EngOpt 2008 - International Conference on Engineering Optimization
Rio de Janeiro, Brazil, 01 - 05 June 2008.

An Efficient Algorithm for the Dynamic Space Allocation Problem

Geiza Cristina da Silva
Tiago Geraldo Ferreira
Tatiana Alves Costa
Universidade Federal de Ouro Preto
Departamento de Ciências Exatas e Aplicadas
Rua 37, 115 – Loanda
35930-970 João Monlevade – MG – Brasil
{geiza, tatiana}@decea.ufop.br
tiago.g.f@gmail.com

Paulo Oswaldo Boaventura Netto
Universidade Federal do Rio de Janeiro
Programa de Engenharia de Produção
Centro de Tecnologia, Sala F113
21949-900 Rio de Janeiro – RJ- Brasil
boaventu@pep.ufrj.br

Luiz Satoru Ochi
Universidade Federal Fluminense
Departamento de Ciência da Computação
Rua Passo da Pátria 156 – Bloco E – Sala 328
24210-240 Niterói – RJ – Brasil
satoru@ic.uff.br

Abstract

The problem of dynamic space allocation (DSAP) was recently formalized in the literature. It was inspired by the need of optimizing the distances traveled by the resources associated to the activities of a project. Owing to its highly combinatorial nature it is difficult to solve exactly and even a feasible solution is not straightforward. In this work we propose new heuristic algorithms combining construction and refinement techniques. The results of these heuristics show them as being competitive for the available instances from the literature.

Keywords: Problem of Dynamic Space Allocation, Tabu Search, Metaheuristics.

1. Introduction

Heuristic techniques are used to solve problems of artificial intelligence when one of the following situations arises (Luger 2004): either the problem may not have an exact solution owing to inherent ambiguities in its formulation or because there are missing or unavailable data; or the problem may have an exact solution, but the computing cost of determining it is prohibitively high. The Dynamic Space Allocation Problem (DSAP) is a combinatorially explosive problem of the second type, where the number of possible states grows exponentially with search depth. In these cases, brute-force exhaustive search techniques, both in depth or in breadth, may not arrive at a solution within an acceptable time. The use of heuristic techniques to deal with this complexity by guiding searches through the solution space along promising paths is thus justified.

2. The dynamic space allocation problem (DSAP)

DSAP can be described as follows: given a project divided into periods, a set of activities has to be completed in each period. Each activity demands a set of necessary *resources* in order to be accomplished, with the resources not used in a period by any of its activities being considered to be *idle*. The problem is how to associate resources with appropriate locations, that is, associate resources used by activities with *workspaces* and idle resources with *storage spaces* or *depots*, in such a manner that the total distance traveled by the resources between locations is kept to a minimum.

The problem was first conceived for resource management during the construction of electrical power distribution grids, it was later found to have potential applications in projects where moving resources is difficult or where resource congestion is undesirable, such as bridge construction and mining activities.

In every case, space is a scarce resource which makes the reduction of materials handling a key factor, through the efficient allocation of activities and active or idle equipment and building materials, with efficient use of space being fundamental to reducing

congestion.

Despite the fact that the DSAP is a recently formulated problem, its importance can be justified by applications such as the above. From a theoretical point of view, the relevance of DSAP derives from its association with two widely-studied combinatorial optimization problems, the dynamic facility layout problem (DFLP) and the quadratic assignment problem (QAP).

The QAP is one of the most difficult among NP-hard problems [6]. It can be defined as the allocation of *n* facilities to *n* locations. The distances between pairs of locations and the flows of some type of demand being known, the objective is to find a minimum cost allocation.

The DFLP aims to minimize the cost of materials flow and the reallocation cost of each facility to a specific location at multiple periods. The problem can then be seen as a succession of QAP's, that is, a QAP has to be solved for each period.

A quick bibliography search produced little work related to DSAP. In [9] the problem was defined and a mathematical programming formulation was used together with a solver to obtain exact solutions for some of the proposed cases. Two instances based on simulated annealing were also proposed in order to obtain approximate solutions.

In [8] construction algorithms and a tabu search were proposed and the results thus obtained were compared with those in the literature. The article reported that tabu search outperformed the latter ones both in solution quality and computing time.

Problem data are:

- The *project agenda*, where the *total number* of activities, which activities have to be completed within each period and the required resources for each activity are specified;
- The *locations* (workspaces and depots), the *distances* between them and the capacity of each location, that is, the quantity of resources each one can host.

The problem definition assumes that a single activity can be associated with each workspace in a given period and that the capacity of this workspace is sufficient to accommodate its associated resources. In addition, each activity requires only one workspace and at least one resource.

In order to avoid unnecessary resource movement it is assumed that, if an activity is carried out during two or more consecutive periods, it should stay at the same workspace. In a similar manner, a resource which remains idle during multiple consecutive periods should be associated with the same depot during all of them.

A DSAP solution is then composed of two matrices: an *activity matrix* that associates periods, workspaces and activities, and an *idle resource matrix* that describes periods, depots, idle resources and the total distances to be traveled by resources.

In the following example, Table 1 presents the agenda for a small instance case with 3 periods, 4 activities and 8 resources.

Table 1. Scheduling of activities

Periods	Activities	Resources	Idle Resources
1	A1	4,5,8	6,7
	A2	1,23	
2	A2	1,23	4,5,8
	A3	6,7	
3	A3	6,7	1,2,3,5
	A4	4,8	

Table 2 shows the distances among the locations, where locations 1 to 3 are workspaces and locations 4 to 6 are depots. We take as a given in this instance that each location can support up to 3 resources.

Table 2. Distance matrix among locations

	1	2	3	4	5	6
1	0	1	2	1	2	3
2	1	0	1	2	1	2
3	2	1	0	3	2	1
4	1	2	3	0	1	2
5	2	1	2	1	0	1
6	3	2	1	2	1	0

Tables 3 and 4 present an instance solution for one resource's total distance traveled of 10 units. Table 3 shows the allocation of activities and their resources to workspaces (ws_i) and Table 4 shows the association between idle resources and depots (d_i).

Table 3. Activity matrix

Periods	ws_1	ws_2	ws_3

Periods	ws_1	ws_2	ws_3
1	A1(4,5,8)	A2(1,23)	
2		A2(1,23)	A3(6,7)
3	A4(4,8)		A3(6,7)

Table 4. Idle resources matrix

Periods	d_1	d_2	d_3
1			6,7
2	4,5,8		
3	5	1,2,3	

3. Mathematical Formulation

The DSAP can be written as quadratic integer programming by using the following notation:

Index sets:

- J = total number of activities (j = 1, 2, ..., J);
- T = total number of resources (t = 1, 2, ..., T);
- P = total number of periods (p = 1, 2, ..., P);
- N = total number of locations (workspaces and depots);
- L = location set, $|L|$ = N (L = 1, 2, ..., N);
- W = workspace set, where $W \in L$ ($w \in W$);
- S = depot set, onde $S \in L$ ($s \in S$) e $W \in S = L$;
- Rj = resource set for activity j to be accomplished;
- Ip = idle resource set during period p;
- Ap = activity set during period p.

Start parameters:

- dkl = distance between locations k and l;
- Cs = capacity of depot s;
- NRj = number of resources associated with activity j.

Formulation:

Minimize:

$$\sum_{t=1}^{T}\sum_{k=1}^{N}\sum_{l=1}^{N}\sum_{p=1}^{P-1} d_{kl}\, x_{ptk}\, x_{p+1tl} \qquad (1)$$

Sujeito a:

$$\sum_{\forall s \in S} x_{pts} = 1, \forall p, \forall t \in I_p, \qquad (2)$$

$$\sum_{\forall s \in S} x_{pts} \le C_s, \forall s \in S, \forall p, \qquad (3)$$

$$\sum_{\forall w \in W} y_{jw} = 1, \forall j, \qquad (4)$$

$$\sum_{\forall j \in A_p} y_{jw} \le 1, \forall w \in W, \forall p, \qquad (5)$$

$$\sum_{t \in R_j} x_{ptk} = NR_j\, y_{jw}, \forall p, \forall j \in A_p, \forall w \in W, \qquad (6)$$

$$x_{ptk} = 0 \text{ ou } 1, \forall p, \forall t, Ak \in L, \qquad (7)$$

$$x_{jw} = 0 \text{ ou } 1, \forall j, \forall w \in W. \qquad (8)$$

The objective function (1) minimizes the resources' transportation cost between workspaces and depots during the periods when the activities will be carried out. Constraints (2) guarantee that every idle resource will be allocated to a depot during a period and

constraints (3) guarantee that that depot's capacity will be respected. Constraints (4) reserve a workspace for each activity and constraints (5) guarantee that an activity will occupy one workspace. Equation (6) associate the resources used by a given activity to the same workspace to which it was allocated. Constraints (7) and (8) indicate that the problem variables are binary.

4. Construction heuristic for the DSAP

The construction heuristic presented in this section can be divided into two parts. The first is an adaptation of the constructive randomized heuristic (CRH) proposed in [10]. This heuristic, hereafter denoted as ACRH, aims to generate an initial solution for the DSAP, corresponding to an activity association. The second part, called *randomized storage policy* (RSP), was proposed in [8] and deals with the association of idle resources with depots.

ACRH is based on cluster generation: the number of clusters is the number of workspaces and each one contains a set of activities. Two activities can belong to the same cluster (workspace) if their periods do not coincide.

The first pass of ACRH is used to calculate and store the three less similar pairs of activities: among them, one is randomly chosen to compose the first two clusters. The similarity measure used between two activities I and j is the number of resources they share.

Since the activities are analyzed according to the value of their labels, in order to avoid having only smaller label activities be chosen, the algorithm implements a probabilistic function which allows an activity I to be replaced by an activity j with a probability of ρ%. After each replacement, this probability decreases by μ%.

To build the following clusters, a similarity measure is calculated for each activity not yet allocated with respect to each cluster already built. Here, the measure will be the number of common resources between the activity and each cluster. The least similar activity is allocated to the cluster. If a tie occurs, the activity with the smaller label is chosen.

Finally the ARCH begins to make allocations of the remaining activities to the clusters. For each activity, a similarity measure is calculated with respect to each cluster and the activity is allocated to the most similar cluster, provided that there is no period conflict with some activity already in that cluster.

The RSP heuristic allocates the idle resources to those depots that are nearest the workspace to which an activity that will use them has been allocated.

5. A refinement heuristic for the DSAP

The refinement heuristic presented in this section applies the following activity movements defined in [8]. Movement 1 is the exchange of workspaces of two or more activities during one or more periods. Movement 2 is the removal of an activity from a workspace and its allocation to another available workspace during one or more periods. Movement 3 is the association of Movements 1 and 2.

Starting with a solution built by ARCH, the following steps are executed until a given number of iterations without improvement is attained:

- at the beginning, the algorithm evaluates every possible activity movement that can be made in the current solution;
- for each one, the RSP heuristic is executed to reallocate the idle resources in accordance with the new activity matrix;
- the best movement is executed and the new solution becomes the current one.

The proposed refinement heuristic has the possibility of being stuck to local optima. A list of the most recent movements is compiled to avoid their repetition. This strategy is based on the tabu search concept [1]; [5]; [2]; [3].

The movement list implemented in this work can be considered dynamic since its size varies between given upper and lower limits, changing at each γ iteration without improvement. This limitation was adopted because an excessively long list could bring the search to a premature end. On the other hand, too short a list could permit the search to repeat recent movements which would allow for inefficient use of space.

6. Computational results

In order to test the efficiency of our proposal, two test batteries were executed, both using the 96 instances (P01-P96) proposed in [8], kindly furnished by the authors. The heuristics were implemented on C language using Version 4.0.3 of the GNU gcc compiler. All experiments were run on Dual Intel Pentium 4 3192 Mhz computers with 1010 Mb RAM, under Ubuntu Linux 6.0.4.

The purpose of the first test battery was to test the efficiency of the proposed construction algorithm, since a good performance of a refinement algorithm is directly dependent on the initial solution it receives. The results were compared with those from the RC *(Randomized Clustering)* heuristic also described in [8]. This comparison is justified by the similar nature of RC and ARCH, that is, both are based on algorithms for manufacturing cell problems.

Table 5 presents the sum of costs obtained by each construction algorithm for instances with 6, 12, 20 and 30 locations. The graph in Figure 2 illustrates the same results.

Table 5. Comparison between costs obtained by ARCH and RC algorithms

Problems	**ACRH**	**RC**
6-locations	*1,010*	1,057
12-locations	2,548	*1,931*
20-locations	*3,482*	3,752
32-locations	*5,734*	7,238
Total	*11,764*	13,978

From these results, we can see that ARCH was able to find a total of 1204 fewer distance units than RC did. Only for 12-location

cases was the ARCH sum of costs greater than that of the algorithm from the literature. We can see that ARCH performs satisfactorily with the larger instances.

The literature reports the optimal cost for 25 of the 96 instances (P01 to P24 and P27). The ARCH heuristic is able to find the optimum value for 5 of these instances, surpassing the clustering algorithm from the literature, which obtained the optimum for only 4 of these instances.

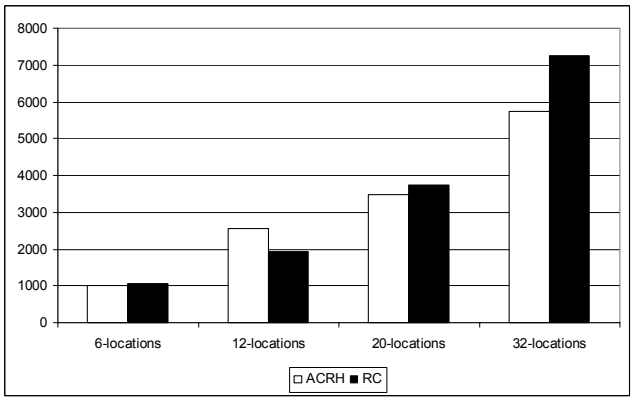

Figure 1. AHRC versus RC results

The next test battery aimed to evaluate ARCH results when combined with the proposed refinement heuristic, which thus constitute the PA (Proposed Algorithm). The configurations used by the PA algorithm are described in Figure 2.

Lower bound of the list of recent moves	0.7 x *n*
Upper bound of the list of recent moves	1.1 x *n*
Number of iterations without improvement	100
Parameter ρ	25%
Parameter μ	1%
Parameter γ	10

Figure 2. Parameters used in tests, n=number of activities

For comparison, we used the results obtained with the RC heuristic combined with the tabu search (TS) algorithm from the same authors.

Table 6 indicates a total cost reduction of 242 distance units, that is, about 2.5 units per instance. This reduction is obtained for the larger instances of 20 and 32 locations (Figure 3).

Table 6. Comparison between costs obtained by PA and RC+TS algorithms

Problems	**PA**	**RC+TS**
6-locations	*977*	969
12-locations	2,203	*1,898*
20-locations	*2,918*	2,989
32-locations	*4,593*	5,247
Total	*9,987*	10,134

Table 6 indicates a total cost reduction of 242 distance units, that is, about 2.5 units per instance. This reduction is obtained for the larger instances of 20 and 32 locations. We can thus ascertain that PA obtained better results for 35 among the 96 instances, when compared with RC+TS.

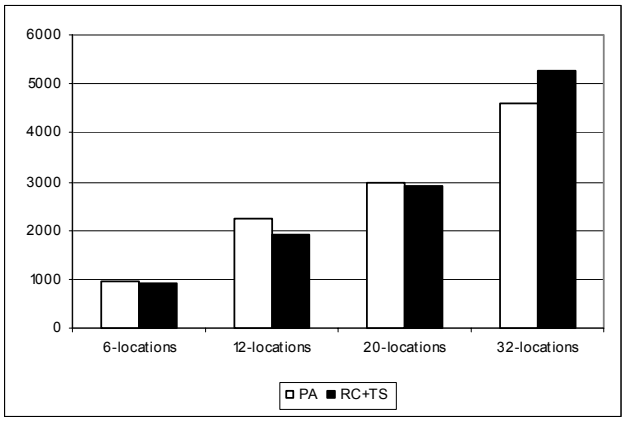

Figure 3. Results of PA compared with RC+TS

Figures 4 to 7 show the results of each algorithm with each test problem, for 6, 12, 20 and 32 locations, respectively. Through them, we can see that PA obtained equal or better results for most of the instances. Only for 12-location problems do the literature algorithms show better results. These results show that, the better the initial solution, the better the results obtained by the refining algorithm: we recall that Table 5 showed that, for the 12-location series, the ARCH algorithm was not able to surpass the algorithm from the literature, on average.

There was no comparison between computing time for the heuristics, since different machines were used in the tests. Nevertheless, it was found that for every instance the time used by PA was less than 140 seconds.

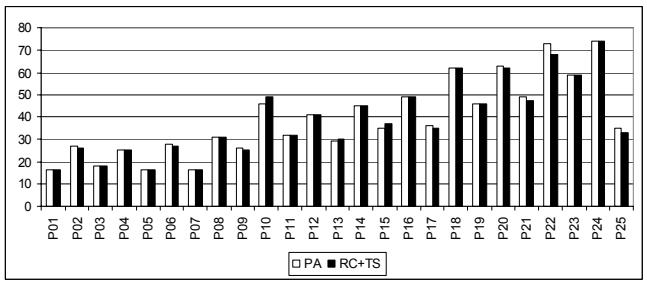

Figure 4. Results for 6-location problems

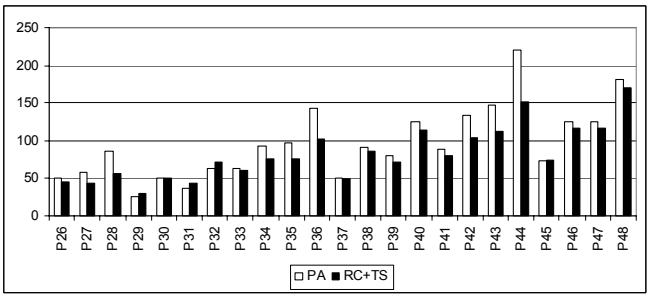

Figure 5. Results for 12-location problems

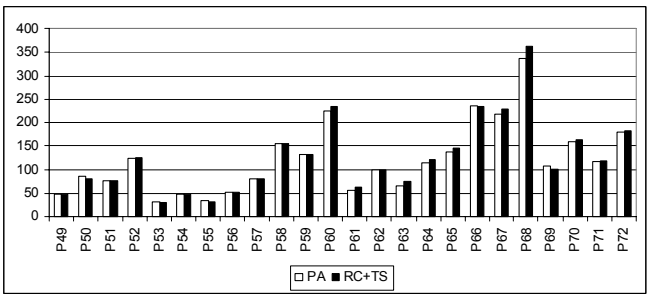

Figure 6. Results for 20-location problems

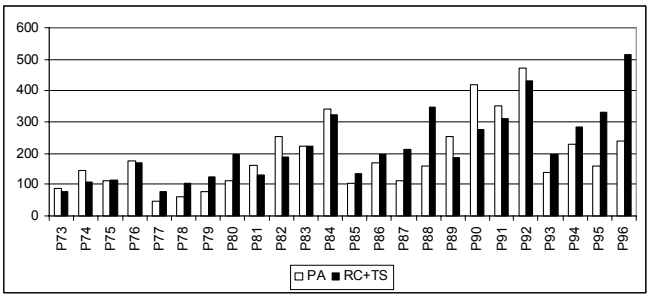

Figure 7. Results for 32-location problems

7. Conclusions

The first objective of this work was to adapt to DSAP definition a heuristic already proven to be efficient for manufacturing cell problems. We expected that a good construction heuristic could lead to better global solutions.

To verify this, we implemented the proposed construction and refinement algorithms and we used a set of instances from the literature. It was verified through extensive testing that the proposed heuristics are competitive with respect to the problem, since a number of better results were obtained in satisfactorily low computing time.

Acknowledgement

This research has been partially supported by the CT-Info/MCT/CNPq Universal 15/2007. The first author is grateful to CAPES and FAPEMIG for financial support.

References

[1] Glover, F. 1986. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research* 5: 553-549.

[2] Glover, F. 1990. Tabu Search: a Tutorial. *Interfaces* 20: 74-94.

[3] Glover, F. e Laguna, M. 1998. *Tabu Search.* Kluwer Academic Publishers, The Netherlands.

[4] Glover, F. e Laguna, M. (2000). Fundamentals of Scatter Search and Path Relinking, *Control and Cybernetics* 29: 653-684.

[5] Hansen, P. 1986. The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. In: Proceedings of Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy.

[6] Loiola, E.M.; Abreu, N.M.M.; Netto, P.O.B., Hahn, P.M. e Querido, T. 2007. A Survey for the Quadratic assignment problem, *European Journal of Operational Research* 176: 657-690.

[7] Luger, G.F. 2004. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Addison Wesley.

[8] Mckendall, A.R. & Jaramillo, J.R. 2006. A Tabu Search Heuristic for the Dynamic Space Allocation Problem. *Computers & Operations Research* 33: 768-789.

[9] Mckendall, A.R.; Noble, J.S. & Klein, C.M. 2006. Simulated Annealing Heuristics for Maninging Resources During Planned Outages at Eletric Power Plants. *Computers & Operations Research* 32: 107-125.

[10] Trindade, A.R. e Ochi, L.S. 2006. Um Algoritmo Evolutivo Híbrido para a Formação de Células de Manufatura em Sistemas de Produção. *Pesquisa Operacional* 26: 255-294 (*Int. Abstracts in Operations Research; e Statistical Theory and Methods Abstracts*).