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. Cunka

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

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

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

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

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:

$$ \max \sum_{i=1}^{m} \sum_{j=1}^{n} (h_{ij}x_{ij} - f_{ij}(y_{ij})) $$
subject to
\[ \sum_{j=1}^{n} (p_{ij} x_{ij} - y_{ij}) \leq c_i, \quad i = 1, \ldots, m, \quad (4) \]
\[ y_{ij} \leq x_{ij} u_{ij}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n, \quad (5) \]
\[ \sum_{i=1}^{m} x_{ij} \leq 1, \quad j = 1, \ldots, n, \quad (6) \]
\[ x_{ij} \in \{0, 1\}, \quad y_{ij} \geq 0, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n. \quad (7) \]

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 \( \approx 114.97 \) 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

\[
\max H - \sum_{i=1}^{m} \sum_{j=1}^{n} k_{ij}y_{ij}^2
\]

with the following constraints

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

\[
0 \leq y_{ij} \leq x_{ij}u_{ij}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n
\]

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

\[
H = \sum_{i=1}^{m} \sum_{j=1}^{n} h_{ij}x_{ij}
\]

\[
A_i = \sum_{j=1}^{n} p_{ij}x_{ij}, \quad i = 1, \ldots, m
\]

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

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

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

\[
\Phi'_{y_{ij}} = -2k_{ij}y_{ij} + \lambda_iInd_i = 0,
\]

\[
\lambda_i(A_i - c_i - \sum_{j=1}^{n} y_{ij}) = 0, \quad i = 1, \ldots, m.
\]
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

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

(14)

where Lagrangean coefficients are calculated from the following formulas

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

(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];
A GA Approach for Solving MJACPT

- 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_index := Objective_Function(index);
      Put_Into.Cache_Memory(index, objvalue_index);
      if (Full_Cache_Memory)) then
        Remove_LRU_Block_From_Cache_Memory();
    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

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

$$f_{ind} = \frac{obj_{ind_{max}} - obj_{ind}}{obj_{ind_{max}} - obj_{ind_{min}}}.$$ 

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:

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

All elite individuals, their number being \(N_{\text{elite}}\), are automatically passed on to the next generation. All non-elite individuals, their number being \(N_{\text{nnel}} = N_{\text{pop}} - N_{\text{elite}}\), 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 \(N_{\text{nnel}}\), so that exactly \(N_{\text{nnel}}\) 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 \(F_{\text{tour}}\), 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 \([F_{\text{tour}}]\), and the other type is held \(k_2\) times, with tournament size \([F_{\text{tour}}]\). From here \(F_{\text{tour}} \approx \frac{k_1 [F_{\text{tour}}] + k_2 [F_{\text{tour}}]}{N_{\text{nnel}}}\).

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 \(N_{\text{pop}} = 150\) the adequate proportion is \(N_{\text{elite}} = 100\) and \(N_{\text{nnel}} = 50\). Corresponding \(k_1\) and \(k_2\) parameters in deciding tournament size are 30 and 20, respectively. Typically, for \(N_{\text{pop}} = 150\) 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 $\left\lfloor \frac{N_{nnel}}{2} \right\rfloor$ 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 CMJ2N100_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 \(\sigma\) of the average gap. A percentage gap \(agap\) is defined as \(agap = \frac{1}{20} \sum_{i=1}^{20} gap_i\), where \(gap_i = 100 \times \frac{GA_i - GA_{best}}{GA_{best}}\) and \(GA_i\) represents the GA solution obtained in the \(i\)-th run, while \(\sigma\) 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.
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 13th decimal, whereas the CPLEX result deviates already on the 2nd decimal.
A GA Approach for Solving MJACPT

<table>
<thead>
<tr>
<th>Instance name</th>
<th>CPLEX$_{sol}$</th>
<th>GA$_{opt}$</th>
<th>$t$ (sec)</th>
<th>$t_{sol}$ (sec)</th>
<th>$gen$</th>
<th>$agap$ (%)</th>
<th>$\sigma$ (%)</th>
<th>$eval$</th>
<th>$cache$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMJ2_N150_M1_ins0</td>
<td>158.498732</td>
<td>158.498735</td>
<td>1.133</td>
<td>37.556</td>
<td>101</td>
<td>0.000</td>
<td>0.000</td>
<td>16.30</td>
<td>10.9</td>
</tr>
<tr>
<td>CMJ2_N150_M1_ins1</td>
<td>156.649826</td>
<td>156.649895</td>
<td>1.208</td>
<td>29.210</td>
<td>101</td>
<td>0.000</td>
<td>0.000</td>
<td>16.40</td>
<td>10.7</td>
</tr>
<tr>
<td>CMJ2_N150_M1_ins2</td>
<td>145.304105</td>
<td>145.304127</td>
<td>1.471</td>
<td>48.870</td>
<td>101</td>
<td>0.000</td>
<td>0.000</td>
<td>14.59</td>
<td>11.7</td>
</tr>
<tr>
<td>CMJ2_N150_M1_ins3</td>
<td>155.072402</td>
<td>155.072406</td>
<td>1.116</td>
<td>39.004</td>
<td>101</td>
<td>0.000</td>
<td>0.000</td>
<td>16.78</td>
<td>10.0</td>
</tr>
<tr>
<td>CMJ2_N150_M1_ins4</td>
<td>159.137388</td>
<td>159.137365</td>
<td>1.998</td>
<td>51.548</td>
<td>101</td>
<td>0.000</td>
<td>0.000</td>
<td>16.95</td>
<td>11.4</td>
</tr>
<tr>
<td>CMJ2_N150_M1_ins5</td>
<td>308.380354</td>
<td>308.380356</td>
<td>6.716</td>
<td>48.632</td>
<td>112</td>
<td>0.000</td>
<td>0.000</td>
<td>5.11</td>
<td>11.5</td>
</tr>
<tr>
<td>CMJ2_N150_M2_ins0</td>
<td>306.466615</td>
<td>306.466620</td>
<td>1.923</td>
<td>59.851</td>
<td>101</td>
<td>0.000</td>
<td>0.000</td>
<td>15.28</td>
<td>12.9</td>
</tr>
<tr>
<td>CMJ2_N150_M2_ins1</td>
<td>306.476914</td>
<td>306.477038</td>
<td>2.037</td>
<td>64.436</td>
<td>101</td>
<td>0.000</td>
<td>0.000</td>
<td>4.51</td>
<td>13.1</td>
</tr>
<tr>
<td>CMJ2_N150_M2_ins2</td>
<td>295.108654</td>
<td>295.108662</td>
<td>4.533</td>
<td>56.072</td>
<td>106</td>
<td>0.000</td>
<td>0.000</td>
<td>4.81</td>
<td>11.6</td>
</tr>
<tr>
<td>CMJ2_N150_M2_ins3</td>
<td>306.719459</td>
<td>306.719503</td>
<td>4.199</td>
<td>54.772</td>
<td>105</td>
<td>0.000</td>
<td>0.000</td>
<td>4.77</td>
<td>11.9</td>
</tr>
<tr>
<td>CMJ2_N150_M3_ins0</td>
<td>449.679904</td>
<td>449.682666</td>
<td>10.239</td>
<td>66.315</td>
<td>116</td>
<td>0.000</td>
<td>0.000</td>
<td>5.64</td>
<td>4.6</td>
</tr>
<tr>
<td>CMJ2_N150_M3_ins1</td>
<td>439.922295</td>
<td>439.922499</td>
<td>45.548</td>
<td>93.546</td>
<td>168</td>
<td>0.015</td>
<td>0.019</td>
<td>8.17</td>
<td>4.7</td>
</tr>
<tr>
<td>CMJ2_N150_M3_ins2</td>
<td>449.126137</td>
<td>449.126141</td>
<td>62.689</td>
<td>112.319</td>
<td>175</td>
<td>0.067</td>
<td>0.077</td>
<td>8.48</td>
<td>4.9</td>
</tr>
<tr>
<td>CMJ2_N150_M3_ins3</td>
<td>424.195738</td>
<td>424.196692</td>
<td>27.467</td>
<td>77.111</td>
<td>150</td>
<td>0.011</td>
<td>0.023</td>
<td>7.29</td>
<td>4.9</td>
</tr>
<tr>
<td>CMJ2_N150_M3_ins4</td>
<td>433.641230</td>
<td>433.653701</td>
<td>56.574</td>
<td>110.268</td>
<td>172</td>
<td>0.039</td>
<td>0.032</td>
<td>8.36</td>
<td>4.7</td>
</tr>
<tr>
<td>CMJ2_N200_M1_ins0</td>
<td>208.589725</td>
<td>208.589725</td>
<td>2.106</td>
<td>63.542</td>
<td>101</td>
<td>0.000</td>
<td>0.000</td>
<td>4.73</td>
<td>8.9</td>
</tr>
<tr>
<td>CMJ2_N200_M1_ins1</td>
<td>210.556831</td>
<td>210.556835</td>
<td>2.817</td>
<td>78.042</td>
<td>101</td>
<td>0.000</td>
<td>0.000</td>
<td>4.73</td>
<td>9.0</td>
</tr>
<tr>
<td>CMJ2_N200_M1_ins2</td>
<td>200.233448</td>
<td>200.233450</td>
<td>3.680</td>
<td>84.405</td>
<td>101</td>
<td>0.000</td>
<td>0.000</td>
<td>4.72</td>
<td>9.1</td>
</tr>
<tr>
<td>CMJ2_N200_M1_ins3</td>
<td>204.161481</td>
<td>204.161493</td>
<td>3.511</td>
<td>93.942</td>
<td>101</td>
<td>0.000</td>
<td>0.000</td>
<td>4.70</td>
<td>9.4</td>
</tr>
<tr>
<td>CMJ2_N200_M1_ins4</td>
<td>413.876559</td>
<td>413.876565</td>
<td>4.973</td>
<td>113.092</td>
<td>101</td>
<td>0.000</td>
<td>0.000</td>
<td>4.68</td>
<td>11.2</td>
</tr>
<tr>
<td>CMJ2_N200_M1_ins5</td>
<td>406.545872</td>
<td>406.545878</td>
<td>18.031</td>
<td>141.491</td>
<td>111</td>
<td>0.000</td>
<td>0.000</td>
<td>5.14</td>
<td>10.1</td>
</tr>
<tr>
<td>CMJ2_N200_M2_ins0</td>
<td>419.161115</td>
<td>419.161786</td>
<td>19.694</td>
<td>128.483</td>
<td>115</td>
<td>0.002</td>
<td>0.006</td>
<td>5.23</td>
<td>10.5</td>
</tr>
<tr>
<td>CMJ2_N200_M2_ins1</td>
<td>394.457708</td>
<td>394.457714</td>
<td>6.149</td>
<td>116.582</td>
<td>103</td>
<td>0.000</td>
<td>0.000</td>
<td>4.65</td>
<td>12.2</td>
</tr>
<tr>
<td>CMJ2_N200_M2_ins2</td>
<td>404.434976</td>
<td>404.434981</td>
<td>14.215</td>
<td>150.545</td>
<td>109</td>
<td>0.000</td>
<td>0.000</td>
<td>5.04</td>
<td>10.3</td>
</tr>
<tr>
<td>CMJ2_N200_M2_ins3</td>
<td>590.293082</td>
<td>590.293090</td>
<td>77.275</td>
<td>183.368</td>
<td>158</td>
<td>0.017</td>
<td>0.016</td>
<td>7.77</td>
<td>3.8</td>
</tr>
<tr>
<td>CMJ2_N200_M2_ins4</td>
<td>594.798928</td>
<td>594.798932</td>
<td>127.190</td>
<td>196.726</td>
<td>189</td>
<td>0.070</td>
<td>0.068</td>
<td>9.22</td>
<td>4.1</td>
</tr>
<tr>
<td>CMJ2_N200_M2_ins5</td>
<td>592.648084</td>
<td>591.766738</td>
<td>113.350</td>
<td>223.170</td>
<td>177</td>
<td>0.017</td>
<td>0.016</td>
<td>9.64</td>
<td>4.2</td>
</tr>
<tr>
<td>CMJ2_N200_M2_ins6</td>
<td>574.517165</td>
<td>574.332825</td>
<td>112.154</td>
<td>193.934</td>
<td>181</td>
<td>0.020</td>
<td>0.039</td>
<td>8.86</td>
<td>4.1</td>
</tr>
<tr>
<td>CMJ2_N200_M2_ins7</td>
<td>588.798115</td>
<td>588.798120</td>
<td>139.055</td>
<td>255.742</td>
<td>184</td>
<td>0.025</td>
<td>0.023</td>
<td>9.01</td>
<td>3.9</td>
</tr>
</tbody>
</table>

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 13th 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
MJACPT is a maximization problem, coefficients of objective function (matrix $H$) were obtained by following formula $h_{ij}^{\text{MJACPT}} = \max H + \min H - h_{ij}^{\text{GAP}}$, where $\max H = \max_{i,j} h_{ij}^{\text{GAP}}$ and $\min H = \min_{i,j} h_{ij}^{\text{GAP}}$.

On these large-scale instances, the CPLEX run was bounded at maximum 7200 seconds. Numerical values obtained by CPLEX and GA are given in Table 4. Since CPLEX did not finish its work within 7200 seconds for any instance, suboptimal result obtained at 7200 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-1-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

<table>
<thead>
<tr>
<th>Instance name</th>
<th>Opt.sol</th>
<th>Instance name</th>
<th>Opt.sol</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMJ2_N01_M1_n50</td>
<td>49.6148574185696</td>
<td>CMJ2_N01_M1_n50</td>
<td>155.49873256103862</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n51</td>
<td>50.9836276363933</td>
<td>CMJ2_N01_M1_n51</td>
<td>156.6469845071603</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n52</td>
<td>49.4012148958418</td>
<td>CMJ2_N01_M1_n52</td>
<td>145.3041237020900</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n53</td>
<td>57.2738306637174</td>
<td>CMJ2_N01_M1_n53</td>
<td>155.5724063245655</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n54</td>
<td>48.6627169304325</td>
<td>CMJ2_N01_M1_n54</td>
<td>159.381560812547</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n55</td>
<td>96.0504638126502</td>
<td>CMJ2_N01_M1_n55</td>
<td>203.830357289660</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n56</td>
<td>102.86855322583</td>
<td>CMJ2_N01_M1_n56</td>
<td>306.466619543429</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n57</td>
<td>93.505242120723</td>
<td>CMJ2_N01_M1_n57</td>
<td>306.477038197365</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n58</td>
<td>49.7030943641975</td>
<td>CMJ2_N01_M1_n58</td>
<td>295.1086624543456</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n59</td>
<td>104.4087295836986</td>
<td>CMJ2_N01_M1_n59</td>
<td>306.7195301962992</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n60</td>
<td>105.5204778747314</td>
<td>CMJ2_N01_M1_n60</td>
<td>449.826660346746</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n61</td>
<td>146.0709784188812</td>
<td>CMJ2_N01_M1_n61</td>
<td>439.922499034549</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n62</td>
<td>134.3203691945234</td>
<td>CMJ2_N01_M1_n62</td>
<td>449.526141003849</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n63</td>
<td>143.6944161293727</td>
<td>CMJ2_N01_M1_n63</td>
<td>424.196610942328</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n64</td>
<td>142.8815259980024</td>
<td>CMJ2_N01_M1_n64</td>
<td>433.653705607537</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n65</td>
<td>102.5891345863531</td>
<td>CMJ2_N01_M2_n65</td>
<td>208.5897252356628</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n66</td>
<td>103.9195389388555</td>
<td>CMJ2_N01_M2_n66</td>
<td>210.556834855265</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n67</td>
<td>96.6965637705956</td>
<td>CMJ2_N01_M2_n67</td>
<td>200.2334498731282</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n68</td>
<td>105.1304152605631</td>
<td>CMJ2_N01_M2_n68</td>
<td>204.1613092073876</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n69</td>
<td>106.4550969083245</td>
<td>CMJ2_N01_M2_n69</td>
<td>413.8765340493246</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n70</td>
<td>201.2398675065352</td>
<td>CMJ2_N01_M2_n70</td>
<td>436.4587577795447</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n71</td>
<td>205.7838017264405</td>
<td>CMJ2_N01_M2_n71</td>
<td>419.161757029970</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n72</td>
<td>196.145842890325</td>
<td>CMJ2_N01_M2_n72</td>
<td>394.455714050253</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n73</td>
<td>197.1896537591701</td>
<td>CMJ2_N01_M2_n73</td>
<td>404.434981036758</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n74</td>
<td>201.8047515684939</td>
<td>CMJ2_N01_M2_n74</td>
<td>590.2930901105431</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n75</td>
<td>290.532091460782</td>
<td>CMJ2_N01_M2_n75</td>
<td>594.789320213267</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n76</td>
<td>291.685018552467</td>
<td>CMJ2_N01_M2_n76</td>
<td>591.7667378549382</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n77</td>
<td>294.243765270240</td>
<td>CMJ2_N01_M2_n77</td>
<td>574.5171832693950</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n78</td>
<td>285.7260742234939</td>
<td>CMJ2_N01_M2_n78</td>
<td>588.7981205500656</td>
</tr>
<tr>
<td>CMJ2_N01_M1_n79</td>
<td>287.5533120463056</td>
<td>CMJ2_N01_M2_n79</td>
<td>588.7981205500656</td>
</tr>
</tbody>
</table>

Table 3. Optimal results with 13 decimals
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_less_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.

<table>
<thead>
<tr>
<th>Instance name</th>
<th>CPLEX_val</th>
<th>GAbest</th>
<th>t (sec)</th>
<th>t_{tot} (sec)</th>
<th>gen</th>
<th>agap (%)</th>
<th>σ</th>
<th>eval</th>
<th>cache (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gap_less_M20_N400</td>
<td>23 718.418 54</td>
<td>21 180.893 57</td>
<td>306.359</td>
<td>390.986</td>
<td>135.5</td>
<td>0.741</td>
<td>0.432</td>
<td>10 091.0</td>
<td>0.3</td>
</tr>
<tr>
<td>gap_less_M20_N200</td>
<td>9 509.968 452</td>
<td>8 217.285 527</td>
<td>76.122</td>
<td>91.998</td>
<td>165.7</td>
<td>1.276</td>
<td>0.632</td>
<td>10 096.2</td>
<td>0.3</td>
</tr>
<tr>
<td>gap_less_M20_N400</td>
<td>18 764.930 16</td>
<td>16 193.758 021</td>
<td>450.573</td>
<td>532.063</td>
<td>167.8</td>
<td>1.541</td>
<td>0.816</td>
<td>10 077.0</td>
<td>0.2</td>
</tr>
<tr>
<td>gap_less_M20_N100</td>
<td>9 339.540 846</td>
<td>9 636.813 165</td>
<td>4.277</td>
<td>5.833</td>
<td>142.4</td>
<td>0.620</td>
<td>0.226</td>
<td>9 857.0</td>
<td>0.7</td>
</tr>
<tr>
<td>gap_less_M20_N200</td>
<td>19 272.175 99</td>
<td>19 301.319 034</td>
<td>16.927</td>
<td>21.333</td>
<td>157.3</td>
<td>0.263</td>
<td>0.148</td>
<td>10 088.0</td>
<td>0.3</td>
</tr>
<tr>
<td>gap_less_M20_N400</td>
<td>39 553.554 36</td>
<td>38 519.015 27</td>
<td>75.344</td>
<td>85.909</td>
<td>172.8</td>
<td>0.242</td>
<td>0.160</td>
<td>10 015.1</td>
<td>0.2</td>
</tr>
<tr>
<td>gap_less_M40_N100</td>
<td>4 771.736 802</td>
<td>5 514.614 569</td>
<td>9.012</td>
<td>10.165</td>
<td>174.7</td>
<td>1.195</td>
<td>0.663</td>
<td>9 992.5</td>
<td>0.3</td>
</tr>
<tr>
<td>gap_less_M40_N200</td>
<td>11 065.833 01</td>
<td>10 992.541 073</td>
<td>40.114</td>
<td>44.855</td>
<td>176.9</td>
<td>0.753</td>
<td>0.395</td>
<td>10 062.5</td>
<td>0.1</td>
</tr>
<tr>
<td>gap_less_M40_N400</td>
<td>22 986.518 75</td>
<td>21 822.183 750</td>
<td>209.075</td>
<td>246.150</td>
<td>168.9</td>
<td>0.604</td>
<td>0.345</td>
<td>10 116.4</td>
<td>0.1</td>
</tr>
<tr>
<td>gap_less_M40_N100</td>
<td>4 102.523 455</td>
<td>4 324.954 104</td>
<td>8.269</td>
<td>10.075</td>
<td>163.8</td>
<td>0.694</td>
<td>0.330</td>
<td>10 117.3</td>
<td>0.4</td>
</tr>
<tr>
<td>gap_less_M40_N200</td>
<td>8 814.769 097</td>
<td>8 429.475 253</td>
<td>46.591</td>
<td>52.702</td>
<td>175.8</td>
<td>1.012</td>
<td>0.642</td>
<td>10 135.2</td>
<td>0.1</td>
</tr>
<tr>
<td>gap_less_M40_N400</td>
<td>18 519.412 78</td>
<td>16 567.598 566</td>
<td>292.817</td>
<td>351.404</td>
<td>165.0</td>
<td>0.945</td>
<td>0.598</td>
<td>10 081.7</td>
<td>0.1</td>
</tr>
<tr>
<td>gap_less_M40_N100</td>
<td>6 698.979 97</td>
<td>9 503.531 442</td>
<td>6.542</td>
<td>8.685</td>
<td>147.2</td>
<td>0.368</td>
<td>0.216</td>
<td>9 895.4</td>
<td>0.3</td>
</tr>
<tr>
<td>gap_less_M40_N200</td>
<td>17 688.245 05</td>
<td>19 381.774 669</td>
<td>21.357</td>
<td>25.323</td>
<td>168.4</td>
<td>0.346</td>
<td>0.151</td>
<td>10 136.1</td>
<td>0.1</td>
</tr>
<tr>
<td>gap_less_M40_N400</td>
<td>37 691.671 52</td>
<td>38 786.228 670</td>
<td>74.563</td>
<td>87.330</td>
<td>170.2</td>
<td>0.200</td>
<td>0.106</td>
<td>10 141.5</td>
<td>0.1</td>
</tr>
<tr>
<td>gap_less_M80_N100</td>
<td>-9 916.809 537</td>
<td>5 533.285 159</td>
<td>9.963</td>
<td>12.471</td>
<td>151.5</td>
<td>0.700</td>
<td>0.358</td>
<td>9 662.7</td>
<td>0.2</td>
</tr>
<tr>
<td>gap_less_M80_N200</td>
<td>3 923.462 81</td>
<td>11 320.073 383</td>
<td>38.349</td>
<td>40.533</td>
<td>189.1</td>
<td>0.651</td>
<td>0.368</td>
<td>10 142.7</td>
<td>0.1</td>
</tr>
<tr>
<td>gap_less_M80_N400</td>
<td>17 854.344 92</td>
<td>22 486.025 034</td>
<td>154.396</td>
<td>167.677</td>
<td>183.8</td>
<td>0.454</td>
<td>0.280</td>
<td>10 146.3</td>
<td>0.0</td>
</tr>
<tr>
<td>gap_less_M80_N100</td>
<td>601.346 291</td>
<td>4 372.501 404</td>
<td>9.557</td>
<td>11.566</td>
<td>164.0</td>
<td>0.969</td>
<td>0.548</td>
<td>10 062.9</td>
<td>0.2</td>
</tr>
<tr>
<td>gap_less_M80_N200</td>
<td>1 757.201 356</td>
<td>8 803.399 238</td>
<td>36.540</td>
<td>40.374</td>
<td>180.8</td>
<td>0.879</td>
<td>0.506</td>
<td>10 142.9</td>
<td>0.1</td>
</tr>
<tr>
<td>gap_less_M80_N400</td>
<td>16 034.775 13</td>
<td>17 266.752 053</td>
<td>167.328</td>
<td>179.676</td>
<td>185.8</td>
<td>0.842</td>
<td>0.438</td>
<td>10 141.5</td>
<td>0.0</td>
</tr>
<tr>
<td>gap_less_M80_N100</td>
<td>7 927.832 39</td>
<td>9 408.795 921</td>
<td>10.786</td>
<td>13.094</td>
<td>162.2</td>
<td>0.472</td>
<td>0.252</td>
<td>9 983.5</td>
<td>0.2</td>
</tr>
<tr>
<td>gap_less_M80_N200</td>
<td>11 913.877 07</td>
<td>19 229.361 546</td>
<td>35.230</td>
<td>38.479</td>
<td>182.9</td>
<td>0.162</td>
<td>0.096</td>
<td>10 142.7</td>
<td>0.1</td>
</tr>
<tr>
<td>gap_less_M80_N400</td>
<td>33 462.171 06</td>
<td>39 037.567 647</td>
<td>101.884</td>
<td>110.599</td>
<td>184.1</td>
<td>0.157</td>
<td>0.094</td>
<td>10 146.7</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 4. GA results on much larger instances.

![Fig. 2. Comparison between CPLEX and GA results on large instances](image-url)
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


A GA Approach for Solving MJACPT


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.