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

UM ALGORITMO CONSTRUTIVO BASEADO EM UMA ABORDAGEM ALGÉBRICA DO PROBLEMA QUADRÁTICO DE ALOCAÇÃO

Leandro Colombi Resendo *
Departamento de Engenharia Elétrica / CT
Universidade Federal do Espírito Santo (UFES)
Vitória – ES
lresendo@yahoo.com.br

Maria Cristina Rangel
Departamento de Informática / CT
Universidade Federal do Espírito Santo (UFES)
Vitória – ES
crangel@inf.ufes.br

* *Corresponding author* / autor para quem as correspondências devem ser encaminhadas

Recebido em 03/2005; aceito em 08/2005 após 1 revisão
Received March 2005; accepted August 2005 after one revision

Resumo

O *Problema Quadrático de Alocação*, PQA, foi estudado utilizando uma abordagem algébrica através de uma relaxação linear, o *Problema de Alocação Linear*, PAL. A utilização dessa abordagem se deve ao fato de existir na literatura o Teorema das Inversões demonstrado por Rangel em [Ran00] que associa o custo de uma solução do PQA ao número de inversões de sua correspondente linear no PAL. Embora seja polinomial o reconhecimento de uma solução linear viável para o PQA, caracterizar no conjunto de todas as soluções do PAL quais são as que satisfazem o PQA é uma tarefa extremamente difícil. Neste trabalho construímos uma matriz que armazena informações de soluções lineares capazes de gerar soluções quadráticas. Combinando esse mapeamento com o Teorema das Inversões apresentamos um método construtivo que gera soluções iniciais de boa qualidade. A grande vantagem dessa matriz é que seu custo computacional e gasto de memória são baixos. Propomos também uma versão paralela deste algoritmo.

Palavras-chave: problema quadrático de alocação; problema de alocação linear; teorema das inversões.

Abstract

The Quadratic Assignment Problem, QAP, was studied using an algebraic approach through a linear relaxation, the Linear Assignment Problem, LAP. The reason for this approach is the Inversion Theorem demonstrated by Rangel [Ran00]. In this theorem, the QAP solution cost is associated to the number of inversions of the linear correspondent. Although recognizing if a linear solution correspond to a QAP solution is polynomial, there are much more LAP solutions than QAP solutions, and therefore to find them is a hard work. We construct a matrix that stores information about LAP solutions that are able to generate QAP solutions. The Inversion Theorem in conjuction with this matrix permitted us to present a constructive method that generates good initial solutions. The great advantage of this matrix is the low computational cost of time and memory. A parallel version of this algorithm is proposed and implemented in this work.

Keywords: quadratic assignment problem; linear assignment problem; inversion theorem.

1. Introdução

Inicialmente formulado em um contexto econômico por Koopmans & Beckmann [KB57], o Problema Quadrático de Alocação (PQA) foi apresentado como a alocação de n facilidades indissociáveis a n localidades com o objetivo de minimizar custos dessa alocação. O PQA é um dos problemas mais difíceis da classe dos problemas NP-hard, devido ao seu alto grau de combinatoriedade, a ponto de problemas de dimensão $n \geq 30$ já serem considerados de grande porte. Os estudos sobre métodos de resolução do PQA basicamente se dividem em dois grupos: algoritmos exatos ou ótimos, e algoritmos heurísticos ou sub-ótimos. Em ambas as estratégias de estudo citadas, está embutido o estudo de limites inferiores. Dentre os limites mais famosos podemos destacar o de Gilmore [Gi62], Lawler [La63] e, mais recentemente, o de Anstreicher & Brixius, que em 2001, [AB01] desenvolveram um limite inferior utilizado nos algoritmos exatos que trabalham com enumeração implícita [ABLG02]. Uma grande dificuldade encontrada pelos métodos exatos é a necessidade de plataformas poderosas e uma grande quantidade de memória para armazenamento. Por exemplo, Anstreicher & Brixius [ABGL02] apresentaram a solução ótima para o problema teste Kra30b com um tempo computacional de 182 dias, usando uma única estação de trabalho.

Justificadas pelas desvantagens apresentadas dos algoritmos exatos, uma grande quantidade de pesquisas voltadas aos algoritmos heurísticos está na literatura, podendo citar como exemplo: o GRASP, proposto por Li, Paradalos & Resende [LPR94], que usa a idéia de algoritmos construtivos associada à técnicas de melhoramento e Burkard & Stratman [BS78] que propuseram uma heurística usando o método de *branch-and-bound.* Não podemos deixar de citar as meta-heurísticas aplicadas ao PQA que utilizam como estratégia de melhoramento os fenômenos naturais, entre elas temos o *Simulated Annealing* desenvolvido por Burkard & Rendl [BR83], o Algoritmo Genético que teve sua aplicação ao PQA por vários pesquisadores, entre eles Tate & Smith [TS94], e o Sistema da Colônia de Formigas, estudo por Maniezzo *et al.* [MCD94] e Gambardella *et al.* [GTD97].

Um livro dedicado ao PQA é DIMACS [DIMACS94], resultado de uma coletânea de trabalhos apresentados em um *workshop* dedicado ao PQA, organizados por Panos M. Pardalos e Henry Wolkowicz. O de Eranda Çela [Ce98], fruto de sua tese de doutorado, apresenta um *survey* do problema detalhando algoritmos exatos e heurísticos. Além disso, é mostrado um estudo sobre algumas estruturas especiais de matrizes e o Problema Biquadrático de Alocação.

Um dos precursores do estudo do PQA através de uma abordagem algébrica foi Abreu [Ab84]. Seguindo na mesma linha, Querido [Que94] apresentou um *Simulated Annealing* cuja vizinhança proposta é baseada na teoria algébrica desenvolvida por Abreu; Rangel [Ra00] provou o Teorema das Inversões através de uma relaxação linear, o Problema de Alocação Linear que associa diretamente o custo de uma solução quadrática ao número de inversões de sua solução linear correspondente, considerando uma classe especial de soluções. Recentemente Resendo & Rangel [RR04] propõe uma construção de soluções quadráticas de boa qualidade baseada no Teorema das Inversões. Entre os trabalhos mais recentes encontrados na literatura, podemos citar: Abreu *et al*. [ABQG02], Hasegawa *et al*. [HIAI02] e Boaventura [Boa03].

O objetivo deste trabalho é apresentar um algoritmo construtivo de melhoramento utilizando soluções iniciais de boa qualidade. Para isso, o trabalho está organizado da seguinte forma: na seção 2 é apresentado o Problema Quadrático de Alocação e uma relaxação linear na forma do Problema de Alocação Linear, o estudo da viabilidade das soluções lineares com respeito

ao PQA e ainda, é enunciado o Teorema das Inversões [Ran00]. Na seção 3 é proposta uma maneira de gerar soluções quadráticas de boa qualidade, usando a relaxação linear e o Teorema das Inversões. As seções 4 e 5 constam das descrições do algoritmo construtivo desenvolvido e a apresentação dos resultados, respectivamente. Ao final, seguem as conclusões.

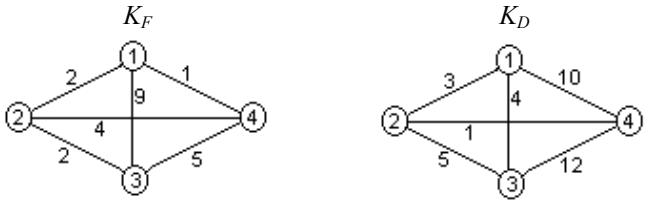
2. Uma Relaxação Linear do Problema Quadrático de Alocação

Considerando que as soluções de um exemplar do Problema Quadrático de Alocação assumem a forma de permutações [KB57], descreve-se matematicamente a busca pela melhor solução da seguinte forma:

$$Z(\varphi^*) = \min_{\varphi \in \Pi_n} \sum_{i,j=1}^{n} f_{ij} d_{\varphi(i)\varphi(j)} , \qquad (1)$$

tal que $1 \le i < j \le n$, Π_n é conjunto de todas as permutações φ de $\Omega_n = \{1, \ldots, n\}$, e F = $[f_{ij}]$ e D = $[d_{kl}]$ são matrizes de fluxo e distância respectivamente.

Exemplo 1: *Considere duas cliques K_F e K_D, representadas por duas matrizes,*



$$F = \begin{bmatrix} 0 & 2 & 9 & 1 \\ 2 & 0 & 2 & 4 \\ 9 & 2 & 0 & 5 \\ 1 & 4 & 5 & 0 \end{bmatrix} \quad \text{e} \quad D = \begin{bmatrix} 0 & 3 & 4 & 10 \\ 3 & 0 & 5 & 1 \\ 4 & 5 & 0 & 12 \\ 10 & 1 & 12 & 0 \end{bmatrix}$$

Figura 1 – Cliques valoradas K_F e K_D e suas respectivas matrizes.

onde uma solução φ é dada por uma sobreposição de cliques representadas por uma permutação. O custo desta alocação é 130 dado pela expressão (1).

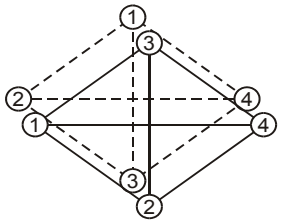


Figura 2 – Sobreposição das cliques, $\varphi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 2 & 4 \end{pmatrix}$.

Utilizando as características das matrizes, tais como simetria e diagonal principal nula, pode-se armazenar suas informações na forma de vetores com dimensão $N = C_{n,2}$, pela ordem lexicográfica dos índices das matrizes. Para isso fazemos uso da bijeção $\psi(i,j) = ((i-1)n - i(i+2)/2) + j$ que associa a cada par $(i,j) \in nxn$ com $i<j$ um natural $z \in \{1, ..., N\}$ que representa uma aresta das cliques. Para as matrizes F e D mencionadas são escritos os vetores **F** = *(2 9 1 2 4 5)* e **D** = *(3 4 10 5 1 12)*.

Considerando apenas as sobreposições de arestas, estabelecemos uma relaxação linear através do Problema de Alocação Linear, PAL, que pode ser definido como: $Z(\xi^*) = \min_{\xi \in \Pi_N} \sum_{i=1}^{N} f_i d_{\xi(i)}$,

onde Π_N é conjunto de todas as permutações ξ de $\Omega_N = \{1, ..., N\}$. O PAL é considerado uma relaxação de um PQA no sentido de o conjunto de soluções lineares conter as soluções quadráticas. No entanto, o número de soluções do PAL é bem maior que o número de soluções quadráticas, conseqüentemente, nem sempre uma permutação de arestas pode representar uma permutação de vértices. Tais soluções são **não-viáveis** para o exemplar do PQA correspondente.

Considere as matrizes $Q = \mathbf{F}^t\mathbf{D}$ e $Q^* = (\mathbf{F}^-)^t\mathbf{D}^+$, onde $\mathbf{F}^-$ e $\mathbf{D}^+$ são vetores com a ordenação não-crescente e não-decrescente de **F** e **D** respectivamente.

Tabela 1 – Matrizes Q e Q^* para o exemplo proposto.

Q	3	4	10	5	1	12
2	6	8	**20**	10	2	24
9	27	36	90	45	**9**	108
1	3	4	10	5	1	**12**
2	6	8	20	**10**	2	24
4	12	**16**	40	20	4	48
5	**15**	20	50	25	5	60

Q^*	12	10	5	4	3	1
1	**12**	24	24	48	60	108
2	10	**20**	20	40	50	90
2	5	10	**10**	20	25	45
4	4	8	8	**16**	20	36
5	3	6	6	12	**15**	27
9	1	2	2	4	5	**9**

Encontrar a solução ξ ótima do PAL é uma tarefa bem simples sendo que tal solução coincide com o traço da matriz Q^*, além disso este é um limite inferior para o PQA definido por **F** e **D**. Analogamente, obtemos um limite superior através da somatória dos elementos da diagonal secundária da matriz Q^*. É fácil observar que os problemas PAL(Q) e PAL(Q*) são isomorfos pois para toda $\xi \in \Pi_N$ (permutação das colunas da matriz Q) existirá uma $\rho \in \Pi_N$ (permutação das colunas da matriz Q^*) correspondente.

Porém, dada uma permutação $\rho \in \Pi_N$, que troca colunas em Q^*, como saber se existe uma permutação de vértices da clique K_F sobre os da clique K_D a ela correspondente?

Para responder a questão colocada é estabelecida uma relação entre ξ e ρ. Para isso basta armazenar em permutações auxiliares ϕ_F e ϕ_D as trocas feitas nas posições dos vetores durante a ordenação dos vetores **F** e **D**. Para o exemplo apresentado na Figura 1 temos

$$\phi_F = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 6 & 1 & 3 & 4 & 5 \end{pmatrix} \text{ e } \phi_D = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 4 & 2 & 3 & 6 & 1 \end{pmatrix}.$$

Tendo as permutações auxiliares ϕ_F e ϕ_D obtemos a relação entre ξ e ρ através da bijeção

$$\xi = \phi_D^{-1} \circ \rho \circ \phi_F .$$

Admitindo uma ρ como sendo o traço da matriz, que é a permutação identidade $\rho = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}$, fazemos uso da bijeção para obtermos a ξ correspondente, ou seja,

$\xi = \phi_D^{-1} \circ \rho \circ \phi_F$, o que implica em,

$$\xi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 3 & 4 & 2 & 1 & 5 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 6 & 1 & 3 & 4 & 5 \end{pmatrix};$$

$$\xi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 5 & 6 & 4 & 2 & 1 \end{pmatrix}.$$

Uma vez feito isso definimos uma solução linear como **viável** ao PQA, se dada uma ρ podemos construir φ através do caminho inverso ao da relaxação, representado pela Figura 3.

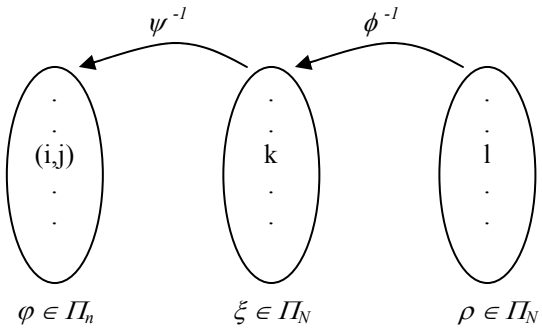


Figura 3 – Visualização da viabilidade de solução.

Para $\rho \in \Pi_N$ ser viável é necessário que exista uma $\varphi \in \Pi_n$ que satisfaça a seguinte seqüência de igualdades:

$$f_r^- d_{\rho(r)}^+ = f_{\phi_F^{-1}(r)}^- d_{\phi_D^{-1}(\rho(r))}^+ = f_p d_{\xi(p)} = f_{\psi^{-1}(p)} d_{\psi^{-1}(\xi(p))} = f_{ij} d_{kl},$$

tal que $r = 1, \ldots, N$; $\varphi(i) = k$ e $\varphi(j) = l$; onde $i,j,k,l = 1, \ldots, n$. Caso contrário, a solução é **não-viável**.

A seguir apresentamos o algoritmo SolViável proposto por Rangel [Ran00] que verifica a viabilidade de ρ com respeito ao PQA.

Algoritmo 1: *SolViável*

1: **Entrada**: ρ, ϕ_F e ϕ^{-1}_D
2: $\xi = \phi^{-1}_D \circ \rho \circ \phi_F$
3: **for** t = 1, ... , N **do**
4: $\quad$ *ListaIJ[t]* $\leftarrow \psi^{-1}(t)$
5: $\quad$ *ListaKL[t]* $\leftarrow \psi^{-1}(\xi(t))$
6: **end for**

7: **if** $\exists\ p \in 1, \ldots, n$ tal que $\varphi(1) = p$ para as $(n-1)$ primeiras combinações **then**
8: $\quad\varphi(1) = p$
9: $\quad$**for** $i = 2, \ldots, n$ **do**
10: $\quad\quad[\varphi(i) = l \neq p] \vee [\varphi(i) = k \neq p]$ para $ListaKL[i-1]$
11: $\quad$**end for**
12: $\quad$**if** φ construída possui todas as N combinações compatíveis **then**
13: $\quad\quad\rho$ é viável
14: $\quad$**else**
15: $\quad\quad\rho$ não é viável
16: $\quad$**end if**
17: **else**
18: $\quad\rho$ não é viável
19: **end if**

O término do algoritmo se dá de forma trivial já que todos os *loop´s* são constantes de ordem no máximo *N*.

Primeiramente o algoritmo gera a permutação ξ através da ρ usando a bijeção da linha 2, e constrói as ListasIJ e ListaKL que são o domínio e imagem da permutação ξ, respectivamente. A linha 7 representa a função que percorrerá os $(n-1)$ primeiros elementos das ListaIJ e ListaKL testando as combinações possíveis para montar uma $\varphi(1)$, caso essa função obtenha uma resposta negativa o algoritmo termina retornando que a solução ρ é não-viável. No caso de uma resposta afirmativa, o *loop* da linha 9 tem como objetivo construir o restante da permutação φ a partir da ListaKL. Após construir φ , a linha 12 representa a função que verifica as *N* combinações compatíveis. No caso de encontrar alguma incompatibilidade, o algoritmo termina retornando que a permutação ρ é não-viável, caso contrário, ρ é viável.

Para simplificar a notação, no restante deste trabalho, representaremos as permutações apenas por sua imagem (como na ρ apresentada no Exemplo 2), omitindo assim seu domínio.

Exemplo 2: *Admita a permutação* $\rho = (6\ 5\ 1\ 4\ 2\ 3) \in \Pi_6$

O primeiro passo do algoritmo será encontrar a permutação ξ (que troca colunas de **Q**) correspondente a ρ (que troca colunas em Q*), para isso faremos uso de ϕ_F e ϕ_D pela bijeção já mencionada:

$\xi = \phi^{-1}_D \circ \rho \circ \phi_F$
$\xi = (6\ 3\ 4\ 2\ 1\ 5) \circ (6\ 5\ 1\ 4\ 2\ 3) \circ (2\ 6\ 1\ 3\ 4\ 5)$
$\xi = (1\ 4\ 5\ 6\ 2\ 3)$

Na Tabela 2 apresentamos a aplicação da ψ^{-1} para construção das listas ListaIJ e ListaKL, bem como as possíveis combinações de vértices.

Note que nas $(n-1)$ linhas construímos $\varphi = (2\ 1\ 3\ 4)$, porém no restante das linhas da tabela são apresentadas algumas incompatibilidades, logo ρ é dita **não-viável**.

Dado *n*, considere $N = C_{n,2}$ e $\xi \in \Pi_N$ uma permutação. Defina os (n – 1) primeiros elementos da ξ de **cabeça** e os N – (n – 1) elementos restantes de **cauda** da permutação de ξ.

Tabela 2 – Tabela auxiliar para o exemplo 2.

ListaIJ	ListaKL	Combinações de vértices
$\psi^{-1}(1) = (1,2)$	$\psi^{-1}(1) = (1,2)$	$[\varphi(1) = 1 \wedge \varphi(2) = 2] \vee [\varphi(1) = 2 \wedge \varphi(2) = 1]$
$\psi^{-1}(2) = (1,3)$	$\psi^{-1}(4) = (2,3)$	$[\varphi(1) = 2 \wedge \varphi(3) = 3] \vee [\varphi(1) = 3 \wedge \varphi(3) = 2]$
$\psi^{-1}(3) = (1,4)$	$\psi^{-1}(5) = (2,4)$	$[\varphi(1) = 2 \wedge \varphi(4) = 4] \vee [\varphi(1) = 4 \wedge \varphi(4) = 2]$
$\psi^{-1}(4) = (2,3)$	$\psi^{-1}(6) = (3,4)$	$[\varphi(2) = 3 \wedge \varphi(3) = 4] \vee [\varphi(2) = 4 \wedge \varphi(3) = 3]$
$\psi^{-1}(5) = (2,4)$	$\psi^{-1}(2) = (1,3)$	$[\varphi(2) = 1 \wedge \varphi(4) = 3] \vee [\varphi(2) = 3 \wedge \varphi(4) = 1]$
$\psi^{-1}(6) = (3,4)$	$\psi^{-1}(3) = (1,4)$	$[\varphi(3) = 1 \wedge \varphi(4) = 4] \vee [\varphi(3) = 4 \wedge \varphi(4) = 1]$

Analisando as construções do algoritmo, observam-se dois tipos de permutações não-viáveis. Em um dos casos o algoritmo não consegue montar a permutação quadrática φ (linha 7) e no outro, o algoritmo consegue montar a φ, ilustrado no exemplo 2, com os elementos da **cabeça** da permutação. Porém é encontrada incompatibilidade com os elementos da **cauda** (linha 12). As permutações que obedecem ao segundo caso descrito serão definidas como **pseudo-viáveis**, assim, o conjunto das soluções viáveis de um exemplar está contido no conjunto das pseudo-viáveis, considerando a capacidade de construção de uma permutação de vértices. O objetivo da definição das soluções pseudo-viáveis é aumentar o espaço de busca de soluções lineares que geram soluções quadráticas.

Propriedade 1: *O número de elementos do conjunto das soluções pseudo-viáveis é dado por* $n![N - (n - 1)]!$.

Prova: O número de soluções viáveis é $n!$ que é dado pelo número de soluções do PQA. Seja $\rho \in \Pi_N$ uma solução viável para o PQA sendo assim, a cabeça da ξ correspondente é capaz de construir uma solução quadrática $\varphi \in \Pi_n$. Fixando a cabeça da ξ, isto é, as $(n - 1)$ primeiras componentes, podemos permutar as $N - (n - 1)$ componentes da cauda gerando desta forma $[N - (n - 1)]!$ pseudo-viáveis. Pelo Princípio Fundamental da Contagem temos que o número de pseudo-viáveis é $n![N - (n - 1)]!$. □

Veja a relação de proporção:

- n=4 => N=6
- N!=720 soluções lineares com n!=24 soluções viáveis.
- N!=720 soluções lineares com n!(n-1)! =24x 6 = 144 soluções pseudo-viávies.

2.1 Teorema das Inversões e Permutações Livremente Comparáveis

Encontramos na literatura o **Teorema das Inversões** demonstrado por Rangel em [Ran00], [RA01] e [RA03], que associa o custo $Z(\rho)$ de uma solução ρ do PAL(Q*) ao o seu número de inversões. Para um melhor entendimento do teorema, seguem algumas definições que lhes dão suporte. Essas definições também são encontradas nas referências citadas acima:

- **inversão:** é dada pelo par $(\rho(i), \rho(j))$ tal que $\rho(j) < \rho(i)$ com $i < j$ e $i,j \in \Omega^N$. Considere $\Im(\rho)$ o conjunto com todos os pares de $(\rho(i), \rho(j))$ que possuem uma inversão.
- **número de inversões de ρ:** é dada pela cardinalidade do conjunto $\Im(\rho)$. Exemplo: Assuma a permutação $\rho = (4\ 1\ 2\ 3\ 6\ 5)$, temos que $\Im(\rho) = \{(4,1), (4,2), (4,3), (6,5)\}$ e $\#\Im(\rho) = 4$.

- **permutações livremente comparáveis**: duas permutações ρ_1 e ρ_2 são ditas **livremente comparáveis** quando podemos afirmar que $Z(\rho_1) \leq Z(\rho_2)$ ou $Z(\rho_2) \leq Z(\rho_1)$ independentemente dos valores dos vetores $\mathbf{F}^-$ e $\mathbf{D}^+$**.**

O **Teorema das Inversões** pode ser compreendido da seguinte forma: *"Os custos das permutações $\rho \in \Pi_N$ que são **livremente comparáveis** crescem junto com os seus números de **inversões**".*

Como existe uma correspondência entre as soluções quadráticas $\varphi \in \Pi_n$ e as soluções lineares $\rho \in \Pi_N$ podemos definir a partir do Teorema das Inversões um critério de pré-avaliação das soluções do PQA. Dada uma solução φ do PQA, se a solução linear $\rho \in \text{PAL}(Q^*)$ correspondente a ela possuir um baixo número de inversões, conseqüentemente seu custo também o é. Vale lembrar que o teorema é válido para o conjunto de permutações livremente comparáveis que está contido no conjunto de todas as permutações $\rho \in \Pi_N$. Sabemos que uma φ, permutação de vértices, pode possuir uma representante ρ que não pertence ao conjunto das permutações Livremente Comparáveis. Contudo, um estudo através de regressão linear feito por Rangel em [Ran00] mostra que essa relação se estende para o conjunto de todas as permutações $\rho \in \Pi_N$. Na próxima seção apresentaremos uma proposta para gerar soluções quadráticas de boa qualidade [Res04] que será parte integrante do algoritmo construtivo apresentado neste trabalho.

3. Mapeamento das Soluções Quadráticas no Universo das Soluções Lineares

Apesar da grande importância do Algoritmo 1, ele se tornará ineficiente se tivermos que enumerar todas as soluções lineares para reconhecer as viáveis. Como visto na seção 2, não são necessários os *N* elementos da permutação linear para que possamos gerar a permutação dos vértices correspondentes, caso exista. Para isso, bastam os *(n – 1)* elementos da cabeça. Recordando o Algoritmo 1, as linhas 7 e 8 são capazes de descobrir um valor de $\varphi(1)$, no caso da condição ser verdadeira. Uma pergunta natural é a seguinte: *"E se no lugar de descobrirmos o valor de $\varphi(1)$, procurarmos quais elementos poderiam construir a cabeça de uma permutação que gere um valor da $\varphi(1)$ pré-fixado?"*.

Como $\varphi(1)$ pode assumir *n* configurações e o número de elementos necessários de uma solução linear ξ para gerar uma solução do PQA é (n – 1), foi desenvolvido um algoritmo que constrói uma matriz nx$(n–1)$ que armazena, em cada *i* linha da matriz ($i = 1, \ldots, n$) a cabeça de uma permutação ξ pseudo-viável que responde ao questionamento anterior, isto é, $\varphi(1) = i$, chamamos matriz **HeadQ**.

De posse disso, podemos gerar todas as pseudo-viáveis com o valor de $\varphi(1)$ que foi pré-fixado. A seguir apresentaremos um algoritmo chamado **ConstroiHead** que alcança os objetivos descritos acima.

Algoritmo 2: *ConstróiHead(N)*
1: **for** $t = 1, \ldots, N$ **do**
2: $ListaIJ[t] \leftarrow \psi^{-1}(t) = (i,j)$
3: $HeadQ[i,j-1] \leftarrow t$
4: $HeadQ[j,i] \leftarrow t$
5: **end for**

Exemplo 2: *Para ilustrar, considere n = 4. A matriz* ***HeadQ*** *assume a seguinte configuração:*

Tabela 3 – Matriz *HeadQ*.

	$\xi(1)$	$\xi(2)$	$\xi(3)$	
1	1	2	3	$\xi = \left(\begin{array}{ccc\|c} 1 & 2 & 3 & \dots \\ 1 & 2 & 3 & \dots \end{array}\right) \rightarrow \varphi(1) = 1$
2	1	4	5	$\xi = \left(\begin{array}{ccc\|c} 1 & 2 & 3 & \dots \\ 1 & 4 & 5 & \dots \end{array}\right) \rightarrow \varphi(1) = 2$
3	2	4	6	$\xi = \left(\begin{array}{ccc\|c} 1 & 2 & 3 & \dots \\ 2 & 4 & 6 & \dots \end{array}\right) \rightarrow \varphi(1) = 3$
4	3	5	6	$\xi = \left(\begin{array}{ccc\|c} 1 & 2 & 3 & \dots \\ 3 & 5 & 6 & \dots \end{array}\right) \rightarrow \varphi(1) = 4$

Prova de Correção: A invariante que rege o Algoritmo 2 é: *"No final de cada iteração a matriz recebe em duas posições ((i,j – 1) e (j,i)) o elemento t, tal que t é o elemento necessário para construir as φ's, para as quais φ(1) = i e φ(1) = j"*.

Início: a matriz e a lista ListaIJ estão vazias, ambas são construídas a cada iteração do *loop*.

Manutenção: Ao final de cada iteração o algoritmo se aproveita da estrutura da ListaIJ para alocar o elemento *t*, o índice da ListaIJ, em posições que irão formar as cabeças das linhas tais que $\varphi(1) = i$ e $\varphi(1) = j$.

Término: Note que $N = n(n-1)/2$, como a matriz tem dimensões $n \times (n-1)$, e a cada iteração são alocados 2 elementos, então ao final do *loop* o número de alocações na matriz é $2*n(n-1)/2$ que é exatamente o número de elementos da matriz. Como a alocação é dada pelos pares $(i,j-1)$ e (j,i) e observando que $1 \le i < j \le n$, temos que não existem duas alocações em uma mesma posição, o que assegura que a matriz é toda preenchida. □

Proposição 1: *Permutando-se os (n – 1) elementos das n linhas existentes na HeadQ, gera-se as n! soluções do PQA.*

Prova: Considere um PQA cujas matrizes F e D são de dimensão $n \times n$. As colunas da matriz *HeadQ* são as imagens de $\xi(1), \xi(2), \dots, \xi(n-1)$. Como a *HeadQ* tem ordem $n \times (n-1)$, cada linha $k = 1, \dots, n$ gera $(n-1)!$ soluções quadráticas com $\varphi(1) = k$. Sendo n linhas, geramos as $n \times (n-1)! = n!$ soluções para o PQA. □

De posse das permutações $\xi \in \Pi_N$, soluções do PAL(Q), podemos gerar $\rho \in \Pi_N$, soluções do PAL(Q^*) que certamente serão capazes de gerar uma $\varphi \in \Pi_n$ solução do PQA. É necessário trabalhar com as soluções do PAL(Q^*) pois o Teorema das Inversões é aplicado quando os vetores que definem o problema são ordenados tais como $\mathbf{F}^-$ e $\mathbf{D}^+$.

Para a construção de uma nova matriz, que chamaremos de ***HeadQ****, fazemos uso da bijeção $\rho = \phi_D \circ \xi \circ \phi^{-1}_F$ para efetuar a transformação entre essas matrizes. Como estamos trabalhando somente com as cabeças das permutações $\xi \in \Pi_N$, temos que assegurar que as imagens das aplicações ϕ^{-1}_D serão *1, ... , n – 1*. Aplicaremos ϕ_F, nos valores de *ξ(i)* com *i = 1, ... , n – 1*, indicando em que posição os elementos das linhas da matriz ***HeadQ**** estão na permutação $\rho \in \Pi_N$.

Para melhor esclarecimento, usaremos as permutações ϕ_F = (2 6 1 3 4 5) e ϕ_D = (5 4 2 3 6 1) do Exemplo 1.

$$\underset{\phi_D}{\begin{pmatrix}1&2&3&4&5&6\\5&4&2&3&6&1\end{pmatrix}} \circ \underset{\xi}{\begin{pmatrix}1&2&3&4&5&6\\1&2&3&4&5&6\end{pmatrix}} \circ \underset{\phi_F^{-1}}{\begin{pmatrix}1&2&3&4&5&6\\3&1&4&5&6&2\end{pmatrix}} = \underset{\rho}{\begin{pmatrix}\underline{1}&\underline{2}&3&4&5&\underline{6}\\\underline{2}&\underline{5}&&&&\underline{4}\end{pmatrix}}$$

$$\circ \begin{pmatrix}1&2&3&4&5&6\\1&4&5&&&\end{pmatrix} = \begin{pmatrix}\underline{1}&\underline{2}&3&4&5&\underline{6}\\\underline{6}&\underline{5}&&&&\underline{3}\end{pmatrix}$$

$$\circ \begin{pmatrix}1&2&3&4&5&6\\2&4&6&&&\end{pmatrix} = \begin{pmatrix}\underline{1}&\underline{2}&3&4&5&\underline{6}\\\underline{1}&\underline{4}&&&&\underline{3}\end{pmatrix}$$

$$\circ \begin{pmatrix}1&2&3&4&5&6\\3&5&6&&&\end{pmatrix} = \begin{pmatrix}\underline{1}&\underline{2}&3&4&5&\underline{6}\\\underline{1}&\underline{2}&&&&\underline{6}\end{pmatrix}$$

Figura 4 – Ilustração da construção da *HeadQ**.

Para conhecer a imagem de cada elemento da matriz basta aplicar a função ϕ_D, pois como vimos a construção da matriz *HeadQ* está baseada na permutação identidade de $\xi \in \Pi_N$ e a imagem da composição das permutações $\phi_D \circ \xi \circ \phi^{-1}_F$ é exatamente ϕ_D.

Exemplo 3: Para um exemplo qualquer com *n = 4*, a matriz *HeadQ* é apresentada na forma exibida no lado esquerdo da Tabela 4, aplicando a forma geral da *HeadQ** no exemplo 1 (com ϕ_F *= (2 6 1 3 4 5) e* ϕ_D *= (5 4 2 3 6 1)*) obtemos a matriz exibida no lado direito da Tabela 4. Note que em negrito está a matriz *HeadQ*.

Tabela 4 – Matriz *HeadQ**.

	$\phi_F(\xi(1))$	$\phi_F(\xi(2))$	$\phi_F(\xi(3))$
1	ϕ_D **(1)**	ϕ_D **(2)**	ϕ_D **(3)**
2	ϕ_D **(1)**	ϕ_D **(4)**	ϕ_D **(5)**
3	ϕ_D **(2)**	ϕ_D **(4)**	ϕ_D **(6)**
4	ϕ_D **(3)**	ϕ_D **(5)**	ϕ_D **(6)**

	2	*6*	*1*
1	*5*	*4*	*2*
2	*5*	*3*	*6*
3	*4*	*3*	*1*
4	*2*	*6*	*1*

Uma vez produzido a matriz *HeadQ** conseguimos gerar soluções do PAL(Q*) que certamente irão gerar uma ξ pseudo-viável. Como o método de pré-seleção das soluções é, pelo Teorema das Inversões, buscar soluções com baixo número de inversões, uma forma simples de alcançar este objetivo é a ordenação dos elementos de uma permutação. Nesse caso, ordenaremos os elementos da matriz ***HeadQ**** fazendo as mesmas trocas na matriz ***HeadQ***.

Exemplo 4: Para ilustrar usaremos a primeira linha da ***HeadQ*** no diagrama da Figura 5.

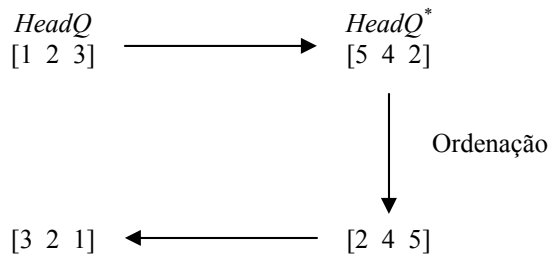


Figura 5 – Diagrama com a idéia de minimização do número de inversões.

A permutação $\varphi = (1\ \ 2\ \ 3\ \ 4)$ que foi gerada pela linha da matriz antes da ordenação apresenta um custo de *126* e após a ordenação temos uma φ com custo de *112* dada pela $\varphi = (1\ \ 4\ \ 3\ \ 2)$. Repetimos esse processo para todas as linhas da matriz, gerando *n* soluções φ's de boa qualidade.

4. Algoritmo Proposto

Em linhas gerais, o algoritmo que chamamos de ***HeuristicHead***, é basicamente um algoritmo híbrido que combina um método construtivo de soluções iniciais com uma técnica de melhoramento (busca local). Para o método de construção das soluções iniciais e um melhoramento prévio usamos a matriz ***HeadQ*** e ***HeadQ**** e toda sua teoria envolvida, destacando o Teorema das Inversões e os resultados probabilísticos relacionados a ele. Para o melhoramento das soluções geradas é usado uma método de busca local em torno de uma vizinhança de cada solução. O funcionamento do algoritmo pode ser traduzido no seguinte pseudo-código:

Algoritmo3: *HeuristicHead(n)*
1: Entrada de Dados;
2: Construção das soluções iniciais na matriz *HeadQ;*
3: **For** i =1, ... , n **do**
4: Constróiφ(*HeadQ*[i]);
5: BuscaLocal(φ);
6: Atualizar Solução;
7: **end for**
8: Retorna a melhor solução encontrada;

A fase de construção das soluções iniciais, que é onde reside o foco desse trabalho, é composta por várias etapas que podem ser dispostas da seguinte forma:

1. Leitura dos dados;
2. Ordenação dos vetores de Fluxo e Distância;
3. Construção da ϕ_F e ϕ_D;
4. Construção da matriz *HeadQ*;
5. Construção da matriz *HeadQ**;
6. Ordenação das linhas na matriz *HeadQ**, com as respectivas trocas da matriz *HeadQ*.

Os itens 2 e 3 devem pertencer a todo algoritmo que trabalhe com uma correspondência entre o Problema de Alocação Linear e sua relaxação na forma do Problema de Alocação Linear, já que são os procedimentos justamente encarregados desta relaxação. A construção da matriz *HeadQ* é independente dos procedimento 1, 2 e 3. A construção da *HeadQ* e da $HeadQ^*$ é o que caracteriza nossa heurística. Uma vez de posse das permutações lineares efetuamos a ordenação de tais permutações com base no Teorema das Inversões, para critério de uma melhoria prévia das soluções iniciais. Essa melhoria faz com que a busca local se torne menos onerosa, por conseqüência aumentando o desempenho do algoritmo.

Note que para um exemplar de dimensão n nossa construção produz n soluções iniciais. Devido à alta combinatoriedade do PQA e à estrutura de vizinhança adotada nesse trabalho, consideramos que seria conveniente promover sistemáticas perturbações nas soluções iniciais com o objetivo de gerar novos elementos para a etapa da busca local. Tais perturbações são descritas da seguinte forma: dada uma linha da matriz *HeadQ* $l_i = [l_{i1}, l_{i2}, \ldots, l_{i(n-1)}]$; $i = 1, \ldots, n$, geramos $(n-2)$ novas soluções da seguinte maneira:

$$l'_i = [l_{i1}, l_{i2}, \ldots, l_{i(j-1)}, l_{i(j)}, \ldots, l_{i(n-1)}];\ \forall j = 1, \ldots, n-2;$$

onde l'_i gerará uma nova solução inicial que sofreu um pequeno aumento no número de inversões, porém ainda é considerada boa.

O procedimento de busca local exige que se defina uma estrutura de vizinhança. Neste trabalho adotamos uma estratégia encontrada em [LPR94], conhecida como k-troca, para este trabalho consideramos k = 2.

Além do algoritmo serial, seguindo uma tendência para problemas de grande porte, foi implementada uma versão paralela do mesmo. A paralelização do algoritmo se justifica pelo baixo custo e por conseguir tempos computacionais relativamente satisfatórios para problemas de grande porte. Tal paralelização se resume na distribuição e balanceamento das tarefas, onde cada tarefa é a construção da permutação quadrática correspondente a uma linha da matriz *HeadQ*, mais as (n – 2) perturbações mencionadas e as buscas locais para cada uma dessas [1 + (n – 2)] soluções. A distribuição é feita através da divisão do número de linhas da matriz pelo número de processadores. Caso exista resto nesta divisão, as linhas restantes serão distribuídas aos processadores que primeiro desocuparem.

5. Resultados Computacionais

A implementação serial foi executada em um Pentium IV, 1,3 GHz com 256 de memória RAM. Nessa etapa dos testes foram executados exemplares de ordem menores ou iguais à 30, pois o objetivo inicial foi analisar o desempenho do algoritmo no sentido de observar a relação **qualidade de solução × tempo**.

Apesar do algoritmo *HeuristicHead* não garantir a otimalidade das soluções, o Teorema das Inversões assegura a boa qualidade das mesmas. Uma característica encorajadora é a relação **qualidade de solução × tempo**, mencionada anteriormente. Como exemplo podemos citar o nug20, que apesar de não ter atingido o ótimo conseguiu um erro de 1,01% gastando para isso 380 iterações em 8,2 segundos. A Tabela 5 apresenta o nome do exemplar, o melhor resultado disponível da QAPLIB, o resultado da heurística *HeristicHead*, tempo computacional e taxa de erro. Para uma comparação podemos citar o trabalho [LPR94] que apresenta uma implementação serial do GRASP. O algoritmo apresentado nesse trabalho

obteve os mesmos resultados do GRASP em [LPR94] para os exemplos da Tabela 5, com exceção dos exemplos tai15a e tai30a que não são apresentados em [LPR94].

Tabela 5 – Resultados obtidos de exemplares conhecidos ($n \leq 30$).

Exemplo	QAPLIB	*HeuristicHead*	Tempo (seg)	Erro
nug12	578	582	0,20	0,68%
rou12	235528	235528	0,18	0%
nug15	1150	1152	1,08	0,17%
tai15a	388214	390782	0,73	0,66%
nug20	2570	2596	8,20	1,01%
scr20	110030	110058	8,80	0,03%
tai30a	1818146	1858226	175,85	2,2%
nug30	6124	6156	153,68	0,52%

Seguindo uma tendência atual para problemas de grande porte [BMCP98] [JG05], o algoritmo foi implementado em uma versão paralela, tal algoritmo foi executado no *cluster* da Universidade Federal do Espírito Santo (UFES) montado no Laboratório de Computação de Alto Desempenho [LCAD], que tem como estrutura a seguinte configuração: 64 processadores ATHLON XP 1800, onde o *Master* possui 512MB de RAM e nós com 256MB de RAM. Os testes da versão paralela foram feitos em dois estágios. Primeiramente, executamos um mesmo exemplo várias vezes com números diferentes de processadores, mesmo sabendo que a solução encontrada seria a mesma, pois esse estágio o objetivo era analisar o progresso da relação **número de processadores × tempo** (em segundos).

Tabela 6 – Relação **número de processadores** × **tempo** para o exemplar tai40a.

tai40a	
Processadores	**Tempo (seg)**
1	1325,25
2	662,88
10	133,57
12	134,53
15	100,05
19	98,47
20	65,85
25	66,77

Para o exemplo tai40a, o custo da melhor solução viável disponível na QAPLIB é igual a 3139370 e o algoritmo *HeuristicHead* obteve custo igual a 3212174 com erro de 2,3%. Na Tabela 6, o algoritmo mostra um comportamento regular na relação **número de processadores × tempo**. Note que o tempo computacional para 10 e 12 processadores, assim como para 15 e 19 e ainda 20 e 25, é bem próximo. Para explicar este fenômeno basta dividirmos a ordem da matriz do exemplo pelo número de processadores. Por exemplo,

temos que 40 ÷ 10 = 4 e 40 ÷ 12 = 3,33..., assim para 10 processadores, cada um computa 4 linhas da matriz. Para 12 processadores teremos cada um computando 3 linhas, totalizando 36 linhas, restando 4 linhas que irão ser distribuídas a quatro processadores (uma linha para cada processador) que primeiro terminarem a tarefa inicial. Sendo assim, temos 4 processadores computando 4 linhas e 8 computando 3 linhas. Isso não é interessante, pois os processadores com a tarefa menor (computar 3 linhas da matriz) são obrigados a "esperar" o término dos demais processadores que estão computando uma linha a mais da matriz, para depois comparar os resultados. Essa "espera" é responsável pela proximidade nos tempos de processamento. O mesmo acontece para 15 e 19, e 20 e 25 processadores.

Uma vez apresentada a regularidade e desempenho do algoritmo e fazendo uso da conclusão anterior, nos próximos testes o número de processadores usados será sempre um divisor da ordem do exemplo referente. A Tabela 7 segue os mesmos moldes da Tabela 5, acrescida apenas do número de processadores.

Ainda sobre a questão do tempo computacional, a pergunta que surge observando a Tabela 7 é: *"por que exemplos de mesma ordem e calculados com o mesmo número de processadores, com sko100a, sko100b, sko100c e sko100d, possuem tempo computacionais* consideravelmente *diferentes?"*. A resposta está no tempo computacional gasto na fase de busca local.

Pela Tabela 7 observamos que para exemplos considerados de grande porte, em nenhum caso o algoritmo proposto encontrou a melhor solução conhecida. Contudo, a relação **qualidade de soluções × tempo** nos dá resultados encorajadores. Como comparação podemos destacar o trabalho [JG05] onde foi apresentado um estudo do PQA usando, Busca Tabu e Colônia de Formigas. O algoritmo apresentado nesse trabalho obteve os mesmos resultados na maioria dos exemplos mostrados na Tabela 7, com exceção dos exemplos tai(40a, 50a, 60a, 80a e 100a) que apresentaram um resultado inferior aos de [JG05] (que para estes exemplos obteve um erro médio de 0,12%, enquanto o nosso algoritmo obteve 2,5% de erro médio).

Tabela 7 – Alguns resultados obtidos de exemplo considerados grandes.

Exemplo	QAPLIB	*HeuristicHead*	Processadores	Tempo (seg)	Erro
sko42	15812	15910	22	135,33	0,6%
sko56	34458	34760	28	1232,98	0,8%
sko64	48498	48782	32	1881,88	0,5%
sko72	66256	66780	36	3983,72	0,8%
sko90	115534	116430	45	24830,23	0,7%
sko100a	152002	152848	25	48494,22	0,5%
sko100b	153890	154796	25	48220,22	0,5%
sko100c	147862	148688	25	46990,67	0,5%
sko100d	149576	150356	25	47668,47	0,5%
tai50a	4941410	5090380	25	252,43	3%
tai60a	7205962	7424510	30	801,47	3%
tai80a	13515450	13853702	40	4508,35	2,5%
tai100a	21075842	21584472	50	17576,03	2,4%
wil50	48816	48860	25	404,87	0,1%
wil100	273038	274026	25	66301,63	0,3%

6. Conclusão

A principal proposta implementada nesse estudo foi de gerar, de forma viável computacionalmente, soluções do PAL que fossem sempre viáveis para o PQA a ele correspondente, isto é, promover um mapeamento das soluções quadráticas no universo das soluções lineares. Vencido esta etapa, através da matriz ***HeadQ***, produziu-se um algoritmo de complexidade polinomial para a construção da mesma. A união de toda a teoria envolvida culminou na heurística ***HeuristicHead***, que utiliza as matrizes ***HeadQ*** e ***HeadQ**** aliadas ao Teorema das Inversões para gerar soluções de boa qualidade que irão ser usadas como soluções iniciais para um processo de busca local. Através de teste, a ***HeuristicHead*** apresentou uma boa relação **qualidade de soluções** × **tempo computacional**. Na versão paralela com os testes de desempenho concluiu-se que para um melhor uso dos recursos o número de processadores deve ser um divisor da dimensão do exemplar para não se perder tempo com a espera de liberação de processadores. Na Tabela 6 pode-se notar que quando aumentamos o número de processadores, os tempos computacionais gastos com o exemplar tai40a possuem uma relação com o número de processadores, por exemplo, 1325.25 é aproximadamente 2 x 662.88, bem como 1325.25 é aproximadamente 20 x 65.85. Sendo assim, levando essa relação aos exemplares da Tabela 7, os tempos computacionais seriam muito altos se utilizássemos num número baixo de processadores. Para trabalhos futuros sugerimos uma implementação do algoritmo introduzindo um componente aleatório na fase de construção das soluções iniciais. Desta forma, haveria uma diversificação no espaço de busca, podendo melhorar o desempenho do algoritmo em relação à qualidade das soluções.

Referências Bibliográficas

[Ab84] Abreu, N.M.M. (1984). Um estudo algébrico e combinatório do problema quadrático de alocação segundo Koopmans e Beckmann. Tese de D.Sc., Programa de Engenharia de Produção – COPPE/UFRJ, Brasil.

[ABQG02] Abreu, N.M.M.; Boaventura, P.O.; Querido, T.M. & Gouvêa, E.F. (2002). Classes of quadratic assignment problem instances: isomorphism and difficulty measure using a statistical approach. *Discrete Applied Mathematics*, **124**, 103-116, Elsevier Science Inc.

[ABGL02] Anstreicher, K.M.; Brixius, N.W.; Goux, J.P. & Linderoth, L. (2002). Solving large quadratic assignment problem on computation grid. *Mathematical Programming*, Series B, **91**, 563-588. Winner, SIAM Activity Group on Optimization (SIAG/OPT) Prize.

[AB01] Anstricher, K.M. & Brixius, N.W. (2001). A new bound for the quadratic assignment problem based on convex quadratic programming. *Mathematical Programming*, **89**, 341-357.

[Boa03] Boaventura, P.O. (2003). Combinatorial instruments in the design of a heuristic for the quadratic assignment problem. *Pesquisa Operacional*, **23**, 282-402.

[BR83] Burkard, R.E. & Rendl, F. (1983). A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operations Research*, **17**, 169-174.

[BMCP98] Bruengger, A.; Marzetta, A.; Clausen, J. & Perregaard, M. (1998). Solving Large-Scale QAP Problems in Parallel with the Search Library ZRAM. *Journal of Parallel and Distributed Computing*, **50**, 157-169.

[BS78] Burkard, R.E. & Stratman, K.H. (1978). Numerical investigations on quadratic assigment problem. *Naval Research Logistics Quartely*, **148**, 25-129.

[Ce98] Çela, E. (1998). *The Quadratic Assignment Problem – theory and algorithms*. Series in Combinatorial Optimization, vol. 1, Kluwer Academic Publishers.

[DIMACS94] Pardalos, P.M. & Wolkowicz, H. (1993). Quadratic Assignment and Related Problem. *DIMACS – Series in Discrete Mathematics and Theoretical Computer Science*, **16**.

[GTD97] Gambardella, L.M.; Taillard, E.D. & Dorigo, M. (1997). Ant Colonies for the QAP. Technical Report IDSIA97-4, IDSIA, Laguno, Switzerland.

[Gi62] Gilmore, P.C. (1963). Optimal and suboptimal algorithms for the quadratic assignment problem. *Siam Journal on Applied Mathematics*, **10**, 305-313.

[HIAI02] Hasegawa, M.; Ikeguchi, T.; Aihara, K. & Itoh, K. (2002). A novel chaotic search for quadratic assignment problem. *European Journal of Operational Research*, **139**(3), 543-556.

[JG05] James, T. & Glover, F. (2005). Sequential and Parallel Path-Relinking Algorithms for the Quadratic Assignment Problem. *IEEE Intelligent Systems*, May.

[KB57] Koopmans, T.C. & Beckmann, M.J. (1957). Assignment problem and the location of economic activities. *Econometrica*, **25**, 53-75.

[La63] Lawler, E. (1963). The quadratic assignment problem. *Management Science*, **9**, 586-599.

[LCAD] Laboratório de Computação de Alto Desempenho. <http://www.inf.ufes.br/~lcad>. Última visita: 13/05/2004.

[LPR94] Li, Y.; Pardalos, P.M. & Resende, M.G.C. (1994). A greedy randomized adaptive search procedures for the quadratic assignment problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, **16**, 237-261.

[MCD94] Maniezzo, V.; Colorni, A. & Dorigo, M. (1994). The Ant System Applied to the quadratic assignment problem. Technical Report IRIDIA/94-28, Université Libre deBruxelles, Belgium.

[Que94] Querido, T.M. (1994). *Simulated annealing* no grafo das inversões do problema quadrático de alocação. Tese de D.Sc., Programa de Engenharia de Produção – COPPE/UFRJ, Brasil.

[Ran00] Rangel, M.C. (2000). Contribuições Algébricas ao Problema Quadrático de Alocação. Tese de Doutorado, Programa de Engenharia de Produção – COPPE/UFRJ.

[RA01] Rangel, M.C. & Abreu, N.M.M. (2001). Um parâmetro para avaliar a qualidade das soluções do problema quadrático de alocação. **In**: CD-Rom *XXXIII SBPO*, Sobrapo, ILTC, **1**, 1277-1287.

[RA03] Rangel, M.C. & Abreu, N.M.M. (2003). Ordenações parciais nos conjuntos das soluções do problema de alocação linear e quadrático. *Pesquisa Operacional*, **23**(2), 265-284, Sobrapo.

[RR04] Resendo, L.C. & Rangel, M.C. (2004). Uma Proposta Para Gerar Soluções de Boa Qualidade para o Problema Quadrático de Alocação. **In**: CD-Rom *XXXVI SBPO*, Sobrapo.

[TS94] Tate, D.E. & Smith, A.E. (1995). A genetic approach to the quadratic assignment problem. *Computers and Operations Research*, **22**, 73-83.