Yugoslav Journal of Operations Research
21 (2011), Number 2, 225-238
DOI: 10.2298/YJOR1102225K

A NEW GENETIC REPRESENTATION FOR QUADRATIC ASSIGNMENT PROBLEM[1]

Jozef KRATICA
Mathematical Institute, Serbian Academy of Sciences and Arts,
Belgrade, Serbia
jkratica@mi.sanu.ac.rs
Dušan TOŠIĆ, Vladimir FILIPOVIĆ, Đorđe DUGOŠIJA
Faculty of Mathematics, University of Belgrade,
Belgrade, Serbia
{*dtosic* | *vladaf* | dugosija}@matf.bg.ac.rs

Received: October 2009 / Accepted: November 2011

Abstract: In this paper, we propose a new genetic encoding for well known Quadratic Assignment Problem (QAP). The new encoding schemes are implemented with appropriate objective function and modified genetic operators. The numerical experiments were carried out on the standard QAPLIB data sets known from the literature. The presented results show that in all cases proposed genetic algorithm reached known optimal solutions in reasonable time.

Keywords: Genetic algorithm, evolutionary computation, combinatorial optimization, quadratic assignment problem.

MSC: 90C59, 68T20, 90e20.

[1] This research was partially supported by Serbian Ministry of Education and Science under the grant no. 174010

1. INTRODUCTION

1.1. Quadratic assignment problem

The Quadratic Assignment Problem (QAP) is firstly proposed in [16] as a mathematical model related to economic activities. Since then, it has appeared in many practical applications as can be seen from [22]. We mention only several recent applications:

- facility layout design problem in order to minimize work-in-process [29];
- website structure improvement [27];
- placement of electronic components [9];
- index assignment problem related to error control in communications [3];
- memory layout optimization in signal processors [34].

Several NP-hard combinatorial optimization problems, such as the traveling salesman problem, the bin-packing problem and the max clique problem, also can be modeled as QAPs.

Since its first formulation, the QAP has been drawing researchers' attention worldwide, not only because of its practical and theoretical importance, but also because of its complexity. The QAP is one of the most difficult combinatorial optimization problems. In [28] was shown that QAP is NP-hard and that, unless P = NP, it is not possible to find an $1+\varepsilon$ - approximation algorithm, for a constant ε. Such results are valid even when flows and distances appear as symmetric coefficient matrices.

In general, the QAP instances of size greater than 30 cannot be solved exactly in a reasonable time. Although heuristic methods do not offer a guarantee for reaching the optimum, they give satisfactory results to a large range of various problems in a reasonable amount of time. Recently, so-called metaheuristics, or general frameworks for building heuristics, became popular for solving difficult combinatorial optimization problems. Metaheuristic approaches use different techniques in order to avoid entrapments in pour local minima and are based mainly on two principles: local search with globalization mechanisms and population search.

In local search methods, an intensive search of the solution space is performed by moving, at each step, from the current solution to another promising solution in its neighborhood. Globalization mechanisms are designed so to ensure diversification of the search. The population search consists of maintaining a pool of good solutions and combining them in order to produce hopefully better solutions.

Thus, a large number of metaheuristic methods have been used to solve the QAP and presentation of all such contributions is out of this paper's scope. We mention only several recent metaheuristic applications for QAP:

- genetic algorithms [6, 10, 33];
- tabu search [7, 14, 24];
- simulated annealing [23];
- ant colony optimization [27];
- particle swarm optimization [20];

- iterated local search [30]
- self-organizing migrating algorithm [4].

As it can be seen from the literature ([22]), hybrid approaches for solving QAP have some advantages compared to single metaheuristic approaches. Some of the recent hybrid approaches are:

- hybrid of genetic algorithm and several variants of tabu search [8];
- variable neighborhood particle swarm optimization [21];
- ant colony optimization approach coupled with a guided local search [12];
- ant colony optimization hybridized with the genetic algorithm and a local search method [32];
- GRASP with path-relinking [26].

1.2 Genetic algorithms

Genetic algorithms (GAs) represent a problem-solving metaheuristic method rooted in the mechanisms of evolution and natural genetics. The main idea was introduced by Holland [13]. In the last three decades GAs have emerged as effective, robust optimization and search methods.

GAs solve problems by creating a population of individuals (usually 10 - 200), represented by chromosomes, which are encoded solutions of the problem. The representation is the genetic code of an individual, and it is often a binary string, although other alphabets of higher cardinality can be used. A chromosome is composed of basic units named genes, which control the features of an individual. To each chromosome, a fitness value measuring its success is assigned. The initial population (the first generation of individuals) is usually randomly initialized. The individuals in the population then pass through a procedure of simulated "evolution" by means of randomized processes of selection, crossover, and mutation.

The selection operator favors individuals more capable to survive through the generations. The probability that a chromosome will be chosen depends on its fitness. The higher fitness value of a chromosome provides higher chances for its survival and reproduction. There are different ways of selecting the best-fitted individuals. One of the most often used is tournament selection (for more details see [1, 11, 25]). Crossover and mutation operators are also used during reproduction. The crossover operator provides a recombination of genetic material by exchanging portions between the parents with the chance that good solutions can generate even better ones.

Mutation causes sporadic and random changes by modifying individual's genetic material with some small probability. Its role is to regenerate the lost or unexplored genetic material into the population. Mutation has a direct analogy with nature, and it should prevent premature convergence of the GA to suboptimal solutions.

There are many different policies for generation replacement. Certain numbers of individuals (elite individuals) may skip selection (or even all genetic operators) going directly into the next generation. This approach is named the steady-state generation replacement policy with elitist strategy. It provides a smaller gradient in the genetic search, but preserves fitted individuals from the past generations.

There can be many modifications of the GA, but implementing the GA usually involves the following steps:

- evaluating the fitness of all individuals in a population;

- selecting the best-fitted individuals;
- creating a new population by performing crossover and mutation operators.

The process of reproduction and population replacement is repeated until a stopping criterion (fixed number of generations or satisfied quality of solutions obtained) is met. Detailed description of GAs is out of this paper's scope, and it can be found in [1,25].

GAs have a wide range of applications, growing rapidly, for example, from strong metric dimensions of graphs [19], through maximally balanced connected partition [5], spanning sets coverage [15], generalized Euclidean distances [2] to hub location [18] and modeling of chemical processes [31]. As it can be seen in previous section, GAs are frequently used for solving QAP in stand-alone or hybrid approaches ([6, 8, 10, 32]).

2. MATHEMATICAL FORMULATION OF THE QAP

The QAP can be described as the problem of assigning n facilities to n locations with given distances between the locations and given flows between the facilities. The goal is to place the facilities on locations in such a way that the sum of the product between flows and distances is minimized.

The QAP can be formulated in different ways. One of the most popular formulations is a permutation based formulation. Let S_n be the set of all permutations with n elements and $\pi \in S_n$. Consider f_{ij} the flows between facilities i and j and d_{kl} the distances between locations k and l. If each permutation π represents an allocation of facilities to locations, the problem expression becomes:

$$\min_{\pi \in S_n} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \cdot d_{\pi(i)\pi(j)} \tag{1}$$

Another equivalent formulation is quadratic integer programming formulation. If we define binary variables $x_{ik} = \begin{cases} 1, & k = \pi(i) \\ 0, & k \neq \pi(i) \end{cases}$ QAP can be formulated as:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} f_{ij} \cdot d_{kl} \cdot x_{ik} \cdot x_{jl} \tag{2}$$

s.t.

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, 2, ..., n \tag{3}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad j = 1, 2, ..., n \tag{4}$$

$$x_{ij} \in \{0,1\} \quad i, j = 1, 2, ..., n \tag{5}$$

In the literature there exist numerous formulations and presentation of the QAP but all of them are out of this paper's scope. Interested reader can seek for more information about different formulations of the QAP including the linear formulations in [22].

3. NEW GENETIC REPRESENTATION FOR THE QAP

3.1. Representation and objective function

Since the QAP is a minimization problem, it is obvious that, in the optimal permutation (solution) pairs of facilities with large flow usually corresponds to the pairs of locations with small distance between them. We introduce a new encoding scheme which forces frequent occurrence of this behavior, i.e. there are very small chances that the pair of facilities with the large flow corresponds to the pairs of locations with large distance. On that way, this encoding scheme push GA search towards promising search regions.

In this encoding scheme every individual consists of $n-1$ genes. Length of the individual is $n-1$ because it is not necessary to remember the last element in permutation when all other elements are set. Each gene is represented by an integer that corresponds to one element in permutation. In contrast to previous representations, that particular integer, let us say the i-th integer in permutation, which represents the gene, is not the index of a location assigned to facility i. Instead, it represents a "distance" of the corresponding partial assignment of locations to facilities $1,2, \ldots, i-1$ from a locally optimal solution of QAP restricted to these facilities. Therefore, the value of i-th gene belongs to $\{0,1,\ldots, n-i\}$.

For a given coded individual the corresponding permutation of locations assigned to facilities is obtained by an iterative procedure expressed by a pseudo-code in Figure 1. In each iteration of the procedure, for every non-assigned location its weight is calculated as the partial sum of the products between flows and distances from this location to all previously assigned locations. Then non-assigned locations are sorted in a sequence according to non-decreasing weights. The location from the sequence for which "distance" from its first member is equal to the value of the corresponding gene, is chosen as the next location in the permutation. In this way locations with lower weights are associated to facilities with smaller gene values.

```
S = []; gv := Take_Gene(1); Pi[1]:=gv+1;
for i:=2 to n-1 do begin
  gv := Take_Gene(i);
  for j:=1 to n do
    if not (j in S) then begin
      w[j] = F(i,i) * D(j,j);
      for k:=1 to i-1 do begin
        w[j] := w[j] + F(k,i) * D(Pi[k],j);
        w[j] := w[j] + F(i,k) * D(j,Pi[k]);
      endfor
    endif
  endfor
  Quick_Sort(n-i+1, w, w_index);
  Pi[i] := w_index[gv+1];
  S := S ∪ {Pi[i]};
endfor
Pi[n] := {1,2,..,n} \ S;
```

Figure 1: Pseudo-code for objective function.

Set `S` represents a set of currently assigned locations, `Pi` is the corresponding permutation of locations, array `w` stores calculated current weights, while `w_index` denotes indices of `w` arranged according to non-decreasing weights by function `Quick_Sort()`. Function `Take_Gene(i)` returns the value of *i*-th gene.

Note that the previous procedure for calculating objective function gives the permutation so that the individuals are always feasible. In other words, if the initial order of locations *1,2,...,n* is respected whenever weights of some locations in sorted sequence are equal, the new encoding scheme is unique and complete, i.e. for each permutation of locations there exists a unique corresponding code and there are no two permutations with the same code.

Example 1. Suppose that GA is solving QAP with n=4 and with following flow and distance matrix:

$$F = \begin{bmatrix} 0 & 3 & 0 & 2 \\ 2 & 0 & 0 & 7 \\ 3 & 4 & 0 & 0 \\ 5 & 2 & 5 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix}$$

In this case, each individual in the population has genetic representation with length *3*. Suppose that the individual has following representation:

2	0	1

Algorithm for calculating permutation represented by this individual is:

- At the beginning, `Pi[1]=gv+1=3`, i.e. location `3` is assigned to facility `1`.

- In the next step, where i=2, non-assigned locations are {1,2,4}. For each of these locations, we are calculated their weights `w[1]=10, w[2]=w[4]=5`, so the sorted array `w_index` is 2,4,1. Since gen value `gv=0` then `Pi[2] = w_index[1] = 2`.
- The same procedure is applied in the third step, so `Pi[3] = w_index[2]=1`.
- In the last step `Pi[4]=4`, which is only remaining location.

3.2. Improving heuristic

In order to improve individuals, we performed a local search on proposed genetic algorithms. The best results are obtained by applying best-known 2-opt heuristic with first improvement. This procedure is repeated until we are sure that swapping of each pair of the elements in permutation will not improve the quality of the permutation. The other strategy is to make only one local search for improving quality of obtained permutation. That approach is faster, but quality of obtained results is not as good as it is in the first case.

Local search procedure like 2-opt usually significantly decreases diversity of the GA population. In case when the solution is improved by heuristic corresponding GA code is not changed, so the diversity of the GA population is preserved. Therefore, 2-opt heuristic deals directly with permutations of locations and not with the new encoding, which imply that the algorithm for obtaining the genetic code from the improved solution is not needed at all.

3.3. Population initialization

The initial population of $N_{pop} = 150$ individuals is randomly generated, allowing maximal diversity of genetic material, but initialization of the genetic code of all individuals in first generation should not be a pure random procedure. Natural model for this behavior is finite decreasing geometric progression with common ratio q.

Probability p_k that the i-th gene has value $k (k \in \{0,1,...,n-i\})$, decreases with increasing of k according to the geometric progression with a given ratio $q \in (0,1)$, i.e.

$$p_k = p_{0\,p^k} \text{ . As } \sum_{k=0}^{n-i} p_k = p_0 \frac{1-q^{n-i+1}}{1-q} = 1 \text{ , then } p_0 = \frac{1-q}{1-q^{n-i+1}} .$$

From the fixed value q, it is easy to calculate all probabilities p_k, k=0,1,...,n-i and to generate i-th gene randomly with these probabilities. In algorithms that we propose, the value of q is equal to 0.5.

4. GENETIC OPERATORS AND OTHER GA ASPECTS

4.1. Fitness function and generation replacement policy

The number of elitist individuals passed directly to the next generation is $N_{elite} = 100$. Non-elitist individuals N_{nonel} (the rest of the population) go through genetic

operators. This means that a lot of time is saved since the objective value is calculated only once for each elite individual.

To prevent undeserved domination of elite individuals over the population, their fitness is decreased by the next formula:

$$Ft_i^{new} = \begin{cases} Ft_i - \overline{Ft}, & Ft_i > \overline{Ft} \\ 0, & Ft_i \le \overline{Ft} \end{cases}, \quad 1 \le i \le N_{elite}, \quad \overline{Ft} = \frac{1}{N_{pop}} \sum_{i=1}^{N_{pop}} Ft_i \tag{6}$$

In this way, even non-elite individuals preserve their chance to survive to the next generation.

As non-elitist individuals go through genetic operators, appearance of duplicated individuals is possible. Such individuals with the same genetic code are discarded - simply by setting fitness value of the duplicate to zero, so that selection operator allows them not to continue to the next generation. Furthermore, too many individuals with the same objective function, but different genetic codes, may predominate in population. This is why limiting the number of such individuals in population to some constant has been shown useful in [19, 5, 18]. Therefore, proposed algorithms prohibit an existence of more than 40 elite individuals with different genetic code and the same objection value, which prevent premature convergence of algorithms and increase diversity of genetic material.

4.2. Selection and crossover

A fine-grained tournament selection (FGTS) scheme has been used in the proposed GAs for deciding which individuals will produce the next generation. The average size of tournament, F_{tour}, is a real number, and is considered to be a constant in practice. We used the value of F_{tour}=5.4, because it gave good results in solving similar problems (for example, see [5,18,19]). Detailed information on FGTS scheme can be found in [11].

For recombination of individuals, we used the classical one-point crossover. The crossover rate is p_{cross}=0.85. This means that about 85% individuals participate in recombination of their genes.

4.3. Mutation

Finally, modified simple mutation operator changes randomly selected genes. During the GA execution, it is possible that all individuals in the population have the same gene in a certain position. These genes are called frozen. If the number of frozen genes is significant, the search space becomes much smaller, and the possibility of premature convergence rapidly increases. For that reason, the basic mutation rates are increased, but only for the frozen genes. The basic mutation rates are:

- $0.1/n$ for the bit on the first position.
- $0.05/n$ for the bit on the second position. Next bits in the gene have repeatedly two times smaller mutation rate ($0.025/n$, $0.0125/n$...).

When compared with the basic mutation rates, frozen bits are mutated by 2.5 times higher rate:

- $0.25/n$ instead of $0.1/n$ if they are at the first position of the gene.
- $0.125/n$ for the bit on the second position. Next bits in the gene have repeatedly two times smaller mutation rate ($0.0625/n$, $0.03125/n$...).

4.4. Caching GA

Caching optimizes run-time of a genetic algorithm. The evaluated objective values are stored in a hash-queue structure using the Least Recently Used caching technique (LRU). Otherwise it would be necessary to calculate the same objective value each time genetic operators produce another individual with the same genetic code. With caching technique, when such individual appears, its objective value is taken from the caching table, and this saves a significant amount of time. Cashing of GAs has no impact on results that are obtained by GAs - it only reduces execution time.

In proposed GA implementations, we limited the number of individuals stored in a caching table to N_{lcache}=5000. Detailed information about caching GA can be found in [17].

Table 1: GA results on QAP instances

Instance name	Opt	*t* (sec)	*gen*	*agap* (%)	σ (%)	*eval*	*cache* (%)
bur26a	5426670	77.422	2014	0.000	0.000	61114	60.6
bur26b	3817852	92.447	2020	0.000	0.000	63242	62.5
bur26c	5426795	81.654	2001	0.000	0.000	60001	59.9
bur26d	3821225	111.720	2299	0.000	0.000	71577	62.0
bur26e	5386879	108.612	2457	0.000	0.002	77288	62.5
bur26f	3782044	113.981	2001	0.000	0.000	58122	58.0
bur26g	10117172	84.253	2092	0.000	0.000	67248	64.2
bur26h	7098658	95.664	2178	0.000	0.000	67481	62.0
chr12a	9552	2.071	2001	0.000	0.000	73872	73.7
chr12b	9742	3.154	2001	0.000	0.000	68129	68.0
chr12c	11156	1.730	2014	0.000	0.000	77339	76.7
chr15a	9896	4.780	2262	0.061	0.148	86105	75.9
chr15b	7990	4.645	2001	0.000	0.000	73500	73.4
chr15c	9504	3.745	2274	1.306	2.229	88766	77.9
chr18a	11098	8.678	2261	0.299	1.026	84592	74.6
chr18b	1534	7.461	2001	0.000	0.000	67921	67.8
chr20a	2192	12.140	2265	0.771	1.377	82531	72.5
chr20b	2298	9.828	2777	5.013	1.725	102941	74.3
chr20c	14142	25.052	2356	0.472	1.454	82246	69.7
chr22a	6156	18.720	2465	0.369	0.291	93180	75.3
chr22b	6194	15.258	2914	1.088	0.481	110681	75.6
chr25a	3796	48.375	2946	2.903	1.719	107506	73.0
els19	17212548	25.806	2194	0.177	0.364	74309	67.4
esc16a	68	3.353	2001	0.000	0.000	62204	62.1
esc16b	292	0.063	25	0.000	0.000	397	17.1
esc16c	160	4.802	2001	0.000	0.000	58745	58.6
esc16d	16	3.679	2001	0.000	0.000	59255	59.1
esc16e	28	2.729	2001	0.000	0.000	60625	60.5
esc16f	0	0.002	1	0.000	0.000	0	0.0
esc16g	26	2.994	2001	0.000	0.000	59974	59.9
esc16h	996	0.008	1	0.000	0.000	0	0.0
esc16i	14	0.492	315	0.000	0.000	8038	43.3
esc16j	8	2.716	2001	0.000	0.000	60994	60.9
esc32e	2	0.050	1	0.000	0.000	0	0.0
esc32f	2	0.050	1	0.000	0.000	0	0.0
had12	1652	2.493	2001	0.000	0.000	77814	77.7
had14	2724	7.797	2001	0.000	0.000	67434	67.3
had16	3720	16.534	2001	0.000	0.000	61102	61.0
had18	5358	22.627	2001	0.000	0.000	57944	57.8
had20	6922	38.445	2001	0.000	0.000	57925	57.8

5. COMPUTATIONAL RESULTS

The tests were made on an Intel 2.5 GHz with 1GB memory, under Windows XP operating system. The algorithm was coded in C programming language. We tested our algorithm on QAPLIB instances (http://www.seas.upenn.edu/qaplib/) with known optimal solutions. The stopping criterion of GA was the maximum number of generations equal to 5000 or at most 2000 generations without the improvement of the objective value.

The GA was run 20 times for each instance, and the results are summarized in Table 1 and Table 2. For all the instances the algorithm reached the optimal solution. The tables are organized as follows:

- the first column contains the test instance's name;
- the second column contains the optimal solution for the particular instance;
- the average total running time (*t*) and the average number of generations for finishing GA (*gen*) are given in the third and the fourth columns;

  the fifth and the sixth column (*agap* and σ) contain information on the average solution quality: agap is a percentage gap defined as $agap = \frac{1}{20}\sum_{i=1}^{20} gap_i$, where $gap_i = 100 \cdot \frac{GA_i - opt}{opt}$ and GA_i represents the GA solution obtained in the *i*-th run, while σ is the standard deviation of gap_i, i=1,2,...,20, obtained by formula $\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(gap_i - agap)^2}$;

- in the last two columns *eval* represents the average number of the objective function evaluations, while *the* cache displays savings (in percent) achieved by using the caching technique.

Table 2: GA results on QAP instances

Instance name	Opt	*t* (sec)	*gen*	*agap* (%)	σ (%)	*eval*	*cache* (%)
kra30a	88900	117.353	2078	0.067	0.302	69757	67.0
kra30b	91420	132.594	2269	0.009	0.029	77643	68.3
lipa20a	3683	11.044	2011	0.000	0.000	70296	69.8
lipa20b	27076	18.240	2001	0.000	0.000	55735	55.6
lipa30a	13178	57.660	2173	0.000	0.000	76183	70.0
lipa30b	151426	70.761	2001	0.000	0.000	62397	62.3
nug12	578	1.589	2001	0.000	0.000	79805	79.6
nug14	1014	3.341	2001	0.000	0.000	74519	74.4
nug15	1150	5.779	2001	0.000	0.000	68544	68.4
nug16a	1610	6.454	2013	0.000	0.000	71358	70.8
nug16b	1240	9.817	2001	0.000	0.000	61327	61.2
nug17	1732	9.315	2266	0.046	0.058	80968	71.2
nug18	1930	10.218	2001	0.000	0.000	71870	71.7
nug20	2570	20.740	2005	0.000	0.000	66932	66.6
nug21	2438	27.497	2064	0.000	0.000	69749	67.3
nug22	3596	35.819	2006	0.000	0.000	70199	69.9
nug24	3488	48.132	2011	0.000	0.000	68349	67.8
nug25	3744	65.388	2043	0.003	0.012	71173	69.5
nug27	5234	57.737	2186	0.002	0.009	76857	70.0
nug28	5166	71.577	2677	0.083	0.128	95321	71.1
nug30	6124	160.993	2525	0.090	0.080	88657	69.9
rou12	235528	1.376	2004	0.000	0.000	77921	77.6
rou15	354210	3.582	2010	0.000	0.000	74054	73.6
rou20	725522	12.864	2512	0.069	0.087	91333	72.6
scr12	31410	1.573	2001	0.000	0.000	70681	70.5
scr15	51140	4.467	2001	0.000	0.000	68398	68.3
scr20	110030	11.825	2187	0.000	0.000	79151	72.3
ste36a	9526	484.717	2943	0.477	0.291	97933	66.3
ste36b	15852	418.567	2206	0.038	0.092	72428	65.4
ste36c	8239110	509.277	2941	0.181	0.122	99200	67.3
tai10a	135028	0.917	2001	0.000	0.000	76443	76.3
tai10b	1183760	1.103	2001	0.000	0.000	71168	71.0
tai12a	224416	1.871	2001	0.000	0.000	72101	72.0
tai12b	39464925	2.713	2002	0.000	0.000	73265	73.1
tai15a	388214	3.178	2001	0.000	0.000	75096	74.9
tai15b	51765268	7.523	2001	0.000	0.000	67744	67.6
tai17a	491812	6.716	2453	0.215	0.203	89057	72.5
tai20a	703482	10.470	2307	0.371	0.221	83099	71.9
tai20b	122455319	43.257	2001	0.000	0.000	65802	65.7
tai25b	344355646	84.672	2081	0.000	0.000	72678	69.6
tho30	149936	171.737	2763	0.099	0.135	97441	70.1

Our approach reached optimal solutions within reasonable running time. If we compare these running time with the running time of other existing GA for QAP [6,10,33], applied to the same set of instances, we can see that our algorithm seems to perform slower. However, our main goals here are to develop a GA approach with completely new encoding and to generate high-quality solutions.

6. CONCLUSIONS

This paper is devoted to exploring the results of the new genetic encoding scheme to the quadratic assignment problem. Arranging possible locations of every facility in non-decreasing order of their weights directs GA to promising search regions. The proposed encoding scheme is performed with adequate objective function and appropriate modified genetic operators. The performance of the genetic algorithm is improved with a local search, the mutation with frozen genes, a limited number of different individuals with the same objective value and the caching GA technique. The experimental results are encouraging and show effectiveness of the new encoding scheme. The proposed GA obtains solutions that match all known optimal solutions from the literature.

Further research should be directed to testing large-scale instances on more powerful and/or parallel computers as well as to investigate the combination of presented GA approach with some other metaheuristic.

REFERENCES

[1] Abraham, A., Nedjah, N., and Mourelle, L., "Evolutionary computation: From genetic algorithms to genetic programming", in: Nedjah et al. (eds.) *Studies in Computational Intelligence*, Springer, (2006) 1-20.

[2] Aleret, R.M., Valls, J., and Fernandez, O., "Evolving generalized Euclidean distances for training RBNN", *Computing and Informatics*, 26 (1) (2007) 33-43.

[3] Ben-David, G., and Malah, D., "Bounds on the performance of vector-quantizers under channel errors", *IEEE Transactions on Information Theory*, 51 (6) (2005), 2227-2235.

[4] Davendra, D., and Zelinka, I., "Optimization of quadratic assignment problem using self organising migrating algorithm", *Computing and Informatics,* 28 (2) (2009), 169-180.

[5] Djurić, B., Kratica, J., Tošić, D., and Filipović, V., "Solving the maximally balanced connected partition problem in graphs by using genetic algorithm", *Computing and Informatics*, 27 (3) (2008), 341-354.

[6] Drezner, Z., "Compounded genetic algorithms for the quadratic assignment problem", *Operations Research Letters*, 33 (5) (2005) 475-480.

[7] Drezner, Z., "The extended concentric tabu for the quadratic assignment problem", *European Journal of Operational Research*, 160 (2) (2005), 416-422.

[8] Drezner, Z., "Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem", *Computers and Operations Research*, 35 (3) (2008) 717-736.

[9] Duman, E., and Or, I., "The quadratic assignment problem in the context of the printed circuit board assembly process", *Computers and Operations Research*, 34 (1) (2007) 163-179.

[10] El-Baz, M.A., "A genetic algorithm for facility layout problems of different manufacturing environments", *Computers and Industrial Engineering*, 47 (23) (2004) 233-246.

[11] Filipović, V., "Fine-grained tournament selection operator in genetic algorithms", *Computing and Informatics*, 22 (2) (2003) 143-161.

[12] Hani, Y., Amodeo, L., Yalaoui, F., and Chen, H., "Ant colony optimization for solving an industrial layout problem", *European Journal of Operational Research*, 183 (2007) 633-642.

[13] Holland, J., *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, 1975.

[14] James, T., Rego, C., and Glover, F., "A cooperative parallel tabu search algorithm for the quadratic assignment problem", *European Journal of Operational Research*, 195 (3) (2009) 810-826.

[15] Khamis, A.M., Girgis, M.R., and Ghiduk, A.S., "Automatic software test data generation for spanning sets coverage using genetic algorithms", *Computing and Informatics*, 26 (4) (2007) 383-401.
[16] Koopmans, T. C., and Beckman, M. J., "Assignment problems and the location of economic activities", *Econometrica*, 25 (1957) 53-76.
[17] Kratica, J.: "Improving performances of the genetic algorithm by caching", *Computers and Artificial Intelligence*, 18, 3 (1999) 271-283.
[18] Kratica, J., Stanimirović, Z., Tošić, D., and Filipović, V., "Two genetic algorithms for solving the uncapacitated single allocation p-hub median problem", *European Journal of Operational Research*, 182 (1) (2007) 15-28.
[19] Kratica, J., Kovačević-Vujičić, V., and Čangalović, M., "Computing strong metric dimension of some special classes of graphs by genetic algorithms", *Yugoslav Journal of Operations Research*, 18 (2) (2008) 143-151.
[20] Liu, H., Abraham A., and Zhang, J., "A particle swarm approach to quadratic assignment problems", in: A. Saad et al. (ed.), *Advances in Soft Computing* , 39, Springer - Verlag, (2007) 213 - 222.
[21] Liu, H., and Abraham, A., "An hybrid fuzzy variable neighborhood particle swarm optimization algorithm for solving quadratic assignment problems", *Journal of Universal Computer Science*, 13 (9) (2007) 1309-1331.
[22] Loiola, E.M., de Abreu, N.M.M., Boaventura-Netto, P.O., Hahn, P., and Querido, T., "A survey for the quadratic assignment problem", *European Journal of Operational Research*, 176 (2) (2007) 657-690.
[23] Misevicius, A., "A modified simulated annealing algorithm for the quadratic assignment problem", *Informatica*, 14 (4) (2003) 497-514.
[24] Misevicius, A., "A tabu search algorithm for the quadratic assignment problem", *Computational Optimization and Applications*, 30 (1) (2005), 95-111.
[25] Mitchell, M., *Introduction to Genetic Algorithms*, MIT Press, Cambridge, Massachusetts, 1999.
[26] Oliveira, C.A.S., Pardalos, P.M., and Resende, M.G.G., "GRASP with path-relinking for the quadratic assignment problem", Lecture Notes in Computer Science, 3059 (2004) 356-368.
[27] Qahri Saremi, H., Abedin, B., and Meimand Kermani, A., "Website structure improvement: Quadratic assignment problem approach and ant colony meta-heuristic technique", *Applied Mathematics and Computation*, 195 (1) (2008) 285-298.
[28] Sahni, S., and Gonzales, T., "P-complete approximation problems", *Journal of the Association for Computing Machinery*, 23 (1976) 555-565.
[29] Singh, S.P., and Sharma, R.R.K., "Two-level modified simulated annealing based approach for solving facility layout problem", *International Journal of Production Research*, 46 (13) (2008) 3563-3582.
[30] Stutzle, T., "Iterated local search for the quadratic assignment problem", *European Journal of Operational Research*, 174 (3) (2006) 1519-1539.
[31] Tao J., Wang N., and Wang X., "Genetic algorithm based recurrent fuzzy neural network modeling of chemical processes", *Journal of Universal Computer Science*, 13 (9) (2007) 1332-1343.
[32] Tseng, L.-Yu., and Liang, S.-C., "A hybrid metaheuristic for the quadratic assignment problem", *Computational Optimization and Applications*, 34 (1) (2006) 85-113.
[33] Wang, R.L., and Okazaki, K., "Solving facility layout problem using an improved genetic algorithm", *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, E88a (2) (2005) 606-610.
[34] Wess, B., and Zeitlhofer, T., "On the phase coupling problem between data memory layout generation and address pointer assignment", *Lecture Notes in Computer Science*, 3199 (2004) 152-166.