Kragujevac Journal of Mathematics
Volume 35 Number 1 (2011), Pages 119–138.

# A GENETIC ALGORITHM FOR SOLVING MULTIPLE WAREHOUSE LAYOUT PROBLEM

DRAGAN MATIĆ [1], VLADIMIR FILIPOVIĆ [2], ALEKSANDAR SAVIĆ [3], AND ZORICA STANIMIROVIĆ [4]

Abstract. In this paper we present a genetic algorithm (GA) for solving NP-hard Multiple Warehouse Layout Problem (MLWLP). New encoding scheme with appropriate objective functions is implemented. Specific representation and modified genetic operators keep individuals correct and help in restoring good genetic material and avoiding premature convergence in suboptimal solutions. The algorithm is tested on instances generated to simulate real life problems. Experimental results show that the algorithm reaches most of optimal solutions for problems containing up to 40 item types. The algorithm is successfully tested on large scale problem instances that can not be handled by CPLEX solver due to memory limits.

## 1. Introduction

The goal of layout problem is to determine locations required for several departments in a given physical space. In practice, these layout problems are often solved by intuition using the capabilities of a human designer. However, in situations when we need fast and effective solutions for large scale input data, a human is at a disadvantage compared to a computer. Increasing needs for fast and effective solutions, especially in situations with very limited space, motivate a number of researches to investigate this problem in order to find solutions to reduce operational costs.

Warehouse layout is one of the main issues of warehousing, which is a key component of most logistical systems. Effective warehouse planning may help in reducing

*Key words and phrases.* Genetic algorithms, Warehouse Layout, Discrete Optimization.
2010 *Mathematics Subject Classification.* Primary: 90C27, Secondary: 90B06.
*Received*: July 18, 2010.
*Revised*: November 21, 2010.

material handling cost and increasing productivity as well. Therefore, it plays an important role in making the best strategy to manage the warehouse. The goal of the Warehouse Layout Problem (WLP) is to determine locations of items in a storage system by taking into account certain constraints. In practice, there exist two types of the WLP: single-level WLP (SLWLP) and multiple-level WLP (MLWLP). In the SLWLP, item transportation costs are directly related to the positions of cells (cells that are closer to an I/O port assume lower transportation costs for items). In the MLWLP, the priority of cells in different levels is item type-dependent and the closeness of the cell has to be considered both horizontally and vertically.

The earliest work on multiple-level layout problems is presented in [1]. The authors investigated the problem of relative location of facilities in a multiple- floor building. They proposed a heuristic solution procedure, based on the CRAFT heuristic [2]

Notable works in solving the MLWLP can be found in [3] and [4]. In [3], the authors developed a genetic algorithm approach by using the multiple storage areas in different levels of a warehouse. At each level, the same set of cells is used to store several item types (according to the inventory requirement), in order to minimize total transportation cost in a planning period. An integer programming model was proposed, due to the similarity with the NP-hard problem described in [5]. In [4], the authors extended their previous work and proposed two hybridizations of the GA and the path relinking strategy. The first hybrid method integrated path relinking as one of the evolution operations in the GA, and the second one applied path relinking when the GA was trapped in a local optimum.

Recent contribution in solving MLWLP [6] is presented by the same authors as in [4]. The authors investigated a new variant of the MLWLP, named the Multi-Level Warehouse Layout Problem with Adjacency Constraints (MLWLPAC). In the MLWLPAC it is required that the same item type is located in adjacent cells, while horizontal and vertical unit travel costs are product dependent. They proposed an integer programming model for the formulation of the MLWLPAC and developed several tabu search techniques for solving it.

Similar problem was investigated in [7], where different types of items needed to be placed in a multi-level warehouse and the monthly demand of each item type and horizontal distance travelled by clamp track are treated as fuzzy variables. Another recent contribution in this field is a study by Önüt et al. in [8]. The authors used a particle swarm optimization algorithm to manage multiple-level warehouse shelf configuration in order to minimize annual carrying costs. A detailed survey of various contributions of warehousing systems is out of scope of this paper and can be found in [9].

### 1.1. PROBLEM DESCRIPTION

In the variant of the MLWLP considered in this paper, we have the following assumptions. The warehouse has multiple-level storage space, divided in cells with one elevator to transport items from the ground to other levels. It is assumed that the elevator has enough capacity, which means that the vertical transportation operation is always available. Each level is divided into cells of the same dimension. The number of cells on different levels may vary. There is only one I/O port on the ground, placed at the same vertical location as the elevator. Different item types need to be stored in the multiple-level warehouse. Each item type has its own monthly demand and inventory volume, vertical unit transportation cost (i.e. the cost to move one unit of the item between the ground and other level) and horizontal unit transportation cost (i.e. the cost to move one unit of the item $1m$ in horizontal distance). Each item must be assigned to exactly one cell. A cell may store more than one item type. The objective is to minimize the total vertical and horizontal transportation cost.

### 1.2. MATHEMATICAL FORMULATION

In this paper, the formulation of the MLWLP from [3] is used. Suppose that $J, L, K_l$ represent the total number of item types, levels and cells available on the $l-th$ level, respectively. The notation of the problem parameters is given in Table 1. Binary variables $x_{jlk} \in \{0, 1\}$ are introduced, taking the value of 1 if the $j$-th item type is assigned to the $k$-th cell of the level $l$, 0 otherwise.

The arrangement of the item types (the solution of the problem) is considered feasible if: (i) each item type is assigned to exactly one cell and (ii) no cell capacity is violated.

Using the above notation, the problem can be represented as following integer-programming formulation [3]:

$$\min \sum_{j=1}^{J} \sum_{l=1}^{L} \sum_{k=1}^{K_l} Q_j (D_{lk} C_j^h + C_{jl}^v) x_{jlk} \tag{1.1}$$

| | |
|---|---|
| $j \in \{1,2,...,J\}$ | item types |
| $l \in \{1,2,...,L\}$ | levels |
| $k \in \{1,2,...,K_l\}$ | available cells for the $l$-th level |
| $Q_j$ | monthly demand of the item type $j$ |
| $S_j$ | inventory requirement of the item type $j$ |
| $C_j^h$ | horizontal unit transportation cost of the item type $j$ |
| $C_{jl}^v$ | vertical unit transportation cost of the item type $j$ to the level $l$ |
| $A$ | storage capacity of a cell (same for all cells) |
| $D_{lk}$ | horizontal distance from the cell $k$ on the level $l$ to the I/O port or eleveator |

Table 1. List of parameters for the MLWLP

subject to

$$\sum_{l=1}^{L}\sum_{k=1}^{K_l} x_{jlk} = 1, \text{for } j = 1,2,...,J \tag{1.2}$$

$$\sum_{j=1}^{J} S_j x_{jlk} \le A, \text{for } l = 1,2,...,L,\ \ k = 1,2,...,K_l \tag{1.3}$$

$$x_{jlk} \in \{0,1\},\ \text{ for all } j,l,k. \tag{1.4}$$

The objective function (1.1) minimizes the total horizontal and vertical transportation cost of all item types. Constraints (1.2) provide that each item type must be assigned exactly to one cell in the warehouse. By constraints (1.3) capacity violation in cells is prevented. Binary nature of variables $x_{jlk}$ is determined by (1.4). The MLWLP is proved to be NP-hard, since it is equivalent to the problem $SP_i$ from [5] when number of levels set to $L = 1$.

*Example* 1.1. Suppose that we have 5 item types ($J = 5$), two levels ($L = 2$) and 3 cells for each level ($K_1 = K_2 = 3$). Let capacity of each cell be equal to 16 ($A = 16$). For each item type, monthly demands, inventory requirements, horizontal costs and vertical costs for both levels (L1 and L2), are given in Table 2. Horizontal distances from the cells to the I/O port are given in Table 3.

If item types are denoted as 1, 2, 3, 4 and 5 and levels as Level I and Level II, one solution is given in Table 4. We see that the first cells from both levels are left empty, while in second cell of the first level two item types (3 and 5) are placed. This is not

| Item | Mon. demand | Inv. req. | H. cost | V. cost L1 | V. cost L2 |
|---|---|---|---|---|---|
| 1 | 136 | 16 | 13.258073 | 1.672291 | 5.205750 |
| 2 | 32 | 16 | 13.470847 | 3.398790 | 6.218008 |
| 3 | 127 | 7 | 13.816301 | 8.647548 | 13.281618 |
| 4 | 15 | 11 | 12.972679 | 2.108475 | 2.963847 |
| 5 | 72 | 7 | 12.028499 | 3.081751 | 8.347578 |

Table 2. Input data for simple example

| Level | Cell1 | Cell2 | Cell3 |
|---|---|---|---|
| 1 | 4 | 2 | 3 |
| 2 | 4 | 2 | 3 |

Table 3. Horizontal distances

| | Cell1 | Cell2 | Cell3 |
|---|---|---|---|
| Level I | | 3,5 | 2 |
| Level II | | 1 | 4 |

Table 4. One solution of the problem

a requirement violation, since the sum of inventory requirements of these two item types is 14 $(7+7)$, which is less than the cell capacity (16).

Transportation costs for each item type are calculated as follows. For item type 1: $136*(2*13.258073+5.205750) = 4314.177856$, for item type 2: $32*(3*13.470847+3.39879) = 1401.962592$, for item type 3: $127*(2*13.816301+8.647548) = 4607.57905$, for item type 4: $15*(3*12.972679+2.963847) = 628.22826$ and for item type 5: $72*(2*12.028499+3.081751) = 1953.989928$. The overall cost (objective value) is equal to 12905.93769. This solution is verified as optimal by CPLEX.

## 2. Genetic Algorithm for Solving MLWLP

GAs are complex and adaptive algorithms that are often used for solving robust optimization problems. They work over a population of individuals, where each population is a subset of the total search space and each individual represents one solution of the problem. Fitness value is assigned to each individual, indicating its relative quality in the population. From generation to generation GA tries to produce the improvement of every solution's quality, as well as better average fitness of the whole population. It is obtained by using genetic operators: selection, crossover and mutation. Detailed description of GA is out of this paper's scope and can be found in [10] and [11]. Some recent works in GA on various optimization problems show that GAs often produce high quality solutions in a reasonable computational time [12], [13] and [14].

The basic scheme of GA can be represented as: [15]:

```
Input Data();
Population Init()
while not Finish() do
    for i:=1 to Npop do
        obj[i] := Objective Function(i);
    endfor
    Fitness Function();
    Selection();
    Crossover();
    Mutation();
endwhile
Output Data();
```

*Npop* denotes the number of individuals in a population and $obj[i]$ is the objective value of the $i$-th individual.

### 2.1. REPRESENTATION AND OBJECTIVE FUNCTION

Proposed GA implementation uses the binary representation of individuals. The genetic code of an individual consists of $J$ genes ($J$ is the number of item types to be arranged into cells). Each gene corresponds to one item type and consists of $c\sqrt{n_g}$ bits, where $n_g$ is the total number of cells and $c$ is a constant with an appropriate value.

For each item type the array of cells is sorted in ascending order, according to distance to that specific item. Suppose that first "1" in a gene (looking from left to right) is located at the $p$-th bit (note that in each gene the numeration of bits starts from 0). That means that the current item is assigned to its $p$-th nearest cell. In the situation when all bits in a gene are equal to 0, the $c\sqrt{n_g}$-th cell is chosen. If the chosen cell doesn't have enough capacity for the item, we take the next cell with sufficient capacity from the sorted array.

The idea of introducing the sorted array of cells for each item comes from the observation that in optimal solutions the indices of "closer" cells appear often in the optimal solution, while the indices of "far away" cells are rare. We should also note that optimal solutions usually do not involve allocation of each item to its nearest cell.

By sorting the array of cells (for each item) in non-decreasing order of distances, it is ensured that closer cells have higher priority. In this way our search is directed to "closer" cells, while the "far away" ones are considered with smaller probability.

Arranging the array of cells is performed for each item type. This may require additional CPU time, but notice that this action is performed only once at the beginning of the algorithm. If the total number of cells is $n_g$ and total number of item types is $J$, the running-time complexity of this step is $O(Jn_g \log n_g)$. Although the total running time is slightly longer, our experiments show that this strategy can significantly improve the quality of solutions.

Calculating of the objective function of an individual is as follows. For each item type, the array of cells is arranged in nondecreasing order with respect to distances to the current cell. We take the gene that corresponds to the current item type and find the index $p$ of the first appearance of bit "1". The found index $p$ indicates that the current item type is assigned to its $p$-th nearest cell. In the case that this cell has insufficient capacity, we take the next one with enough capacity from the sorted array of cells.

After the assigning procedure has been performed, the objective value is simply evaluated by summing the costs for each item type.

*Example* 2.1. Using the same input data as in Example 1.1, the genetic code

$$010|110|001|000|010$$

indicates that the first item is assigned to the second nearest cell (cell 2 on Level II), the second one to its nearest cell (cell 2 on Level I) and the third item to the third nearest cell (cell 3 on Level I). The gene corresponding to the fourth item contains three zeros, which means that the item is assigned to the fourth nearest cell (cell 3 on Level II). The fifth item is attempting to be assigned to the second nearest cell (cell 2 on Level II), but that cell is full, because it already contains item type 1 (inventory requirement for that item type is equal to cell capacity - 16). So, we are trying to place the fifth item to the next cell (the third nearest), which is cell 3 on Level I. This cell is not empty (it contains item type 3 with the inventory requirement 7) and by assigning the fifth item this cell, the inventory requirement will not be violated. Finally, on the first level cell 1 is empty, cell 2 contains item type 2 and cell 3 contains two item types (3 and 5). On the second level first cell is empty, the second cell contains item type 1 and the third cell item type 4. The total cost of this assignment is 15095.59274.

The optimal solution presented in Example 1.1 is encoded as 011|001|111|001|100. The first item type is assigned to the second nearest cell (cell 2 on Level II) and the second and the fourth item types to the third nearest cells (cells 3 on the Level I and Level II, respectively). The third and the fifth item types are assigned to the cell 2 on Level I, which is the nearest to both item types. Note that the representation of optimal solution may not be unique. For example, 010|001|100|001|101 also corresponds to the same (optimal) solution.

## 2.2. GENETIC OPERATORS

The algorithm uses fine-grained tournament selection (FGTS) [16] that is an improvement of standard tournament selection. It is used in cases when the average tournament size $F_{tour}$ is desired to be fractional. The FGTS realizes two types of tournaments: the first type is held $k_1$ times and its size is $[F_{tour}] + 1$. The second type is realized $k_2$ times with $[F_{tour}]$ participants. In our implementation, $F_{tour}$ is set to 5.4. For example, if the FGTS is applied to $N_{nonel} = 50$ non-elitist individuals, tournaments are held $k_1 = 20$ and $k_2 = 30$ times with sizes 6 and 5 respectively. The running time for the FGTS operator is $O(N_{nonel} * F_{tour})$. In our GA implementation, $F_{tour}$ and $N_{nonel}$ are considered to be constant (not depending on a problem size), which gives a constant time complexity.

Standard one-point crossover operator is implemented in the proposed GA. It exchanges whole genes of parent-individuals after randomly chosen crossover point, producing two individuals-offspring. The crossover is performed with the probability $p_{cross} = 0.85$.

Offspring generated by a crossover operator are subject to mutation with frozen bits. The mutation operator is realized by changing a randomly selected gene in the genetic code. The probability of mutation $GA_{prob}$ depends on starting probability parameter $StartP$ (in our case, $StartP = 1.4$) and the number of bits representing each gene, as it is shown by the formula:

$$GA_{prob} = \frac{StartP}{c\sqrt{n_g}}, \tag{2.1}$$

where $n_g$ is the total number of cells, and $c$ is constant(in our case $c = 3$).

During the GA execution it may happen that (almost) all individuals in the population have the same bit value on certain position. These bits are called frozen. If the number of frozen bits is $n_f$, the search space becomes $2^{n_f}$ times smaller and the possibility of premature convergence rapidly increases [17]. Selection and crossover operators can not change any frozen bit value and basic mutation rate is often insufficiently small to restore lost subregions of the search space. However, if we increase basic mutation rate significantly, genetic algorithm becomes random search. For this reason, basic mutation rates are increased only on frozen bits, by multiplying the basic mutation rate by the coefficient called frozen factor. In our case, the frozen factor is set to 5.5.

### 2.3. OTHER GA ASPECTS

The initial population consists of 150 individuals. Each gene of an individual is randomly generated with uniform probability. One-third of the population is replaced in every generation ($N_{nonel} = 50$), except the best 100 individuals that directly pass to the next generation. These elite individuals preserve highly fitted genes of the population. Their objective values are calculated only in the first generation. The applied encoding scheme excludes the appearance of incorrect individuals in the initial population.

If an individual with the same genetic code repeats in the population, its objective value is set to zero. The selection operator disables duplicated individuals from entering the next generation. This strategy helps to preserve the diversity of genetic material and to keep the algorithm away from the local optima trap. Individuals with the same objective value, but different genetic codes may dominate in the population after certain number of iterations. If their codes are similar, it may cause a premature convergence of the GA. For this reason, we have limited the appearance of these individuals to some constant $N_{rv}$ to number 40.

Note that proposed GA concept is quite different from other evolutionary -based approaches dealing with the same variant of MLWLP. In the GA methods described in [4], both proposed approaches use path relinking as a main strategy for achieving good quality solutions and diversity of genetic material. In the first approach, path relinking is used as a crossover operator, while in second one local search uses path relinking to avoid premature convergence in suboptimal solutions.

The main idea of the GA proposed in this paper is the use of effective representation of individuals and problem-specific objective function. The key aspects of the GA concept are good initial assignment and good searching strategy, based on a principle that item types should generally be assigned to "closer" cells. Genetic operators, adequate to the considered problem, are chosen and implemented in such a way to keep the efficiency of the algorithm. The increase of mutation rate on frozen bits and several other GA aspects mentioned above keep the diversibility of genetic material and prevent the GA to finish in local optimum.

Computational results presented in Chapter 3. show that our GA approach achieves high-quality solution. This indicates that local search could not significantly improve the solutions obtained by evolutionary based method, so that any local search would unnecessary increase the execution time of the algorithm. Direct application of genetic operators to the individuals and the absence of local search makes the proposed GA very fast, even for large scale instances, which is proved by computational results, as it can bee seen from Chapter 3.

## 3. Computational Results

In this section we present and discuss computational results of the proposed GA method. The GA implementation was coded in C programming language. All tests were carried out on the Intel Core 2 Quad Q9400 @2.66 GHz with 8 GB RAM. In order to provide optimal solutions for small size problem instances, an integer linear programming model for the MLWLP was tested on CPLEX optimization package version 10.1(www.ibm.com/software/integration/optimization/cplex-optimizer/).

Since instances from [3] were unavailable to us, we have generated instances in the same way, based on the characteristics of the real problems. Input and output parameters of the algorithm for generating instances are given in Table 5 and Table 6, respectively. In the first and the second column (of Tables 5 and 6) parameter names and description are given. In the last column of Table 5 input values of the parameters are presented. The last column of the Table 6 shows the way of calculating the output values.

| Parameter | Description | Value |
|---|---|---|
| $nj$ | number of items | for small instances: 10-40<br>for large instances: 100-400 |
| $nl$ | number of levels | 2-5 |
| $A$ | cell capacity | (same for all cells, $A = 16$) |
| $\alpha$ | controls the perc. of cells with same distance | $\alpha \in [0, 1]$ |
| $\beta$ | controll parameter | $\beta \in [0.5, 1]$ |
| $parafloor[l]$ | for each level $l$, controlling parameter | $\{1, 1.5, 2.0, 2.6, 3.1\}$ |

Table 5. Input parameters for generating an instance

| Parameter | Description | Value |
|---|---|---|
| $s[j]$ | inventory requirement | 70% of item types: less than $A/2$<br>20% of item types: belongs to $[A/2, A)$<br>10% of item types: equal to $A$ |
| $kl[l]$ | array of no. of cells on the level $l$ | $1.5 \sum_{j=1}^{nj} s[j]/A/nl$ |
| $ng$ | total number of cells | $\sum_{m=1}^{nl} kl[m]$ |
| $q[j]$ | monthly demand | $q[j] \in (0.25 * s[j], 35 * s[j])$ |
| $ch[j]$ | horizontal unit cost | [10,15] |
| $d[l][k]$ | hor. dist. from cells to I/O port | $[2, 2\alpha \cdot kl[1]]$.<br>$d[l][k] = d[1][k], l = 2..nl$ |
| $cv[j][l]$ | vertical unit cost | $(0, \beta * ch[j] * d[1][kl[1]] * parafloor[l])$ |

Table 6. Output parameters of generating an instance

Tables 7-11 provide results of the GA approach for small-size problem instances ($J \leq 40,\ L \leq 5$), while Tables 12-13 contain results obtained on large instances ($100 \leq J \leq 400$ ). For each combination of number of item types and levels, the GA was benchmarked on five instances (for small dimensions) and two instances (for large dimensions), varying the parameter $\alpha$. For each problem instance, the GA was run 20 times. GA results in Tables 7-13 are presented as follows. In the first three columns, number of item types, levels and parameter $\alpha$ are given. The next column - *opt* contains the optimal solution of the current instance, obtained by CPLEX. In

Tables 12-13, the column *opt* is omitted, since on these large-scale instances CPLEX stops without generating any solution, due to memory limit.

The best GA value is given in the column $GA_{best}$, with mark *opt* in cases when GA reached optimal solution known in advance. Average time needed to detect the best GA value is given in $t$ column, while $t_{tot}$ represents average total time (in seconds) needed for finishing the GA. On average, the GA stopped after *gen* generations. The solution quality in all 20 executions ($i = 1, 2, \ldots, 20$) is evaluated as a percentage gap named $agap = \frac{1}{20}\sum_{i=1}^{20} gap_i$, where $gap_i = 100 * \frac{sol_i - Opt.sol}{Opt.sol}$ is evaluated with respect to the optimal solution $Opt.sol$, or the best-known solution $Best.sol$, i.e. $gap_i = 100 * \frac{sol_i - Best.sol}{Best.sol}$ in cases where no optimal solution is found ($sol_i$ represents the GA solution obtained in the $i$-th execution). Standard deviation of the average gap $\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(gap_i - agap)^2}$ is also presented.

Computational experiments show that for most small instances less than 500 generations were enough for GA to find the best/optimal solution. In several cases, better solutions were found when the algorithm used up to 5000 generations. However, considering that the algorithm is very fast, the usage of more generations is not a disadvantage of the GA. For the large-scale instances, the maximal number of 5000 generations is set as a stopping criterion. Algorithm also stops if the best individual or the best objective value remains unchanged through $N_{rep} = 2000$ successive generations respectively.

It is evident from Tables 7-11 that the GA quickly reaches all optimal solutions for the instances with up to 30 item types, with the exception of three instances. The column *agap* shows that for all problems with up to 20 item types (except one), the GA reaches optimal solution in each of 20 runs (in these cases, the value *agap* is equal to 0). For instances with 35 and 40 item types, the GA reaches 15 out of 40 optimal solutions. In order to investigate the dependance of instances' nature and behavior of the algorithm, the GA was tested on five instances for the same combination of item types and levels, differing in parameter $\alpha$. As we can see from Tables 7-11, the algorithm is slightly better for instances with smaller value of $\alpha$. The results indicate that instances with smaller percentage of cells of the same distance are slightly "easier" to solve. This could be explained by the fact that greater percentage of cells of the same distance increases the searching space. Therefore, the probability to find optimal solution decreases. From Tables 7-11, it can be seen that GA has found optimal solution for 112 out of 140 instances.

| It | L | $\alpha$ | *opt* | *GA* | $t$(sec) | $t_{tot}(sec)$ | *gen* | *agap*(%) | $\sigma$(%) |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 2 | 0.2 | 21062.300 | opt | 0.001 | 0.391 | 2001 | 0.000 | 0.000 |
| 10 | 3 | 0.2 | 22324.209 | opt | 0.001 | 0.412 | 2003 | 0.000 | 0.000 |
| 10 | 4 | 0.2 | 22389.975 | opt | <0.001 | 0.4025 | 2001 | 0.000 | 0.000 |
| 10 | 5 | 0.2 | 22473.226 | opt | <0.001 | 0.413 | 2003 | 0.000 | 0.000 |
| 15 | 2 | 0.2 | 31144.125 | opt | 0.001 | 0.5245 | 2001 | 0.000 | 0.000 |
| 15 | 3 | 0.2 | 28124.821 | opt | <0.001 | 0.534 | 2001 | 0.000 | 0.000 |
| 15 | 4 | 0.2 | 29878.762 | opt | 0.002 | 0.5115 | 2017 | 0.000 | 0.000 |
| 15 | 5 | 0.2 | 35417.046 | opt | 0.002 | 0.4565 | 2010 | 0.000 | 0.000 |
| 20 | 2 | 0.2 | 35156.605 | opt | <0.001 | 0.6085 | 2001 | 0.000 | 0.000 |
| 20 | 3 | 0.2 | 35682.763 | opt | <0.001 | 0.569 | 2001 | 0.000 | 0.000 |
| 20 | 4 | 0.2 | 35470.801 | opt | 0.007 | 0.5595 | 2036 | 0.000 | 0.000 |
| 20 | 5 | 0.2 | 49135.649 | opt | 0.005 | 0.5735 | 2028 | 0.000 | 0.000 |
| 25 | 2 | 0.2 | 43962.444 | opt | <0.001 | 0.7345 | 2001 | 0.000 | 0.000 |
| 25 | 3 | 0.2 | 41779.067 | opt | <0.001 | 0.733 | 2001 | 0.000 | 0.000 |
| 25 | 4 | 0.2 | 40814.643 | opt | 0.035 | 0.7245 | 2100 | 0.000 | 0.000 |
| 25 | 5 | 0.2 | 57930.108 | opt | 0.022 | 0.6775 | 2070 | 0.000 | 0.000 |
| 30 | 2 | 0.2 | 58000.906 | opt | <0.001 | 0.867 | 2001 | 0.000 | 0.000 |
| 30 | 3 | 0.2 | 51355.355 | opt | <0.001 | 0.8515 | 2001 | 0.000 | 0.000 |
| 30 | 4 | 0.2 | 47189.872 | opt | 0.002 | 0.836 | 2009 | 0.000 | 0.000 |
| 30 | 5 | 0.2 | 49518.000 | opt | 0.011 | 0.8475 | 2030 | 0.000 | 0.000 |
| 35 | 2 | 0.2 | 81913.009 | opt | 0.198 | 1.143 | 2396 | 0.063 | 1.185 |
| 35 | 3 | 0.2 | 59878.908 | opt | 0.001 | 0.9285 | 2001 | 0.000 | 0.000 |
| 35 | 4 | 0.2 | 57463.357 | opt | 0.012 | 0.9095 | 2028 | 0.000 | 0.000 |
| 35 | 5 | 0.2 | 60809.130 | opt | 0.141 | 1.042 | 2311 | 0.000 | 0.000 |
| 40 | 2 | 0.2 | 69045.672 | 69241.59 | 1.226 | 2.156 | 4108 | 0.637 | 0.475 |
| 40 | 3 | 0.2 | 67769.640 | opt | 0.045 | 1.069 | 2084 | 0.000 | 0.000 |
| 40 | 4 | 0.2 | 62302.340 | opt | 0.012 | 1.039 | 2026 | 0.000 | 0.000 |
| 40 | 5 | 0.2 | 64254.562 | opt | 0.002 | 1.004 | 2004 | 0.000 | 0.000 |

TABLE 7. GA results on small instances

Although the comparison with other works can not be fairly done due to the absence of common instances, we can roughly compare the number of achieved optimal solutions with results achieved in [4], which is the most respectable available work in this field and also is the improvement of the algorithm described in [3]. In [4], for the instances with same dimension, only 67 out of 109 instances was resolved in optimal way, with the remark that for 31 instances the authors could not determine if the solution was optimal. Even if we take into account all of these instances, we get the number 98, which is again less than total number of optimal solutions obtained by the proposed GA (112). In order to make more fair comparison, we consider the fact

| It | L | $\alpha$ | *opt* | *GA* | $t$(sec) | $t_{tot}(sec)$ | *gen* | *agap*(%) | $\sigma$(%) |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 2 | 0.4 | 19557.819 | opt | <0.001 | 0.42 | 2001 | 0.000 | 0.000 |
| 10 | 3 | 0.4 | 21909.177 | opt | 0.0015 | 0.4595 | 2012 | 0.000 | 0.000 |
| 10 | 4 | 0.4 | 22817.544 | opt | <0.001 | 0.4075 | 2001 | 0.000 | 0.000 |
| 10 | 5 | 0.4 | 22659.069 | opt | 0.0005 | 0.422 | 2002 | 0.000 | 0.000 |
| 15 | 2 | 0.4 | 25757.229 | opt | 0.001 | 0.528 | 2006 | 0.000 | 0.000 |
| 15 | 3 | 0.4 | 23617.064 | opt | 0.002 | 0.5325 | 2008 | 0.000 | 0.000 |
| 15 | 4 | 0.4 | 38256.884 | opt | <0.001 | 0.5055 | 2001 | 0.000 | 0.000 |
| 15 | 5 | 0.4 | 35376.107 | opt | 0.0015 | 0.4735 | 2007 | 0.000 | 0.000 |
| 20 | 2 | 0.4 | 55555.543 | opt | 0.199 | 0.808 | 2634 | 0.115 | 0.785 |
| 20 | 3 | 0.4 | 60265.259 | opt | 0.03 | 0.6205 | 2102 | 0.000 | 0.000 |
| 20 | 4 | 0.4 | 46134.169 | opt | <0.001 | 0.6165 | 2001 | 0.000 | 0.000 |
| 20 | 5 | 0.4 | 44392.355 | opt | 0.069 | 0.653 | 2230 | 0.000 | 0.000 |
| 25 | 2 | 0.4 | 71392.762 | opt | 0.4235 | 1.1365 | 3224 | 0.040 | 0.339 |
| 25 | 3 | 0.4 | 41933.354 | opt | 0.34 | 1.054 | 2936 | 0.101 | 0.485 |
| 25 | 4 | 0.4 | 51442.910 | opt | 0.1985 | 0.8705 | 2592 | 0.121 | 0.834 |
| 25 | 5 | 0.4 | 41413.674 | opt | 0.001 | 0.697 | 2005 | 0.000 | 0.000 |
| 30 | 2 | 0.4 | 64887.959 | opt | 0.1945 | 0.9935 | 2477 | 0.026 | 0.500 |
| 30 | 3 | 0.4 | 76866.714 | opt | 0.8455 | 1.55 | 3896 | 0.645 | 1.802 |
| 30 | 4 | 0.4 | 59766.960 | opt | 0.229 | 1.025 | 2563 | 0.075 | 0.409 |
| 30 | 5 | 0.4 | 56562.637 | opt | 0.0055 | 0.787 | 2012 | 0.000 | 0.000 |
| 35 | 2 | 0.4 | 174532.680 | 181193.5 | 1.603 | 2.14 | 4573 | 5.667 | 4.363 |
| 35 | 3 | 0.4 | 111466.958 | 113420.3 | 1.0765 | 1.8515 | 4062 | 3.436 | 3.743 |
| 35 | 4 | 0.4 | 104516.722 | 105969.8 | 0.9275 | 1.68 | 3721 | 3.511 | 2.267 |
| 35 | 5 | 0.4 | 74214.030 | opt | 0.235 | 1.114 | 2528 | 0.042 | 0.291 |
| 40 | 2 | 0.4 | 219995.719 | 232370.3 | 1.919 | 2.47 | 4825 | 6.710 | 2.813 |
| 40 | 3 | 0.4 | 93736.417 | 95508.09 | 1.749 | 2.3465 | 4636 | 2.542 | 1.401 |
| 40 | 4 | 0.4 | 84640.983 | opt | 0.2195 | 1.188 | 2451 | 0.025 | 0.171 |
| 40 | 5 | 0.4 | 95826.195 | 97222.24 | 1.1745 | 2.095 | 3862 | 2.802 | 3.162 |

TABLE 8. GA results on small instances

that in paper [4], all instances were created with parameter $\alpha = 0.8$. Even in this case, our algorithm has found 19 optimal solutions which is probably better result (by multiplying 19 with 5), or at least very similar to the result given in [4].

In the case of large-scale instances ($100 \leq J \leq 400$), optimal solutions could not be achieved by CPLEX due to memory limits. On the other hand, the GA succeeds to find the solutions in a reasonable CPU time. For the hardest instances, the total time that the GA needs to finish is up to 50 seconds. Results presented in Tables 12-13 clearly indicate that our approach can be successfully applied on large-scale

| It | L | $\alpha$ | *opt* | *GA* | $t$(sec) | $t_{tot}(sec)$ | *gen* | *agap*(%) | $\sigma$(%) |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 2 | 0.5 | 22858.064 | opt | <0.001 | 0.3965 | 2003 | 0.000 | 0.000 |
| 10 | 3 | 0.5 | 22699.255 | opt | <0.001 | 0.4245 | 2001 | 0.000 | 0.000 |
| 10 | 4 | 0.5 | 20987.273 | opt | <0.001 | 0.3885 | 2001 | 0.000 | 0.000 |
| 10 | 5 | 0.5 | 23005.107 | opt | 0.0035 | 0.4055 | 2009 | 0.000 | 0.000 |
| 15 | 2 | 0.5 | 31823.601 | opt | 0.015 | 0.5235 | 2064 | 0.000 | 0.000 |
| 15 | 3 | 0.5 | 40242.188 | opt | 0.004 | 0.5425 | 2024 | 0.000 | 0.000 |
| 15 | 4 | 0.5 | 30581.509 | opt | 0.0015 | 0.504 | 2006 | 0.000 | 0.000 |
| 15 | 5 | 0.5 | 29707.789 | opt | 0.002 | 0.4985 | 2020 | 0.000 | 0.000 |
| 20 | 2 | 0.5 | 45827.498 | opt | 0.0235 | 0.607 | 2074 | 0.000 | 0.000 |
| 20 | 3 | 0.5 | 44979.495 | opt | 0.009 | 0.5845 | 2035 | 0.000 | 0.000 |
| 20 | 4 | 0.5 | 37847.175 | opt | 0.003 | 0.5495 | 2013 | 0.000 | 0.000 |
| 20 | 5 | 0.5 | 34516.110 | opt | 0.0035 | 0.6095 | 2018 | 0.000 | 0.000 |
| 25 | 2 | 0.5 | 75767.557 | opt | 0.229 | 0.94 | 2644 | 0.171 | 0.189 |
| 25 | 3 | 0.5 | 104124.987 | opt | 0.014 | 0.6995 | 2039 | 0.000 | 0.000 |
| 25 | 4 | 0.5 | 87364.522 | opt | 0.2515 | 0.929 | 2719 | 0.485 | 1.330 |
| 25 | 5 | 0.5 | 56509.162 | opt | 0.005 | 0.6675 | 2015 | 0.000 | 0.000 |
| 30 | 2 | 0.5 | 52880.940 | opt | 0.328 | 1.16 | 2779 | 0.000 | 0.000 |
| 30 | 3 | 0.5 | 76567.909 | opt | 0.1355 | 0.98 | 2318 | 0.000 | 0.000 |
| 30 | 4 | 0.5 | 47069.236 | opt | 0.001 | 0.8165 | 2005 | 0.000 | 0.000 |
| 30 | 5 | 0.5 | 46630.487 | opt | 0.0215 | 0.8365 | 2053 | 0.000 | 0.000 |
| 35 | 2 | 0.5 | 126942.675 | opt | 0.059 | 0.9995 | 2124 | 0.000 | 0.000 |
| 35 | 3 | 0.5 | 62448.599 | 62647.81 | 0.7395 | 1.66 | 3598 | 0.485 | 1.882 |
| 35 | 4 | 0.5 | 59084.842 | opt | 0.356 | 1.228 | 2824 | 0.189 | 0.610 |
| 35 | 5 | 0.5 | 59641.176 | 59641.47 | 0.6285 | 1.483 | 3301 | 0.156 | 0.656 |
| 40 | 2 | 0.5 | 132694.076 | 133420.5 | 1.7805 | 2.4525 | 4764 | 0.968 | 1.472 |
| 40 | 3 | 0.5 | 120387.723 | 120963.6 | 1.0425 | 2.016 | 3985 | 1.277 | 2.483 |
| 40 | 4 | 0.5 | 90775.440 | 91027.21 | 1.052 | 1.8945 | 3776 | 0.756 | 2.061 |
| 40 | 5 | 0.5 | 64575.800 | opt | 0.2695 | 1.261 | 2539 | 0.321 | 0.675 |

TABLE 9. GA results on small instances

problems, in cases where exact methods fail. Rather small values of average gap and $\sigma$ indicate the reliability of the proposed algorithm.

The GA concept cannot prove optimality and adequate finishing criteria that will fine-tune the solution quality does not exist. Therefore, as the columns $t_{tot}$ in Tables 7-13 show, our algorithm runs through additional $t_{tot}$-$t$ time (until finishing criteria is satisfied), although it already reached its best solution. After all, the total running time of the GA is reasonably short, for both in small and large instances in respect to the problems' dimensions.

| It | L | $\alpha$ | *opt* | *GA* | $t$(sec) | $t_{tot}(sec)$ | *gen* | *agap*(%) | $\sigma$(%) |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 2 | 0.6 | 30918.312 | opt | <0.001 | 0.396 | 2004 | 0.000 | 0.000 |
| 10 | 3 | 0.6 | 22218.821 | opt | <0.001 | 0.422 | 2003 | 0.000 | 0.000 |
| 10 | 4 | 0.6 | 26122.856 | opt | <0.001 | 0.374 | 2001 | 0.000 | 0.000 |
| 10 | 5 | 0.6 | 25758.652 | opt | 0.001 | 0.3785 | 2005 | 0.000 | 0.000 |
| 15 | 2 | 0.6 | 33228.058 | opt | 0.006 | 0.4805 | 2033 | 0.000 | 0.000 |
| 15 | 3 | 0.6 | 34761.061 | opt | <0.001 | 0.478 | 2003 | 0.000 | 0.000 |
| 15 | 4 | 0.6 | 33129.275 | opt | 0.0025 | 0.468 | 2010 | 0.000 | 0.000 |
| 15 | 5 | 0.6 | 36192.384 | opt | 0.0055 | 0.4785 | 2031 | 0.000 | 0.000 |
| 20 | 2 | 0.6 | 86095.674 | opt | 0.0995 | 0.639 | 2364 | 0.000 | 0.000 |
| 20 | 3 | 0.6 | 64981.423 | opt | 0.0235 | 0.5655 | 2090 | 0.000 | 0.000 |
| 20 | 4 | 0.6 | 40920.641 | opt | <0.001 | 0.538 | 2009 | 0.000 | 0.000 |
| 20 | 5 | 0.6 | 40293.497 | opt | 0.059 | 0.6195 | 2199 | 0.000 | 0.000 |
| 25 | 2 | 0.6 | 71542.272 | opt | 0.3455 | 1.0255 | 3019 | 0.257 | 1.123 |
| 25 | 3 | 0.6 | 81936.291 | opt | 0.1335 | 0.814 | 2387 | 0.007 | 0.133 |
| 25 | 4 | 0.6 | 52059.420 | opt | 0.182 | 0.87 | 2522 | 0.003 | 0.061 |
| 25 | 5 | 0.6 | 51936.508 | opt | 0.0395 | 0.693 | 2122 | 0.000 | 0.000 |
| 30 | 2 | 0.6 | 212119.726 | 212354.2 | 0.7695 | 1.537 | 3710 | 1.403 | 4.552 |
| 30 | 3 | 0.6 | 76593.419 | opt | 0.363 | 1.161 | 2907 | 0.016 | 0.006 |
| 30 | 4 | 0.6 | 60721.997 | opt | 0.076 | 0.8515 | 2189 | 0.000 | 0.000 |
| 30 | 5 | 0.6 | 84895.245 | opt | 0.865 | 1.5595 | 3938 | 0.719 | 3.677 |
| 35 | 2 | 0.6 | 154200.141 | 158622.7 | 1.623 | 2.1715 | 4701 | 4.202 | 3.175 |
| 35 | 3 | 0.6 | 125140.117 | 126929.9 | 0.9825 | 1.807 | 3949 | 1.711 | 1.490 |
| 35 | 4 | 0.6 | 93518.430 | 93857.1 | 0.705 | 1.5345 | 3436 | 1.667 | 1.961 |
| 35 | 5 | 0.6 | 85915.122 | opt | 0.5495 | 1.4255 | 3200 | 0.178 | 0.895 |
| 40 | 2 | 0.6 | 132187.999 | 132959.2 | 1.4485 | 2.137 | 4368 | 1.041 | 1.126 |
| 40 | 3 | 0.6 | 75738.350 | 75915.98 | 1.3375 | 2.131 | 4240 | 1.232 | 3.837 |
| 40 | 4 | 0.6 | 74256.025 | opt | 1.226 | 1.966 | 4076 | 0.992 | 1.783 |
| 40 | 5 | 0.6 | 130371.240 | 132187 | 1.8105 | 2.5175 | 4733 | 2.768 | 2.769 |

TABLE 10. GA results on small instances

## 4. Conclusions

In this paper, we describe the GA metaheuristic for solving the MLWLP based on the binary encoding. New encoding scheme is used, which gives suitable representation of an individual. By arranging the cells in the array sorted in the increasing order of their distances for each item type, we direct the GA to promising search regions. Effective objective function is based on identifying the indices of the first non zero bits in each gene and searching through the sorted array of cells to find appropriate assignments. The implemented FGTS and one-point crossover operator shows to be

| It | L | $\alpha$ | *opt* | *GA* | $t$(sec) | $t_{tot}(sec)$ | *gen* | *agap*(%) | $\sigma$(%) |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 2 | 0.8 | 19224.563 | opt | <0.001 | 0.3855 | 2001 | 0.000 | 0.000 |
| 10 | 3 | 0.8 | 25336.800 | opt | <0.001 | 0.3745 | 2001 | 0.000 | 0.000 |
| 10 | 4 | 0.8 | 22321.650 | opt | <0.001 | 0.394 | 2001 | 0.000 | 0.000 |
| 10 | 5 | 0.8 | 23153.774 | opt | 0.0025 | 0.378 | 2015 | 0.000 | 0.000 |
| 15 | 2 | 0.8 | 38487.696 | opt | 0.0155 | 0.4445 | 2084 | 0.000 | 0.000 |
| 15 | 3 | 0.8 | 27029.930 | opt | 0.001 | 0.4505 | 2011 | 0.000 | 0.000 |
| 15 | 4 | 0.8 | 24769.594 | opt | 0.004 | 0.426 | 2025 | 0.000 | 0.000 |
| 15 | 5 | 0.8 | 24279.717 | opt | 0.005 | 0.437 | 2034 | 0.000 | 0.000 |
| 20 | 2 | 0.8 | 39261.904 | opt | 0.0045 | 0.5875 | 2019 | 0.000 | 0.000 |
| 20 | 3 | 0.8 | 59359.668 | opt | 0.017 | 0.5545 | 2060 | 0.000 | 0.000 |
| 20 | 4 | 0.8 | 38962.218 | opt | 0.0225 | 0.576 | 2084 | 0.000 | 0.000 |
| 20 | 5 | 0.8 | 37114.598 | opt | 0.0025 | 0.576 | 2010 | 0.000 | 0.000 |
| 25 | 2 | 0.8 | 79158.018 | 79369.41 | 0.187 | 0.799 | 2601 | 0.359 | 0.521 |
| 25 | 3 | 0.8 | 122439.500 | opt | 0.3675 | 1.038 | 3082 | 0.578 | 2.123 |
| 25 | 4 | 0.8 | 63208.601 | opt | 0.043 | 0.733 | 2121 | 0.000 | 0.000 |
| 25 | 5 | 0.8 | 39551.933 | opt | 0.0375 | 0.6205 | 2114 | 0.000 | 0.000 |
| 30 | 2 | 0.8 | 62563.537 | opt | 0.858 | 1.5105 | 4075 | 0.921 | 3.239 |
| 30 | 3 | 0.8 | 122539.976 | 122804.9 | 1.051 | 1.733 | 4334 | 0.310 | 0.868 |
| 30 | 4 | 0.8 | 74068.193 | opt | 0.6645 | 1.3 | 3810 | 0.488 | 3.984 |
| 30 | 5 | 0.8 | 74408.722 | opt | 0.5095 | 1.204 | 3358 | 0.092 | 0.595 |
| 35 | 2 | 0.8 | 85351.410 | 86283.69 | 1.351 | 1.943 | 4565 | 1.996 | 1.877 |
| 35 | 3 | 0.8 | 154474.559 | opt | 0.961 | 1.7745 | 3813 | 1.178 | 2.539 |
| 35 | 4 | 0.8 | 116061.308 | 116626.2 | 1.412 | 1.9895 | 4700 | 1.908 | 5.879 |
| 35 | 5 | 0.8 | 85853.758 | 85855.08 | 0.8025 | 1.5495 | 3705 | 0.793 | 1.666 |
| 40 | 2 | 0.8 | 338547.775 | 349660.2 | 2.076 | 2.529 | 4846 | 4.815 | 4.284 |
| 40 | 3 | 0.8 | 144358.864 | 148026.2 | 1.815 | 2.416 | 4708 | 3.274 | 2.022 |
| 40 | 4 | 0.8 | 196334.483 | 197342.4 | 1.183 | 1.9495 | 4192 | 0.690 | 0.601 |
| 40 | 5 | 0.8 | 92246.284 | 92311.57 | 1.2095 | 1.9855 | 4080 | 0.738 | 3.102 |

TABLE 11. GA results on small instances

appropriate in the proposed GA concept. The idea of mutation with frozen bits and several other strategies are used to help in increasing the diversity of genetic material and avoiding premature convergence.

According to computational results on small and large-scale test instances, the applied GA approach proves to be successful. The achievement of the optimal solution for 112 of total of 140 small instances and rather small average gaps for others, indicate that the GA approach can be reliably used for solving the MLWLP. Considering the fact that there are no common test instances available, rough comparisons with other methods are carried out, showing that the proposed GA gives more optimal

| It | L | $\alpha$ | $GA$ | $t$(sec) | $t_{tot}(sec)$ | $gen$ | $agap$(%) | $\sigma$(%) |
|---|---|---|---|---|---|---|---|---|
| 100 | 2 | 0.5 | 886029.67 | 5.779 | 7.310 | 4764 | 1.017 | 2.200 |
| 100 | 3 | 0.5 | 599825.17 | 4.231 | 6.531 | 4525 | 1.806 | 2.882 |
| 100 | 4 | 0.5 | 545607.15 | 5.191 | 6.743 | 4885 | 0.996 | 2.503 |
| 100 | 5 | 0.5 | 531832.02 | 5.883 | 6.992 | 4903 | 0.896 | 2.281 |
| 150 | 2 | 0.5 | 2497848.77 | 9.472 | 11.764 | 4691 | 1.070 | 2.120 |
| 150 | 3 | 0.5 | 1736344.65 | 8.579 | 11.450 | 4710 | 0.863 | 1.630 |
| 150 | 4 | 0.5 | 1627720.06 | 9.120 | 11.328 | 4818 | 0.665 | 1.297 |
| 150 | 5 | 0.5 | 958259.04 | 7.966 | 10.897 | 4779 | 0.885 | 2.147 |
| 200 | 2 | 0.5 | 3407078.25 | 14.719 | 18.548 | 4859 | 0.819 | 1.678 |
| 200 | 3 | 0.5 | 3044631.21 | 13.458 | 17.047 | 4758 | 0.338 | 0.929 |
| 200 | 4 | 0.5 | 1736508.16 | 11.262 | 16.341 | 4473 | 0.489 | 1.260 |
| 200 | 5 | 0.5 | 1649873.13 | 14.303 | 16.831 | 4884 | 0.979 | 2.115 |
| 250 | 2 | 0.5 | 8007271.66 | 19.032 | 24.474 | 4808 | 0.369 | 0.835 |
| 250 | 3 | 0.5 | 3561326.29 | 19.530 | 25.112 | 4885 | 0.828 | 1.377 |
| 250 | 4 | 0.5 | 4182619.77 | 17.344 | 22.128 | 4674 | 0.667 | 1.408 |
| 250 | 5 | 0.5 | 2626602.19 | 17.685 | 22.626 | 4819 | 0.443 | 0.933 |
| 300 | 2 | 0.5 | 13488441.57 | 23.381 | 31.022 | 4710 | 0.396 | 0.811 |
| 300 | 3 | 0.5 | 5687478.20 | 20.840 | 29.413 | 4393 | 0.335 | 0.729 |
| 300 | 4 | 0.5 | 3879476.68 | 20.975 | 31.050 | 4555 | 0.652 | 1.200 |
| 300 | 5 | 0.5 | 5080521.97 | 21.912 | 29.009 | 4692 | 0.396 | 0.983 |
| 350 | 2 | 0.5 | 11303227.28 | 33.169 | 42.345 | 4850 | 0.345 | 0.643 |
| 350 | 3 | 0.5 | 7092958.06 | 26.606 | 38.839 | 4471 | 0.344 | 0.785 |
| 350 | 4 | 0.5 | 4571192.03 | 26.144 | 37.935 | 4614 | 0.629 | 0.660 |
| 350 | 5 | 0.5 | 5908626.74 | 26.579 | 35.847 | 4683 | 0.666 | 1.085 |
| 400 | 2 | 0.5 | 12951254.84 | 35.930 | 51.168 | 4687 | 0.285 | 0.373 |
| 400 | 3 | 0.5 | 14815517.70 | 36.696 | 47.462 | 4802 | 0.258 | 0.534 |
| 400 | 4 | 0.5 | 7055304.14 | 31.713 | 45.953 | 4453 | 0.268 | 0.702 |
| 400 | 5 | 0.5 | 4741152.81 | 32.813 | 47.175 | 4613 | 0.463 | 0.730 |

TABLE 12. GA results on large instances

solutions for smaller instances and seems to work better on large-scale instances. In cases of large-scale problem instances, when CPLEX solver can not provide optimal solutions due to memory limits, the GA quickly finds solutions. This indicates that our approach can be applied on real-life situations when exact methods can not be used.

The GA implementation described in this paper can be extended in several ways. It would be interesting to compare obtained results with other metaheuristics on the same instances and to hybridize the GA with other exact or heuristic approaches.

| It | L | $\alpha$ | $GA$ | $t$(sec) | $t_{tot}(sec)$ | $gen$ | $agap$(%) | $\sigma$(%) |
|---|---|---|---|---|---|---|---|---|
| 100 | 2 | 0.8 | 2121400.17 | 4.444 | 6.638 | 4525 | 1.140 | 2.752 |
| 100 | 3 | 0.8 | 635835.65 | 4.835 | 6.859 | 4563 | 2.176 | 3.839 |
| 100 | 4 | 0.8 | 360014.75 | 4.411 | 6.396 | 4420 | 0.572 | 1.660 |
| 100 | 5 | 0.8 | 627278.24 | 5.440 | 7.132 | 4765 | 1.028 | 2.208 |
| 150 | 2 | 0.8 | 4324473.12 | 8.553 | 11.653 | 4596 | 0.607 | 1.577 |
| 150 | 3 | 0.8 | 2330456.23 | 8.494 | 11.663 | 4620 | 0.641 | 1.340 |
| 150 | 4 | 0.8 | 1803497.57 | 8.070 | 11.292 | 4600 | 1.745 | 2.881 |
| 150 | 5 | 0.8 | 1495425.95 | 6.146 | 10.028 | 4325 | 1.513 | 2.535 |
| 200 | 2 | 0.8 | 6363820.94 | 14.059 | 17.820 | 4642 | 0.640 | 0.959 |
| 200 | 3 | 0.8 | 3253969.22 | 13.344 | 17.788 | 4673 | 0.449 | 1.044 |
| 200 | 4 | 0.8 | 3471624.50 | 11.544 | 16.360 | 4537 | 0.665 | 1.512 |
| 200 | 5 | 0.8 | 3797900.27 | 13.496 | 17.259 | 4905 | 0.730 | 1.713 |
| 250 | 2 | 0.8 | 12477844.33 | 20.883 | 25.663 | 4992 | 0.454 | 0.943 |
| 250 | 3 | 0.8 | 6077753.05 | 17.956 | 24.764 | 4894 | 0.880 | 1.607 |
| 250 | 4 | 0.8 | 4191459.92 | 16.996 | 23.675 | 4661 | 0.484 | 1.097 |
| 250 | 5 | 0.8 | 4494294.91 | 17.568 | 23.261 | 4874 | 0.854 | 1.395 |
| 300 | 2 | 0.8 | 23368751.78 | 23.695 | 31.974 | 4737 | 0.271 | 0.744 |
| 300 | 3 | 0.8 | 9532127.78 | 23.421 | 30.853 | 4648 | 0.720 | 1.042 |
| 300 | 4 | 0.8 | 7966096.83 | 23.597 | 30.977 | 4730 | 0.462 | 1.032 |
| 300 | 5 | 0.8 | 6924902.50 | 23.047 | 28.765 | 4693 | 0.638 | 1.188 |
| 350 | 2 | 0.8 | 17019324.08 | 28.491 | 41.080 | 4619 | 0.529 | 0.678 |
| 350 | 3 | 0.8 | 13459788.95 | 29.341 | 38.953 | 4743 | 0.318 | 0.913 |
| 350 | 4 | 0.8 | 9987551.66 | 28.352 | 37.978 | 4594 | 0.413 | 0.835 |
| 350 | 5 | 0.8 | 9232550.61 | 25.990 | 36.754 | 4630 | 0.724 | 0.990 |
| 400 | 2 | 0.8 | 30567646.34 | 41.026 | 51.654 | 4836 | 0.414 | 0.858 |
| 400 | 3 | 0.8 | 20004110.91 | 39.730 | 49.479 | 4891 | 0.203 | 0.621 |
| 400 | 4 | 0.8 | 11911918.60 | 33.581 | 44.790 | 4500 | 0.584 | 0.844 |
| 400 | 5 | 0.8 | 10116167.30 | 28.413 | 42.687 | 4375 | 0.370 | 0.796 |

TABLE 13. GA results on large instances

**Acknowledgement:** This research was partially supported by Serbian Ministry of Education and Science, under the grants no. 174010 and 47017.

## REFERENCES

[1] R. V. Johnson, *SPACECRAFT for multi-floor layout planning*, Management Sciences, **28 (4)** (1982), 407–417.

[2] R. L. Francis, L. F. McGinnis, Jr, and J. A. White, *Facility Layout and Location: An Analytical Approach*, Prentice-Hall, Inc., Englewood Cliffs, N.J. (1992).

[3] G. Q. Zhang, J. Xue, and K. K. Lai, *A class of genetic algorithms for multiple-level warehouse layout problems*, International Journal of Production Research, **40 (3)** (2002), 731–744.

[4] G. Q. Zhang and K. K. Lai, *A Combining path relinking and genetic algorithms for the multiple-level warehouse layout problem*, European Journal of Operational Researh, **169 (2)** (2006), 413–425.
[5] K. K. Lai, J. Xue, and G. Zhang, *Layout design for a paper reel warehouse: A two-stage heuristic approach*, International Journal of Production Economics, **75 (3)** (2002), 231-243.
[6] G. Q. Zhang, K. K. Lai, *Tabu search approach for multi-level warehouse layout problem with adjacent constraints*, Engineering Optimization, **42 (6)** (2010), 775–790.
[7] L. Yang, and Y. Feng, *Fuzzy multi-level warehouse layout problem: New model and algorithm*, Journal of Systems Science and Systems Engineering, **15 (4)** (2006), 493–503.
[8] S. Önüt, U. R. Tuzkaya and B. Doga, *A particle swarm optimization algorithm for the multiple-level warehouse layout design problem*, Computers and Industrial Engineering, **54 (4)** (2008), 783–799.
[9] M. A. Shouman, M. Khater and A. A. Boushaala, *Comprehensive survey and classification scheme of warehousing systems*, Proceedings of the 2005 International Conference on Simulation and Modeling V. Kachitvichyanukul, U. Purintrapiban, P. Utayopas, eds., Nakompathon, Thailand, 2005.
[10] Z. Stanimirovic, *Genetic algorithms for solving some NP-hard hub location problems*, (in Serbian), PhD. thesis, University of Belgrade, Faculty of Mathematics, 2007.
[11] V. Filipovic, *Selection and migration operators and Web services in parallel evolutionary algorithms* (in Serbian). PhD thesis, University of Belgrade, Faculty of Mathematics, 2006.
[12] J. Kratica, V. Kovacevic-Vujcic and M. Cangalovic, *Computing strong metric dimension of some special classes of graphs by genetic algorithms*, Yugoslav Journal of Operations Research, **18 (2)** (2008), 143–151.
[13] J. Kratica, M. Cangalovic and V. Kovacevic-Vujcic, *Computing minimal doubly resolving sets of graphs*, Computers & Operations Research, **36 (7)** (2009), 2149–2159.
[14] J. Kratica, V. Kovacevic-Vujcic and M. Cangalovic, *Computing the metric dimension of graphs by genetic algorithms*, Computational Optimization and Applications, **44 (2)** (2009), 343–361.
[15] J. Kratica, *Parallelization of genetic algorithms for solving some NP-complete problems* (in Serbian), PhD thesis, Faculty of Mathematics, University of Belgrade, 2000.
[16] V. Filipovic, *Fine-grained tournament selection operator in genetic algorithms*, Computing and Informatics, **22 (2)** (2003), 143-161.
[17] J. Kratica, Z. Stanimirovic, D. Tosic and V. Filipovic, *Two Genetic Algorithms for Solving the Uncapacitated Single Allocation p-Hub Median Problem*, European Journal of Operational Research, **182 (1)** (2007), 15–28.

[1] Department of Mathematics and Informatics, Faculty of Science,
University of Banja Luka,
Bosnia and Herzegovina
*E-mail address*: matic.dragan@gmail.com

[2,3,4] Faculty of Mathematics,
University of Belgrade,
Serbia
*E-mail address*: vladofilipovic@hotmail.com
*E-mail address*: aleks3rd@gmail.com
*E-mail address*: zoricast@matf.bg.ac.rs