Um mapeamento das soluções do Problema Quadrático de Alocação (PQA)
no universo das soluções do Problema de Alocação Linear (PAL)

Leandro Colombi Resendo

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática

Aprovada por:

Profa. Maria Cristina Rangel, D.Sc.

Profa. Nair Maria Maia de Abreu, D.Sc.

Prof. Arlindo Gomes de Alvarenga, D.Sc.

VITÓRIA, ES – BRASIL
JUNHO DE 2004

RESENDO, LEANDRO COLOMBI

Um Mapeamento das soluções do Problema Quadrático de Alocação (PQA) no universo das soluções do Problema de Alocação Linear (PAL) [Vitória] 2000

x, 62 p., 29.7 cm, (PPGI/UFES, M.Sc., Informática, 2004)

Tese - Universidade Federal do Espírito Santo, PPGI

1 - Problema Quadrático de Alocação

2 - Otimização Combinatória

3 - Meta-Heurística

I. PPGI/UFES II. Título

Aos meus pais Cassemiro e Ana

Agradecimentos

Gostaria de agradecer a todos que, direta ou indiretamente, contribuiram para que este trabalho se realizasse. Em especial, agradeço:

À Profa. Maria Cristina Rangel, e orientadora, pela amizade e dedicação tão fundamentais para a evolução e conclusão desse trabalho.

À CAPES pelo apoio financeiro através da bolsa de mestrado, indispensável à realização deste trabalho.

Aos colegas de estudos, que compartilharam de tantas horas de trabalho durante o cumprimento dos créditos. Destacando Moacir Canella Bortoloso e Soterio Ferreira de Souza.

Ao grupo do Laboratório de Computação de Alto Desempenho (LCAD), em especial ao amigo Leonardo Muniz Lima que foi o principal colaborador e entusiasta da paralelização do algoritmo.

Aos meus amigos e à minha noiva Rosimery Aliprandi Ribeiro, que contribuíram me passando tranqüilidade nos momentos de maior dificuldade.

Resumo da Tese apresentada ao PPGI/UFES como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

Um mapeamento das soluções do Problema Quadrático de Alocação (PQA)
no universo das soluções do Problema de Alocação Linear (PAL)

Leandro Colombi Resendo

Junho/2004

Orientador: Maria Cristina Rangel
Programa: Informática

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. Descobrir quais soluções lineares são viáveis para o PQA é uma tarefa extremamente difícil devido ao número de soluções lineares ser bem maior que a número de soluções quadráticas. 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 algoritmo 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.

Abtract of thesis presented to PPGI/UFES as a parcial fulfillment of the requeriments for the degree of Master of Science (M.Sc.)

Mapping of Quadratic Assignment Problem solutions into
the universe of Linear Assignment Problem solutions.

Leandro Colombi Resendo

June/2004

Advisor: Maria Cristina Rangel
Departament: Informatica

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 cost of QAP solution is associated to the number of inversions of its linear correspondent. To find out which linear solutions are QAP feasible is very difficult because there are much more LAP solutions than QAP solutions. In this work we construct a matrix that stores information of LAP solutions that are able to generate QAP solutions. Using together the concepts of this mapping and the Inversion Theorem, we present a constructive algorithm that generates good initial solutions. The great advantage of this matrix is the low cost of computational time and memory. A parallel version of this algorithm is also proposed and implemented in this work.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Problema Quadrático de Alocação

A primeira apresentação do Problema Quadrático de Alocação (PQA) foi dada por Koopmans e Beckmann [KB57] em 1957, quando definiram um modelo matemático para analisar problemas de produção envolvendo recursos indissociáveis. A partir de então, muito pesquisadores se dedicaram ao PQA tanto na parte teórica quanto na aplicação prática. Aplicações importantes deste modelo são encontradas nos problemas de *Lay-Out* ou problemas de arranjo físico. Formulando este problema com grafos, temos definidos dois grafos completos (cliques K_F e K_D) com as arestas valoradas pelos fluxos entre as atividades econômicas e distâncias entre as localidades. Em linhas gerais, esse problema é a alocação de n facilidades à n localidades com o objetivo de minimizar o custo dessa alocação, sendo esse custo definido pelo somatório dos produtos das distâncias pelos fluxos.

O PQA possui muitas aplicações as quais podemos citar um trabalho sobre *design* de painéis de controle e teclados por Pollatschek *et al.* [PQR76]; o balanceamento de turbinas hidráulicas por Laporte e Mercure [LM88]; estudo da otimização de circuitos impressos por Steinberg [St61]. Uma particularização do PQA é o conhecido problema do caixeiro viajante que tem uma estrutura especial para as matrizes de Fluxo e Distância.

O Problema Quadrático de Alocação pertence a classe de problemas *NP-Hard*, demonstrado por Sahni e Gonzalez [SG76]. Para mensurar sua dificuldade vale dizer que os problemas testes de ordem superior a 25 ainda são considerados de

grande porte e os de ordem 30 ou mais, permaneceram insolúveis otimamente por décadas, como a Nug30, Kra30a, Kra30b e outros disponíveis na QAPLIB. Estes foram resolvidos usando algoritmos exatos baseados no método de *branch and bound*, por Anstreicher *et al.* em 2000 [ABGL00].

Na literatura encontramos alguns artigos publicados na década de 60 que serviram de estímulo para o surgimento de novas pesquisas. Entre esses precursores podemos citar Lawler [La63] que foi um dos primeiros a formular o problema quadrático de alocação como um problema de programação linear, e Gilmore [Gi62] que introduziu um limite inferior para o custo das soluções ótimas do PQA. Esses provavelmente foram grandes entusiastas de muitas pesquisas.

Ainda podemos citar textos importantes na literatura que são dedicados ao PQA: a edição do primeiro livro exclusivamente sobre PQA, de Pardalos *et al.* [PRW94], contendo artigos sobre limite inferior, complexidade computacional, aproximações heurísticas e generalizações sobre o problema; mais recentemente Burkard e Çela escreveram um *survey* sobre o PQA publicado em [BCPP98]; atualmente um dos textos de referência para o PQA é o livro de Eranda Çela [Ce98]; e um *survey* comentado contento quase todas as publicações relacionada ao PQA de Abreu *et al.* [ABL04].

Além dos citados é importante registrar os textos que tiveram uma influência direta na parte algébrica deste trabalho que são a tese de doutorado de Abreu [Ab84], seguida do trabalho de Querido [Que94] e mais recentemente a tese de doutorado de Rangel [Ran00] e o relatório técnico [Ran01].

1.2 Formulações

A seguir vamos mostrar a primeira formulação apresentada por Koopmans e Beckmann [KB57] para o PQA. Essa apresentação é basicamente uma associação um-a-um entre os elementos do conjunto das permutações da matriz $X = (x_{ij})$ de dimensões $n \times n$, com o objetivo de minimização.

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} a_{ij} b_{kl} x_{ik} x_{jl}, \tag{1.1}$$

sujeito a

$$\sum_{i=1}^{n} x_{ij} = 1 \quad 1 \leq j \leq n, \tag{1.2}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad 1 \leq i \leq n, \tag{1.3}$$

$$x_{ij} \in \{0,1\} \quad 1 \leq i,j \leq n, \tag{1.4}$$

onde $x_{ij} = 1$ se a facilidade i for alocada na posição j e $x_{ij} = 0$ caso contrário. As restrições 1.2, 1.3 e 1.4 obrigam que a alocação de uma facilidade seja única por localidade.

A formulação linear apresentada por Lawler [La63] foi feita usando a introdução de variáveis binárias $y_{ijkl} = x_{ik}x_{jl}$ e custo $c_{ijkl} = a_{ij}b_{kl}$ obtendo assim a seguinte formulação:

$$min \sum_{i,j,k,l,=1}^{n} c_{ijkl}y_{ijkl}, \tag{1.5}$$

sujeito a

$$\sum_{i=1}^{n} x_{ik} = 1, \qquad 1 \leq k \leq n, \tag{1.6}$$

$$\sum_{k=1}^{n} x_{ik} = 1, \qquad 1 \leq i \leq n, \tag{1.7}$$

$$\sum_{i,j,k,l=1}^{n} y_{ijkl} = n^2, \tag{1.8}$$

$$x_{ik} + x_{jl} - 2y_{ijkl} \geq 0, \quad 1 \leq i,j,k,l \leq n, \tag{1.9}$$

onde $x_{ij} = 1$ se a facilidade i for alocada na posição j; $x_{ij} = 0$ caso contrário; e $y_{ijkl} = 1$ se as unidades i e j são alocadas em k e l, ao mesmo tempo. O problema dessa formulação é o grande aumento do número de variáveis e de restrições.

O modelo usado nesse trabalho, que faremos uma apresentação mais detalhada no próximo capítulo, apresenta uma solução na forma de permutações. Burkard e Stratmann [BS78] apresentaram o PQA dado da seguinte forma:

$$Z_\varphi = \underset{\varphi \in \Pi_n}{\text{Min}} \sum_{i,j=1}^{n} C_{ij\varphi(i)\varphi(j)}. \tag{1.10}$$

Os custos da função objetivo são apresentados, segundo o contexto econômico dado por Koopmans e Beckmann, pelo produto $f_{ij}d_{\varphi(i)\varphi(j)}$ tal que $1 \leq i < j \leq n$, Π_n

é o conjunto de todas as permutações φ de $\Omega_n = \{1, ..., n\}$ e F= (f_{ij}) e D= (d_{pq}) são matrizes do fluxo e da distância respectivamente. Mais recentemente temos as seguintes publicações utilizando esse modelo, Abreu *et al.* em 2002 [ABQG02], Hasegawa *et al.* em 2002 [HIAI02], Boaventura-Netto em 2003 [Boa03] e Rangel e Abrel em 2003 [RA03]. Outras formulações para o PQA podem ser encontrados no livro de Çela [Ce98].

1.3 Métodos de Solução

Os estudos apresentados até hoje se dividem em dois grupos básicos: algoritmos que apresentam solução exata (ou ótima) e algoritmos heurísticos (ou de soluções sub-ótimas).

Para o desenvolvimento de ambos os métodos está embutido o estudo de limites inferiores. O desempenho dos algoritmos de solução ótima, em geral, trabalham com métodos de enumeração implícita que dependem diretamente da qualidade de seus limites inferiores. Um exemplo disso é o método de *branch and bound* que necessita de um bom limite inferior para obter desempenho satisfatório, sendo também empregado em algoritmos heurísticos para teste da qualidade das soluções. Dentre os limites mais conhecidos, temos o de Gilmore [Gi62] e Lawler [La63] por serem mais simples e de baixo custo computacional, porém sua inconveniência está no fato de obtermos limites de baixa qualidade para exemplares grandes. Outro limite de destaque para o PQA é o de Anstreicher e Brixius em 2000 [AB01], que desenvolveram um limite inferior para o PQA usando programação semidefinida e programação quadrática convexa.

Devido a complexidade da resolução das instâncias maiores que 25, para encontrar a solução ótima são necessárias melhorias em algoritmos de programação matemática e a utilização de plataformas computacionais poderosas. Anstreicher *et al.* em seu artigo [ABGL00] apresentam a solução ótima para a instância Kra30b usando uma técnica exata do tipo *branch and bound* com um novo limite inferior determinado em [AB01]. O tempo de computação necessário atingiu 182 dias usando uma única estação de trabalho. Os algoritmos exatos apresentam a desvatagem de precisarem de muito tempo de processamento e um grande espaço de armazenamento em memória. Estas complexidades justificam as pesquisas direcionadas a

algoritmos heurísticos que tenderam para a seguinte classificação:

- algoritmos construtivos;
- algoritmos de melhoramento;
- algoritmos híbridos;
- algoritmos baseados na teoria dos grafos;

Nos algoritmos construtivos a solução é construída passo a passo, em outras palavras, as alocações são feitas uma a uma, até que o arranjo esteja completo.

Algoritmos de melhoramento tem como característica básica gerar aleatoriamente uma solução inicial, e com base nesta solução, realizar trocas sistemáticas entre as alocações e avaliar os resultados. Uma heurística que explora essa idéia é o GRASP - *Greedy Randomize Adaptive Search Procedures* apresentada por Li *et al.* [LPR94], que é dividido em duas fases, a construção de uma solução inicial aleatória, porém tomando alguns cuidados para gerar boas soluções. Portanto, é exigido um menor esforço computacional na fase de melhoramento (busca local), aumentando seu desempenho.

Os híbridos tem por característica o fato de combinar estratégias de algoritmos ótimos e sub-ótimos. Burkard e Stratman [BS78] propuseram um algoritmo heurístico que usa o método de *branch and bound* para determinar uma solução inicial e um algoritmo de melhoramento que faz trocas sucessivas em pares de alocações e pára quando não houver melhoria na solução.

Não podemos deixar de citar ainda as meta-heurísticas que usam como estratégias de melhoramento os fenômenos naturais. Dentre elas destacamos: o *Simulated Annealing*, aplicado ao PQA por Burkard e Rendl [BR83]; o Algoritmo Genético, que teve sua aplicação ao PQA por vários pesquisadores, entre eles Tate e Smith [TS94] e Fleurent e Ferland [FF94]; dentre as meta-heutríticas mais atuais temos a técnica conhecida como Sistema de Colônia de Formigas, que foi aplicado ao PQA por Maniezzo *et al.* [MCD94] e Gambardella *et al.* [GTD97].

1.4 Objetivos

O objetivo desse trabalho foi desenvolver uma heurística que tivesse um critério para gerar boas soluções iniciais e com um custo computacional baixo baseado no Teorema das Inversões de Rangel [Ran00]. Apresentamos o problema e algumas características relevantes a esse estudo: apresentação dos critérios de avaliação das soluções iniciais; criação de uma estrutura para gerar essas soluções, que é a parte que contém a maior contribuição teórica desse trabalho; e finalmente a apresentação da heurística desenvolvida para processamento em série e em paralelo, com alguns resultados empíricos.

No 2^o capítulo foi definido o Problema Quadrático de Alocação utilizando a formulação mais apropriada a esse estudo descrito em [KB57], e definido uma relaxação na forma do Problema de Alocação Linear (PAL). A relaxação se dá pelo fato do conjunto de soluções do PAL conter o das soluções quadráticas para uma mesma instância. A dificuldade de se trabalhar com a relaxação é reconhecer as soluções lineares que representam as soluções quadráticas, isto é, as soluções do PAL que são viáveis para o PQA. As vantagens da relaxação são vistas no capítulo seguinte, onde é mostrado um critério de avaliação da qualidade das soluções quadráticas baseado no Teorema das Inversões [Ran00], que será discutido no capítulo 3.

No capítulo 4 é apresentado uma forma de gerar soluções lineares que serão sempre factíveis ao problema quadrático, isto é, sempre existirão suas correspondentes quadráticas. Uma vantagem que vale ser mencionada é o baixo custo computacional para gerar essas soluções e a pequena quantidade de memória de armazenamento exigida.

No capítulo 5, apresentamos a heurística para processamento em série e em paralelo, mostrando sua eficiência e alguns resultados obtidos para exemplares conhecidos encontrados na QAPLIB [QAPLIB].

Ao final, apresentamos as conclusões e sugerimos trabalhos futuros que dão continuidade às idéias aqui discutidas.

Capítulo 2

Problema Quadrático de Alocação e sua relaxação

Neste Capítulo iremos discutir o **Problema Quadrático de Alocação** (PQA) e uma relaxação conhecida como **Problema de Alocação Linear** (PAL), onde vamos definir alguns conjuntos de soluções do PAL e mostrar suas propriedades particulares.

2.1 Problema Quadrático de Alocação - PQA

O Problema Quadrático de Alocação (PQA) inicialmente formulado por Koopmans e Beckmann, em 1957 [KB57], tem como objetivo a alocação de n facilidades à n localidades, duas a duas, com o objetivo de minimizar os custos. Aqui usaremos a formulação de Burkard e Stratmann [BS78] que apresentaram uma solução do PQA na forma de uma permutação. O PQA pode ser visto como a sobreposição de uma clique K_D valorada com as distâncias entre as localidades sobre outra clique K_F, de mesma dimensão, valorada com os fluxos, tendo a seguinte formulação matemática:

$$Z_\varphi = \underset{\varphi \in \Pi_n}{\text{Min}} \sum_{i,j}^{n} f_{ij} d_{\varphi(i)\varphi(j)}, \tag{2.1}$$

tal que $1 \leq i < j \leq n$ e Π_n é o conjunto de todas as permutações φ de $\Omega_n = \{1, ..., n\}$.

Assuma o seguinte exemplo para as cliques K_F e K_D de dimensão 4 na Figura 2.1,

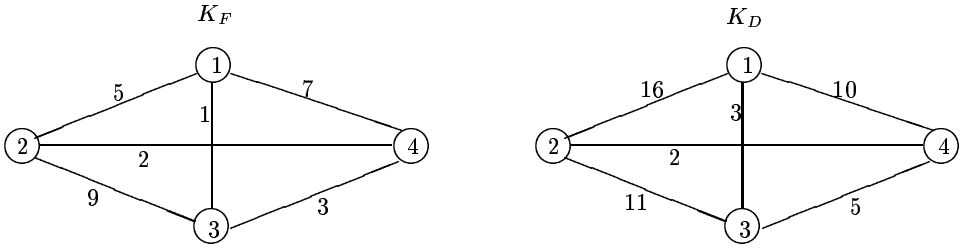

Figura 2.1: Cliques K_F e K_D

representadas pelas matrizes F e D de fluxo e custo respectivamente

$$F = \begin{bmatrix} 0 & 5 & 1 & 7 \\ 5 & 0 & 9 & 2 \\ 1 & 9 & 0 & 3 \\ 7 & 2 & 3 & 0 \end{bmatrix} \quad e \quad D = \begin{bmatrix} 0 & 16 & 3 & 10 \\ 16 & 0 & 11 & 2 \\ 3 & 11 & 0 & 5 \\ 10 & 2 & 5 & 0 \end{bmatrix}.$$

Admita uma sobreposição, como na Figura 2.2.

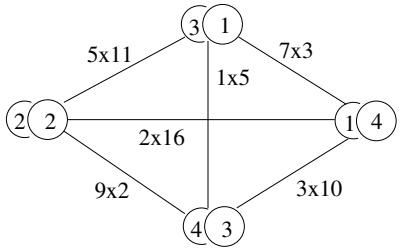

Figura 2.2: Sobreposição das cliques

Esta sobreposição é representada pela permutação $\varphi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 1 \end{pmatrix}$, que é obtida fixando a clique K_F e observando as posições dos nós da clique K_D. O custo desta sobreposição é definido explicitamente por

$$Z = 5 \times 11 + 1 \times 5 + 7 \times 3 + 9 \times 2 + 2 \times 16 + 3 \times 10 = 161$$

A estrutura das matrizes chama a atenção por suas características, simétricas e com a diagonal principal nula, o que facilita no armazenamento, podendo ser feito em vetores $F = (f_y)$ e $D = (d_y)$ de dimensão $N = C_{n,2}$, $y = 1, \ldots, N$ que são gerados pela ordem lexicográfica das arestas das cliques K_F e K_D. Para isto contamos com a bijeção ψ, que possui a seguinte definição:

$$\psi(i,j) = (i-1)n - \frac{i(i+1)}{2} + j; \tag{2.2}$$

tal que $\forall (i,j)$ com $1 \leq i < j \leq n$, tendo como imagem elementos y pertencentes ao conjunto $\Omega^N = \{1, 2, ..., N\}$.

Exemplo 2.1 *Para a Figura 2.1 temos que as matrizes F e D são representadas pelos vetores* $\mathbf{F} = (\ 5 \quad 1 \quad 7 \quad 9 \quad 2 \quad 3\ )$ *e* $\mathbf{D} = (\ 16 \quad 3 \quad 10 \quad 11 \quad 2 \quad 5\ )$.

2.2 Problema de Alocação Linear - PAL

Podemos definir um problema de Otimização onde estaremos buscando sobrepor as arestas sem a preocupação com os nós que as definem, com o objetivo de minimização dos custos. Problema este que é conhecido como Problema de Alocação Linear (PAL), com a seguinte formulação matemática.

$$Z_\xi = \underset{\xi \in \Pi_N}{\text{Min}} \sum_{i=1}^{N} f_i d_{\xi(i)}, \tag{2.3}$$

tal que Π_N é o conjunto de todas as permutações ξ de $\Omega^N = \{1, ..., N\}$. Este problema é considerado uma relaxação linear do problema quadrático devido ao fato de que o conjunto das soluções do problema quadrático está contido no das soluções do problema linear. Assim, partindo de uma permutação $\varphi \in \Pi_n$ que representa uma sobreposição de nós das cliques K_F sobre K_D,

$$\varphi = \begin{pmatrix} 1 & 2 & \dots & n \\ \varphi(1) & \varphi(2) & \dots & \varphi(n) \end{pmatrix}$$

encontra-se a respectiva permutação $\xi \in \Pi_N$ da sobreposição de arestas das mesmas cliques, através da bijeção ψ:

$$\xi = \begin{pmatrix} \psi(1,2) & \psi(1,3) & \dots & \psi(n-1,n) \\ \psi(\varphi(1),\varphi(2)) & \psi(\varphi(1),\varphi(3)) & \dots & \psi(\varphi(n-1),\varphi(n)). \end{pmatrix} \tag{2.4}$$

A recíproca nem sempre é válida. Como o número de soluções da alocação de nós é $n!$ e o número de soluções da alocação das arestas é $N!$, temos soluções do problema linear que não representam soluções quadráticas. Quando isto ocorre dizemos que a solução ξ é uma *solução não-viável* do PQA. Para ξ ser dita uma *solução viável* deve existir φ tal que seja possível escrever ξ de acordo com a expressão 2.4.

Para representar as possibilidades de alocação dos vetores **F** e **D**, considere a matriz $Q = F^T D$. Diferente do PQA(F,D), encontrar a solução ótima do PAL(Q) é bem fácil, intuitivamente associamos a aresta de maior fluxo à de menor distância, sucessivamente, até que a aresta de menor fluxo seja associada à de maior diatância. Para isto basta ordenar os vetores **F** e **D** de forma não-crescente e não-descrescente gerando respectivamente F^- e D^+, e calcular o produto interno $Z^* = \langle F^-, D^+ \rangle$.

Q	16	3	10	11	2	5
5	80	15	50	55	10	**25**
1	**16**	3	10	11	2	5
7	112	**21**	70	77	14	35
9	144	27	90	99	**18**	45
2	32	6	20	**22**	4	10
3	48	9	**30**	33	6	15

Q^*	2	3	5	10	11	16
9	**18**	27	45	90	99	144
7	14	**21**	35	70	77	112
5	10	15	**25**	50	55	80
3	6	9	15	**30**	33	48
2	4	6	10	20	**22**	32
1	2	3	5	10	11	**16**

Tabela 2.1: Matriz Q e Q^* para o exemplo proposto

Sendo $Q^* = (F^-)^T(D^+)$, temos assim a solução ótima Z^* coincidente com o traço da matriz Q^*, notada por $Tr(Q^*)$. Este é um limite inferior para as soluções do PQA(F,D), que raramente representa uma solução para o mesmo, e analogamente, obtemos um limite superior através da soma da diagonal secundária da matriz Q^*.

Exemplo 2.2 *Para exemplificar, assuma a ordenação do exemplo proposto anteriormente,*

$F^- = (\ 9 \quad 7 \quad 5 \quad 3 \quad 2 \quad 1\ )$ e $D^+ = (\ 2 \quad 3 \quad 5 \quad 10 \quad 11 \quad 16\ )$.

Nas tabelas das matrizes Q e Q^*, Tabela 2.1 em destaque, estão os elementos de Q representados pela permutação $\xi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 1 & 2 & 5 & 4 & 3 \end{pmatrix}$, que corresponde ao limite inferior da instância do PAL. Para simplificar a notação podemos utilizar apenas as imagens das permutações, sendo assim, $\xi = (\ 6 \quad 1 \quad 2 \quad 5 \quad 4 \quad 3\ )$ cujo custo é igual a $Z^* = \langle F^-, D^+ \rangle = 132$.

2.3 Correspondência entre PQA e PAL

Ao longo do texto usaremos $\xi \in \Pi_N$ para representar uma solução do PAL(Q) e $\rho \in \Pi_n$ para uma solução do PAL(Q^*), onde ρ representa uma permutação das colunas de Q^*. Como existe uma correspondência entre Q e Q^*, isto é, entre ξ e ρ, devemos encontrar uma maneira de descobrir a permutação ξ que corresponde a uma permutação ρ dada. Para isso, basta armazenar em permutações auxiliares

que denotamos por ϕ_F e ϕ_D, as trocas feitas nas posições dos vetores durante sua ordenação para F^- e D^+, respectivamente. Assim uma $\phi(i)$ é a posição em que o elemento está após a ordenação do i-ésimo elemento nos vetores originais. Da mesma forma temos que ϕ^{-1} é a posição em que o elemento estava antes da ordenação do i-ésimo elemento nos vetores originais.

As ϕ's correspondentes do exemplo são:

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

Tendo estas informações, a partir de uma ρ solução do PAL(Q^*), conseguimos a correspondente ξ solução do PAL(Q) pela seguinte relação:

$$\xi = \phi_D^{-1} \circ \rho \circ \phi_F. \tag{2.5}$$

Admita uma ρ como sendo o traço da matriz, $Tr(Q^*)$, que é a permutação identidade, então $\rho = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}$. Fazendo uso da relação, obtemos ξ, facilmente.

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

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

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

A ξ encontrada é, como esperado, a permutação apresentada no exemplo 2.2.

É claro que se tivermos de posse do ξ e interessados em encontrar a ρ correspondente, basta aplicar o inverso da relação 2.5, que é dada por:

$$\rho = \phi_D \circ \xi \circ \phi_F^{-1}. \tag{2.6}$$

Já sabemos que o universo das soluções lineares é muito maior que o das soluções quadráticas. A seguir, definiremos uma forma de se reconhecer a factibilidade de uma solução linear em relação ao problema quadrático , isto é, dada solução representada pela permutação $\rho \in \Pi_N$ queremos encontrar a solução quadrática representada pela $\varphi \in \Pi_n$ correspondente a ela.

Definição 2.1 (Viabilidade) *Sejam $\rho \in \Pi_N$, o vetor $S_\rho = (f_1^- d^+_{\rho(1)} \dots f_N^- d^+_{\rho(N)})$ e um par de cliques K_F e K_D, valoradas pelas coordenadas que determinam F^- e D^+. Se existe uma permutação $\varphi \in \Pi_n$ de vértices de uma clique sobre outra, tal que*

$$f_r^- d^+_{\rho(r)} = f_{\phi_F^{-1}(r)} d_{\phi_D^{-1}(\rho(r))} = f_p d_{\xi(p)} = f_{\psi^{-1}(p)} d_{\psi^{-1}(\xi(p))} = f_{ij} d_{kl}, \tag{2.7}$$

para $r = 1, ..., N$; $\varphi(i) = k$ e $\varphi(j) = l$; $i, j, k, l = 1, ..., n$; dizemos que ρ é uma solução **viável** *para o PQA(F,D), definido pelo par de cliques.*

A aplicação de ϕ_F^{-1} e ϕ_D^{-1} na expressão 2.7 é para descobrir as posições originais antes das ordenações dos vetores **F** e **D**.

A verificação da viabilidade é feita por um algoritmo que tenta associar a sobreposição das arestas à sobreposição dos nós. Para isto utiliza-se de duas listas $ListaIJ$ e $ListaKL$ que armazenam os nós que dão origem as arestas. A $ListaIJ$ se refere ao domínio e a $ListaKL$, à imagem. A cada sobreposição de arestas temos duas possíveis sobreposições de nós.

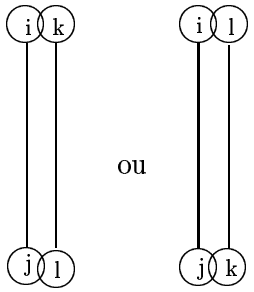

Figura 2.3: Combinação de vértices

Isto pode ser representado matematicamente por:

$$[\varphi(i) = k \wedge \varphi(j) = l] \vee [\varphi(i) = l \wedge \varphi(j) = k]; \forall i, j, k, l \in \{1, \dots, n\}.$$

Para uma permutação ser viável a combinação de arestas das listas $ListaIJ$ e $ListaKL$ deve ser compatível $\forall i = 1, \dots, N$ a uma sobreposição de vértices definido por uma $\varphi \in \Pi_n$.

O algoritmo aqui apresentado foi desenvolvido por Rangel [Ran00] denominado SolViável.

Algoritmo 1: *SolViável*

1: $\xi = \phi_D^{-1} \circ \rho \circ \phi_F$
2: **Entrada:** ρ, ϕ_F e ϕ_D^{-1}
3: **for** t = 1, ... , N **do**
4: $ListaIJ[t] \leftarrow \psi^{-1}(t)$
5: $ListaKL[t] \leftarrow \psi^{-1}(\xi(t))$
6: **end for**
7: **if** $\exists$ p $\in$ 1, ... , n tal que $\varphi(1) = p$ para as (n - 1) primeiras combinações **then**
8: $\varphi(1) \leftarrow$p
9: **for** i = 2, ... , n **do**
10: $[\varphi(i) = l \neq p] \vee [\varphi(i) = k \neq p]$ para $ListaKL[i-1]$
11: **end for**
12: **if** φ construída possui todas as N combinações compatíveis **then**
13: ρ é viável
14: **else**
15: ρ não é viável
16: **end if**
17: **else**
18: ρ não é viável
19: **end if**

Nos exemplos a seguir ilustraremos o algoritmo SolViável com a instância apresentada anteriormente onde temos $n = 4$ e $N = C_{n,2} = 6$. Para facilitar a notação no restante do texto, doravante utilizaremos somente a imagem das permutações para representá-las.

Exemplo 2.3 *Admita a permutação* $\rho_1 = (\ 1\ \ 3\ \ 2\ \ 5\ \ 6\ \ 4\ ) \in \Pi_6$.

O primeiro passo do algoritmo é a construção da ξ, que é dado por:
$\xi_1 = \phi_D^{-1} \circ \rho_1 \circ \phi_F$,
$\xi_1 = (\ 5\ \ 2\ \ 6\ \ 3\ \ 4\ \ 1\ ) \circ (\ 1\ \ 3\ \ 2\ \ 5\ \ 6\ \ 4\ ) \circ (\ 3\ \ 6\ \ 2\ \ 1\ \ 5\ \ 4\ )$,
$\xi_1 = (\ 2\ \ 3\ \ 6\ \ 5\ \ 1\ \ 4\ )$.

Na Tabela 2.2 apresentamos a aplicação da ψ^{-1} para construção das listas

$ListaIJ$ e $ListaKL$ bem como as combinações de vértices entre eles.

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

Tabela 2.2: Tabela do exemplo 2.3

Note que por esta Tabela não é possível definir uma $\varphi(1)$ que seja compatível nas (n - 1) primeiras linhas, logo dizemos que ρ_1 é uma permutação **não-viável**.

Exemplo 2.4 *Seja* $\rho_2 = (\ 5\ \ 1\ \ 4\ \ 2\ \ 6\ \ 3\ ) \in \Pi_6$

Para este caso temos que $\xi_2 = (\ 3\ \ 6\ \ 5\ \ 4\ \ 1\ \ 2\ )$, as construções das listas e combinações são representadas na Tabela 2.3.

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

Tabela 2.3: Tabela do exemplo 2.4

Observe que na Tabela 2.3 conseguimos definir $\varphi = (\ 4\ \ 1\ \ 3\ \ 2\ )$ nas $(n-1)$ primeiras linhas, porém na quarta e sexta linhas encontramos incompatibilidades fazendo com que ρ_2, mesmo sendo possível montar uma φ, seja uma permutação **não viável**.

Exemplo 2.5 *Seja* $\rho_3 = (\ 2\ \ 1\ \ 4\ \ 5\ \ 6\ \ 3\ ) \in \Pi_6$.

A correspondente do PAL(Q) é $\xi_3 = (\ 3\ \ 6\ \ 5\ \ 2\ \ 1\ \ 4\ )$, as listas e combinações dos vértices estão na Tabela 2.4.

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

Tabela 2.4: Tabela do exemplo 2.5

Note que nas $(n-1)$ primeiras linhas construímos $\varphi = (\ 4\ \ 1\ \ 3\ \ 2\ )$, a mesma do exemplo 2.4, e como no restante da Tabela 2.4 não é apresentado nenhuma incompatibilidade, ρ_3 é dita uma permutação **viável** para o PQA.

2.4 Pseudo-Viável

Para facilitar o entendimento e a escrita deste texto faremos a seguir uma definição dividindo a permutação $\rho \in \Pi_N$ em duas partes.

Definição 2.2 *Dado um número natural n, considere $N = C_{n,2}$ e $\rho \in \Pi_N$ uma permutação. Chamaremos os $(n-1)$ primeiros elementos da ρ de* **cabeça** *e os $N-(n-1)$ restante dos elementos de* **cauda**.

Como vimos, os exemplos 2.3 e 2.4 mostram permutações ρ_1 e ρ_2 ambas não-viáveis, porém por motivos diferentes. Em 2.3 a resposta da condicional da linha 7 do algoritmo é falsa sendo portanto, ρ_1 não-viável. No exemplo 2.4 foi possível construir uma permutação de vértices $\varphi \in \Pi_n$ usando as $(n-1)$ primeiras linhas, porém nas linhas restantes foi detectado incompatibilidade fazendo com que a solução ρ_2 do problema linear seja considerada não-viável para o PQA dado que a condicional da linha 12 é falsa. Estes fatos nos levam a definir um novo conjunto de soluções

consideradas interessantes para este estudo. Sendo assim, definiremos formalmente o caso do exemplo 2.4, que será chamado de **pseudo-viável**.

Definição 2.3 *Seja $\rho \in \Pi_N$ solução do PAL(Q*). Se for possível construir uma permutação $\varphi \in \Pi_n$ utilizando a cabeça da permutação ρ, diz-se que ρ é* **Pseudo-Viável** *para o PQA(F,D).*

Com esta definição aumentamos consideravelmente nossa capacidade de busca.

Propriedade 2.1 *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 de forma simples pelo número de soluções do PQA(F,D). Seja $\rho \in \Pi_N$ uma solução viável para o PQA(F,D) 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 finais (cauda) gerando desta forma $[N-(n-1)]!$ pseudo-viáveis. Pelo princípio fundamental da contagem temos que o número de pseudo-viáveis é $n![N-(n-1)]!$. □

Seguindo esta definição temos que os exemplos 2.4 e 2.5 são ambos pseudo-viáveis gerando a permutação de vértices $\varphi = (\ 4\ \ 1\ \ 3\ \ 2\ )$ onde a cabeça da permutação das arestas dada por $\xi(1)$, $\xi(2)$ e $\xi(3)$ são respectivamente 3, 6 e 5. Como vimos para cada permutação viável temos $[N-(n-1)]!$ permutações pseudo-viáveis. No nosso exemplo com $n = 4$ o número de pseudo-viáveis para cada solução viável é 6, sendo esses elementos justamente as permutações dos elementos da cauda da permutação viável.

Para o exemplo citado destacamos em negrito os elementos da cabeça da permutação e com sublinhado os elementos da cauda:

1. Permutação Viável
   $\xi_1 = (\ 3\ \ 6\ \ 5\ \ 2\ \ 1\ \ 4\ )$,
   $\rho_1 = \phi_D \circ \xi \circ \phi_F^{-1}$,
   $\rho_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 2 & 4 & 5 & 1 & 3 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \mathbf{3} & \mathbf{6} & \mathbf{5} & 2 & 1 & 4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & \mathbf{3} & \mathbf{1} & 6 & 5 & \mathbf{2} \end{pmatrix}$,
   $\rho_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & \mathbf{1} & \mathbf{4} & 5 & 6 & \mathbf{3} \end{pmatrix}$.

2. $\xi_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \mathbf{3} & \mathbf{6} & \mathbf{5} & 2 & 4 & 1 \end{pmatrix}$,
$\rho_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & \mathbf{1} & \mathbf{4} & 6 & 5 & \mathbf{3} \end{pmatrix}$.

3. $\xi_3 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \mathbf{3} & \mathbf{6} & \mathbf{5} & 4 & 1 & 2 \end{pmatrix}$,
$\rho_3 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & \mathbf{1} & \mathbf{4} & 2 & 6 & \mathbf{3} \end{pmatrix}$.

4. $\xi_4 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \mathbf{3} & \mathbf{6} & \mathbf{5} & 4 & 2 & 1 \end{pmatrix}$,
$\rho_4 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & \mathbf{1} & \mathbf{4} & 6 & 2 & \mathbf{3} \end{pmatrix}$.

5. $\xi_5 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \mathbf{3} & \mathbf{6} & \mathbf{5} & 1 & 2 & 4 \end{pmatrix}$,
$\rho_5 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & \mathbf{1} & \mathbf{4} & 2 & 5 & \mathbf{3} \end{pmatrix}$.

6. $\xi_6 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \mathbf{3} & \mathbf{6} & \mathbf{5} & 1 & 4 & 2 \end{pmatrix}$,
$\rho_6 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & \mathbf{1} & \mathbf{4} & 5 & 2 & \mathbf{3} \end{pmatrix}$.

Assim é possível dividir o conjunto de soluções lineares em três conjuntos da seguinte forma: **Não-Viável**, **Pseudo-Viável** e **Viável**. Sendo que os conjuntos Não-Viável e Pseudo-Viável são disjuntos e o conjunto das soluções Viáveis está contido no conjunto das soluções Pseudo-Viáveis.

Como nosso objetivo é o PQA iremos considerar para esse estudo todo o conjunto das Pseudo-Viáveis. Desta forma aumentamos consideravelmente a possibilidade de encontrar soluções lineares capazes de gerar soluções quadráticas. O universo de busca, passa de um espaço de ordem $\frac{n!}{N!}$ para $\frac{n![N-(n-1)]!}{N!}$. O primeiro quociente dá a razão entre o número de soluções viáveis e o de soluções lineares. A segunda relaç ao quantifica a razão entre o número de pseudo-viáveis e o número de soluções lineares.

Capítulo 3

A base teórica

teorema

Neste Capítulo apresentamos um resultado exibido em [Ran00] e [RA03] conhecido como Teorema das Inversões, que tem por objetivo afirmar que as soluções lineares do PAL(Q^*) que fazem parte do conjunto das livremente comparáveis tem seu custo Z_ρ (onde $Z_{\rho_i} = \langle F^-, \rho_i(D^+)\rangle$) associado ao número de inversões da permutação ρ. Baseado nesse teorema podemos gerar soluções de boa qualidade para a heurística que será apresentada no capítulo 5.

As demonstrações dos teoremas, lemas e proposições desse capítulo estão apresentadas no anexo. Pois optamos por uma forma de apresentação que conduzirá o leitor ao entendimento da teoria.

3.1 Teorema da Ordenação Parcial Livre

Sejam F^- e D^+ vetores de ordem N tal que F^- é a ordenação não-crescente do vetor de fluxo F e D^+ é a ordenação não-decrescente do vetor distância D. Além disso, considere $\rho(D^+)$ como a imagem da aplicação ρ no vetor D^+ e os produtos escalares $Z_{\rho_1} = \langle F^-, \rho_1(D^+)\rangle$ e $Z_{\rho_2} = \langle F^-, \rho_2(D^+)\rangle$ que representam os custos de duas soluções do PAL, onde $\rho_1, \rho_2 \in \Pi_N$ permutações do conjunto $\Omega^N = \{1, \ldots, N\}$.

Fazendo $Z_{\rho_1} - Z_{\rho_2}$ obtemos:

$$Z_{\rho_1} - Z_{\rho_2} = \sum_{t=1}^{N} f_t^- d^+_{\rho_1(t)} - \sum_{t=1}^{N} f_t^- d^+_{\rho_2(t)} = \sum_{t=1}^{N} f_t^- (d^+_{\rho_1(t)} - d^+_{\rho_2(t)}). \qquad (3.1)$$

Com relação aos valores $\rho_1(t)$ e $\rho_2(t)$ podemos particionar o conjunto $\Omega_N = \{1, \ldots, N\}$ da seguinte forma:

- $P_N = \{t \in \Omega^N / \rho_1(t) - \rho_2(t) < 0\}$, termos não-positivos em 3.1;
- $P_P = \{t' \in \Omega^N / \rho_1(t') - \rho_2(t') > 0\}$, termos não-negativos em 3.1;
- $P_O = \{t'' \in \Omega^N / \rho_1(t'') - \rho_2(t'') = 0\}$, termos nulos em 3.1;

Sendo possível eliminar os elementos de P_O em 3.1, pois $d_{\rho_1(t'')} - d_{\rho_2(t'')} = 0$, reduzimos a equação 3.1 à

$$Z_{\rho_1} - Z_{\rho_2} = \sum_{t \in P_N} f_t^- (d^+_{\rho_1(t)} - d^+_{\rho_2(t)}) + \sum_{t' \in P_P} f_t^- (d^+_{\rho_1(t')} - d^+_{\rho_2(t')}). \tag{3.2}$$

É chamado de **Partição Canônica** a inserção de termos simétricos de cada fator-diferença em cada parcela da equação 3.2 de modo que não haja alteração no seu resultado, obtendo assim:

$$\begin{aligned} Z_{\rho_1} - Z_{\rho_2} = \sum_{t \in P_N} f_t^- \sum_{i=0}^{\rho_2(t)-\rho_1(t)-1} (d^+_{\rho_1(t)+i} - d^+_{\rho_1(t)+i+1}) + \\ \sum_{t' \in P_P} f_{t'}^- \sum_{j=0}^{\rho_1(t')-\rho_2(t')-1} (d^+_{\rho_1(t')-j} - d^+_{\rho_1(t')-j-1}). \end{aligned} \tag{3.3}$$

Exemplo 3.1 *Considere* $\rho_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 6 & 5 & 2 & 4 & 3 \end{pmatrix}$ *e* $\rho_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 2 & 5 & 1 & 4 & 6 \end{pmatrix}$.

Analisando os valores de $\rho_1(t)$ e $\rho_2(t)$ para $t \in \{1, \ldots, 6\}$, observamos que:

- $\rho_1(1) - \rho_2(1) = 1 - 3 < 0$;
- $\rho_1(6) - \rho_2(6) = 3 - 6 < 0$;
- $\rho_1(2) - \rho_2(2) = 6 - 2 > 0$;
- $\rho_1(4) - \rho_2(4) = 2 - 1 > 0$;

- $\rho_1(3) - \rho_2(3) = 5 - 5 = 0$;
- $\rho_1(5) - \rho_2(5) = 4 - 4 = 0$.

Então temos a seguinte partição das permutações $P_N = \{1, 6\}$, $P_P = \{2, 4\}$ e $P_O = \{3, 5\}$. Representando na equação 3.2 obtemos a seguinte expressão

$$\begin{aligned} Z_{\rho_1} - Z_{\rho_2} =& f_1^-(d^+_{\rho_1(1)} - d^+_{\rho_2(1)}) + f_6^-(d^+_{\rho_1(6)} - d^+_{\rho_2(6)}) + \\ & f_2^-(d^+_{\rho_1(2)} - d^+_{\rho_2(2)}) + f_4^-(d^+_{\rho_1(4)} - d^+_{\rho_2(4)}). \end{aligned} \tag{3.4}$$

Substituindo os valores de $\rho_1(t)$ e $\rho_2(t)$,

$$\begin{aligned} Z_{\rho_1} - Z_{\rho_2} =& f_1^-(d_1^+ - d_3^+) + f_6^-(d_3^+ - d_6^+) + \\ & f_2^-(d_6^+ - d_2^+) + f_4^-(d_2^+ - d_1^+), \end{aligned} \tag{3.5}$$

e inserindo os termos simétricos de cada fator-diferença em cada parcela,

$$\begin{aligned} Z_{\rho_1} - Z_{\rho_2} =& f_1^-(d_1^+ - d_2^+ + d_2^+ - d_3^+) + \\ & f_6^-(d_3^+ - d_4^+ + d_4^+ - d_5^+ + d_5^+ - d_6^+) + \\ & f_2^-(d_6^+ - d_5^+ + d_5^+ - d_4^+ + d_4^+ - d_3^+ + d_3^+ - d_2^+) + \\ & f_4^-(d_2^+ - d_1^+). \end{aligned} \tag{3.6}$$

Conseguimos assim a Partição Canônica para ρ_1 e ρ_2 do exemplo 3.1.

Lema 3.1 *Na partição canônica de $Z_{\rho_1} - Z_{\rho_2}$ temos que*

$$-\sum_{t \in P_N} \sum_{i=0}^{\rho_2(t)-\rho_1(t)-1} (d^+_{\rho_1(t)+i} - d^+_{\rho_1(t)+i+1}) = \sum_{t' \in P_P} \sum_{j=0}^{\rho_1(t')-\rho_2(t')-1} (d^+_{\rho_1(t')-j} - d^+_{\rho_2(t')-j-1}).$$

Além disso, o valor absoluto de cada fator diferença do primeiro membro corresponde a um fator-diferença do segundo membro e reciprocamente.

A prova desse lema, explora o fato de

$$\sum_{t\in\Omega_N}(d^+_{\rho_1(t)} - d^+_{\rho_2(t)}) = 0.$$

Daí particionamos os índices $t \in \Omega_N$ nos subconjuntos P_N, P_0 e P_P tal que a união deles seja Ω_N. Como $d_{\rho_1(t)} - d_{\rho_2(t)} = 0$ para $t \in P_0$, podemos igualar as somatórias com índices em P_N e P_P.

Introduzindo os termos simétricos da Partição Canônica, o valor da expressão não se altera e assim atingimos a equação objetivo do Lema 3.1.

Para entender a segunda parte do Lema, devemos analisar a expressão objetivo do mesmo. Observamos que $\forall t \in P_N$ e $i \in \{0,\ldots,\rho_2(t) - \rho_1(t) - 1\}$ existe um $t' \in P_P$ e $j \in \{0,\ldots,\rho_1(t') - \rho_2(t') - 1\}$ tal que

$$\rho_1(t) + i = \rho_1(t') - j - 1 \text{ e } \rho_1(t) + i + 1 = \rho_1(t') - j.$$

Para melhor visualização dessa segunda parte do Lema, faremos uso do exemplo 3.1, dividindo os termos da Partição Canônica pelos índices na ordem em que aparecem:
(i) $\quad -[(d^+_1 - d^+_2) + (d^+_2 - d^+_3) + (d^+_3 - d^+_4) + (d^+_4 - d^+_5) + (d^+_5 - d^+_6)] \to t = 1, 6;$
(ii) $\quad (d^+_6 - d^+_5) + (d^+_5 - d^+_4) + (d^+_4 - d^+_3) + (d^+_3 - d^+_2) + (d^+_2 - d^+_1) \to t' = 2, 4;$
$-(i) = (ii).$

Sendo assim ilustramos

$$-(d^+_{\rho(t)+i} - d^+_{\rho(t)+i+1}) = (d^+_{\rho(t')-j} - d^+_{\rho(t')-j-1}).$$

Dada a equação 3.3 da partição canônica, sejam as seguintes definições: para as parcelas da partição canônica cujos valores são não-positivos temos $N^i_t = f^-_t(d^+_{\rho_1(t)+i} - d^+_{\rho_1(t)+i+1})$ onde $t \in P_N$ e $i = 0,\ldots,\rho_2(t) - \rho_1(t) - 1$ e para as parcelas da partição canônica cujos valores são não-negativos temos $P^j_{t'} = f^-_{t'}(d^+_{\rho_1(t')-j} - d^+_{\rho_2(t')-j-1})$ onde $t' \in P_P$ e $j = 0,\ldots,\rho_1(t') - \rho_2(t') - 1$.

Pelo Lema 3.1 para todo elemento N^i_t existe um correspondente $P^j_{t'}$, assim podemos definir uma bijeção β tal que $\beta(N^i_t) = P^j_{t'}$; para todo $t \in P_N$; $i = 0,\ldots,\rho_2(t) - \rho_1(t) - 1$; $t' \in P_P$; $j = 0,\ldots,\rho_1(t') - \rho_2(t') - 1$. Através da construção de β identificamos os pares ordenados (t, t') com $t \in P_N$ e $t' \in P_P$, e dado que os fatores-diferença da expressão 3.3 são simétricos, podemos colocá-los em evidência obtendo a seguinte expressão:

$$Z_{\rho_1} - Z_{\rho_2} = \sum_{t \in P_N} \sum_{i=0}^{\rho_2(t)-\rho_1(t)-1} (f_t^- - f_{t'}^-)(d^+_{\rho_1(t)+i} - d^+_{\rho_1(t)+i+1}). \tag{3.7}$$

Exemplo 3.2 *Com* $\rho_1 = (\ 1\ \ 6\ \ 5\ \ 2\ \ 4\ \ 3\ )$ *e* $\rho_2 = (\ 3\ \ 2\ \ 5\ \ 1\ \ 4\ \ 6\ )$ *do exemplo 3.1 vimos que* $P_N = \{1, 6\}$ *e* $P_P = \{2, 4\}$. *Construímos a seguinte bijeção* β. *Que está na Tabela 3.1*

N_t^i	$P_{t'}^j$	β	(t, t')
$N_1^0 = f_1^-(d_1^+ - d_2^+)$	$P_2^0 = f_2^-(d_6^+ - d_5^+)$	$\beta(N_1^0) = P_4^0$	$(1, 4)$
$N_1^1 = f_1^-(d_2^+ - d_3^+)$	$P_2^1 = f_2^-(d_5^+ - d_4^+)$	$\beta(N_1^1) = P_2^3$	$(1, 2)$
$N_6^0 = f_6^-(d_3^+ - d_4^+)$	$P_2^2 = f_2^-(d_4^+ - d_3^+)$	$\beta(N_6^0) = P_2^2$	$(6, 2)$
$N_6^1 = f_6^-(d_4^+ - d_5^+)$	$P_2^3 = f_2^-(d_3^+ - d_2^+)$	$\beta(N_6^1) = P_2^1$	$(6, 2)$
$N_6^2 = f_6^-(d_5^+ - d_6^+)$	$P_4^0 = f_4^-(d_2^+ - d_1^+)$	$\beta(N_6^2) = P_2^0$	$(6, 2)$

Tabela 3.1: Construção da bijeção

Note que na grande maioria das vezes a bijeção β não é única pois pode existir repetição do fator-diferença $(d_k^+ - d_l^+)$. Nesse caso poderíamos optar por outra construção da correspondência entre $N_t^i \longrightarrow P_{t'}^j$.

Seja $C(Z_\rho) = \{Z_\rho = \langle F^-, \rho(D^+)\rangle / \rho \in \Pi_N\}$ conjunto dos custos de um PAL(Q^*). Definimos que Z_{ρ_1} e Z_{ρ_2} são **livremente comparáveis** quando $Z_{\rho_1} \leq Z_{\rho_2}$ ou $Z_{\rho_2} \leq Z_{\rho_1}$ independentemente das coordenadas dos vetores F^- e D^+ (notaremos por $Z_{\rho_1} \leq_l Z_{\rho_2}$ ou $Z_{\rho_2} \leq_l Z_{\rho_1}$). O teorema a seguir fornece instrumento para descobrir se dois custos de um PAL(Q^*) são livremente comparáveis.

Teorema 3.1 Teorema da Ordenação Parcial Livre - TOPL: *Temos* $Z_{\rho_1} \leq_l Z_{\rho_2}$ $(Z_{\rho_2} \leq_l Z_{\rho_1})$, *se e somente se, existir uma bijeção* β *tal que* $\beta(N_t^i) = P_{t'}^j$; $t \in P_N$; $t' \in P_P$; $i = 0, \ldots, \rho_2(t) - \rho_1(t) - 1$ *e* $j = 0, \ldots, \rho_1(t') - \rho_2(t') - 1$ *capaz de induzir o par ordenado* (t, t'), *com* $t < t'(t' < t)$, $\forall t \in P_N$ *e* $\forall t' \in P_P$.

A prova deste teorema explora o fato dos vetores F^- e D^+ serem ordenados, possibilitando afirmar que $Z_{\rho_1} \leq_l Z_{\rho_2}$ apenas observando os índices $t \in P_N$ e $t' \in P_P$. Maiores detalhes da demonstração está no anexo.

Assim pelo Teorema da Ordenação Parcial Livre podemos estabelecer uma relação de ordem (" $\leq_l$ ") entre o elementos do conjunto $C(Z_\rho)$, produtos escalares, resultando em um *poset* $(C(Z_\rho), \leq_l)$.

Para estabelecer uma relação entre uma permutação e seu custo seja a seguinte função f que associa cada elemento do conjunto Π_N a um do conjunto $C(Z_\rho)$

$$\begin{aligned} f :&\Pi_N \rightarrow C(Z_\rho) \\ &\rho \rightarrow Z_\rho = \langle F^-, \rho(D^+) \rangle. \end{aligned} \tag{3.8}$$

Através desta função f podemos estabelecer uma relação de ordem também para o conjunto Π_N, dada pela relação:

$$\rho_i \leq_l \rho_j \Leftrightarrow f(\rho_i) \leq_l f(\rho_j),$$

tal que $f(\rho) = Z_\rho$.

Acabamos de definir um *poset* $(\Pi_N, \leq_l)$ compatível com o *poset* $(C(Z_\rho), \leq_l)$. Assim para duas permutações serem chamadas de livremente comparáveis, estas devem pertencer ao *poset* $(\Pi_N, \leq_l)$. O grafo de comparabilidade [Ran00] do *poset* $(\Pi_N, \leq_l)$ é o grafo $G_{Liv} = (\Pi_N, M)$, onde $M = \{(\rho_i, \rho_j) \in \Pi_N \times \Pi_N / \rho_i \leq_l \rho_j, i, j = 1 \ldots N!\}$.

3.2 Teorema das Inversões

Uma inversão em $\rho \in \Pi_N$ é caracterizada pelo par $(\rho(i), \rho(j))$ tal que $\rho(j) < \rho(i)$ com $i < j$; $i, j \in \Omega^N$. Para contar as inversões de uma permutação ρ, considere $\Im(\rho)$ como sendo o conjunto dos pares das inversões e $|\Im(\rho)|$, a cardinalidade de $\Im(\rho)$, como o número de inversões de ρ.

Exemplo 3.3 *Assuma a permutação* $\rho = (\ 4\ \ 1\ \ 2\ \ 3\ \ 6\ \ 5\ )$, *temos que* $\Im(\rho) = \{(4,1),(4,2),(4,3),(6,5)\}$ *e* $|\Im(\rho)| = 4$.

O grafo $G_{inv} = (\Pi_N, W)$, tal que $W = \{(\rho_r, \rho_s) \in \Pi_n \times \Pi_N / \rho_s$ é obtido de ρ_r pela troca de pares de elementos adjacentes com $|\Im(\rho_s)| - |\Im(\rho_r)| = 1\}$, é dividido por níveis que indicam o número de permutações. Tal grafo é iniciado pela permutação

identidade no nível 0 e o último nível é dado pela permutação reversa, $rev = (N, N-1, N-2, \ldots, 1)$, que possui exatamente $\frac{N(N-1)}{2} + 1$ inversões. Este grafo definido será chamado de **Grafo das Inversões**, conhecido como Permutaedro, Vernet et al. [VRA95]. Este grafo é o **Grafo de Comparabilidade** do *poset* $(\Pi_N, <)$ onde a relação $\rho_r < \rho_s$ é válida se $(\rho_r, \rho_s) \in W$. Vale salientar que os nós do grafo são todas as permutações de $\Omega^N = \{1, 2, \ldots, N\}$, por tanto todo algoritmo de construção desse grafo é ordem $O(N!)$, o que torna inviável sua construção para N grande.

Lema 3.2 *Sejam* $\rho_1, \rho_2 \in \Pi_N$. *Se* $(\rho_1, \rho_2) \in W \Rightarrow Z_{\rho_1} \leq_l Z_{\rho_2}$.

O Lema 3.2 nos afirma que se fizermos uma troca de pares de elementos adjacentes em ρ_1 e gerarmos a permutação ρ_2, então essas permutações são livremente comparáveis.

Como conseqüência direta do Lema 3.2 temos que o Grafo de Comparabilidade G_{inv} do *poset* $(\Pi_N, <)$ esta contido em G_{Liv}, grafo de comparabilidade do *poset*$(\Pi_N, \leq_l)$ que representa todas as permutações livremente comparáveis.

Para exemplificar mostraremos o grafo G_{inv}, na Figura 3.1, para $N = 4$, mas tendo em mente que este grafo não representa nenhuma instância do PQA pois não existe n pertencente ao conjunto do números naturais tal que $C_{n,2} = 4$. Se tentarmos representar graficamente um PQA com $n = 4$ teríamos $N = 6$ o que nos dá um grafo de 720 nós, tornando inviável sua representação gráfica.

Teorema 3.2 *Sejam* ρ_i *e* ρ_j *vértices do grafo* G_{inv} *ligados por um caminho de arcos em* W. *Tem-se* $|\Im(\rho_i)| < |\Im(\rho_j)|$, *se o somente se,* $Z_{\rho_i} \leq_l Z_{\rho_j}$.

A demonstração do Teorema 3.2 pode ser feita pelo método da indução. Para vértices pertencentes a W que possuem um nível de diferença, já foi provado pelo Lema 3.2. Basta supor verdadeiro para $m - 1$ níveis e provar, por transitividade, que é válido para os m níveis, quando considerado m o comprimento do caminho ρ_i entre e ρ_j.

Com esse novo conjunto de permutações livremente comparáveis dado pelo Teorema 3.2 define-se um novo grafo que é o fecho transitivo de $G_{inv} = (\Pi_N, W)$ e será

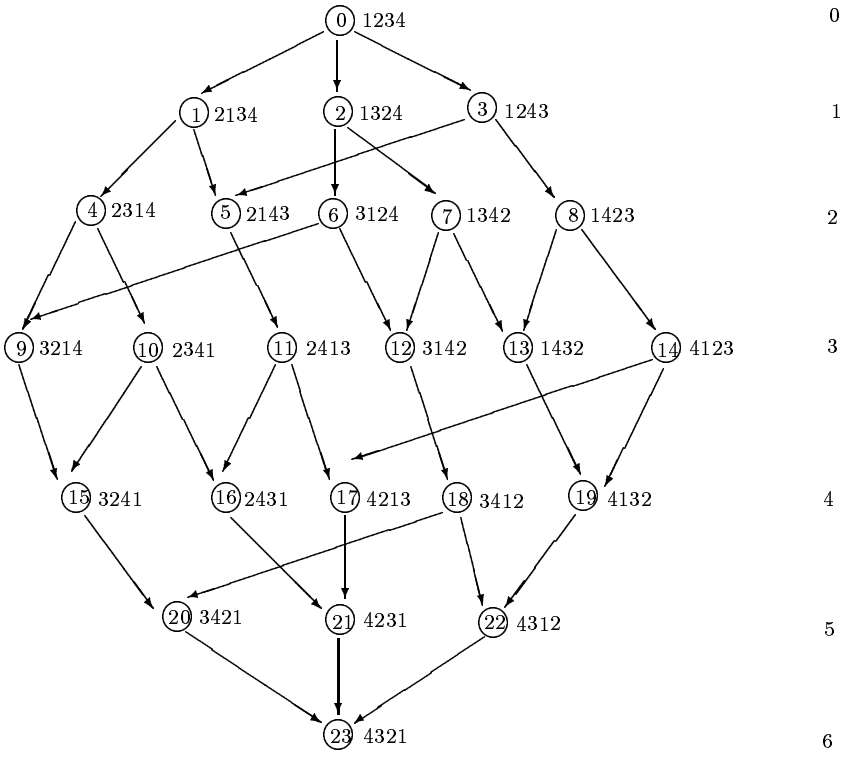

Figura 3.1: Grafo das Inversões G_{inv} para $N = 4$

notado por $\widehat{G_{inv}} = (\Pi_N, \widehat{W})$, onde $\widehat{W}$ é a união de W com os demais arcos resultantes da transitividade, ou seja, se existe um caminho entre ρ_i e ρ_j então existirá o arco (ρ_i, ρ_j).

O objetivo agora é estender esses resultados para todas as permutações livremente comparáveis, iniciando com as permutações de um nível de diferença no grafo. Para isso defina o conjunto de arcos tal que $W' = \{(\rho_i, \rho_j)/Z_{\rho_i} \leq_l Z_{\rho_j}$ e $|\Im(\rho_i)| - |\Im(\rho_j)| = 1\}$, definindo o grafo $G'_{inv} = (\Pi_N, W')$. Note que $W \subseteq W'$ portanto, temos que $G_{inv} \subseteq G'_{inv} \subseteq G_{Liv}$.

Lema 3.3 *Sejam ρ_1 e ρ_2 duas permutações tais que $||\Im(\rho_1)| - |\Im(\rho_2)|| = 1$ então Z_{ρ_1} e Z_{ρ_2} são livremente comparáveis, se e somente se, ρ_1 e ρ_2 diferem entre si de dois elementos.*

Observe que o Lema 3.3 inclui os arcos $(\rho_1, \rho_2) \in W$ resultantes de trocas de elementos adjacentes da permutação. A demonstração da condição necessária do Lema, pode ser feita por contradição utilizando duas permutações que diferem de mais de dois elementos, e a suficiente, com a aplicação da expressão 3.2 nos conjuntos unitários de P_N e P_P.

A Figura 3.2 ilustra o grafo G'_{inv} para $N = 4$. Aqui destacamos as arestas que pertencem ao conjunto $W' - W$

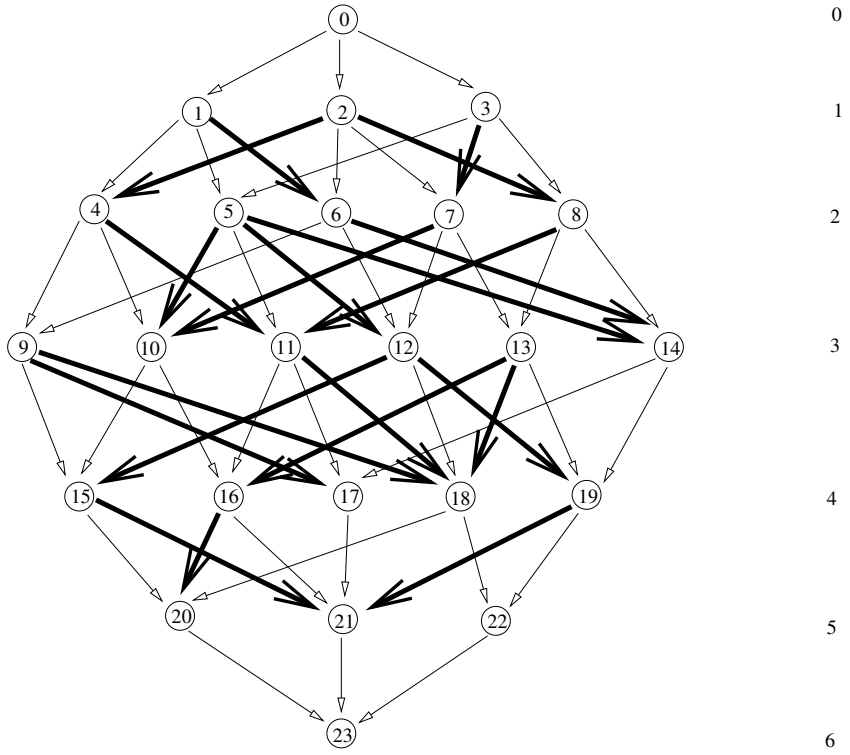

Figura 3.2: Grafo das Inversões G'_{inv} para $N = 4$

Lema 3.4 *Sejam ρ_1 e $\rho_2 \in \Pi_N$. Se $(\rho_1, \rho_2) \in W'$ então $Z_{\rho_1} \leq_l Z_{\rho_2}$.*

Este lema é semelhante ao Lema 3.2 e no anexo está demonstrado por contradição.

Teorema 3.3 *Sejam ρ_i e ρ_j vértices do grafo G'_{inv} ligados por um caminho de arcos em W'. Tem-se $|\Im(\rho_i)| < |\Im(\rho_j)|$, se e somente se, $Z_{\rho_i} \leq_l Z_{\rho_j}$.*

A demonstração deste lema é análoga ao do Teorema 3.2, considerando neste caso que os arcos pertencem a W'.

Assim como foi definido o grafo $\widehat{G_{inv}}$, podemos agora devido ao Teorema 3.3, definir o fecho transitivo de $G'_{inv} = (\Pi_N, W')$ notado por $\widehat{G'_{inv}} = (\Pi_N, \widehat{W'})$, tal que $\widehat{W'}$ é a união de W' com os demais arcos resultantes da transitividade.

Com a construção desta cadeia de grafos, $G_{inv} \subseteq G'_{inv} \subseteq G_{Liv}$, foi possível enunciar o **Teorema das Inversões** que é o objetivo principal desse Capítulo. Ele nos afirma que dado dois vértices ρ_i e ρ_j pertencentes à G_{Liv} dizemos que eles são livremente comparáveis e a permutação que tiver o menor número de inversões também terá o menor custo. Quando consideramos dois vértices quaisquer ρ_i e $\rho_j \in \Pi_N$, isto é, em todo o conjunto das permutações, podemos dizer que existe uma grande probabilidade dessas permutações possuirem seus custos diretamentes proporcionais ao seus números de inversões. O estudo estatístico que nos levou a essa conclusão foi feito por Rangel [Ran00] em sua tese de doutorado. A formulação do Teorema das Inversões é dada da seguinte forma:

Teorema 3.4 *Sendo o grafo* $\widehat{G'_{inv}}$ *o fecho transitivo de* $G'_{inv} = (\Pi_N, W')$ *e* $G_{Liv} = (\Pi_N, M)$ *o grafo de comparabilidade do Poset* $(\Pi_N, \leq_l)$. *Tem-se que* $\widehat{G'_{inv}} = G_{Liv}$.

A demonstração do Teorema das Inversões, assim como as outras demonstrações desse capítulo, deixamos para o anexo.

Para melhor visualização do Teorema das Inversões, utilizaremos o exemplo da Figura 2.1, no capítulo 2, para construir a Tabela 3.2. Tal Tabela consta em suas colunas o nível em que a permutação se encontra no Grafo das Inversões, o número de permutações em cada nível e a média dos custos de cada nível. Nesta Tabela, como já era previsto, o crescimento da média dos custos das permutações em cada nível é proporcional ao crescimento do nível a que pertencem.

Tomando o mesmo exemplo do capítulo 2, temos que a solução ótima do PQA é $\varphi = (\ 4\ \ 3\ \ 1\ \ 2\ )$ com custo igual a 143 e $\rho = (\ 2\ \ 1\ \ 3\ \ 6\ \ 5\ \ 4\ ) \in \Pi_N$ correspondente a φ. Nota-se que ρ pertence ao nível 3 do grafo, confirmando o Teorema das Inversões, isto é, ρ possui baixo número de inversões. Sendo assim, procurar $\varphi \in \Pi_n$ que possuam ρ correspondente com baixo número de inversões pode ser uma boa estratégia para a busca da solução ótima do PQA.

nível	número de permutações no nível	média dos custos
0	1	132.0000
1	5	136.4000
2	14	143.2857
3	29	152.5172
4	49	163.8163
5	71	176.4789
6	90	189.9667
7	101	204.0396
8	101	218.3564
9	90	232.6333
10	71	246.5070
11	49	259.6939
12	29	271.6207
13	14	281.5000
14	5	289.0000
15	1	294.0000

Tabela 3.2: Tabela de médias de custos para cada nível do Grafo das Inversões

Capítulo 4

O reconhecimento das soluções lineares viáveis para o PQA

4.1 Construção da matriz $HeadQ$

Já sabemos reconhecer, pelo algoritmo SolViável, se uma dada solução do problema linear é viável ou não para o problema quadrático. Diante da resposta afirmativa é possível construir uma permutação de vértices compatível com PQA. No entanto, este algoritmo seria de pouca utilidade se tivermos que enumerar todas as soluções lineares para reconhecer as viáveis para o PQA, que é o objetivo do nosso estudo. Assim a pergunta que norteia essa pesquisa é: "*É possível rastrear no conjunto de soluções do problema linear as soluções que são interessantes (no sentido de ser viável e de boa qualidade) para o problema quadrático?*". Esta pergunta é apropriada pois além do possível reconhecimento da viabilidade para o PQA de uma solução linear, o Teorema das Inversões nos diz que as permutações $\rho \in \Pi_N$ que possuem baixo número de inversões possuem custo baixo. Combinando esses fatos, vamos nos ater às permutações viáveis (ou pseudo-viáveis) com baixo número de inversões, gerando desta forma, permutações $\varphi \in \Pi_n$ de custo relativamente baixo.

Vimos no Capítulo 1 que analisando as componentes da cabeça de uma permutação linear podemos identificar se essa permutação é não-viável ou pseudo-viável. Com as componentes da cauda dizemos se ela é viável ou apenas pseudo-viável. Como esse estudo tem por objetivo o PQA vamos considerar as duas possibilidades (viável e pseudo-viável), o que aumentará o espaço de busca de soluções.

A partir daqui estaremos tentando responder à questão colocada anteriormente

através de uma matriz cuja construção é bem específica. Para isto deve-se recordar que o algoritmo SolViável indica se uma solução linear é não-viável ou constrói a sua equivalente do problema quadrático.

Analisando as construções feitas por este algoritmo, nota-se que o trabalho inicial é o de descobrir $\varphi(1)$. Para esta tarefa, percorre-se todas as imagens de $\psi^{-1}(t)$, com $t = 1, ..., N$. Depois de encontrar $p \in 1, ..., n$ tal que $\varphi(1) = p$, o algoritmo construirá $\varphi(i)$, com $i = 2, ..., n$ alocando $\varphi(i) = l \neq p$ ou $\varphi(i) = k \neq p$ onde l e k são retirados da **ListaKL**. A pergunta que fazemos agora é a seguinte: *"E se no lugar de descobrirmos a $\varphi(1)$, procurarmos quais elementos poderiam construir a cabeça de uma permutação que gere um valor de $\varphi(1)$ pré-determinado"*.

Para isso foi criado um algoritmo que armazena as cabeças das permutações em uma matriz $n \times (n-1)$ que chamaremos de $HeadQ$, onde as n linhas são respectivamente as componentes das cabeças das permutações ξ que geram $\varphi(1) = r$ para todo $r = 1, ..., n$. Para construção desta matriz usamos como base a permutação identidade $\xi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}$.

Algoritmo 2: *ConstróiHeadQ1(N)*

1: **for** t = 1, ... , N **do**
2: $\quad ListaIJ[t] \leftarrow \psi^{-1}(t) = (i, j)$;
3: **end for**
4: **for** r = 1, ... , n **do**
5: $\quad q \leftarrow 1$;
6: $\quad$ **for** t = 1, ... , N **do**
7: $\quad\quad$ **if** $ListaIJ(t).i = r$ ou $ListaIJ(t).j = r$ **then**
8: $\quad\quad\quad HeadQ(r, q) \leftarrow t$;
9: $\quad\quad\quad q \leftarrow q + 1$;
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: **end for**

Exemplo 4.1 *Para n=4 o algoritmo ConstróiHeadQ1(N) gera uma matriz HeadQ que possui a configuração da Figura 4.1:*

	$\xi(1)$	$\xi(2)$	$\xi(3)$	
1	1	2	3	$\xi = \begin{pmatrix} 1 & 2 & 3 & \dots \\ 1 & 2 & 3 & \dots \end{pmatrix} \rightarrow \varphi(1) = 1$
2	1	4	5	$\xi = \begin{pmatrix} 1 & 2 & 3 & \dots \\ 1 & 4 & 5 & \dots \end{pmatrix} \rightarrow \varphi(1) = 2$
3	2	4	6	$\xi = \begin{pmatrix} 1 & 2 & 3 & \dots \\ 2 & 4 & 6 & \dots \end{pmatrix} \rightarrow \varphi(1) = 3$
4	3	5	6	$\xi = \begin{pmatrix} 1 & 2 & 3 & \dots \\ 3 & 5 & 6 & \dots \end{pmatrix} \rightarrow \varphi(1) = 4$

Figura 4.1: matriz $HeadQ$ para $n = 4$

Um algoritmo é dito correto se para cada instância de entrada ele pára com a saída correta. Então dizemos que o algoritmo correto *resolve* o problema computacional dado, [CLRV01].

Aqui mostraremos correção do algoritmo, para isto será apresentado o progresso, invariante e término do mesmo.

O término do algoritmo se dá de forma trivial, visto que todos os *loop*'s são dados em função de n ou N, assim fica claro que o algoritmo "pára".

O objetivo do algoritmo é criar uma matriz que armazene as cabeças das permutações linerares capazes de criar uma permutação de vértices, onde em cada elemento t da linha r da matriz temos que $\psi^{-1}(t) = (i, j)$ então ou $i = r$ ou $j = r$, assim construímos a permutação de vértices com $\varphi(1) = r$. Daí conseguimos retirar a invariante do nosso algoritmo, dada por:"Ao término da linha 11 foi alocada na matriz a posição (r, q) com o elemento t, tal que um dos componentes da $ListaIJ(t)$ vai ser necessário para montar $\varphi(1) = r$". A prova da invariante será dividida em início, que é antes da primeira iteração, manutenção, que é o resultado do algoritmo após cada iteração e término, sendo o estado final do algoritmo.

Início: Antes da primeira iteração, a matriz $HeadQ$ se encontra vazia e temos somente a $ListaIJ$ construída.

Manutenção: A cada iteração do algoritmo na linha 7 temos duas possibilidades, o *if* pode ser verdadeiro ou falso. Se verdadeiro, ele constrói a $HeadQ$ na posição da linha r e coluna q com o elemento t da $ListaIJ$ tal que um dos com-

ponentes da posição t na $ListaIJ$ será necessário para gerar uma permutação de vértices com $\varphi(1) = r$. Se falso, o algoritmo simplesmente continua a busca, pelas componentes da $ListaIJ$ de modo que preencha a linha da matriz com os elementos adequados.

término: No término a matriz $HeadQ$ está preenchida. Com o *for* da linha 4 percorreremos todas as linhas da matriz e com o *for* da linha 6 buscaremos os elementos necessários para construir a cabeça da permutação.

Proposição 4.1 *A complexidade do algoritmo é dada por* $O(n^3)$.

Prova: Os *loop*'s apresentados no algoritmo são dependentes somente de n. Na linha 1 o primeiro *loop* é dado por N, lembramos que $N = \frac{n(n-1)}{2}$. Na quarta linha o segundo for com n iterações e o terceiro com N na linha 6, são dependentes tendo assim uma multiplicação entre eles. Desta forma obtemos $N + n \cdot N = N(n+1)$, substituindo o valor de N temos $\frac{n(n-1)(n+1)}{2} = \frac{n^3-n}{2}$. Portanto a complexidade é dada por $O(n^3)$. □

Exemplo 4.2 *Para exemplificar assuma* $n = 4$, *(Figura 4.2) e generalizando para* n *a (Figura 4.3).*

$$ListaIJ = \{(1,2);(1,3);(1,4);(2,3);(2,4);(3,4)\}.$$

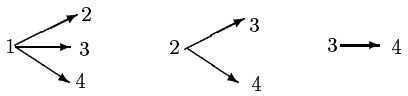

Figura 4.2: Distribuição dos elementos na $ListaIJ$ para $n = 4$

$$ListaIJ = \{(1,2);(1,3);\ldots;(1,n);(2,3);(2,4);\ldots;(2,n);\ldots;((n-1),n)\}.$$

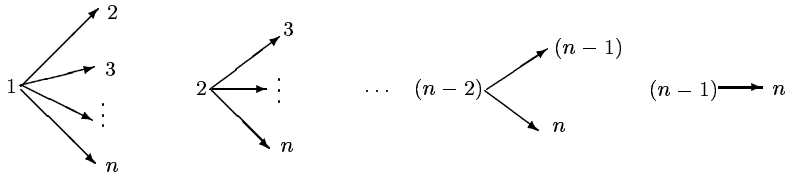

Figura 4.3: Distribuição dos elementos na $ListaIJ$

Proposição 4.2 *Permutando os* $(n-1)$ *elementos das* n *linhas existentes na* $HeadQ$*, geramos as* $n!$ *soluções do PQA.*

Prova: Considere um PQA(F,D) cujas matrizes F e D são de dimensão $n \times n$. As colunas da matriz $HeadQ$ são as imagens de $\xi(1)$, $\xi(2)$, $\xi(3)$, ..., $\xi(n-1)$. Como a $HeadQ$ tem ordem $n \times (n-1)$, cada linha $r = 1, \ldots, n$ gera (n - 1)! soluções quadráticas com $\varphi(1) = r$. Sendo n linhas, geramos as $n \cdot (n-1)! = n!$ soluções para o PQA(F,D). □

Desta forma está criada uma matriz de dimensão $n \times (n-1)$ que guarda as componentes das cabeças das permutações $\xi \in \Pi_N$ que com certeza são capazes de montar uma permutação $\varphi \in \Pi_N$. O algoritmo a seguir também constrói a matriz $HeadQ$. Este algoritmo propõe uma construção da $HeadQ$ por uma forma mais simples com complexidade $O(N)$.

Algoritmo 3: *ConstróiHeadQ2(N)*

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

A prova de correção aqui é análoga ao algoritmo anterior. A "parada" no algoritmo *ConstróiHeadQ2* é dado de forma trivial, devido ao fato do único *loop* existente no algoritmo ser finito. A invariante 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* $\varphi(1) = i$ *e* $\varphi(1) = j$*.*

início: É análogo ao *ConstróiHeadQ1*, a diferença que a matriz é construída durante a construção da $ListaIJ$.

Manutenção: Ao final de cada iteração o algoritmo se aproveita da estrutura da $ListaIJ$, como já apresentamos, 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 $N = \frac{n(n-1)}{2}$, 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 $2 \cdot \frac{n(n-1)}{2}$ 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 $1 \leq i < j \leq n$, temos que não existe duas alocações em uma mesma posição, o que assegura que a matriz é toda preenchida.

Assim temos uma resposta para a pergunta proposta no começo da discussão. Com a matriz $HeadQ$ podemos indicar em um universo de $N!$ soluções da relaxação linear exatamente quais são as $n!$ permutações interessantes para o problema quadrático. Destacamos a importância da $HeadQ$ pois esta possui uso de memória e tempo computacional para sua construção mínimos.

Nas tabelas abaixo ilustraremos o comportamento das φ's, também de ordem $n = 4$, diante de uma permutação de uma linha da matriz $HeadQ$. Para o primeiro caso tome $\xi(1) = 1$, $\xi(2) = 2$ e $\xi(3) = 3$, a primeira linha da matriz $HeadQ$ na Figura 4.1. Utilizando a construção feita pelo algoritmo SolViável citado no Capítulo 2 geramos $\varphi = (\ 1 \quad 2 \quad 3 \quad 4\ )$ que podemos acompanhar com a ajuda da Tabela 4.1.

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

Tabela 4.1: Cabeça da permutação dada pela primeira linha da matriz $HeadQ$

Permutando a linha em questão para $\xi(1) = 2$, $\xi(2) = 1$ e $\xi(3) = 3$ alteramos a posição dos elementos na construção da φ para $\varphi = (\ 1 \quad 3 \quad 2 \quad 4\ )$, de acordo com a Tabela 4.2.

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

Tabela 4.2: Cabeça gerada por uma permutação da primeira linha da matriz $HeadQ$

Note que se continuarmos permutando a primeira linha da matriz, vamos gerar

todas as 6 φ's tais que $\varphi(1) = 1$. A Tabela 4.3 mostra o restante das permutações.

Permutações da linha 1		φ gerado
[1 2 3]	$\longrightarrow$	$\varphi =$ (1 2 3 4)
[1 3 2]	$\longrightarrow$	$\varphi =$ (1 2 4 3)
[2 1 3]	$\longrightarrow$	$\varphi =$ (1 3 2 4)
[2 3 1]	$\longrightarrow$	$\varphi =$ (1 3 4 2)
[3 1 2]	$\longrightarrow$	$\varphi =$ (1 4 2 3)
[3 2 1]	$\longrightarrow$	$\varphi =$ (1 4 3 2)

Tabela 4.3: Todas as permutações da primeira linha com $n = 4$

4.2 Construção da $HeadQ^*$

Sabemos que todas as permutações $\xi \in \Pi_N$, soluções do PAL(Q), possuem correspondentes $\rho \in \Pi_N$ soluções do PAL(Q^*). Sendo assim, de posse da matriz HeadQ que guarda das cabeças das permutações $\xi \in \Pi_N$, uma solução do PAL(Q), queremos agora saber quais as posições que $\xi(1), \ldots, \xi(n-1)$ assumem em Q^* para podermos montar as permutações $\rho \in \Pi_N$, soluções do PAL(Q^*) que com certeza serão capazes de gerar uma $\varphi \in \Pi_n$ solução do PQA(F,D).

Para isto fazemos uso da bijeção $\rho = \phi_D \circ \xi \circ \phi_F^{-1}$ para efetuar a transformação da matriz $HeadQ$ na matriz que chamaremos de $HeadQ^*$.

Para melhor entendimento, usaremos as permutações $\phi_F =$ (3 6 2 1 5 4) e $\phi_D =$ (6 2 4 5 1 3) do exemplo 2.1 do capítulo 2 e a matriz $HeadQ$ da Figura 4.1.

Como estamos trabalhando somente com as cabeças das permutações, nos interessa saber em quais posições da permutação ρ as componentes das cabeças serão alocadas. Ver as posições dos elementos em negrito na Figura 4.4 (coluna relativa a ϕ_F^{-1}). Para isso, basta aplicar $\phi_F(\xi(i)), i = 1, \ldots, n-1$. A seguir, devemos conhecer a imagem que essas posições assumem em ρ, observe a imagem em negrito das permutações $\rho' s$ na Figura 4.4 (coluna relativa a ρ). Com esse objetivo, aplica-se ϕ_D

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

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

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

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

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

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

Figura 4.4: Ilustração da construção da $HeadQ^*$

em todos os elementos da matriz $HeadQ$, Tabela 4.4. A Tabela 4.5 faz um paralelo da matriz $HeadQ$ e $HeadQ^*$ para $n = 4$ e instância do exemplo 2.1 do capítulo 2. Observe que esta tabela está perfeitamente associada a Figura 4.4.

Para uma instância qualquer com $n = 4$, a matriz $HeadQ^*$ é apresentada da seguinte forma. Note que em negrito está exatamente a matriz $HeadQ$.

	$\phi_F(\xi(\mathbf{1}))$	$\phi_F(\xi(\mathbf{2}))$	$\phi_F(\xi(\mathbf{3}))$
1	$\phi_D(\mathbf{1})$	$\phi_D(\mathbf{2})$	$\phi_D(\mathbf{3})$
2	$\phi_D(\mathbf{1})$	$\phi_D(\mathbf{4})$	$\phi_D(\mathbf{5})$
3	$\phi_D(\mathbf{2})$	$\phi_D(\mathbf{4})$	$\phi_D(\mathbf{6})$
4	$\phi_D(\mathbf{3})$	$\phi_D(\mathbf{5})$	$\phi_D(\mathbf{6})$

Tabela 4.4: Matriz $HeadQ^*$ com $n = 4$

A matriz $HeadQ$ independe da instância, o que não ocorre com $HeadQ^*$ que está diretamente ligada a ordenação dos vetores **F** e **D**.

As permutações com esses valores fixados geram pseudo-viáveis em Q^*, bastando para isto completar aleatoriamente as posições vazias.

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

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

Tabela 4.5: Matriz $HeadQ$ e $HeadQ^*$

4.3 Buscando Minimização

Com o objetivo de encontrar soluções melhores, isto é, de menor custo, faremos uso das informações e conclusões contidas neste trabalho. Vimos, no Capítulo 2 a correspondência entre uma solução do PAL(Q), $\xi \in \Pi_N$, e uma solução do PAL(Q*), $\rho \in \Pi_N$. No Capítulo 3 temos o Teorema das Inversões que será a base da idéia aplicada neste processo de minimização.

O Teorema das Inversões se aplica às permutações ρ, buscando uma permutação com baixo número de inversões e consequentemente um baixo custo, para então encontrarmos a correspondente ξ, solução do PAL(Q). O grande problema é que a maioria das soluções lineares não são viáveis para o PQA. Essa questão foi resolvida com o "mapeamento" das soluções quadráticas no universo das soluções lineares através da matriz $HeadQ$.

Através da matriz $HeadQ^*$ conseguimos gerar soluções do PAL(Q^*) que certamente irão gerar uma ξ pseudo-viável. Como o método de melhoramento das soluções é buscar soluções com baixo número de inversões, uma forma simples de alcançar este objetivo é a ordenação dos valores de uma permutação $\rho \in \Pi_N$.

Para não correr o risco de gerar soluções não-viáveis, vamos ordenar somente as posições das permutações $\rho \in \Pi_N$ de tal forma que continuem sendo pseudo-viáveis. Um facilitador dessa tarefa é usar a matriz $HeadQ^*$, já que ela é formada pelos elementos correspondentes às cabeças de $\xi \in \Pi_N$. Portanto, basta fazer uma ordenação nas linhas da matriz $HeadQ^*$.

Como nossas permutações quadráticas, $\varphi \in \Pi_n$, são geradas a partir da matriz $HeadQ$, devemos transferir cada troca de posição durante a ordenação da $HeadQ^*$

para a $HeadQ$. Assim temos as cabeças das permutações $\xi \in \Pi_N$ capazes de gerar $\varphi's$ de boa qualidade.

Exemplo 4.3 *Veja o diagrama feito com a primeira linha da HeadQ*

$$\begin{array}{ccc} HeadQ & & HeadQ^* \\ [\ 1 \quad 2 \quad 3\ ] & \longrightarrow & [\ 4 \quad 6 \quad 2\ ] \\ & & \downarrow \text{Ordenação} \\ [\ 3 \quad 1 \quad 2\ ] & \longleftarrow & [\ 2 \quad 4 \quad 6\ ] \end{array}$$

Feito isso, note que a φ gerada pela primeira linha da matriz antes da ordenação é $\varphi = (\ 1 \quad 2 \quad 3 \quad 4\ )$ tendo o custo igual a 200 e após a ordenação temos a $\varphi = (\ 1 \quad 4 \quad 2 \quad 3\ )$ com custo igual a 148. Como previsto, o custo teve uma melhora. Este processo se repete para cada linha da $HeadQ^*$ gerando n permutações $\varphi's$ de boa qualidade. No capítulo seguinte, descreveremos um algoritmo que explora esta idéia aqui apresentada.

Capítulo 5

Implementação computacional

Neste Capítulo é apresentado uma implementação computacional das idéias desenvolvidas neste trabalho. Teoricamente é possível encontrar a solução ótima para uma instância com a simples enumeração de todas as soluções, porém sabemos que o PQA é um problema altamente combinatorial o que inviabiliza essa tarefa. Então optamos por uma heurística construtiva que explora as idéias deste trabalho. A heurística surge, de forma natural, possui duas fases: preparação e armazenamento dos dados; e uma busca local com as soluções geradas.

Após apresentação do algoritmo produzido nesse trabalho mostraremos alguns resultados obtidos em instâncias conhecidas retiradas da QAPLIB.

5.1 O Algoritmo

Em linha gerais, o algoritmo que chamamos de **HeuristicHead**, consiste em um método iterativo onde temos uma etapa inicial que construirá as soluções iniciais usadas na busca local. Para a construção das soluções iniciais é usada a matriz $HeadQ$ que irá gerar essas soluções, e como critério para um melhoramento prévio, o Teorema das Inversões e os resultados probabilísticos relacionados a ele em [Ran00]. Em cada iteração da **HeuristicHead** é tomada uma solução inicial e aplicamos uma busca local em uma vizinhança desta solução para melhoria do resultado. Após o término das iterações é escolhida a melhor solução. A idéia principal do algoritmo **HeuristicHead** é traduzido em pseudo-código da seguinte forma:

Algoritmo 4: *HeuristicHead(n)*

1: Entrada de Dados;
2: Construção das soluções iniciais na Matriz $HeadQ$;
3: **for** i = 1, ... , n **do**
4: Constrói$\varphi(HeadQ[\mathrm{i}])$;
5: BuscaLocal(φ);
6: Atualizar Solução;
7: **end for**
8: Retorna a melhor solução encontrada;

5.1.1 Fase Inicial do Algoritmo

A fase inicial, que caracteriza o algoritmo, é composta de alguns procedimentos que apresentamos a seguir.

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 $HeadQ^*$ e $HeadQ$.

Os itens 2 e 3 devem pertencer à todo algoritmo que trabalhe com uma correspondência entre o Problema Quadrático de Alocação e sua relaxação na forma do Problema de Alocação Linear, pois são esses procedimentos que fazem a relaxação do PQA para o PAL. Os itens 3 e 4 irão caracterizar nossa heurística, pois neles estão contidos as idéias desenvolvidas no capítulo anterior. Sabemos que a construção da matriz $HeadQ$ é independente dos procedimentos 1, 2 e 3. De posse das cabeças das permutações lineares, matriz $HeadQ$, as usamos no 6^o procedimento, que é a aplicação do Teorema das Inversões como critério para a melhoria prévia das

soluções iniciais. Então teremos soluções lineares com baixo número de inversões que, com certeza, irão gerar soluções quadráticas com baixo custo. Essas soluções são adequadas para iniciar um processo de busca local.

Antes da descrição da busca local, vale lembrar que dada uma instância de ordem n, pela construção da matriz $HeadQ$, obtemos n soluções iniciais. Para a estrutura de vizinhança adotada nesse trabalho consideramos que essas n soluções iniciais não são suficientes para explorar o espaço de busca satisfatoriamente. Para solucionar esse problema foram feitas perturbações nas soluções iniciais com o objetivo de gerar novos elementos para a etapa da busca local. Dada uma linha da matriz $HeadQ$ $l_i = [h_{i1}, h_{i2}, \ldots, h_{i(n-1)}]; i = 1, \ldots, n$, uma cabeça de uma permutação linear que é pseudo-viável, faremos $n-2$ perturbações da seguinte maneira:

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

onde l'_i gerará uma nova solução inicial que sofreu um pequeno aumento no número de inversões, de forma que ainda sejam consideradas boas. Isso ajuda para que o algoritmo não fique restrito a um baixo número de soluções iniciais.

Uma desvantagem desse método apresentado de geração de soluções iniciais, é que não possui nenhum fator aleatório. Como conseqüência, uma vez computado uma instância seu resultado é estático.

5.1.2 Busca Local

Para o procedimento de busca local devemos definir uma estrutura de vizinhança. Na literatura encontramos diversas estratégias propostas para a busca local, com algumas estruturas de vizinhanças bem definidas. Neste trabalho adotamos uma estratégia bem conhecida, k-troca, encontrada em [LPR94].

Sejam φ_1 e φ_2 permutações, temos que a diferença entre elas é dada por

$$\delta(\varphi_1, \varphi_2) = \{i | \varphi_1(i) \neq \varphi_2(i)\}$$

e a distância entre φ_1 e φ_2 é definida por

$$d(\varphi_1, \varphi_2) = |\delta(\varphi_1, \varphi_2)|.$$

A vizinhança de uma permutação φ são todas as permutações φ' onde a distância entre φ e φ' é menor ou igual à k, tal que $2 \leq k \leq n$. Para este trabalho foi

considerado $k = 2$, assim nossa vizinhança é conhecida como 2-trocas. Para exemplo tome $\varphi = (\ 1\ \ 3\ \ 2\ \ 4\ )$ com $n = 4$ e $|Viz(\varphi)| = C_{4,2} = 6$, então temos

$$Viz(\varphi) = \{(\ 3\ \ 1\ \ 2\ \ 4\ ), (\ 2\ \ 3\ \ 1\ \ 4\ ), (\ 4\ \ 3\ \ 2\ \ 1\ ),$$
$$(\ 1\ \ 2\ \ 3\ \ 4\ ), (\ 1\ \ 4\ \ 2\ \ 3\ ), (\ 1\ \ 3\ \ 4\ \ 2\ )\}.$$

Essa estrutura descrita é a primeira fase em uma estratégia de Busca por Largura e Profundidade. O próximo passo é escolher a solução de menor custo e tomá-la como nova solução inicial de uma nova busca local, e este processo é repetido até que não haja mais melhorias numa vizinhança de uma solução dada.

5.2 Paralelização

Analisando o algoritmo **HeuristicHead**, observamos que as operações em cada linha da matriz *HeadQ* são independentes. Durante a computação, a única comunicação entre as linhas da matriz será no final para a escolha da melhor solução ótima local obtida. Essas observações fazem com que o algoritmo tenha uma fácil conversão ao paralelismo.

5.2.1 Por que paralelizar?

A paralelização é uma técnica utilizada em tarefas grandes e complexas para obter resultados mais rápidos, dividindo-as em tarefas pequenas que serão distribuídas em diversos processadores para serem executadas simultaneamente. A definição de Almasi e Gottlieb [AG98] representa bem os elementos de uma idéia de programação paralela: "*um computador paralelo é um conjunto de elementos de processamento que se comunicam e cooperam para resolver grandes problemas*".

O principal atrativo de se paralelizar são os baixos preços dos computadores pessoais, que são fabricados em massa, e o avanço na velocidade das redes de comunicação. Isto fez com que *clusters* de computadores tenham emergido como boa opção de sistemas para alto desempenho a baixo custo.

O grupo de Computação de Alto Desempenho da Universidade Federal do Espírito Santo, UFES, tem trabalhado com aplicações em paralelo. Seguindo a tendência

de montagem de sistemas paralelos de baixo custo, foi montado no Laboratório de Computação de Alto Desempenho (LCAD) [**?**] um *cluster* com máquinas do tipo computador pessoal. A estrutura do laboratório apresenta a seguinte configuração: 64 processadores ATHLON XP 1800, onde o Master possui 512MB e 90GB de disco, e nós com 256 RAM, 20 GB de disco; a interconexão é dada por um Switch 4300 3COM, 48 portas (Fast-Ethernet) + 1 porta Gigabit Ethernet e um Switch 4300 3COM, 48 portas (Fast-Ethernet) + 2 porta Gigabit Ethernet; com o sistema operacional Linux Red Hat 7.1.

5.2.2 Algoritmo em Paralelo

A conversão do algoritmo serial para o paralelo foi feito segundo o padrão de comunicação MPI (*Message Passing Interface*), que teve sua padronização pelo comitê do MPI Forum [MPI]. O MPI é um padrão de biblioteca com passagem de mensagens para sistemas paralelos. Foi desenvolvido para fornecer uma base comum de desenvolvimento de programas paralelos em plataformas distintas. Em virtude disso, ele incorpora aspectos de variados sistemas ao invés de adotar como padrão um sistema específico.

A paralelização do algoritmo **HeuristicHead** 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 $[1+(n-2)]$ permutações. O balanceamento é feito 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. Essa distribuição será discutida mais adiante.

5.3 Resultados

Nesta sessão vamos apresentar os resultados computacionais obtidos pelo algoritmo **HeuristicHead**. Para os testes foram usadas instâncias disponíveis na QAPLIB.

Em ambas implementações (Serial e Paralelo), o número de soluções iniciais

geradas para a busca local é proporcional ao tamanho da instância. Para verificar isso, basta lembrar que a cada linha da matriz *HeadQ* são geradas $n-1$ soluções iniciais, e como temos n linhas, o número de soluções iniciais é dado por $n*(n-1)$. Para o exemplo de instâncias de ordem 30 são geradas 870 soluções.

5.3.1 Serial

O algoritmo foi implementado usando a linguagem C++ e executado em um Pentium IV, 1,3GHr com 256 MB de memória RAM. Para o algoritmo em série foram testadas instâncias de ordem menor que 30. Na Tabela 5.1, apresentamos os resultados obtidos na seguinte ordem: solução apresentada na QAPLIB, resultado da computação da **HeuristicHead** e tempo computacional (dado em segundos) gasto para cada instância. A solução oferecida pela QAPLIB foi aqui apresentada, para que se possa mensurar a qualidade das soluções do algoritmo proposto.

Nosso objetivo inicial é analisar o desempenho do algoritmo comparando a qualidade das soluções encontradas com o tempo gasto para isso.

O método não garante a otimalidade das soluções, contudo o Teorema das Inversões assegura a boa qualidade das mesmas. Uma característica encorajadora do algoritmo é a relação entre o tempo computacional e a qualidade das soluções. Como exemplo podemos citar o nug20, que não foi atingido o ótimo, porém gasto apenas 8,2 segundos para 380 iterações com 1,01% de distância do valor ótimo. Observando o bom desempenho, passaremos para os testes com instâncias maiores.

5.3.2 Paralelo

O algoritmo paralelo foi executado no *cluster* do laboratório LCAD, descrito anteriormente. Foram testadas algumas instâncias para verificação de desempenho, executando uma mesma instância várias vezes com números diferentes de processadores. Mesmo sabendo que a solução seria a mesma, o resultado deste teste foi apresentado pois o interesse nesse momento é de analisarmos o progresso da relação **número de processadores** $\times$ **tempo** (em segundos).

Para a instância lipa40a, o custo da melhor solução viável disponível na QAPLIB é igual a 31538 e o algoritmo **HeuristicHead** obteve custo igual a 31544 com erro

Instância	QAPLIB	*HeadQ*	Tempo (seg)	Erro
chr12a	9552	9552	0,17	0%
chr12b	9742	9742	0,20	0%
nug12	578	582	0,20	0,69%
rou12	235528	235528	0,18	0%
scr12	31410	31410	0,18	0%
had12	1652	1652	0,22	0%
nug15	1150	1152	1,08	0,17%
rou15	354210	360356	0,90	1,74%
tai15a	388214	390782	0,73	0,66%
lipa20a	3683	3800	6,23	3,18%
nug20	2570	2596	8,20	1,01%
scr20	110030	110058	8,80	0,03%
lipa30a	13178	13190	115,58	0,09%
nug30	6124	6156	153,68	0,52%

Tabela 5.1: Resultados obtidos de instâncias conhecidas ($n \leq 30$)

de 0.02%. Na Tabela 5.2, o algoritmo mostra um comportamento regular na relação **número de processadores** $\times$ **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 o tamanho da instância pelo número de processadores. Por exemplo, temos que $40 \div 10 = 4$ e $40 \div 12 = 3,33...$, 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.

lipa40a	
Processadores	Tempos (seg)
1	999,87
2	495,25
10	101,18
12	101,08
15	73,98
19	75,88
20	49,60
25	46,98

Tabela 5.2: Relação **número de processadores** $\times$ **tempo** para a instância lipa40a

Observe ainda a regularidade: para 2 processadores o tempo é aproximadamente a metade de 1 processador; para um único processador, ele terá que computar as quarenta linhas da matriz $HeadQ$, enquanto para 10 e 12 processadores o máximo de trabalho para um processador é de quatro linhas, assim temos que o tempo gasto para 10 e 12 processadores deve ser aproximadamente um décimo do tempo gasto para 1 processador, como de fato acontece; seguindo esse raciocínio temos que para 20 e 25 processadores serão efetuadas no máximo duas linhas por cada processador, então o tempo gasto deverá ser aproximadamente um vigésimo do tempo gasto para 1 processador. Esta regularidade, comprovada em todos os testes, é explicada pela independência da computação entre as linhas e a comunicação entre os processadores acontecer somente na escolha do melhor resultado parcial, para assim apresentar a solução final.

Nas tabelas 5.3 e 5.4 apresentamos resultados dos problemas teste Lipa60a e Lipa80a com a relação **número de processadores** $\times$ **tempo**. Acreditamos que esses resultados irão reforçar a idéia da regularidade do algoritmo. Devido ao tamanho das instâncias (Lipa60a e Lipa80a) os testes só foram feitos com mais de 20 processadores.

lipa60a	
Processadores	Tempos (seg)
20	888,42
25	847,87
30	578,07
35	556,38

Tabela 5.3: Relação número de processadores × tempo para a instância lipa60a

lipa80a	
Processadores	Tempos (seg)
20	6892,77
25	6949,95
30	5222,18
35	5325,48
40	3464,03

Tabela 5.4: Relação número de processadores × tempo para a instância lipa80a

Analisando os resultados das tabelas 5.2, 5.3 e 5.4, conclui-se que para um melhor uso dos recursos computacionais é interessante que ao computar um problema teste, o número de processadores seja um divisor da sua ordem para que não tenhamos processadores "parados" apenas esperando resultados.

Uma vez apresentada a regularidade e desempenho do algoritmo, é de extrema importância mostrar a qualidade das soluções encontradas. Seguindo a conclusão anterior, o número de processadores adotados aqui será sempre um divisor da ordem do problema teste referente. A Tabela 5.5 segue os mesmos moldes da Tabela 5.1: a melhor solução apresentada na QAPLIB, a solução encontrada pelo algoritmo proposto, o número de processadares usado para a instância, o tempo gasto e porcentagem do erro.

Ainda sobre o tempo computacional, a pergunta que vem de forma natural é:

"por que instâncias de mesma ordem e calculadas com o mesmo número de processadores, como em sko100a, sko100b, sko100c e sko100d, possuem tempo computacional consideravelmente diferentes?", A resposta para a pergunta provavelmente está na qualidade das soluções iniciais geradas pelo algoritmo para cada exemplar do PQA, fazendo com que o algoritmo demore mais tempo computacional na fase de busca local.

Instância	QAPLIB	*HeadQ*	Processadores	Tempo (seg)	Erro
lipa40a	31538	31544	20	49,60	0,02%
lipa50a	62093	62144	25	209,32	0,08%
lipa60a	107218	108112	30	578,07	0,83%
lipa70a	169755	170930	35	1565,30	0,69%
lipa80a	253195	254862	40	3464,03	0,66%
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%

Tabela 5.5: Alguns resultados obtidos de instância consideradas grandes

Observamos na Tabela 5.5 que para instâncias de ordem consideradas de grande porte, em nenhum caso o algoritmo encontrou a melhor solução viável conhecida. Contudo, a relação **qualidade de soluções** $\times$ **tempo** nos dá resultados animadores. Em todos os casos o algoritmo conseguiu soluções viáveis de ótima qualidade, podendo destacar Lipa40a, Lipa50a e Sko100a-d.

Uma vez apresentado os resultados da **HeuristicHead**, podemos fazer algumas comparações com resultados encontrados na literatura. Dentre eles podemos citar um resultado obtido por Resende *et al.* em [PPR95] que conseguiu um erro de 0,06% para o Nug30 usando o GRASP e os trabalhos de Rangel [Ran00] e [RAB00] que obteve um erro de 0,5% nas soluções de Sko100a-d sendo este o mesmo erro encontrado neste trabalho.

Capítulo 6

Conclusão

Neste trabalho apresentamos o Problema Quadrático de Alocação com uma relaxação na forma do Problema de Alocação Linear. Ainda nesta mesma etapa foram estabelecidas algumas relações entre esses dois problemas, tal como uma classificação quanto a factibilidade das soluções lineares em relação ao Problema Quadrático de Alocação. No capítulo seguinte temos um resultado importante, conhecido na literatura como Teorema das Inversões, que foi provado por Rangel [Ran00]. Utilizamos este resultado como sendo um outro recurso para critério de uma pré-seleção de soluções para o PQA de boa qualidade.

Um grande desafio era procurar uma solução linear que levasse o algoritmo a encontrar uma solução quadrática correspondente. A proposta desse trabalho foi promover um mapeamento das soluções lineares no sentido de apresentar soluções do problema linear que tivéssemos a certeza que seriam soluções pseudo-viáveis que conseguem gerar uma solução do PQA, que é de grande interesse para o estudo.

Assim, com a matriz $HeadQ$, construímos uma estrutura de baixo custo computacional e com pouco uso de memória que identifica quais dentre as soluções da relaxação linear são pseudo-viáveis para o problema quadrático de alocação. Para isso, foram propostos dois algoritmos, *ConstróiHeadQ1* e *ConstróiHeadQ2*, ambos com complexidade polinonial. O segundo, por ser melhor, foi utilizado no algoritmo construtivo apresentado nesse trabalho para construir soluções viáveis de boa qualidade para o PQA. Uma forma de utilização da matriz $HeadQ$ é combinar seus resultados com as conclusões obtidas do Teorema das Inversões, ou seja, construir soluções do PAL capazes de gerar soluções do PQA com um baixo número de inversões e como conseqüência, um baixo custo da função objetivo.

Com essas idéias, para cada instância de ordem n, geramos n soluções iniciais consideradas boas. Aplicando a busca local, por largura e profundidade, na vizinhança definida para cada solução inicial, apresentamos um algoritmo construtivo, que chamamos de **HeuristicHead**. Finalmente, apresentamos os resultados da implementação computacional. Na versão serial para instâncias pequenas, foram encontrados os ótimos e, nos demais casos, consideramos que os resultados são satisfatórios em relação a qualidade dasoluções. Na versão paralela, os testes foram feitos de duas formas: um para mostrar desempenho, onde o algoritmo demonstrou regularidade, facilitando a previsão do tempo gasto nos demais testes; e outra mostrando a qualidade das soluções, que mesmo não alcançando as soluções disponíveis na QAPLIB para grandes instâncias, apresentou uma boa relação **qualidade de solução $\times$ tempo computacional**. O erro para todas as instâncias de ordem $n = 100$ foi de 0.5% tomando como base a melhor solução viável disponível na QAPLIB. Para as outras instâncias testadas, a média dos erros foi de $0,45\%$.

Durante os testes de desempenho e tempo computacional, concluiu-se que o melhor número de processadores é um divisor da dimensão de n. Desta forma, todos os processadores terminaram suas tarefas aproximadamente ao mesmo tempo, evitando atrasos na etapa final do algoritmo na versão paralela. Tal etapa se refere à comparação de todos os ótimos locais encontrados por cada processador para eleger a melhor solução.

Para trabalhos futuros podemos sugerir estudos e implementação de algoritmos exatos usando a teoria envolvendo as matrizes $HeadQ$ e $HeadQ^*$, uma estratégia interessante é o uso do método de *branch-and-cut*. A idéia do processo seria caminhar no grafo G_{Liv} partindo de uma permutação com um baixo número de inversões e tentando fazer cortes em ramos do grafo onde teremos a certeza que não haveria melhorias na solução. Para a eficiência do mesmo, pode-se usar a matriz $HeadQ$ para que os passos do algoritmo sejam dados sobre as soluções viáveis, um vez que já conseguimos mapeá-las no G_{Liv}.

Para introduzir um componente aleatório no algoritmo, seria interessante que as perturbações feitas nas linhas da matriz fossem aleatórias. Há o risco de se perder a qualidade das soluções geradas, porém uma diversificação no espaço de busca é fator positivo.

Apêndice A

Demonstrações do Capítulo 3

Nesse anexo constam as demonstrações dos teoremas e lemas do capítulo 2. Aqui usaremos as definições e equações contidas no mesmo capítulo 2.

As demonstrações aqui apresentadas estão contidas em [Ran00] e [RA03].

A.1 Demonstrações

Para a demonstração do Lema 3.1 utilizaremos as definições dos conjuntos P_N, P_P e P_0 e da Partição Canônica.

Lema A.1 *Na partição canônica de $Z_{\rho_1} - Z_{\rho_2}$ temos que*

$$-\sum_{t\in P_N}\sum_{i=0}^{\rho_2(t)-\rho_1(t)-1}(d^+_{\rho_1(t)+i}-d^+_{\rho_1(t)+i+1}) = \sum_{t'\in P_P}\sum_{j=0}^{\rho_1(t')-\rho_2(t')-1}(d^+_{\rho_1(t')-j}-d^+_{\rho_2(t')-j-1}).$$

Além disso, o valor absoluto de cada fator diferença do primeiro membro corresponde a um fator-diferença do segundo membro e reciprocamente.

Prova: Seja $\rho_1, \rho_2 \in \Pi_N$, temos que

$$\sum_{t\in\Omega_N}(d^+_{\rho_1(t)} - d^+_{\rho_2(t)}) = 0,$$

reescrevendo utilizando as informações da equação 3.2, que é a eliminação dos elementos pertencentes a P_0 $(\rho_1(t) - \rho_2(t) = 0)$, conseguimos

$$\sum_{t\in P_N}(d^+_{\rho_1(t)} - d^+_{\rho_2(t)}) + \sum_{t'\in P_P}(d^+_{\rho_1(t')} - d^+_{\rho_2(t')}) = 0,$$

logo $-\sum_{t\in P_N}(d^+_{\rho_1(t)} - d^+_{\rho_2(t)}) = \sum_{t'\in P_P}(d^+_{\rho_1(t')} - d^+_{\rho_2(t')})$.

Introduzindo os termos simétricos, o valor da expressão acima não se altera e chegamos a equação que é objetivo da proposição,

$$-\sum_{t\in P_N}\sum_{i=0}^{\rho_2(t)-\rho_1(t)-1}(d^+_{\rho_1(t)+i} - d^+_{\rho_1(t)+i+1}) = \sum_{t'\in P_P}\sum_{j=0}^{\rho_1(t')-\rho_2(t')-1}(d^+_{\rho_1(t')-j} - d^+_{\rho_2(t')-j-1}).$$

Ainda observamos que para todo $t \in P_N$ e $i \in \{0,\ldots,\rho_2(t)-\rho_1(t)-1\}$ existe um $t' \in P_P$ e $j \in \{0,\ldots,\rho_1(t')-\rho_2(t')-1\}$ tal que

$$\rho_1(t) + i = \rho_1(t') - j - 1 \text{ e } \rho_1(t) + i + 1 = \rho_1(t') - j,$$

resultando em

$$-(d^+_{\rho(t)+i} - d^+_{\rho(t)+i+1}) = (d^+_{\rho(t')-j} - d^+_{\rho(t')-j-1}).$$

□

No Teorema 3.1 serão usadas as definições de N^i_t, $P^{\prime j}_t$, permutações livremente comparáveis e ainda, da bijeção β com $\beta(N^i_t) = P^{\prime j}_t$.

Teorema A.1 Teorema da Ordenação Parcial Livre - TOPL: *Temos $Z_{\rho_1} \leq_l Z_{\rho_2}$ ($Z_{\rho_2} \leq_l Z_{\rho_1}$), se e somente se, existir uma bijeção β tal que $\beta(N^i_t) = P^j_{t'}$; $t \in P_N$; $t' \in P_P$; $i = 0,\ldots,\rho_2(t)-\rho_1(t)-1$ e $j = 0,\ldots,\rho_1(t')-\rho_2(t')-1$ capaz de induzir o par ordenado (t,t'), com $t < t'(t' < t)$, $\forall t \in P_N$ e $\forall t' \in P_P$*

Prova:

" $\Rightarrow$ " Decorre imediatamente do fato de toda bijeção β tal que $\beta(N^i_t) = P^j_{t'}$ pode-se induzir pares ordenados (t,t'); $t \in P_N$ e $t' \in P_P$ e do fato de F^- e D^+ serem ordenados.

" $\Leftarrow$ " Considere $Z_{\rho_1} \leq_l Z_{\rho_2}$ então temos que existe uma bijeção β tal que $\beta(N^i_t) = P^j_{t'}$ com $t \in P_N$; $t' \in P_P$; $i = 0,\ldots,\rho_2(t)-\rho_1(t)-1$ e $j = 0,\ldots,\rho_1(t')-\rho_2(t')-1$

Suponha por absurdo termos $Z_{\rho_1} \leq_l Z_{\rho_2}$ e $t'^* < t^*$ para algum $t^* \in P_N$ e $t'^* \in P_P$ e uma bijeção β. Da equação 3.7, tem-se a partição canônica

$$Z_{\rho_1} - Z_{\rho_2} = \sum_{t\in P_N}\sum_{i=0}^{\rho_2(t)-\rho_1(t)-1}(f^-_t - f^-_{t'})(d^+_{\rho_1(t)+i} - d^+_{\rho_1(t)+i+1}) \leq 0.$$

Considerando a ordenação de F^- e D^+ temos que $(f^-_{t^*} - f^-_{t'^*}) \leq 0$ e $(d^+_{\rho_1(t)+i} - d^+_{\rho_1(t)+i+1}) \leq 0$. Seja L o valor da expressão

$$L = \sum_{i=0}^{\rho_2(t^*)-\rho_1(t^*)-1} (f^-_{t^*} - f^-_{t'_*})(d^+_{\rho_1(t^*)+i} - d^+_{\rho_1(t^*)+i+1}) \geq 0.$$

Tome uma instância do PQA tal que L seja suficientemente grande para termos

$$L \geq \sum_{t \in P_N - \{t^*\}} \sum_{i=0}^{\rho_2(t)-\rho_1(t)-1} (f^-_t - f^-_{t'})(d^+_{\rho_1(t)+i} - d^+_{\rho_1(t)+i+1}).$$

Para todo par (t, t'), induzido da bijeção β. Para esse caso, ocorre a contradição, dado que $Z_{\rho_1} - Z_{\rho_2} \leq 0$. □

O Lema 3.2 começa a nos mostrar a relação que existe entre as permutações livrementes comparáveis e o Grafo das Inversões, que é notado por $G_{inv} = (\Pi_N, W)$.

Lema A.2 *Sejam* $\rho_1, \rho_2 \in \Pi_N$. *Se* $(\rho_1, \rho_2) \in W \Rightarrow Z_{\rho_1} \leq_l Z_{\rho_2}$

Prova: Dado $\rho_1, \rho_2 \in \Pi_N$ tal que $\rho_1 = (\rho_1(1), \ldots, \rho_1(i), \rho_1(i+1), \ldots, \rho_1(N))$ e $\rho_2 = (\rho_1(1), \ldots, \rho_1(i+1), \rho_1(i), \ldots, \rho_1(N))$ com $1 \leq i < N$ e $\rho_1(i) < \rho_1(i+1)$, isto é $(\rho_1, \rho_2) \in W$.

Fazendo a classificação dos termos das permutações nos três conjuntos, temos $P_N = \{i\}$, $P_P = \{i+1\}$ e $P_O = \{1, \ldots, i-1, i+2, \ldots, N\}$. Assim podemos induzir a bijeção $\beta(N^0_i) = P^0_{i+1}$ com $i \in P_N$, $i+1 \in P_P$ e ainda tendo que $i < i+1$. Pelo Teorema da Ordenação Parcial Livre temos $Z_{\rho_1} \leq_l Z_{\rho_2}$. □

Teorema A.2 *Sejam* ρ_i *e* ρ_j *vértices do grafo* G_{inv} *ligados por um caminho de arcos em* W. *Tem-se* $|\Im(\rho_i)| < |\Im(\rho_j)|$, *se o somente se,* $Z_{\rho_i} \leq_l Z_{\rho_j}$.

Prova: Suponha inicialmente que $||\Im(\rho_i)| - |\Im(\rho_j)|| = 1$, pelo Lema 3.2 fica claro que neste caso o teorema é verdadeiro. Suponha agora que a afirmação seja verdadeira para um caminho onde $||\Im(\rho_i)| - |\Im(\rho_j)|| = m-1$, para simplificar a notação admita $\rho_i = \rho_1$ e $\rho_j = \rho_m$. Por hipótese temos que $Z_{\rho_1} \leq_l Z_{\rho_m}$, note que $Z_{\rho_m} \leq_l Z_{\rho_{m+1}}$ já que $||\Im(\rho_m)| - |\Im(\rho_{m+1})|| = 1$, então por transitividade conseguimos que $Z_{\rho_1} \leq_l Z_{\rho_{m+1}}$. De volta a notação original temos que $|\Im(\rho_i)| < |\Im(\rho_j)|$, se e somente se, $Z_{\rho_i} \leq_l Z_{\rho_j}$.
□

Com o Teorema 3.2 aumentamos o número de permutações livremente comparáveis que são conhecida. O objetivo é mapear no Grafo de Comparabilidade todas as permutações livremente comparáveis.

Lema A.3 *Sejam ρ_1 e ρ_2 duas permutações tais que $||\Im(\rho_1)| - |\Im(\rho_2)|| = 1$ então Z_{ρ_1} e Z_{ρ_2} são livremente comparáveis, se e somente se, ρ_1 e ρ_2 diferem entre si de dois elementos.*

Prova: ($\Rightarrow$) Suponha por absurdo a divisão das permutações nos três conjuntos $P_N = \{t_1, t_2\}$, $P_P = \{t'\}$ e $P_0 = \Omega^N - \{P_N \cup P_P\}$, isso nos dá permutações que diferem de mais de 2 elementos. Como a construção dos conjuntos P_N, P_P e P_0 é feita em ordem, temos que $t_1 < t_2$.

Ainda como os elementos de duas permutações de Ω^N só diferem da ordem tem-se que

$$\sum_{t \in \Omega^N} \rho_i(t) - \rho_j(t) = 0. \tag{A.1}$$

Eliminando os elementos de $t \in P_0$, a equação A.1 torna-se

$$\rho_i(t_1) - \rho_j(t_1) + \rho_i(t_2) - \rho_j(t_2) + \rho_i(t') - \rho_j(t') = 0. \tag{A.2}$$

Referente à ordenação entre os elementos de P_N e P_P temos três possibilidades: (a) $t_1 < t_2 < t'$; (b)$t' < t_1 < t_2$; e (c)$t_1 < t' < t_2$.

Analisando o item (a) mediante as características dos conjuntos P_N e P_P, podemos enumerar dois casos.

(I)

$$\begin{aligned} \rho_i(t_1) = \rho_j(t_2),\ \text{como } \rho_i(t_1) < \rho_j(t_1) \Rightarrow \rho_j(t_2) < \rho_j(t_1); \\ \rho_i(t_2) = \rho_j(t'),\ \text{como } \rho_i(t_2) < \rho_j(t_2) \Rightarrow \rho_j(t') < \rho_j(t_2); \\ \rho_i(t') = \rho_j(t_1),\ \text{como } \rho_i(t') > \rho_j(t') \Rightarrow \rho_j(t') < \rho_j(t_1). \end{aligned} \tag{A.3}$$

(II)

$$\begin{aligned} \rho_i(t_1) = \rho_j(t'),\ \text{como } \rho_i(t_1) < \rho_j(t_1) \Rightarrow \rho_j(t') < \rho_j(t_1); \\ \rho_i(t_2) = \rho_j(t_1),\ \text{como } \rho_i(t_2) < \rho_j(t_2) \Rightarrow \rho_j(t_1) < \rho_j(t_2); \\ \rho_i(t') = \rho_j(t_2),\ \text{como } \rho_i(t') > \rho_j(t') \Rightarrow \rho_j(t') < \rho_j(t_2). \end{aligned} \tag{A.4}$$

Para visualizar os casos acima, sejam as permutações ρ_i e ρ_j

$$\begin{aligned} &\ldots \rho_i(t_1) \ldots \rho_i(t_2) \ldots \rho_i(t') \ldots \\ &\ldots \rho_j(t_1) \ldots \rho_j(t_2) \ldots \rho_j(t') \ldots \end{aligned} \tag{A.5}$$

Utilizando as relações em A.3 nas permutações ilustrada em A.5 e, em seguida, A.4 nas mesmas permutações, tem-se que $|\Im(\rho_j)| - |\Im(\rho_i)| \leq 2$. Contradição, pois por hipótese e diferença deve ser de uma unidade.

Para o item (b) o raciocínio é analogo, porém no sentido contrário com $t' < t_1 < t_2$, resultando na mesma contradição com relação ao número de inversões.

Para o último caso (c), onde $t_1 < t' < t_2$, temos que $Z_{\rho_i} - Z_{\rho_j}$ pode ser expressa por

$$Z_{\rho_i} - Z_{\rho_j} = f^-_{t_1}(d^+_{\rho_i(t_1)} - d^+_{\rho_j(t_1)}) + f^-_{t_2}(d^+_{\rho_i(t_2)} - d^+_{\rho_j(t_2)}) + f^-_{t'}(d^+_{\rho_i(t')} - d^+_{\rho_j(t')}). \tag{A.6}$$

Subtituindo as igualdades de A.3 em A.6, tem-se:

$$\begin{aligned} Z_{\rho_i} - Z_{\rho_j} &= f^-_{t_1}(d^+_{\rho_i(t_1)} - d^+_{\rho_i(t')}) + f^-_{t_2}(d^+_{\rho_i(t_2)} - d^+_{\rho_i(t_1)}) + f^-_{t'}(d^+_{\rho_i(t')} - d^+_{\rho_i(t_2)}) \\ &= d^+_{\rho_i(t_1)}(f^-_{t_1} - f^-_{t_2}) + d^+_{\rho_i(t_2)}(f^-_{t_2} - f^-_{t'}) + d^+_{\rho_i(t')}(f^-_{t'} - f^-_{t_1}). \end{aligned} \tag{A.7}$$

Apesar do vetor F^- estar em ordem não-crescente, não há como afirmar que $Z_{\rho_i} - Z_{\rho_j} \leq 0$ ou que $Z_{\rho_j} - Z_{\rho_i} \leq 0$, sem o conhecimento prévio dos valores das componentes e F^-.

Quando substituímos as igualdades A.4 na equação A.7, o raciocínio é análogo e chegamos a mesma contradição. Sendo assim, quando $t_1 < t' < t_2$ os produtos não são livremente comparáveis.

Assim, os conjuntos P_N e P_P devem ser unitários.

($\Leftarrow$) Considere a expressão 3.2

$$Z_{\rho_1} - Z_{\rho_2} = \sum_{t \in P_N} f^-_t(d^+_{\rho_1(t)} - d^+_{\rho_2(t)}) + \sum_{t \in P_P} f^-_t(d^+_{\rho_1(t')} - d^+_{\rho_2(t')}). \tag{A.8}$$

Como, por hipótese, os conjuntos P_N e P_P são unitários, imediatamente temos que $\rho_1(t) = \rho_2(t')$ e $\rho_2(t) = \rho_1(t')$. Colocando em evidência os termos, a expressão A.8 pode ser reescrita com

$$Z_{\rho_1} - Z_{\rho_2} = (f^-_t - f^-_{t'})(d^+_{\rho_1(t)} - d^+_{\rho_2(t)}). \tag{A.9}$$

Sabendo-se que os vetores D^+ e F^- são ordenados, temos $(d^+_{\rho_1(t)} - d^+_{\rho_2(t)}) \leq 0$ e para $t < t' \Rightarrow Z_{\rho_1} \leq_l Z_{\rho_2}$. Analogamente, $t' < t \Rightarrow Z_{\rho_2} \leq_l Z_{\rho_1}$. Conclui-se que ρ_1 e ρ_2 são livremente comparáveis. □

Lema A.4 *Sejam ρ_1 e ρ_2 em Π_N. Se $(\rho_1, \rho_2) \in W'$ então $Z_{\rho_1} \leq_l Z_{\rho_2}$.*

Prova: Se $(\rho_1, \rho_2) \in W \subset W'$, do Lema A.2, $Z_{\rho_1} \leq_l Z_{\rho_2}$.

Consideremos $(\rho_1, \rho_2) \in (W' - W) \rightarrow |\Im(\rho_2)| - |\Im(\rho_1)| = 1$.

Suponhamos por contradição que $Z_{\rho_2} \leq_l Z_{\rho_1}$. Daí, $\exists \beta_c$ tal que $\beta_c(N^j_{t'}) = P^j_{t'}$ com $t \in P_N$; $t' \in P_P$; $i = 0, \ldots, \rho_2(t) - \rho_1(t) - 1$ e $j = 0, \ldots, \rho_1(t') - \rho_2(t') - 1$, que induz pares ordenados (t, t') com $t' < t \forall t \in P_N$ e $\forall t' \in P_P$. Pelo Lema A.3, os conjuntos $P_N = t$ e $P_P = t'$ são unitários, em outras palavras diferem de apenas 2 elementos. Sejam tais permutações P_N e P_P.

$$\ldots \rho_1(t') \ldots \rho_1(t) \ldots$$
$$\ldots \rho_2(t') \ldots \rho_2(t) \ldots$$

e por definição dos conjuntos segue-se

$$\rho_1(t') > \rho_2(t') \text{ e } \rho_1(t) < \rho_2(t).$$

Como essas permutações diferem apenas de 2 elementos, é fácil ver que

$$\rho_1(t') = \rho_2(t) > \rho_1(t),$$
$$\rho_2(t') = \rho_1(t) < \rho_2(t).$$

Quando o número de inversões é calculado, tem-se que $|\Im(\rho_1)| > |\Im(\rho_2)|$. Contradição pois temos a hipótese $|\Im(\rho_2)| - |\Im(\rho_1)| = 1$. □

Teorema A.3 *Sejam ρ_i e ρ_j vértices do grafo G'_{inv}, ligados por um caminho de arcos em W'. Tem-se $|\Im(\rho_i)| < |\Im(\rho_j)|$, se e somente se, $Z_{\rho_i} \leq_l Z_{\rho_j}$.*

Prova: A demonstração deste lema é análoga ao do Teorema 3.2, claro que neste caso os arcos pertencem a W'. □

Finalmente o Teorema que é o objetivo do capítulo 3.

Teorema A.4 *Sendo o grafo* $\widehat{G'_{inv}}$ *o fecho transitivo de* $G'_{inv} = (\Pi_N, W')$ *e* $G_{Liv} = (\Pi_N, M)$ *o grafo de comparabilidade do Poset* $(\Pi_N, \leq_l)$. *Tem-se que* $\widehat{G'_{inv}} = G_{Liv}$.

Prova: ($\subseteq$) $\widehat{G'_{inv}}$ é um subgrafo de G_{Liv} pois possui o mesmo conjunto de nós e pelo Teorema A.3, tem-se que $W' \subseteq M$, onde $M = \{(\rho_i, \rho_j)/\rho_i \leq_l \rho_j, i, j = \{1, \ldots, N!\}\}$.

($\supseteq$) Agora devemos mostrar que $G_{Liv} \subseteq \widehat{G'_{inv}}$, ou seja, dadas duas permutações ρ_i e ρ_j em Π_N, com $Z_{\rho_i} \leq_l Z_{\rho_j}$ e $|\Im(\rho_i)| - |\Im(\rho_j)| = m$, deve sempre existir entre eles um caminho em que os m arcos pertencem a W'.

Suponha por contradição que, para todos os caminhos entre ρ_i e ρ_j, existe pelo menos um arco tal que $(\rho''_i, \rho''_j) \notin W'$. Considerem os caminhos parciais dados por $\mu_1 = (\rho_i, \rho'_i, \rho''_i)$ e $\mu_2 = (\rho''_j, \rho'_j, \rho_j)$, tais que $\{(\rho_i, \rho'_i), (\rho'_i, \rho''_i), (\rho''_j, \rho'_j), (\rho'_j, \rho_j)\} \subseteq W'$.

Segue-se do Teorema A.3 que

$$Z_{\rho_i} \leq_l Z_{\rho'_i} \leq_l Z_{\rho''_i} \Rightarrow Z_{\rho_i} - Z_{\rho''_i} \leq 0,$$

e

$$Z_{\rho''_j} \leq_l Z_{\rho'_j} \leq_l Z_{\rho_j} \Rightarrow Z_{\rho''_j} - Z_{\rho_j} \leq 0.$$

Somando as desigualdades resultantes, temos que

$$(Z_{\rho_i} - Z_{\rho''_i}) + (Z_{\rho''_j} - Z_{\rho_j}) \leq 0.$$

Como $(\rho''_i, \rho''_j) \notin W'$, tomemos uma instância do PQA tal que $Z_{\rho''_i} > Z_{\rho''_j}$ sendo assim, é possível encontrar um $K > 0$, suficientemente grande, tal que $Z_{\rho''_i} - Z_{\rho''_j} = K$ e que somando este valor a equação acima, tenha

$$(Z_{\rho_i} - Z_{\rho''_i}) + (Z_{\rho''_i} - Z_{\rho''_j}) + (Z_{\rho''_j} - Z_{\rho_j}) \geq 0,$$

que é igual a $Z_{\rho_i} - Z_{\rho_j} \geq 0$, contrariando o fato de $(\rho_i, \rho_j) \in M$. □

Bibliografia

[Ab84] Abreu, N.M.M., *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, (1984).

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

[ABGL00] Anstreicher, K.M., Brixius, N.W., Goux, J.P. e Linderoth, J., *Solving large quadratic assignment problem on computational grid*, disponível no site http://www.optimization-online.org/DB-HTML/2000/10/233.html, (2000).

[ABL04] Abreu, N.M.M., Boaventura, N.P.O. e Loiola, E.M., *Uma revisão comentada das abordagens do problema quadrático de alocação*, Pesquisa Operacional, SOBRAPO, vol. 24, 1, (2004), pp.73-110.

[AG98] Almasi, G. e Gottlieb, A., *Highly Parallel Computing*, Benjamom-Cummings Publishing Co., 1998.

[AB01] Anstreicher, K.M. and Brixius, N.W, *A new bound for the quadratic assignment problem based on convex quadratic programming*, Mathematical Programming, 89(3), pp.341-357.

[BCPP98] Burkard, R.E., Çela, E., Pardalos, P.M. e Pitsoulis, L.S., *The quadratic assignment problem.* In Handbook of Combinatorial Optmization, vol. 3, ZHU and P.M. Pardalos, Eds., Kluwer, pp. 241-337 (1998).

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

[BR83] Burkard, R.E. e Rendl, F., *A termodynamically motivated simulation procedure for combinatorial optimization problems*, European Journal of Operational Research, vol. 17, (1983), pp. 169-174.

[BS78] Burkard, R.E. e Stratman, *Numerical investigations on quadratic assignment problems*, Naval Research Logistics Quartely, vol. 25 (1978), pp. 129-148.

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

[CLRV01] Cormen, T.H., Leiserson, C.E., Rivest, R.L. e Stein, C., *Introduction to Algorithms*, The Massachusetts Institute of Technology, (2001), ed. 2.

[FF94] Fleurent, C. e Ferland, J.A., *Genetic hybrids for the quadratic assignment problem*, DIMACS Series Discr. Math. Theor. Comp. Sci., vol. 16 (1994), pp. 173-187.

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

[Gi62] Gilmore, P.C., *Optimal and suboptimal algorithms for the quadratic assignment problem*, SIAM Journal on Applied Mathematics, vol. 10 (1962), pp. 305-313.

[HIAI02] Hasegawa, M., Ikeguchi, T., Aihara, K. e Itoh, K., *A novel chaotic search for quadratic assignment problems*, European Journal of Operational Research, 2002, pp. 543-556.

[KB57] Koopmans, T.C. e Beckmann, M.J., *Assignment problems and the location of economics activities*, Econometrica, vol. 25 (1957), pp. 53-76.

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

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

[LM88] Laporte, G. e Mercure, H., *Balancing hydraulic turbine runners: A quadratic assignment problem*, European Journal of Operational Research, (1988), vol. 35, pp. 378-382.

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

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

[MPI] Message Passing Interface Forum, http://www.mpi-forum.org, última visita: 10/05/2004.

[PQR76] Pollatschek, M.A., Gershoni, N. e Radday, Y.T., *Optimization of the typewriter keyboard by computer simulation*, Informatik, (1976), vol. 10, pp. 438-439.

[PPR95] Pardalos, P.M., Pitsoulis, L.S. e Resende, M.G.C., *A parallel GRASP implementation for the quadratic assignment problem*, Parallel Agorithms for Irregular Problems: State of the Art, Kluwer Academic Publishers, pp.111-128, (1995).

[PRW94] Pardalos, P.M., Rendl, F. e Wolkowicz, H., *The quadratic assignment problem: a survey and recent developments*, DIMACS Series Discr. Math. Theor. Comp. Sci., vol. 16 (1994), pp. 1-42.

[QAPLIB] A Quadratic Assignment Problem Library, http://www.opt.math.tu-graz.ac.at/qaplib, última visita:10/05/2004.

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

[Ran00] Rangel, M.C., *Contribuições Algébricas ao Problema Quadrático de Alocação*, Tese de D.Sc., Programa de Engenharia de Produção - COPPE/UFRJ, Brasil, (2000).

[Ran01] Rangel, M.C., *Uma Ampliação no Espaço de Busca para Soluções do Problema Quadrático de Alocação através de uma Relaxação Linear*, Relatório Técnico, UFES - Universidade Federal do Espírito Santo, (2001).

[RA03] Rangel, M.C. e Abreu, N.M.M., *Ordenações parciais nos conjuntos das soluções dos problemas de alocação linear e quadrático*, Pesquisa Operacional, (2003), vol. 23, pp. 265-284.

[RAB00] Rangel, M.C., Abreu, N.M.M. e Boaventura-Netto, P.O., *GRASP para o PQA: um limite de aceitação para soluções iniciais*, Pesquisa Operacional, vol. 20 (2000), pp. 45-58.

[SG76] Sahni, S. e Gonzalez, T., *P-complete approximation problems*, Journal of the Association of Computing Machinery, vol. 23 (1976), pp. 555-565.

[St61] Steinberg, L., *The backboard wiring problem: a placement algorithm*, SIAM Review, vol. 3 (1961), pp. 37-50.

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

[VRA95] Vernet, O., Rodrigues, R.M.N.D. e Abreu, N.M.M., *Reticulados de Permutações*, Relatório Técnico, EP-03/95 Série P.O., Programa de Engenharia de Produção - COPPE/UFRJ, Brasil, (1995).