PUBLICATIONS DE L'INSTITUT MATHÉMATIQUE
Nouvelle série, tome 92(106) (2012), 25–34 DOI: 10.2298/PIM1206025L

AN INTEGER LINEAR PROGRAMMING FORMULATION AND GENETIC ALGORITHM FOR THE MAXIMUM SET SPLITTING PROBLEM

Bojana Lazović, Miroslav Marić, Vladimir Filipović, and Aleksandar Savić

Communicated by Žarko Mijajlović

Abstract. We consider the maximum set splitting problem (MSSP). For the first time an integer linear programming (ILP) formulation is presented and validity of this formulation is given. We propose a genetic algorithm (GA) that uses the binary encoding and the standard genetic operators adapted to the problem. The overall performance of the GA implementation is improved by a caching technique. Experimental results are performed on two sets of instances from the literature: minimum hitting set and Steiner triple systems. The results show that CPLEX optimally solved all hitting set instances up to 500 elements and 10000 subsets. Also, it can be seen that GA routinely reached all optimal solutions up to 500 elements and 50000 subsets. The Steiner triple systems seems to be much more challenging for maximum set splitting problems since the CPLEX solved to optimality, within two hours, only two instances up to 15 elements and 35 subsets. For these instances GA reached all solutions as CPLEX but in much smaller running time.

1. Introduction

Let Ω be a finite set with cardinality $m = |\Omega|$ and a family of subsets be given $S_1, \dots, S_n \subseteq \Omega$. A partition of Ω is a pair of subsets (P_1, P_2) of Ω such that $P_1 \cap P_2 = \emptyset$ and $P_1 \cup P_2 = \Omega$. We say that a subset S_j of Ω is split by the partition (P_1, P_2) of Ω if S_j intersects with both P_1 and P_2 (i.e., $S_i \cap P_1 \neq \emptyset$, $S_i \cap P_2 \neq \emptyset$). For the partition (P_1, P_2) let us denote $\mathrm{Obj}(P_1, P_2)$ as a number of subsets which are split. Now, the maximum set splitting problem (MSSP) can be formulated as finding $\max \mathrm{Obj}(P_1, P_2)$ over all partitions of Ω. Weighted maximum set splitting problem can be similarly defined by finding $\max \sum_{j=1,\, S_j \text{ is split}}^{n} w_j$.

Let us demonstrate properties of MSSP on two little illustrative examples.

2010 *Mathematics Subject Classification:* Primary 90C10, 90C59; Secondary 65K05.
Key words and phrases: genetic algorithm, set splitting, Steiner triple systems.
Partially supported by Serbian Ministry of Science, grant 174010.

EXAMPLE 1.1. Let our first set consist of ten elements ($m = 10$) and four subsets ($n = 4$). The subsets are: $S_1 = \{1, 2, 4, 7, 9\}$; $S_2 = \{3, 4, 5, 7, 9, 10\}$; $S_3 = \{2, 8, 9, 10\}$, $S_4 = \{2, 3, 4, 6, 7, 8, 9, 10\}$. One of the optimal solutions is the partition (P_1, P_2), $P_1 = \{1, 2, 7, 9\}$; $P_2 = \{3, 4, 5, 6, 8, 10\}$. The optimal objective value is 4 because $S_1 \cap P_1 = \{1, 2, 7, 9\}$; $S_1 \cap P_2 = \{4\}$; $S_2 \cap P_1 = \{7, 9\}$; $S_2 \cap P_2 = \{3, 4, 5, 10\}$; $S_3 \cap P_1 = \{2, 9\}$; $S_3 \cap P_2 = \{8, 10\}$; $S_4 \cap P_1 = \{2, 7, 9\}$; $S_4 \cap P_2 = \{3, 4, 6, 8, 10\}$.

EXAMPLE 1.2. Let our second set consist of three elements ($m = 3$) and three subsets ($n = 3$). The subsets are: $S_1 = \{1, 2\}$; $S_2 = \{1, 3\}$; $S_3 = \{2, 3\}$. One of the optimal solutions is the partition (P_1, P_2), $P_1 = \{1, 2\}$; $P_2 = \{3\}$. The optimal objective value is 2 because $S_1 \cap P_1 = \{1, 2\}$; $S_1 \cap P_2 = \emptyset$; $S_2 \cap P_1 = \{1\}$; $S_2 \cap P_2 = \{3\}$; $S_3 \cap P_1 = \{2\}$; $S_3 \cap P_2 = \{3\}$.

The MSSP, as well as weighted variant of the problem, is NP-hard in general [**10**]. Even more, the variant of the problem, when all subsets in the family are of size exactly k, is also NP-hard for $k \geqslant 2$. Also, the approximation of the MSSP with $k = 3$, within a factor greater than $\frac{11}{12}$, is NP-hard [**11**].

As can be seen in [**2**,**3**], a probabilistic approach is used for developing of deterministic kernelization algorithm for the maximum set splitting problem. Running time of a subset partition technique is bounded by $O^*(2^k)$. That algorithm can be de-randomized, which leads to a deterministic parameterized algorithm of running time $O^*(4^k)$ for the weighted maximum set splitting problem, and gives the first proof that the problem is fixed-parameter tractable. The kernelization technique is also used in [**5**, **6**].

In [**1**] is given the first quadratic integer formulation of the MSSP. Semidefinite programming (SDP) relaxation of that formulation was used for constructing 0.724-approximation algorithm of the MSSP. Slightly better, 0.7499-approximation algorithm, given in [**23**], is based on strengthened SDP relaxation, improved rounding method, and tighter analysis. The quadratic integer formulation, used in SDP relaxation, is given as follows.

$$\max \sum_{j=1}^{n} z_j \quad \text{subject to}$$

$$\frac{1}{|S_j| - 1} \sum_{\substack{i_1, i_2 \in S_j \\ i_1 \neq i_2}} \frac{1 - y_{i_1} y_{i_2}}{2} \geqslant z_j, \qquad \text{for every } j = 1, \ldots, n$$

$$z_j \in \{0, 1\}, \quad \text{for every } j = 1, \ldots, n$$

$$y_i \in \{-1, 1\}, \quad \text{for every } i = 1, \ldots, m$$

2. An integer linear programming formulation

It is useful to formulate, when it is possible, discrete optimization problems as integer or mixed integer programming models, in order to use different well-known optimization techniques for their solving [**16**, **17**, **21**]. Following that idea we have used the CPLEX solver on the new integer linear programming formulation for the MSSP described below.

Let us define parameters and variables:

$$s_{ij} = \begin{cases} 1, & i \in S_j, \\ 0, & i \notin S_j, \end{cases} \quad i = 1, \ldots, m, \quad j = 1, \ldots, n \tag{2.1}$$

$$x_i = \begin{cases} 1, & i \in P_1, \\ 0, & i \in P_2, \end{cases} \quad i = 1, \ldots, m \tag{2.2}$$

$$y_j = \begin{cases} 1, & S_j \text{ is split}, \\ 0, & S_j \text{ is not split}, \end{cases} \quad j = 1, \ldots, n \tag{2.3}$$

Now, an integer linear programming model can be formulated as follows:

$$\max \sum_{j=1}^{n} y_j \tag{2.4}$$

$$y_j \leqslant \sum_{i=1}^{m} s_{ij} x_i, \qquad \text{for every } j = 1, \ldots, n \tag{2.5}$$

$$y_j + \sum_{i=1}^{m} s_{ij} x_i \leqslant |S_j|, \qquad \text{for every } j = 1, \ldots, n \tag{2.6}$$

$$y_j \in \{0, 1\}, \qquad \text{for every } j = 1, \ldots, n \tag{2.7}$$

$$x_i \in \{0, 1\}, \qquad \text{for every } i = 1, \ldots, n \tag{2.8}$$

As it can be seen, there are $m + n$ binary variables and $2n$ constraints.

The following lemmas show that the solution of this ILP formulation is the solution of the MSSP. First of all, we can define $\mathrm{Obj}_{\mathrm{ILP}}(x, y) = \sum_{j=1}^{n} y_j$ subject to (2.5)–(2.8).

LEMMA 2.1. *Let* $\mathrm{Obj}(P_1, P_2)$ *be a partition of* Ω. *Then there is a solution* (x, y) *of system* (2.5)–(2.8) *such that* $\mathrm{Obj}_{\mathrm{ILP}}(x, y) \geqslant \mathrm{Obj}(P_1, P_2)$.

PROOF. Let variables (x, y) be defined as (2.1)–(2.3). We will prove that these variables satisfy system (2.5)–(2.8) and $\mathrm{Obj}_{\mathrm{ILP}}(x, y) \geqslant \mathrm{Obj}(P_1, P_2)$.

According to the definition of variables y_j, we have: S_j is split $\Leftrightarrow y_j = 1$. Therefore, the number of splitting sets S_j is equal to $\sum_{j=1}^{n} y_j$ which implies $\mathrm{Obj}_{\mathrm{ILP}}(x, y) \geqslant \mathrm{Obj}(P_1, P_2)$. Constraints (2.7) and (2.8) are obviously satisfied from definitions (2.2) and (2.3).

If S_j is not split, then $y_j = 0$. In that case $0 = y_j \leqslant \sum_{i=1}^{m} s_{ij} x_i$, because s_{ij} and x_i are nonnegative, so constraints (2.5) are satisfied. Also we have $y_j + \sum_{i=1}^{m} s_{ij} x_i = \sum_{i=1}^{m} s_{ij} x_i \leqslant \sum_{i=1}^{m} s_{ij} = |S_j|$ and with that, constraints (10) are satisfied.

In the other case, if S_j is split, then $y_j = 1$, $S_j \cap P_1 \neq \emptyset$ and $S_j \cap P_2 \neq \emptyset$. Therefore, $(\exists u)(u \in S_j \wedge u \in P_1)$ and $(\exists v)(v \in S_j \wedge v \in P_2)$ imply $s_{uj} = 1$, $x_u = 1$, $s_{vj} = 1$, $x_v = 0$. The next step shows that we have $y_j = 1 = s_{uj} x_u \leqslant \sum_{i=1}^{m} s_{ij} x_i$, which satisfies constraints (2.5). As previously implied, there is

$$\sum_{i=1}^{m} s_{ij} x_i = x_v s_{vj} + \sum_{i=1,\, i \neq v}^{m} s_{ij} x_i = \sum_{i=1,\, i \neq v}^{m} s_{ij} x_i \leqslant \sum_{i=1,\, i \neq v}^{m} s_{ij} = |S_j| - 1$$

which implies $y_j + \sum_{i=1}^m s_{ij}x_i = 1 + \sum_{i=1}^m s_{ij}x_i \leqslant |S_j|$, so constraints (2.6) are satisfied. $\square$

LEMMA 2.2. *If* (x, y) *is the solution of system* (2.5)–(2.8)*, then there exists a partition* $\mathrm{Obj}(P_1, P_2)$ *of* Ω *such that* $\mathrm{Obj}(P_1, P_2) \geqslant \mathrm{Obj}_{\mathrm{ILP}}(x, y)$.

PROOF. Define $P_1 = \{i \mid x_i = 1 \wedge i \in \Omega\}$ and P_2 as its complement. Now, we should prove $y_j = 1 \Rightarrow S_j$ is split. From constraints (2.5), we have

$$\begin{aligned} y_j = 1 \Rightarrow \sum_{i=1}^m s_{ij}x_i \geqslant 1 &\Rightarrow (\exists u)(s_{uj} = 1 \wedge x_u = 1) \\ &\Rightarrow (\exists u)(u \in S_j \wedge u \in P_1) \Rightarrow S_j \cap P_1 \neq \emptyset. \end{aligned}$$

Constraints (2.6) imply

$$\begin{aligned} y_j = 1 \Rightarrow y_j + \sum_{i=1}^m s_{ij}x_i \leqslant |S_j| \Rightarrow \sum_{i=1}^m s_{ij}x_i \leqslant |S_j| - 1 \\ \Rightarrow (\exists v)(s_{vj} = 1 \wedge x_v = 0) \Rightarrow (\exists v)(v \in S_j \wedge v \in P_2) \Rightarrow S_j \cap P_2 \neq \emptyset. \end{aligned}$$

From above it follows $y_j = 1 \Rightarrow (S_j \cap P_1 \neq \emptyset \wedge S_j \cap P_2 \neq \emptyset) \Rightarrow S_j$ is split. Therefore, $\sum_{j=1}^n y_j$ is greater than or equal to the number of splitting sets S_j, which directly implies $\mathrm{Obj}(P_1, P_2) \geqslant \mathrm{Obj}_{\mathrm{ILP}}(x, y)$. $\square$

Now we are ready to state our main theoretical result.

THEOREM 2.1. *Let be given a family of subsets* $S_1, \ldots, S_n \subseteq \Omega$ *and a partition* (P_1, P_2)*. Let variables* s, x *and* y *be defined by* (2.1)–(2.3)*. Then partition* (P_1, P_2) *splits the maximum number of subsets* S_j*,* $j = 1, \ldots, n$ *if and only if there is an optimal solution* (x, y) *of* (2.4)–(2.8)*.*

PROOF. The direction $\Rightarrow$ can be easily deduced from Lemma 1. The reverse direction follows from Lemma 2. $\square$

Let us demonstrate CPLEX behavior on MSSP instances from Examples 1 and 2.

EXAMPLE 2.1. One possible optimal solution of MSSP instance from Example 1 is $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 0$, $x_5 = 0$, $x_6 = 0$, $x_7 = 1$, $x_8 = 0$, $x_9 = 1$, $x_{10} = 0$, $y_1 = 1$, $y_2 = 1$, $y_3 = 1$, $y_4 = 1$. Objective value is 4.

EXAMPLE 2.2. One possible optimal solution of MSSP instance from Example 2 is $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $y_1 = 0$, $y_2 = 1$, $y_3 = 1$. Objective value is 2.

Note that, integer linear programming formulation of the weighted maximum set splitting problem is given by $\max \sum_{j=1}^n w_jy_j$ with constrains (2.5)–(2.8).

3. Genetic algorithm

Genetic algorithms are stochastic search techniques which imitate some spontaneous optimization processes in the natural selection and reproduction. At each iteration (generation) GA manipulates a set (population) of encoded solutions (individuals), starting from either randomly or heuristically generated one. Individuals from the current population are evaluated using a fitness function to determine their qualities. Good individuals are selected to produce the new ones (offspring), applying operators inspired from those of genetics (crossover and mutation), and they replace some of the individuals from the current population. A detailed description of GA is out of this paper's scope and it can be found in [**18**]. Extensive computational experience on various optimization problems shows that GA often produces high quality solutions in a reasonable time, as can be seen from the following recent applications [**7**, **13**–**15**, **19**, **20**, **22**].

In this section we shall describe a GA implementation for determining the maximal splitting set.

The binary encoding of the individuals used in this implementation is the following. Each partition (P_1, P_2) is naturally represented in the population by a binary string of length m. Digit 1 at the i-th place of the string denotes that an element i belongs to P_1, while 0 shows it is in P_2. For partition determined in this way, it is easy to check for every subset if it is split. Objective value is the number of split subsets.

The fitness f_{ind} of individual ind is computed by scaling objective values $\mathrm{obj}_{\mathrm{ind}}$ of all individuals (there are N_{pop} individuals in population) into the interval [0,1]. Therefore, the best individual $\mathrm{ind}_{\max}$ has fitness 1 and the worst one $\mathrm{ind}_{\min}$ has fitness 0. More precisely, $f_{\mathrm{ind}} = \frac{\mathrm{obj}_{\mathrm{ind}} - \mathrm{obj}_{\mathrm{ind}_{\min}}}{\mathrm{obj}_{\mathrm{ind}_{\max}} - \mathrm{obj}_{\mathrm{ind}_{\min}}}$. Next, individuals are arranged in nonincreasing order of their fitness: $f_1 \geqslant f_2 \geqslant \cdots \geqslant f_{N_{\mathrm{pop}}}$.

Usually GAs have relatively small number of elite individuals, because they twice have a chance to pass into the next generation: once through selection operator and once as elite individuals. Such common practice is not adequate for our purpose. In order to obtain satisfactory results of our GA implementation, we need sufficient number of elite individuals to preserve good solutions for exploitation as well as a sufficient number of nonelite individuals for exploration. To prevent an undeserved domination of N_{elite} elite individuals over the population, their fitness are decreased by the following formula:

$$(3.1) \qquad f_{\mathrm{ind}} = \begin{cases} f_{\mathrm{ind}} - \bar{f}, & f_{\mathrm{ind}} > \bar{f}, \\ 0, & f_{\mathrm{ind}} \leqslant \bar{f}; \end{cases} \qquad 1 \leqslant \mathrm{ind} \leqslant N_{\mathrm{elite}}; \quad \overline{f} = \frac{1}{N_{\mathrm{pop}}} \sum_{\mathrm{ind}=1}^{N_{\mathrm{pop}}} f_{\mathrm{ind}}$$

In this way, even nonelite individuals preserve their chance to survive to the next generation. This approach gives a possibility to allow high elitism without too high selection pressure and thus too much exploitation in the algorithm. Such elitist strategy is applied to N_{elite} elite individuals, which are directly passing to the next generation. The genetic operators are applied to the rest of the population ($N_{\mathrm{nnel}} = N_{\mathrm{pop}} - N_{\mathrm{elite}}$ nonelite individuals). The objective value of elite individuals are the same as in the previous generation, so they are calculated only once and this provides significant time savings.

Duplicated individuals, i.e., individuals with the same genetic code are redundant. In order to prevent them to enter the next generation their fitness values are set to zero, except for the first occurrence. Individuals with the same objective value, but different genetic codes, in some cases may dominate in the population by number, which implies that the other individuals with potentially good genes are rare. For this reason, it is useful to limit the number of their appearance to some constant N_{rv}. This is a very effective technique for saving the diversity of the genetic material and keeping the algorithm away from a premature convergence. It consists of two steps for every individual in the population:

Step 1: Check whether the genetic code of the current individual ind is identical with the genetic code of any of the individuals from 1 to ind -1. If the answer is positive, set the fitness of ind to 0. Otherwise go to Step 2;

Step 2: Count the number of the individuals from 1 to ind -1 which did not get fitness 0 in Step 1 and which have the same objective value as ind. If it is greater than or equal to N_{rv}, set the fitness of ind to 0.

The selection operator chooses the individuals that will produce offspring in the next generation, according to their fitness. Low fitness-valued individuals have less chance to be selected than high fitness-valued ones. In the standard tournament scheme, one tournament is performed for every nonelitist individual. The tournament size is a given parameter and tournament candidates are randomly chosen from the current population. Only the winner of the tournament, i.e., a tournament candidate with the best fitness participates in the crossover. So, the selection operator (tournament) is applied N_{nnel} times on the set of all N_{pop} individuals in the population to choose the N_{nnel} parents for crossover. The same individual from the current generation can participate in several tournaments. The standard tournament selection uses an integral tournament size, which in some cases can be a limiting factor.

We use an improved tournament selection operator, known as the fine-grained tournament selection-FGTS, proposed in [**8**]. This operator uses a real (rational) parameter F_{tour} which denotes the desired average tournament size. The first type of tournaments is held k_1 times and its size is $\lfloor F_{\text{tour}} \rfloor$, while the second type is performed k_2 times with $\lceil F_{\text{tour}} \rceil$ individuals participated, so $F_{\text{tour}} \approx \frac{k_1 \lfloor F_{\text{tour}} \rfloor + k_2 \lceil F_{\text{tour}} \rceil}{N_{\text{nnel}}}$.

In [**8**, **20**] extensive numerical experiments for different optimization problems have indicated that FGTS with $F_{\text{tour}} = 5.4$ gives the best results. So, in this implementation we adopted that value as a reasonable choice. The running time for FGTS operator is $O(N_{\text{nnel}} \cdot F_{\text{tour}})$. In practice F_{tour} and N_{nnel} are considered to be constant (not depending on n) that gives a constant running time complexity. For detailed information about FGTS see [**8**].

In the crossover operator all nonelitist individuals chosen to produce offspring for the next generation are randomly paired for crossover in $\lfloor N_{\text{nnel}}/2 \rfloor$ pairs. After a pair of parents is selected, a crossover operator is applied to them producing two offspring. The operator we use in this GA implementation is the one-point crossover. This operator is performed by exchanging segments of two parents' genetic codes starting with a randomly chosen crossover point. The crossover operator is realized with probability $p_{\rm cross} = 0.85$. It means that approximately 85% pairs of individuals exchange their genetic material.

The standard simple mutation operator is performed by changing a randomly selected gene in the genetic code of the individual, with a certain mutation rate. During the GA execution it may happen that all individuals in the population have the same gene on a certain position. This gene is called frozen. If the number of frozen genes is l, the search space becomes 2^l times smaller and the possibility of a premature convergence rapidly increases. The crossover operator can not change the bit value of any frozen gene and the basic mutation rate is often too small to restore lost subregions of the search space. On the other hand, if the basic mutation rate is increased significantly, a genetic algorithm becomes a random search.

For this reason, the simple mutation operator is modified such that the mutation rate is increased only on frozen genes. In this implementation the mutation rate for frozen genes is 2.5 times higher $(1.0/n)$, comparing to nonfrozen ones $(0.4/n)$. In each generation, we determine positions where all individuals have a given gene fixed and define them as frozen genes. Obviously the set of frozen genes is not fixed, i.e., it may change during the generations.

The initial population is randomly generated, providing the maximal diversity of the genetic material. That function also computes values of all the individuals of the population.

In order to obtain satisfactory results of our GA implementation, we need a sufficient number of elite individuals to preserve good solutions for the exploitation as well as a sufficient number of nonelite individuals for the exploration. The population size of $N_{\rm pop} = 150$ individuals with $N_{\rm elite} = 100$ elite and $N_{\rm nnel} = 50$ nonelite individuals is a good compromise between exploitation and exploration part of GA search. The corresponding values in k_1 and k_2 in FGTS are then 20 and 30, respectively. The maximal allowed number of individuals with the same objective value is $N_{rv} = 40$.

The run-time performance of GA is optimized by a caching technique. The main idea is to avoid computing the same objective value every time when genetic operators produce individuals with the same genetic code. The evaluated objective values are stored in a hash-queue data structure using the least recently used (LRU) caching technique. When the same code is obtained again, its objective value is taken from the cache memory, that provides time-savings. In this implementation the number of individuals stored in the cache memory is limited to 5000. For detailed information about caching GA see [**12**].

4. Experimental results

All computations were executed on 2.5 GHz single processor PC computer with 1 Gb RAM under Windows operating system. For experimental testings, we used hitting set instances from [**4**]. Those instances include different numbers of elements $(m = 50, 100, 250, 500)$ and different numbers of subsets $(n = 100, 10000, 50000)$.

In order to show effectiveness of the proposed ILP formulation, we tested it on those instances by using CPLEX 10.1 solver. These results are compared with GA solutions.

Results obtained by CPLEX and GA are given in Table 1. In the first and second column there are the number of elements m and the number of subsets n, respectively. The third column contains optimal solutions which were obtained by CPLEX in case when the method finished its work. In the fourth and fifth columns, the value and running time of CPLEX are given, respectively. There was a time limitation of 7.200 seconds, approximately. The mark "opt" is written if CPLEX finished its work and produced optimal solution. The sixth and seventh columns consider results of GA, and they are presented in the same way as for the CPLEX. For the last instance GA reached the solution with objective value of 50000. Since the overall number of subsets is 50000, it is easy to see that this solution is optimal.

TABLE 1. Results on the hitting set instances

m	n	opt	CPLEX		GA	
			sol	t (sec)	sol	t (sec)
50	1000	1000	opt	0.078	opt	2.582
50	10000	10000	opt	3.265	opt	60.039
100	1000	1000	opt	0.188	opt	4.67
100	10000	10000	opt	8.297	opt	168.603
100	50000	50000	opt	155.203	opt	683.147
250	1000	1000	opt	0.219	opt	8.626
250	10000	10000	opt	30.063	opt	336.894
500	1000	1000	opt	0.500	opt	13.325
500	10000	10000	opt	106.094	opt	437.909
500	50000	50000	out of memory		opt	2086.517

As can be seen from Table 1, CPLEX on the proposed ILP formulation, routinely found optimal solution for all hitting set instances, except the largest one, which is solved by GA. Furthermore, from Table 1, it is clear that all subsets are split. From these facts we can conclude that the hitting set instances from [**4**] are easy for maximal set splitting problem. Therefore, running time for GA is greater than for CPLEX, because of its robustness.

In order to check the effectiveness of both approaches on harder instances, we tested them on set covering instances derived from Steiner triple systems [**9**], and results are presented in Table 2. The data are presented in a similar way as in Table 1, adding column named 'ub' which contains the upper bound of the solution in case where CPLEX has not finished its work in time limitation of 7200 seconds.

The results from the Table 2 clearly demonstrate that instances derived from Steiner triple systems are challenging for maximum set splitting problem. CPLEX program, based on previous formulation, optimally solved only two smallest instances up to $m = 15$ and $n = 35$. All other instances from that collection are out of reach for exact solving by CPLEX within 2 hours of the running time. GA reached all solutions as CPLEX but in much smaller running time.

TABLE 2. Results on the Steiner triple systems

m	n	opt	CPLEX			GA	
			sol	ub	t (sec)	sol	t (sec)
9	12	10	opt		0.031	opt	0.193
15	35	28	opt		0.343	opt	0.233
27	117		91	93	7200	91	0.382
45	330		253	302	7200	253	0.914
81	1080		820	1058	7200	820	2.893
135	3015		2278	3001	7200	2278	7.858
243	9801		7381	9794	7200	7381	65.409

5. Conclusion

This paper is devoted to the maximum set splitting problem. We introduced its integer linear programming formulation. Also, we proved the correctness of the corresponding formulation. Numbers of variables and constraints were relatively small compared to the dimension of the problem.

Additionally, an evolutionary metaheuristic for solving the maximum set splitting problem is presented. The binary representation, mutation with frozen genes, limited number of different individuals with the same objective value and the caching technique were used.

We carried out numerical experiments using two data sets proposed from the literature. Numerical results showed that both CPLEX solver, based on this ILP formulation, and the genetic algorithm, produced very good solutions. On harder instances GA well performed and obtained solutions more quickly than CPLEX, because of its heuristic nature.

Our work can be extended in several ways. It would be desirable to investigate the application of an exact method using the proposed ILP formulation. Also, it should be directed to parallelization of the presented genetic algorithm and testing on more powerful multiprocessor computer systems.

References

1. G. Andersson, L. Engebretsen, *Better approximation algorithms for set splitting and not-all-equal sat*, Inform. Process. Lett. **65** (1998), 305–311.
2. J. Chen, S. Lu, *Improved algorithm for weighted and unweighted set splitting problems*, Lect. Notes Comp. Sci. **4598** (2007), 573–547.
3. H. Chen, S. Lu, *Improved parameterized set splitting algorithms: A probabilistic approach*, Algorithmica **54** (2009), 472–489.
4. V. Cutello, G. Nicosia, *A clonal selection algorithm for coloring, hitting set and satisfiability problems*, Lect. Notes Comp. Sci. **3931** (2006), 324–337. http://www.dmi.unict.it/~nicosia/cop.html
5. F. Dehne, M. Fellows, F. Rosamond, *An FPT algorithm for set splitting*, Lect. Notes Comp. Sci. **2880** (2003), 180–191.
6. F. Dehne, M. Fellows, F. Rosamond, P. Shaw, *Greedy localization, iterative compression, modeled crown reductions: New FPT techniques, and improved algorithm for set splitting, and a novel 2k kernelization of vertex cover*, Lect. Notes Comp. Sci. **3162** (2004), 127–137.

7. B. Djurić, J. Kratica, D. Tošić, V. Filipović, *Solving the maximally balanced connected partition problem in graphs by using genetic algorithm*, Comput. Inform. **27**(3), (2008), 341–354.
8. V. Filipović, J. Kratica, D. Tošić, I. Ljubić, *Fine grained tournament selection for the simple plant location problem*, in: *Proceedings of the 5th Online World Conference on Soft Computing Methods in Industrial Applications – WSC5*, 2000, 152–158.
9. D. R. Fulkerson, G. L. Nemhauser, L. E. Trotter, *Two computationally difficult set covering problems that arise in computing the l-width of incidence matrices of steiner triple systems*, Math. Prog. Study **2** (1974), 72–81. http://www.research.att.com/~mgcr/data/steiner-triples.tar.gz
10. M. Garey, D. Johnson, *Computers and Intractability: a Guide to the Theory of NP-completeness*, Freeman, San Francisco 1979.
11. V. Guruswami, *Inapproximability results for set splitting and satisfiability problems with no mixed clauses*, Algorithmica **38** (2004), 451–469.
12. J. Kratica, *Improving performances of the genetic algorithm by caching*, Comput. Artif. Intell. **18** (1999), 271–283.
13. J. Kratica, V. Kovačević-Vujčić, M. Čangalović, *Computing strong metric dimension of some special classes of graphs by genetic algorithms*, Yugoslav. J. Oper. Res. **18**(2) (2008), 43–51.
14. J. Kratica, M. Čangalović, V. Kovačević-Vujčić, *Computing minimal doubly resolving sets of graphs*, Comput. Oper. Res. **36**(7) (2009), 2149–2159.
15. J. Kratica, V. Kovačević-Vujčić, M. Čangalović, *Computing the metric dimension of graphs by genetic algorithms*, Comput. Optim. Appl. **44**(2) (2009), 343–361.
16. R. H. Kwon, G. V. Dalakouras, C. Wang, *On a posterior evaluation of a simple greedy method for set packing*, Optim. Lett. **2** (2008), 587–597.
17. L. Liberti, N. Maculan, Y. Zhang, *Optimal configuration of gamma ray machine radiosurgery units: the sphere covering subproblem*, Optim. Lett. **3** (2009), 109–121.
18. M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, Massachusetts, 1999.
19. A. Paszynska, M. Paszynski, *Application of a hierarchical chromosome based genetic algorithm to the problem of finding optimal initial meshes for the self-adaptive hp-FEM*, Comput. Inform. **28**(2) (2009), 209–223.
20. Z. Stanimirović, J. Kratica, Dj. Dugošija, *Genetic algorithms for solving the discrete ordered median problem*, Eur. J. Oper. Res. **182**(3) (2007), 983–1001.
21. C. Wang, M. T. Thai, Y. Li, F. Wang, W. Wu, *Optimization scheme for sensor coverage scheduling with bandwidth constraints*, Optim. Lett. **3** (2009), 63–75.
22. B. Yuan, M. Orlowska, S. Sadiq, *Extending a class of continuous estimation of distribution algorithms to dynamic problems*, Optim. Lett. **2** (2008), 433–443.
23. J. Zhang, Y. Ye, Q. Han, *Improved approximations for max set splitting and max NAE SAT*, Discrete Appl. Math. **142** (2004), 133–149.

Faculty of Mathematics
University of Belgrade
Belgrade
Serbia
boka.lazovic@gmail.com
maricm@matf.bg.ac.rs
vladaf@matf.bg.ac.rs
aleks3rd@gmail.com

(Received 02 07 2010)