Serdica J. Computing **6** (2012), 369–384

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

A VARIABLE NEIGHBORHOOD SEARCH APPROACH FOR SOLVING THE MAXIMUM SET SPLITTING PROBLEM

Dragan Matić

Abstract. This paper presents a Variable neighbourhood search (VNS) approach for solving the Maximum Set Splitting Problem (MSSP). The algorithm forms a system of neighborhoods based on changing the component for an increasing number of elements. An efficient local search procedure swaps the components of pairs of elements and yields a relatively short running time. Numerical experiments are performed on the instances known in the literature: minimum hitting set and Steiner triple systems. Computational results show that the proposed VNS achieves all optimal or best known solutions in short times. The experiments indicate that the VNS compares favorably with other methods previously used for solving the MSSP.

1. Introduction. Let C be a collection of subsets of the finite set S. The Set Splitting Problem (SSP) is to determine if there exists a partition of the set S into two subsets P_1 and P_2, such that each element of the collection C has a non-empty intersection with each set P_1 and P_2, i.e., no element from C

ACM Computing Classification System (1998): I.2.8.

Key words: variable neighborhood search, combinatorial optimization, maximum set splitting problem, 2-coloring of the hypergraph, Steiner triple systems.

completely belongs to one partition set P_1 or P_2. The optimization version of the problem is called Maximum Set Splitting Problem (MSSP), where the objective is to find the partition of the set S, such that the number of elements from C with a non-empty intersection with both P_1 and P_2 is maximized. The subsets from C, with this property, are called split subsets. In the weighted variant of the problem, the subsets in C have weights, so the objective is to maximize the total weight of the split subsets.

This well-known problem was proven to be NP-hard by L. Lovász in [17], where the author investigated some invariants of the hypergraphs. A hypergraph is a generalization of a graph, where, for a given set V of vertices, more (or fever) than two vertices can be incident with one "edge" (such "edge" is called a hyperedge). The problem of finding the minimum number of colors to color the vertices in such a way, that no hyperedge is contained in any color class is proven to be as hard as determining the chromatic number in a graph. A special case is coloring the vertices in exactly two colors and the maximization variant of this problem is called Max Hypergraph 2-Coloring. More precisely, 2-Coloring of the hypergraph is to find if all vertices from V can be colored in two colors (for example white and black), such that each hyperedge has at least one vertex of each color. The Max Hypergraph 2-Coloring problem is to maximize the number of hyperedges containing at least one white and at least one black vertex.

It is well known that MSSP and 2-Coloring of the hypergraph are the same problems. Indeed, for a given set V of vertices of a hypergraph, each hyperedge defines a subset of V, containing the elements incident with the hyperedge. Then the set of all hyperedges is in fact one collection C of the subsets of V. Thus, to determine the 2-Coloring of vertices that maximizes the number of hyperedges containing vertices of both colors is the same task as to find a partition of the set V such that each subset contains vertices in both partitions.

A special case of the MSSP is also the well-known Max Cut problem. For a given graph $G = (V, E)$, the Max Cut problem is to find the partition of the set V into two subsets V_1 and V_2 such that the number of edges with the ends in different partitions is maximized. So, we can say that the Max Cut problem is a case of Max Hypergraph 2-Coloring if each hyperedge is incident with exactly two vertices, or a case of MSSP when all sets in C have cardinality 2.

Example 1. Let $S = \{1, 2, 3, 4, 5, 6, 7\}$ and $C = \{S_1, S_2, S_3, S_4, S_5\}$, $S_1 = \{1, 2, 3, 7\}$, $S_2 = \{2, 3\}$, $S_3 = \{2, 3, 4, 5\}$, $S_4 = \{4, 5, 7\}$, $S_5 = \{1, 6, 7\}$. Let us take a look on Figure 1. The figure shows a hypergraph, where the vertices $\{v_1, v_2, \ldots, v_7\}$ corresponds to the elements of S, and the hyperedges to the subsets form C. The vertices v_2 and v_7 are colored white, and the others

black. From the figure, it can be perceived that each hyperedge (each subset) contains at least one vertex (element) of each color. So, the partition $(P_1, P_2) = (\{2, 7\}, \{1, 3, 4, 5, 6\})$ is optimal, because each subset contains at least one element from each partition.

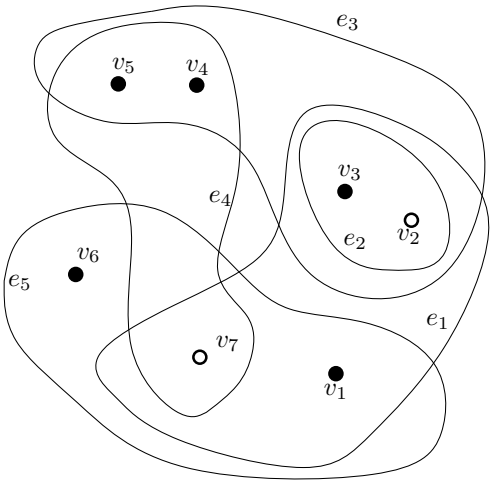

Fig. 1. An example of a hypergraph

2. Previous work. In recent years, the MSSP and (Max) Hypergraph 2-Coloring problems, as well as their variants and similar problems, have been widely investigated by many researches, both from combinatorial and algorithmic viewpoints. Recall that the first proof that the problem is NP-hard is given in [17] and the problem remains NP-hard even if all subsets have cardinality are less or equal to 3 [9]. Furthermore, the decision variant of the problem is NP complete, because it is obvious that any solution can be verified in polynomial time. Even if all subsets in the collection are of fixed size k, $k \geq 2$, the problem is also NP-hard. The MSSP is also APX-complete, i.e., cannot be approximated in polynomial time within a factor greater than 11/12 [10]. A simplified variant of the problem is the case when each element of C has at most two elements. The decision variant then becomes a "graph 2 colorability problem" which can be solved in polynomial time, for example by using the well known depth first search algorithm.

In [1], the first quadratic integer formulation (QIF) is introduced and is used for constructing the 0.724-approximation algorithm of the MSSP. In [21],

the authors improved that formulation by adding some new valid inequalities and by improving the rounding method. By the improvement, a slightly better, 0.7499-approximation is reached.

Several results related to the kernelization method have been improving the upper bounds of the algorithm's running times. Starting with the running time $O(72^k N^{O(1)})$ and $O(8^k N^{O(1)})$ in [6] and [7], in [15] the improvement is obtained with the value $O(2.65^k N^{O(1)})$. Finally, in [3, 4] a randomized algorithm with the running time $O(2^k + N)$ for a weighted version of problem is provided and in [16] the result $O(1.96^k + N)$ is presented.

A DNA-based algorithm for solving the MSSP is proposed in [2]. In order to construct solution spaces of DNA strands in SSP, the authors used so called sticker based model for DNA computation. To develop a DNA based algorithm, biological operations were applied to the Adleman-Lipton model. The proposed method proves that molecular computing can be an useful tool for solving this and also other NP-hard problems.

Recent notable results in solving MSSP are presented in paper [14]. The authors introduced the first integer linear formulation (ILP) and proposed a genetic algorithm (GA) for solving MSSP. Based on the ILP model, tests are run in an ILP solver and it is shown that the solver succeeds in finding optimal solutions for all smaller-and most medium-scale instances. The GA implementation uses binary encoding and the standard genetic operators adapted to the problem, improved by a caching technique. Experimental results are performed on two sets of instances, proving the efficiency and reliability of the proposed methods.

Another recent work deals with an electromagnetism-like (EM) heuristic [13]. The EM implements a hybrid approach consisting of the attraction-repulsion mechanisms for the movement of the particles, combining with the appropriate scaling technique. The EM uses a fast local search procedure which additionally improves the efficiency of the overall system. The algorithm was examined on the same instances as in [14] and the results obtained clearly indicate that the proposed EM is a useful tool for solving MSSP.

As many other set partition problems, the MSSP problem has direct and closely related applications in various fields of science.

For example, a useful application of the set splitting approach is presented in [19, 20]. The authors deals with the ternary content addressable memory (TCAM) to solve the multi-match classification problem, which is needed for network applications: intrusion detection systems, packet-level accounting. In order to improve the performances of the use of TCAM, the authors use a set splitting algorithm (SSA) to split the filters into multiple groups. SSA then

performs separate TCAM lookups into these groups in parallel. By this approach, at least half of the intersections are removed when a filter set is split into two sets. This approach results in a low TCAM memory usage and better performances of the overall algorithm.

Solving MSSP can be useful for finding solutions to some practical organizational problems. Suppose that a set of tasks $S = \{t_1, t_2, \ldots, t_m\}$ and the set of employees $C = \{E_1, E_2, \ldots, E_n\}$ are given, where each employee can do several tasks. One wants to divide the set S into two subsets (for example tasks are disposed in two periods), in a way that the number of employees who can do at least one task in each period is as large as possible.

In education, the MSSP can be used to improve course organization and lesson scheduling. Suppose that a course contains m lessons: $L_1, L_2, \ldots, L_m$. In a wider context, lessons can belong to the fields: $F_1, F_2, \ldots, F_n$. Each field contains several lessons and one lesson can belong to several fields. The task is to divide the set of lessons into two periods (e.g. winter and summer semester), in a way that as many fields as possible are covered in each period. The motivation of this approach could be the methodical reason indicating that the basics (and not only the basics) of each subject field should be repeated and continuously processed during the whole academic year.

The rest of the paper is organized as follows. The next section presents two integer programming formulations. In Section 4 the VNS algorithm for solving MSSP is described. Section 5 contains experimental results, based on the instances used in [14] and also in [13]. Section 6 is the conclusion.

3. Mathematical formulation. This section presents two already known integer programming formulations: quadratic integer formulation from [1] and integer linear programming (ILP) from [14].

Before the formulations are listed, let us introduce the appropriate notation. Let S be a finite set with cardinality $m = |S|$ and let $C = \{S_1, S_2, \ldots, S_n\}$ be a collection of subsets of S. Without loss of generality, we suppose that $S = \{1, 2, \ldots, m\}$. Let (P_1, P_2) be a partition of S. Recall that for the subset $S_j \in C$ we say that it "is split", if S_j has non-empty intersections with both P_1 and P_2.

The quadratic integer formulation (1)–(4) introduces two sets of variables:

$$y_i = \begin{cases} 1, & i \in P_1 \\ -1, & i \in P_2 \end{cases}, \quad \text{for } i = 1, \ldots, m$$

and

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

The QIP is defined as

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

subject to

$$\frac{1}{|S_j| - 1} \sum_{\substack{i_1, i_2 \in S_j \\ i_1 \neq i_2}} \frac{1 - y_{i_1} \cdot y_{i_2}}{2} \geq z_j, \quad \text{for all } S_j \in C \tag{2}$$

$$z_j \in \{0, 1\}, \quad \text{for } j = 1, \ldots, n \tag{3}$$

$$y_i \in \{-1, 1\}, \quad \text{for } i = 1, \ldots, m \tag{4}$$

For the Integer linear programming (ILP) formulation (5)–(9), the parameters

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

are introduced to denote if the element i from S belongs to the subset S_j. The decision variables are defined as

$$x_i = \begin{cases} 1 & i \in P_1 \\ 0 & i \in P_2 \end{cases}$$

$$y_j = \begin{cases} 1 & S_j \text{ is split} \\ 0 & S_j \text{ is not split} \end{cases}$$

The ILP model for the MSSP is formulated as follows:

$$\max \sum_{j=1}^{n} y_j \tag{5}$$

subject to

$$y_j \leq \sum_{i=1}^{m} s_{ij} \cdot x_i, \qquad \text{for every } j = 1, \ldots, n \tag{6}$$

$$y_j + \sum_{i=1}^{m} s_{ij} \cdot x_i \leq |S_j| \quad , \qquad \text{for every } j = 1, \ldots, n \tag{7}$$

$$y_j \in \{0, 1\} \quad , \qquad \text{for every } j = 1, \ldots, n \tag{8}$$

$$x_i \in \{0, 1\} \quad , \qquad \text{for every } i = 1, \ldots, m \tag{9}$$

It can easily be counted that the ILP formulation deals with $m+n$ binary variables and $2 \cdot n$ constraints.

4. VNS for solving MSSP. The VNS technique was introduced by Mladenović and Hansen in [18]. As an effective metaheuristic approach, in recent years VNS has been intensively used for solving hard and complex optimization problems. The main strategy of the VNS approach is to investigate the quality of the solutions which belong to some neighborhood of the current best one. By a systematic change of the neighborhoods, the algorithm is steered to leave the suboptimal solutions and to change over better ones. This approach has a reasonable explanation in the fact that multiple local optima are often in a kind of correlation. So, investigating the quality of the current solution's neighbors and seeking a better local optimum during the local search can lead to better objective value.

In general, the VNS generates a family of neighbors of the arbitrary solution and at each step of the algorithm, a neighbor of the current solution is explored by a local search. This strategy of the VNS technique is usually combined with the general strategy of most metaheuristic approaches, i.e., disseminating the search into unexplored search spaces. To achieve this general proposition, the VNS usually deals with neighborhoods of increasing cardinality. More precisely, if x is a solution and N_k, $(k = k_{\min}, \ldots, k_{\max})$ is a finite set of pre-selected neighborhood structures, then $N_k(x)$, the set of solutions in kth neighborhood of x, is usually defined to satisfy the condition $|N_{k-1}(x)| < |N_k(x)|$. For an incumbent x and an integer k associated to the kth neighborhood of the solution x, a feasible solution x' is generated in $N_k(x)$, and a local search is then applied to x' in order to obtain a possibly better solution x''. If x'' is better than x, then x'' becomes the new incumbent and the next search begins at the first neighborhood $N_{k_{\min}}(x'')$; otherwise, the next neighborhood in the sequence is considered in order to try to improve upon solution x. If the last neighborhood $N_{k_{\max}}$ is explored and a solution better than the incumbent is not found, the search restarts at the first neighborhood $N_{k_{\min}}$ until a stopping condition is satisfied. The stopping criteria

most often used are maximum number of iterations, maximum number of iterations between two improvements or maximum CPU time allowed. A detailed description of different VNS variants is out of the paper's scope and can be found in [11, 12].

The basic scheme of VNS is shown in Figure 2. The whole VNS is imple-

Initialization:

1. Read input data.
2. Determine the initial solution x (usually at random).
3. Define the values $k = k_{min}$ and k_{max}
   and select the set of neighborhood structures N_k, for $k = k_{min}, ..., k_{max}$;
4. Define the stopping criterion;

While the stopping criterion is not met do

1. Set $k := k_{min}$;
2. While $k <= k_{max}$ do

   Shaking procedure. Generate a point x' at random from the k-th neighborhood of x ($x' \in N_k(x)$);

   Local Search procedure. Apply a local search method with x' as initial solution;

   Denote x'' the obtained local optimum;

   Move or not procedure. If this local optimum is better than the incumbent, move

   there ($x := x''$) and go to Step (1); Otherwise, set $k := k + 1$.

Collect output data and print the solution

Fig. 2. VNS algorithm scheme

mented through two nested loops. The outer loop controls the overall iteration process, while the inner loop controls the major search via two main VNS functions, shaking and local search. The shaking procedure suggests a new potential solution from the current neighborhood and local search is tries to improve that solution "inside" the local neighborhood. If the current best solution can no longer be improved by the shaking-local search system, the inner loop stops and the outer one starts the next iteration. This process is repeated until the stopping criterion is satisfied.

4.1. Initialization and objective function. Let S be a finite set and $|S| = m$. A partition (P_1, P_2) of the set S, which is in fact a solution of the VNS for solving MSSP, is represented as a binary array x with the length m. The elements of the array correspond to elements of the set S, and indicate in which of two subsets of S the elements are arranged, i.e., $i \in P_1$ if $x_i = 1$ and $i \in P_2$ if $x_i = 0$. The initial solution is chosen randomly.

Each set from the collection C is represented as an array of belonging

elements. For a given partition (P_1, P_2), for each element $S_j \in C$, the value y_j is calculated by the formula

$$y_j = \begin{cases} 1, & S_j \text{ is split} \\ 0, & S_j \text{ is not split} \end{cases}$$

For the current solution, the algorithm calculates the objective value by the formula

$$obj(P_1, P_2) = \sum_{j=1}^{n} y_j \tag{10}$$

For each $j \in \{1, \ldots, n\}$ the time complexity for determining the value y_i is $O(m)$, so the overall time complexity of the calculating objective function is $O(mn)$.

4.2. Neighborhoods and the shaking procedure. The shaking procedure creates a new solution x', $x' \in N_k(x)$ based on the current best solution x. To define the kth neighborhood, the VNS randomly chooses some k elements from S. For each chosen element, the component is changed: all chosen vertices belonging to P_1 are moved into P_2 and vice versa. Formally, the k-th neighborhood of the vector x can be written as $N_k(x) = \{x' : \{i_1, i_2, \ldots, i_k\} \subset \{1, 2, \ldots, |S|\}\ x'_{i_j} = 1 - x_{i_j}\}$.

In the algorithm, the value $k_{\min}$ is set to 2. In order to satisfy the theoretical condition which states that the size of each neighborhood should be increased at each step, $k_{\max}$ is defined as $k_{\max} = \min\{20, |S|/2\}$. Since the size of the k-th neighborhood is $\binom{|S|}{k}$, $k < |S|/2$ implies $\binom{|S|}{k-1} < \binom{|S|}{k}$, i.e., $|N_{k-1}(x)| < |N_k(x)|$ and the condition is satisfied. For larger instances, experiments show that $k_{\max} = 20$ is enough for reaching good solutions.

It is clear that shaking consists of k steps with $O(|S|)$, so the overall time complexity of the Shaking() is $O(k_{\max} \cdot |S|)$.

4.3. Local search. For the solution x' obtained by the shaking procedure, the local search is called. In each iteration of the local search, the algorithm tries to improve the solution by swapping the components of two elements from S. For example, if $a \in P_1$ and $b \in P_2$, after the swapping the status is $a \in P_2$ and $b \in P_1$. Let us denote a new solution as x''. In the case when x'' is better solution than x', x' becomes equal to x''. The local search stops after the first such improvement. Otherwise, if there was no improvement by the swapping, the solution x' is not changed and the local search continues with the next pair of vertices.

When the local search is finished, three cases are analyzed:

a. If the objective value of x' is strictly less than the objective value of the incumbent, the search is repeated with the same x and the next neighborhood.

b. In the case when the objective value of x' is greater than of x, the currently best solution x gets the value x'.

c. If the objective values of the two solutions x and x' are the same, then $x = x'$ is set with probability p_{move} and the algorithm continues the search with the same neighborhood. In the other case, the search is repeated with the same x and the next neighborhood with probability $1 - p_{move}$. In the algorithm, p_{move} is set to 0.4.

Experiments show that the proposed algorithm can reliably work even for $p_{move} = 0$, but in that case the algorithm does not have any chance to move to another solution with the same objective value. On the other hand, the values of p_{move} close to 1 can cause cycling through the solutions with the same objective values. So, setting the p_{move} close to the middle of these two boundary values generally enables the best performances of the algorithm.

It is obvious that the local search forms pairs of elements and the total number is $O(m^2)$. Swapping the component for each pair is done in $O(1)$, and the calculation of the objective value of the new solution is done in $O(mn)$. So, the overall time complexity of the local search is $O(m^3 n)$.

When all neighborhoods have been considered, the algorithm begins again with the first one, until the stopping criterion, maximum number of iterations reached, is satisfied.

Summarizing the overall strategy, the algorithm directs the search to the unexplored area by examining the quality of the solutions in some neighborhood of the current one. The candidate solution is constructed by swapping the component for some k elements. After that, the algorithm tries to improve that solution locally, by swapping the components for pairs of elements. Choosing new solutions from the neighborhoods avoids the algorithm becoming trapped in sub-optimal solutions, while the moving procedure decreases the probability of cycling. This overall strategy seems to be a good compromise between the exploration and the exploitation part of the searching process, which is the key factor for a good direction of improvement.

5. Experimental results. This section presents some computational results of the proposed VNS method. The VNS implementation was coded in

the C programming language. All tests were carried out on the Intel Core 2 Quad Q9400 @2.66 GHz with 8 GB RAM. The tests were performed on two sets of instances: minimum hitting set instances (MHS) from [5] and Steiner triple systems (STS) described in [8]. The first class (MHS) contains ten instances with different numbers of elements ($m = 50,\ 100,\ 250,\ 500$) and different numbers of subsets ($n = 100,\ 10000,\ 50000$). The set of STS consists of seven instances: the smallest has 9 elements and 12 subsets and the largest has 243 elements and 9801 subsets. The STS instances are marked as harder, because the ILP solver was unable to reach optimal solutions for middle and large scale sets, containing 27 and more elements [14]. For each problem instance, the VNS was run 20 times. Each run stopped after 100 iterations.

Table 1 and Table 2 provide testing results of the proposed VNS approach on MHS and STS instances. Both tables are organized in the same way: the first two columns contain information on the number of elements (m) and number of subsets (n); the next column (o/b) contains the best known solution, marked with an asterisk '*' symbol, if the solution is not proven as globally optimal. The best VNS value is given in the column VNS, with the mark 'opt' in cases when the VNS reached an global optimal solution known in advance, or the mark 'best' if the solution is the best known, but not proved as globally optimal.

It is clear that no solution can have an objective value greater than the total number of subsets. So, for the first class of instances (MHS instances), it is obvious that the values reached are the global optima. For the second class of instances (STS instances), the optimality of the results for the first two instances is proven by the exact method from [14]. For other STS instances, we cannot guarantee that achieved best solutions are globally optimal.

The average time needed to detect the best VNS value is given in the t column. The solution quality in all 20 executions ($i = 1, 2, \ldots, 20$) is evaluated as a percentage gap named $agap = \frac{1}{20}\sum_{i=1}^{20} gap_i$, where $gap_i = 100 * \frac{Opt.sol - sol_i}{Opt.sol}$ is evaluated with respect to the optimal solution $Opt.sol$, or the best-known solution $Best.sol$, i.e. $gap_i = 100 * \frac{Best.sol - sol_i}{Best.sol}$ in cases where no optimal solution is found (sol_i represents the VNS solution obtained in the i-th execution). The standard deviation of the average gap $\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(gap_i - agap)^2}$ is also presented.

Table 1. VNS results on MHS instances

m	n	o/b	VNS	t_{VNS}	agap	σ
50	1000	1000	opt	0.17835	0.0	0.00
50	10000	10000	opt	2.71175	0.0	0.00
100	1000	1000	opt	0.2969	0.0	0.00
100	10000	10000	opt	3.7231	0.0	0.00
100	50000	50000	opt	124.589	0.0	0.00
250	1000	1000	opt	0.80325	0.0	0.00
250	10000	10000	opt	12.3331	0.0	0.00
500	1000	1000	opt	1.64985	0.0	0.00
500	10000	10000	opt	20.11155	0.0	0.00
500	50000	50000	opt	225.2309	0.0	0.00

Table 2. VNS results on STS instances

m	n	o/b	VNS	t_{VNS}	agap	σ
9	12	10	opt	0.0009	0.0	0.00
15	35	28	opt	0.002	0.0	0.00
27	117	91*	best	0.00915	0.0	0.00
45	330	253*	best	0.04215	0.0	0.00
81	1080	820*	best	0.27625	0.0	0.00
135	3015	2278*	best	1.2607	0.0	0.00
243	9801	7381*	best	11.84055	0.0	0.00

From Tables 1 and 2 it is clear that VNS reaches all known globally optimal and best solutions. Also, these solutions are reached in each of the 20 runs, which indicates high reliability of the VNS algorithm. The computational time is rather small even for the largest MHS instances, with 100 and 500 elements and 50 000 subsets. The computational times for these instances goes up to 124 and 225 seconds, respectively. For all STS instances, the VNS easily reaches all known optimal/best solutions in a short time.

Tables 3 and 4 contain comparative results obtained on MHS and STS instances by various techniques from papers [14, 13] and the proposed VNS. Table columns are organized as follows:

- the first two columns contain m and n;
- optimal value, if it is known, and otherwise the best known solution, is

marked with an asterisk '*'.

- the next two columns contain the value and running time obtained by CPLEX, with the mark N/A if the solution is not available due to memory or time limits [14];
- the next two columns contain the value and running time obtained by GA [14];
- the next two columns contain the value and running time obtained by EM [13];
- the last two columns contain the value and running time obtained by VNS;

In all cases, if the technique reached the optimal(respectively best) solution, the mark 'opt(best)' is used instead of the optimal(best) value.

Table 3. Comparative results on MHS instances

m	n	o/b	CPL	t_{CPL}	GA	t_{GA}	EM	t_{EM}	VNS	t_{VNS}
50	1000	1000	opt	0.078	opt	2.582	opt	0.158	opt	0.17835
50	10000	10000	opt	3.265	opt	60.039	opt	3.212	opt	2.71175
100	1000	1000	opt	0.188	opt	4.67	opt	0.334	opt	0.2969
100	10000	10000	opt	8.297	opt	168.603	opt	10.593	opt	3.7231
100	50000	50000	opt	155.203	opt	683.147	49998	216.316	opt	124.589
250	1000	1000	opt	0.219	opt	8.626	opt	1.062	opt	0.80325
250	10000	10000	opt	30.063	opt	336.894	opt	45.393	opt	12.3331
500	1000	1000	opt	0.500	opt	13.325	opt	2.336	opt	1.64985
500	10000	10000	opt	106.094	opt	437.909	opt	94.473	opt	20.11155
500	50000	50000	N/A	—	opt	2086.517	opt	486.124	opt	225.2309

Data shown in Tables 3 and 4 indicate high performances of the proposed VNS. As well as the GA, the VNS achieves all optimal/best solutions, while the EM algorithm fails to find the best solution for the MHS instance with 10000 elements and 50000 subsets. Comparing the computational times of the heuristics, it is clear that the GA is about 5–10 times slower than the two remaining methods for most instances. For the MHS instances, the VNS is up to twice faster than the EM for most instances, (for the instances with 10000 subsets the VNS is even 3–5 time, faster than the EM). For the STS instances, the EM and the VNS computational times are similar.

Table 4. Comparative results on STS instances

m	n	o/b	CPL	t_{CPL}	GA	t_{GA}	EM	t_{EM}	VNS	t_{VNS}
9	12	10	opt	0.031	opt	0.193	opt	0.001	opt	0.0009
15	35	28	opt	0.343	opt	0.233	opt	0.003	opt	0.002
27	117	91*	N/A	—	best	0.382	best	0.005	best	0.00915
45	330	253*	N/A	—	best	0.914	best	0.030	best	0.04215
81	1080	820*	N/A	—	best	2.893	best	0.173	best	0.27625
135	3015	2278*	N/A	—	best	7.858	best	0.905	best	1.2607
243	9801	7381*	N/A	—	best	65.409	best	14.953	best	11.84055

Although both set of instances are relatively small, any fair comparison between existing methods and the proposed VNS can only be made by using the same benchmark data. Experimental results performed on the instances known in the literature indicate that the proposed VNS for solving the MSSP shows better overall performances than the GA from [14] and the EM from [13]. The results obtained in the short execution times make a reliable assumption that the proposed VNS can be used efficiently for solving the MSSP.

6. Conclusions. This paper presents the VNS technique for solving the NP hard Maximum Set Splitting problem. The VNS implements two main procedures: shaking and local search. In the shaking procedure, the algorithm forms the system of neighborhoods, which are based on changing the component for an increasing number of elements. This approach disseminates the search in a good direction and enables the effective application of the local search to the current solution candidate. In order to improve the solution in local search, the VNS algorithm swaps the components for pairs of elements, trying to move to a better solution in the local neighborhood.

According to the computational results, the applied VNS approach proves to be successful. The VNS achieves all known global optima and best solutions in short times. The experiments indicate that the VNS algorithm shows better overall performances than other methods used for solving the MSSP.

Possible directions for further work include the combination of the VNS approaches with other heuristics or exact methods and their applications on similar problems.

REFERENCES

[1] Andersson G., L. Engebretsen. Better approximation algorithms for set splitting and not-all-equal sat. *Inform. Process. Lett.*, **65** (1998), 305–311.

[2] Chang W. L., M. Guo, M. Ho. Towards solution of the set-splitting problem on gel-based DNA computing.*Future Gener. Comput. Syst.,* **20** (2004), No 5, 875–885.

[3] Chen J., S. Lu. Improved algorithms for weighted and unweighted set splitting problems. LNCS, Vol. **4598**, Springer, 2007, 537–547.

[4] Chen, H., S. Lu. Improved parameterized set splitting algorithms: A probabilistic approach. *Algorithmica*, **54** (2009), 472–489.

[5] Cutello V., G. Nicosia. A clonal selection algorithm for coloring, hitting set and satisfiability problems. LNCS, Vol. **3931**, Springer, 2006, 324–337. http://www.dmi.unict.it/∼nicosia/cop.html

[6] Dehne F. K. H. A., M. R. Fellows, F. A. Rosamond. An FPT algorithm for set splitting. In: WG, LNCS, Vol. **2880**, Springer, 2003, 180–191.

[7] Dehne F. K. H. A., M. R. Fellows, F. A. Rosamond, P. Shaw. Greedy localization, iterative compression, modeled crown reductions: New fpt techniques, an improved algorithm for set splitting, and a novel $2k$ kernelization for vertex cover. In: IWPEC, LNCS, Vol. **3162**, Springer, 2004, 271–280.

[8] Fulkerson D. R., 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

[9] Garey M., D. Johnson. Computers and intractability: A guide to the theory of NP-completeness. Freeman, San Francisco, 1979.

[10] Guruswami V. Inapproximability results for set splitting and satisfiability problems with no mixed clauses. LNCS, Vol. **1913**, Springer, 2000, 155–166.

[11] Hansen P., N. Mladenović, J. A. Moreno-Pérez. Variable neighbourhood search: methods and applications (invited survey). *4OR: A Quarterly Journal of Operations Research*, **6** (2008), 319–360.

[12] Hansen P., N. Mladenović, J. A. Moreno-Pérez. Variable neighbourhood search: algorithms and applications. *Ann. Oper. Res.,* **175** (2010), No 1, 367–407.

[13] Kratica J. An Electromagnetism-like method for the maximum set splitting problem. *YUJOR*, **23** (2013), No 1, 1–11.

[14] Lazović B., M. Marić, V. Filipović, A. Savić. An integer linear programming formulation and genetic algorithm for the maximum set splitting problem. *Publications de l'Institut Mathematique*, **92**(2012), No 106, 25–34.

[15] Lokshtanov D., C. Sloper. Fixed parameter set splitting, linear kernel and improved running time. In: *ACiD*, **4** (2005), 105–113.

[16] Lokshtanov D., S. Saurabh. Even faster algorithm for set splitting!. In: Proceedings of the International Workshop on Parameterized and Exact Computation (IWPEC), 2009, 288–299.

[17] Lovász L. Coverings and colorings of hypergraphs. In: Proceedings of the 4th Southeastern Conf. on Comb., Utilitas Math., 1973, 3–12.

[18] Mladenović N., P. Hansen. Variable neighbourhood search. *Comput. Oper. Res.*, **24** (1997), 1097–1100.

[19] Yu F., T. V. Lakshman, M. A. Motoyama, R. H. Katz. SSA: A Power and Memory Efficient Scheme to Multi-Match Packet Classification. In: Proceedings of the ACM Symp. Architecture for Networking and Comm. Systems (ANCS 05), 2005, 105–113.

[20] Yu F., T. V. Lakshman, M. A. Motoyama, R. H. Katz. Efficient Multimatch Packet Classification for Network Security Applications. *IEEE J. Selected Areas in Comm.*, **24** (2006), No 10, 1805–1816.

[21] Zhang J., Y. Ye, Q. Han. Improved approximations for max set splitting and max NAE SAT. *Discrete Appl. Math.*, **142** (2004), 133–149.

Dragan Matić
Mladena Stojanovića 2
Department of Mathematics and Informatics,
Faculty of Science and Mathematics
University of Banja Luka
Bosnia and Herzegovina
e-mail: matic.dragan@gmail.com

Received April 3, 2012
Final Accepted December 2, 2012