Applied Mathematical Modelling 37 (2013) 9191–9202

Contents lists available at SciVerse ScienceDirect

Applied Mathematical Modelling

journal homepage: www.elsevier.com/locate/apm

A memetic algorithm approach for solving the multidimensional multi-way number partitioning problem

Petrică C. Pop [a,*], Oliviu Matei [b]

[a] Technical University of Cluj-Napoca, North University Center Baia Mare, Department of Mathematics and Computer Science, Baia Mare, Romania
[b] Technical University of Cluj-Napoca, North University Center Baia Mare, Department of Electrical Engineering, Baia Mare, Romania

ARTICLE INFO

Article history:
Received 16 May 2012
Received in revised form 23 October 2012
Accepted 29 March 2013
Available online 7 May 2013

Keywords:
Number partitioning
Genetic algorithms
Local search
Memetic algorithm
Combinatorial optimization

ABSTRACT

In this paper, we describe a generalization of the multidimensional two-way number partitioning problem (MDTWNPP) where a set of vectors has to be partitioned into p sets (parts) such that the sums per every coordinate should be exactly or approximately equal. We will call this generalization the multidimensional multi-way number partitioning problem (MDMWNPP). Also, an efficient memetic algorithm (MA) heuristic is developed to solve the multidimensional multi-way number partitioning problem obtained by combining a genetic algorithm (GA) with a powerful local search (LS) procedure. The performances of our memetic algorithm have been compared with the existing numerical results obtained by CPLEX based on an integer linear programming formulation of the problem. The solution reveals that our proposed methodology performs very well in terms of both quality of the solutions obtained and the computational time compared with the previous method of solving the multidimensional two-way number partitioning problem.

1. Introduction

Number partitioning problem is a classical, challenging and surprisingly difficult problem in combinatorial optimization and it is defined as follows: given a set S of n integers, the two-way number partitioning problem (TWNPP) asks for a division of S into two subsets such that the sums of number in each subset are as close as possible (equal or approximately equal).

Though the number partitioning problem is NP-complete (see [1]), there have been proposed heuristic algorithms that solve the problem in many instances either optimally or approximately. This is one of the reasons for which the problem has been called ''The Easiest Hard Problem'' by Hayes [2].

A variation of the number partitioning problem is the 3-partition problem, in which a set of numbers S must be partitioned into triples such that the sums in each subset to be equal or approximately equal.

The number partitioning problem has drawn a lot of attention due to its theoretical aspects and properties and important real-world applications in multiprocessor scheduling, the minimization of VLSI circuit size and delay, public key cryptography, voting manipulation, etc. For a more detailed description of the applications we refer to [3,4].

There are several ways to solve the TWNPP in exponential time in n: the most naive algorithm would be to cycle through all the subsets of n numbers and for every possible subset S_1 and for its corresponding complementary $S_2 = S \setminus S_1$ calculate their sums. Obviously, this algorithm is impracticable for large instances, since its time complexity is $O(2^n)$. A better exponential time algorithm which runs in $O(2^{n/2})$ was described by Horowitz and Sahni [5].

* Corresponding author.
E-mail address: petrica.pop@ubm.ro (P.C. Pop).

0307-904X/$ - see front matter
http://dx.doi.org/10.1016/j.apm.2013.03.075

Various heuristic algorithms have been developed for solving the TWNPP including: a natural greedy algorithm obtained by sorting the numbers in decreasing order and then assigning each number in turn to the subset with the smaller sum so far; a complete greedy algorithm described by Korf [6] where based on a binary tree each level assigns a different number and each branch point alternately assigns that number to one subset or the other; the set differencing heuristic introduced by Karmarkar and Karp [7] that repeatedly replaces the two largest numbers with their difference, inserting the new number in the sorted order until there is only one number left which is the final partition difference, the complete Karmarkar-Karp algorithm developed by Korf [6], a hybrid recursive algorithm obtained by combining several existing algorithms with some new extensions developed by Korf [8] in the case of the multi-way partition and tested on the three, four and five-way partitioning. Alidaee et al. [9] presented a new modeling of the multi-way partition problem as an unconstrained quadratic binary program and solved it by efficient metaheuristic algorithms.

Several metaheuristic approaches have been proposed for solving the two-way number partitioning problem including a Simulated Annealing algorithm by Johnsonn et al. [10], genetic algorithm by Ruml et al. [11], GRASP by Arguello et al. [12], Tabu Search by Glover and Laguna [13], memetic algorithm by Berretta et al. [14], etc.

The multidimensional two-way number partitioning problem (MDTWNPP) was introduced by Kojic [15] and is a generalization of the TWNPP in which given a set of vectors we are looking for a partition of the vectors into two subsets such that the sums per every coordinate should be as close as possible.

The MDTWNPP is NP-hard, as it reduces when the vectors have dimension one to the TWNPP, which is known to be an NP-hard problem.

Kojic [15] described as well an integer programming formulation and tested the model on randomly generated sets using CPLEX. The obtained experimental results show that the MDTWNPP is very hard to solve even in the case of medium size instances. To the best of our knowledge, this is the only existing approach for solving the MDTWNPP.

The aim of this paper is to describe a generalization of MDTWNPP called the multidimensional multi-way number partitioning problem (MDMWNPP) where a set of vectors has to be divided into a collection of mutually exclusive and collectively exhaustive subsets such that the sums per every coordinate in each of the subsets are as nearly equal as possible. In addition, we develop an efficient memetic algorithm based heuristic, obtained by combining a genetic algorithm with a powerful local search procedure, for solving the MDMWNPP. The results of extensive computational experiments in the case of multidimensional two, three and four-way partitioning are presented and analyzed. In the case of MDTWNPP the results reveal that our proposed methodology performs very well in terms of both quality of the solutions obtained and the computational time compared with the previous method introduced by Kojic [15].

2. Definition of the problem

Given a set of n vectors of dimension m

$$S = \{v_i \mid v_i = (v_{i1}, v_{i2}, \dots, v_{im}),\ i \in \{1, \dots, n\},\ m \in \mathbb{N}\},$$

then according to Kojic [15] the multidimensional two-way number partitioning problem consists in splitting the elements of S into two sets, S_1 and S_2 such that

1. $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$;
2. the sums of elements in the subsets S_1 and S_2 are equal or almost equal for all the coordinates.

If we introduce the variable t that denotes the greatest difference in sums per coordinate, i.e.

$$t = \max\left\{\left|\sum_{i\in S_1} v_{ij} - \sum_{i\in S_2} v_{ij} \mid j \in \{1, \dots, m\}\right\}\right.,$$

then the objective function of the MDTWNPP is to minimize t. If $\min t = 0$ then the partition will be called *perfect partition* for obvious reasons.

Next we define the multidimensional multi-way number partitioning problem (MDMWNPP) as a generalization of MDTWNPP where a set of vectors is partitioned into a given number of subsets rather than into two subsets.

Let again S be a set of n vectors of dimension m and $p \in \mathbb{N}, p \geqslant 2$, then the multidimensional multi-way number partitioning problem consists in splitting the elements of S into p subsets, $S_1, S_2, \dots, S_p$ such that.

1. $S_1 \cup S_2 \cup \ldots \cup S_p = S$ and $S_i \cap S_j = \emptyset$, for all $i, j \in \{1, \dots, p\}$ and $i \neq j$;
2. the sums of elements in the subsets $S_1, S_2, \dots, S_p$ are equal or almost equal for all the coordinates.

In particular, if the the set of vectors is partitioned into two subsets we get the MDTWNPP. For partitioning into more than two subsets, the objective function to be minimized is the greatest difference between maximum and minimum subset sums per every coordinate. Introducing the variable t denoting the greatest difference between maximum and minimum subset sums per every coordinate, i.e.

$$t = \max\left\{\left|\max\left\{\sum_{i\in S_l} v_{ij} \mid l \in \{1,\ldots,p\}\right\} - \min\left\{\sum_{i\in S_l} v_{ij} \mid l \in \{1,\ldots,p\}\right\} \mid j \in \{1,\ldots,m\}\right\},$$

then the objective function of the MDMWNPP is to minimize t.

Example. Let $S = \{(1,3); (4,4); (3,-2); (2,5); (2,-1)\}$ and we want to partition its elements into three subsets S_1, S_2 and S_3. We can do this partition in several ways, some candidates are:

- $S_1 = \{(1,3)\}, S_2 = \{(4,4)\}, S_3 = \{(3,-2); (2,5); (2,-1)\}$, then the sums are $(1,3), (4,4), (7,2)$, the difference between the maximum and minimum values per coordinates is $(6,2)$ and $t = 6$;
- $S_1 = \{(1,3); (3,-2)\}, S_2 = \{(4,4); (2,5)\}, S_3 = \{(2,-1)\}$, then the sums are $(4,1), (6,9), (2,-1)$, the difference between the maximum and minimum values per coordinates is $(4,10)$ and $t = 10$;
- $S_1 = \{(1,3)\}, S_2 = \{(4,4); (2,-1)\}, S_3 = \{(3,-2); (2,5)\}$, then the sums are $(1,3), (6,3), (5,3)$, the difference between the maximum and minimum values per coordinates is $(5,1)$ and $t = 5$;
- $S_1 = \{(2,5)\}, S_2 = \{(4,4); (2,-1)\}, S_3 = \{(3,-2); (1,3)\}$, then the sums are $(2,5), (6,3), (4,1)$, the difference between the maximum and minimum values per coordinates is $(4,4)$ and $t = 4$;
- $S_1 = \{(4,4)\}, S_2 = \{(1,3); (2,-1)\}, S_3 = \{(3,-2); (2,5)\}$, then the sums are $(4,4), (3,2), (5,3)$, the difference between the maximum and minimum values per coordinates is $(2,2)$ and $t = 2$.

Therefore, the minimum of the maximal elements of the listed candidates is in the fifth case with $\min t = 2$.

3. The memetic algorithm for solving the MDMWNPP

Memetic algorithms have been introduced by Mascato [16] to denote a family of metaheuristic algorithms that emphasis on the use of a population-based approach with separate individual learning or local improvement procedures for problem search. Therefore a memetic algorithm (MA) is a genetic algorithm (GA) hybridized with a local search procedure applied to all the individuals in order to intensify the search space.

Genetic algorithms are not well suited for fine-tuning structures which are close to optimal solutions. Therefore, embedding of local improvement operators into the recombination step of a GA is essential in order to obtain a competitive GA.

Memetic algorithms have been recognized as a powerful algorithmic paradigm for evolutionary computing, being applied successfully to solve combinatorial optimization problems such as the VRP (Vehicle Routing Problem) [17,18] and the CARP (Capacitated Arc Routing Problem) [19], the generalized traveling salesman problem [20], etc.

Our effective heuristic algorithm for solving the MDMWNPP is a memetic algorithm, which combines the power of genetic algorithm with that of local search. The details of the proposed MA are formally expressed as:

Step 1 Construct the initial population (see Section 3.1.2).
Step 2 Use local search procedures to replace each of the individuals of the initial population by the local optimum (see Section 3.2).
Step 3 Use genetic operators: crossover and mutation to produce the non-optimized next generation (see Section 3.1.4).
Step 4 Use local search procedures to replace each of the current generation solution by the local optimum (see Section 3.2).
Step 5 Repeat Steps 3 and 4 until a termination condition is reached.

The general scheme of our heuristic is given in Fig. 1.

Next we give the description of our memetic algorithm for solving the multidimensional multi-way number partitioning problem.

3.1. The genetic algorithm

3.1.1. Representation

It is known that a good representation scheme is important for the performance of the GA and it should define noteworthy crossover, mutation and other specific genetic operators to the problem in order to minimize the computational effort within these procedures. Designing such a representation is a hard problem in evolutionary computation.

In order to meet this requirement we use an efficient representation in which the solution structure is a fixed size ordered structure (n-dimensional vector) of integer numbers from the interval $[1, p]$. These integer numbers identify the set partitions as assigned to the vectors' (see Fig. 2). This representation ensures that the set of vectors belonging to the set S is partitioned into p subsets $S_1, S_2, \ldots, S_p$.

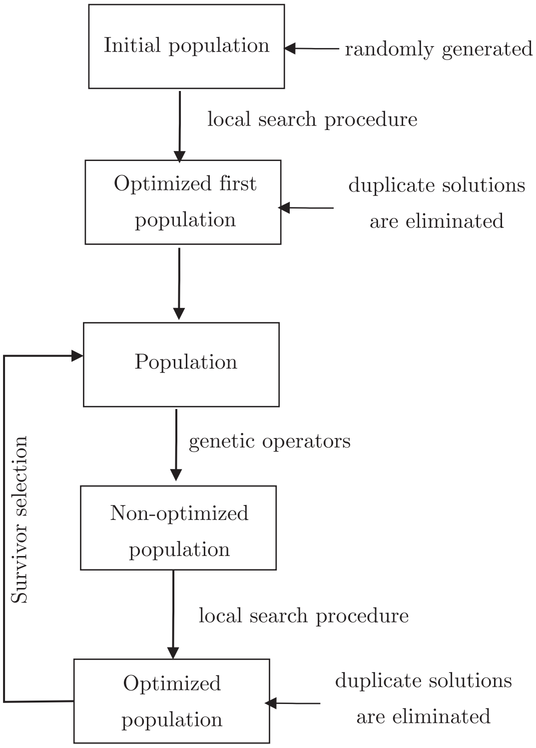

Fig. 1. Generic form of the proposed memetic algorithm.

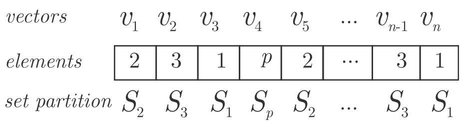

Fig. 2. Representation of an individual's chromosome.

3.1.2. Initial population

The construction of the initial population is of great importance to the performance of GA, since it contains most of the material the final best solution is made of.

Experiments have been carried out with two different ways of generating the initial population:

(1) A common method of population generation is random generation. Each gene for a chromosome assumes a value of $i, i \in \{1, \ldots, p\}$, with probability p_i, where $\sum_{i=1}^{p} p_i = 1$. This approach is efficient and provides a population covering the feasible region but it may lead to large values of the objective function yielding poor performance of the GA algorithm.
(2) Another method considered is based on generating the initial population partially randomly and partially based on the problem structure. A random number $q \in \{2, \ldots, n\}$ is generated and then for the vectors belonging to $\{2, \ldots, q\}$ the genes are generated randomly and the other vectors are partitioned iteratively such that by adding each vector we reduce the greatest difference in sums per coordinate.

Generating the population using as well the information about the problem structure permitted us to improve the initial population by 50% with respect to the fitness value in comparison to the randomly generation of the initial population.

3.1.3. The fitness value

Every solution has a fitness value assigned to it, which measures its quality. The fitness function will use an evaluation function to measure a value of worth for the individual so that they can be compared against each other and basically determines which possible solutions get passed onto multiply and mutate into the next generation of solutions. The rest of the genetic algorithm will discard any solutions with a "poor" fitness value and accept any with a "good" fitness value.

In our case, the fitness value of the MDMWNPP, for a given partition of the vectors into p subsets is given by the greatest difference between maximum and minimum subset sums per every coordinate. The aim is to find the partition that minimize this value denoted in the previous section by t.

3.1.4. Genetic operators

Genetic operators are used in genetic algorithms to bring diversity (mutation-like operators) and to combine existing solutions into others (crossover-like operators). The main difference among them is that mutation operators operate on one chromosome, while the crossover operators are binary operators.

Crossover operator

During each successive generation, a proportion of the existing population is selected to breed a new generation. The crossover operator requires some strategy to select two parents from the previous generation. In our case we selected the two parents using the binary tournament method, where two solutions, called parents, are picked from the population, their fitness is compared and the better solution is chosen for a reproductive trial. In order to produce a child, two binary tournaments are held, each of which produces one parent.

We experimented both single and double point crossover. Since there was not a big difference in the results we got from both methods, we decided to use single point crossover. The crossover point is determined randomly by generating a random number between 1 and $n - 1$. We decided upon crossover rate of 85 % by testing the program with different values. This means that 85% of the new generation will be formed with crossover and 15 % will be copied to the new generation.

Mutation operator

Mutation is a genetic operator that alters one ore more genes in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible. Mutation is an important part of the genetic search as helps to prevent the population from stagnating at any local optima and its purpose is to maintain diversity within the population and to inhibit the premature convergence.

We consider a mutation operator that changes the new offspring by flipping values from i to j, where $i, j \in \{1, \dots, p\}$. Mutation can occur at each value position in the string with 10 % probability.

3.1.5. Selection

Selection is the stage of a genetic algorithm in which individuals are chosen from a population for later breeding (crossover or mutation). The selection process is deterministic and is based on the fitness of the population. In our algorithm we investigated and used the properties of (μ, λ) selection, where μ parents produce λ $(\lambda \gg \mu)$ and only the offspring undergo selection. In other words, the lifetime of every individual is limited to only one generation. The limited life span allows to forget the inappropriate internal parameter settings. This may lead to short periods of recession, but it avoids long stagnation phases due to unadapted strategy parameters, because the new population is not attracted into local optima by the old population.

However, as the individuals evolve towards optima, the difference between the new population and the old one is getting smaller, because the offspring are always in the vicinity of their parents. This is the mechanism which makes the MA benefit of elitism. The high value of $\lambda = 10 \cdot \mu$ increases the chances that a optimum is reached rather than a new population gets out of it.

3.1.6. Genetic parameters

The genetic parameters are very important for the success of a GA, equally important as the other aspects, such as the representation of the individuals, the initial population and the genetic operators. Based on preliminary computational experiments, we set the following genetic parameters:

- The population size μ has been set to 10 times the number of the vectors. This turned out to be the best number of individuals in a generation.
- The intermediate population size λ was chosen ten times the size of the population: $\lambda = 10 \cdot \mu$.
- Mutation probability was set at 10%.
- The maximum number of generations (epochs) in our algorithm was set to 10,000.

In our algorithm the termination strategy is based on a maximum number of generations to be run if there is no improvement in the objective function for a sequence of 15 consecutive generations.

3.2. Local improvement procedure

Computational experiments showed that our proposed GA involving just the crossover and the mutation operators is effective in producing good solutions. However, based on the fact that classical GAs are not aggressive enough for some combinatorial optimization problems, we improved our GA algorithm by combining with local search procedures.

A local search heuristic tries to improve a solution by moving to a better neighbor solution. Whenever the neighboring solution is better than the current solution, it replaces the current solution. When no better neighbor solution can be found, the search terminates.

For each solution belonging to the initial population and for each new child obtained using the genetic operators, we use a local improvement procedure that runs several local search heuristics sequentially. Once an improvement move is found, it is immediately executed, meaning that the first improvement strategy is used rather than best-improvement strategy.

In our algorithm we used the k-change neighborhood local search heuristic where up to k bits are changed at a time in a complementary manner. We apply it for $k = 1, 2, 3$ as follows:

- 1-change neighbor. We select randomly an entry from the string representation. Suppose that the entry has a value $i \in \{1, \ldots, p\}$ than change its value randomly with another one $j \in \{1, \ldots, p\}, j \neq i$, meaning that we assign a vector from a partition to another partition. If there is an acceptable quality gain after the change, then it is accepted. The complexity of this procedure is $O(n)$, where n is the number of vectors. The 1-change procedure is repeated as long as improvements are achieved.
- 2-change neighbor. We select randomly two different entries from the string representation and swap their values, meaning that we interchanged two vectors belonging to different partitions. If there is an acceptable quality gain after the swap, then it is accepted. The complexity of this procedure is $O(n^2)$ and again the procedure is repeated as long as improvements are achieved.
- 3-change neighbor. We select randomly three different entries from the string representation and swap their values, meaning that we interchanged three vectors belonging to different partitions. The complexity of this procedure is $O(n^3)$ and again the procedure is repeated as long as improvements are achieved.

Our improvement procedure applies all the described local search heuristics cyclically. In the case of the MDTWNPP, we apply successively the 1, 2 and 3-change neighborhoods, in this order.

4. Computational results

In this section we present computational results in order to asses the effectiveness of our proposed memetic algorithm for solving the multidimensional multi-way number partitioning problem: two, three and four-way partitioning and we compare them with the results Kojic [15] using CPLEX. It is worth to mention that while the MA was designed for our specific problem and provides a suboptimal solution whose optimality cannot be checked, CPLEX is an optimization software package solving optimally linear programming problems, integer (mixed) programming problems, convex and non-convex quadratic programming problems, semidefinite programming problems, etc. and can provide a verification. (See Figs. 3, 4).

We conducted our computational experiments for solving the MDMWNPP on a set of instances generated randomly and following the general format $n - m$, where n represents the number of elements (vectors) and m represents the dimension of the vectors. We consider for each $n - m$ five instances denoted by a, b, c, d and e.

These instances were used by Kojic [15] in her computational experiments in the case of the multidimensional two-way number partitioning problem (MDTWNPP).

In our computational experiments we performed 10 independent runs for each instance.

The testing machine was an an Intel Dual-Core 1,6 GHz and 1 GB RAM with Windows XP Professional as operating system. The algorithm was developed in Java, JDK 1.6.

The following two tables show the computational results obtained with our memetic algorithm in comparison with those obtained by Kojic [15] using CPLEX for instances containing between 50 and 300 vectors and with the dimension between 2 and 20 in the case of the MDTWNPP.

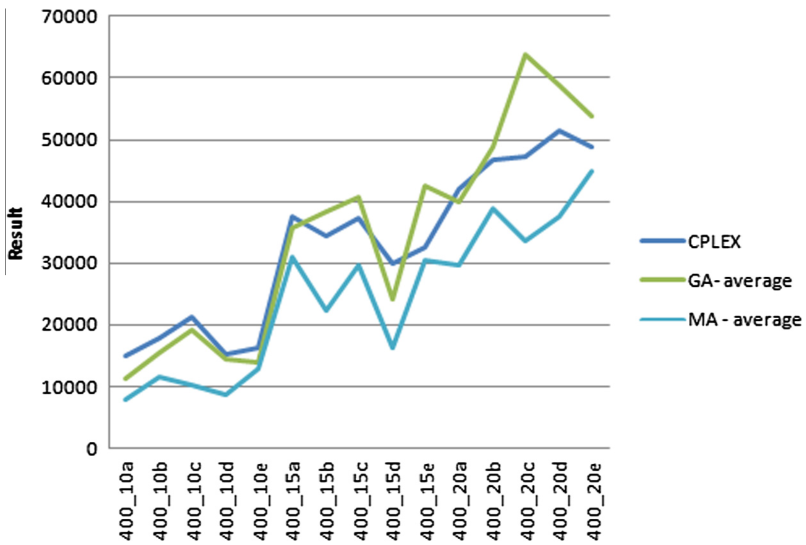

Fig. 3. MA versus GA and CPLEX for instances containing 400 vectors with the dimension between 10 and 20.

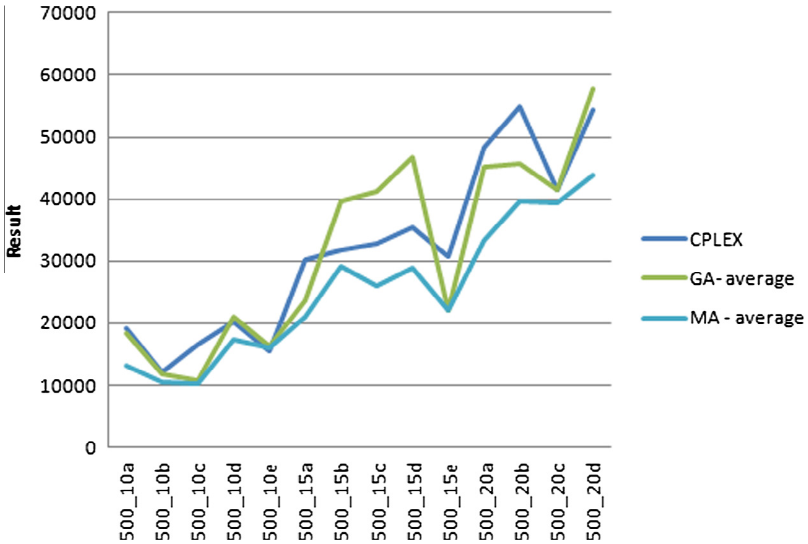

Fig. 4. MA versus GA and CPLEX for instances containing 500 vectors with the dimension between 10 and 20.

Table 1
Computational results for instances containing 50 and 100 vectors with the dimension between 2 and 20.

Problem instance	Results of CPLEX		Results of MA			Problem instance	Results of CPLEX		Results of MA		
	Best sol.	time	Best sol.	Avg. sol.	Avg. time		Best sol.	time	Best sol.	Avg. sol.	Avg. time
50_2a	3.204	552.60	5.438	5.502	35.73	100_2a	19.513	482.98	19.513	20.152	302.37
50_2b	9.193	63.75	9.193	9.546	32.38	100_2b	3.915	266.72	4.362	4.849	386.35
50_2c	9.191	161.9	9.191	9.756	38.98	100_2c	7.975	836.79	5.472	6.302	362.38
50_2d	4.753	92.01	4.753	5.023	42.65	100_2d	3.055	766.01	3.055	4.523	372.37
50_2e	6.719	1340.84	7.241	7.892	44.71	100_2e	2.077	1399.84	3.372	4.241	365.48
50_3a	303.581	465.91	283.563	289.541	162.34	100_3a	130.255	975.09	124.566.	130.332	397.28
50_3b	350.828	430.88	314.392	325.493	165.28	100_3b	97.935	80.71	98.372	108.283	392.64
50_3c	152.089	774.38	156.287	160.388	168.62	100_3c	263.199	383.98	256.822.	268.379	390.71
50_3d	102.516	1378.31	101.283	110.228	172.27	100_3d	247.043	1523.05	245.388	263.992	396.12
50_3e	217.903	312.16	198.854	212.211	171.64	100_3e	153.615	462.47	152.393	172.391	397.62
50_4a	909.765	1663.43	783.046	821.247	273.31	100_4a	520.496	664.68	521.283	531.635	402.83
50_4b	1272.224	1581.08	1017.283	1182.235	261.94	100_4b	1021.628	835.00	1021.628	1082.114	407.56
50_4c	461.161	1275.95	456.392	458.380	287.36	100_4c	908.24	232.91	900.394	910.838	402.66
50_4d	1024.681	827.7	987.27	1002.832	279.02	100_4d	1096.223	1678.59	1102.226	1132.432	411.42
50_4e	1199.574	498.3	1187.273	1201.398	281.07	100_4e	517.443	102.99	502.277	521.843	410.96
50_5a	926.164	281.77	918.278	925.226	286.45	100_5a	2441.489	570.23	2347.346	2536.142	521.76
50_5b	3202.875	1192.94	3002.869	3112.633	290.02	100_5b	2825.848	284.94	2539.387	2573.377	526.83
50_5c	2696.703	143.42	2563.904	2655.387	291.28	100_5c	2833.222	1236.07	2783.479	2837.741	534.21
50_5d	2275.792	357.38	2183.634.	2217.288	296.43	100_5d	2975.937	15.03	2637.388	2717.689	532.87
50_5e	4823.935	197.43	4689.382	4881.732	296.45	100_5e	4160.207	1072.36	4001.352	4112.556	540.02
50_10a	16176.578	1432.87	15722.29	16234.725	452.41	100_10a	17699.079	46.88	15792.742	16353.822	601.22
50_10b	19560.318	5.97	19560.318	20182.273	462.16	100_10b	18993.443	704.88	18367.389	18928.321	609.16
50_10c	17757.097	1308.68	15823.832.	16373.839	467.82	100_10c	15386.568	703.95	14628.836	15262.796	628.36
50_10d	14925.023	1104.7	14925.023	15025.637	425.76	100_10d	18276.246	337.87	16892.521	17066.389	615.27
50_10e	15369.009	472.44	14527.381	15263.833	476.45	100_10e	16516.277	689.61	15738.892	15938.556	612.67
50_15a	33208.019	1777.83	30728.546	32383.653	683.21	100_15a	32143.237	1082.82	30286.522	31762.362	642.37
50_15b	35003.301	1121.87	34362.391	35572.124	674.57	100_15b	28723.793	202.41	27893.837	28638.321	642.35
50_15c	29920.923	1443.76	28736.382	30826.276	684.65	100_15c	33363.206	1157.74	32982.347	33829.226	650.17
50_15d	21652.841	1594.98	20356.836	21723.622	690.25	100_15d	30706.17	1098.49	30706.17	31126.352	652.45
50_15e	31800.69	332.25	29018.285.	30025.245	692.46	100_15e	30253.623	280.81	28393.876	29731.532	659.81
50_20a	52826.34	71.83	50647.836	52876.443	635.26	100_20a	49992.607	1643.4	46782.412	47383.933	827.37
50_20b	51917.902	1378.83	50382.384	51662.929	657.41	100_20b	46691.489	1518.05	45673.837	46377.318	824.18
50_20c	50560.864	493.27	51829.374	51983.338	624.67	100_20c	45739.714	1042.18	44839.372	45211.776	826.31
50_20d	53955.965	166.26	51538.574	52632.307	635.82	100_20d	45371.992	406.33	45728.972	46182.389	834.38
50_20e	48281.499	234.75	47829.865	48292.728	652.36	100_20e	52315.704	1271.84	50732.852	51822.651	836.27

Table 2
Computational results for instances containing 200 and 300 vectors with the dimension between 2 and 20.

Problem instance	Results of CPLEX		Results of MA			Problem instance	Results of CPLEX		Results of MA		
	Best sol.	time	Best sol.	Avg. sol.	Avg. time		Best sol.	time	Best sol.	Avg. sol.	Avg. time
200_2a	11.463	766.53	8.241	9.739	235.46	300_2a	2.744	1753.38	3.627	3.746	529.37
200_2b	3.919	915.91	4.211	4.764	253.46	300_2b	6.958	682.5	5.377	5.824	530.29
200_2c	0	704.92	2.382	2.653	246.48	300_2c	2.73	1273.13	7.821	7.937	540.26
200_2d	2.691	1146.03	4.271	4.365	252.83	300_2d	0.881	1446.18	1.718	1.975	541.28
200_2e	0.971	661.39	1.822	1.927	255.61	300_2e	1.522	1647.92	1.522	1.661	543.14
200_3a	181.368	1773.64	152.832	158.021	261.26	300_3a	6.1	941.22	5.241	5.568	542.1
200_3b	137.584	256.78	102.653	115.874	267.36	300_3b	110.139	1097.1	98.28	102.631	654.39
200_3c	3.059	537.67	10.283	14.983	271.27	300_3c	226.933	459.49	210.082	215.822	560.18
200_3d	224.645	1129.01	189.732	200.648	273.37	300_3d	137.587	952.38	135.85	136.026	563.1
200_3e	120.542	844.37	102.762	112.390	278.37	300_3e	188.581	263.35	152.39	162.710	570.8
200_4a	537.018	372.95	521.761	528.301	273.37	300_4a	15.268	537.98	416.387	421.386	575.47
200_4b	1188.248	32.82	1067.442	1152.223	271.27	300_4b	1068.095	293.00	563.076	578.102	582.65
200_4c	6.109	469.97	7.566	7.738	271.98	300_4c	900.62	355.95	870.065	892.651	586.9
200_4d	1094.743	92.75	992.55	996.365	277.28	300_4d	1004.401	972.48	820.382	856.309	592.3
200_4e	1264.715	24.06	1288.543	1290.846	281.28	300_4e	908.869	449.44	873.648	891.280	592.21
200_5a	1931.064	440.7	1836.645	1903.833	290.3	300_5a	1847.76	320.52	1500.277	1578.098	591.13
200_5b	2734.271	188.68	2583.557	2648.890	293.62	300_5b	4195.209	138.36	3647.364	3748.481	592.14
200_5c	3576.93	236.02	3476.526	3567.447	286.21	300_5c	2658.01	1621.06	2502.882	2675.087	598.63
200_5d	2782.748	58.78	2538.372	2738.474	297.29	300_5d	2396.939	1339.89	2037.532	2283.816	601.27
200_5e	3798.611	98.85	3578.36	3647.498	300.24	300_5e	2499.651	135.62	2103.37	2326.145	603.45
200_10a	16530.321	146.75	16450.302	16635.145	378.37	300_10a	16112.376	40.06	12839.361	14263.705	700.27
200_10b	19616.619	151.34	18360.366	18837.398	384.12	300_10b	19954.971	205.00	17282.376	18262.881	708.27
200_10c	16158.656	759.02	15830.382	15928.005	403.38	300_10c	15996.203	184.77	15996.203	16002.847	710.28
200_10d	17399.449	1640.67	15482.076	16377.385	410.27	300_10d	20282.178	1768.84	18293.364	18937.982	715.34
200_10e	18107.353	151.13	18002.322	18272.902	420.39	300_10e	19620.941	70.3	17823.355	18373.755	721.28
200_15a	35139.957	669.95	31026.391	33384.028	480.2	300_15a	37524.309	1678.59	35672.273	35852.406	842.19
200_15b	34575.029	649.44	32948.021	33262.398	483.41	300_15b	34673.445	737.63	30748.487	31934.125	851.82
200_15c	35016.095	934.43	30464.392	32443.839	482.43	300_15c	30553.455	208.9	28938.388	29461.056	853.2
200_15d	33160.395	742.52	32647.473	33023.752	489.38	300_15d	36264.63	179.13	35483.364	35884.485	861.2
200_15e	29600.126	493.5	27483.366	28363.822	510.28	300_15e	32237.793	186.39	30823.478	31262.384	865.38
200_20a	44991.718	872.96	41937.357	42393.927	520.39	300_20a	47297.493	1302.81	45362.379	45965.495	1002.3
200_20b	49884.338	377.89	48393.228	49272.912	523.39	300_20b	44127.831	940.69	44017.288	44938.205	1034.21
200_20c	48451.593	334.35	45627.277	47397.803	530.38	300_20c	43594.894	1033.94	40382.273	41540.227	1057.12
200_20d	43631.462	382.39	43251.189	43526.398	531.29	300_20d	48814.817	1338.33	45637.146	46779.021	1127.44
200_20e	41768.116	1247.7	40272.654	41435.037	539.27	300_20e	50067.495	799.1	50067.495	50829.114	1183.39

The first and the seventh columns in the tables give the instances, the second, third, eighth and ninth columns provide the results obtained by Kojic [15] using CPLEX: the best solution and the necessary computational time in order to get it and the forth, fifth, sixth and the last three columns provide the results obtained by our novel memetic algorithm: the best solution, the average solution and the required time to get these average solutions. Because CPLEX did not finish its work in any considered instance in the tables are provided the best solutions obtained for a maximum of 30 min run for each test.

Analyzing the computational results from Tables 1 and 2, it should be noted that our approach compares favorable with the approach provided by Kojic [15] with respect to the best solution values: in 107 out of 140 instances we have been able to improve the objective function of the MDTWNPP, in 11 out of 140 we obtained the same solutions and in 22 out 140 instances our solutions are higher than those obtained using CPLEX [15]. As well, we can observe that in 85 out of 140 instances, even the average solutions provided by our MA algorithm are better than the solutions provided by Kojic using CPLEX.

Tables 3 and 4 depict the results of the memetic algorithm along with a comparison of the genetic algorithm and those obtained by Kojic [15] using CPLEX for instances containing between 400 and 500 vectors and with the dimension between 2 and 20 in the case of the multidimensional two-way number partitioning problem.

The first columns describe the instances, next two columns give the results of CPLEX: the best solution and the necessary computational time in order to get it, next four columns give the results of the GA alone: the best solution, the relative gap (as a percentage) between the best solution provided by CPLEX and the best solution provided by the GA, the average solution and the required time to get these average solutions and the last four columns give the results of the proposed MA: the best solution, the relative gap (as a percentage) between the best solution provided by CPLEX and the best solution provided by the MA, the average solution and the required time to get these average solutions.

Analyzing the results presented in Tables 3 and 4, we observe that our proposed heuristics MA performs very well in terms of both solution quality and computational times in comparison with the approach provided by Kojic [15]: in 60

Table 3
Computational results for instances containing 400 vectors with the dimension between 2 and 20.

Problem instance	Results of CPLEX		Results of GA				Results of MA			
	Best solution	time	Best solution	Gap %	Average solution	Average time	Best solution	Gap %	Average solution	Average time
400_2a	12.592	1708.5	7.438	0.409	10.785	148.53	7.216	0.426	9.452	121.44
400_2b	1.53	830.32	1.53	0	1.982	146.9	1.53	0	1.604	132.38
400_2c	4.354	956.34	4.354	0	5.188	131.65	3.711	0.147	4.115	128.32
400_2d	4.42	151.52	4.775	−0.08	5.546	135.8	4.42	0	5.168	130.29
400_2e	5.185	1427.76	5.185	0	5.966	156.38	4.839	0.066	5.104	149.32
400_3a	194.372	704.74	78.283	0.595	189.555	146.87	72.352	0.627	108.211	142.12
400_3b	175.9	478.81	129.352	0.264	193.818	154.38	121.235	0.31	167.233	123.81
400_3c	220.83	735.1	165.45	0.25	215.337	157.81	158.902	0.28	201.837	117.34
400_3d	257.849	80.78	178.345	0.308	292.76	158.27	127.746	0.504	228.287	134.18
400_3e	194.681	176.9	194.681	0	236.764	157.36	192.921	0.09	199.244	121.29
400_4a	1409.159	518.64	834.456	0.407	1008.728	188.28	821.822	0.416	892.384	156.39
400_4b	749.984	1237.82	683.273	0.088	856.671	192.28	662.002	0.117	664.623	136.34
400_4c	914.349	200.49	914.349	0	1132.511	191.27	901.255	0.014	905.115	176.35
400_4d	941.826	446.63	846.273	0.101	1149.915	193.24	842.721	0.105	1003.576	164.58
400_4e	902.761	576.52	627.75	0.304	896.435	195.46	622.382	0.31	723.197	176.59
400_5a	3737.767	79.9	3547.291	0.051	4912.599	262.89	3281.229	0.122	3385.147	221.72
400_5b	1494.313	1225.9	1453.29	0.027	1848.155	272.26	1421.065	0.049	1566.948	254.36
400_5c	2680.899	53.76	2640.928	0.014	3282.496	256.89	2638.989	0.015	2687.392	219.77
400_5d	2539.113	135.63	2423.948	0.045	3160.938	287.27	2421.321	0.046	2641.911	287.27
400_5e	1634.702	65.29	1489.725	0.088	2084.417	276.54	1465.103	0.103	1502.449	252.16
400_10a	14836.579	1622.34	7728.546	0.479	11235.401	356.27	7701.829	0.48	7838.938	340.66
400_10b	17918.141	1215.03	10918.141	0.39	15417.294	376.89	10827.980	0.395	11550.494	354.75
400_10c	21213.818	1703.88	9208.251	0.565	19088.936	324.5	9202.612	0.566	10144.09	290.43
400_10d	15212.906	1283.81	8212.906	0.46	14562.008	352.28	8212.906	0.46	8654.046	336.52
400_10e	16369.531	1530.48	13332.372	0.185	14009.345	342.56	12839.782	0.215	12992.594	320.45
400_15a	37574.022	926.03	29529.332	0.214	35754.439	321.35	29431.928	0.216	31043.73	304.65
400_15b	34390.093	62.52	21390.093	0.378	38253.683	326.28	21278.924	0.381	22411.202	279.04
400_15c	37161.817	1463.43	28621.829	0.229	40813.076	378.29	28425.883	0.235	29609.658	356.84
400_15d	30019.198	1203.22	16223.857	0.459	24117.878	381.1	16172.368	0.461	16293.763	366.54
400_15e	32561.093	26.19	30261.649	0.07	42612.356	378.29	30127.653	0.074	30466.451	321.76
400_20a	41974.284	767.3	28363.836	0.324	39800.183	390.28	28134.232	0.329	29765.538	366.02
400_20b	46751.348	354.05	38275.503	0.181	48713.445	368.45	37938.930	0.188	38740.68	354.77
400_20c	47259.514	313.95	32748.920	0.307	63799.921	372.32	32762.871	0.306	33469.524	342.41
400_20d	51544.421	31.38	36728.927	0.287	58840.664	310.38	36726.269	0.287	37467.746	289.36
400_20e	48792.272	251.36	43788.54	0.102	53750.069	302.65	43765.028	0.103	44902.726	263.47

out of 70 instances we have been able to improve the value of the objective function of the MDTWNPP, in 7 out of 70 instances we obtained the same solutions and in the case of the instance 500_2a, 500_3e and 500_5c the solutions provided by the MA are higher than the solutions obtained using CPLEX [15]. In addition, in 55 out of 70 instances even the average solutions provided by the MA improved the quality of the solutions provided by CPLEX.

As well we observe that the performance of the MA approach is superior to the GA in terms of both solution quality and computational times for MDTWNPP: in 63 out of 70 we have been able to improve the value of the objective function of the MDTWNPP, in 6 out of 70 instances we obtained the same solutions and in the case of the instance 400_20c the solution provided by MA is higher that the solution obtained using just the GA. The reason for obtaining better results in shorter amount of time is that when the search technique is incorporated in GA then the solution space in better searched.

According to the experimental results we can conclude that the local search heuristic is playing an important role in the GA process.

In the next two figures we represent the average solutions obtained using our proposed MA-based heuristic in comparison with the average solutions provided by the GA alone and the best solutions provided by CPLEX for instances containing 400 and 500 vectors with the dimension between 10 and 20.

Regarding the computational times, it is difficult to make a fair comparison between the approach of Kojic using CPLEX and our method because they have been evaluated on different computers, they are implemented in different languages and in addition Kojic did not mention the exact model of processor used in her computational experiments. The running time of our MA is proportional with the number of generations. However, from Tables 1–4, it should be noted that in average our heuristic is faster than the approach developed by Kojic [15].

In Fig. 5, we present a comparison of the average computational times of the MA versus GA, for instances containing between 400 and 500 vectors and with the dimension between 2 and 20.

Table 4
Computational results for instances containing 500 vectors with the dimension between 2 and 20.

Problem instance	Results of CPLEX		Results of GA				Results of MA			
	Best solution	time	Best solution	Gap %	Average solution	Average time	Best solution	Gap %	Average solution	Average time
500_2a	0.62	1745.39	1.354	−1.18	1.544	547.38	1.354	−1.18	1.438	544.65
500_2b	1.879	1797.03	2.213	-0.177	2.653	546.39	1.879	0	2.242	492.29
500_2c	2.013	1076.04	2.839	-0.41	3.048	548.49	2.013	0	2.539	478.44
500_2d	1.003	1687.53	1.003	0	1.202	539.3	1.003	0	1.017	421.26
500_2e	0.913	1726.6	1.234	-0.351	1.077	529.39	0.913	0	1.006	382.48
500_3a	213.539	1698.09	124.54	0.416	219.304	580.52	124.54	0.416	165.983	524.42
500_3b	142.16	806.88	132.878	0.065	149.71	589.3	132.878	0.065	139.076	502.22
500_3c	270.729	466.64	221.463	0.182	270.558	540.41	219.763	0.188	260.085	522.79
500_3d	13.736	1134.38	12.9485	0.057	14.1217	592.46	12.1321	0.116	13.589	520.04
500_3e	7.625	1091.25	12.453	-0.63	14.94	521.43	8.746	−0.147	9.225	381.33
500_4a	1562.920	171.29	1243.564	0.204	1864.105	638.37	1232.920	0.211	1749.848	567.08
500_4b	1186.722	429.41	980.253	0.174	1399.302	621.37	977.271	0.176	1082.888	465.59
500_4c	1093.756	63.74	837.384	0.234	1190.668	640.83	832.244	0.239	910.455	652.82
500_4d	4.58	714.37	4.58	0	5.961	610.27	4.580	0	4.869	552.16
500_4e	1410.694	106.31	938.384	0.334	949.861	634.52	937.823	0.335	1213.708	541.36
500_5a	3665.484	36.99	1922.135	0.475	3598.075	672.39	1903.802	0.48	2241.563	615.22
500_5b	4448.741	21.29	2833.853	0.363	4928.04	590.46	2824.579	0.365	3296.385	500.85
500_5c	3511.837	523.39	3928.203	-0.118	4367.228	696.59	3839.652	−0.09	4176.629	590.72
500_5d	2597.644	81.42	1823.239	0.298	2383.455	621.47	1821.393	0.298	2028.585	621.47
500_5e	2572.429	955.58	1822.343	0.291	2448.885	654.73	1816.680	0.293	2023.652	579.12
500_10a	19183.301	1718.29	12938.304	0.325	18317.79	736.74	12892.029	0.328	13213.687	637.01
500_10b	12161.350	128.48	10393.382	0.145	11712.657	701.29	10387.625	0.145	10625.234	622.27
500_10c	16594.760	368.13	10283.385	0.38	10645.003	692.03	10282.295	0.38	10324.87	612.28
500_10d	20284.381	1699.01	16378.394	0.192	20874.022	678.9	16328.752	0.195	17283.48	678.90
500_10e	15548.670	1680.47	14950.76	0.038	16200.764	732.39	14738.344	0.052	15928.492	635.37
500_15a	30316.775	1055.81	20394.564	0.327	23566.966	720.31	20332.662	0.329	21087.687	682.31
500_15b	31878.383	1591.08	28348.563	0.11	39679.188	743.86	28283.569	0.112	29115.211	721.62
500_15c	32792.472	803.77	25484.567	0.222	41151.566	783.09	25461.223	0.223	26118.369	719.81
500_15d	35555.260	881.27	27394.64	0.229	46711.844	810.28	27283.392	0.232	28970.01	724.37
500_15e	30806.719	455.06	21849.57	0.29	22171.456	807.62	21652.005	0.297	22028.363	728.90
500_20a	48281.977	1000.12	32934.495	0.317	45219.074	843.3	32897.388	0.318	33238.612	843.30
500_20b	54921.900	237.63	38494.084	0.299	45750.657	873.39	38467.134	0.299	39675.389	818.39
500_20c	41578.884	1382.98	39495.452	0.05	41516.187	812.73	38763.093	0.067	39474.872	736.90
500_20d	54293.200	1728.58	43840.674	0.192	57730.837	843.3	43637.932	0.193	43763.0979	792.38
500_20e	41092.622	1713.03	40352.904	0.018	53286.063	921.83	40341.827	0.018	40654.702	918.22

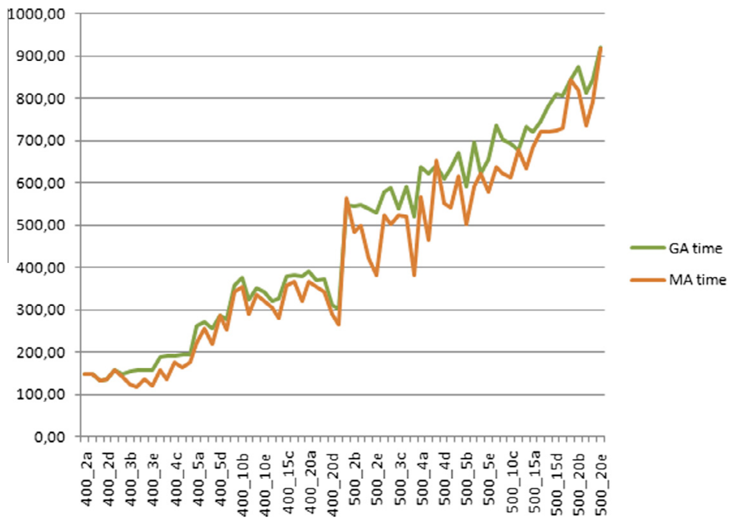

Fig. 5. Running time comparison of the GA and MA.

Table 5
Computational results three-way and four way multidimensional partition.

Problem instance	Results of MA			Problem instance	Results of MA		
	Best sol.	Avg. sol.	Avg. time		Best sol.	Avg. sol.	Avg. time
50_2a	84.5	86.2	182.45	50_2a	370.7	391.4	342.37
50_3a	329.9	334.4	202.34	50_3a	650.3	678.3	536.24
50_4a	3370.6	3382.5	673.83	50_4a	801.2	836.7	1023.39
50_5a	4106.4	4125.8	781.82	50_5a	1006.4	1094.4	1243.28
50_10a	37485.4	37521.6	1189.38	50_10a	41829.1	42005.6	1647.76
50_15a	55960.2	56015.2	1212.27	50_15a	55960.2	56034.7	1843.92
50_20a	102394.8	102652.0	1235.22	50_20a	123484.8	123627.8	2135.48
100_2a	161.7	178.1	342.39	100_2a	662.2	687.2	564.02
100_3a	510.2	531.3	428.38	100_3a	1193.3	1213.7	847.37
100_4a	833.9	867.5	673.22	100_4a	1728.5	1924.6	922.14
100_5a	6169.4	6224.5	834.62	100_5a	8272.2	8356.8	1972.39
100_10a	46690.6	47004.8	1436.08	100_10a	74638.6	75034.8	2819.32
100_15a	96661.7	96827.3	2073.76	100_15a	122421.3	122892.7	3193.84
100_20a	112838.5	113112.5	2564.38	100_20a	174383.1	174981.6	4392.01

We can seen that the MA is more efficient that the GA as almost all the results are achieved in shorter time.

Table 5 shows the results obtained using our proposed MA in the case of the three-way and four-way multidimensional number partitioning problem for instances containing 50, respectively 100 vectors with dimension between 2 and 20.

In the first part of Table 5 we presented the best solutions, average solutions and the average computational times in the case of the three-way multidimensional partition problem and in the second part the results obtained in the case of the four-way multidimensional partition problem.

Taking into account the computational results presented in Tables 1–5, we can conclude that our proposed MA-based heuristic leads to good results within reasonable times and outperforms the approach introduced by Kojic [15] and the genetic algorithm alone. The success of our algorithm relies on the combination of the genetic algorithm with the local search procedure.

5. Conclusions

In this paper, we considered a generalization of the of the multidimensional two-way number partitioning problem (MDTWNPP) where a set of vectors has to be partitioned into p sets (parts) such that the sums per every coordinate should be exactly or approximately equal, called the multidimensional multi-way number partitioning problem (MDMWNPP).

We developed an efficient memetic algorithm for solving the MDMWNPP and in particular the MDTWNPP, that combines a genetic algorithm with a powerful local search procedure consisting of three local search heuristics.

The extensive computational results show that our memetic algorithm is robust and compares favorably in comparison to the existing approach and the genetic algorithm alone.

In the future, we plan to explore the possibility of building a parallel implementation of the system in order to improve the execution time. In addition, we will need to asses the generality and scalability of the proposed heuristic by testing it on larger instances.

Acknowledgments

This work was supported by a grant of the Romanian National Authority for Scientific Research, CNCS - UEFISCDI, project number PN-II-RU-TE-2011–3-0113. The authors are grateful to the anonymous referees for reading the manuscript very carefully and providing constructive comments which helped to improve substantially the paper.

References

[1] M.R. Garey, D.S. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness, W.H. Freeman, New York, 1997.
[2] B. Hayes, The easiest hard problem, Am. Sci. 90 (2002) 113–117.
[3] E. Coffman, G.S. Lueker, Probabilistic Analysis of Packing and Partitioning Algorithms, John Wiley & Sons, New York, 1991.
[4] T. Walsh, Where are the really hard manipulation problems? The phase transition in manipulating the veto rule, in: Proceedings of IJCAI-09 (2009) pp. 324–329.
[5] E. Horowitz, S. Sahni, Computing partitions with applications to the Knapsack problem, J. ACM 21 (2) (1974) 277–292.
[6] R.E. Korf, A complete anytime algorithm for number partitioning, Art. Intel. 106 (2) (1998) 181–203.
[7] N. Karmarkar, R.M. Karp, The differencing method of set partitioning, Technical Report UCB/CSD 82/113, Computer Science Division, University of California, Berkeley, 1982.
[8] R.E. Korf, A hybrid recursive multi-way number partitioning algorithm, in: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (2011) pp. 591–596.
[9] B. Alidaee, F. Glover, G. Kochenberger, C. Rego, A new modeling and solution approach for the number partitioning problem, J. Appl. math. decis. sci. 9 (2) (2005) 135–145.

[10] D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, Optimization by simulated annealing: an experimental evaluation. Part II: Graph coloring and number partitioning, Oper. Res. 39 (3) (1991) 378–406.
[11] W. Ruml, J.T. Ngo, J. Marks, S.M. Shieber, Easily searched encodings for number partitioning, J. Optim. Theory Appl. 89 (2) (1996) 251–291.
[12] M.F. Arguello, T.A. Feo, O. Goldschmidt, Randomized methods for the number partitioning problem, Comput. Oper. Res. 23 (2) (1996) 103–111.
[13] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1997.
[14] R.E. Berretta, P. Moscato, C. Cotta, Enhancing a memetic algorithms' performance using a matching-based recombination algorithm: results on the number partitioning problem, in: M.G.C. Resende, J. Souza (Eds.), Metaheuristics: Computer Decision-Making, Kluwer, 2004.
[15] J. Kojic, Integer linear programming model for multidimensional two-way number partitioning problem, Comput. Math. Appl. 60 (2010) 2302–2308.
[16] P. Moscato, On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms, Caltech Concurrent Computation Program, Report 826, 1989.
[17] j.e. Mendoza, B. Castanier, C. Gueret, A.L. Medaglia, N. Velasco, A memetic algorithm for the multi-compartment vehicle routing problem with stochastic demand, Comput. Oper. Res. 37 (2010) 1886–1898.
[18] S.U. Ngueveu, C. Prins, R.W. Calvo, An effective memetic algorithm for the cumulative capacitated vehicle routing problem, Comput. Oper. Res. 37 (2010) 1877–1885.
[19] C. Prins, S. Bouchenoua, A memetic algorithm solving the vrp, the carp and general routing problems with nodes, edges and arcs, Stud. Fuzziness Soft Comput. 166 (2005) 65–85.
[20] B. Bontoux, C. Artigues, D. Feillet, A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem, Comput. Oper. Res. 37 (11) (2010) 1844–1852.