Serdica J. Computing **4** (2010), 435–446

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

SOLVING THE TASK ASSIGNMENT PROBLEM WITH A VARIABLE NEIGHBORHOOD SEARCH*

Jozef Kratica, Aleksandar Savić,
Vladimir Filipović, Marija Milanović

Abstract. In this paper a variable neighborhood search (VNS) approach for the task assignment problem (TAP) is considered. An appropriate neighborhood scheme along with a shaking operator and local search procedure are constructed specifically for this problem. The computational results are presented for the instances from the literature, and compared to optimal solutions obtained by the CPLEX solver and heuristic solutions generated by the genetic algorithm. It can be seen that the proposed VNS approach reaches all optimal solutions in a quite short amount of computational time.

1. Introduction. Recent developments in computer industry present problems of a more sophisticated approach to allotment of memory and processors

ACM Computing Classification System (1998): G.1.6.

Key words: Task assignment, multiprocessor systems, variable neighborhood search, assignment problems, combinatorial optimization.

*This research was partially supported by the Serbian Ministry of Science and Ecology under project 144007.

to various tasks. This is particularly true if multiple processors and a larger number of tasks are involved. Assignment of tasks to processors where running times varied was vital for obtaining streamline of processing jobs. The problem was complicated by mutual interference of multiple tasks with interchange of information.

In this paper a special Task Assignment Problem (TAP), sometimes called TAS with non-uniform communication costs, is studied. The roots of this problem can be traced to the earliest times of use of computers but its complexity and enormous time consumption did not allow it to be solved. The development of new metaheuristics, specifically variable neighborhood search, allowed solving this problem in short times with reasonably good results.

The problem of task assignment with non-uniform communication costs can be modelled as a quadratic integer 0-1 minimization problem. Note that the problem is fully 0-1 integer, it is hard for exact methods, so it must be solved by metaheuristic approaches such as genetic algorithm (GA) or variable neighborhood search (VNS).

The TAP problem is also known as Quadratic Semi-Assignment Problem, which is discussed in [1] mostly on theoretical basis. In [1] it is stated that the problem is NP-hard, and methods of reduction of a linear part of objective function are given. In that paper only experimental results are given for the number of reduction steps and time of their execution and not for solving respective instances, so a comparison was not possible.

In [13] a genetic algorithm (GA) for the task assignment problem (TAP) is considered. In that paper an integer representation with standard genetic operators is used. The proposed GA approach reaches 17 of 20 optimal solutions in a short amount of computational time. Because of this fact a comparison became possible and both those results and respective VNS results are given in Table 2.

Similar to our problem are the Memory-constrained Allocation Problem (MCAP) and Constrained Module Allocation Problem (CMAP). MCAP is considered in detail in [12]. This problem has similar structure to TAP but has additional constraints on memory amount assigned to every individual task. CMAP considerations can be found in [5], especially problems' lower bounds.

2. Mathematical formulation. In this section we will use the quadratic integer 0-1 programming formulation for TAP given in [4]. The problem of

task assignment with non-uniform communication costs is related to finding an assignment of N tasks to M processors providing that:

- the total cost of execution for given tasks,
- the total cost of all communications between processors, while they execute allocated tasks,

are minimal.

Let there be N tasks and M processors, and e_{ik} be a cost of executing task i on a processor k. Let c_{ijkl} be the communication cost between tasks i and j if they are respectively assigned to processors k and l. Let us denote with 0-1 integer variable x_{ik} which has value 1 if task i is assigned to processor k.

Now we can formulate the quadratic integer programming model for TAP as follows:

$$\min \sum_{i=1}^{N}\sum_{k=1}^{M} e_{ik}x_{ik} + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\sum_{k=1}^{M}\sum_{l=1}^{M} c_{ijkl}x_{ik}x_{jl} \tag{1}$$

subject to

$$\sum_{k=1}^{M} x_{ik} = 1, \qquad i = 1, \ldots, N \tag{2}$$

$$x_{ik} \in \{0,1\} \qquad i = 1, \ldots, N, \;\; k = 1, \ldots, M \tag{3}$$

The constraints (2) reflect the natural request that any particular task should be executed on only one processor.

3. Proposed VNS method. Variable Neighborhood Search (VNS) was first introduced in the literature by Hansen and Mladenović in ([8]). It is a robust and effective metaheuristic, which can be seen from the large number of its successful applications. The detailed description of different VNS variants is out of this paper's scope and can be found in [6]. We shall only mention some of the recent and successful VNS applications:

- mixed integer programming [7];
- minimum labelling Steiner tree[3];

- bandwidth reduction[9];
- variable selection and determination [10];
- container loading [11];
- nurse rostering [2];
- quadratic assignment problem [14].

The chief purpose of this method is to change neighborhoods as the search for solution is progressing (hence the name variable neighborhood) and to combine a local search method for finding local extremes. VNS systematically exploits the concept of neighborhood change, both in descent to local minima and in escape from the valleys which contain them. Two complementary concepts are combined to achieve this: local search and shaking.

First, it is necessary to define a suitable neighborhood structure for the solution space. The method is constructed firstly to find local minima for running neighborhood by using local search. When this is achieved, the neighborhood is changed so that other parts of the solution space can be searched.

Let N^k ($k = k_{\min}, \ldots, k_{\max}$) be a finite set of neighborhoods, where $N^k(S)$ is the set of solutions in the kth neighborhood of the solution S. The simplest and most common choice is a structure in which the neighborhoods have increasing cardinality: $|N^{k_{\min}}(S)| < |N^{k_{\min}+1}(S)| < \cdots < |N^{k_{\max}}(S)|$.

Given an incumbent S and an integer $k \in \{k_{\min}, \ldots, k_{\max}\}$ associated to a current neighborhood, a feasible solution S' is generated in $N^k(S)$, and a local search is then applied to S' in order to obtain a possibly better solution S''. If S'' is better than S, then S'' becomes the new incumbent and the next search begins at the first neighborhood $N^{k_{\min}}$; otherwise, the next neighborhood in the sequence is considered in order to try to improve upon solution S. Should the last neighborhood $N^{k_{\max}}$ be reached without a solution better than the incumbent being found, the search begins again at the first neighborhood $N^{k_{\min}}$ until a stopping condition, e.g., a maximum number of iterations, is satisfied.

The process of choosing solution S' in the current neighborhood is called shaking. This procedure can be performed by random choice or by some other strategy and is used to diversifies the search process. If the structure of the neighborhoods is such that the order of neighborhood cardinalities is increasing, this means that the search for solutions will be diversified on each step over an increasing part of the solution space.

The pseudo-scheme of the variable neighborhood search for the task assignment problem is given in Figure 1.

Input parameters for VNS are: minimal and maximal number of neighborhoods which should be searched, $k_{\min}$ and $k_{\max}$, maximal number of iterations and p, which represents the probability of moving from one solution to another with the same value of the objective function.

Initialization is carried out randomly. At the beginning of the VNS procedure a randomly chosen processor is assigned to each task and this assignment is the starting solution S.

On every iteration function Shaking() is performed as follows. For a given k, the k elements from the set $\{1, \ldots, N\}$ are chosen randomly. For each of the chosen k tasks we randomly change its assigned processor to some random value from the set $\{1, .., M\}$, while for the other tasks the assignments in the S' are preserved from S. After that a local search, which is explained later in all details, is carried out until there is no improvement. At the end of the local search a new solution S'' is obtained. If this new solution is better than the current solution S, then S'' replaces S and the algorithm continues in the same neighborhood. If the local search produces a worse solution S' than S, a new neighborhood is tried ($k = k + 1$). Because k increases it is clear that the cardinality of $N_{k+1}(S)$ is larger than that of $N_k(S)$. This can be seen from the fact that $|N_{k+1}(S)| = M^{k+1} > M^k = |N_k(S)|$. In the case $k = k_{\max}$ we set the neighborhood to $k = k_{\min}$.

Basic VNS moves strictly from one solution to the next solution, which has an objective function value strictly greater than previous solution. For problems with many local optima with the same value of objective function, this approach is too *tight*. Switching to one of these optima can broaden the search and increase the chances of improvement. Contrariwise, if the algorithm switches from one solution to another every time value of objective function is repeated, the chance of *moving in circles* increases. Contrary to basic VNS here is implemented a modification of the function Compare(), which switches from one solution to another with the same value of the objective function with some probability p. In this way, both drawbacks mentioned before are successfully prevented.

The local search procedure implemented in this VNS method is now explained. The local search traces assignment of tasks to processors going from task 1 to task N. For each task, a local search tries to find a better assignment of that task with some other processor. If there is improvement in the objective function, we change the assignment of that task and continue the process

```
Function VNS(k_min, k_max, iter_max, p)
    S:= Random_Init();
    k := k_min;
    iter := 0;
    repeat
       iter := iter + 1;
       S' := Shaking(S,k);
       S'' := Local_Search(S');
       if Compare(S, S'', p)
       then
          S := S'';
       else
          if k < k_max
          then
             k := k + 1;
          else
             k := k_min;
          endif
       endif
    until iter ≤ iter_max;
    return S;
end;

Function Compare(S, S'', p)
 if(Obj(S'') < Obj(S))
 then
    return true;
 else
    if(Obj(S'') > Obj(S))
    then
       return false;
    else
       return Random(0,1)<p;
    endif
 endif
end;
```

Fig. 1. Pseudo-code of the VNS for TAP

again starting from task 1. If there is no improvements, the local search continues tracing the assignment of the next task. If there is no improvement in the objective function after tracing all N tasks, we end the local search procedure and the newly obtained solution is S''. This strategy is known in the literature as "first improvement strategy", because we reset the local search step after the first improvement.

4. Experimental results. All computations were executed on a Quad Core 2.5 GHz PC computer with 4 GB RAM. The VNS implementation was coded in C language. For experimental testings in this implementation the instances described at [4] were used. These instances include different numbers of tasks (N = 10, 15) and different number of processors (M = 3, 5). For each pair of task-processor (N, M), there is a set of ten instances. Note that the optimal solution values for $M = 3$ differ from the values described in [4] by at most 1. The solution values for $M = 5$ differ significantly. Therefore we used optimal solution values obtained by CPLEX, because we cannot deduce the difference from the results described in [4].

The finishing criterion of VNS is the maximal number of iterations N_{iter} = 100. In this experiment the VNS parameters $k_{\min}$, $k_{\max}$ and probability p have values of 2, 30 and 0.4, respectively. Because VNS is a non-deterministic algorithm, all experiments were executed 30 times.

Table 1 summarizes the results of VNS on these instances. In the first column the names of instances are given. The instance's name carries information about the number of tasks N, the number of processors M and the number of generated cases with the same N and M. (For example, the instance tassnu_10_3_1 is an instance which has $N = 10$ tasks on $M = 3$ processors and is the first case generated for this N and M.) The second and third column contain optimal solution values and values of the solutions obtained by VNS. In the fourth column the VNS running time is given. The fifth and sixth column contain the relative error *err* and standard deviation σ in all 30 runs. The seventh column contains the average overall number of local search steps through all 100 iterations. In the following two columns data about first occurrence of best VNS solution are presented: average number of iterations and average number of local search steps, respectively.

Direct comparisons of the VNS results with CPLEX and GA implementation from [13] are given in Table 2. The first column contains the instances'

Table 1. VNS results on TAP instances

Instance name	opt_{sol}	VNS_{sol}	t(s)	err(%)	σ (%)	$LS_{totiter}$	$iter$	LS_{iter}
tassnu_10_3_1	−719	opt	< 0.001	0.000	0.000	894	2	19
tassnu_10_3_2	−790	opt	< 0.001	0.000	0.000	762	2	17
tassnu_10_3_3	−624	opt	< 0.001	0.000	0.000	747	19	131
tassnu_10_3_4	−734	opt	< 0.001	0.000	0.000	769	6	49
tassnu_10_3_5	−871	opt	< 0.001	0.000	0.000	732	7	48
tassnu_10_3_6	−677	opt	< 0.001	0.000	0.000	684	10	69
tassnu_10_3_7	−613	opt	< 0.001	0.000	0.000	841	4	35
tassnu_10_3_8	−495	opt	< 0.001	0.000	0.000	888	2	21
tassnu_10_3_9	−750	opt	< 0.001	0.000	0.000	800	3	22
tassnu_10_3_10	−486	opt	< 0.001	0.000	0.000	755	7	48
tassnu_15_5_1	−1985	opt	< 0.001	0.000	0.000	1664	6	91
tassnu_15_5_2	−1568	opt	< 0.001	0.493	0.822	1832	17	268
tassnu_15_5_3	−1892	opt	< 0.001	0.000	0.000	1796	7	116
tassnu_15_5_4	−1806	opt	< 0.001	0.000	0.000	1630	6	92
tassnu_15_5_5	−1881	opt	< 0.001	0.000	0.000	1735	12	208
tassnu_15_5_6	−1950	opt	< 0.001	0.000	0.000	1690	16	269
tassnu_15_5_7	−1893	opt	< 0.001	0.000	0.000	1762	15	227
tassnu_15_5_8	−1733	opt	< 0.001	0.000	0.000	1647	15	241
tassnu_15_5_9	−1798	opt	< 0.001	0.100	0.301	1700	27	436
tassnu_15_5_10	−1763	opt	< 0.001	0.004	0.014	1632	32	509

names, while the second and third columns contain optimal solutions and running times of CPLEX solver. The best values of genetic algorithm (GA) GA_{sol} are given in the following column. The mark *opt* is given if optimal solution is reached and there is no difference between that solution and the solution obtained by CPLEX. Average overall running times of the GA values are given in the t column. The next column contains relative errors given in percents. In the next three columns best values, running times and relative errors of VNS are given.

As can be seen in Tables 1 and 2, VNS reached all optimal solutions and the running time on all instances is smaller than 0.001 seconds. The CPLEX solver also runs very fast on instances with 10 tasks, but on larger instances,

Table 2. Comparison of CPLEX, GA and VNS results on TAP instances

	CPLEX		*GA*			*VNS*		
Instance name	opt_{sol}	t(s)	*sol*	t(s)	err(%)	*sol*	t(s)	err(%)
tassnu_10_3_1	−719	< 1	opt	0.162	0.000	opt	< 0.001	0.000
tassnu_10_3_2	−790	< 1	opt	0.165	0.000	opt	< 0.001	0.000
tassnu_10_3_3	−624	< 1	opt	0.210	0.641	opt	< 0.001	0.000
tassnu_10_3_4	−734	< 1	opt	0.172	0.000	opt	< 0.001	0.000
tassnu_10_3_5	−871	< 1	opt	0.162	0.000	opt	< 0.001	0.000
tassnu_10_3_6	−677	< 1	opt	0.165	0.214	opt	< 0.001	0.000
tassnu_10_3_7	−613	< 1	opt	0.162	0.000	opt	< 0.001	0.000
tassnu_10_3_8	−495	< 1	opt	0.164	0.000	opt	< 0.001	0.000
tassnu_10_3_9	−750	< 1	opt	0.163	0.000	opt	< 0.001	0.000
tassnu_10_3_10	−486	< 1	opt	0.164	0.021	opt	< 0.001	0.000
tassnu_15_5_1	−1985	51832	opt	0.254	3.131	opt	< 0.001	0.000
tassnu_15_5_2	−1568	129840	−1539	0.328	4.827	opt	< 0.001	0.493
tassnu_15_5_3	−1892	52955	−1856	0.257	9.905	opt	< 0.001	0.000
tassnu_15_5_4	−1806	91146	opt	0.292	1.462	opt	< 0.001	0.000
tassnu_15_5_5	−1881	78795	opt	0.248	2.818	opt	< 0.001	0.000
tassnu_15_5_6	−1950	79872	opt	0.254	5.285	opt	< 0.001	0.000
tassnu_15_5_7	−1893	51547	opt	0.236	4.691	opt	< 0.001	0.000
tassnu_15_5_8	−1733	108092	opt	0.241	2.796	opt	< 0.001	0.000
tassnu_15_5_9	−1798	107982	−1780	0.246	2.496	opt	< 0.001	0.100
tassnu_15_5_10	−1763	109963	opt	0.246	4.169	opt	< 0.001	0.004

with 15 tasks, running times are huge (> 50 000 seconds). As can be seen in Table 2, GA did not reach the optimal solution for 3 of 20 instances. Though running times of GA are quite small, VNS results are obtained almost instantly and all are optimal. It is obvious that CPLEX cannot handle larger instances with more than 15 tasks. The GA quickly gives solutions of acceptable quality even in these instances, but it is evident that VNS performs better. Also, relative errors for VNS are smaller than those of GA by an order of magnitude.

5. Conclusions. In this paper a robust and effective variable neighborhood search metaheuristic for solving the task assignment problem is presented. The good choice of neigborhood structures and efficient implementation of shaking and local search procedures are crucial for producing excellent experimental results. The VNS reaches optimal solutions for all instances in almost instant running time.

Based on the presented results, we can conclude that VNS has the great potential of being a useful metaheuristic for solving other similar problems. The hybridization of the VNS with exact and other metaheuristic methods are further most promising directions of future work.

REFERENCES

[1] Billionnet A., S. Elloumi. Best reduction of the quadratic semi-assignment problem. *Discrete Applied Mathematics*, **109** (2001), 197–213.

[2] Burke E. K., L. Jingpeng , Q. Rong. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, **203** (2010), No 2, 484–493.

[3] Consoli S., K. Darby-Dowman , N. Mladenović , J. A. Moreno-Pérez. Variable neighbourhood search for the minimum labelling Steiner tree problem. *Annals of the Operations Research*, **172** (2009), 71–96.

[4] Elloumi S. The task assignment problem, a library of instances. `http://cedric.cnam.fr/oc/TAP/TAP.html`

[5] Elloumi S., F. Roupin , E. Soutif. Comparison of different lower bounds for the constrained module allocation problem. *RAIRO Operations Research*, in press.

[6] Hansen P., N. Mladenović. Variable neighborhood search: principles and applications. *European Journal of Operational Research*, **130** (2001), 449–467.

[7] Lazić J., S. Hana , N. Mladenović , D. Urošević. Variable neighbourhood decomposition search for 0-1 mixed integer programs. *Computers & Operations Research*, **37** (2010), No 6, 1055–1067.

[8] Mladenović N., P. Hansen. Variable neighbourhood search. *Computers & Operations Research*, **24** (1997), 1097–1100.

[9] Mladenović N., D. Urošević, D. Pérez-Brito, C. G. Garca-González. Variable neighbourhood search for bandwidth reduction. *European Journal of Operational Research*, **200** (2010), 14–27.

[10] Pacheco J., S. Casado, L. Nunez. Use of VNS and TS in classification: variable selection and determination of the linear discrimination function coefficients. *IMA Journal of Management Mathematics*, **18** (2007), No 2, 191–206.

[11] Parreno F., R. Alvarez-Valdes, J. F. Oliveira, J. M. Tamarit. Neighborhood structures for the container loading problem: a VNS implementation. *Journal of Heuristics*, **16** (2010), No 1, 1–22.

[12] Roupin F. From Linear to Semidefinite Programming: An Algorithm to Obtain Semidefinite Relaxations for Bivalent Quadratic Problems. *Journal of Combinatorial Optimization*, **8** (2004), 469–493.

[13] Savić A., D. Tošić, M. Marić, J. Kratica. Genetic algorithm approach for solving the task assignment problem. *Serdica Journal of Computing*, **2** (2008), No 3, 267–276.

[14] Zhang C., Zh. Lin, Zu Lin. Variable neighborhood search with permutation distance for QAP. *Lecture Notes in Computer Science*, Springer, **3684** (2005), 81–88.

Jozef Kratica
Mathematical Institute
Serbian Academy of Sciences and Arts
Kneza Mihaila 36/III
11000 Belgrade, Serbia
e-mail: jkratica@mi.sanu.ac.rs

Aleksandar Savić
Faculty of Mathematics
University of Belgrade,
Studentski trg 16/IV
11 000 Belgrade, Serbia
e-mail: aleks3rd@gmail.com

Vladimir Filipović
Faculty of Mathematics
University of Belgrade,
Studentski trg 16/IV
11 000 Belgrade, Serbia
e-mail: vladofilipovic@hotmail.com

Marija Milanović
Faculty of Mathematics
University of Belgrade,
Studentski trg 16/IV
11 000 Belgrade, Serbia
e-mail: marija.milanovic@gmail.com

Received July 5, 2010
Final Accepted September 6, 2010