6.149	116.582	103	0.000	0.000	4 658	12.2
CMJ2_N200_M2_ins4	404.4349761	404.434981	14.215	150.545	109	0.000	0.000	5 044	10.3
CMJ2_N200_M3_ins0	590.2930825	590.293090	77.257	183.368	158	0.017	0.016	7 776	3.8
CMJ2_N200_M3_ins1	594.7989282	594.798932	127.190	196.726	189	0.070	0.068	9 225	4.1
CMJ2_N200_M3_ins2	591.7248894	591.766738	113.350	223.170	177	0.017	0.016	8 665	4.2
CMJ2_N200_M3_ins3	574.5171615	574.433285	112.154	193.934	181	0.029	0.030	8 862	4.1
CMJ2_N200_M3_ins4	588.7981153	588.798120	139.055	255.742	184	0.025	0.023	9 010	3.9

Table 2. GA results on larger instances

In only one case, the GA method did not reach an optimal solution and was smaller than the CPLEX value for the instance CMJ2_N200_M3_ins3. The CPLEX result was 574.517161 and the GA result, 574.433285. For an optimal x_{ij} determined by CPLEX in the instance mentioned above, y_{ij} were calculated by the method for solving FixedMJACPT, and after that, the value of objective function was obtained. This optimal value was 574.517183, and it was free of any rounding off error. This means that CPLEX reached the optimum, but was not exact due to the rounding off error.

All other solutions of the GA method were optimal, their error was on the 13^{th} decimal, so all optimal solutions with high precision are given in Table 3. The GA method could not verify optimality of solutions alone, but in this case, a study of solutions of CPLEX and GA points to the conclusion that all other GA solutions were optimal.

The second part of the experiment was performed to emphasize the use of GA and comparing differences with CPLEX required testing on much greater instances. Since there do not exist large-scale MJACPT instances, testing was also performed on modified large-scale ORLIB GAP instances, with parameter m up to 80 and parameter n up to 400, presented in [6]. Missing data (matrices k_ij and u_ij) was randomly generated as in [1]. Because GAP is a minimization problem and

Instance name	*Opt.sol*	*Instance name*	*Opt.sol*
CMJ2_N50_M1_ins0	49.6184571840096	CMJ2_N150_M1_ins0	158.4987326103862
CMJ2_N50_M1_ins1	50.9836276836393	CMJ2_N150_M1_ins1	156.6469845071603
CMJ2_N50_M1_ins2	49.4010214895848	CMJ2_N150_M1_ins2	145.3041273209200
CMJ2_N50_M1_ins3	57.2783306637174	CMJ2_N150_M1_ins3	155.0724060324565
CMJ2_N50_M1_ins4	48.6627167903425	CMJ2_N150_M1_ins4	159.1381650812547
CMJ2_N50_M2_ins0	96.0046381268228	CMJ2_N150_M2_ins0	308.3803557208960
CMJ2_N50_M2_ins1	102.0868553225832	CMJ2_N150_M2_ins1	306.4666195431429
CMJ2_N50_M2_ins2	93.5052424120723	CMJ2_N150_M2_ins2	306.4770378197365
CMJ2_N50_M2_ins3	99.5703094364975	CMJ2_N150_M2_ins3	295.1086624543456
CMJ2_N50_M2_ins4	104.4963258036986	CMJ2_N150_M2_ins4	306.7195031962992
CMJ2_N50_M3_ins0	150.5204777847314	CMJ2_N150_M3_ins0	449.6826660346746
CMJ2_N50_M3_ins1	146.0709784188812	CMJ2_N150_M3_ins1	439.9224990034459
CMJ2_N50_M3_ins2	134.3200594945234	CMJ2_N150_M3_ins2	449.1261410038149
CMJ2_N50_M3_ins3	143.6944161293770	CMJ2_N150_M3_ins3	424.1966916942326
CMJ2_N50_M3_ins4	142.8815259890024	CMJ2_N150_M3_ins4	433.6537005607537
CMJ2_N100_M1_ins0	102.5891345863531	CMJ2_N200_M1_ins0	208.5897252356628
CMJ2_N100_M1_ins1	103.9195389338855	CMJ2_N200_M1_ins1	210.5568348855265
CMJ2_N100_M1_ins2	96.6965633770596	CMJ2_N200_M1_ins2	200.2334498731282
CMJ2_N100_M1_ins3	105.1304152605631	CMJ2_N200_M1_ins3	204.1614929078376
CMJ2_N100_M1_ins4	106.4550936083245	CMJ2_N200_M2_ins0	413.8765646932486
CMJ2_N100_M2_ins0	201.2384675063532	CMJ2_N200_M2_ins1	406.5458779954473
CMJ2_N100_M2_ins1	205.7838017264405	CMJ2_N200_M2_ins2	419.1617857029970
CMJ2_N100_M2_ins2	196.1458428409325	CMJ2_N200_M2_ins3	394.4577140505253
CMJ2_N100_M2_ins3	197.1896537591701	CMJ2_N200_M2_ins4	404.4349810136758
CMJ2_N100_M2_ins4	201.8047515689439	CMJ2_N200_M3_ins0	590.2930901105431
CMJ2_N100_M3_ins0	296.5320941467872	CMJ2_N200_M3_ins1	594.7989320213267
CMJ2_N100_M3_ins1	291.6850188552467	CMJ2_N200_M3_ins2	591.7667378549382
CMJ2_N100_M3_ins2	294.2437654270240	CMJ2_N200_M3_ins3	574.5171832693950
CMJ2_N100_M3_ins3	285.7296742234393	CMJ2_N200_M3_ins4	588.7981203560063
CMJ2_N100_M3_ins4	287.5533120463056		

Table 3. Optimal results with 13 decimals

MJACPT is a maximization problem, coefficients of objective function (matrix H) were obtained by following formula $h_{ij}^{MJACPT} = \max H + \min H - h_{ij}^{GAP}$, where $\max H = \max_{i,j} h_{ij}^{GAP}$ and $\min H = \min_{i,j} h_{ij}^{GAP}$.

On these large-scale instances, the CPLEX run was bounded at maximum 7 200 seconds. Numerical values obtained by CPLEX and GA are given in Table 4. Since CPLEX did not finish its work within 7 200 seconds for any instance, suboptimal result obtained at 7 200 seconds was presented. GA was run on the same criterion as for Tables 1 and 2. The value and meaning of every column are the same as in Tables 1 and 2.

The summary of the comparison of GA and CPLEX for these large-scale instances is graphically presented in Figure 2. Note that, on instances with $m = 20$, CPLEX obtained better results than GA in 4 of 6 cases. On instances where $m = 40$, GA was better in 5 of 9 cases; and finally, on instances where $m = 80$, GA produced better results in all cases. To sum up, GA gave better results in 16 of 24 cases. Differences in particular $m = 80$ were considerable, and above all, in one case, CPLEX could not even obtain a meaningful result (instance ga_mjacpt_gap_d_M80_N100). Running time of GA in all these instances is reasonably short, up to 450 seconds).

Some characteristic parameters of GA were also tested. They included elitism and frozen genes as explained in Section 4. There were two tests. The first test was

Instance name	$CPLEX_{sol}$	GA_{best}	t (sec)	t_{tot} (sec)	*gen*	*agap* (%)	σ (%)	*eval*	*cache* (%)
gap_d_M20_N400	23 718.41854	21 180.893573	306.359	390.986	155.6	0.731	0.432	10 091.0	0.2
gap_e_M20_N200	9 509.968452	8 217.285275	76.122	91.998	165.7	1.276	0.632	10 096.2	0.3
gap_e_M20_N400	18 764.93016	16 193.758021	450.573	532.063	167.8	1.541	0.816	10 077.0	0.2
gap_f_M20_N100	9 339.540846	9 636.813165	4.277	5.833	142.4	0.620	0.226	9 857.0	0.7
gap_f_M20_N200	19 272.17599	19 301.319034	16.927	21.333	157.3	0.263	0.148	10 088.0	0.3
gap_f_M20_N400	39 553.55436	38 519.015278	75.344	85.909	172.8	0.242	0.160	10 015.1	0.2
gap_d_M40_N100	4 771.738602	5 514.614569	9.012	10.165	174.7	1.195	0.663	9 992.5	0.3
gap_d_M40_N200	11 065.83301	10 992.541073	40.114	44.855	176.9	0.753	0.395	10 062.5	0.1
gap_d_M40_N400	22 986.51878	21 822.183750	209.075	246.150	168.9	0.604	0.345	10 116.4	0.1
gap_e_M40_N100	4 102.523455	4 324.954403	8.269	10.075	163.8	0.694	0.330	10 117.4	0.3
gap_e_M40_N200	8 814.769097	8 429.475253	46.591	52.702	175.8	1.012	0.642	10 135.2	0.1
gap_e_M40_N400	18 519.41278	16 567.585986	292.817	351.404	165.0	0.945	0.598	10 081.7	0.1
gap_f_M40_N100	6 698.9799	9 503.531442	6.542	8.685	147.2	0.368	0.216	9 895.4	0.3
gap_f_M40_N200	17 688.24505	19 381.774669	21.357	25.323	168.4	0.346	0.151	10 136.1	0.1
gap_f_M40_N400	37 691.67152	38 786.228670	74.563	87.330	170.2	0.200	0.106	10 141.5	0.1
gap_d_M80_N100	−9916.809537	5 533.285159	9.963	12.471	151.5	0.700	0.358	9 662.7	0.2
gap_d_M80_N200	3 923.46281	11 320.078339	38.349	40.533	189.1	0.651	0.368	10 142.7	0.1
gap_d_M80_N400	17 854.34492	22 486.025034	154.396	167.677	183.8	0.454	0.280	10 146.3	0.0
gap_e_M80_N100	601.346291	4 372.501404	9.557	11.566	164.0	0.969	0.548	10 062.9	0.2
gap_e_M80_N200	1 787.201356	8 803.339238	36.540	40.374	180.8	0.879	0.506	10 142.9	0.1
gap_e_M80_N400	16 034.77513	17 266.752053	167.328	179.676	185.8	0.842	0.438	10 141.5	0.0
gap_f_M80_N100	7 927.832539	9 408.795921	10.786	13.094	162.2	0.472	0.252	9 983.5	0.2
gap_f_M80_N200	11 913.87707	19 229.361546	35.230	38.479	182.9	0.162	0.096	10 142.7	0.1
gap_f_M80_N400	33 462.17106	39 037.567647	101.884	110.599	184.1	0.157	0.094	10 146.7	0.0

Table 4. GA results on much larger instances

run without frozen genes. The second test was run without elite individuals, e.g., the number of elite individuals was set at 0. Since GA is quite robust, it was useful to test it on one of the largest MJACPT instances, namely gap_f_M80_N400. The result for every test was the best value obtained from 20 running GA on the instance mentioned above. The summary of these tests is given in Table 5 and Figure 3.

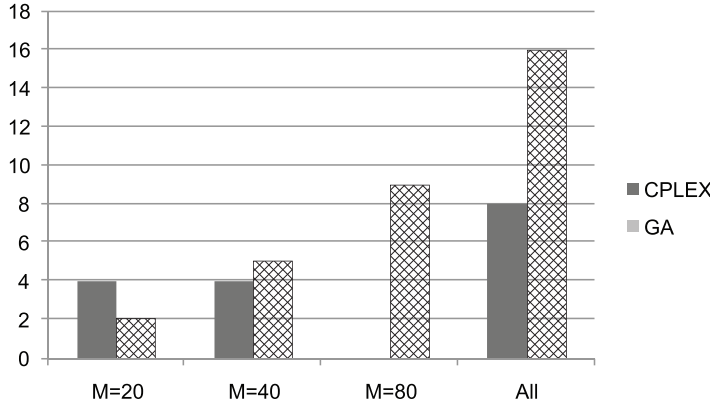


Fig. 2. Comparison between CPLEX and GA results on large instances

Kind of test	GA_{best}	t_{tot}
test with standard parameters	39 037.56765	110.599
test mutation without frozen genes	39 026.89113	113.23
test without elite individuals	39 033.41195	310.05

Table 5. Parameter sensitivity on the gap_f_M80_N400

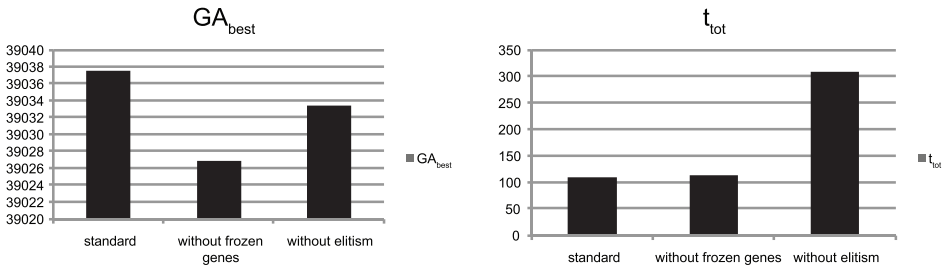


Fig. 3. Parameter sensitivity on the gap_f_M80_N400

As can be seen from Table 5, results obtained by GA without frozen genes and elite individuals were slightly worse than results obtained by GA with standard parameter values. Testing of all aspects of GA was beyond the scope of this work and can be found in literature. For example, a detailed discussion of selection and cross-over parameters can be found in [29].

6 CONCLUSIONS

The GA metaheuristic for solving MJACPT is presented in this paper. The integer representation of the job assignment was used with a constrained nonlinear convex optimization problem for obtaining controllable processing times. Local search heuristic was implemented for improving job assignment. One-point crossover and simple mutation with frozen genes were used. Computational performance of GA was improved by caching. For almost all smaller instances, except one, GA calculated solutions that matched optimal ones, obtained in a reasonable CPU time. For large-scale instances GA also produced very good solutions in comparison with CPLEX.

Based on the results, GA has the potential to be a useful metaheuristic for solving other similar problems, whether for machine job assignment or flexible manufacturing. Parallelization of the GA and its integration with exact methods are the most promising directions of future work.

Acknowledgement

The author is grateful to Jozef Kratica for his useful suggestions and comments and great general help in making this paper and to Djordje Dugosija for useful comments on a draft version of this paper.

REFERENCES

[1] Aktürk, M.S.—Atamtürk, A.—Gürel, S.: A Strong Conic Quadratic Reformulation for Machine-Job Assignment With Controllable Processing Times. BCOL Research report 07.01, Industrial Engineering & Operations Research, University of California, Berkeley, CA, 2007, 14 pages.

[2] Aparácio, J. N.—Correia, L.—Pires F. M.: Populations are Multisets – PLATO – GECCO-99. Proceedings of the Genetic and Evolutionary Computation Conference. A Joint Meeting of the Eighth International Conference on Genetic Algorithms (ICGA-99) and the Fourth Annual Genetic Programming Conference (GP-99), Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela and Robert E. Smith (Eds.), Morgan Kaufmann 1999, pp. 1845–1850.

[3] Chu, P. C. H.—Beasley, J. E.: A Genetic Algorithm for the Generalised Assignment Problem. Computers & Operations Research, Vol. 24, 1997, pp. 17–23.

[4] Djurić, B.—Kratica, J.—Tošić, D.—Filipović, V.: Solving the Maximally Balanced Connected Partition Problem in Graphs by Using Genetic Algorithm. Computing and Informatics, Vol. 27, 2008, No. 3, pp. 341–354.

[5] Filipović, V.—Kratica, J.—Tošić, D.—Ljubić, I.: Fine Grained Tournament Selection for the Simple Plant Location Problem. Proceedings of the 5th Online World Conference on Soft Computing Methods in Industrial Applications – WSC5, September 2000, pp. 152–158.

[6] Feltl, H.—Raidl G.: An Improved Hybrid Genetic Algorithm for the Generalized Assignment Problem. SAC '04, March 14–17, 2004, Nicosia, Cyprus.

[7] Filipović, V.: Fine-Grained Tournament Selection Operator in Genetic Algorithms. Computing and Informatics, Vol. 22, 2003, pp. 143–161.

[8] Filipović, V.: Operatori Selekcije i Migracije i Web Servisi Kod Paralelnih Evolutivnih Algoritama. Ph. D. thesis, University of Belgrade, Faculty of Mathematics, 2006.

[9] Gürel, S.—Aktürk, M. S.: Considering Manufacturing Cost and Scheduling Performance on a CNC Turning Machine. European Journal of Operational Research, Vol. 177, 2007, pp. 325–343.

[10] Kovacevic, J.: Hybrid Genetic Algorithm For Solving The Low-Autocorrelation Binary Sequence Problem. Yugoslav Journal of Operations Research, Vol. 19, No. 2 (to appear).

[11] Kratica, J.: Improving Performances of the Genetic Algorithm by Caching. Computers and Artificial Intelligence, Vol. 18, 1999, pp. 271–283.

[12] Kratica, J.: Improvement of Simple Genetic Algorithm for Solving the Uncapacitated Warehouse Location Problem. Advances in Soft Computing – Engineering Design and Manufacturing, R. Roy, T. Furuhashi and P. K. Chawdhry (Eds.), Springer-Verlag London Limited 1998, pp. 390–402.

[13] Kratica, J.—Tošić, D.—Filipović, V.—Ljubić, I.: Solving the Simple Plant Location Problem by Genetic Algorithms. RAIRO – Operations Research, Vol. 35, 2001, pp 127–142.

[14] Kratica J.—Tošić, D.—Filipović, V.—Ljubić, I.: A Genetic Algorithm for the Uncapacitated Network Design Problem. Soft Computing in Industry – Recent Applications, Engineering series, 2002, pp. 329–338.

[15] Kratica J.—Ljubić, I.—Tošić: A Genetic Algorithm for the Index Selection Problem. Springer Lecture Notes in Computer Science, Vol. 2611, 2003, pp. 281–291.

[16] Kratica J.—Stanimirović Z.—Tošić—Filipović: Genetic Algorithm for Solving Uncapacitated Multiple Allocation Hub Location Problem. Computers and Informatics, Vol. 24, 2005, pp. 415–426.

[17] Kratica, J.—Stanimirović, Z.: Solving the Uncapacitated Multiple Allocation P-Hub Center Problem by Genetic Algorithm. Asia Pacific Journal of Operational Research, Vol. 23, 2006, pp. 425–437.

[18] Kratica, J.—Stanimirović, Z.—Tošić, D.—Filipović, V.: Two Genetic Algorithms for Solving the Uncapacitated Single Allocation P-Hub Median Problem. European Journal of Operational Research, Vol. 182, No. 1, 2007, pp. 15–28.

[19] Kratica, J.—Kovačević-Vujčić, V.—Čangalović, M.: Computing the Metric Dimension of Graphs by Genetic Algorithms. Computational Optimization and Applications, Vol. 44, 2007, No. 2, pp. 343–361.

[20] Kratica, J.—Kovačević-Vujčić, V.—Čangalović, M.: Computing Strong Metric Dimension of Some Special Classes of Graphs by Genetic Algorithms. Yugoslav Journal of Operations Research, Vol. 18, No. 2, pp. 143–151.

[21] Kratica, J.—Kostić, T.—Tošić, D.—Dugošija, D.—Filipović, V.: A Genetic Algorithm for the Routing and Carrier Selection Problem. Computer Science and Information Systems, Vol. 9, 2012, No. 1, pp. 49–62.

[22] Ljubić, I.—Raidl, G. R.: A Memetic Algorithm for Minimum-Cost Vertex-Biconnectivity Augmentation of Graphs. Journal of Heuristics Vol. 9, 2003, pp. 401–427.

[23] Ljubić, I.: Exact and Memetic Algorithms for Two Network Design Problems. Ph. D. thesis, Institute of Computer Graphics, Vienna University of Technology, 2004.

[24] Magnasarian, O. L.: Nonlinear Programming. McGraw-Hill, New York, 1969.

[25] Marić, M.: An Efficient Genetic Algorithm for Solving the Multi-Level Uncapacitated Facility Location Problem. Computing and Informatics, Vol. 29, 2010, No. 2, pp. 183–201.

[26] Mitchell, M.: Introduction to Genetic Algorithms. MIT Press, Cambridge, Massachusetts, 1999.

[27] Savelsbergh, M. W. P.: A Branch-and-Cut Algorithm for the Generalized Assignment Problem. Intractability: A Guide to the Theory of Operations Research, Vol. 45, No. 6, 1997, pp. 831–841.

[28] Shabtay, D.—Steiner, G.: A Survey of Scheduling with Controllable Processing Times. Discrete Applied Mathematics, Vol. 155, 2007, pp. 1643–1666.

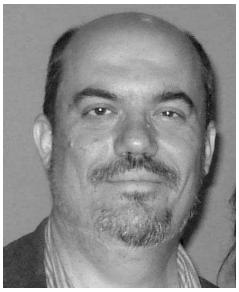
[29] Stanimirović, Z.—Kratica, J.—Dugošija, Dj.: Genetic Algorithms for Solving the Discrete Ordered Median Problem. European Journal of Operational Research, Vol. 182, 2007, No. 3, pp. 983–1001.

[30] Stanimirović, Z.: Genetic Algorithms for Solving Some NP-Hard Hub Location Problems. Ph. D. thesis, University of Belgrade, Faculty of Mathematics, 2007.

[31] Stanimirović, Z.: A Genetic Algorithm Approach for the Capacitated Single Allocation P-Hub Median Problem. Computing and Informatics 27 (in press).

[32] Wang, X.—Liu, J.—Ren, W.—Huang, M.—Gao, N.: A Job Assignment Method Based on Auction Model and Genetic Algorithm for Grid Computing. Proceedings of the Fifth international Conference on Grid and Cooperative Computing Workshops, October 21–23, 2006.

[33] Wu, C. C.—Lai, K. C.—Sun, R. Y.: GA-Based Job Scheduling Strategies for Fault Tolerant Grid Systems. IEEE Asia-Pacific Conference on Services Computing, 2008, pp. 27–32.



Aleksandar Savić received his B. Sc. degree in mathematics (1990), M. Sc. in mathematics (2000) and Ph. D. in mathematics (2010) from University of Belgrade, Faculty of Mathematics. Since 1992, he has been a teaching assistant at the Faculty of Mathematics. His research interests include genetic algorithms, mathematical programming, and operational research.