STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVIII**, Number 3, 2013

HEURISTIC ALGORITHMS FOR SOLVING THE BI-DIMENSIONAL TWO-WAY NUMBER PARTITIONING PROBLEM

LEVENTE FUKSZ[(1)], PETRICĂ POP [(2)], AND IOANA ZELINA [(2)]

Abstract. The bi-dimensional two-way number partitioning problem is a generalization of the classical number partitioning problem, where instead of a set of integers we have a set of vectors of dimension 2 that have to be divided into two subsets so that the sums of the vectors in the subsets are equal or almost equal for both coordinates. This work presents three heuristic algorithms for solving the problem. The algorithms are analyzed, implemented and tested on randomly data instances.

1. Introduction

The number partitioning problem (NPP) is a well known combinatorial optimization problem defined as follows: given a finite set of n integers $S = \{a_1, a_2, ..., a_n\}$, find a partition of S into two subsets S_1 and S_2 such that it minimizes the difference of the sums of the elements in the subsets:

$$\Big| \sum_{i \in S_1} a_i - \sum_{i \in S_2} a_i \Big| \rightarrow min.$$

This difference is called *discrepancy* and the partitions with the property that the discrepancy is 0 or 1 are called *perfect partitions.*

The NPP is a difficult problem belonging to the the class of *NP-hard* problems [3] and therefore it is difficult to be solved in both theory and practice.

Received by the editors: April 22, 2013.

2010 *Mathematics Subject Classification.* 90C27, 68T15.

1998 *CR Categories and Descriptors.* G.2.1 [**Mathematics of Computing**]: Discrete mathematics – *Combinatorial Algorithms*; I.2.8 [**Computing Methodologies**]: Artificial intelligence – *Problem Solving, Control Methods and Search.*

Key words and phrases. heuristic algorithms, number partitioning problem, bi-dimensional two-way number partitioning problem.

This paper has been presented at the International Conference KEPT2013: Knowledge Engineering Principles and Techniques, organized by Babeş-Bolyai University, Cluj-Napoca, July 5-7 2013.

The problem finds interesting applications in public key encryption, scheduling problems, minimization of the VLSI circuit size, choosing up fair sides in a ball game, etc.

There are several ways to solve the NPP 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].

There have been proposed several heuristic algorithms in order to provide high-quality solutions for the NPP: the set differencing heuristic introduced by Karmarkar and Karp [7], a Simulated Annealing algorithm described by Johnsonn et al. [6], genetic algorithm by Ruml et al. [11], GRASP by Arguello et al. [1], Tabu Search by Glover and Laguna [4], memetic algorithm by Berretta et al. [2], etc.

Kojic [8] described a generalization of the NPP called *the multidimensional two-way number partitioning problem* (MDTWNPP), where instead of numbers we have a set of vectors and 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. Kojic provided 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 instances. Recently, Pop and Matei [10] introduced an efficient memetic algorithm for solving the MDTWNPP.

In this work we confine to the case when the vectors have dimension two and the aim of this paper is to describe three heuristic approaches: a greedy algorithm and a novel use of genetic algorithms with the goal of solving the bi-dimensional two-way number partitioning problem and a hybrid GA-VNS heuristic that combines the use of genetic algorithms (GA) and Variable Neighborhood Search (VNS). The results of preliminary computational experiments are presented and analyzed.

2. Definition of the bi-dimensional two-way number partitioning problem

Given a set of n bi-dimensional vectors:

$$S = \{v_i \mid v_i = (v_{i1}, v_{i2}),\ i \in \{1, ..., n\}\}$$

then the *bi-dimensional two-way number partitioning problem* (BTWNPP) 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 both coordinates.

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

$$t = \max\left\{ \left| \sum_{i \in S_1} v_{ij} - \sum_{i \in S_2} v_{ij} \right| \ : \ j \in \{1, 2\} \right\}$$

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

Defined in this way, we can observe that the BDTWNPP is a special case of the multidimensional two-way number partitioning problem introduced by Kojic [8], where the vectors are bi-dimensional.

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

Next we define the *bi-dimensional multi-way number partitioning problem* (BDMWNPP) as a generalization of BDTWNPP 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 bi-dimensional vectors and $p \in \mathbb{N}$, $p \geq 2$, then the bi-dimensional multi-way number partitioning problem consists in splitting the elements of S into p subsets, $S_1, S_2, ..., S_p$ such that

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

In particular, if the set of vectors is partitioned into two subsets we get the BDTWNPP. For partitioning into more than two subsets, the objective function to be minimized is the greatest difference between maximum and minimum subset sums for both coordinates.

Introducing the variable t denoting the greatest difference between maximum and minimum subset sums per every coordinate, i.e.

$$t = \max\left\{ \left| \max\left\{ \sum_{i \in S_l} v_{ij} : l = \overline{1, p} \right\} - \min\left\{ \sum_{i \in S_l} v_{ij} : l = \overline{1, p} \right\} \right| \ j \in \{1, 2\} \right\}$$

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

Example Let $S = \{(1, 4), (2, 9), (7, 2), (5, 5), (3, 7), (4, 10), (3, 2)\}$ and we want to partition its elements into two subsets S_1 and S_2. We can do this partition in several ways, some candidates are:

- $S_1 = \{(1, 4), (2, 9), (7, 2)\}$, $S_2 = \{(5, 5), (3, 7), (4, 10), (3, 2)\}$, then the sums are $(10, 15)$ and $(15, 24)$, the difference between the maximum and minimum values per coordinates is $(5, 9)$ and therefore $t = 9$;

- $S_1 = \{(1,4),(2,9),(7,2),(5,5)\}$, $S_2 = \{(3,7),(4,10),(3,2)\}$, then the sums are $(15,20)$ and $(10,19)$, the difference between the maximum and minimum values per coordinates is $(5,1)$ and $t = 5$;
- $S_1 = \{(1,4),(4,10),(7,2)\}$, $S_2 = \{(5,5),(3,7),(2,9),(3,2)\}$ then the sums are $(12,16)$ and $(13,23)$, the difference between the maximum and minimum values per coordinates is $(1,7)$ and $t = 7$;
- $S_1 = \{(1,4),(2,9),(7,2),(3,7)\}$, $S_2 = \{(5,5),(4,10),(3,2)\}$, then the sums are $(13,22)$ and $(12,17)$, the difference between the maximum and minimum values per coordinates is $(1,5)$ and $t = 5$;

Therefore, the minimum of the maximal elements of the listed candidates is in the second and fourth cases with $\min t = 5$.

3. Heuristic algorithms for solving the bi-dimensional two-way number partitioning problem

3.1. **The Greedy algorithm.** For the number partitioning problem, the obvious greedy heuristic is to sort the numbers in decreasing order, place the largest number in one of the two subsets and then place each of the remaining numbers in the subset with the smallest sum.

However, for the two-way number partitioning problem, there are two sums for each subset. In this case, each time we need to put a pair in one of the two existing subsets, we have to analyze both possibilities and then choose the one which gives the smallest difference.

For the given example, $S = \{(1,4),(2,9),(7,2),(5,5),(3,7),(4,10),(3,2)\}$, we put $(1,4)$ in S_1 and $(0,0)$ in S_2. The current sub-sums are $(1,4)$ respectively $(0,0)$. To put $(2,9)$ in one of the two subsets, we calculate the sub-sums and we obtain $(3,13)|(0,0)$, respectively $(1,4)|(2,9)$. In the first case the difference is $max\{|3-0|,|13-0|\} = 13$ and in the second case the difference is $max\{|1-2|,|4-9|\} = 5$. Therefore, $(2,9)$ goes in S_2 and we have the sub-sums $(1,4)|(2,9)$. For $(7,2)$ we have the sub-sums $(8,6)|(2,9)$, respectively $(1,4)|(9,11)$ with the smallest difference $t = 6$ so $(7,2)$ goes in S_1. For $(5,5)$, the sub-sums are $(13,11)|(2,9)$, respectively $(8,6)|(7,14)$ with the smallest difference $t = 8$ and $(5,5)$ goes in S_2. For $(3,7)$, the sub-sums are $(11,13)|(7,14)$ respectively $(8,6)|(10,21)$ with the smallest difference $t = 4$ and $(3,7)$ goes in S_1. For $(4,10)$, the sub-sums are $(15,23)|(7,14)$ respectively $(11,13)|(11,24)$ with the smallest difference $t = 9$ and $(4,10)$ goes in S_1. For $(3,2)$, the sub-sums are $(18,25)|(7,14)$ respectively $(15,23)|(10,16)$ with the smallest difference $t = 7$ and $(4,10)$ goes in S_2. We obtain then the partition $S_1 = \{(1,4),(7,2),(3,7),(4,10)\}$, $S_2 = \{(2,9),(5,5),(3,2)\}$, the sub-sums are $(15,23)$ and $(10,16)$ and the smallest difference is $t = 7$. We can rewrite the

sequence as $(1,4)|(0,0),(1,4)|(2,9),(8,6)|(2,9),(8,6)|(7,14),(11,13)|(7,14),(15,23)|(7,14),(15,23)|(10,16)$ and the smallest difference $t=7$.

Each time we put a pair in one of the two subsets we have to perform 11 computational steps: 8 additions and 3 comparisons, so we perform $(n-1)\times 11$ computational steps for a set with n pairs. For our example, we perform $6\times 11=66$ computational steps. If both current subsumes in one of the subsets are smaller or equal to the sub-sums in the other subset and the pairs are pairs of positive integers, then we can put the next pair directly in the subset with the smallest sub-sums. In this case, we perform only 4 additions and 3 comparisons instead of 8 additions and 3 comparisons. For our example, we put $(7,2)$ directly in S_1 because both subsumes in S_1, $(1,4)$ are smaller than those in S_2, $(2,9)$. The number of computational steps becomes $7+7+11+11+11+7=54$.

The suboptimal solution provided by the greedy heuristic algorithm depends on the order the pairs in the set S are chosen.

If we sort the pairs in S in decreasing order according to the second component, we obtain $S=\{(4,10),(2,9),(3,7),(5,5),(1,4),(3,2),(7,2)\}$. The sequence that describes the greedy algorithm becomes: $(4,10)|(0,0),(4,10)|(2,9),(4,10)|(5,16),(9,15)|(5,16),(9,15)|(6,20),(9,15)|(13,22),(12,17)|(13,22)$ and the smallest difference is $t=5$. The subsets are $S_1=\{(4,10),(5,5),(3,2)\}$ and $S_2=\{(2,9),(3,7),(1,4),(7,2)\}$. The number of computational steps is $7+7+7+11+11+7=50$. In this case the solution is better and the number of computational steps is smaller than in the previous case.

3.2. **The genetic algorithm.** The Genetic Algorithms (GA) were introduced by Holland in the early 1970s, and were inspired by Darwins theory. The idea behind GA is to model the natural evolution by using genetic inheritance together with Darwins theory. GA have seen a widespread use among modern metaheuristics, and several applications to combinatorial optimization problems have been reported. Next we give the description of our genetic algorithm for solving the BDTWNPP.

Representation

In order to represent a potential solution to the BDTWNPP, we used a binary representation where every chromosome is a fixed size (n-dimensional vector) ordered string of bits 0 or 1, identifying the set of partition as assigned to the pairs. This representation ensures that the set of vectors belonging to the set S is partitioned into two subsets S_1 and S_2.

Initial population

The construction of the initial population is of great importance to the performance of GA, since it contains part of the building blocks the final solution is made of, which is then combined by the crossover operator.

We considered a novel method for generating the initial population: partially randomly and partially based on the problem structure. In this case, we pick randomly a number $q \in \{2, ..., n\}$ and then for the pairs belonging to $\{2, ..., q\}$ the genes are generated randomly and the other pairs are partitioned iteratively such that by adding each pair we reduce the greatest difference in sums per coordinate.

The fitness value

GAs require a fitness function which allocates a score to each chromosome in the current population. Thus, it can calculate how well the solutions are coded and how well they solve the problem. In our case, the fitness value of the BDTWNPP, for a given partition of the pairs into two subsets is given by the corresponding greatest difference in sums per coordinate and the aim of the problem is to minimize this value.

Selection

Selection is the process used to select individuals for reproduction to create the next generation. This is driven by a fitness function that makes higher fitness individuals more likely to be selected for creating the next generation. We have implemented three different selection strategies: the fitness proportionate selection, the elitist selection and the tournament selection. Several BDTWNPP were tested and the results show that the tournament selection strategy outperformed the other considered selection strategies, achieving best solution quality with low computing times.

Genetic operators

During each successive generation, a proportion of the existing population is selected to produce a new generation. The *crossover operator* requires some strategy to select two parents from previous generation. In our case we selected the two parents using the binary tournament method, where p solutions, called parents, are picked from the population, their fitness is compared and the best 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 have experimented a 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 0.85 based on preliminary experiments with different values.

Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state and it 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 bits from 1 to 0 or from 0 to 1. Mutation can occur at each bit position in the string with 0.1 probability.

An important feature of our GA, that increased its performance, is that every time a new population is produced, we eliminate the duplicate solutions.

In our algorithm the termination strategy is based on a maximum number of generations to be run if the optimal solution of the problem is not found or no improvement of the discrepancy value is not observed within 15 consecutive generations.

As we will see in the Computational results section, our proposed GA is effective in producing good solutions. However, due to the weakness of GAs to intensify the search in promising areas of the solutions space, we will combine our GA with the local search ability of VNS in order to enhance the exploitation ability of GAs.

3.3. **The GA-VNS hybrid algorithm.** Variable neighborhood search (VNS) is quite a recent metaheuristic for solving combinatorial optimization and global optimization problems introduced by Mladenovic and Hansen [9]. Its basic idea is a systematic change of neighborhood both within a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley.

VNS is based on two simple facts:

- **Fact 1:** A local minimum w.r.t. one neighborhood structure is not necessary a local minimum with another;
- **Fact 2:** A global minimum is a local minimum w.r.t. all possible neighborhood structures.

The hybrid algorithm that we are going to present in this section combines the GA described in the previous section with a Variable Neighborhood Search procedure.

Applying local search to all the individuals of a current population will lead to highly time consuming procedure, therefore we selected a subset of individuals in each generation with a specified probability and then the VNS procedure is applied to each of them separately. In the case better individuals are found they are introduced in the current population.

Our VNS algorithm applies 10 types of neighborhoods, denoted by $\mathcal{N}_i$, $i \in \{1, ..., 10\}$. The neighborhoods are implemented as inversions of bits (representing either 0 or 1) within the chromosome with positions generated randomly. The ten neighborhoods correspond to the number of bits which are inverted $i \in \{1, ..., 10\}$. For each neighborhood the following repetitive loop is applied: we choose randomly the positions within the chromosome made up of bits and then we inverse the corresponding genes by logical negation.

The first neighborhood is the set of candidate solutions that have one bit difference against the current solution. We select randomly an entry from the string representation and then inverse the value of the corresponding gene,

meaning that we assign an integer from one set to the other one. If by changing the value of a gene, we obtain a neighbor having a lower cost than the current solution, than the neighbor becomes the new current solution and the search proceeds. The search process continues until no better solution is found in the neighborhood, then the search switches to the second neighborhood, which consists of candidate solutions having exactly two bits difference against the current solution. This new neighborhood is examined in order to find an improvement solution and the search continues till a better feasible solution cannot be found.

Then the search switches to the new neighborhood and the process goes on iteratively.

The switching of neighborhoods prevents the search being struck at a local optimum. When there is no better solution found in a current neighborhood, it can be a local optimum, but by changing the neighborhood, it is highly probable that a better feasible solution can be found and the local optimum is skipped.

The described strategy show how to use VNS in descent in order to escape from a local optimum of and now we are interested in finding promising regions for sub-optimal solutions.

Our implementation of the VNS procedure is described in Algorithm 1.

Algorithm 1 Variable Neighborhood Search Framework

Initialization. Select a set of neighborhoods structures $\mathcal{N}=\{\mathcal{N}_l \mid l = 1, ..., 10\}$; an initial solution x and a stopping criterion
Repeat the following sequence till the stopping criterion is met:
(1) Set $l = 1$;
(2) Repeat the following steps until $l = 10$:
Step 1 (Shaking): Generate $x' \in \mathcal{N}_l$ at random;
Step 2 (Local Search): Apply a local search method starting with x' as initial solution and denote by x'' the obtained local optimum;
Step 3 (Move or not): If the local optimum x'' is better than the incumbent x,
then move there ($x \leftarrow x''$) and continue the search with $\mathcal{N}_1$
otherwise set $l = l + 1$ (or if $l = 10$ set ($l = 1$);
Go back to Step 1.

According to this basic scheme, we can observe that our VNS is a random descent first improvement heuristic.

The algorithm starts with an initial feasible solution x from the selected individuals from the current population and with the set of the 10 nested

neighborhood structures: $\mathcal{N}_1$,..., $\mathcal{N}_{10}$, having the property that their sizes are increasing:

$$\mathcal{N}_1(x) \subseteq \mathcal{N}_2(x) \subseteq ... \subseteq \mathcal{N}_{10}(x).$$

Then a point x' at random (in order to avoid cycling) is selected within the first neighborhood $\mathcal{N}_1(x)$ of x and a descent from x' is done with the local search routine. This will lead to a new local minimum x''. At this point, there exists three possibilities:

1) $x'' = x$, i.e. we are again at the bottom of the same valley and we continue the search using the next neighborhood $\mathcal{N}_l(x)$ with $l \geq 2$;
2) $x'' \neq x$ and $f(x'') \geq f(x)$, i.e. we found a new local optimum but which is worse than the previous incumbent solution. Therefore, also in this case, we will continue the search using the next neighborhood $\mathcal{N}_l(x)$ with $l \geq 2$;
3) $x'' \neq x$ and $f(x'') < f(x)$, i.e. we found a new local optimum but which is better than the previous incumbent solution. In this case, the search is re-centered around x'' and begins with the first neighborhood.

If the last neighborhood has been reached without finding a better solution than the incumbent, then the search begins again with the first neighborhood $\mathcal{N}_1(x)$ until a stopping criterion is satisfied. In our case, as stopping criterion we have chosen a maximum number of iterations since the last improvement.

4. Computational results

This section presents the obtained results for solving the BDTWNPP. The experiments were carried out on instances obtained using the randomly number generator Random.org.

The testing machine was an Intel Core i5-2450M and 4 GB RAM with Windows 7 as operating system. The greedy algorithm, GA and GA-VNS hybrid algorithm have been developed in Microsoft .NET Framework 4 using C #.

Based on preliminary computational experiments, we set the following genetic parameters: the size of the initial population consists of 50 individuals generated half randomly, tournament selection with groups of 7, one-point crossover, mutation probability 10% and maximum number of generations 100.

In Table 1, we present the obtained computational results using our proposed greedy algorithm, GA and GA-VNS hybrid algorithm. In our experiments, we performed 5 independent runs for each instance.

The first three columns in the table give the dimension of the instance: the interval from were have been selected the numbers and the number of pairs

Instances			Greedy algorithm		Results of GA		Results of GA-VNS	
Min	Max	# of pairs	Best sol.	Time	Best sol.	Time	Best sol.	Time
1	100	10	15 (15;15)	0.6	11 (3;11)	0.24	3 (3;3)	0.15
1	100	50	44 (44;38)	0.0	4 (4;4)	0.567	0 (0;0)	2.75
1	100	100	59 (59;53)	0.1	3 (3;1)	0.159	1 (1;1)	0.125
1	100	500	243 (228;243)	0.0	1 (0;1)	1.340	1 (0;1)	2.578
1	100	1000	48 (1;48)	0.31	1 (1;0)	6.140	1 (1;0)	6.234
1	1000	10	499 (499;337)	0.0	55 (55;25)	0.359	55 (55;25)	0.109
1	1000	50	285 (235;285)	0.0	269 (269;193)	0.156	15 (15;3)	4.672
1	1000	100	368 (368;330)	0.1	20 (12;20)	0.390	6 (4;6)	5.390
1	1000	500	548 (511;548)	0.0	6 (1;6)	1.734	2 (1;2)	24.266
1	1000	1000	926 (890;926)	0.31	4 (4;2)	7.672	2 (2;2)	39.236
1000	10000	10	736 (736;510)	0.0	2014 (2014;722)	0.343	736 (736;510)	0.93
1000	10000	50	1040 (97;1040)	0.0	1422 (293;1422)	0.234	54 (19;54)	4.765
1000	10000	100	2651 (708;2651)	0.0	227 (134;227)	6.78	23 (12;23)	4.140
1000	10000	500	17663 (17663;17233)	0.15	61 (61;59)	2.843	11 (3;11)	24.485
1000	10000	1000	7169 (7126;7169)	0.31	41 (10;41)	24.329	10 (10;9)	53.80
10000	1000000	10	468796 (387067;468796)	0.0	308916 (306299;308916)	1.93	129927 (129927;127386)	0.46
10000	1000000	50	298653 (298653;155068)	0.0	229024 (81191;229024)	0.359	7691 (7691;6272)	1.453
10000	1000000	100	796974 (771643;796974)	0.1	96800 (4525;96800)	1.109	3019 (3019;10)	6.140
10000	1000000	500	916103 (916103;906447)	0.15	5319 (5319;3875)	13.781	1771 (37;1771)	49.689
10000	1000000	1000	3456189 (3456189;3440327)	0.31	2689 (2357;2689)	17.735	2013 (1505;2013)	7.774
1000000	10000000	10	2356617 (2356617;747913)	0.0	2540301 (2540301;460699)	0.15	407361 (51473;407361)	0.14
1000000	10000000	50	2158720 (2158720;435126)	0.0	1511498 (1511498;848162)	0.140	48618 (48618;3624)	1.125
1000000	10000000	100	392774 (156892;392774)	0.0	322636 (152242;322636)	0.140	28888 (5420;28888)	5.844
1000000	10000000	500	32243919 (31950547;32243919)	0.11	104339 (3359;104339)	20.266	3671 (355;3671)	37.48
1000000	10000000	1000	27098586 (24681685;27098586)	0.31	69835 (69835;32814)	36.783	6677 (6677;4940)	52.867

considered, the next two columns provide the results and computational times obtained by using the greedy heuristic algorithm and the last four columns give the best solutions and required necessary computational times provided by the GA and the hybrid GA-VNS algorithm.

Analyzing the results presented in Table 1, we observe that our proposed hybrid GA-VNS heuristic algorithm performs favorable in terms of the solution quality in comparison with the GA alone and the greedy heuristic algorithm: in 22 out of 25 instances the hybrid GA-VNS provided the best solutions and in the other 3 instances we obtained the same solutions as those obtained using the GA alone. For all the considered instances, the solutions provided by GA and GA-VNS have better quality than the solutions provided by the greedy algorithm.

The running times of our GA and hybrid GA-VNS are proportional with the number of generations. From Table 1, we observe that the greedy algorithm is faster comparing to GA and GA-VNS approaches.

5. Conclusions

This paper deals with the bi-dimensional two-way number partitioning problem, where a set of pairs has to be partitioned into two subsets such that the sums of numbers in each subset should be equal or are close to be equal for both coordinates.

We developed three heuristics for solving the BDTWNPP: a greedy algorithm, a genetic algorithm and an efficient hybrid approach to the problem that combines the use of genetic algorithms (GA) and Variable Neighborhood Search (VNS). Some important features of our hybrid algorithm are:

- using a novel method for generating the initial population: partially randomly and partially based on the problem structure.
- elimination of the duplicate solutions from each population;
- using the VNS procedure along the GA in order to intensify the search within promising areas of the solution space.

The preliminary computational results show that our hybrid GA-VNS algorithm compares favorably in terms of the solution quality in comparison to the greedy algorithm and the genetic algorithm alone.

In the future, we plan to asses the the generality and scalability of the proposed hybrid heuristic by testing it on more instances and to apply it also in the case of multi-way number partitioning problem.

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.

References

[1] M.F. Arguello, T.A. Feo, O. Goldschmidt, *Randomized methods for the number partitioning problem*, Computers & Operations Research, Vol. 23, Issue 2, pp. 103-111, 1996.

[2] 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 Metaheuristics: Computer Decision-Making (M.G.C. Resende and J. Souza, editors), Kluwer, 2004.

[3] M. R. Garey, D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W.H. Freeman, New York, 1997.

[4] F. Glover, M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1997.

[5] E. Horowitz, S. Sahni, *Computing partitions with applications to the Knapsack problem*, Journal of ACM, Vol. 21, Issue 2, pp. 277-292, 1974.

[6] 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*, Operations Research, Vol. 39, Issue 3, pp. 378-406, 1991.

[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] J. Kojic, *Integer linear programming model for multidimensional two-way number partitioning problem*, Comput. Math. Appl., Vol. 60, pp. 2302-2308, 2010.

[9] N. Mladenovic and P. Hansen, *Variable neighborhood search*, Computers and Operations Research, Vol. 24, Issue 11, pp. 1097-1100, 1997.

[10] P.C. Pop and O. Matei, *A memetic algorithm approach for solving the multidimensional multi-way number partitioning problem*, Applied Mathematical Modelling (to appear).

[11] W. Ruml, J.T. Ngo, J. Marks, S.M. Shieber, *Easily searched encodings for number partitioning*, Journal of Optimization Theory and Applications, Vol. 89, Issue 2, pp. 251-291, 1996.

(1) Indeco Soft, 5 Magnoliei St., 430094 Baia Mare, Romania
E-mail address: levi.fuksz@yahoo.com

(2) Department of Mathematics and Computer Science, North University Center of Baia Mare, Technical University of Baia Mare, 76 Victoriei St., 430122 Baia Mare, Romania
E-mail address: petrica.pop@ubm.ro

E-mail address: ioanazelina@yahoo.com