Carlos Jones Rebello Junior

Um Algoritmo baseado no Teorema das Inversões com Path Relinking para o Problema Quadrático de Alocação

Vitória - ES, Brasil

09 de dezembro de 2013

Carlos Jones Rebello Junior

Um Algoritmo baseado no Teorema das Inversões com Path Relinking para o Problema Quadrático de Alocação

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Informática pela Universidade Federal do Espírito Santo.

Orientadora:
Maria Cristina Rangel

DEPARTAMENTO DE INFORMÁTICA
CENTRO TECNOLÓGICO
UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Vitória - ES, Brasil

09 de dezembro de 2013

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Informática sob o título *"Um Algoritmo baseado no Teorema das Inversões com Path Relinking para o Problema Quadrático de Alocação"*, defendida por Carlos Jones Rebello Junior e aprovada em 09 de dezembro de 2013, em Vitória, Estado do Espírito Santo, pela banca examinadora constituída pelos professores:

Profª Drª Maria Cristina Rangel
Universidade Federal do Espírito Santo
Orientadora

Profª Drª Maria Claudia Silva Boeres
Universidade Federal do Espírito Santo

Prof. Dr. Landro Colombi Resendo
Instituto Federal do Espírito Santo

Resumo

O Problema Quadrático de Alocação (PQA) é um problema clássico de otimização combinatória com inúmeras aplicações dentre elas, a localização de facilidades, particionamento de grafos, clique máxima, *scheduling* e análise de dados estatísticos. O PQA é um problema com alto grau de dificuldade a ponto de instâncias de tamanho maiores que 30 já serem consideradas impraticáveis. Neste trabalho, o PQA é abordado por meio de uma relaxação linear conhecida como Problema de Alocação Linear (PAL). Utiliza-se essa abordagem porque existe na literatura o Teorema das Inversões que associa o custo de uma solução do PQA ao número de inversões de sua correspondente linear. Para otimizar a busca por soluções viáveis de boa qualidade, um rastreamento de soluções quadráticas no universo das soluções lineares é realizado através da construção de duas matrizes denotadas por $HeadQ$ e $HeadQ^*$ de dimensões $n \times (n-1)$. Combinando estas matrizes com o Teorema das Inversões, um algoritmo construtivo que gera soluções iniciais de boa qualidade é apresentado. Para isso, criou-se uma função denominada **DiminuirNumInversões** que avalia se uma troca feita entre duas posições nas linhas da matriz $HeadQ^*$ reduz o número de inversões da solução linear. Uma diversificação nas soluções encontradas pelo algoritmo é feita através do procedimento *path relinking*. Os resultados dos testes computacionais mostraram que o algoritmo desenvolvido é bastante eficiente para instâncias de dimensões até 30. Uma versão paralela do algoritmo é apresentada para tratar instâncias maiores.

Abstract

The quadratic assignment problem (QAP) is a classical combinatorial optimization problem with numerous applications, among them, the location of facilities, graph partitioning, maximum clique, scheduling and analysis of statistical data. The PQA is a problem with a high degree of difficulty to the point of instances greater than 30 are already considered impractical. In this work, the PQA is approached through a linear relaxation known as Linear Allocation Problem (LAP). This approach is used because there is in the literature the Theorem of Inversions which associates a cost QAP solution to number of inversions of its corresponding linear. To optimize the search for good viable solutions, a trace of quadratic solutions into the universe of linear solutions is accomplished by building two $n \times (n-1)$ matrices denoted by $HeadQ$ e $HeadQ^*$. Combining these matrices with the Theorem of Inversions, a constructive algorithm to generate good initial solutions is presented. For this, we created a function called **Diminuir-NumInversões** that evaluates whether an exchange made between two positions on the lines of $HeadQ^*$ matrix reduces the number of inversions of the linear solution. Diversification in the solutions found by the algorithm is done by path relinking procedure. The computational tests results showed that the algorithm developed is fairly efficient for instances of up to 30 dimensions. A parallel version of the algorithm is presented for treating bodies of larger scale.

Dedicatória

Dedico este trabalho a minha esposa Luciana Camizão Rebello e a minha orientadora Maria Cristina Rangel, que acreditaram em mim e me apoiaram em todos os momentos, a todos familiares, amigos e demais professores que contribuiram de alguma forma para que sua conclusão fosse possível.

Agradecimentos

Agradeço primeiramente a Deus pela capacidade física e intelectual para realização deste trabalho.

À minha esposa Luciana por estar sempre ao meu lado me incentivando.

Também agradeço às professoras Maria Cristina Rangel e Maria Claudia Boeres pelos ensinamentos nesta jornada e por tornarem o desenvolvimento da pesquisa mais agradável. Aos demais professores do Programa de Pós-Graduação em Informática pela importância na minha formação acadêmica.

Aos meus colegas de trabalho da Coordenadoria do Curso Técnico de Automação Industrial do Instituto Federal do Espírito Santo *campus* Linhares, ao colega Judismar Arpini e demais colegas do Laboratório de Otimização, que estiveram ao meu lado durante esse jornada.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

Neste capítulo o Problema Quadrático de Alocação é apresentado com uma breve descrição de sua origem, as principais formulações matemáticas propostas para este problema e alguns trabalhos existentes na literatura.

1.1 Problema Quadrático de Alocação

Introduzido por Koopmans e Beckmann (1957), no contexto de atividades econômicas, o Problema Quadrático de Alocação (PQA) tem como objetivo determinar a alocação ótima de pares de atividades a pares de localidades, dadas as distâncias entre as n localidades e o fluxo entre as n atividades.

Uma das aplicações mais importantes do PQA reside nos problemas de *Layout*, também conhecidos como problemas de arranjos físicos, onde se deseja instalar facilidades em locais previamente determinados e demandas entre eles. Os custos para este problema correspondem a soma dos produtos das distâncias das localidades, pelo fluxo das facilidades. Como exemplos para esta abordagem, pode-se citar: o trabalho de Elshafei (1977) que discute a localização de departamentos de um hospital no Cairo; o de McCormik (1970) que trata o problema de otimizar o teclado de máquinas de escrever e o trabalho de Steinberg (1961) com o estudo da disposição de componentes eletrônicos em placas de circuitos impressos.

De acordo com Sahni e Gonzalez (1976), o PQA é um dos problemas mais difíceis da classe *NP-hard*, por possuir um alto grau de combinatoriedade. Para Burkard, Karisch e Rendl (1997) problemas de dimensão $n \geq 20$ já são considerados de grande porte mesmo para os avançados recursos computacionais existentes hoje. Anstreicher et al. (2000) consideram problemas de dimensão $n \geq 30$ impraticáveis em termos de tempo computacional aceitável. Instâncias como Nug30, Kra30a, Kra30b e outras disponíveis na *Quadratic Assignment Problem Library* (QAPLIB), Burkard et al. (2013), permaneceram por décadas sem que a solução ótima fosse comprovada, até que em 2000 foram resolvidas, de forma ótima, através de um algoritmo

branch and bound paralelo proposto por Anstreicher et al. (2000). Para este algoritmo, por exemplo, Anstreicher et al. (2000) apresentaram a solução ótima para a instância Kra30b com um tempo computacional de 182 dias, usando uma única estação de trabalho. Para a solução da instância Nug30, uma média de 650 estações de trabalho em rede foram utilizadas ao longo de um período de uma semana, fornecendo o equivalente a cerca de 7 anos de cálculo de apenas uma destas estações.

Diversos problemas de otimização combinatória podem ser tratados como casos particulares do PQA, dentre eles destacam-se o Problema do Caixeiro Viajante de Gutin e Yeo (2002), problemas envolvendo Clique Máxima citados por Burkard et al. (1998) e o Problema de Isomorfismo de Grafos reformulado como PQA por Lee (2007).

Como primeiras contribuições teóricas para o Problema Quadrático de Alocação, temos o trabalho de Gilmore (1962) que introduz um limite inferior para o PQA e o trabalho de Lawler (1963) que faz uma formulação mais genérica do trabalho de Koopmans e Beckmann (1957), além de modificar o limite de Gilmore (1962) criando um dos mais importantes limites inferiores para PQA, conhecido como limite de **Gilmore-Lawler**.

Além das contribuições citadas pode-se destacar na literatura outros textos que são dedicados ao PQA, como exemplos tem-se Pardalos, Rendl e Wolkowicz (1994) com a publicação do primeiro livro dedicado ao PQA que contém artigos relacionados a limite inferior, complexidade computacional, aproximações heurísticas e generalizações sobre o problema e o livro de Cela (1998). Mais recentemente Loiola et al. (2007) escreveram um *survey* sobre o PQA.

Tão importante quanto os textos já citados e que tiveram influência direta, tanto na parte algébrica como na fase de implementação deste trabalho, estão a tese de doutorado de Abreu (1984), seguida da tese de doutorado de Rangel (2000) e seu relatório técnico Rangel (2001), e mais recentemente, as dissertações de mestrado de Resendo (2004), Simoes (2006) e Lee (2007).

1.2 Principais formulações para o PQA

Na literatura o PQA é abordado de várias formas por diferentes autores. A seguir apresentam-se algumas da principais formulações adotadas para solução de problemas.

A primeira formulação dada ao PQA foi apresentada por seus precursores, Koopmans e Beckmann (1957), com os custos da função objetivo baseados no contexto econômico. Os custos c_{ijkl} eram dados pelo produto $f_{ij}d_{kl}$, onde $F = [f_{ij}]$ e $D = [d_{kl}]$ são matrizes reais $n \times n$

que representam, respectivamente, os fluxos entre as atividades i e j e as distâncias entre as localidades k e l.

Lawler (1963) introduziu duas novas ideias para a formução do PQA, a primeira foi o PQA genérico que trata da alocação de pares (i, j) de n atividades a pares (k,l) de n localidades, tratando-se de um problema de minimização expresso por:

$$min \sum_{i,j,k,l=1}^{n} c_{ijkl} x_{ik} x_{jl}, \tag{1.1}$$

sujeito a

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

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

onde $x_{ik} = 1$ se a facilidade i for alocada em k e $x_{ik} = 0$ caso contrário. As restrições 1.2 e 1.3 garantem que apenas uma atividade i é alocada um local k. É importante perceber que os custos aqui tratados por c_{ijkl}, são similares aos produtos das matrizes $n \times n$ reais F e D trabalhadas por Koopmans e Beckmann (1957).

A segunda proposta de Lawler (1963) foi uma formulação linear através do uso de variáveis binárias $y_{ijkl} = x_{ik} x_{jl}$ e custo c_{ijkl}, obtendo assim :

$$min \sum_{i,j,k,l,=1}^{n} c_{ijkl} y_{ijkl}, \tag{1.4}$$

sujeito a

$$\sum_{i=1}^{n} x_{ik} = 1, \qquad 1 \leq k \leq n \tag{1.5}$$

$$\sum_{i,j,k,l=1}^{n} y_{ijkl} = n^2, \quad 1 \leq i, j, k, l \leq n \tag{1.6}$$

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

$$x_{ik} + x_{jl} - 2y_{ijkl} \geq 0, \quad 1 \leq i, j, k, l \leq n \tag{1.8}$$

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

O modelo usado nesse trabalho apresenta uma solução através de permutações. Esse modelo foi proposto por Burkard e Stratman (1978), da seguinte forma:

$$\min_{\varphi \in \Pi_n} \sum_{i,j=1}^{n} f_{ij} d_{\varphi(i)\varphi(j)}, \tag{1.9}$$

onde $1 \leq i, j \leq n$ e Π_n é o conjunto de todas as permutações φ de $\Omega^n = \{1, \ldots, n\}$.

1.3 Algoritmos Exatos e Heurísticos

Os estudos sobre o PQA basicamente dividem-se em dois grupos: algoritmos exatos ou ótimos e algoritmos heurísticos ou sub-ótimos. Em ambas as estratégias de estudo citadas, estão embutidos o estudo de limites inferiores para o custo da solução ótima do PQA.

As primeiras contribuições teóricas da introdução destes limites foram dadas por Gilmore (1962). Lawler (1963), em uma formulação mais genérica que Koopmans e Beckmann (1957), modificou o limite de Gilmore (1962) criando o chamado limite de Gilmore-Lawler ainda muito utilizado no estudo do PQA.

Quando se fala em métodos exatos, uma grande dificuldade a destacar é a necessidade de poderosas plataformas computacionais disponíveis e uma grande quantidade de memória para armazenamento. Como exemplos pode-se citar Anstreicher et al. (2000) que apresentam a solução ótima para a instância Kra30b com um tempo computacional de 182 dias com o uso de apenas uma estação de trabalho e também Anstreicher et al. (2000) com instância Nug30 resolvida de forma ótima após um período de 7 dias, porém com um conjunto de 650 computadores em rede ao redor do mundo, veja seção 1.1.

Dadas as limitações apresentadas pelos algoritmos exatos, uma grande quantidade de pesquisas surgiram voltadas ao estudo dos algoritmos heurísticos, em especial, as meta-heurísticas.

Na literatura pode-se citar como exemplos o *Greedy Randomized Adaptive Search Procedures* (GRASP), proposto por Li, Pardalos e Resende (1994), que utilizam algoritmos construtivos associados a técnicas de melhoramento, a Busca Tabu (BT) em paralelo de James, Rego e Glover (2009) com o objetivo de aproveitar o ambiente paralelo para aumentar a intensificação e diversificação estratégica do algoritmo. Existem ainda as meta-heurísticas aplicadas ao PQA que utilizam como estratégia de melhoramento os fenômenos naturais, como o *Simulated Annealing* (SA) desenvolvido por Burkard e Rendl (1983). Estão também na literatura Seyedkashi et al. (2010), que propõem um algoritmo guloso para melhorar o SA. Tate e Smith (1995) desenvolveram um Algoritmo Genético (AG) aplicado ao PQA, além de Misevicius (2003) que

implementa um Algoritmo Genético híbrido com um procedimento chamado *ruin and recreate* (*R and R*), onde uma solução existente é "destruída" e logo em seguinda passa por um processo de reconstrução, uma ideia similar a uma busca local. Existe ainda uma meta-heurística que simula o comportamento de colônias de formigas, *Ant Colony*, desenvolvida por Maniezzo, Colorni e Dorigo (1994) e Gambardella, Taillard e Dorigo (1997). Mais recentemente uma meta-heurística baseada na formação em V de vôo de aves migratórias, proposta por Duman, Uysal e Alkaya (2012), *Migrating Birds Optimization*, que mostra-se muito eficiente quando comparada às meta-heurísticas conhecidas.

1.4 Objetivos

Este trabalho apresenta um algoritmo que utiliza o Teorema das Inversões, Rangel (2000), e as matrizes propostas por Resendo e Rangel (2006), chamadas de $HeadQ$ e $HeadQ^*$, que mapeiam as soluções quadráticas do PQA no universo das soluções lineares da relaxação do PQA caracterizada pelo Problema de Alocação Linear (PAL). O algoritmo procura gerar soluções iniciais de boa qualidade e, como fase de melhoramento, aplica-se uma busca local e *path relinking*. Uma versão paralela do algoritmo foi desenvolvida para tratar instâncias com $n > 30$.

1.5 Organização do Texto

O trabalho está organizado da seguinte forma: no capítulo 2 descreve-se o Problema Quadrático de Alocação e sua relaxação linear na forma do Problema de Alocação Linear com o estudo da viabilidade das soluções quadráticas no universo de soluções lineares. No capítulo 3 apresentam-se os dois principais teoremas que foram usados como embasamento teórico para a construção do algoritmo desenvolvido neste trabalho. Na capítulo 4 são descritas algumas técnicas de uso comum existentes na literatura para a construção e refinamento das soluções geradas pelo algoritmo para a solução do problema. O capítulo 5 é dedicado à apresentação detalhada da evolução do algoritmo desenvolvido desde o início de sua concepção, passando por várias fases de melhoria, até a sua versão final utilizada para execução e avaliação dos testes computacionais, descritos no capítulo 6. Por fim, o capítulo 7 apresenta as conclusões e propostas de trabalhos futuros.

2 Problema Quadrático de Alocação e sua relaxação linear

Neste capítulo discute-se o **Problema Quadrático de Alocação** (PQA) e sua relaxação através do **Problema de Alocação Linear** (PAL) mostrando suas definições, propriedades particulares e sua correspondência com o PQA.

2.1 Problema Quadrático de Alocação (PQA)

O Problema Quadrático de Alocação (PQA) foi introduzido por Koopmans e Beckmann (1957) no contexto de localização de atividades econômicas com o objetivo de determinar a alocação ótima de pares de atividades a pares de localidades, dados como entrada as distâncias entre as localidades e os fluxos entre as atividades. Para este problema, além do modelo proposto por Koopmans e Beckmann (1957), existem outros como o PQA relaxado com o uso de variáveis binárias de Lawler (1963) e o PQA genérico de Burkard e Stratman (1978) que utiliza permutações de elementos no conjunto $\Omega^n = \{1, \ldots, n\}$ para representar as alocações, vide Seção 1.2.

Neste trabalho utiliza-se como modelo o PQA proposto por Burkard e Stratman (1978) que descreve matematicamente a busca pela melhor solução da seguinte forma:

$$\min_{\varphi \in \Pi_n} \sum_{i,j=1}^{n} C_{ij\varphi(i)\varphi(j)}, \tag{2.1}$$

onde os custos da função objetivo são apresentados segundo o contexto econômico dado por Koopmans e Beckmann (1957) pelo produto $f_{ij}d_{\varphi(i)\varphi(j)}$, tal que $1 \leq i, j \leq n$, Π_n é conjunto de todas as permutações φ de $\Omega^n = \{1, \ldots, n\}$, e F = $[f_{ij}]$ e D = $[d_{kl}]$ são matrizes de fluxo e distância respectivamente.

Pode-se dizer então que resolver uma instância PQA dadas as matrizes de ordem n, F e D,

de coordenadas reais (ou inteiras) não negativas, denotada por PQA(F, D), é encontrar

$$\min_{\varphi \in \Pi_n} \sum_{i,j=1}^{n} f_{ij} d_{\varphi(i)\varphi(j)}, \tag{2.2}$$

onde Π_n é conjunto de todas as permutações φ de $\Omega^n = \{1, \ldots, n\}$.

Tomem como exemplo a instância Gavett e Plyter (1966), denotada por GP66, aqui tratada como um problema de *Layout* e visualizada através de duas cliques, K_F e K_D, uma de fluxo e outra de distância que são definidas pelas matrizes da Figura 2.1. Chama-se φ uma permutação qualquer que represente uma sobreposição dos nós. Vale ressaltar que uma sobreposição de nós possui uma sobreposição de arestas. O somatório dos produtos de $f_{ij} d_{\varphi(i)\varphi(j)}$ das arestas sobrepostas é o valor da função objetivo 2.2. A Figura 2.2 apresenta uma solução viável para o GP66, onde o custo é igual a 904, representada por $\varphi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \end{pmatrix}$. Para este problema a solução ótima é $\varphi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 3 & 2 \end{pmatrix}$ cujo custo é 806.

$$\mathrm{F} = \begin{vmatrix} 0 & 28 & 25 & 13 \\ 28 & 0 & 15 & 4 \\ 25 & 15 & 0 & 23 \\ 13 & 4 & 23 & 0 \end{vmatrix} \qquad \mathrm{D} = \begin{vmatrix} 0 & 6 & 7 & 2 \\ 6 & 0 & 5 & 6 \\ 7 & 5 & 0 & 1 \\ 2 & 6 & 1 & 0 \end{vmatrix}$$

Figura 2.1: Matrizes F e D para o GP66

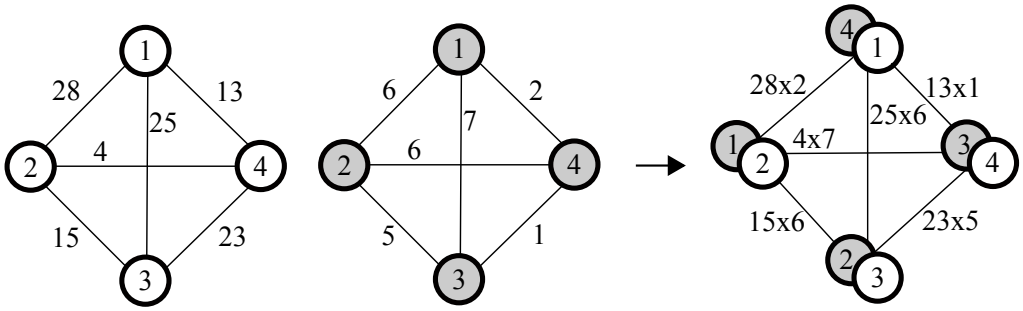

Figura 2.2: Sobreposição das cliques, uma solução viável para o problema GP66

2.1.1 Relaxação Linear do Problema Quadrático de Alocação

Considerando algumas características das matrizes F e D como simetria e diagonal principal nula, pode-se armazenar suas informações em vetores com dimensão $N = C_{n,2}$, pela ordem lexicográfica dos índices das matrizes. Para isso usa-se uma bijeção, citada por Rangel (2000),

descrita como

$$\psi(i,j) = ((i-1)n - i(i+2)/2) + j, \tag{2.3}$$

que associa a cada par $(i,j) \in n \times n$ com $i < j$ um natural $z \in \{1,\ldots,N\}$ que representa uma aresta das cliques. Para o exemplo das matrizes F e D da instância GP66 têm-se, após a bijeção, os vetores $\mathbf{F} = \begin{pmatrix} 28 & 25 & 13 & 15 & 4 & 23 \end{pmatrix}$ e $\mathbf{D} = \begin{pmatrix} 6 & 7 & 2 & 5 & 6 & 1 \end{pmatrix}$.

Considerando agora apenas as sobreposições de arestas, estabelece-se uma relaxação linear através do Problema de Alocação Linear (PAL), que pode ser definido como:

$$\min_{\xi \in \Pi_N} \sum_{i=1}^{N} f_i d_{\xi(i)}, \tag{2.4}$$

onde $N = C_{n,2}$ e Π_N é o conjunto de todas as permutações ξ de $\Omega^N = \{1,\ldots,N\}$. O PAL é considerado uma relaxação de um PQA no sentido de o conjunto de soluções viáveis do primeiro conter as do segundo. O número de soluções lineares, $N!$, é bem maior que o número de soluções quadráticas, $n!$, consequentemente, nem sempre uma permutação de arestas pode representar uma permutação de vértices. Tais soluções são ditas não-viáveis para o PQA.

Considere agora as matrizes $Q = \mathbf{F}^t\mathbf{D}$ e $Q^* = (F^-)^t D^+$, Figura 2.3, onde F^- e D^+ são vetores formados pela ordenação não-crescente e não-decrescente de **F** e **D** respectivamente. Para encontrar a solução ξ ótima do PAL(Q) é muito simples, visto que esta solução coincide com o traço da matriz Q^*, soma da diagonal principal, além disso este é um limite inferior para o PQA(F,D) definido por **F** e **D**. Vale ressaltar que o problema PAL(Q^*) é um problema isomorfo ao PAL(Q), pois o conjunto de soluções viáveis do PAL(Q^*) é o mesmo que o PAL(Q) e possuem valores das soluções ótimas iguais. Denota-se por ξ a solução do PAL(Q) e por ρ a solução do PAL(Q^*).

Q	6	7	2	5	6	1
28	168	196	56	140	168	**28**
25	150	175	**50**	125	150	25
13	78	91	26	65	**78**	13
15	**90**	105	30	75	90	15
4	24	**28**	8	20	24	4
23	138	161	46	**115**	138	23

Q^*	1	2	5	6	6	7
28	**28**	56	140	168	168	196
25	25	**50**	125	150	150	175
23	23	46	**115**	138	138	161
15	15	30	75	**90**	90	105
13	13	26	65	78	**78**	91
4	4	8	20	24	24	**28**

Figura 2.3: Matrizes Q e Q* para o GP66

Dada uma permutação $\rho \in \Pi_N$, solução do PAL(Q^*), pode-se estabelecer uma relação com $\xi \in \Pi_N$, solução do PAL(Q), através da Equação (2.3).

Para isso armazenam-se as trocas feitas nas posições dos vetores durante a ordenação de

F e **D** em permutações auxiliares denotadas por ϕ_F e ϕ_D, Rangel (2000). Para o exemplo da instância GP66 tem-se,

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

Tendo as permutações auxiliares ϕ_F e ϕ_D, e a permutação ρ, obtém-se ξ através da relação

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

Admitindo uma permutação ρ como sendo o traço da matriz Q^*, utiliza-se a permutação identidade $\rho = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}$. Aplica-se a composição da relação 2.5 da esquerda para a direita obtendo-se assim,

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

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

Se dada uma permutação ρ, solução do PAL(Q^*), for possível construir φ, solução do PQA(F,D), através do caminho inverso ao da relaxação, diz-se então que há uma solução linear viável para o PQA(F,D). Para $\rho \in \Pi_N$ ser viável é necessário que exista uma $\varphi \in \Pi_n$ que satisfaça a seguinte sequência de igualdades:

$$f_r^- d_{\rho(r)}^+ = f_{\phi_F^{-1}(r)} d_{\phi_D^{-1}(\rho(r))} = f_p d_{\xi(p)} = f_{\psi^{-1}(p)} d_{\psi^{-1}(\xi(p))} = f_{ij} d_{kl}, \tag{2.6}$$

tal que $r,p = \{1,\ldots,N\}$; $\varphi(i) = k$ e $\varphi(j) = l$; onde $i,j,k,l = \{1,\ldots,n\}$. Caso contrário, a solução ρ é não-viável para o PQA(F,D).

O Algoritmo 1, SolViável, proposto por Rangel (2000), apresentado a seguir, verifica a viabilidade de uma permutação ρ para a geração de uma φ viável.

Algorithm 1 SolViável

Entrada: ρ, ϕ_F e ϕ_D^{-1}
$\xi = \phi_D^{-1} \circ \rho \circ \phi_F$
for t = 1,... , N **do**
  $ListaIJ[t] \leftarrow \psi^{-1}(t)$ e $ListaKL[t] \leftarrow \psi^{-1}(\xi(t))$ //origem e dest. das arestas sobrepostas (i,j) e (k,l)
end for
if $\exists$ p $\in$ 1, ... , n tal que $\varphi(1) = p$ para as (n - 1) primeiras combinações **then**
  $\varphi(1)$ =p
  for i = 2, ... , n **do**
    $[\varphi(i) = l \neq p] \vee [\varphi(i) = k \neq p]$ para $ListaKL[i-1]$
  end for
  if φ construída possui todas as N combinações compatíveis **then**
    ρ é viável
  else
    ρ não é viável
  end if
else
  ρ não é viável
end if

Com o propósito de ilustrar o Algoritmo 1 SolViável, apresentam-se 2 exemplos para a instância GP66 citados por Rangel (2000). Para estes exemplos, tem-se $n = 4$ e $N = 6$. A permutações ϕ_F e ϕ_D, responsáveis por armazenar as posições das trocas dos vetores **F** e **D** após as ordenação de F^- e D^+, são as mesmas apresentadas nesta seção.

Neste algoritmo, duas listas são construídas: *ListaIJ* e *ListaKL*. Estas listas armazenam os nós de origem e destino das arestas sobrepostas através da permutação ξ, considerando a aresta (i, j) sobre a aresta (k, l). Neste contexto, uma permutação $\rho \in \Pi_N$ para ser considerada viável necessita que todas as N combinações dos vértices das arestas pertencentes às *ListaIJ* e *ListaKL* sejam compatíveis com a sobreposição dos vértices determinados por $\varphi \in \Pi_n$.

Exemplo 2.1 *Considere a permutação* $\rho_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 3 & 6 & 5 & 4 \end{pmatrix} \in \Pi_6$.

Calculando-se ξ_{ρ_1} através da composição $\xi_{\rho_1} = \phi_D^{-1} \circ \rho \circ \phi_F$, tem-se:

$$\xi_{\rho_1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 3 & 4 & 1 & 5 & 2 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 3 & 6 & 5 & 4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 5 & 4 & 6 & 3 \end{pmatrix}$$

$$\xi_{\rho_1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 6 & 5 & 2 & 1 & 4 \end{pmatrix}.$$

Utiliza-se a Tabela 2.1 a seguir para a verificação da viabilidade da solução. Conhecendo-se ξ_{ρ_1} aplica-se ψ^{-1}, a partir da qual gera-se as combinações dos vértices.

Tabela 2.1: Tabela do Exemplo 2.1

t	*ListaIJ* $\psi^{-1}(t)$	$\xi(t)$	*ListaKL* $\psi^{-1}(\xi(t))$	Combinações de Vértices
1	$(1,2)$	3	$(1,4)$	$(\varphi(1)=1 \wedge \varphi(2)=4) \vee (\varphi(1)=4 \wedge \varphi(2)=1)$
2	$(1,3)$	6	$(3,4)$	$(\varphi(1)=3 \wedge \varphi(3)=4) \vee (\varphi(1)=4 \wedge \varphi(3)=3)$
3	$(1,4)$	5	$(2,4)$	$(\varphi(1)=2 \wedge \varphi(4)=4) \vee (\varphi(1)=4 \wedge \varphi(4)=2)$
4	$(2,3)$	2	$(1,3)$	$(\varphi(2)=1 \wedge \varphi(3)=3) \vee (\varphi(2)=3 \wedge \varphi(3)=1)$
5	$(2,4)$	1	$(1,2)$	$(\varphi(2)=1 \wedge \varphi(4)=2) \vee (\varphi(2)=2 \wedge \varphi(4)=1)$
6	$(3,4)$	4	$(2,3)$	$(\varphi(3)=2 \wedge \varphi(4)=3) \vee (\varphi(3)=3 \wedge \varphi(4)=2)$

Seguindo-se o Algoritmo 1 SolViável, nota-se que as três primeiras linhas da Tabela 2.1 determinam uma $\varphi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 3 & 2 \end{pmatrix}$, para saber se esta permutação é viável, precisa-se testar as combinações para as $N-(n-1)$ linhas restantes da Tabela 2.1. Neste caso verificam-se nos testes restantes que:

- $((\varphi(2)=1)$ E $(\varphi(3)=3))$ OU $((\varphi(2)=3)$ E $(\varphi(3)=1))-$ a primeira condição é verdadeira o que torna a expressão verdadeira;
- $((\varphi(2)=1)$ E $(\varphi(4)=2))$ OU $((\varphi(2)=2)$ E $(\varphi(4)=1))-$ a primeira condição é verdadeira o que torna a expressão verdadeira;
- $((\varphi(3)=2)$ E $(\varphi(4)=3))$ OU $((\varphi(3)=3)$ E $(\varphi(4)=2))-$ a primeira condição é falsa, porém, a segunda condição é verdadeira, o que torna a expressão verdadeira.

Como todos os testes restantes são verdadeiros, diz-se, pelo Algoritmo 1 SolViável, que ρ_1 é uma solução **viável**.

Exemplo 2.2 *Dada a permutação* $\rho_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 1 & 6 & 5 & 3 & 2 \end{pmatrix} \in \Pi_6$.

Calcula-se a correspondente $\xi_{\rho_2} = \phi_D^{-1} \circ \rho \circ \phi_F$ e obtém-se $\xi_{\rho_2} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 6 & 4 & 5 & 3 & 2 \end{pmatrix}$.

Note que agora não é possível construir uma φ para as $n-1$ primeiras linhas da Tabela 2.2. Neste caso, diz-se, pelo Algoritmo 1 SolViável, que ρ_2 é uma solução **não viável**.

Tabela 2.2: Tabela do Exemplo 2.2

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

3 A Base Teórica

Neste capítulo são apresentados dois teoremas, o Teorema da Ordenação Parcial Livre (TOPL) e o Teorema das Inversões (TI), Rangel (2000). O primeiro teorema, TOPL, trata as permutações livremente comparáveis, isto é, pode-se avaliar o custo das permutações sem o conhecimento prévio das componentes dos vetores. O segundo teorema, TI, diz que os custos de permutações livremente comparáveis são diretamente propocionais aos seus números de inversões. Baseado no Teorema das Inversões pode-se gerar soluções de boa qualidade para o algoritmo desenvolvido neste trabalho, o Algoritmo baseado no Teorema das Inversões com *Path Relinking* (ATIPR).

3.1 O 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**. Considere também que os produtos escalares $Z_{\rho_1} = \sum_{i=1}^{N} f_i^- d_{\rho_1(i)}^+$ e $Z_{\rho_2} = \sum_{i=1}^{N} f_i^- d_{\rho_2(i)}^+$ representam os custos de duas soluções do PAL(Q^*), onde $\rho_1,\ \rho_2 \in \Pi_N$ permutações do conjunto $\Omega^N = \{1,\ldots,N\}$.

Dadas duas permutações ρ_1 e $\rho_2 \in \Pi_N$, seria possível concluir que seus custos $Z_{\rho_1} \leq Z_{\rho_2}$ ou $Z_{\rho_2} \leq Z_{\rho_1}$ sem conhecimento prévio das componentes dos vetores?

Considere as permutações $\rho_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 4 & 1 & 6 & 2 & 3 \end{pmatrix}$ e $\rho_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 2 & 1 & 6 & 3 & 5 \end{pmatrix}$. Fazendo a operação $\rho_1(t) - \rho_2(t)$ para $t \in \{1,\ldots,6\}$, tem-se:

- $t = 1 \Rightarrow \rho_1(1) - \rho_2(1) = 5 - 4 > 0$
- $t = 2 \Rightarrow \rho_1(2) - \rho_2(2) = 4 - 2 > 0$
- $t = 3 \Rightarrow \rho_1(3) - \rho_2(3) = 1 - 1 = 0$
- $t = 4 \Rightarrow \rho_1(4) - \rho_2(4) = 6 - 6 = 0$

- $t = 5 \Rightarrow \rho_1(5) - \rho_2(5) = 2 - 3 < 0$
- $t = 6 \Rightarrow \rho_1(6) - \rho_2(6) = 3 - 5 < 0$

Observando os termos não-positivos, não-negativos e nulos, pode-se particionar os elementos $t \in \Omega^N$ nos seguintes subconjuntos disjuntos:

- $P_N = \{t \in \Omega^N / \rho_1(t) - \rho_2(t) < 0\}$, termos não-positivos;
- $P_P = \{t' \in \Omega^N / \rho_1(t') - \rho_2(t') > 0\}$, termos não-negativos;
- $P_0 = \{t'' \in \Omega^N / \rho_1(t'') - \rho_2(t'') = 0\}$, termos livremente nulos;

Para o exemplo em questão tem-se: $P_N = \{5,\ 6\}$, $P_0 = \{3,\ 4\}$ e $P_P = \{1,\ 2\}$.

Quando se faz a operação $Z_{\rho_1} - Z_{\rho_2}$, obtém-se:

$$Z_{\rho_1} - Z_{\rho_2} = \sum_{t=1}^{N} f_t^- d_{\rho_1(t)}^+ - \sum_{t=1}^{N} f_t^- d_{\rho_2(t)}^+ = \sum_{t=1}^{N} f_t^- (d_{\rho_1(t)}^+ - d_{\rho_2(t)}^+) \tag{3.1}$$

e separando nas partições, tem-se:

$$Z_{\rho_1} - Z_{\rho_2} = \sum_{t \in P_N} f_t^- (d_{\rho_1(t)}^+ - d_{\rho_2(t)}^+) + \sum_{t'' \in P_0} f_{t''}^- (d_{\rho_1(t'')}^+ - d_{\rho_2(t'')}^+) + \sum_{t' \in P_P} f_{t'}^- (d_{\rho_1(t')}^+ - d_{\rho_2(t')}^+) \tag{3.2}$$

Após a separação das parcelas da equação 3.1 em não-positivas, não-negativas e livremente nulas, desenvolvem-se os somatórios referentes a cada uma dessas parcelas, exceto as livremente nulas, pois $d_{\rho_1(t'')}^+ - d_{\rho_2(t'')}^+ = 0$, sendo possível assim, eliminar os elementos de P_0. Neste trabalho a parcela $(d_{\rho_1(k)}^+ - d_{\rho_2(k)}^+)$, onde k é um elemento de P_N e P_P, será denotada por **fator-diferença**.

$$Z_{\rho_1} - Z_{\rho_2} = \sum_{t \in P_N} f_t^- (d_{\rho_1(t)}^+ - d_{\rho_2(t)}^+) + \sum_{t' \in P_P} f_{t'}^- (d_{\rho_1(t')}^+ - d_{\rho_2(t')}^+) \tag{3.3}$$

Considerando os conjuntos $P_N = (5,6)$ e $P_P = (1,2)$ construídos com as permutações citadas, tem-se:

$$Z_{\rho_1} - Z_{\rho_2} = f_5^- (d_2^+ - d_3^+) + f_6^- (d_3^+ - d_5^+) + f_1^- (d_5^+ - d_4^+) + f_2^- (d_4^+ - d_2^+) \tag{3.4}$$

É fácil ver que é possível inserir termos simétricos em cada fator diferença cujos índices não são consecutivos nas parcelas da equação 3.4 de modo que não haja alteração no seu resultado.

Após alguma manipulação chega-se à equação 3.5, chamada de Partição Canônica.

$$Z_{\rho_1} - Z_{\rho_2} = f_5^-(d_2^+ - d_3^+) + f_6^-(d_3^+ - d_4^+ + d_4^+ - d_5^+) + f_1^-(d_5^+ - d_4^+) + f_2^-(d_4^+ - d_3^+ + d_3^+ - d_2^+) \tag{3.5}$$

$$Z_{\rho_1} - Z_{\rho_2} = f_5^-(d_2^+ - d_3^+) + f_6^-(d_3^+ - d_4^+) + f_6^-(d_4^+ - d_5^+) + f_1^-(d_5^+ - d_4^+) + f_2^-(d_4^+ - d_3^+) + f_2^-(d_3^+ - d_2^+) \tag{3.6}$$

Para dar continuidade à análise da avaliação da diferença dos custos $Z_{\rho_1} - Z_{\rho_2}$, o Teorema da Ordenação Parcial Livre é mostrado a seguir, porém este se encontra enunciado formalmente em Rangel (2000) com maiores detalhes.

O **Teorema da Ordenação Parcial Livre (TOPL)** proposto em Abreu (1984) afirma que pode-se comparar os custos Z_{ρ_1} e Z_{ρ_2} independentemente dos valores das coordenadas de F^- e D^+ se e somente se, para cada $t \in P_N$ existir um $t' \in P_P$ capaz de induzir o par ordenado (t,t'), com $t < t'(t' < t)$, $\forall t \in P_N$ e $\forall t' \in P_P$. Rangel (2000) propôs uma demonstração mais simples para este teorema.

Quando se põe em evidência os fatores-diferença comuns da equação 3.5, tem-se que os índices das parcelas evidenciadas correspondentes a F^- induzem exatamente ao par ordenado (t,t') que se deseja encontrar, ou seja, $(f_t^- - f_{t'}^-)$. Logo, para se saber se todo $t < t'(t' < t)$, $\forall t \in P_N$ e $\forall t' \in P_P$, basta formar os pares ordenados (t,t') a partir desses índices e compará-los. Veja a seguir a construção dos referidos pares ordenados.

$$Z_{\rho_1} - Z_{\rho_2} = (f_5^- - f_2^-)(d_2^+ - d_3^+) + (f_6^- - f_2^-)(d_3^+ - d_4^+) + (f_6^- - f_1^-)(d_4^+ - d_5^+) \tag{3.7}$$

Tem-se então as parcelas correspondentes a F^- da equação 3.7, que induzem ao par ordenado (t,t'):

Tabela 3.1: Construção dos pares ordenados (t,t') da equação 3.7

$(f_t^- - f_{t'}^-)$	$(d^+_{\rho_{1(t)}} - d^+_{\rho_{2(t)}})$	(t,t')
$(f_5^- - f_2^-)$	$(d_2^+ - d_3^+)$	$(5,2)$
$(f_6^- - f_2^-)$	$(d_3^+ - d_4^+)$	$(6,2)$
$(f_6^- - f_1^-)$	$(d_4^+ - d_1^+)$	$(6,1)$

Construídos os pares (t,t'), é necessário verificar se $\forall t \in P_N$ e $\forall t' \in P_P$ existe $t < t'(t' < t)$. Veja que para todos os casos tem-se que $(t' < t)$, ou seja, $(2 < 5)$, $(2 < 6)$ e $(1 < 6)$, portanto

Z_{ρ_1} e Z_{ρ_2} são ditos livremente comparáveis e, além disso, pode-se afirmar que $Z_{\rho_1} \geq Z_{\rho_2}$ pois os vetores F^- e D^+ são ordenados, facilitando a análise da comparação apenas pela posição das componentes nos vetores.

A seguir o conceito de *poset* é apresentado para auxiliar na compreensão desta seção. Um conjunto parcialmente ordenado (*poset*, em inglês *partially ordered set*) é um conjunto equipado com uma relação binária de ordem parcial. Esta relação formaliza o conceito intuitivo de ordem, sequência, ou arrumação dos elementos do conjunto. Em alguns casos, todos os elementos do conjunto podem ser ordenados seguindo tal relação gerando uma ordenação total.

Estabelecidos os conceito de *poset* e custos livremente comparáveis, define-se $C(Z_\rho)$ como o conjunto de todos valores de custos das permutações $\rho \in \Pi_N$ onde cada permutação representa uma solução do PAL(Q^*). Nesse universo de $N!$ valores de custos, pode-se definir Z_{ρ_1} e Z_{ρ_2} como permutações livremente comparáveis se satisfazem o TOPL. Sendo assim, uma relação de ordem entre os elementos de $C(Z_\rho)$ é estabelecida, denotada por $\leq_l$, resultando em um conjunto parcialmente ordenado (*poset*) $P(C(Z_\rho), \leq_l)$. Define-se neste momento um outro *poset* $P_\rho(\Pi_N, \leq_l)$ no universo de $N!$ permutações de Π_N compatível com $P(C(Z_\rho), \leq_l)$. Esta compatibilidade é trivial pois existe um custo associado a cada permutação. Desta forma, um par de permutações são livremente comparáveis se pertencerem ao *poset* $P_\rho(\Pi_N, \leq_l)$.

A seguir, um exemplo da aplicação do TOPL.

Considere novas permutações $\rho_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 6 & 3 & 1 & 5 \end{pmatrix}$ e $\rho_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 2 & 6 & 4 & 1 & 5 \end{pmatrix}$.
As mesmas operações sobre as permutações são realizadas, isto é, calcular $\rho_1(t) - \rho_2(t)$ para $t \in \{1, \ldots, 6\}$.

- $t = 1 \Rightarrow \rho_1(1) - \rho_2(1) = 2 - 3 < 0$
- $t = 2 \Rightarrow \rho_1(2) - \rho_2(2) = 4 - 2 > 0$
- $t = 3 \Rightarrow \rho_1(3) - \rho_2(3) = 6 - 6 = 0$
- $t = 4 \Rightarrow \rho_1(4) - \rho_2(4) = 3 - 4 < 0$
- $t = 5 \Rightarrow \rho_1(5) - \rho_2(5) = 1 - 1 = 0$
- $t = 6 \Rightarrow \rho_1(6) - \rho_2(6) = 5 - 5 = 0$

Para este exemplo tem-se as partições: $P_N = \{1,\ 4\}$, $P_0 = \{3,\ 5,\ 6\}$ e $P_P = \{2\}$.

Fazendo a operação $Z_{\rho_1} - Z_{\rho_2}$, obtém-se:

$$Z_{\rho_1} - Z_{\rho_2} = f_1^-(d_2^+ - d_3^+) + f_4^-(d_3^+ - d_4^+) + f_2^-(d_4^+ - d_2^+) \tag{3.8}$$

Inserindo os termos simétricos em cada fator-diferença para a obtenção da Partição Canônica:

$$\begin{aligned} Z_{\rho_1} - Z_{\rho_2} &= f_1^-(d_2^+ - d_3^+) + f_4^-(d_3^+ - d_4^+) + f_2^-(d_4^+ - d_3^+ + d_3^+ - d_2^+) \\ Z_{\rho_1} - Z_{\rho_2} &= f_1^-(d_2^+ - d_3^+) + f_4^-(d_3^+ - d_4^+) + f_2^-(d_4^+ - d_3^+) + f_2^-(d_3^+ - d_2^+) \end{aligned} \tag{3.9}$$

Colocando em evidência os fatores-diferença comuns da equação para formar os pares ordenados (t,t') a partir de $(f_t^- - f_{t'}^-)$, tem-se:

$$Z_{\rho_1} - Z_{\rho_2} = (f_1^- - f_2^-)(d_2^+ - d_3^+) + (f_4^- - f_2^-)(d_3^+ - d_4^+) \tag{3.10}$$

Na Tabela 3.2 as parcelas correspondentes a F^- da equação 3.10, que induzem ao par ordenado (t,t'), são apresentadas.

Tabela 3.2: Construção dos pares ordenados (t,t') da equação 3.10

$(f_t^- - f_{t'}^-)$	$(d^+_{\rho_{1(t)}} - d^+_{\rho_{2(t)}})$	(t,t')
$(f_1^- - f_2^-)$	$(d_2^+ - d_3^+)$	$(1,2)$
$(f_4^- - f_2^-)$	$(d_3^+ - d_4^+)$	$(4,2)$

Todos os pares ordenados na Tabela 3.2 são analisados para verificar se satisfazem o TOPL. Veja que agora nem todos os casos atendem a restrição $(t < t')$, ou seja, $(4 < 2)$, é uma afirmativa falsa. Neste caso, pode-se concluir pelo TOPL que as permutações Z_{ρ_1} e Z_{ρ_2} não são livremente comparáveis.

É importante ressaltar neste momento que para afirmar que Z_{ρ_1} e Z_{ρ_2} são livremente comparáveis, deve-se existir pelo menos uma relação que associe t a um t' para formar o par (t,t'), onde $\forall t \in P_N$ e $\forall t' \in P_P$ tem-se $t < t'(t' < t)$. No entanto a unicidade dos fatores-diferença não é garatinda. Vejamos o exemplo a seguir.

Considere as permutações $\rho_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 6 & 1 & 3 & 5 \end{pmatrix}$ e $\rho_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 2 & 3 & 4 & 1 & 5 \end{pmatrix}$. Analisando os valores de $\rho_1(t) - \rho_2(t)$ para $t \in \{1,\ldots,6\}$, tem-se: $P_N = \{1,\ 4\}$, $P_0 = \{6\}$ e $P_P = \{2,\ 3,\ 5\}$.

Com os conjuntos definidos, a operação $Z_{\rho_1} - Z_{\rho_2}$ é efetuada e obtém-se:

$$Z_{\rho_1} - Z_{\rho_2} = f_1^-(d_2^+ - d_6^+) + f_4^-(d_1^+ - d_4^+) + f_2^-(d_4^+ - d_2^+) + f_3^-(d_6^+ - d_3^+) + f_5^-(d_3^+ - d_1^+) \tag{3.11}$$

Insere-se os termos simétricos de cada fator-diferença:

$$\begin{aligned} Z_{\rho_1} - Z_{\rho_2} = {} & f_1^-(d_2^+ - d_3^+ + d_3^+ - d_4^+ + d_4^+ - d_5^+ + d_5^+ - d_6^+) \\ & + f_4^-(d_1^+ - d_2^+ + d_2^+ - d_3^+ + d_3^+ - d_4^+) + f_2^-(d_4^+ - d_3^+ + d_3^+ - d_2^+) \\ & + f_3^-(d_6^+ - d_5^+ + d_5^+ - d_4^+ + d_4^+ - d_3^+) + f_5^-(d_3^+ - d_2^+ + d_2^+ - d_1^+) \end{aligned} \tag{3.12}$$

$$\begin{aligned} Z_{\rho_1} - Z_{\rho_2} = {} & f_1^-(d_2^+ - d_3^+) + f_1^-(d_3^+ - d_4^+) + f_1^-(d_4^+ - d_5^+) + f_1^-(d_5^+ - d_6^+) \\ & + f_4^-(d_1^+ - d_2^+) + f_4^-(d_2^+ - d_3^+) + f_4^-(d_3^+ - d_4^+) + f_2^-(d_4^+ - d_3^+) + f_2^-(d_3^+ - d_2^+) \\ & + f_3^-(d_6^+ - d_5^+) + f_3^-(d_5^+ - d_4^+) + f_3^-(d_4^+ - d_3^+) + f_5^-(d_3^+ - d_2^+) + f_5^-(d_2^+ - d_1^+) \end{aligned} \tag{3.13}$$

Evidenciando-se os fatores comuns da equação para formar os pares ordenados (t,t') a partir de $(f_t^- - f_{t'}^-)$, uma possível solução é apresentada na Tabela 3.3 :

Tabela 3.3: Construção dos pares ordenados (t,t') da equação 3.13

$(f_t^- - f_{t'}^-)$	$(d^+_{\rho_{1(t)}} - d^+_{\rho_{2(t)}})$	(t,t')
$(f_1^- - f_2^-)$	$(d_2^+ - d_3^+)$	$(1,2)$
$(f_1^- - f_2^-)$	$(d_3^+ - d_4^+)$	$(1,2)$
$(f_1^- - f_3^-)$	$(d_2^+ - d_3^+)$	$(1,3)$
$(f_1^- - f_3^-)$	$(d_5^+ - d_6^+)$	$(1,3)$
$(f_4^- - f_5^-)$	$(d_1^+ - d_2^+)$	$(4,5)$
$(f_4^- - f_5^-)$	$(d_2^+ - d_3^+)$	$(4,5)$
$(f_4^- - f_3^-)$	$(d_3^+ - d_4^+)$	$(4,3)$

Porém, dadas as possíveis repetições de fatores diferença, poderia-se ter optado pela construção dos pares ordenados (t,t') da Tabela 3.4.

Para se chegar a conclusão de que duas permutações ρ_1 e ρ_2 são livremente comparáveis, devem ser feitas todas as combinações possíveis de pares ordenados para se encontrar (t,t'), onde pelo menos uma dessas combinações deve atender a restrição que $\forall t \in P_N$ e $\forall t' \in P_P$ tem-se $t < t'(t' < t)$. São essas combinações que caracterizam o TOPL, porém, como demonstrado por Rangel (2000), quando tais combinações são executados em uma ordem estabelecida (controlada), contrói-se apenas uma vez o conjunto dos pares ordenados a serem avaliados. Tal conjunto é suficiente para definir se as permutações são livremente comparáveis. Maiores

Tabela 3.4: Outra construção dos pares ordenados (t,t') da equação 3.13

$(f_t^- - f_{t'}^-)$	$(d^+_{\rho_{1(t)}} - d^+_{\rho_{2(t)}})$	(t,t')
$(f_1^- - f_5^-)$	$(d_2^+ - d_3^+)$	$(1,5)$
$(f_1^- - f_2^-)$	$(d_3^+ - d_4^+)$	$(1,2)$
$(f_1^- - f_3^-)$	$(d_2^+ - d_3^+)$	$(1,3)$
$(f_1^- - f_3^-)$	$(d_5^+ - d_6^+)$	$(1,3)$
$(f_4^- - f_5^-)$	$(d_1^+ - d_2^+)$	$(4,5)$
$(f_4^- - f_2^-)$	$(d_2^+ - d_3^+)$	$(4,2)$
$(f_4^- - f_3^-)$	$(d_3^+ - d_4^+)$	$(4,3)$

detalhes podem ser encontrados em Rangel (2000).

3.2 O Teorema das Inversões

Nesta seção é apresentada a sequência de inclusão dos grafos $G_{Inv} \subset G'_{Inv} \subset G_{Liv}$, grafos direcionados, onde G_{Liv} representa o grafo de todas as permutações livremente comparáveis entre si e G_{Inv} e G'_{Inv} são grafos de inversões. Algumas propriedades desses grafos facilitaram a demonstração apresentada por Rangel (2000) do Teorema das Inversões. Este teorema é a demonstração da Conjectura das Inversões proposta por Abreu (1984) que envolve número de inversões das permutações ρ livremente comparáveis aos seus custos associados Z_ρ.

3.2.1 O Grafo das Inversões

Para a construção deste grafo, o conceito de inversões em uma permutação ρ é importante. Considere Π_N o conjunto de todas as permutações dos elementos $\Omega^N = \{1,\ 2,\ ...,\ N\}$ e $\rho \in \Pi_N$. Uma inversão em ρ é um par $(\rho(i),\rho(j))$ tal que $\rho(i) > \rho(j)$, $i < j$, com $i,\ j = \{1,\ ...,\ N\}$.

Exemplo 3.1 *Seja* $\rho_r = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 5 & 6 & 4 & 3 \end{pmatrix}$ *uma permutação de* Π_N.

Observando as imagens da permutação, destacam-se os pares de inversões na Tabela 3.5:

Nota-se pela Tabela 3.5 que podem existir mais de uma inversão, portanto, $Inv(\rho_r) = \{(\rho_r(i)),(\rho_r(j)) : \rho_r(i) > \rho_r(j),\ i < j,\ i,j = 1,\ldots,\ N\}$. A cardinalidade de $Inv(\rho_r)$, aqui denotada por $|Inv(\rho_r)|$, é o número de inversões de ρ_r. Para o Exemplo 3.1 tem-se que, $Inv(\rho_r) = \{(5,4),(5,3),(6,4),(6,3),(4,3)\}$, portanto, $|Inv(\rho_r)| = 5$.

Tabela 3.5: Pares de inversões de ρ

i	j	$(\rho_r(i), \rho_r(j))$
3	5	(5,4)
3	6	(5,3)
4	5	(6,4)
4	6	(6,3)
5	6	(4,3)

Utilizando o mesmo Exemplo 3.1, gera-se $\rho_s = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 5 & 2 & 6 & 4 & 3 \end{pmatrix}$ através da troca dos elementos $\rho_r(2)$ e $\rho_r(3)$ com $|Inv(\rho_s) - Inv(\rho_r)| = 1$.

Após as definições e análise do Exemplo 3.1, pode-se agora definir o grafo das inversões $G_{Inv}(\Pi_N, W)$, onde os nós são todas as permutações dos elementos $\Omega^N = \{1, \ldots, N\}$ e os arcos são as ligações das permutações $(\rho_r, \rho_s) \in \Pi_N \times \Pi_N$ tais que $\rho_r = \rho_s$ exceto para duas posições, i e $i+1$, onde $i = \{1, \ldots, N-1\}$ e $|Inv(\rho_s)| - |Inv(\rho_r)| = 1$. G_{Inv} possui $N!$ nós que podem ser organizados em níveis por número de inversões, iniciando com a permutação identidade $\rho_0 = (1\ 2\ 3\ 4\ \ldots N)$ finalizando com a permutação reversa $\rho_N = (N\ N-1\ \ldots 1)$.

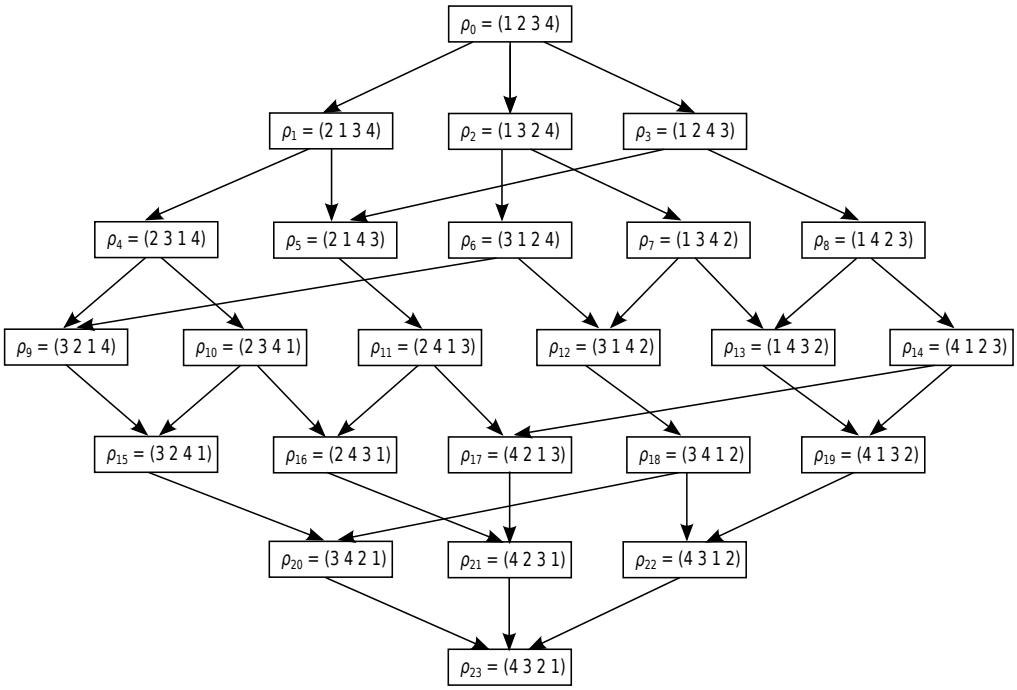

Figura 3.1: Grafo das Inversões G_{Inv} com $N = 4$

Como exemplo, veja o grafo G_{Inv} para $N = 4$ da Figura 3.1. Este grafo possui 24 nós distribuídos em $\frac{N(N-1)}{2} + 1$ níveis. Note que neste caso o grafo é usado apenas para fins didáticos, pois não representa uma instância PQA válida, visto que não existe um $n \in \mathbb{N}$ tal que $C_{n,2} = 4$.

Para um PQA de ordem $n = 4$, sua relaxação linear é de ordem $N = 6$ que por consequência gera um grafo com 720 nós, tornando difícil sua representação gráfica.

É fácil ver que os pares de permutações $(\rho_r, \rho_s) \in W$ são livremente comparáveis, denota-se por $Z(\rho_r) \leq_l Z(\rho_s)$, pois $P_0 = \{1,\ ...,\ i-1,\ i+2,\ ...,\ N\}$, $P_N = \{i\}$ e $P_P = \{i+1\}$ quando $\rho_r = (\rho_r(1),\ \ldots, \rho_r(i),\ \rho_r(i+1),\ \ldots, \rho_r(N))$ e $\rho_s = (\rho_r(1),\ \ldots, \rho_r(i+1),\ \rho_r(i),\ \ldots, \rho_r(N))$ com $\rho_r(i) < \rho_r(i+1)$.

Sendo livremente comparáveis tem-se $Z_{\rho_r} < Z_{\rho_s}$, pois aplicando o TOPL, o par induzido é (t, t'). Observe a avaliação da diferença dos custos $Z_{\rho_r} - Z_{\rho_s}$ desenvolvida a seguir.

$$Z_{\rho_r} - Z_{\rho_s} = \sum_{i=1}^{N} f_i^- (d^+_{\rho_r(i)} - d^+_{\rho_s(i)})$$

Separando as parcelas de P_0, P_N e P_P, tem-se:

$$Z_{\rho_r} - Z_{\rho_s} = (f_i^- - f_{i+1}^-)(d^+_{\rho_r(i)} - d^+_{\rho_r(i+1)}).$$

Como D^+ representa a ordenação não-decrescente do vetor D e F^- representa a ordenação não-crescente do vetor F, pode-se concluir que $(d^+_{\rho_r(i)} - d^+_{\rho_r(i+1)}) \leq 0$ e $(f_i^- - f_{i+1}^-) \geq 0$. Assim, $Z_{\rho_r} \leq Z_{\rho_s}$ e $|Inv(\rho_r)| < |Inv(\rho_s)|$.

A relação de transitividade é válida entre os pares de permutações livremente comparáveis, portanto, se $(\rho_m, \rho_r) \in W$ e $(\rho_r, \rho_n) \in W$ então (ρ_m, ρ_n) são livremente comparáveis (Rangel (2000)).

Sejam as permutações $(\rho_1, \rho_5) \in W$ e $(\rho_5, \rho_{11}) \in W$ pertencentes a G_{Inv}, Grafo das Inversões da Figura 3.1, pode-se afirmar que ρ_1 e ρ_{11} são livremente comparáveis. Portanto, $Z_{\rho_1} \leq Z_{\rho_{11}}$ e $|Inv(\rho_1)| < |Inv(\rho_{11})|$.

Definição 3.1 *O fecho transitivo do grafo direcionado $G = (V, A)$ é o grafo denotado por $\widehat{G} = (V, \widehat{A})$. Se existe um caminho do nó i para o nó j no grafo $G = (V, A)$, então existe um arco (i, j) no grafo $\widehat{G} = (V, \widehat{A})$, isto é, $(i, j) \in \widehat{A}$.*

Aplicando esta definição no grafo G_{Inv}, tem-se o grafo $\widehat{G_{Inv}}$. Desta forma, $\rho(i)$ e $\rho(j)$ são livremente comparáveis com $Z_{\rho_i} \leq Z_{\rho_j}$ e $|Inv(\rho_i)| < |Inv(\rho_j)|$.

O grafo $\widehat{G_{Inv}}$ é um grafo parcial do G_{Liv}, onde G_{Liv} é um grafo que representa todas as relações de livremente comparáveis. Pode-se citar como exemplo de duas permutações ρ_i e ρ_j que são livremente comparáveis mas não estão representadas em $\widehat{G_{Inv}}$ da Figura 3.1 as

permutações $\rho_3 = (1\ 2\ 4\ 3)$ e $\rho_7 = (1\ 3\ 4\ 2)$. Para este caso, procuram-se os pares (t,t') resultante da avaliação de $Z_{\rho_3} - Z_{\rho_7}$. Deve-se separar os conjuntos $P_0 = \{1,3\}$, $P_N = \{2\}$ e $P_P = \{4\}$.

$$Z_{\rho_3} - Z_{\rho_7} = \sum_{i=1}^{N} f_i^-(d^+_{\rho_3(i)} - d^+_{\rho_7(i)})$$

retirando os elementos de P_0 da expressão:

$$Z_{\rho_3} - Z_{\rho_7} = f_2^-(d_2^+ - d_3^+) + f_4^-(d_3^+ - d_2^+)$$
$$Z_{\rho_3} - Z_{\rho_7} = (f_2^- - f_4^-)(d_2^+ - d_3^+)$$

Observe que $(f_2^- - f_4^-) \geq 0$, $(d_2^+ - d_3^+) \leq 0$, logo $(f_2^- - f_4^-)(d_2^+ - d_3^+) \leq 0$.

Sempre que a troca envolve apenas dois elementos, o número de inversões difere de apenas uma unidade, Rangel (2000). Se estão em níveis consecutivos e os conjuntos P_N e P_P são unitários, é facil ver que tais permutações são livremente comparáveis.

Sendo assim, define-se o grafo G'_{Inv} que possui os mesmos nós de G_{Inv} e o novo conjunto de arcos W' que são arcos (ρ_r, ρ_s) entre permutações que diferem de apenas duas posições e $|Inv(\rho_s)| - |Inv(\rho_r)| = 1$

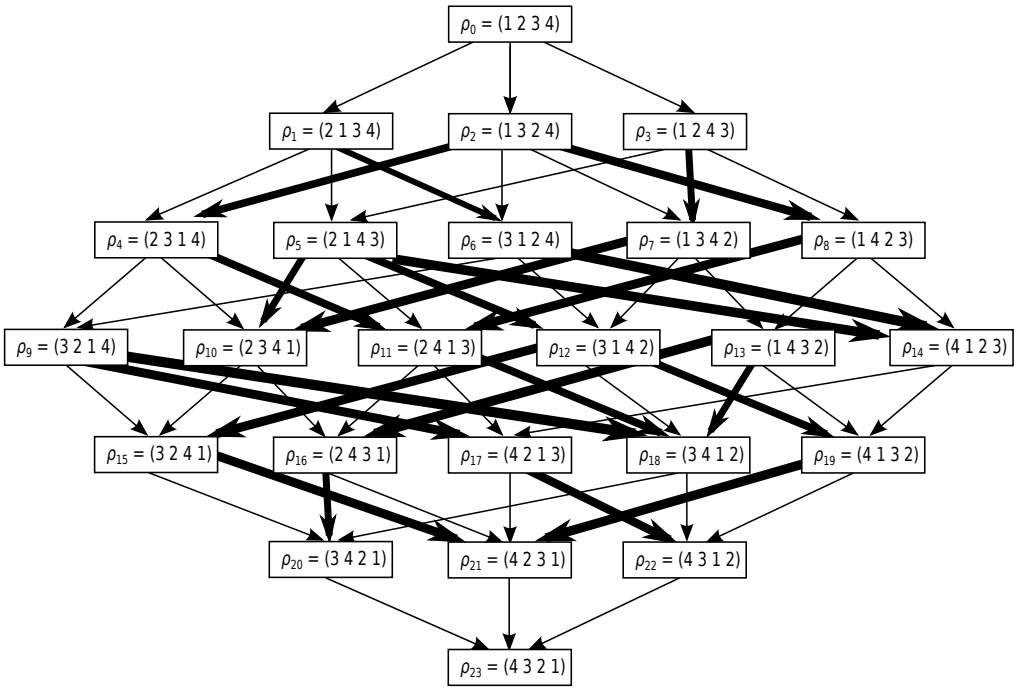

Figura 3.2: Grafo das Inversões $G'_{Inv}(\Pi_4, W')$

Observe que $W \subset W'$. Analogamente define-se o grafo fecho transitivo $\widehat{G'_{Inv}}$ e os resultados sobre permutações livremente comparáveis ρ_i e ρ_j, os custos $Z_{\rho_i} \leq Z_{\rho_j}$ com $|Inv(\rho_i)| < |Inv(\rho_j)|$

são válidos. Após a construção dessa sequência de grafos, Rangel (2000) enuncia o Teorema das Inversões como apresentado a seguir:

Sendo o grafo $\widehat{G'_{Inv}}$ o fecho transitivo de $G'_{Inv} = (\Pi_N, W')$ e $G_{Liv} = (\Pi_N, M)$, onde M é o conjunto das arestas que relacionam todas as permutações livremente comparáveis, tem-se que $\widehat{G'_{Inv}} = G_{Liv}$.

Em outras palavras se duas permutações são livremente comparáveis, seus custos são diretamente proporcionais ao seu número de inversões.

A demonstração deste teorema, assim como todos os detalhes formais desta seção podem ser encontrados em Rangel (2000).

4 Técnicas para a solução do problema

Neste capítulo são apresentadas algumas técnicas encontradas na literatura, necessárias para que o algoritmo desenvolvido neste trabalho apresente resultados satisfatórios. Na primeira seção apresenta-se uma heurística construtiva utilizada para a geração das soluções iniciais, a seguir é apresentado o método de busca local utilizado neste trabalho e por fim faz-se uma breve descrição do *Path Relinking* que é um processo de intensificação na melhora da qualidade das soluções.

4.1 A Heurística Construtiva

Apesar do Algoritmo SolViável, da Seção 2.1.1, ser capaz de reconhecer a viabilidade de uma solução linear para o problema quadrático, sabe-se que ele seria pouco eficiente, do ponto de vista computacional, se tiver de enumerar todas as soluções do PAL(Q^*) para conseguir reconhecer as suas respectivas soluções no PQA(F, D). Para resolver este problema Resendo (2004) propõe o rastreamento de soluções viáveis de boa qualidade através da construção de matrizes específicas de dimensões $n \times (n-1)$, denominadas matrizes $HeadQ$ e $HeadQ^*$. Na matriz $HeadQ$ cada linha i representa permutações ξ que geram $\varphi(1) = i$ para todo $i = \{1, \ldots, n\}$. O Algoritmo 2 a seguir proposto por Resendo (2004), mostra como é feita a construção da matriz $HeadQ$.

Algorithm 2 ConstróiHeadQ2

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

A Figura 4.1 mostra como seria o formato da matriz $HeadQ$ considerando $n = 4$. A partir das linhas 7-10 do Algoritmo 1 (SolViável), pode-se observar que a linha 1 da ma-

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

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

triz HeadQ gera a permutação $\varphi^1 = \begin{pmatrix} 1 & 2 & 3 & ... & n \\ 1 & 2 & 3 & ... & n \end{pmatrix}$, a linha 2 gera a permutação $\varphi^2 = \begin{pmatrix} 1 & 2 & 3 & ... & n \\ 2 & 1 & 3 & ... & n \end{pmatrix}$, linha 3 gera a permutação $\varphi^3 = \begin{pmatrix} 1 & 2 & 3 & ... & n \\ 3 & 1 & 2 & ... & n \end{pmatrix}$, e assim por diante até a linha n, que gera $\varphi^n = \begin{pmatrix} 1 & 2 & 3 & ... & n \\ n & 1 & 2 & ... & n-1 \end{pmatrix}$. Essas permutações φ^i são chamadas neste trabalho de $\varphi's$ ordenadas. Se fizer uma troca de algumas componentes da linha da matriz, essa nova linha gera uma permutação φ com a mesma troca, lembrando que a primeira posição das $\varphi's$ são fixadas. Para $n = 4$, por exemplo, considere a linha 1 da $HeadQ$ igual a 1 2 3, faça uma troca qualquer, por exemplo, 3 2 1. A φ gerada por essa linha a partir do Algoritmo 1 (SolViável) é $\varphi^1 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 3 & 2 \end{pmatrix}$. As observações destacadas acima são importantes para o entendimento do algoritmo proposto neste trabalho. Sabe-se que toda permutação $\xi \in \Pi_N$ no PAL(Q) possui uma correspondente $\rho \in \Pi_N$ no PAL(Q^*), sendo assim, Resendo (2004) propõe a construção de uma matriz $HeadQ^*$ que esquematiza as permutações $\rho \in \Pi_N$ capazes de gerar uma $\varphi \in \Pi_n$, solução do PQA(F, D). É necessário que se trabalhe com as soluções do PAL(Q^*), pois o Teorema das Inversões é aplicado quando os vetores que definem o problema são ordenados tais como F^- e D^+. Para a construção da $HeadQ^*$, necessita-se novamente das permutações auxiliares ϕ_F e ϕ_D, pois estas permutações contêm uma memória das trocas efetuadas para a geração dos vetores F^- e D^+ para a obtenção da relação entre ξ e ρ através da equação 2.5. Agora se faz um processo inverso para efetuar a transformação entre essas matrizes.

Como os algoritmos trabalham somente com cabeças das permutações $\xi \in \Pi_N$, é importante assegurar que as imagens das aplicações ϕ_D^{-1} serão $1, \ldots, n$–1. Aplica-se ϕ_F, nos valores de $\xi(i)$ com $i = 1, \ldots, n$–1, indicando em que posição os elementos das linhas da matriz $HeadQ^*$ estão na permutação $\rho \in \Pi_N$. Exemplificando esta situação, analise a instância GP66 dadas as suas

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

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

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

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

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

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

Para conhecer a imagem de cada elemento da matriz basta aplicar a função ϕ_D pois, como visto, a construção da matriz $HeadQ$ está baseada na permutação identidade de $\xi \in \Pi_N$ e a imagem da composição das permutações é exatamente ϕ_D. A Figura 4.2 mostra como fica a matriz $HeadQ^*$ depois de gerada. É importante lembrar que a $HeadQ$ possui o mesmo formato, independente dos valores dos dados de entrada do PQA. Contudo, a $HeadQ^*$ é totalmente dependente desses dados, pois é construída através das permutações que guardam as trocas feitas para as ordenações de F^- e D^+.

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

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

Figura 4.2: Matriz $HeadQ^*$

4.2 A Busca Local

Um algoritmo de busca local é um algoritmo que varre uma determinada vizinhança na busca por soluções melhores que a solução atual. Seu término se dá quando esgotadas todas as possibilidades, não há uma solução melhor que a atual. A Algoritmo 3 mostra a ideia de uma busca local genérica.

Para este procedimento deve-se inicialmente definir uma estrutura de vizinhança, no caso deste trabalho utiliza-se a $k-$troca, que pode ser encontrada em Li, Pardalos e Resende (1994)

Algorithm 3 Busca Local

1: **while** solução não é localmente ótima **do**
2: Encontrar uma melhor solução $t \in Viz(s)$;
3: $s = t$;
4: **end while**
5: Retornar (s como localmente ótima);

e Resendo (2004), com *k=2*. Como estratégia de busca, utilizou-se a Busca em Largura juntamente com a Busca em Profundidade.

Dadas duas permutações, φ_1 e φ_2, define-se a diferença entre elas como $\delta(\varphi_1, \varphi_2) = \{i | \varphi_1(i) \neq \varphi_2(i)\}$ e a distância por $d(\varphi_1, \varphi_2) = |\delta(\varphi_1, \varphi_2)|$.

Toma-se por vizinhança de uma permutação φ todas as permutações φ' onde sua distância é menor ou igual a k, tal que $2 \leq k \leq n$. Neste trabalho considerou-se $k = 2$, vizinhança conhecida como $2-$troca.

Como exemplo toma-se $\varphi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 \end{pmatrix}$ com $n = 4$, o número de elementos da vizinhança é dado por $|Viz(\varphi)| = C_{4,2} = 6$, então tem-se

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

Neste trabalho utiliza-se uma Busca em Largura, onde são feitas sucessivas trocas na solução corrente, $2-$troca, a fim de encontrar uma melhor solução. Quando uma melhor solução é encontrada na vizinhança, uma nova busca é realizada, agora para a vizinhança dessa nova melhor solução. Este processo termina quando nenhuma melhor solução é encontrada. Ao final, percebe-se que foi efetuada a construção de uma árvore com diferentes níveis, onde de cada nível é extraída a melhor solução, veja a Figura 4.3.

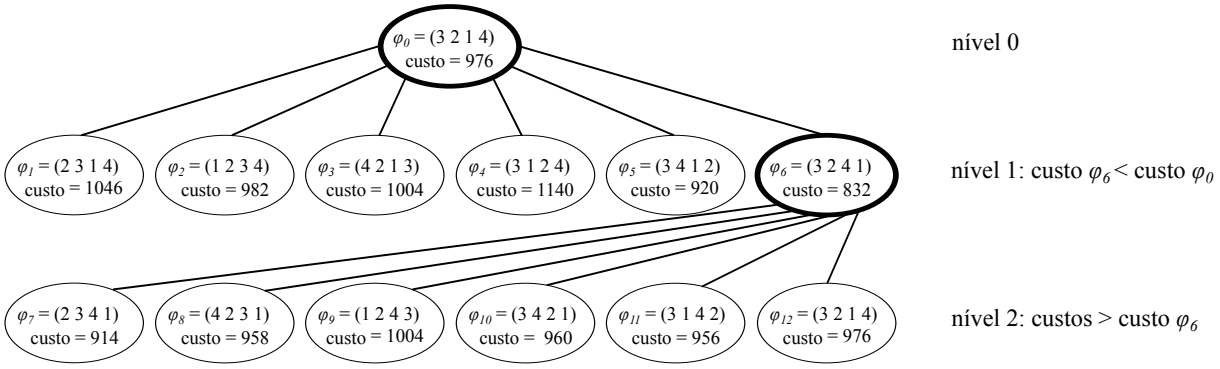

Figura 4.3: Árvore construída no processo de busca local para $n = 4$

4.3 O *Path Relinking* (PR)

A técnica *Path Relinking* (PR), também conhecida como reconexão por caminhos, descrita originalmente por Glover (1996), consiste em explorar o caminho ou trajetória entre duas soluções dadas chamadas de inicial e guia, vide Seção 5.3.3, esta última é a solução a qual deseja-se chegar. Queiroz e Mendes (2011) explicam que basicamente a sua aplicação a cada iteração realiza um movimento na solução inicial com o objetivo de aproximá-la da solução guia. Para que isso ocorra deve-se introduzir atributos encontrados na solução guia na solução inicial, até que se tornem iguais. O Algoritmo 4 apresenta a ideia geral desta técnica.

Algorithm 4 *Path Relinking*

```
1: Entrada de dados(soluçãoInicial, soluçãoGuia);
2: soluçãoAtual != soluçãoInicial;
3: while soluçãoAtual = soluçãoInicial do
4:     soluçãoAtual = soluçãoAtual + movimento(soluçãoGuia);
5: end while
6: Retornar (melhor soluçãoAtual encontrada);
```

Como exemplo de aplicação desta técnica em soluções do PQA observe a Figura 4.4, em negrito estão sendo destacadas as ações da função **movimento(soluçãoGuia)**, linha 4 do Algoritmo 4, que neste trabalho, são as trocas dos elementos, um em cada nível, com o objetivo de aproximar a cada passo, a solução inicial da solução guia. O caminho seguido pelo PR é compreendido como um processo de intensificação que permite explorar espaços de solução no caminho de aproximação entre as duas soluções. É possível que ao longo desta trajetória ótimos locais sejam alcançados, produzindo novas soluções viáveis de qualidade. Ainda para esse exemplo a solução marcada com o asterisco é a melhor solução encontrada no trajeto, que para a instância GP66 , vide Seção 2.1, é a solução ótima do problema.

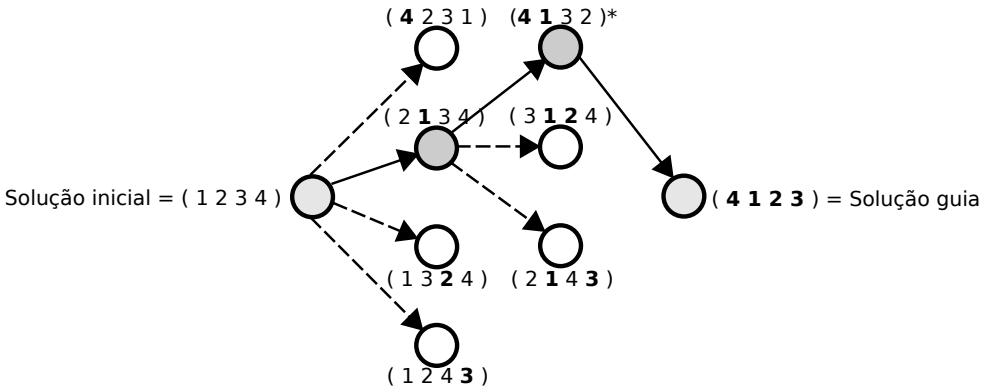

Figura 4.4: Um exemplo de *Path Relinking* para o GP66

Quanto as formas de implementação, Resende et al. (2010) discutem a questão de atua-

lização estática e atualização dinâmica. Na atualização estática um conjunto Elite é criado e atualizado ao longo das iterações do algoritmo. Ao final o PR é aplicado como pós-otimização entre as soluções do conjunto Elite, com o intuito de explorar o espaço existente entre cada par destas soluções. Na atualização dinâmica para cada solução gerada durante uma iteração, uma solução do conjunto Elite é sorteada e o PR é aplicado.

Há ainda algumas outras variações de PR na literatura citadas por Resende e Ribeiro (2003), por exemplo, *forward relinking*, Figura 4.4, partindo da solução inicial até alcançar a solução guia, *backward relinking*, que faz o inverso da técnica anterior e *mixed relinking*, onde as duas trajetórias são simultaneamente exploradas, a primeira iniciando com a solução inicial e a segunda iniciando com a solução guia até que se encontrem.

O Capítulo 5 mostra com mais detalhes como é a feita a implementação desta técnica e quais as variações utilizadas para que resultados satisfatórios sejam alcançados.

5 As Fases de Construção do Algoritmo ATIPR

Nesta seção encontra-se a principal contribuição desse trabalho, o algoritmo ATIPR. Para o melhor entendimento deste algoritmo, são apresentadas as fases de sua construção. Todas as fases são discutidas em seus prós e contras até se obter uma solução satisfatória em qualidade e tempo de execução.

5.1 Visão Geral do ATIPR

O algoritmo desenvolvido neste trabalho, Algoritmo baseado no Teorema das Inversões com *Path Relinking* (ATIPR), é um algoritmo que combina o método construtivo de soluções iniciais proposto por Resendo (2004), enriquecido com uma técnica de refinamento de soluções com forte embasamento no Teorema das Inversões, Rangel (2000), e uma diversificação caracterizada por um *Path Relinking*. O Algoritmo 5 apresenta o pseudo-código do Algoritmo ATIPR.

Na linha 1 do algoritmo faz-se a leitura dos dados de entrada. Na linha 2 são construídos os vetores ordenados F^- e D^+, necessários para formar os vetores ϕ_F e ϕ_D, que guardam as posições das trocas efetuadas na construção de F^- e D^+, conforme descrito na Seção 2.1.1. Além disso, são construídas as matrizes $HeadQ$ e $HeadQ^*$ já apresentadas na Seção 4.1. A matriz $HeadQ$ é necessária para gerar as φ^k ordenadas (ainda na linha 2). A linha 3 faz a inicialização das variáveis $custo\varphi_{best}$ e φ_{best} que armazenam o custo da melhor solução encontrada para o problema e a sua permutação φ correspondente. O laço da linha 4 controla a linha k da $HeadQ^*$ e, por consequência, a φ^k ordenada que está sendo avaliada. Para cada valor de k, uma busca local é realizada na φ^k ordenada (linha 5) e as atualizações de $custo\varphi_{best}$ e φ_{best} são feitas nas linhas 6 e 7. Os laços das linhas 8 e 9 do algoritmo controlam os elementos a serem testados dois a dois na linha k da matriz $HeadQ^*$ e suas correspondentes posições no vetor ϕ_F. Tais testes são efetuados na linha 10 pela função **DiminuirNumInversões**, Algoritmo 6.

Algorithm 5 ATIPR

1: Entrada de dados (n, F, D);
2: gerar F^-, D^+, ϕ_F, ϕ_D, $HeadQ$, $HeadQ^*$ e φ^k's ordenadas: $\varphi^1, \varphi^2 \ldots \varphi^n$;
3: $custo_{\varphi_{best}} \leftarrow \infty$; $\varphi_{best} \leftarrow \emptyset$;
4: **for** $k = 1, \ldots, n$ **do**
5: $\quad \varphi_{bestLocal} \leftarrow$ Busca Local(φ^k);
6: $\quad custo\varphi_{best} \leftarrow$ atualizar custo $(custo\varphi_{bestLocal}, custo\varphi_{best})$;
7: $\quad \varphi_{best} \leftarrow$ atualizar $(\varphi_{bestLocal}, custo\varphi_{best}, custo\varphi_{bestLocal})$;
8: $\quad$ **for** $i = 1, \ldots, (n-1)-1$ **do**
9: $\quad\quad$ **for** $j = i+1, \ldots, n-1$ **do**
10: $\quad\quad\quad$ **if** DiminuirNumInversões$(HeadQ^*, \phi_F, i, j, k)$ **then**
11: $\quad\quad\quad\quad \varphi \leftarrow$ troca$(\varphi^k[i+1], \varphi^k[j+1])$;
12: $\quad\quad\quad\quad \varphi_{bestLocal} \leftarrow$ Busca Local(φ);
13: $\quad\quad\quad\quad custo\varphi_{best} \leftarrow$ atualizar custo $(custo\varphi_{bestLocal}, custo\varphi_{best})$;
14: $\quad\quad\quad\quad \varphi_{best} \leftarrow$ atualizar $(\varphi_{bestLocal}, custo\varphi_{best}, custo\varphi_{bestLocal})$;
15: $\quad\quad\quad$ **end if**
16: $\quad\quad$ **end for**
17: $\quad$ **end for**
18: $\quad \varphi_{bestPR} \leftarrow$ PR-Ale $(n, \varphi_{bestLocal})$;
19: $\quad custo\varphi_{best} \leftarrow$ atualizar custo $(custo\varphi_{bestPR}, custo\varphi_{best})$;
20: $\quad \varphi_{best} \leftarrow$ atualizar $(\varphi_{bestPR}, custo\varphi_{best}, custo\varphi_{bestPR})$;
21: **end for**
22: Retornar φ_{best} e $custo\varphi_{best}$;

Se o retorno for verdadeiro, faz-se a troca de elementos na solução φ^k que está sendo avaliada no momento (linha 11). As linhas de 12 a 14 são responsáveis pela atualização das variáveis $custo\varphi_{best}$ e φ_{best} a cada troca de posições efetuadas. Antes do término de cada iteração k um *path relinking* é executado (linha 18), Algoritmo 11 − PR-Ale, Seção 5.3.4. As linhas 19 e 20 são responsáveis pela atualização das variáveis $custo\varphi_{best}$ e φ_{best} ao final de cada iteração do laço da linha 4. Ao se esgotarem as n iterações do laço da linha 4, o algoritmo retorna a melhor solução encontrada para o problema, linha 22.

5.2 Os Testes de Qualidade de Solução

Nesta seção apresenta-se a função **DiminuirNumInversões** que avalia se uma troca feita na linha $k = \{1, \ldots, n\}$ da $HeadQ^*$ induz uma diminuição do número de inversões na permutação ρ, levando-se em consideração as posições ocupadas no vetor ϕ_F.

5.2.1 A função DiminuirNumInversões1

Observe o Algoritmo 6 a seguir.

Algorithm 6 DiminuirNumInversões1

1: Entrada de dados $(HeadQ^*, \phi_F, i, j, k)$;
2: **if** $((HeadQ^*[k][i] > HeadQ^*[k][j])$ **and** $(\phi_F[i] < \phi_F[j]))$ **or** // posições i e j na linha k
$((HeadQ^*[k][i] < HeadQ^*[k][j])$ **and** $(\phi_F[i] > \phi_F[j]))$ **then**
3: Retornar Verdadeiro
4: **else**
5: Retornar Falso
6: **end if**

O teste descrito na linha 2 do Algoritmo 6, é a principal contribuição deste trabalho, ele está dividido em duas partes: o **Teste de Qualidade 1** (TQ1) avalia se um determinado elemento i da linha k da $HeadQ^*$ é maior que um determinado elemento j da mesma linha k da $HeadQ^*$ e se este elemento i se encontra em uma posição $\phi_F[i]$ menor que a posição do elemento j, $\phi_F[j]$, lembrando que ϕ_F indica a posição do elemento na permutação ρ associada, veja a Seção 2.1.1 para esclarecimentos sobre o ϕ_F; o **Teste de Qualidade 2** (TQ2) avalia se um determinado elemento i da linha k da $HeadQ^*$ é menor que um determinado elemento j da mesma linha k da $HeadQ^*$ e se este elemento i se encontra em uma posição $\phi_F[i]$ maior que a posição do elemento j, $\phi_F[j]$. Caso alguma dessas condições seja satisfeita efetua-se a troca do elemento $HeadQ^*[k][i]$ com o elemento $HeadQ^*[k][j]$ pois assim há uma redução no número de inversões da ρ associada a $HeadQ^*$. De acordo com o Teorema das Inversões , Seção 3.2, espera-se que a nova solução φ do PQA, associada a esta permutação ρ do PAL(Q^*), seja de boa qualidade. Vejamos a Figura 5.1 a seguir.

		ϕ_F		
		1	2	5
k	1	4	6	2
	2	4	3	5
	3	6	3	1
	4	2	5	1

Figura 5.1: Exemplo de uma $HeadQ^*$ para avaliação de testes de qualidade de solução.

Analisando a matriz da Figura 5.1 através dos testes de qualidade de solução para $k = 1$, $i = 2$ e $j = 3$, temos:

- $HeadQ^*[1][2] > HeadQ^*[1][3] \Rightarrow 6 > 2 \Rightarrow$ condição satisfeita;
- $\phi_F[2] < \phi_F[3] \Rightarrow 2 < 5 \Rightarrow$ condição satisfeita.

Logo, a troca deve ser efetuada pois neste momento um elemento de maior valor se encontra em uma posição de menor valor na $HeadQ^*$.

Analisando agora a Figura 5.1 através dos testes de qualidade de solução para $k = 2$, $i = 2$ e $j = 3$, temos:

- $HeadQ^*[2][2] > HeadQ^*[2][3] \Rightarrow 3 > 5 \Rightarrow$ condição não satisfeita;
- $\phi_F[2] < \phi_F[3] \Rightarrow 2 < 5 \Rightarrow$ condição satisfeita;
- $HeadQ^*[2][2] < HeadQ^*[2][3] \Rightarrow 3 < 5 \Rightarrow$ condição satisfeita;
- $\phi_F[2] > \phi_F[3] \Rightarrow 2 > 5 \Rightarrow$ condição não satisfeita.

Logo, a troca não deve ser efetuada pois neste momento um elemento de maior valor se encontra em uma posição de maior valor na $HeadQ^*$, conforme o Teorema das Inversões , Seção 3.2.

5.2.2 As Alterações nos Testes de Qualidade de Soluções

Os testes TQ1 e TQ2 do Algoritmo 6, Seção 5.2.1, auxiliam muito o Algoritmo ATIPR no que diz respeito à qualidade de solução, porém, o tempo de resposta do algoritmo é um fator muito importante.

Neste trabalho apesar de bons resultados de tempo de resposta do algoritmo, novos estudos e testes foram feitos para alcançar uma melhor relação de **tempo** $\times$ **qualidade de solução**. Foram realizados vários conjuntos de testes para algumas variações do algoritmo e percebeu-se que com uma simples modificação o algoritmo se torna mais rápido com perdas aceitáveis na qualidade das soluções encontradas. Os testes computacionais mostraram que grande parte das soluções iniciais que satisfazem aos **Testes de Qualidade de Soluções**, TQ1 e TQ2, Seção 5.2, são estruturalmente semelhantes. A busca local aplicada a essas soluções não seguem caminhos muito diferentes no espaço de soluções do problema, tornando-se desnecessário a realização dos dois testes para o algoritmo. Portanto, foram propostas duas modificações no Algoritmo 6.

1. Retirou-se o TQ2 pois durante os testes efetuados, percebeu-se que na maioria dos casos a função **DiminuirNumInversões1** usando apenas TQ1, foi tão eficiente quanto a versão original (TQ1 e TQ2);
2. Uma componente aleatória foi inserida ao teste para reduzir o número de buscas locais, visto que, as soluções são parecidas (diferença em apenas duas posições) resultando em pouca diversificação, logo, estatisticamente, a busca local irá acontecer em apenas 50% dos casos em que TQ1 for verdadeiro.

O Algoritmo 7, denominado de **DiminuirNumInversões**, proposto nesta nova etapa é apresentado a seguir.

Algorithm 7 DiminuirNumInversões

1: Entrada de dados $(HeadQ^*, \phi_F, i, j, k)$;
2: **if** $(HeadQ^*[k][i] > HeadQ^*[k][j])$**and**$(\phi_F[i] < \phi_F[j])$**and**$(sortear(4) < 3)$ **then**
3: Retornar Verdadeiro
4: **else**
5: Retornar Falso
6: **end if**

Os testes computacionais mostraram que apenas com a retirada do TQ2 os tempos de execução do algoritmo foram reduzido a mais de 50%, veja a Tabela 5.1.

Tabela 5.1: Tempo de processamento do ATIPR com a retirada do TQ2 para instâncias nug.

INSTÂNCIA	TQ1+TQ2 (s)	TQ1 (s)
nug12	0,29	0,10
nug15	2,15	0,66
nug20	19,77	5,57
nug30	586,42	144,06

Com a inserção da componente aleatória os tempos caíram ainda mais, melhorando o desempenho do ATIPR, esses novos resultado podem ser vizualizados na Tabela 5.2

Tabela 5.2: Tempo de processamento do ATIPR com a retirada do TQ2 e inserção de uma componente aleatória para instâncias nug.

INSTÂNCIA	TQ1+TQ2 (s)	TQ1+Comp. Aleatória (s)
nug12	0,29	0,08
nug15	2,15	0,43
nug20	19,77	3,70
nug30	586,42	87,51

5.3 Aplicação do *Path Relinking* (PR)

Como já citado na Seção 4.3, a aplicação do *path relinking* se dá a cada iteração realizando um conjunto de movimentos na solução inicial, com o objetivo de aproximá-la da solução guia.

5.3.1 O *Path Relinking* Padrão (PRP)

O Algoritmo 8, denominando PRP a seguir, descreve como foi aplicada esta técnica neste trabalho.

Algorithm 8 PRP

1: Entrada de dados $(n, \varphi_{guia}, \varphi_{inicial})$; // n é a dimensão do PQA
2: $\varphi_{bestPR} \leftarrow \varphi_{inicial}$;
3: **for** $(i = 1, \ldots, n-1)$ **do**
4: $posVetInicial = \text{buscaPos}(\varphi_{inicial}, n, \varphi_{guia}[i])$; //busca a posição que $\varphi_{guia}[i]$ se encontra no $\varphi_{inicial}$
5: $\text{trocar}(\varphi_{inicial}, i, posVetInicial)$; // efetua a troca na $\varphi_{inicial}$
6: **if** $(custo(\varphi_{inicial}) < custo(\varphi_{bestPR}))$ **then**
7: $\varphi_{bestPR} \leftarrow \varphi_{inicial}$;
8: **end if**
9: **end for**
10: Retornar φ_{bestPR};

Na linha 1 temos os dados de entrada necessários para o funcionamento do algoritmo. Na linha 2 é feita a inicialização da solução φ_{bestPR}, pois nas linhas 6 e 7 esta solução será atualizada caso haja um vetor de menor custo encontrado durante o percurso do *path relinking*, caso contrário, o valor retornado permanece o próprio valor com o qual foi inicializada φ_{bestPR}. O laço da linha 3 é responsável pelos $n-1$ movimentos que o *path relinking* irá realizar enquanto a $\varphi_{inicial}$ se aproxima da φ_{guia}. Na linha 4 é feita uma busca da posição em que o elemento $\varphi_{guia}[i]$ se encontra na solução $\varphi_{inicial}$. Com esta infomação é realizada a troca dos elementos das posições i e $posVetInicial$ da solução $\varphi_{inicial}$, linha 5, a fim de aproximá-la do vetor φ_{guia}. Esgotadas todas as trocas necessárias para $\varphi_{inicial}$ se igualar a φ_{guia}, na linha 10 o algoritmo retorna φ_{best} a melhor solução do PRP.

5.3.2 O *Path Relinking (PR)*

O Algoritmo 9, *Path Relinking*, notado por PR, é exatamente o Algoritmo 8, PRP, acrescido de uma fase de intensificação para a busca de melhores soluções.

A fase de intensificação, Busca Local, encontra-se na linha 11, porém para executá-la, linha 10, é gerado um valor aleatório com uma probabilidade de 50% de satisfazer o teste condicional.

A análise dos resultados, em termos de qualidade de solução, mostrou melhoras consideráveis para 100% das instâncias testadas que não haviam encontrado a solução ótima, por outro lado, houve um pequeno prejuízo no tempo de resposta do algoritmo como pode ser visto na Tabela 5.3.

5.3.3 O *Path Relinking* com Atualização Dinâmica (PR-Din)

O *Path Relinking* com Atualização Dinâmica (PR-Din) proposto neste trabalho, baseia-se na aplicação da técnica *forward relinking*, ver Seção 4.3, com um processo de atualização

Algorithm 9 PR

1: Entrada de dados $(n, \varphi_{guia}, \varphi_{inicial})$;
2: $\varphi_{bestPR} \leftarrow \varphi_{inicial}$;
3: **for** $(i = 1, \ldots, n-1)$ **do**
4: $posVetInicial =$ buscaPos$(\varphi_{inicial}, n, \varphi_{guia}[i])$; //busca a posição de $\varphi_{guia}[i]$ em $\varphi_{inicial}$
5: trocar$(\varphi_{inicial}, i, posVetInicial)$;
6: **if** (custo$(\varphi_{inicial})$ <custo(φ_{bestPR})) **then**
7: $\varphi_{bestPR} \leftarrow \varphi_{inicial}$;
8: **end if**
9: //sorteia-se aleatoriamente um inteiro $\in \{1, 2, 3, 4\}$ e testa se ele é menor do que 3
10: **if** (sortear$(4) < 3$) **then**
11: $\varphi_{bestBusca} \leftarrow$ Busca Local$(\varphi_{inicial})$;
12: $custo\varphi_{best} \leftarrow$ atualizar custo $(custo\varphi_{bestBusca}, custo\varphi_{best})$;
13: $\varphi_{best} \leftarrow$ atualizar $(\varphi_{bestBusca}, custo\varphi_{best}, custo\varphi_{bestBusca})$;
14: **end if**
15: **end for**
16: Retornar φ_{bestPR};

Tabela 5.3: Tempo de processamento do PR com e sem a fase de intensificação.

INSTÂNCIA	PR com intensificação (s)	PR sem intensificação (s)
nug12	0,08	0,02
nug15	0,43	0,21
nug20	3,70	1,93
nug30	87,51	55,11

dinâmica, discutido por Resende et al. (2010). No modelo de atualização dinâmica, a cada iteração do PR-Din, uma nova solução guia será selecionada de um conjunto de soluções elites, aqui denominado *Pool*, que contém as melhores soluções encontradas durante a execução do algoritmo sendo atualizado a cada iteração. O Algoritmo 10, mostra o funcionamento do PR-Din.

Vale aqui ressaltar que o Algoritmo ATIPR é iterativo, ou seja, a cada iteração realiza uma busca local na solução inicial construída, vide o Algoritmo 5, Seção 5.1.

Na linha 1 temos: n o tamanho da instância, k que indica qual a linha das φ's ordenadas que está sendo analisada, *Pool* é o conjunto de soluções elites, $\varphi_{bestLocal}$ a melhor solução encontrada durante a busca local, φ^k é a φ ordenada que está sendo avaliada no momento, estes dados são fornecidos pelo Algoritmo 5, ATIPR. A linha 2 inicializa a melhor solução encontrada no PR-Din, que ao término do processo, será retornada pela função. As linhas 3 e 4 são responsáveis pelo início da construção do *Pool*, isto é, $k = 1$. O $\varphi_{bestLocal}$, que é um dado de entrada do PR-Din, é a melhor solução encontrada durante o processo de busca local, linhas 8-17, do Algoritmo 5, ATIPR. A linha 6 é responsável pela atribuição da solução inicial

Algorithm 10 PR-Din

1: Entrada de dados$(n, k, Pool, \varphi_{bestLocal}, \varphi^k)$
2: $\varphi_{bestPR-Din} \leftarrow \emptyset$;
3: **if** $(k = 1)$ **then**
4: $Pool \leftarrow Pool + \varphi_{bestLocal}$;
5: **else**
6: $\varphi_{inicial} \leftarrow \varphi_{bestLocal}$; // melhor solução encontrada na busca local do Algoritmo ATIPR
7: // # é a cardinalidade e $n/2$ é tamanho máximo de *Pool*
8: **if** $(\#Pool < n/2)$ **then**
9: $\varphi_{guia} \leftarrow$ sortear $\varphi \in Pool$;
10: $\varphi_{bestPR} \leftarrow$ PR $(\varphi_{inicial}, \varphi_{guia})$;
11: **if** (custo (φ_{bestPR}) < custo $(\varphi_{bestLocal})$) **then**
12: $Pool \leftarrow Pool + \varphi_{bestPR}$;
13: **else if** (custo$(\varphi_{bestLocal})$ < custo (φ^k)) **then**
14: $Pool \leftarrow Pool + \varphi_{bestLocal}$;
15: **else**
16: $Pool \leftarrow Pool + \varphi^k$;
17: **end if**
18: **else if** $(\#Pool = n/2)$ **then**
19: $\varphi_{piorPool} \leftarrow \varphi$ de maior custo $\in Pool$;
20: $Pool \leftarrow Pool - \varphi_{piorPool}$;
21: $\varphi_{guia} \leftarrow$ sortear $\varphi \in Pool$;
22: $\varphi_{bestPR} \leftarrow$ PR $(\varphi_{inicial}, \varphi_{guia})$;
23: **if** ((custo (φ_{bestPR}) < custo $(\varphi_{bestLocal})$)**and**$(\varphi_{bestPR} \notin Pool)$) **then**
24: $Pool \leftarrow Pool + \varphi_{bestPR}$;
25: **else if** ((custo$(\varphi_{bestLocal})$ < custo (φ^k))**and**$(\varphi_{bestLocal} \notin Pool)$) **then**
26: $Pool \leftarrow Pool + \varphi_{bestLocal}$;
27: **else**
28: $Pool \leftarrow Pool + \varphi^k$;
29: **end if**
30: **end if**
31: **end if**
32: Atualizar $\varphi_{bestPR-Din}$; // melhor solução encontrada para o problema
33: Retornar $\varphi_{bestPR-Din}$;

preparando os parâmetros para a execução do PR na linha 10. Caso o *Pool* não esteja vazio e a sua cardinalidade for menor do que $n/2$, linha 8, sorteia-se uma solução que será usada como φ_{guia}, linha 9. Para o PR da linha 10, pode-se observar que a solução inicial, $\varphi_{inicial}$, sempre será a $\varphi_{bestLocal}$ que já é um dado de entrada do Algoritmo PR-Din, enquanto a solução guia, φ_{guia}, sempre será uma solução selecionada aleatoriamente do conjunto de soluções elites. Se a melhor solução encontrada no *path relinking* for melhor que $\varphi_{bestLocal}$, linha 11, o *Pool* então receberá φ_{bestPR}, linha 12. Caso a melhor solução encontrada no *path relinking* não seja melhor do que a $\varphi_{bestLocal}$, testa-se na linha 13 se esta última é melhor solução que a φ^k ordenada da linha k atual. Em caso verdadeiro, o *Pool* recebe $\varphi_{bestLocal}$, linha 14. Em caso contrário, o *Pool* receberá φ^k como a melhor solução encontrada, linha 16.

A partir da linha 18, o algoritmo analisa se o *Pool*, conjunto das soluções elites, encontra-se completo. Neste caso, a $\varphi_{piorPool}$ será selecionada e descartada do *Pool*, linha 19 e 20 respectivamente. Na linha 21, sorteia-se uma solução do *Pool* que será usada como φ_{guia}. Assim como o PR da linha 10, no PR da linha 22 pode-se observar que a solução inicial, $\varphi_{inicial}$, sempre será a $\varphi_{bestLocal}$ que já é um dado de entrada do Algoritmo PR-Din, enquanto a solução guia, φ_{guia}, sempre será uma solução selecionada aleatoriamente do conjunto de soluções elites. Se a melhor solução encontrada do *path relinking* for melhor que $\varphi_{bestLocal}$ e se essa mesma solução não se encontra no *Pool*, linha 23, a $\varphi_{piorPool}$ dá lugar à $\varphi_{bestLocal}$ no *Pool*, linha 24. Caso a melhor solução encontrada no *path relinking* não seja melhor que a $\varphi_{bestLocal}$, testa-se na linha 25 se esta última é melhor solução do que a φ^k ordenada da linha k atual e se essa mesma solução não se encontra no *Pool*. Em caso verdadeiro, a $\varphi_{piorPool}$ dá lugar à $\varphi_{bestLocal}$ no *Pool*, linha 26. Porém se $\varphi_{bestLocal}$ não for melhor solução que a φ^k ordenada, linha 27, então o *Pool* irá descartar a $\varphi_{piorPool}$ para receber a φ^k, linha 28. Para cada iteração k, é feita uma atualização da melhor solução encontrada no Algoritmo 10, linha 32, sendo este valor retornado ao fim do algoritmo, linha 33.

Diferentemente de Resende e Ribeiro (2003), onde os resultados obtidos do *path relinking* usando atualização dinâmica são totalmente satisfatórios, neste trabalho a aplicação do PR com soluções elites mostrou-se pouco eficiente. As soluções elites inseridas no *Pool* não possuem uma grande diversificação, devido ao fato de Algoritmo 6 trabalhar com soluções que possuem uma estrutura padrão advinda da $HeadQ^*$. Como essas soluções que se encontram do *Pool* são muito parecidas, o algoritmo fica preso em ótimos locais, o que inviabiliza a sua utilização. Veja a Figura 5.2 onde ilustra um conjunto de soluções elites em um determinado momento da execução do programa para a instância nug12. Observe que há pouca diversificação entre as soluções. Este fator contribui para que o *path relinking* não consiga alcançar melhores resultados que os já existentes.

2	1	8	3	10	7	11	9	5	6	4	12
10	5	6	4	2	1	7	8	3	9	11	12
1	2	10	3	8	7	11	9	4	5	6	12
4	1	2	3	5	6	7	8	9	10	11	12
5	1	2	3	4	6	7	8	9	10	11	12
5	10	2	3	6	7	11	9	8	5	1	12

Figura 5.2: Um exemplo de conjunto de soluções elites para a instância nug12.

5.3.4 O *Path Relinking* com Soluções Guias Aleatórias (PR-Ale)

A ideia deste algoritmo é bastante simples e ao mesmo tempo mais eficiente do ponto de vista computacional, uma vez que, comparado com o PR-Din, o PR-Ale não constrói o conjunto de soluções elites (*Pool*).

A proposta neste caso é gerar uma solução guia de forma totalmente aleatória, denominada $\varphi_{aleatoria}$, bem diferente da $\varphi_{inicial}$. Caso isso ocorra, dado que se trata de um processo aleatório, obtém-se assim uma maior diversificação de soluções no processo do *path relinking*. O Algoritmo 11 a seguir mostra essa situação.

Algorithm 11 PR-Ale

1: Entrada de dados(n, $\varphi_{bestLocal}$);
2: $\varphi_{guia} \leftarrow$ gerar $\varphi_{aleatoria}(n)$;
3: $\varphi_{bestPR} \leftarrow$ PR ($\varphi_{bestLocal}$, φ_{guia});
4: Retornar φ_{bestPR};

6 Resultados e Discussões

Neste capítulo são apresentados os resultados computacionais do Algoritmo ATIPR, em suas versões Serial e Paralela, comparando-os com os algoritmos *Heuristic Head* (HH) de Resendo (2004) e *Greedy Randomized Adaptive Search Procedures* (GRASP) de Resende et al. (2010), encontrados na literatura. Os algoritmos HH e GRASP possuem a estratégia de construção de soluções iniciais de boa qualidade a cada iteração, seguidos de uma fase de melhoramento através de uma busca local. Esta é a razão da escolha desses algoritmos para comparação com o comportamento do ATIPR, tanto na qualidade de solução quanto em tempo computacional. Para os testes do Algoritmo Serial foram usadas instâncias de tamanho até 30 e para o Algoritmo Paralelo, instâncias de ordem até 50. Todas as instâncias usadas encontram-se disponíveis na biblioteca *online* QAPLIB, Burkard et al. (2013).

6.1 O Algoritmo Serial

A ATIPR foi implementado na linguagem de programação C e os testes computacionais foram executados em um notebook com processador Intel *Core* 2 Duo T6500 2.1GHz $\times$ 2, memória RAM DDR2 de 3GB e Sistema Operacional Linux Ubuntu 12.04 LTS: *The Precise Pangolin*. Apesar do processador possuir dois núcleos, por se tratar de um algoritmo na versão serial apenas um núcleo foi utilizado para os testes.

Nesta etapa foram efetuados testes em instâncias de ordem até 30 disponíveis na QAPLIB com o objetivo de analisar o **tempo de execução $\times$ qualidade das soluções**, esta última visando alcançar as soluções ótimas, ou melhores soluções encontradas para o problema até o momento.

Para as comparações entre os algoritmos ATIPR e GRASP, por serem algoritmos dependentes de componentes aleatórias, executou-se cada instância cinco vezes a fim de se obter o melhor, o pior e o valor médio dos tempos de execução e dos custos das soluções, para que pudessem ser comparados de uma forma mais justa com o Algoritmo HH. Diferentemente do ATIPR e do GRASP, o Algoritmo HH quando executado para uma mesma instância possui re-

sultado estático. Ainda em termos de comparação, o número de iterações médio dos cinco testes executados para cada instância do ATIPR foi usado como número de iterações do GRASP, para que este pudesse, de forma justa, ser executado nas mesmas condições.

Foram efetuados testes referentes a instâncias de tamanhos 12, 15, 20 e 30, onde para cada grupo foram organizadas duas tabelas, a primeira, onde se apresentam: o nome da instância, a melhor solução encontrada disponível na QAPLIB até o momento, os melhores resultados alcançados pelo ATIPR, os melhores resultados alcançados pelo HH, os melhores resultados alcançados pelo GRASP e a porcentagem de erro em relação aos resultados da QAPLIB do ATIPR, do HH e do GRASP, respectivamente. A segunda, onde se apresentam: o nome da instância, a média dos tempos do ATIPR para encontrar a melhor solução, o tempo gasto pelo HH para encontrar a melhor solução e a média dos tempos do GRASP para encontrar a melhor solução. Os tempos de execução dos algoritmos são mostrados em segundos.

Os melhores resultados alcançados estão destacados em negrito.

6.1.1 Instâncias de tamanho 12

A Tabela 6.1 apresenta os resultados obtidos com os testes referentes as instâncias chr12a, had12, nug12, rou12 e scr12. Para Burkard, Karisch e Rendl (1997), instâncias menores do que 20 são consideradas de pequeno porte e relativamente fáceis de se encontrar o resultado ótimo.

Tabela 6.1: Comparação dos custos para instâncias de tamanho 12

INSTÂNCIA	QAPLIB	ATIPR	HH	GRASP	% ATIPR	% HH	% GRASP
chr12a	9552	**9552**	**9552**	**9552**	0,00	0,00	0,00
had12	1652	**1652**	**1652**	**1652**	0,00	0,00	0,00
nug12	578	**578**	582	**578**	0,00	0,69	0,00
rou12	235528	**235528**	**235528**	**235528**	0,00	0,00	0,00
scr12	31410	**31410**	**31410**	**31410**	0,00	0,00	0,00

Observando-se a Tabela 6.1, percebe-se que para instâncias de pequeno porte, todos os algoritmos, em relação a qualidade de solução, obtiveram ótimos resultados. Excetuando-se o Algoritmo HH para a instância nug12, que teve um erro de **0,69%**, todas os outros testes alcançaram a solução ótima disponível para o problema pela QAPLIB.

A Tabela 6.2 apresenta os tempos de execução dos três algoritmos comparados na Tabela 6.1.

Para as instâncias chr12a, rou12 e scr12 percebe-se uma grande vantagem do HH, porém, na seção 6.1.5 efetuam-se testes onde os algoritmos, ATIPR e GRASP, são forçados a parar para

Tabela 6.2: Comparação dos tempos para instâncias de tamanho 12

INSTÂNCIA	ATIPR(s)	HH(s)	GRASP(s)
chr12a	0,222	**0,080**	0,300
had12	0,060	0,100	**0,042**
nug12	0,082	0,100	**0,058**
rou12	0,160	**0,080**	0,196
scr12	0,150	**0,080**	0,204

avaliar o tempo gasto para se encontrar a melhor solução de problema disponível na literatura. Tais testes foram efetuados por se perceber que algumas instâncias alcançam a solução ótima algumas centenas de vezes como, por exemplo, a instância chr12a, que durante a execução completa do algoritmo chegou a solução ótima aproximadamente **600** vezes, mostrando assim, que as soluções iniciais geradas para o algoritmo são de boa qualidade e que este é capaz de rapidamente alcançar a solução ótima do problema sem que para isso necessite executar todos os testes.

Para as instâncias nug12 e had12 os algoritmos ATIPR e GRASP obtiveram os melhores resultados.

6.1.2 Instâncias de tamanho 15

A Tabela 6.3 apresenta os resultados obtidos com os testes referentes as instâncias chr15a, nug15, rou15, scr15 e tai15a, tais instâncias na literatura, assim como as de tamanho 12, ainda são consideradas de pequeno porte e relativamente fáceis de se encontrar os seu resultado ótimo.

Tabela 6.3: Comparação dos custos para instâncias de tamanho 15

INSTÂNCIA	QAPLIB	ATIPR	HH	GRASP	% ATIPR	% HH	% GRASP
chr15a	9896	**9896**	10440	10070	0,00	5,50	1,76
nug15	1150	**1150**	1152	**1150**	0,00	0,17	0,00
rou15	354210	**354210**	360356	**354210**	0,00	1,74	0,00
scr15	51140	**51140**	53114	**51140**	0,00	3,84	0,00
tai15a	388214	**388214**	390782	**388214**	0,00	0,66	0,00

Observando-se a Tabela 6.3, percebe-se que apesar das instâncias serem ligeiramente maiores que as instâncias de tamanho 12, o ATIPR apresenta soluções ótimas em todos os testes, no caso do GRASP isso não ocorre apenas na instância chr15a. O Algoritmo HH, neste conjunto de testes, não alcançou a solução ótima em nenhuma oportunidade, mantendo ainda um alto percentual de erro no que diz respeito a qualidade da solução. Como exemplos, basta observar as instâncias chr15a com **5,50%** de erro e scr15 com **3,84%**.

A Tabela 6.4 apresenta os tempos de execução dos três algoritmos comparados na Tabela 6.3. Percebe-se claramente que o HH é o mais rápido dos 3 algoritmos, sendo superado apenas na instância nug15 pelo ATIPR, porém, como já dito, seus custos não são tão satisfatórios quando comparados aos algoritmos de componentes aleatórias (ATIPR e GRASP).

Tabela 6.4: Comparação dos tempos para instâncias de tamanho 15

INSTÂNCIA	ATIPR(s)	HH(s)	GRASP(s)
chr15a	1,324	**0,380**	1,784
nug15	**0,438**	0,470	0,442
rou15	0,804	**0,380**	0,804
scr15	0,896	**0,450**	1,298
tai15a	0,562	**0,400**	0,660

6.1.3 Instâncias de tamanho 20

A Tabela 6.5 apresenta os resultados obtidos com os testes referentes as instâncias chr20a, had20, nug20, rou20 e scr20, que já são consideradas de grande porte segundo Burkard, Karisch e Rendl (1997).

Tabela 6.5: Comparação dos custos para instâncias de tamanho 20

INSTÂNCIA	QAPLIB	ATIPR	HH	GRASP	% ATIPR	% HH	% GRASP
chr20a	2192	**2244**	2506	2246	2,37	14,32	2,46
had20	6922	**6922**	**6922**	**6922**	0,00	0,00	0,00
nug20	2570	**2570**	2596	**2570**	0,00	1,01	0,00
rou20	725522	**725522**	731348	**725522**	0,00	0,80	0,00
scr20	110030	**110030**	110058	**110030**	0,00	0,03	0,00

Observando-se a Tabela 6.5, percebe-se uma dificuldade dos três algoritmos para encontrar a solução ótima da instância chr20a, a exemplo do que ocorre com HH e o GRASP para o chr15a, seção 6.1.2. Neste conjunto de testes, os algoritmos ATIPR e GRASP encontram melhores soluções que o HH, com uma ligeira vantagem do ATIPR em relação ao GRASP de 0,09% na melhor solução encontrada para a instância chr20a, justamente a única para a qual o ótimo não foi encontrado.

A Tabela 6.6 apresenta os tempos de execução dos três algoritmos comparados na Tabela 6.5. É facil observar que os tempos alcançados pelo HH são consideravelmente menores, sendo superado apenas na instância had20 pelo ATIPR, porém, quando se avalia o conjunto **tempo de execução** $\times$ **qualidade das soluções**, percebe-se que para algumas instâncias essa relação não é muito boa. Por exemplo, a instância chr20a, apesar do seu tempo ser aproximadamente **3,61**

vezes menor que o tempo do ATIPR, sua porcentagem de erro em relação a solução ótima é aproximadamente **6,04** vezes maior em relação ao mesmo.

Tabela 6.6: Comparação dos tempos para instâncias de tamanho 20

INSTÂNCIA	ATIPR(s)	HH(s)	GRASP(s)
chr20a	10,366	**2,870**	17,102
had20	**3,938**	4,120	5,366
nug20	3,704	**3,450**	4,486
rou20	7,778	**2,870**	11,218
scr20	11,236	**3,680**	16,078

6.1.4 Instâncias de tamanho 30

A Tabela 6.7 apresenta os resultados obtidos com os testes referentes as instâncias kra30a, kra30b, nug30, tai30a e tho30, que já são considerados inpraticáveis quando se avalia a variável tempo, principalmente em algoritmos exatos, segundo Anstreicher et al. (2000).

Tabela 6.7: Comparação dos custos para instâncias de tamanho 30

INSTÂNCIA	QAPLIB	ATIPR	HH	GRASP	% ATIPR	% HH	% GRASP
kra30a	88900	**88900**	90760	90220	0,00	2,05	1,48
kra30b	91420	**91490**	91950	91810	0,08	0,58	0,43
nug30	6124	**6124**	6156	6146	0,00	0,52	0,36
tai30a	1818146	**1824150**	1858226	1846744	0,33	2,20	1,57
tho30	149936	**149936**	150810	150468	0,00	0,58	0,35

Observando-se a Tabela 6.7, a exemplo do que ocorre nas instâncias de tamanho 20, vide 6.1.3, o ATIPR mostra-se mais eficiente que o HH, porém, agora com uma boa vantagem em relação ao GRASP, que igualmente ao HH não alcançou o ótimo para nenhuma das instâncias executadas. Em apenas duas oportunidades o ATIPR não alcançou o valor ótimo, porém, a porcentagem de erro é tão pequena que justifica a eficiência do algoritmo, para a instância kra30b, o erro foi de **0,08%** em relação ao valor ótimo e para a instância tai30a, o erro foi de **0,33%**.

A Tabela 6.8 apresenta os tempos de execução dos três algoritmos comparados na Tabela 6.7. Percebe-se agora que não há um domínio absoluto do HH pois, para as instâncias kra30a e kra30b o ATIPR obteve os melhores resultados. Nas outras instâncias o HH obteve melhores tempos, mantendo também uma boa relação de **tempo de execução** $\times$ **qualidade das soluções**. Neste conjunto de testes o GRASP não apresentou resultados tão bons quanto os do ATIPR e do HH.

Tabela 6.8: Comparação dos tempos para instâncias de tamanho 30

INSTÂNCIA	ATIPR(s)	HH(s)	GRASP(s)
kra30a	**35,604**	56,860	44,148
kra30b	**36,948**	57,020	42,002
nug30	87,510	**67,850**	136,630
tai30a	185,146	**49,450**	265,404
tho30	98,400	**68,770**	137,844

6.1.5 ATIPR $\times$ GRASP - buscando as soluções ótimas

Nesta seção efetuaram-se conjuntos de testes com as instâncias de tamanho 12, 15, 20 e 30 já citadas nas seções anteriores, a fim de mostrar que os algoritmos ATIPR e o GRASP são capazes de encontrar, em grande parte das instâncias, as soluções ótimas dos problemas com poucas iterações, reduzindo-se consideravelmente o seu tempo de resposta computacional. Tais testes não foram realizados no HH, pois para instâncias maiores do que 12 são poucos os casos em que o algoritmo acha a solução ótima e por se tratar de um algoritmo que sempre irá retornar a mesma resposta. Caso este teste fosse efetuado, as respostas seriam exatamente as mesmas já descritas em seções anteriores, não havendo portanto a necessidade de executá-los novamente.

Para cada grupo de instâncias foi criada uma tabela onde se apresentam: o nome da instância, o custo da melhor solução encontrada até o momento, disponível na QAPLIB, a média dos tempos do ATIPR na busca das melhores soluções, os melhores resultados alcançados pelo ATIPR, a média dos tempos do GRASP na busca das melhores soluções e os melhores resultados alcançados pelo GRASP. No caso do GRASP foi adotado um número de 1600 iterações para a execução do algoritmo. O valor foi escolhido durante os testes, pois percebeu-se que com número de iterações a partir deste valor, quando o algoritmo não era capaz de alcançar a solução ótima, o seu tempo ultrapassava o tempo médio dos testes das seções anteriores, mostrando que nem sempre o algoritmo alcançava o ótimo com poucas iterações ou dentro de um tempo aceitável computacionalmente. Os melhores tempos alcançados estão destacados em negrito.

A Tabela 6.9 mostra os testes realizados para instâncias de tamanho 12. Comparando-se os tempos desta tabela com os da Tabela 6.2, percebe-se claramente uma diferença considerável no tempo do Algoritmo ATIPR na busca pela melhor solução. Como exemplos mais evidentes, pode-se observar as instâncias chr12a e scr12, onde o tempo foi reduzido em aproximadamente **84,23%** e **77,33%**, respectivamente. No caso do GRASP, houve também um redução de tempo em todas as instâncias, as mais consideráveis ocorreram nas instâncias chr12a e scr12, onde o tempo foi reduzido em média **68,75%** e **84,79%**, respectivamente. Neste conjunto de testes os

dois algoritmos alcançaram as soluções ótimas para todas as instâncias.

Tabela 6.9: Comparação dos tempos entre o ATIPR e o GRASP para instâncias de tamanho 12

INSTÂNCIA	**Custo QAPLIB**	**Tempo (s) ATIPR**	**Custo ATIPR**	**Tempo (s) GRASP**	**Custo GRASP**
chr12a	9552	**0,035**	9552	0,100	9552
had12	1652	0,034	1652	**0,033**	1652
nug12	578	**0,028**	578	0,073	578
rou12	235528	**0,156**	235528	0,177	235528
scr12	31410	**0,021**	31410	0,033	31410

A Tabela 6.10 mostra os testes realizados para instâncias de tamanho 15. Comparando-se os tempos desta tabela com os da Tabela 6.4, também é possível perceber uma diferença considerável no tempo do Algoritmo ATIPR na busca pela melhor solução. Pode-se observar as instâncias rou15 e scr15, onde o tempo foi reduzido em média **88,18%** e **84,22%**, respectivamente. No caso do GRASP, houve também um redução de tempo em três das cinco instâncias executadas, para a instância tai15a os valores foram mantidos, as mudanças mais consideráveis ocorreram nas instâncias chr15a, onde houve um acréscimo do tempo **152,13%** em relação ao valor anterior, porém, desta vez o valor ótimo foi alcançado. Para scr15, o tempo diminuiu em **65,33%** do valor anterior. Neste conjunto de testes, novamente, os dois algoritmos alcançaram as soluções ótimas para todas as instâncias.

Tabela 6.10: Comparação dos tempos entre o ATIPR e o GRASP para instâncias de tamanho 15

INSTÂNCIA	**Custo QAPLIB**	**Tempo (s) ATIPR**	**Custo ATIPR**	**Tempo (s) GRASP**	**Custo GRASP**
chr15a	9896	**0,752**	9896	4,498	9696
nug15	1150	**0,073**	1150	0,367	1150
rou15	354210	**0,095**	354210	0,650	354210
scr15	51140	**0,071**	51140	0,450	51140
tai15a	388214	**0,496**	388214	0,660	388214

A Tabela 6.11 mostra os testes realizados com o ATIPR para instâncias de tamanho 20. Comparando-se os tempos desta tabela com os da Tabela 6.6, apenas a instância chr20a não sofreu redução considerável no seu tempo, isso ocorre quando o algoritmo não é capaz de encontrar a solução ótima, assim, todas as iterações programadas devem ser executadas a fim de retornar a melhor solução possível dentro do seu espaço de busca. Relembrando, na Seção 6.1.3, os testes computacionais executam todas as iterações programadas.

Para os demais testes a exemplo das instâncias de ordem 12 e 15, a redução de tempo é considerável. Como exemplos mais evidentes para o ATIPR, pode-se observar as instâncias

had20, onde o tempo foi reduzido em aproximadamente **91,72%** e a instância nug20, com o tempo reduzido em aproximadamente **86,71%**. Para o GRASP apenas as instâncias chr20a e rou20 não alcançaram a solução ótima, tais instâncias estão marcadas com um asterisco para indicar que seus tempos foram muito altos pois tiveram de executar 1600 iterações antes de parar, sendo assim o tempo ficou mais alto que o tempo do teste anterior. A instância que chamou mais a atenção com relação a redução de tempo foi a had20, onde teve-se uma queda de **93,68%** em relação ao teste anterior. Neste conjunto de testes o ATIPR não encontrou a solução ótima apenas para a instância chr20a, com uma distância da solução ótima de **1,46%**, no caso do GRASP também não foi alcançado o ótimo para as instâncias chr20a e rou20, distanciando do ótimo de **6,11%** e **0,18%**, respectivamente.

Tabela 6.11: Comparação dos tempos entre o ATIPR e o GRASP para instâncias de tamanho 20

INSTÂNCIA	**Custo QAPLIB**	**Tempo (s) ATIPR**	**Custo ATIPR**	**Tempo (s) GRASP**	**Custo GRASP**
chr20a	2192	**10,361**	2224*	33,800*	2326
had20	6922	**0,326**	6922	0,339	6922
nug20	2570	**0,492**	2570	2,793	2570
rou20	725522	**1,129**	725522	36,716*	726812
scr20	110030	**3,843**	110030	7,172	110030

A Tabela 6.12 mostra os testes realizados para instâncias de tamanho 30. Comparando-se os tempos do ATIPR desta tabela com os da Tabela 6.8, duas instâncias, kra30b e tai30a não sofreram redução considerável no seu valor, como já mencionada anteriormente. Isso ocorre pois como o algoritmo não foi capaz de encontrar a solução ótima, executou assim, todas as iterações programadas.

Para os demais testes a exemplo das instâncias de ordem 12, 15 e 20, a redução de tempo é considerável. Para a instância kra30, o tempo foi reduzido em aproximadamente **36,89%** do teste anterior. Com relação ao GRASP pode-se destacar a instância tho30, que apesar do alto tempo de resposta, desta vez alcançou o valor ótimo.

Tabela 6.12: Comparação dos tempos entre o ATIPR e o GRASP para instâncias de tamanho 30

INSTÂNCIA	**Custo QAPLIB**	**Tempo (s) ATIPR**	**Custo ATIPR**	**Tempo (s) GRASP**	**Custo GRASP**
kra30a	88900	**22,470**	88900	317,683*	89800
kra30b	91420	**36,663**	91490	317,683*	91490
nug30	6124	**66,471**	6124	348,299*	6128
tai30a	1818146	**180,522**	1824150	281,183*	1850238
tho30	149936	**82,086**	149936	318,816	149936

6.2 O Algoritmo Paralelo

Após a análise do algoritmo ATIPR, é possível observar que todas as operações são sempre realizadas para cada φ ordenada de forma independente, por exemplo, a φ^{k+2} só será utilizada após o término das operações realizadas sobre a φ^{k+1}. Usando esta linha de raciocínio, é fácil perceber que o ATIPR é um algoritmo de fácil paralelização, onde a única comunicação que será efetuada entre os núcleos de processamento será para retornar a melhor solução de todos os processos envolvidos.

A paralelização para este tipo de problema se faz necessária, devido ao seu alto grau de combinatoriedade gerando uma quantidade exaustiva de cálculos. A ideia principal da paralelização é dividir grandes tarefas complexas em tarefas menores que sejam mais fáceis de ser executadas por diferentes processos, sendo que ao final de todos os processos, os melhores resultados são comparados a fim de retornar o melhor. Para este tipo de tarefa, computadores são interligados através de uma rede, com a finalidade de compartilhar as tarefas demandadas, a este tipo de configuração dá-se o nome de *cluster*. O *cluster* é uma ótima opção de alto desempenho computacional com baixo custo de investimento.

6.2.1 Configuração utilizada para o paralelismo

A versão paralela do Algoritmo ATIPR foi implementada na linguagem de programação C e os testes computacionais foram executados em uma estação de trabalho com processador Intel *Core* 2 Quad Q8200 2.33GHz $\times$ 4, memória RAM DDR2 de 6GB e Sistema Operacional Linux Ubuntu 12.04 LTS: *The Precise Pangolin*. Sem a disponibilidade de um *cluster*, nesta etapa, fez-se necessário o uso de apenas 4 núcleos disponíveis na estação de trabalho. Na conversão do algoritmo serial para o paralelo utilizou-se o padrão de comunicação MPI (*Messege Passing Interface*), padronizado pelo comitê do MPI Forum (www.mpi-forum.org). O MPI é uma de biblioteca padrão que trabalha o envio de mensagens em sistemas paralelos.

Seguindo a sugestão de Resendo e Rangel (2006) de se utilizar como número de processadores um valor que fosse divisor exato da dimensão da instância, com a intenção de se evitar que alguns processos se tornem ociosos durante a execução, escolheu-se instâncias que fossem múltiplas de 4, devido a disponibilidade do *hardware* utilizado neste trabalho. A exceção nesses testes foi o uso de instâncias de tamanho 50, a fim de avaliar o comportamento para instâncias de um porte maior que 40.

Nesta etapa foram efetuados testes em instâncias de ordem maior que 30 disponíveis na QAPLIB com o objetivo de analisar o **tempo de execução $\times$ qualidade das soluções**, esta

última, assim como no algoritmo serial, Seção 6.1, visando alcançar as soluções ótimas, ou melhores soluções encontradas para o problema até o momento.

Para as comparações com o HH, cada instância foi executado cinco vezes no ATIPR a fim de se obter o melhor, o pior e o valor médio dos tempos de execução e dos custos das soluções, para que pudessem ser comparados de uma forma mais justa. Como já citado na Seção 6.1 diferentemente do ATIPR, o Algoritmo HH quando executado para uma mesma instância possui resultado estático, não necessitando assim de várias execuções para cálculo de um valor médio.

Devido a limitação do *hardware* e ao alto tempo computacional de resposta, foram efetuados testes em instâncias de no máximo ordem 50. A apresentação dos resultados foram disponibilizadas em duas tabelas, a primeira, onde se apresentam: o nome da instância, a melhor solução encontrada disponível na QAPLIB até o momento, os melhores resultados alcançados pelo ATIPR, os melhores resultados alcançados pelo HH e a porcentagem de erro em relação aos resultados da QAPLIB do ATIPR e do HH, respectivamente. A segunda, onde se apresentam: o nome da instância, a média dos tempos do ATIPR para encontrar a melhor solução e o tempo gasto pelo HH para encontrar a melhor solução. Os tempos de execução dos algoritmos são mostrados em segundos.

Os melhores resultados alcançados estão destacados em negrito.

6.2.2 Instâncias de tamanho 36

A Tabela 6.13 apresenta os resultados obtidos com os testes referentes as instâncias ste36a, ste36b e ste36c, como já citado na Seção 6.1.4, tais instâncias, são consideradas de grande porte.

Tabela 6.13: Comparação dos custos para instâncias de tamanho 36

INSTÂNCIA	QAPLIB	ATIPR	HH	% ATIPR	% HH
ste36a	9526	9898	**9888**	3,76	3,66
ste36b	15852	**16294**	16332	2,71	2,94
ste36c	8239110	**8334128**	8422370	1,14	2,18

Observando-se a Tabela 6.13, percebe-se uma dificuldade dos dois algoritmos para encontrar a solução ótima para todas as instâncias. Neste conjunto de testes, os algoritmos ATIPR e HH possuem resultados que se equivalem, com uma ligeira vantagem do ATIPR, obtendo melhor resultado em duas das três instâncias executadas. Observando as duas últimas colunas, apenas o ATIPR para a instância ste36c obteve um erro de **1,14%** com relação ao ótimo. As demais instâncias, o erro médio foi de **3,05%**, considerando os dois algoritmos.

A Tabela 6.14 apresenta os tempos de execução dos dois algoritmos comparados na Tabela 6.13. Percebe-se que aqui também há um equilíbrio entre os algoritmos. Na instância ste36a o ATIPR leva uma boa vantagem em relação ao HH, ao contrário do que ocorre na instância ste36c, onde o HH leva uma vantagem considerável sobre o ATIPR. Para a instância ste36b os algoritmos têm praticamente o mesmo resultado, a exemplo do que ocorre também na Tabela 6.13. Observe que o ATIPR para a instância ste36a ganha em tempo mais perde em qualidade de solução para o HH, porém, para a instância ste36c é o HH quem ganha em tempo e perde na qualidade de solução, compare as tabelas 6.13 e 6.14.

Tabela 6.14: Comparação dos tempos para instâncias de tamanho 36

INSTÂNCIA	**ATIPR(s)**	**HH(s)**
ste36a	**96,98**	112,03
ste36b	**145,33**	146,10
ste36c	145,42	**125,43**

6.2.3 Instâncias de tamanho 40 e 50

Como já visto na Seção 6.2, devido as limitações de *hardware* foram executadas instâncias de no máximo ordem 50. O alto tempo computacional de resposta para o uso de apenas quatro núcleos de processamento, inviabilizou os testes com instâncias de ordens maiores. A Tabela 6.15 a seguir apresenta os tempos obtidos com os testes referentes as instâncias tai40a, tho40, tai50a e wil50.

Tabela 6.15: Comparação dos tempos para instâncias de tamanho 40 e 50

INSTÂNCIA	**ATIPR(s)**	**HH(s)**
tai40a	399,25	**189,72**
tho40	300,25	**293,43**
tai50a	2300,24	**1025,62**
wil50	2449,07	**1667,02**

Nesta fase, o HH obteve os melhores tempos em todos os testes, superando assim o ATIPR.

Quanto os custos, a Tabela 6.16 apresenta os valores alcançados para as instâncias tai40a, tho40, tai50a e wil50.

Observando-se a Tabela 6.16, percebe-se que os resultados alcançados para as instâncias tho40 e wil50 são de boa qualidade, porém há uma dificuldade dos dois algoritmos para encontrar as soluções ótimas para todas as instâncias, em particular as instâncias **tai**. Neste conjunto de testes, as respostas dos algoritmo para as instâncias tai40a e tai50a, tiveram comportamentos

Tabela 6.16: Comparação dos custos para instâncias de tamanho 40 e 50

INSTÂNCIA	QAPLIB	ATIPR	HH	% ATIPR	% HH
tai40a	3139370	3218062	**3212174**	2,45	2,27
tho40	240516	**242044**	243152	0,63	1,08
tai50a	4938796	**5069038**	5090380	2,57	2,98
wil50	48816	48986	**48890**	0,35	0,15

bem semelhantes com relação a aproximação da melhor solução para o problema. Com relação à instância tho40, o ATIPR se aproximou mais da solução encontrada na QAPLIB. No caso da instância wil50 os papéis se inverteram. Vale destacar que no caso das instâncias de ordem 50, além do melhor resultado obtido pelo HH, o tempo de execução foi significativamente melhor que o tempo do ATIPR.

7 Conclusão e Trabalhos Futuros

Neste trabalho foi desenvolvido um algoritmo que aplica o Teorema das Inversões com o objetivo de gerar soluções do PQA de boa qualidade. Pode-se notar através da análise dos resultados que as soluções obtidas pelo ATIPR alcançam as soluções ótimas ou as melhores disponíveis na QAPLIB em grande parte dos testes. A versão serial do algoritmo mostrou-se muito eficiente em termos de **qualidade de solução** $\times$ **tempo computacional**, excetuando-se a instância chr20a cujo erro com relação à solução ótimo foi igual a **2,46%**. Nas poucas instâncias que não alcançaram o ótimo, os erros não ultrapassaram **0,33%**. A versão paralela apesar dos bons resultados alcançados, ainda necessita da alguns ajustes para melhorar tanto o tempo, quanto a qualidade das soluções. O erro médio para todas as instâncias testadas, em paralelo, foi de **1,94%**. Sendo assim, conclui-se que a aplicação do Teorema das Inversões foi de grande valia para que o ATIPR obtivesse sucesso na qualidade das soluções apresentadas.

Como trabalhos futuros pretende-se continuar os testes com a versão paralela do algoritmo em instâncias maiores. Melhorias na geração das soluções iniciais devem ser pesquisadas e uma revisão detalhada na implementação do processo de busca local para melhorar seu desempenho deve ser efetuada. Além disso, investigar uma nova técnica de busca local que seja mais eficiente para aplicar no ATIPR, levando-se em conta a diminuição do número de inversões na geração de soluções vizinhas. Um estudo mais profundo da implementação paralela também será realizado a fim de avaliar a comunicação entre os processos. Todas essas possíveis alterações são de suma importância para tornar ATIPR competitivo em termos de desempenho computacional, visto que, em termos de qualidade de solução o algoritmo é bastante satisfatório.

Referências Bibliográficas

ABREU, N. M. M. *Um estudo algébrico e combinatório do problema quadrático de alocação segundo Koopmans e Beckmann*. Tese (Doutorado) — Programa de Engenharia de Produção - COPPE/UFRJ, Brasil, 1984.

ANSTREICHER, K. M. et al. *Solving Large Quadratic Assignment Problems on Computational Grids*. 2000.

BURKARD, R. E. et al. *QAPLIB - A Quadratic Assignment Problem Library*. [S.l.], 2013.

BURKARD, R. E. et al. *The Quadratic Assignment Problem*. 1998.

BURKARD, R. E.; KARISCH, S. E.; RENDL, F. Qaplib - a quadratic assignment problem library. *Journal of Global Optimization*, v. 1, p. 373–391, 1997.

BURKARD, R. E.; RENDL, F. A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operations Research*, v. 17, p. 169–174, 1983.

BURKARD, R. E.; STRATMAN, K. H. *Numerical investigations on quadratic assignment problems*. [S.l.]: Naval Research Logistics Quartely, 1978.

CELA, E. ¸ *The Quadratic Assignment Problem - theory and algorithms*. [S.l.]: Kluwer Academic Publishers, Series in Combinatorial Optimization, 1998.

DUMAN, E.; UYSAL, M.; ALKAYA, A. F. Migrating birds optimization: A new metaheuristic approach and its performance on quadratic assignment problem. *Information sciences*, v. 217, p. 65–77, 2012.

ELSHAFEI, A. N. *Hospital layout as a quadratic assignment problem*. [S.l.]: Operation Research Quartely, 1977.

GAMBARDELLA, L. M.; TAILLARD, E. D.; DORIGO, M. *Ant Colonies for the QAP*. [S.l.], 1997.

GAVETT, J. W.; PLYTER, N. V. The optimal assignment of facilities to locations by branch and bound. *Operations Research*, v. 14, p. 210–232, 1966.

GILMORE, P. C. Optimal and suboptimal algorithms for the quadratic assignment problem. *SIAM Journal on Applied Mathematics*, v. 10, p. 305–313, 1962.

GLOVER, F. Tabu search and adaptive memory programming advances, applications and challenges. *Interfaces in Computer Science and Operations Research*, v. 7, p. 1–75, 1996.

GUTIN, G.; YEO, A. Polynomial approximation algorithms for the tsp and the qap with a factorial domination number. *Discrete Applied Mathematics*, v. 119, p. 107–116, 2002.

JAMES, T.; REGO, C.; GLOVER, F. A cooperative parallel tabu search algorithm for the quadratic assignment problem. *European journal of operational research*, v. 195, p. 810–826, 2009.

KOOPMANS, T. C.; BECKMANN, M. J. Assignment problems and the location of economics activities. *Econometrica*, v. 25, 1957.

LAWLER, E. The quadratic assignment problem. *Management Science*, v. 9, p. 586–599, 1963.

LEE, L. *Reformulação do Problema de Isomorfismo de Grafos como um Problema Quadrático de Alocação*. Dissertação (Mestrado) — Programa de Pós-Graduação em Informática - UFES/PPGI, Brasil, 2007.

LI, Y.; PARDALOS, P. M.; RESENDE, M. G. C. A greedy randomized adaptive search procedures for the quadratic assignment problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, v. 16, p. 237–261, 1994.

LOIOLA, E. M. et al. A survey for the quadratic assignment problem. *European Journal of Operational Research*, v. 176, p. 657–690, 2007. ISSN 0377-2217.

MANIEZZO, V.; COLORNI, A.; DORIGO, M. *The Ant System Applied to the quadratic assignment problem*. [S.l.], 1994.

MCCORMIK, E. J. *Human Factors Engineering*. [S.l.]: McGraw-Hill, 1970.

MISEVICIUS, A. Genetic algorithm hybridized with ruin and recreate procedure: application to the quadratic assignment problem. *Knowledge-based systems*, v. 16, p. 261–268, 2003.

PARDALOS, P. M.; RENDL, F.; WOLKOWICZ, H. The quadratic assignment problem: a survey and recent developments. *DIMACS Series Discr. Math. Theor. Comp. Sci.*, v. 16, p. 1–42, 1994.

QUEIROZ, M. M.; MENDES, A. B. Metaheurística grasp com path relinking aplicada ao problema de programação de embarcações plv. *XLIII Simpósio Brasileiro de Pesquisa Operacional*, p. 1723–1734, 2011.

RANGEL, M. C. *Contribuições Algébricas ao Problema Quadrático de Alocação*. Tese (Doutorado) — Programa de Engenharia de Produção - COPPE/UFRJ, 2000.

RANGEL, M. C. *Uma Ampliação no Espaço de Busca para Soluções do Problema Quadrático de Alocação atrevés de uma Relaxação Linear*. [S.l.], 2001.

RESENDE, M. G. C. et al. Grasp and path relinking for the max-min diversity problem. *Computers & Operations Research*, v. 37, p. 498–508, 2010.

RESENDE, M. G. C.; RIBEIRO, C. C. *Grasp with path relinking: recent advances and applications.* [S.l.], 2003. 1-24 p.

RESENDO, L. C. *Um mapeamento de soluções do Problema Quadrático de Alocação(PQA) no universo de soluções do Problema de Alocação Linear (PAL)*. Dissertação (Mestrado) — Programa de Pós-Graduação em Informática - UFES/PPGI, Brasil, 2004.

RESENDO, L. C.; RANGEL, M. C. Um algoritmo construtivo baseado em uma abordagem algébrica do problema quadrático de alocação. *Pesquisa Operacional*, v. 26, p. 129–144, 2006.

SAHNI, S.; GONZALEZ, T. P-complete approximation problems. *Journal of the Association of Computing Machinery*, v. 23, p. 555–565, 1976.

SEYEDKASHI, S. M. H. et al. New simulated annealing algorithm for quadratic assignment problem. In: . [S.l.: s.n.], 2010. p. 105–110.

SIMOES, R. M. *Análise do comportamento da resolução do Problema Quadrático de Alocação através de instãncias isomorfas*. Dissertação (Mestrado) — Programa de Pós-Graduação em Informática - UFES/PPGI, Brasil, 2006.

STEINBERG, L. The backboard wiring problem: a placement algoritm. *SIAM Review*, v. 3, p. 37–50, 1961.

TATE, D. E.; SMITH, A. E. A genetic approach to the quadratic assignment problem. *Computers and Operations Research*, v. 22, p. 73–83, 1995.