versão impressa ISSN 0101-7438 / versão online ISSN 1678-5142

COMBINATORIAL INSTRUMENTS IN THE DESIGN OF A HEURISTIC FOR THE QUADRATIC ASSIGNMENT PROBLEM

Paulo Oswaldo Boaventura-Netto
Laboratoire Pri*S*M
Université de Versailles à St. Quentin-en-Yvelines
Versailles Cedex – França
Programa de Engenharia de Produção / COPPE
Universidade Federal do Rio de Janeiro
Rio de Janeiro – RJ
pboav@prism.uvsq.fr; boaventu@pep.ufrj.br

Recebido em 08/2002; aceito em 08/2003
Received August 2002; accepted August 2003

Abstract

This work discusses the use of a neighbouring structure in the design of specific heuristics for the Quadratic Assignment Problem (QAP). This structure is formed by the 4- and 6-cycles adjacent to a vertex in the Hasse diagram of the permutation lattice and it can be adequately partitioned in subsets of linear and quadratic cardinalities, a characteristics which frequently allows an economy in the processing time. We propose also a restart strategy and a mechanism for generating initial solutions which constitute, together with the neighbouring structure, a possible QAP-specific heuristic proposal. For the construction of these instruments we used the relaxed ordered set of QAP solutions.

Keywords: quadratic assignment problem; combinatory; heuristics.

Resumo

Este trabalho discute o uso de uma estrutura de vizinhança em heurísticas específicas para o Problema Quadrático de Alocação (PQA). Esta estrutura envolve os ciclos de comprimento 4 e 6 adjacentes a um vértice do diagrama de Hasse do reticulado das permutações e pode ser particionada em subconjuntos de cardinalidade linear e quadrática em relação à ordem da instância, o que permite frequentemente uma economia de tempo de processamento. Propõem-se ainda uma estratégia de repartida e um mecanismo de geração de soluções iniciais, que constituem, ao lado da estrutura de vizinhança, uma proposta de heurística específica para o PQA. Na construção desses instrumentos foi utilizada a noção de conjunto relaxado ordenado das soluções do PQA.

Palavras-chave: problema quadrático de alocação; combinatória; heurísticas.

1. Introduction

QAP is a NP-hard problem that has been studied by a number of researchers in combinatorial optimization through the second half of the 20th century and which continues to excite the curiosity of many others, both in the study of exact algorithms and through the formulation of information mechanisms for heuristics. The exact approach is severely limited by the high complexity of the problem, a difficult 30-order instance (Nug30) having been solved exactly only through a huge work of metacomputing. More recent years have seen an increased interest in the use of metaheuristic schemes: some examples are [AQB99], [BT94], [AOT00], [CMMT97], [Ma00], [TRFR01] and [TS95].

Specific heuristics for the QAP are normally classified as being *constructive*, *limited enumeration* and *improvement* ones.

Constructive methods build a solution by choosing an image element at each step through the use of a given criterion, until a complete solution is obtained ([BAV62], [AB63], [SWHH95], [TB98], [SWH98], [Bu91], [Lo00], [AHS01], [GY02] and [YSA03]).

Limited enumeration methods use available information to guide solution enumeration. Evidently an optimum value cannot be guaranteed unless the whole solution set is enumerated, so it is necessary to establish stopping criteria. This approach seems to have been abandoned since the eighties ([We83], [BB83]).

Improvement methods involve local search algorithms and most of the specific QAP heuristics are classified in this category. The main elements in these methods are the neighborhood definition and the order user for the analysis of the neighbors ([MO79], [Br84], [LS95], [BÇ95], [An96], [THKG98]). These methods are frequently utilized within the logic of the metaheuristics. The technique here presented can be included in this group.

We refer to [Ma00] for the discussion of QAP origins and for its definitions, from which we retain only that a *QAP instance* is a pair of matrices $(\mathbf{M}_F,\mathbf{M}_D)$ where $\mathbf{M}_F = [\phi_{ij}]$ corresponds to *flows* and $\mathbf{M}_D = [\delta_{ij}]$ to *distances* traversed by these flows in a context of n *machines* functionally connected two by two in a shop. Through this work we will consider $\mathbf{M}_F$ and $\mathbf{M}_D$ as symmetric.

In what concerns the theoretical work on QAP the majority of the studies concerned the definition of better lower bounds for the optimal solution to be used in exact algorithms, such as in [KÇCE00]. A characterization of some polynomial cases has also been presented [Çe98].

The approach used in this paper is an algebraic and combinatorial one (AC) [Ab84], which considers some properties of the permutation set and its graph-theoretical description in order to propose three basic instruments which constitute together a proposal of a specific heuristic. The AC approach has already been used to inform metaheuristics such as simulated annealing [QAB99] and GRASP [RAB00] and also in the definition of a new difficulty measure for QAP instances [ABQG02].

The AC approach associates the solution set of a QAP instance to the set $\mathbf{S}_n$ of n-permutations. Graph-theoretical models are applied to allow the definition of a neighbourhood structure [LB01]. The corresponding definitions and notation follow [Be73] and [Bo01]. A relaxing and ordering scheme is applied to $\mathbf{S}_n$ which allows to the definition of a generator of initial solutions and a restart strategy. This whole set of instruments constitutes a QAP-specific heuristic algorithm.

Sections 2 and 3 present some concepts, definitions and instruments from AC. Sections 4 and 5 discuss some specific questions related to the proposed neighbourhood. Section 6 is related to the algorithm construction and finally Section 7 presents some results of tests with QAPLIB instances [BKR97].

2. A relaxed set and the question of feasibility

2.1 A graph-theoretical model

Let $\varphi \in \mathbf{S_n}$ be a permutation,

$$\varphi = (\varphi_1, \varphi_2, \ldots, \varphi_n) \tag{2.1}$$

and consider the function

$$\psi_{ij} = (i - 1)n - i(i + 1)/2 + j \qquad i < j, \tag{2.2}$$

where ψ_{ij} is the numerical ordering of the couple (i,j) within a lexicographical ordering.

We can then define a N-order permutation (where $N = C_{n,2}$), corresponding to φ:

$$\xi = (\xi_1, \xi_2, \ldots, \xi_N) \tag{2.3}$$

whose k^{th} element, $k = \psi_{ij,}$ is, with $\varphi(i) = \varphi_i$,

$$\xi_k = \xi_{\psi(\varphi(i),\varphi(j))} \tag{2.4}$$

a permutation of *couples* of elements from φ.

Let $\mathbf{S_N}$ be the set of permutations with N elements. Every solution from $\mathbf{S_n}$ has a corresponding solution within $\mathbf{S_N}$. The reciprocal is not true and we have then to distinguish *feasible* and *infeasible* solutions in $\mathbf{S_N}$. This situation is more easily examined through a graph-theoretical model where we associate $\mathbf{M}_F$ and $\mathbf{M}_D$ respectively to complete undirected graphs $\mathbf{K}_F$ and $\mathbf{K}_D$, whose edges will be valued after the corresponding entries in the matrices (Figure 1):

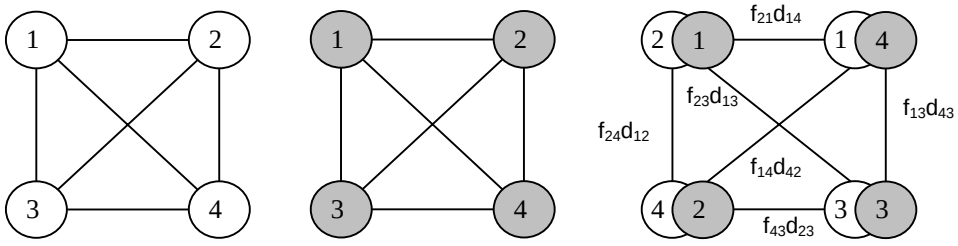

Figure 1 – A graph model for QAP relaxation

In this model the elements of a permutation set as defined in (2.1) will be *vertex-permutations* and those following (2.3) will be *edge-permutations*, so we will use these designations from now on. We will call $\mathbf{S_N}$ the *relaxed set* of $\mathbf{S_n}$.

The infeasibility of most solutions corresponds to the fact that when we make a vertex exchange all edge extremities concerned with the exchanged vertices have also to exchange.

Then every edge permutation having not a coherent allocation of edge extremities will not be associated to a vertex permutation, so it will not be feasible. It is important to observe that instance *values* have nothing to do with this question.

2.2 The feasibility matrix

We can define a *feasibility matrix* in order to verify if a given solution $\xi \in \mathbf{S}_N$ is feasible or not. To obtain it we can observe that an edge (i,j) from $\mathbf{K}_D$ can be associated to an edge (k,l) from $\mathbf{K}_F$ either by associating k to i and l to j, or by associating k to j and l to i. So we build a matrix $\mathbf{M}_\xi$ whose elements are the sums of the association possibilities for every pair (i,j), (k,l) of edges. If a relaxed solution ξ_k is feasible we will find in this matrix n independent positions with value n – 1.

For this calculation we use the inverse function $(\psi_{ij})^{-1}$ to recover the vertex couples defining the edges.

Example: $\xi = (3\ 6\ 5\ 2\ 1\ 4) = \begin{pmatrix} 12 & 13 & 14 & 23 & 24 & 34 \\ 14 & 34 & 24 & 13 & 12 & 23 \end{pmatrix}$

Let's sum a unity over a null 4 x 4 matrix to the positions

from ξ_1 : (1,1), (1,4), (2,1), (2,4); from ξ_2 : (1,3), (1,4), (3,3), (3,4);

from ξ_3 : (1,2), (1,4), (4,2), (4,4); from ξ_4 : (2,1), (2,3), (3,1), (3,3);

from ξ_5 : (2,1), (2,2), (4,1), (4,2); from ξ_6 : (3,2), (3,3), (4,2), (4,3).

The final matrix is

$$M_\xi = \begin{array}{|cccc|} \hline 1 & 1 & 1 & 3 \\ 3 & 1 & 1 & 1 \\ 1 & 1 & 3 & 1 \\ 1 & 3 & 1 & 1 \\ \hline \end{array} \quad \text{then } \varphi = (\,4\ \ 1\ \ 3\ \ 2\,)$$

Here we found n – 1 edges associated with each vertex and the n independent positions with this value correspond to a vertex permutation. The solution is feasible.

If we take, on the other hand, the permutation $\xi = (2\ 1\ 4\ 3\ 6\ 5) = \begin{pmatrix} 12 & 13 & 14 & 23 & 24 & 34 \\ 13 & 12 & 23 & 14 & 34 & 24 \end{pmatrix}$, we will obtain

$$M_\xi = \begin{array}{|cccc|} \hline 2 & 2 & 2 & 0 \\ 2 & 0 & 2 & 2 \\ 2 & 2 & 0 & 2 \\ 0 & 2 & 2 & 2 \\ \hline \end{array}$$

where the matrix structure does not show any association with a vertex permutation. This solution is infeasible.

It is convenient to observe that the element sum is constant for every row and for every column, only the element distribution changes from a solution to another.

3. The ordering by value within an instance

Let we consider an instance $(\mathbf{M}_F,\mathbf{M}_D)$. The *cost* of a solution $\varphi \in S_n$ from $(\mathbf{M}_F,\mathbf{M}_D)$ is

$$z(\varphi) = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} f_{\varphi(i)\varphi(j)} \qquad \forall \varphi \in S_n \tag{3.1}$$

For the symmetric QAP we can define N-component vectors $\mathbf{F} = [f_i]$ and $\mathbf{D} = [d_j]$ containing the values of the upper triangles of $\mathbf{M}_F$ and $\mathbf{M}_D$. Then we can obtain their product [GP66]:

$$\mathbf{Q} = \mathbf{F}\mathbf{D}^T. \tag{3.2}$$

The N x N matrix $\mathbf{Q}$ contains every cost parcel for every $\varphi \in \mathbf{S}_n$. The cost is associated to the elements of the permutation $\xi \in \mathbf{S_N}$ corresponding to φ,

$$z(\xi) = \sum_{i=1}^{N} d_i f_{\xi(i)} \qquad \forall \xi \in S_N \tag{3.3}$$

It is possible to define a partial order by value on $\mathbf{S}_N$. We order $\mathbf{F}$ and $\mathbf{D}$ by opposite orders, for instance $\mathbf{F} \rightarrow \mathbf{F}^+$ (non-decreasing) and $\mathbf{D} \rightarrow \mathbf{D}^-$ (non-increasing). Then we can define a new matrix,

$$\mathbf{Q^*} = (\mathbf{F}^+)(\mathbf{D}^-)^T \tag{3.4}$$

whose *trace* $\sum q_{ii}$ (i = 1,…,N) is an *absolute lower bound* for the instance (we can also observe that the *opposite trace* $\sum q_{i,N+1-i}$ (i = 1,…,N) is an *absolute upper bound* for it).

We have a new (*ordered*) solution set which corresponds to a new permutation lattice. To distinguish it from the former we will respectively denote them $\mathbf{S_N}(\mathbf{Q^*})$ and $\mathbf{S_N}(\mathbf{Q})$.

Figure 2 shows a scheme of $\mathbf{S_N}(\mathbf{Q^*})$, a polygon, as a pictorial representation of level cardinalities: the upper and lower vertices of the polygon correspond to the single-solution extreme levels and between them the level cardinality grows from 1 (at N_0) through the lower half of the figure, goes to a maximum in the middle and shrinks to 1 (at N_N) through its upper half. For more details see Item 4.2 and Eq. 5.4 below.

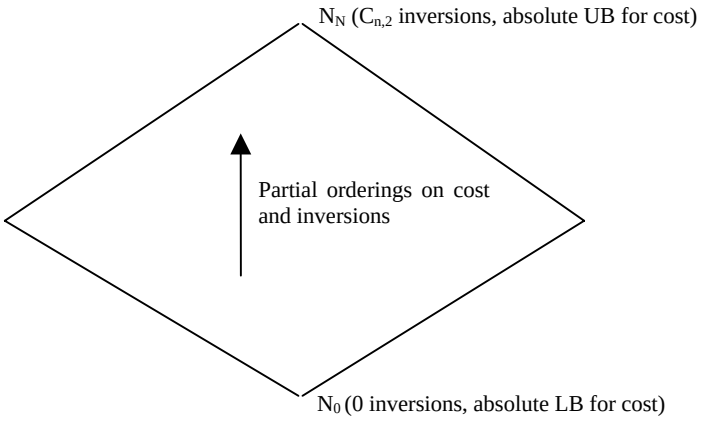

Figure 2 – Orderings and bounds on $S_N(Q^*)$

4. The Hasse diagram of the permutation set

4.1 The permutation lattice

The n-order permutation set $\mathbf{S}_n$ can be described as a *lattice* $(\mathbf{S}_n, \leq)$ [Be68] where the partial ordering is given by the *number of inversions*, that is, the number of times an element of a permutation has another element lesser than it in a more advanced position, for each element concerned. The *Hasse diagram* of this lattice is an undirected graph $\mathbf{G}_n = (\mathbf{S}_n, \mathbf{U})$ where $\mathbf{U}$ is the set of permutation pairs differing between them by one and only one inversion. This (partial) ordering is not the same as that of $\mathbf{S}_N(\mathbf{Q}^*)$ but it is close enough in most instances to allow its use as a guide for finding better solutions. That is, we can think of inversion reduction as a strategy for getting better costs.

The number of inversions related to a given element $\pi_i \in \pi$ will be

$$x(\pi_i) = |\{\pi_j \mid j > i,\ \pi_j < \pi_i\}. \tag{4.1}$$

and the inversion number $\nu(\pi)$ of a permutation will be the sum of the inversion numbers related to their elements,

$$\nu(\pi) = \sum_{i=1}^{n} x(\pi_i)\ . \tag{4.2}$$

It is convenient to define an inversion at the positions (i,j) as corresponding to the effect of an *inversion operator* τ_{ij} which exchanges the positions of the i^{th} and j^{th} elements of a permutation.

4.2 Some properties of $\mathbf{G_n}$

Most of the properties here discussed was presented in [LB01] but the discussion made here has been elaborated in some details.

Property 1: $\mathbf{G_n}$ is a regular graph of degree n – 1.

> ***Proof.*** (Immediate) ■

Property 2: $\mathbf{G_n}$ is bipartite and we can partition $\mathbf{S}_n$ by the number of inversions of its elements.

> ***Proof.*** We can define subsets $\mathbf{N}(\nu_i) \subset \mathbf{S}_n$ $(i = 0, \ldots, N = C_{n,2})$ such that every permutation $\varphi \in \mathbf{N}(\nu_i)$ has ν_i inversions. As every permutation has a unique inversion number these subsets constitute a partition of $\mathbf{S}_n$. Owing to the definition of $\mathbf{G_n}$, there are no edges within vertices of the same level and every $\mathbf{G_n}$ edge will connect vertices of consecutive levels. Then we can designate even and odd colors to the $\mathbf{N}(\nu_i)$ according to their index, in the sense of vertex coloring. So $\mathbf{G_n}$ is bipartite. ■
>
> **Obs.:** The zero level $\mathbf{N}_0$ and the opposite level $\mathbf{N}_N$ have exactly one element (resp. identity and opposite permutations).

Property 3: We can define a level set for $\mathbf{G}_n$ having in $\mathbf{N}_0$ a given permutation.

> ***Proof.*** We can take a permutation $\varphi \in \mathbf{S_n}$ and do the necessary inversions to obtain a first level, a second level and so on. The content of the image is irrelevant. ■
>
> ***Remark.*** We will use this property to define a level set on $\mathbf{G}_n$ and we will then speak of a level set *related* to a given permutation φ.

Property 4: $\mathbf{G_n}$ has girth $g(\mathbf{G_n}) = 4$.

Proof. $\mathbf{G_n}$ is a 1-graph, so there are no 2-cycles. As $\mathbf{G_n}$ is bipartite, it has no odd cycles. Finally, the successive operations τ_{ij}, τ_{kl}, τ_{ji} and τ_{lk} (i, j consecutive, k, l consecutive, $\{i,j\} \cap \{k,l\} = \varnothing$) define a 4-cycle which establish the girth value. ■

Example: For n = 4 we have $(1234) \rightarrow (2134) \rightarrow (2143) \rightarrow (1243) \rightarrow (1234)$.

Property 5: A vertex $\varphi \in \mathbf{S_n}$ belongs to $C_{n-1,2}$ adjacent 4- and 6-cycles, from which $C_{n-2,2}$ 4-cycles and n – 2 6-cycles.

Proof. From Property 1 the immediate neighbouring $\Gamma(\varphi)$ has n – 1 vertices. A cycle adjacent to φ will have two vertices φ_1 and φ_2 in $\Gamma(\varphi)$. Then there will be $C_{n-1,2}$ ways to choose a pair of these vertices. From these, there will be n – 2 intersecting pairs made of consecutive triples and $C_{n-1,2} - (n - 2) = C_{n-2,2}$ disjoint pairs which, by Property 4, give way to 4-cycles. The remaining n – 2 cases being those of φ-element triples we can have 6 permutations of their positions, giving 6 different solutions. Then we will have 6-cycles. ■

Example: For n = 3 on a $(123) \rightarrow (132) \rightarrow (312) \rightarrow (321) \rightarrow (231) \rightarrow (213) \rightarrow (123)$.

Property 6: No 6-cycle from $\mathbf{G_n}$ has chords.

Proof. A 6-cycle corresponds to 2 non-disjoint pairs, a 4-cycle to 2 disjoint pairs. So a 6-cycle cannot contain every vertex of a 4-cycle. As a consequence, no 6-cycle on $\mathbf{G_n}$ has chords. ■

5. A cycle-built neighbouring structure

We can now define a neighbouring structure based on the 4- and 6-cycles adjacent to a vertex φ in $\mathbf{G_n}$. We call it a *rosace* of order n, $\mathbf{R_n} = \mathbf{R_n}(\varphi) = (\mathbf{S}^{34}, \mathbf{U}^{34})$. The vertex φ is the *root* of $\mathbf{R_n}(\varphi)$. The notation involves the fact that the cycles are generated by exchanges on three or four elements.

Applying Property 3 we can say that a rosace contains vertices from the levels $\mathbf{N_1}$, $\mathbf{N_2}$ and $\mathbf{N_3}$ with respect to its root φ.

The rosace $\mathbf{R_4}(\varphi)$, for $\varphi = (3\ 1\ 4\ 2)$ is (Figure 3):

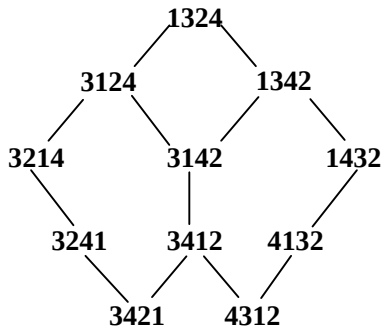

Figure 3 – The rosace $R_4(3\ 1\ 4\ 2)$

Theorem 1: The cardinality of $\mathbf{S}^{34}$ is $O(n^2)$ owing to the 4-cycle external extremities.

Proof. We have $C_{n-2,2} = (n-2)(n-3)/2$, which is $O(n^2)$. The only vertices which are exclusive of 4-cycles are their external vertices. From Properties 1 and 5, both $\mathbf{N}_1$ and 6-cycle cardinalities are O(n). ■

Remark. From Props. 1 to 5 we can obtain $\mathbf{G}_n$ order and size,

$$|\mathbf{S}^{34}| = n + (n-2)(n-3)/2 + 3(n-2) = (n-2)(n+3)/2 + n \qquad (5.1)$$

and

$$|\mathbf{U}^{34}| = n - 1 + (n-2)(n-3) + 4(n-2) = n^2 - 3. \qquad (5.2)$$

Theorem 2: A rosace $\mathbf{R_n}$ is the union of $\mathbf{N}_1$ and $\mathbf{N}_2$ vertices with the $\mathbf{N}_3$ vertices opposite to the root in the 6-cycles.

Proof. We have $\varphi \cup \Gamma(\varphi) \subset \mathbf{R_n}$. As we take $C_{n-1,2}$ pairs from $\mathbf{N}_1$ to apply the inversion operator we use it in every possible situation, then we will generate the whole $\mathbf{N}_2$ level. But in $\mathbf{N}_3$ we will attain only the n – 2 vertices opposite to φ in the 6-cycles. ■

So $\mathbf{N}_2$ will have a vertex within each 4-cycle and two vertices within each 6-cycle, then

$$|\mathbf{N}_2| = (n-2)(n-3)/2 + 2(n-2) = (n-2)(n+1)/2. \qquad (5.3)$$

For n = 6, we will have $|\mathbf{N}_2| = 14$, which is also the value of the third coefficient of this lattice's generating function (see for instance [Ka68]):

$$F(G_n) = \prod_{k=1}^{n-1} \sum_{r=0}^{k} t^r = (1+t)(1+t+t^2)\ldots(1+t+\ldots+t^k) \qquad (5.4)$$

For n = 6 we obtain (1, 5, 14, 29, 49, 71, 90, 101, 101, 90, …).

When exploring a rosace we will then visit the 6-cycle farthest vertices and also do a two-level exhaustive local search (that is, to explore $\Gamma(\varphi)$ and $\Gamma(\varphi_1)$ for every $\varphi_1 \in \Gamma(\varphi)$). Nevertheless there are two important differences:

– economy of repetitions (e.g., with n = 6 we would have (n – 1)(n – 2) = 20 vertices, while $\mathbf{R}_6$ has only 14 vertices);
– we can easily explore separately the linear substructures ($\Gamma(\varphi)$, intermediate and farthest 6-cycle points) and 4-cycle extremities and select what to explore according to the obtained results and to the behavior of the instance.

We refer to [LB01] and [Bo02] for additional information about the properties of a rosace.

6. A proposal of a rosace-based algorithm

6.1 The initial solutions

As it was already discussed we can easily obtain the feasibility matrix of any solution of the relaxed set $\mathbf{S}_N(\mathbf{Q})$. One could then think about solving the QAP by transposing to $\mathbf{S}_N(\mathbf{Q})$ the identity permutation from $\mathbf{S}_N(\mathbf{Q^*})$ – whose cost is the absolute lower bound of the instance –

and so look for the nearest feasible solution by trying to find a set of greater-valued elements in its matrix.

Unfortunately the feasibility matrix is not so a precise instrument, owing to the very high value of the ratio N!/n! even for modest-sized instances (for n = 5 we have already 10!/5! = 30240). So it is not generally possible to identify such a set in the matrix, the closest feasible solution being frequently so far that its influence on the matrix entries becomes negligible.

We used it nevertheless to induce a sort of proximity, in order to generate intermediate-valued solutions. The process is as follows:

- we take the lower-bound and upper-bound solutions in $\mathbf{S}_N(\mathbf{Q}^*)$, transpose them to $\mathbf{S}_N(\mathbf{Q})$ and calculate their feasibility matrices and the difference between them;
- we sum to each entry a pseudorandom integer value $r_{ij} \in \{0, n/k\}$, where $k \leq n$;
- we subtract every matrix entry from the maximum entry.

This way we are trying to get nearer to the lower bound and farther from the upper bound. The practical result is a set of intermediate solutions which are obtained by iteratively applying the Hungarian algorithm to the current matrix after penalizing the selected entries from the preceding iteration by summing one unity to them. Finally we order the solutions by non-decreasing cost and use the first *nsol* ones (*nsol* being the number of initial solutions obtained from each pseudorandom seed).

6.2 Generating the neighbouring structure

The initial rosace was built at the initialization stage, the identity permutation I_n being used as root. The vertices were stocked in matrices of the form $\mathbf{M}(2,p,q)$ where the first and second dimensions receive the position of an exchange and its content. The third dimension corresponds to the list of exchanges. We have p = 3 for the 6-cycles and p = 4 for the 4-cycle extremities. There are two 6-cycle matrices, one for the $\mathbf{N}_3$- elements and another for the $\mathbf{N}_2$-elements.

Example: for n = 5 the $\mathbf{N}_3$- element matrix of the 6-cycles is, with q = n – 2 = 3,

1	2	3		2	3	4		3	4	5	
3	2	1		4	3	2		5	4	3	

and the 4-cycle matrix is, with $q = C_{n-2,2} = C_{3,2} = 3$,

1	2	3	4		1	2	4	5		2	3	4	5		
2	1	4	3		2	1	5	4		3	2	5	4		

A set of routines is used that starts with $\mathbf{N}_1$ vertices and couples them to obtain the 4- and the 6-cycle vertices. The process begins at a management routine, *gerapar* and for the non-intersecting couples it is executed there, the control passing then to a stock routine *stocpar* which fills the adequate positions in the 4-cycle matrix. If the couple intersection is not void three vertices will be calculated and this is done for each one by *C6* and *gama* routines. After each pass *stocpar* is called to fill in the corresponding matrix. The scheme of the process is as follows:

```
begin
    # N1 generation (by enumeration)
    gerapar
      begin
        for i from 1 to n – 2
            for j from i + 1 to n – 1
                begin
                    generate a 4-tuple = (i, i + 1, j, j + 1)
                    if {i,i + 1} ∩ {j,j + 1} = ∅  then
                                                    #pair inversion
                                                  else ( #C6 )
                                                    gama(4-tuple(1),4-tuple(4))
                                                    stocpar
                                                    gama(4-tuple(3),4-tuple(4))
                                                    stocpar
                                                    gama(4-tuple(1),4-tuple(2))
                                                    stocpar
                end;
      end;
end.
```

Example of a 6-cycle generation

Let n = 5 and let (2 1 3 4 5) and (1 3 2 4 5) be two permutations from $\mathbf{N}_1$. The first 4-tuple will be (1,2,2,3) with non-void intersection. We apply *gama*(4-tuple(1),4-tuple(4)) to (2 1 3 4 5), looking for the values 1 and 3 at the image and exchanging them to obtain (2 3 1 4 5). Now we apply *gama*(4-tuple(3),4-tuple(4)): the elements 3 and 4 of the 4-tuple are 2 and 3, so we obtain (3 2 1 4 5). Finally with *gama*(4-tuple(1),4-tuple(2)) we get (3 1 2 4 5). We could also begin with the second permutation (1 3 2 4 5) to obtain the same result.

6.3 The use of the structure

As we have a current solution φ we calculate the products π o φ, where $\pi \in \mathbf{R}_n(I_n)$ is a permutation from the initial rosace. As the operations are limited to the exchanged elements, the process is of constant complexity order for each permutation. On the other hand we can thus preserve the adjacency relations within $\mathbf{G}_n$ despite the fact that the composition of permutations does not generally preserve these relations.

Each matrix is inspected to determine the cost differences which allow the algorithm to choose the most convenient exchange. The linear-order rosace subsets are explored in the first place, beginning by the most external ones. We usually limit the search to these sets if a better solution is found; the quadratic-order 4-cycle subset is normally explored in case of failure of the former exploration and normally just until the point where a better solution is found.

The main exploration routine is called *varre*. It receives the current solution and uses a subsidiary named *autom* to make the subset-to-subset transfer of the basic rosace to this new root.

We use two types of blocking strategies to avoid direct return of the algorithm path, one-iteration blockings and taboo blockings. These last ones were applied with the aid of taboo vectors. If the new solution has a higher cost we call for a restart routine.

6.4 A restart strategy

This strategy has already been used as an information tool for a simulated annealing application to QAP [QAB99]. We work within the ordered relaxed set $\mathbf{S_N}(\mathbf{Q}^*)$ where the cost and inversion orderings can be approximately matched.

If we have a current solution φ which is considered not adequate as a rosace root for the next iteration, we try to obtain new starting solutions φ', φ'', …, for which the corresponding $\mathbf{S_N}(\mathbf{Q}^*)$ permutations have less inversions. For that we determine an edge-permutation $\rho \in \mathbf{S_N}(\mathbf{Q}^*)$ corresponding to the current φ and look into it for a pair of positions whose exchange brings maximum reduction to ρ inversion number. After that we search for vertex permutations differing by one exchange from φ such that this exchange will imply the edge exchange we selected in ρ.

We then begin by applying (2.2) to obtain the edge-permutation $\xi \in \mathbf{S_N}(\mathbf{Q})$ associated to φ.

The composition allowing us to find $\rho \in \mathbf{S_N}(\mathbf{Q}^*)$ corresponding to ξ is

$$\rho = \phi_F^{-1} \text{ o } \xi \text{ o } \phi_D \qquad (6.1)$$

where $\phi_F : \mathbf{F} \rightarrow \mathbf{F}^+$ and $\phi_D : \mathbf{D} \rightarrow \mathbf{D}^-$ are the permutations resulting respectively from the sortings of $\mathbf{F}$ into $\mathbf{F}^+$ and of $\mathbf{D}$ into $\mathbf{D}^-$.

The first target to be found in this permutation is a position whose exchange (with another unknown position) implies the greatest inversion reduction: this is fulfilled by k_1 such that $|k_1 - \rho(k_1)|$ has maximum value. The second position for the exchange should be one bringing the minimum loss – if a loss should arrive – on the inversion reduction, that is, a k_2 such that the sum of crossed differences $|k_1 - \rho(k_2)| + |k_2 - \rho(k_1)|$ will be minimum.

As we already pointed, the resulting solution will not be feasible but we will look for feasible solutions whose exchanges, when made on φ, imply the one we selected. For that we have to return into vertex permutations.

We then apply ϕ_F^{-1} to k_1 and k_2 to find the corresponding m_1 and m_2 positions into ξ and their images $p_1 = \xi(m_1)$ and $p_2 = \xi(m_2)$. Their exchange will give us a new permutation ξ'. To return into $\mathbf{G}_n$ we have to apply the inverse ψ^{-1} of (2.2). We will obtain four different vertex permutations if the edges k_1 and k_2 are non-adjacent or three ones if they are adjacent.

Example: We will take $\mathbf{F}$ = (5, 2, 3, 1, 3, 0, 2, 0, 0, 5) and $\mathbf{D}$ = (1, 1, 2, 3, 2, 1, 2, 1, 2, 1), the instance Nug05 from the literature. A (non unique) ordering possibility for $\mathbf{F}$ and $\mathbf{D}$ is

$$\phi_F = (9\ 5\ 8\ 4\ 7\ 1\ 6\ 2\ 3\ 10) \quad \text{and} \quad \phi_D = (6\ 7\ 2\ 1\ 3\ 8\ 4\ 9\ 5\ 10).$$

Let $\varphi = (4\ 1\ 3\ 5\ 2)$ be the current solution. The edge permutation associated to φ is

$$\xi = (3\ 8\ 10\ 6\ 2\ 4\ 1\ 9\ 5\ 7).$$

The cost of the current solution is 32 and the permutation $\rho \in \mathbf{S_N}(\mathbf{Q}^*)$ is $\rho = (1\ 5\ 3\ 8\ 9\ 6\ 7\ 10\ 2\ 4)$. It has 18 inversions; taking the differences $|i - \rho(i)|$ for every i we find $k_1 = 9$ and, with the values of $| k_1 - \rho(i) | + | i - \rho(k_1) |$ for every i we find $k_2 = 4$, then the new permutation is $\rho' = (1\ 5\ 3\ 2\ 9\ 6\ 7\ 10\ 8\ 4)$.

By going into $\mathbf{S_N}(\mathbf{Q})$ we find, for the new permutation ξ',

$$\phi_F^{-1}(9) = 1 \text{ then } \xi(1) = 3 \qquad \text{and} \qquad \phi_F^{-1}(4) = 4 \text{ then } \xi(4) = 6$$

and finally

$$\psi^{-1}(3) = (1,4) \qquad \text{and} \qquad \psi^{-1}(6) = (2,4).$$

In this case the two edges are adjacent, so we will have only three possible exchanges, (1,2), (1,4) and (2,4). It is important to observe that we are working on the image, so when we take the current solution (4 1 3 5 2) we will obtain as new solutions (4 2 3 5 1), (1 4 3 5 2) and (2 1 3 5 4). The last one has a favorable difference of 2 cost units, then it will have cost 30.

In order to assure a greater distance from the last rosace we doubled this scheme to obtain two 2-exchanges. A better cost is certainly not guaranteed as we selected a single edge exchange among n – 2 ones. With two exchanges, according to edge adjacencies, we will have between 9 to 16 solutions to choose among.

This work is done by the routine *novsol*. It finds the permutation $\rho \in \mathbf{S_N}(\mathbf{Q}^*)$ and looks for the best ε exchange first positions in it, then it makes a random choose for k_1 and goes through the whole process already described. (We found it convenient to use $\varepsilon = 3 + [n/12]$ in the tests).

The cost differences for every new solution are determined and ordered, the first difference being added to the current value and the corresponding solution is sent to a local search routine, *buscaloc*.

We can use just the best solution of the ordered list or go further while the solutions have favorable differences. The best solution is always used but if it is worse than the current one we reject the remaining list regardless of the strategy used.

6.5 Value repetition

The algorithm has two security schemes to avoid its path to stake at local optima. In the first, the routine *verepet* examines the set of the last p values and looks there for q equal ones, calling for a restart if it founds them. Best results have been found with p = 5 and q = 2. The second scheme looks for a given percent of the (specified) iteration number without global improvements and goes through a double restart scheme if this percent is attained.

7. Some results

Two codes were used in the tests whose results are presented: ***Code 1***, with simple blocking and use of a single restart solution and ***Code 2***, with tabu blocking and use of several better restart solutions. The tests were run on **QAPLIB** instances [BKR97], [QAPLIB] and Drezner and new Taillard instances [DHT02]. Both codes used the option of restart after a worse solution.

The programming was done in Fortran 77. The execution times here presented correspond to the use of an onboard computer with an Athlon 2.4 chip. Typical values for both codes, compiled with optimization option, are shown at Figure 4.

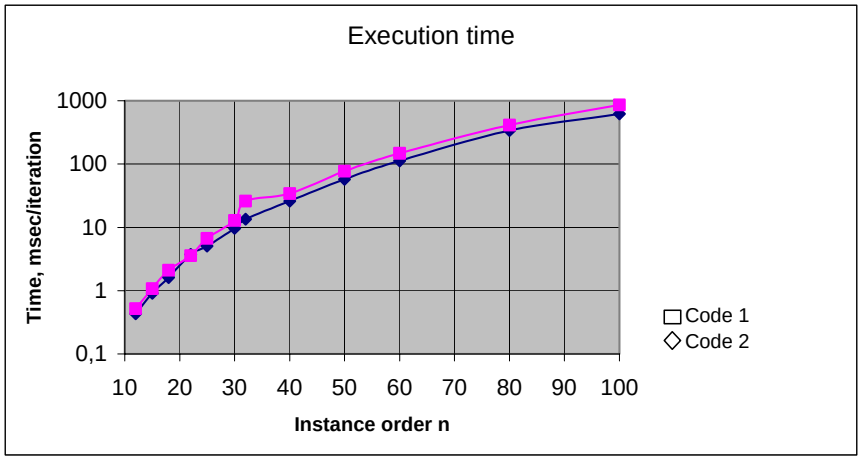

Figure 4 – Typical execution times for the two codes

The results were quite equilibrated among the two codes with respect to the set of instances utilized. We can distinguish four groups of instances according to the results:

(a) NugXX, SkoXX, WilXX, ThoXX, RouXX, Chr18b and Kra30X: optimal / better known value obtained, or approximation below 1% for greater instances. Low average with good convergence.

(b) EscXX, ScrXX, Els19, Chr12a, Chr18a: like (a) for optimal / better value, higher average.

(c) TaiXXa: better value with approximation over 1% for orders over 40. High averages.

(d) ChrXX with $n \geq 20$, DreXX ($n \leq 28$) and TaiXXe0x ($n = 27, 45, 75$), poor general average performance, unstable with respect to size for ChrXX and worsening with size for the other instances.

In the tables that follows we show the results obtained with series of **T** tests with **its/T** iterations for each one. The best obtained values (**min**) and the series averages (**avg**) are given in percentual difference over the optimal or best known cost value, so the zero entries indicate the algorithm attained this value.

Table 1 – Results for little and medium instances, Groups (a) - (b)

Inst.	T	Its/T	*Code 1*			*Code 2*		
			min	**avg**	**%**	**min**	**avg**	**%**
Nug12	50	1000	0	0	1.0	0	0	2.0
Chr12a	50	1000	0	0.23	1.0	0	0.54	2.0
Nug15	50	1000	0	0.04	2.0	0	0.10	2.0
	25	2000	0	0.01	1.0	0	<0.01	1.0
Rou15	50	1000	0	0.25	2.0	0	0.47	1.0
	25	2000	0	0.09	1.0	0	0.11	1.0
Scr15	50	1000	0	0.71	2.0	0	0.18	2.0
	25	2000	0	0.80	1.0	0	0	1.0
Chr15a	50	1000	0	3.81	2.0	0	4.27	2.0
	25	2000	0	1.79	1.0	0	2.21	1.0
Chr18a	25	2000	0	5.77	1.0	0.18	8.59	1.0
	25	4000	0	2.61	0.5	0	5.85	0.5
Chr18b	25	2000	0	0	1.0	0	0	1.0
Els19	25	2000	0	2.56	1.0	0	0.75	0.5
Nug20	25	2000	0	0.01	1.0	0	<0.01	1.0
Scr20	25	2000	0	0.87	1.0	0	0.20	1.0
Rou20	25	2000	0	0.36	1.0	0	0.32	1.0
Nug25	25	2000	0	0.10	1.0	0	0.12	1.0
Nug30	25	2000	0	0.34	1.0	0	0.36	1.0
Tho30	25	2000	0	0.30	1.0	0	0.39	1.0
Kra30a	25	2000	0	1.40	1.0	0	1.38	1.0
Kra30b	25	2000	0	0.32	1.0	0	0.36	1.0
Esc32a	25	2000	0	1.54	1.0	0	2.46	1.0
Esc32b	25	2000	0	0.48	1.0	0	0	1.0
Esc32h	25	2000	0	0.94	1.0	0	<0.01	1.0
Tho40	25	2000	0.04	0.46	1.0	0.05	0.52	0.5
	25	4000	0.01	0.33	0.5	0.01	0.26	1.0
Sko42	25	2000	0.08	0.30	1.0	0.15	0.35	1.0
	25	4000	0.03	0.20	0.5	0.04	0.24	0.5
Wil50	25	2000	0.02	0.17	1.0	0.05	0.15	1.0
	25	4000	0.02	0.10	1.0	0.07	0.13	0.5

Table 2 – Results for greater instances, Groups (a) to (c)

Inst.	T	Its/T	*Code 1*			*Code 2*		
			min	**avg**	**%**	**min**	**avg**	**%**
Sko56	5	2000	0.30	0.43	1.0	0.33	0.52	1.0
Sko64	5	2000	0.21	0.38	1.0	0.14	0.34	1.0
Sko72	5	2000	0.41	0.55	1.0	0.42	0.66	1.0
Sko81	5	2000	0.43	0.66	1.0	0.34	0.46	1.0
Sko90	5	2000	0.44	0.56	1.0	0.33	0.49	1.0
Sko100a	5	2000	0.35	0.55	1.0	0.41	0.53	1.0
Sko100b	5	2000	0.51	0.61	1.0	0.53	0.66	1.0
Sko100c	5	2000	0.22	0.52	1.0	0.25	0.53	1.0
Sko100d	5	2000	0.32	0.56	1.0	0.37	0.78	1.0
Wil100	5	2000	0.20	0.31	1.0	0.32	0.36	1.0
Esc64a	5	2000	0	0.69	1.0	0	0.34	1.0
Tai60a	5	2000	2.08	2.57	1.0	2.15	2.56	1.0
Tai80a	5	2000	2.07	2.16	1.0	1.96	2.11	1.0
Tai100a	5	2000	1.77	1.91	1.0	1.56	1.90	1.0

Table 3 – Results for Group (c) instances

Inst.	T	Its/T	*Code 1*			*Code 2*		
			min	**avg**	**%**	**min**	**avg**	**%**
Tai20a	25	2000	0	1.04	1.0	0	0.91	1.0
		4000	0	0.89	0.5	0	0.63	0.5
Tai25a	25	2000	0.73	1.76	1.0	0.71	1.86	1.0
		4000	0	1.40	0.5	0.80	1.55	0.5
Tai30a	25	2000	0.53	1.62	1.0	0.59	1.58	1.0
		4000	0.02	1.34	0.5	0.51	1.36	0.5
Tai40a	25	2000	1.58	2.17	1.0	1.30	2.13	1.0
		4000	1.58	1.98	0.5	1.38	1.96	0.5
Tai50a	25	2000	1.79	2.47	1.0	2.15	2.58	1.0
		4000	1.82	2.33	0.5	1.99	2.35	0.5

Table 4 – Results for Group (d) instances

Inst.	T	Its/T	*Code 1*			*Code 2*		
			min	**avg**	**%**	**min**	**avg**	**%**
Chr20a	25	2000	4.38	11.49	1.0	2.37	9.14	1.0
		4000	2.74	9.07	0.5	1.37	7.99	0.5
Chr20b	25	2000	4.87	12.51	1.0	4.79	11.85	1.0
		4000	2.79	12.46	0.5	3.31	11.43	0.5
Chr22a	25	2000	2.40	4.20	1.0	3.02	4.71	1.0
		4000	0.88	4.29	0.5	1.17	4.45	0. 5
Chr22b	25	2000	2.10	5.00	1.0	2.24	5.00	1.0
		4000	2.68	4.46	0.5	1.61	4.29	0.5
Chr25a	25	2000	8.38	17.57	2.0	7.22	17.91	1.0
		4000	2.32	14.70	1.0	9.33	16.20	0.5
Dre15	50	2000	0	6.08	2.0	0	4.44	2.0
		4000	0	1.11	2.0	0	1.11	2.0
		4000	0	3.46	0.5	0	3.85	0.5
		4000	0	1.57	1.0	0	3.01	1.0
Dre18	50	2000	0	4.04	2.0	0	6.26	1.0
		4000	0	1.63	1.0	0	2.71	0.5
Dre21	50	2000	0	34.61	1.0	0	33.03	1.0
		8000	10.11	33.48	1.0	0	32.47	0.5
Dre24	50	4000	8.59	37.98	2.0	9.60	39.34	2.0
		8000	12.63	35.56	0.25	0	33.38	0.25
Dre28	50	8000	28.15	42.35	1.0	0	39.37	1.0
Tai27e01	25	2000	0	154.20	1.0	0	151.73	1.0
		2000	0	152.35	0.5	0.78	153.67	0.5
		4000	0	137,14	0.5	0	150.66	0.5
Tai27e02	25	2000	1.26	240.05	1.0	0	256.52	1.0
		2000	0	237.42	0.5	0	252.28	0.5
		4000	0	236.18	0.5	0	131.65	0.5
Tai45e01	25	2000	0	167.38	1.0	1.65	135.21	1.0
		4000	0	164.82	0.5	0	131.65	0.5
Tai75e01	5	2000	16.29	130.53	0.5	10.73	69.20	1.0
	10	8000	9.94	122.50	0.25	8.72	94.59	1.0

8. Conclusions

8.1 Discussion

For many instances, specially those from Group (a), the instance order does not sensibly influence the performance, which is an interesting characteristic of the method. This can be seen quite well with the instances SkoXX.

Group (b) goes well from the same point of view for the best value found but the average suffers the influence of high-valued differences when compared to best known value, as it can be seen with EscXX instances.

Group (c) seems to show failed approaches, probably owing to the presence of many bad solutions around good ones, local optima at which the algorithm stakes. This should ask for local approach improvement.

Group (d) seems to show an influence of the arborescence structure in the polynomial instances ChrXX, which offers difficulties for heuristic algorithms. Owing to the sparsity of one of their matrices the DreXX instances are also difficult for heuristic methods, despite the fact that they are also polynomial, as tr(**Q***) is an optimal value for them. Finally the staged TaiXXe0x instances present difficulties associated to the starting points utilized and also to the easy stacking at local optima. The average values are high owing to the values of these local optima. With some of these instances the second code was able to find an optimum value, while thw first was not.

Some time economy can be obtained through the use of more than one solution from *novsol* list, but it depends on the instance structure, as for some of them it is frequently difficult to obtain more than one positive difference. The difference in processing time between the two codes corresponds to this economy. The taboo blocking seems to be at least as efficient as the single blocking to avoid path return.

8.2 Further developments

Different restart approaches could be designed: for example, to use the rosace building functions to expand it in order to get a solution set for choosing a restart point. This could eventually give better chances of finding good restarts, specially if we could detect directions of inversion reduction which could be used to build a variable neighborhood.

We could also imagine the use of a path-relinking strategy using a set of good solutions obtained from a test or from three tests run in parallel. This last strategy would present the advantage of producing again three solutions which could be used to continue the process the same way. The speed could be improved by using parallelism on rosace exploration.

Acknowledgements

This work belongs to a research project developed within the Group of Graphs, Combinatorics and Operations Research Applications from COPPE/Federal University of Rio de Janeiro (UFRJ). We are indebted to COPPE/UFRJ and to the National Council for Scientific and Technological Development (CNPq), whose sponsorship was essential for the consecution of the stage at the PRISM Laboratory, University of Versailles at St. Quentin-en-Yvelines,

where we developed the basic work from where this paper was written. We are also indebted to the colleagues of PRISM who gave support to our work, specially Catherine Roucairol, Van-Dat Cung, Adriana Alvim and Thierry Mautor. Finally, we are grateful for the very useful criticism and suggestions from the anonymous referees.

References

[Ab84] Abreu, N.M.M. (1984). An algebraic and combinatorial study of the quadratic assignment problem on the sense of Koopmans and Beckmann (in Portuguese). D.Sc. Thesis, COPPE/UFRJ.

[AB63] Armour, G.C. & Buffa, E.S. (1963). Heuristic algorithm and simulation approach to relative location of facilities. *Man. Sci.*, **9**, 294-309.

[ABQG02] Abreu, N.M.M.; Boaventura Netto, P.O.; Querido, T.M. & Gouvêa, E.F. (2002). Classes of quadratic assignment instances: isomorphism and difficulty measures using a statistical approach. *Discr. Appl. Maths.*, **124**, 103-116.

[AHS01] Arkin, E.M.; Hassin, R. & Sviridenko, M. (2001). Approximating the maximum quadratic assignment problem. *Inf. Proc. Lett.*, **77**(1), 13-16.

[An96] Anderson, E.J. (1996). Mechanisms for local search. *EJOR*, **88**, 139-151.

[AQB99] Abreu, N.M.M.; Querido, T.M. & Boaventura-Netto, P.O. (1999). RedInv-SA: A simulated annealing for the quadratic assignment problem. *RAIRO Opns. Res.*, **33**, 249-273.

[AOT00] Ahuja, R.; Orlin, J.B. & Tivari, A. (2000). A greedy genetic algorithm for the quadratic assignment problem. *Comp. Opns. Res.*, **27**, 917-937.

[BAV62] Buffa, E.S.; Armour, G.C. & Vollmann, T.E. (1962). Allocating facilities with CRAFT. *Harvard Business Review*, **42**, 136-158.

[BB83] Burkard, R.E & Bonniger, T. (1983). A heuristic for quadratic boolean programs with applications to quadratic assignment problems. *EJOR*, **13**, 374-386.

[BÇ95] Burkard, R.E. & Çela, E. (1995). Heuristics for biquadratic assignment problems and their computational comparison. *EJOR*, **83**(2), 283-300.

[Br84] Bruijs, P.A. (1984). On the quality of heuristic solutions to a 19 x 19 quadratic assignment problem. *EJOR*, **17**, 21-30.

[Bu91] Burkard, R.E. (1991). *Locations with spatial interactions: the quadratic assignment problem*, Discrete Location Theory. John Wiley & Sons, New York, 387-437.

[BR94] Battiti, R. & Tecchiolli, G. (1994). The reactive tabu search. *ORSA J. Comp.*, 126-140.

[Be68] Berge, C. (1968). *Principes de combinatoire*. Dunod, Paris.

[Be73] Berge, C. (1973). *Graphes et hypergraphes*. 2ème édition. Dunod, Paris.

[BKR97] Burkard, R.E.; Karisch, S.E & Rendl, F. (1997). QAPLIB – A quadratic assignment problem library. *J. Global Opt.*, **10**, 391-403.

[Bo01] Boaventura-Netto, P.O. (2001). *Graphs: theory, models, algorithms* (in Portuguese). Ed. Edgard Blucher, São Paulo.

[CMMT97] Cung, V-D; Mautor, T.; Michelon, P. & Tavares, A. (1997). A scatter search based approach for the quadratic assignment problem. *Proc. IEEE Int. Conf. on Evolutionary Computation*, 165-169.

[Çe98] Çela, E. (1998). *The quadratic assignment problem*. Kluwer, Dordrecht.

[DHT02] Drezner, Z.; Hahn, P. & Taillard, E. (2002). A study of quadratic assignment problem instances that are difficult for meta-heuristic methods. Invited for submission to *Ann. Opns. Res.: State of the Art and Recent Advances in Integer Programming.* Available at Internet: <http://www.seas.upenn.edu/~hahn/new_instances.html>.

[GP66] Gavett, J.W. & Plyter, N.V. (1966). The optimal assignment of facilities to locations by branch-and-bound. *Opns. Res.*, **14**, 210-232.

[GY02] Gutin, G. & Yeo, A. (2002). Polynomial approximation algorithms for TSP and QAP with a factorial domination number. *Discr. Appl. Maths.*, **119**(1-2), 107-116.

[Ka68] Kaufmann, A. (1968). *Introduction à la combinatorique et ses applications.* Dunod, Paris.

[KÇCE00] Karisch, S.E.; Çela, E.; Clausen, J. & Espersen, T. (2000). A dual framework for lower bounds of the quadratic assignment problem based on linearization. SFB Rep. 120, Technical University of Graz / Tech. Rep. IMM-REP-1998-02, Department of Mathematical Modeling, Technical University of Denmark.

[LB01] Loiola, E.M. & Boaventura-Netto, P.O. (2001). A neighbouring structure for the QAP (in Portuguese). *An. XXXIII SBPO*, 1288-1297, Campos de Jordão, Brazil, novembre.

[Lo00] Loiola, E.M. (2000). An algorithm with statistical parameters for the QAP (in Portuguese). M.Sc. Thesis, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

[LS95] Li, W.-J. & Smith, M. (1995). An Algorithm for Quadratic Assignment Problems. *EJOR*, **81**, 205-216.

[Ma00] Mautor, T. (2000). Application des métaheuristiques au PAQ. Rapport #2000/15, Laboratoire PRISM, Université de Versailles à St. Quentin-en-Yvelines.

[MO79] Mirchandani, P.B. & Obata, T. (1979). Algorithms for a class of quadratic assignment problems. Presented at the *Joint ORSA/TIMS National meeting*, New Orleans.

[QAB99] Querido, T.M.; Abreu, N.M.M. & Boaventura Netto, P.O. (1999). RedInv-SA: a simulated annealing for the quadratic assignment problem. *RAIRO Opns. Res.*, **33**, 249-273.

[QAPLIB] QAPLIB Home Page: <http://www.opt.math.tu-graz.ac.at/qaplib/>.

[RAB00] Rangel, M.C.; Abreu, N.M.M. & Boaventura-Netto, P.O. (2000). GRASP in the QAP: an acceptance bound for initial solutions (in Portuguese). *Pesq. Operacional*, **20**, 45-58.

[SWH98] Sarker, B.R.; Wilhelm, W.E. & Hogg, G.L. (1998). One-dimensional machine location problems in a multi-product flowline with equidistant locations. *EJOR*, **105**(3), 401-426.

[SWHH95] Sarker, B.R.; Wilhelm, W.E.; Hogg, G.L. & Han, M.-H. (1995). Backtracking of jobs in one-dimensional machine location problems. *EJOR*, **85**(3), 593-609.

[TB98] Tansel, B.C. & Bilen, C. (1998). Move based heuristics for the unidirectional loop network layout problem. *EJOR*, **108**(1), 36-48.

[THKG98] Talbi, E.-G.; Hafidi, Z.; Kebbal, D. & Geib, J.-M. (1998). A fault-tolerant parallel heuristic for assignment problems. *Fut. Gen. Comp. Sys.*, **14**(5-6), 425-438.

[TRFR01] Talbi, E.G.; Roux, O.; Fonlupt, C. & Robillard, D. (2001). Parallel ant colonies for the quadratic assignment problem. *Fut. Gen. Comp. Sys.*, **17**, 441-449.

[TS95] Tate, D.E. & Smith, A.E. (1995). A genetic approach for the quadratic assignment problem. *Comp. Opns. Res.*, **22**, 73-83.

[We83] West, D.H. (1983). Algorithm 608: Approximate solution of the quadratic assignment problem. *ACM Trans. Math. Softw.*, **9**, 461-466.

[YSA03] Youssef, H.; Sait, S.M. & Ali, H. (2003). Fuzzy simulated evolution algorithm for VLSI cell placement. *Comp. & Ind. Eng.*, **44**(2), 227-247.