EFFICIENT METAHEURISTICS FOR THE DYNAMIC SPACE ALLOCATION PROBLEM

Geiza Cristina da Silva[1], Laura Bahiense[2], Luiz Satoru Ochi[3], Paulo Oswaldo Boaventura-Netto[2]

[1](Corresponding author) Department of Statistics, Federal University of Pernambuco, e-mail: geiza.silva@gmail.com
[2]Production Engineering Program, COPPE, Federal University of Rio de Janeiro, e-mails: laura@pep.ufrj.br, boaventu@pep.ufrj.br
[3]Computer Science Institute, Fluminense Federal University, e-mail: satoru@ic.uff.br

Abstract – This work is devoted to the Dynamic Space Allocation Problem (DSAP), where project duration is divided into a number of consecutive periods, each of them associated with a number of activities. The resources required by the activities have to be available in the corresponding workspaces and those sitting idle during a period have to be stored. This problem contains the Quadratic Assignment Problem (QAP) as a particular case, which places it in the NP-hard class. In this context, the difficulty of identifying optimal solutions, even for instances of medium size, justifies the use of heuristic techniques. This work presents new construction and local search methods and heuristic algorithms based on the VNS and ILS metaheuristics to obtain near optimal solutions for the DSAP. Comp arisons are presented for values obtained by VNS, ILS, and results from the literature. Computational results show the proposed methods to be competitive in relation to instances in the literature and to existing techniques.

Keywords – Dynamic Space Allocation Problem, Quadratic Assignment Problem, Metaheuristics, Computational Intelligence.

1 Introduction

The dynamic space allocation problem (DSAP), very new in the literature, was introduced by McKendall et al. in 2005 [5], and was inspired by the necessity to optimize the cost of rearranging resources when we assign activities to workspaces and resources to work/storage spaces during a multiperiod planning horizon. When the resources are used to perform activities, they are assigned to workspaces. Otherwise, they are deemed to be idle and assigned to storage areas.

A *project* is divided in consecutive *periods* of time and along each one a given set of *activities* is run. Each activity demands a set of *resources* to be executed. If a given resource is not utilized during a period, it is considered *idle*. The project layout is divided in workspaces and storage areas, the resources in use being associated with workspaces and the idle ones, with storage areas. The set of periods, with their respective activities (and their respective resources) to be carried on during them, is the project's *agenda*. The objective of the problem is to allocate the resources in a way that the total distance they have to travel be minimized along the project horizon.

A small DSAP instance is shown in Figure 1, where the agenda has 4 periods, 5 activities and 9 resources. The space layout has 3 workspaces (E_1 to E_3) along with 3 depots (E_4 to E_6). This type of layout, with the rows of workspaces and depots alongside each other, is considered in every instance of the literature. The Manhattan metric is used in to determine distances. Each space can receive up to 3 resources.

Period	**Activities (Necessary Resources)**	**Idle Resources**
1	A_1 (6,7), A_2 (1,5)	2,3,4,8,9
2	A_2 (1,5), A_3 (3,4)	2,6,7,8,9
3	A_4 (2,8)	1,3,4,5,6,7,9
4	A_4 (2,8), A_5 (6,9)	1,3,4,5,7

E_1	E_2	E_3
E_4	E_5	E_6

Figure 1 – A small DSAP instance and layout of the facility

For a given solution to be considered feasible, the following conditions must be met:

- during a given period, exactly one activity can be carried out in a workspace;
- at any time, a given activity is always carried out in the same workspace;

- the capacity of a workspace must be sufficient to contain the resources required by its activity;
- the depot capacities also have to be respected.

An optimal solution for this instance found using Cplex is shown in Figure 2. The total cost (associated with distance) is 12. During Period 1, Resources 1 and 5, used by Activity A_2, were allocated to Workspace E_1 and Resources 6 and 7, used by A_1, to Workspace E_3, while the idle resources were allocated to Depots E_4, E_5 and E_6 as shown in the figure. During Period 2, the resources from A_1 become idle and go to E_6, covering 1 distance unit. At the same time, A_3 is allocated to E_2, which means that its resources 3 and 4 have to come from E_5 (1 distance unit). The third period finds only A_4 being carried out at E_1, where A_2 is deactivated, with its resources going to E_4. Consequently, Resources 2 and 8 travel from E_4 to E_3 and 1 and 5 move in the opposite direction. Lastly, Period 4 has A_4 continuing to be executed (in E_1) while A_5 is carried out in E_3, receiving its resources from E_6 (1 distance unit). The sum of the distances traveled by all resources is 12, the optimal problem value.

A_2(1,5)		A_1(6,7)
2,8	3,4	9

Period 1

A_2(1,5)	A_3(3,4)	
2,8		6,7,9

Period 2

A_4(2,8)		
1,5	3,4	6,7,9

Period 3

A_4(2,8)		A_5(6,9)
1,5	3,4	7

Period 4

Figure 2 – A solution for DSAP example

DSAP is related to other well-known combinatorial optimization problems: the problem of associating activities with workspaces (quadratic assignment problem, QAP) [4], the problem of allocating activities to multiple time periods (dynamic facility layout problem, DFLP) [10] and the problem of associating idle resources with storage areas (generalized quadratic assignment problem, GQAP) [3]. In McKendall et al [8], a mathematical model is presented and two simulated annealing heuristics are developed. McKendall and Jaramillo [7] presented five constructive methods together with a simple tabu search heuristic. McKendall [6] presented three tabu search heuristics.

In this paper, we present two heuristic procedures based on VNS and ILS metaheuristics in order to obtain near optimal solutions for the DSAP. Section 2 presents the proposed heuristics. Section 3 presents computational results for a set of test problems taken from the literature. Section 4 provides conclusions and suggestions for future research.

2 Proposed methods for the DSAP

This work presents new construction and local search methods, and heuristic algorithms based on VNS and ILS metaheuristics to obtain near optimal solutions for the DSAP, along with the isolated implementation of these methods.

2.1 Construction algorithms

The proposed construction algorithm involves two stages to build a DSAP solution. The first calculates a partial solution where activities are associated with workspaces. Let A be the set of activities (a = 1, 2, ..., $|A|$) and W (w = 1, 2, ..., $|W|$) the set of workspaces (WS). The second stage generates partial solutions by allocating idle resources (IR). Let S then be the set of storage areas, C_s the capacity of storage area s, P the set of periods, R the set of resources and I_p the set of idle resources over period p.

A partial activity solution (Figure 3) is built as follows: for each activity a, a *restricted candidate list* (RCL) is created in lexicographic order, with the available WS from the beginning to the end of the activity execution period. A WS is randomly selected from the RCL to be associated with the activity a. The cardinality of the RCL is given by the number of WS available for the activity.

```
Partial Activities Solution Algorithm (A, W)
1.  for a ← 1 until |A| do
2.      Create RCL;
3.      F ← first period of a;
4.      L ← last period of a;
5.      for each w ∈ W do
6.          if w is available at all times from F to L do
7.              Add w in RCL;
8.          end if
9.      end for
10.     Sel ← Choose a random element in RCL;
11.     Allocate the activity a to workspace Sel;
12.     Free RCL;
13. end for
end.
```

Figure 3 – Pseudo code for the heuristic of association of activities with workspaces.

For the association of idle resources (Figure 4), we adapt the RSP heuristic proposed in [8], because it considers the resource association made at the previous period. This way, from the second period on, resources that remain idle from a period to the next, are prioritarily allocated to the same depot they occupied during the previous period. The remaining IR during the period are allocated, as in RSP, to the nearest storage area with respect to the most recently used WS.

```
Partial Idle Resource Solution Algorithm (S, C_s, P, R, Ip, α)
1. In the first period, allocate each idle resource to the depot where the resource was most recently used
(in activity solution);
2. for p ← 1 to |P| do
3.      Allocate idle resources that occur in p =1to the same storage area;
4.      Allocate the remaining idle resources to the storage area closest to the workspaces they are assigned to, when
performing activities.
5. end for
end.
```

Figure 4 – Pseudo code for the heuristic of association of idle resources to depots.

2.2 Movements and Neighborhoods

The following movements, defined in [8] and [7] are considered:

- *M1*: WS exchanges of two or more activities within consecutive periods during which they were allocated.
- *M2*: Removal of one activity from a WS to another one that is available during one or more consecutive periods when it is allocated.
- *M3*: Combines *M*1 and *M*2.
- *M4*: Exchange of depots between two resources.
- *M5*: Removal of a resource from an storage area to another when the capacity allows it.
- *M6*: Depot exchange of two or more resources during consecutive periods available when they are allocated.
- *M7*: Transfer of a resource from one storage area to another during consecutive periods available when it is allocated.
- *M8*. Combines *M*6 and *M*7.

Starting with a solution *s* and using these movements, the following neighborhood structures are utilized:

- *N*1: neighborhood explored by *M*1 to *M*3, applying every possible movement to the partial activity solution.
- *N*2: neighborhood explored by *M*4 and *M*5, every possible individual IR movement applied to the partial IR solution.
- *N*3: neighborhood explored by *M*6 to *M*8, every possible movement being applied to the partial IR solution during consecutive periods.
- *N*4: neighborhood explored by combining *N*1 and *N*2, that is, for each neighbor generated in *N*1, all *N*2 neighbors are investigated.

2.3 Local Search Algorithms

Throughout this work, two local search strategies are utilized, *Best Improvement* and *First Improvement*.

Best Improvement explores the whole neighborhood *V(s)* from a current solution *s* obtained by the construction method. A better neighbor, after execution, becomes the current solution for the next iteration. The search goes on until no better neighbor is found.

First Improvement is a non-exhaustive local search. A movement is applied to the first better neighbor of s, that is, the first solution $t \in V(s)$ such that $f(t) < f(s)$ will be the current solution for the next iteration. The search goes on until no better neighbor is found.

Based on the above defined neighborhoods and local search implementations, we propose the following local searches (Figure 5):

Local search	Implementation	Neighborhoods
LS1	FI	*N*1 and *N*3
LS2	BI	*N*1
LS3	BI	*N*2
LS4	BI	*N*3

Figure 5 – Proposed local search algorithms.

2.4 Variable Neighborhood Search metaheuristic

The VNS (Variable Neighborhood Search) metaheuristic was proposed in [2]. It explores more than one neighborhood and it can combine deterministic and stochastic neighborhood changes[9]. First, a set of neighborhood structures is defined, after which they can be arbitrarily chosen to compose a sequence $|N_1| < |N_2| < ... < |N_{kmax}|$ where *kmax* is a starting parameter [1].

An initial solution *s* is generated and a neighborhood counter *k* is initialized. A neighbor *s'* from *s* within $N_k(s)$ is randomly chosen and a local search based on *s'* is executed. If a solution *s"* better than s' is found, then *s"* becomes the current solution and the procedure is reinitialized with $k = 1$. Otherwise, *k* is incremented and a new search is started. The process ends when there is no better solution within any $N_k(s)$. An enveloping loop allows for more iterations until a stopping criterion is matched.

We use a VNS version where $k_{max} = 3$ (Figure 6). The neighborhood structures are those defined in Section 2.2. For each $k \in \{1, 2, 3\}$ a solution s', neighbor of s, is randomly generated as follows (Line 5):

- $k = 1$: An activity or IR movement is randomly chosen. In the first case, a neighbor is generated within *N1*, otherwise, within *N2*.
- $k = 2$: A neighbor is generated within *N4*.
- $k = 3$: As for $k = 1$ with activity movements; the neighborhood *N3* is used with IR movements.

At the local search stage (Line 6), the searches LS2. LS3 and LS4 are executed, in that order.

```
Heuristic VNS (N, k_max, max_no_improv)
1.      s ← constructionAlgorithm ();
2.      while iter < max _no_improv do
3.              k ← 1;
4.              while k < k_max do
5.                      s' ← Choose a random neighborhood in N_k(s)
6.                      s'' ← LocalSearch(s');
7.                      if f(s'') < f(s) then
8.                              s ← s'';
9.                              k ← 1; iter ← 0;
10.                     else
11.                             k ← k + 1; iter ← iter + 1;
12.                     end if
13.             end while
14.     end while
end.
```

Figure 6 – Pseudo code for VNS heuristic.

2.5 Iterated Local Search metaheuristic

The ILS (iterated Local Search) metaheuristic is an iterative process where local searches are applied to new starting solutions obtained through perturbations applied to local optima. The method works as follows (Lourenço et al, 2003) [5]: Given a current solution *s**, a perturbation is applied to it generating a solution *s'*. Then *s'* is submitted to a local search, giving *s*'*, which is a local optimum. An acceptance criterion is utilized to decide which of these two solutions will be the current one for the next iteration.

The proposed ILS algorithm is presented in Figure 7. Six perturbation levels are utilized:

- Level 1: Two different randomly chosen WS have their elements exchanged within every period.
- Level 2: Two activities allocated to different WS are randomly chosen for WS exchange. This is done *t* times (*t* is a parameter). Since the solution has to be kept feasible, a new perturbation must be done if one activity is allocated to different WS in consecutive periods.
- Level 3: Two different pairs of WS (at least one of them not empty) are randomly chosen in a period and the elements of each pair are exchanged. This is done *t* times.
- Level 4: One perturbation Level 2 and *t* perturbations Level 3.
- Level 5: One period and two different depots are chosen and one resource from one depot is exchanged with one resource from the other depot. This is done *t* times. When there is space in one of these depots, a reallocation movement can be done.
- Level 6: Two IR from different depots are chosen to be exchanged over every consecutive period when they are present. The process is repeated *t* times. A new attempt is made if the exchange of any IR pair allows one or more IR to go to different depots in consecutive periods.

```
Heuristic ILS (max_iter, max_times)
1.      s0 ← GenerateInitialSolution();
2.      s0 ← LocalSearch(s0);
3.      iter ← 0; level ← 1; t ← 2;
4.      while iter < max_iter do
5.              times ← 0;
6.              while times < max_times do
7.                      s' ← Perturbation(s0, level, t);
8.                      s'* ← LocalSearch(s');
9.                      if f(s'*) < f(s0) then
10.                             f(s0) ← f(s'*);
11.                             iter ← 0; level ← 1; times ← 0;
12.                     else
13.                             iter ← iter + 1; times ← times + 1;
14.                     end if
15.             end while
16.             level ← level + 1;
17.             if level > 6 then
18.                     level ← 1; t ← t + 1;
19.             end if
20.     do while
end.
```

Figure 7 – Pseudo code for ILS heuristic.

Each perturbation level is applied to a solution a given number of times, limited by the parameter *max_times*, as we can see in the internal loop (Lines 6 to 15). The process is repeated until a given number of unsuccessful iterations is executed, defined by *max_iter* (Lines 4 to 20). Each time a solution passes through all levels, *t* is incremented by one unit (Line 18).

3 Computational results

The set of tested instances is available in [8]. It is composed of 96 instances (P01 to P96) containing problems with 6, 12, 20 and 32 locations and 9, 18, 30 and 48 resources, each one with 10, 15 and 20 periods. Half of the locations are workspaces and the other half are depots. Each depot has a maximum capacity of three resources, and the number of required resources per

activity varies between 1 and 3. Lastly, the number of activities ranges between 6, for small instances, and 87, for larger instances.

All parameters involved in the proposed techniques were determined with the aid of preliminary testing. Each instance was tested ten times with each algorithm (VNS and ILS) using different seeds. VNS requires a single parameter, which is the iteration number (fixed as 1,500), while ILS needs two: the number *max_times* of applications of a given perturbation level (fixed as 20) and the accepted number *max_iter* of iterations without a cost improvement (fixed as 700). Both values affect the execution time of the algorithm.

The algorithms were written in C, using the GCC 4.2.3 compiler with –O3 option. We used a computer with an Intel® Core™2 Quad Processor Q6600 with 2.40 GHz, 4 Gbytes of RAM, and the Linux 2.6.24-19 operating system.

The computer reported in [6], a Pentium IV 2.4 GHz, has an estimated power of 4595 MFlops (http://www.activewin.com/reviews/hardware/processors/intel/p424ghz/benchs.shtml), while our computer has an estimated power of 44300 MFlops (http://techgage.com/print/intel_core_2_quad_q6600). Thus, in order to perform a fair comparison between the computational times, we use the rate of 4595/44300 Mflops to adjust the best literature times (fifth column of Table 2). All the computational times are expressed in seconds.

Optimal solutions were reported in the references for 25 instances, P01–P24 and P27. We solved the formulation described in [8] using Cplex 11 and, besides the previous known optimal values, we were able to find optimal solutions to instances P25, P26, P28, P29, P30, P31, P32, P35, P36, P39 and P40. Surprisingly, for instances P25, P26, P28, P32, P35, P36, P39 and P40, the best known upper bounds (heuristic algorithm solutions) reported in [6] were smaller than the optimal solutions found by Cplex 11.

Table 1 reports the results for the instances solved by Cplex 11, comparing the solutions and computational times for Cplex 11, the literature, and VNS and ILS algorithms. The first column presents the instances and the second column presents the solutions found by Cplex 11, followed in the third column by their computational times. The fourth column presents the best solutions found by the literature, followed by their computational times, in the fifth. The sixth column presents the best solutions found by VNS procedure, followed in the seventh column by their computational times. Finally, the eighth column presents the best solutions found by ILS procedure, followed in the ninth column by their computational times. The results are presented in two sections for the sake of space. Best solutions are set off in **bold** typeface. Those values reported by [6] that are lower than those found by Cplex are indicated by *underlined italics*. The last line in Table 1 shows the average processing time for each algorithm.

Inst.	Cplex	T_Cplex	Lit.	T_Lit.	VNS	T_VNS	ILS	T_ILS	Inst.	Cplex	T_Cplex	Lit.	T_Lit.	VNS	T_VNS	ILS	T_ILS
P01	**16**	0,3	**16**	0,8	**16**	1,9	**16**	0,9	P19	**46**	26,6	**46**	5,2	**46**	5,1	**46**	1,5
P02	**25**	0,3	**25**	1	**25**	1,6	**25**	0,5	P20	**60**	57,8	**60**	4,7	**60**	3,9	**60**	1,1
P03	**18**	0,3	**18**	0,8	**18**	1,9	**18**	0,4	P21	**46**	46,5	**46**	4,7	47	6,1	**46**	3,9
P04	**25**	3,5	**25**	0,8	**25**	1,3	**25**	0,3	P22	**67**	103,2	**67**	4,7	**67**	5,3	**67**	1,7
P05	**16**	1,3	**16**	1,6	**16**	1,8	**16**	0,5	P23	**55**	24,6	**55**	4,2	**55**	6	**55**	1,4
P06	**27**	5,5	**27**	2,1	**27**	2	**27**	0,5	P24	**74**	32,6	**74**	3,1	**74**	4,6	**74**	0,9
P07	**16**	3,5	**16**	2,3	**16**	1,8	**16**	0,4	P25	**31**	59551	*30*	7,5	**31**	13,7	**31**	5,4
P08	**31**	0,9	**31**	1,6	**31**	1,1	**31**	0,3	P26	**43**	20663,1	*42*	8,3	**43**	13,8	**43**	11,3
P09	**25**	6,8	**25**	1,8	**25**	3,6	**25**	1,1	P27	**43**	1008,4	**43**	4,9	**43**	14,3	**43**	11
P10	**46**	19	**46**	2,1	**46**	3,4	**46**	1,1	P28	**55**	582,3	*54*	4,4	**55**	9,3	**55**	6,3
P11	**32**	7,7	**32**	1,8	**32**	3,1	**32**	0,9	P29	**29**	94397,6	**29**	10,2	**29**	13,9	**29**	5,1
P12	**41**	15	**41**	2,1	**41**	2,3	**41**	0,7	P30	**49**	27609,2	50	8,6	**49**	14,5	**49**	9,7
P13	**28**	11	**28**	2,6	**28**	3,6	**28**	0,9	P31	**42**	1950,3	**42**	7,8	43	14,7	43	6,4
P14	**45**	18,9	**45**	1,8	**45**	3	**45**	1,4	P32	**69**	3716,4	*66*	7,3	**69**	10,3	**69**	4,3
P15	**35**	17,8	**35**	2,1	**35**	3,6	**35**	0,7	P35	**73**	790670,6	*68*	12,8	74	30,4	76	17,7
P16	**49**	4,5	**49**	1,8	**49**	2,8	**49**	1,1	P36	**95**	57012	*90*	117,9	96	19,8	**95**	7,5
P17	**35**	16,2	**35**	7,5	**35**	5,9	**35**	2,3	P39	**68**	177400,5	*67*	13	**68**	28	71	19,1
P18	**60**	62,3	**60**	4,9	61	5,5	**60**	2,7	P40	**108**	211216,3	*104*	15,1	**108**	18,6	**108**	13,1
Aver.	-	10,8	-	2,2	-	2,8	-	0,9	-	-	80337,2	-	13,6	-	12,9	-	7,1

Table 1 – Results of Cplex, literature, VNS and ILS.

When comparing the proposed algorithms, we see that ILS obtained the optimal value for 33 out of the 40 instances, while VNS obtained 31 optimals. The average percent deviation from the optimum for VNS and ILS was respectively 0,24% and 0,30%. This deviation is calculated as

$$\sum_{i=1}^{n} (\cos t[i] - lower\cos t[i]) / lower\cos t[i] * 100) / n ,$$

where *n* is the number of instances, *cost [i]* is the cost obtained by the algorithm and *lowercost[i]* is the the best (or optimal) cost for the instance i, found by the algorithms being compared. The lesser is the per cent average deviation, the better is the algorithm on average.

It is important to observe the time increases of the exact algorithm with the instance order. As for the heuristics, both the literature one and the VNS have an average time of 8 seconds and the ILS, of 4 seconds.

Table 2 shows the comparison between the best solution achieved by our VNS and ILS algorithms and the best results reported in [6], for the 60 instances where the optimal value is not known, i.e., P33, P34, P37, P38 and from P41 to P96. The first column shows the instance, the second column shows the best solution reported in [6], followed in the third column by its computational time. The fourth column shows the best cost obtained by the proposed VNS algorithm, followed by its time, in the fifth. The sixth column shows the best cost obtained by the proposed ILS algorithm, followed by its time, in the seventh. Finally, the last line in Table 1 shows the average processing time for each algorithm. The results are again presented in two sections for the sake of space.

Inst.	Lit.	T_Lit.	VNS	T_VNS	ILS	T_ILS	Inst.	Lit.	T_Lit.	VNS	T_VNS	ILS	T_ILS
P33	**53**	17,2	57	29,0	55	16,9	P67	157	119,5	155	296,0	**150**	114,3
P34	**72**	20,6	73	26,2	74	21,3	P68	234	84,6	227	356,8	**224**	112,8
P37	**47**	12,0	50	29,0	50	17,7	P69	**112**	112,5	119	407,0	118	170,0
P38	**77**	26,6	82	23,5	82	33,3	P70	178	262,9	176	336,7	**173**	103,0
P41	**78**	25,0	80	50,1	**78**	50,0	P71	170	176,2	173	324,7	**169**	96,3
P42	**104**	20,0	106	44,3	**104**	22,4	P72	265	128,1	252	318,7	**247**	212,1
P43	**110**	267,9	112	48,8	**110**	50,1	P73	74	130,4	**71**	402,9	**71**	135,0
P44	**137**	20,8	143	31,1	140	32,8	P74	97	106,2	**92**	383,6	96	193,6
P45	**66**	47,6	68	49,8	71	42,4	P75	110	116,4	**105**	406,3	109	300,7
P46	**111**	39,6	116	48,3	115	53,1	P76	155	80,4	155	307,1	**150**	144,2
P47	**111**	33,8	118	43,1	116	26,7	P77	73	97,4	**70**	445,3	**70**	241,7
P48	**169**	23,7	171	31,8	171	19,9	P78	101	163,2	**97**	458,5	98	668,9
P49	45	25,3	**44**	59,9	**44**	23,2	P79	110	117,7	**109**	443,9	110	142,0
P50	63	25,5	60	54,5	**59**	53,7	P80	175	87,7	171	334,7	**164**	133,6
P51	**55**	24,0	**55**	56,2	56	19,9	P81	119	266,8	**117**	1114,8	118	472,5
P52	98	20,8	**89**	37,6	**89**	17,6	P82	176	240,8	181	817,7	**168**	738,4
P53	49	36,2	**47**	69,8	**47**	28,2	P83	192	357,4	**191**	1356,2	197	783,8
P54	67	24,2	**63**	67,0	**63**	86,7	P84	**282**	247,8	291	885,0	287	339,4
P55	63	22,6	**60**	93,3	63	28,5	P85	**125**	343,9	126	1131,8	**125**	516,1
P56	97	27,6	90	67,1	**89**	34,2	P86	**192**	540,5	193	1010,3	196	528,9
P57	**67**	81,2	**67**	202,8	69	73,9	P87	193	461,8	199	890,1	**191**	234,4
P58	106	106,2	102	167,3	**96**	90,7	P88	302	342,3	**292**	910,0	**292**	626,0
P59	101	96,6	**96**	169,3	97	45,7	P89	**171**	447,5	**171**	1679,3	174	1303,0
P60	159	70,8	**154**	121,0	155	124,7	P90	262	696,7	275	1782,5	**256**	1214,5
P61	82	120,5	**75**	209,6	**75**	73,5	P91	**284**	400,7	295	1632,3	287	901,4
P62	129	100,2	**117**	233,1	**117**	119,8	P92	395	1458,4	395	2316,7	**374**	728,0
P63	121	68,2	121	189,8	**120**	95,3	P93	189	770,3	199	2003,1	**185**	514,3
P64	190	68,5	180	149,7	**176**	85,5	P94	**281**	887,0	288	1464,6	282	428,6
P65	105	106,5	105	370,7	**101**	192,9	P95	**318**	717,2	337	1522,8	328	568,5
P66	156	203,1	152	309,6	**150**	117,7	P96	**464**	663,6	482	1588,7	479	368,1
Aver.	-	59,4		102,8		56,6			354,2		910,9		434,5

Table 2 – Best results of literature and our proposed algorithms,

With this set of instances, the literature algorithm obtains better results for 23 instances. The proposed VNS and ILS obtain better costs for 21 and 32 instances, respectively. The corresponding per-cent average deviations are respectively 2.90%, 2.98% and 1.38% from the best solution obtained by the literature, VNS and ILS respectively.

With respect to the processing time, the greater average times are those of VNS (507 seconds). The literature and ILS algorithms presented averages of 207 and 246 seconds respectively.

4 Conclusions and future work

The DSAP problem is relatively new in the literature and it can model many important real-life problems where rearranging resources is a difficult and/or expensive task. It is also significant due to its relations with the well-known hard combinatorial problem QAP.

Two heuristic procedures based in VNS and ILS were proposed in this paper. The computational results obtained were comparable to the best algorithms previously existing in the literature. The heuristic based in ILS was able, on average, to yield solutions with better costs than the best solutions produced earlier by the algorithms available in the literature.

It is also important to notice that by using Cplex 11 over a formulation previously defined in [8] we were able to correct the upper bounds P25, P26, P28, P32, P35, P36, P39 and P40, which were incorrectly presented in the literature previously. We were also able to solve to optimality some open instances.

As directions for future research, we suggest: testing other heuristic procedures; exploiting other related problems, as pointed out in [6], while considering the addition of preparation costs to the data set available in the literature and comparing the results obtained from all the proposed heuristics.

Acknowledgements

We are grateful to CNPq/CT-Info and Universal, CAPES, FAPEMIG and FAPERJ for their support of this work.

References

[1] Blum, C., Roli, A.,Metaheuristics in combinatorial optimization: Overview and conceptual comparison, **ACM Comput. Surv.**, 35 (2003), 268-308. ISSN: 0360-0300. doi: http://doi.acm.org/10.1145/937503.937505.

[2] Hansen, P., Mladenovíc, N. , Variable Neighborhood Search. In: Glover, F., Kochenberger, G. (Eds.), Handbook of Metaheuristics, **Kluwer Academic Publishers**, 145-184, (2003).

[3] Lee, C-G., Ma, Z., The generalized quadratic assignment problem, **Working paper**, U. of Toronto, Dept. of Mechanical and Industrial Engineering, (2005).

[4] Loiola, E. M., Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., Querido, T.M., A survey for the quadratic assignment problem, **European Journal of Operational Research**, 176(2007), 657–690.

[5] Lourenço, H. R., Martin, O., Stützle, T. (2003). Iterated Local Search. In: Glover, F., Kochenberger, G. (Eds.), Handbook of Metaheuristics, **Kluwer Academic Publishers**, 145-184, (2003).

[6] McKendall Jr., A., Improved tabu search heuristic for the dynamic space allocation problem, **Computers & Operations Research**, 35(2008), 3347–3359.

[7] McKendall Jr., A. e Jaramillo, J., A tabu search heuristic for the dynamic space allocation problem, **Computers & Operations Research**, 33(2008), 768–789.

[8] McKendall Jr., A., Noble, J., Klein, C., Simulated annealing heuristics for managing resources during planned outages at electric power plants, **Computers & Operations Research**, 32(2005), 107–125.

[9] Mladenovíc, N., Hansen, P., Variable neighborhood search, **Computers & Operations Research**, 24(1997), 1097–1100.

[10] Urban, T., Solution procedures for the dynamic facility layout problem, **Annals of Operations Research**, 76(1998)323-342.