COPPE/UFRJ

PQA: INVESTIGAÇÕES SOBRE A METAHEURÍSTICA VNS E SOBRE O USO DA VARIÂNCIA EM PROBLEMAS DE ISOMORFISMO DE GRAFOS

Valdir Agustinho de Melo

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Produção, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Produção.

Orientadores: Paulo Oswaldo Boaventura Netto
Laura Silvia Bahiense da Silva Leite

Rio de Janeiro
Setembro de 2010

PQA: INVESTIGAÇÕES SOBRE A METAHEURÍSTICA VNS E SOBRE O USO DA VARIÂNCIA EM PROBLEMAS DE ISOMORFISMO DE GRAFOS

Valdir Agustinho de Melo

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE PRODUÇÃO.

Examinada por:

Prof. Paulo Oswaldo Boaventura Netto, Dr.Ing.

Profa. Laura Silvia Bahiense da Silva Leite, D.Sc.

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

Profa. Lilian Markenzon, D.Sc.

Prof. Luiz Satoru Ochi, D.Sc.

Prof. Madiagne Diallo, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2010

Melo, Valdir Agustinho de

PQA: Investigações sobre a Metaheurística VNS e sobre o uso da Variância em Problemas de Isomorfismo de Grafos/Valdir Agustinho de Melo. – Rio de Janeiro: UFRJ/COPPE, 2010.

XIX, 128 p.: il.; 29, 7cm.

Orientadores: Paulo Oswaldo Boaventura Netto
Laura Silvia Bahiense da Silva Leite

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Produção, 2010.

Referências Bibliográficas: p. 102 – 112.

1. Problema Quadrático de Alocação. 2. Problema de Isomorfismo de Grafos. 3. VNS. I. Boaventura Netto, Paulo Oswaldo *et al.*. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Produção. III. Título.

Aos meus pais, Beatriz e Manoel,
que não tinham nada,
mas nos deram tudo.

Agradecimentos

Agradeço a Deus, por ter me concedido saúde para concluir os estudos;

Ao Professor Paulo Oswaldo Boaventura Netto, pelo voto de confiança, pelo auxílio sempre presente e por ter orientado meus estudos;

Ao Professora Laura Silvia Bahiense da Silva Leite, por também ter orientado meus estudos;

As Professoras Lilian Markenzon e Nair Abreu, pelos conselhos que muito me auxiliaram;

Aos Professores Luiz Satoru Ochi e Madiagne Diallo por terem aceito o convite de participar da Banca;

As pessoas da secretaria do Programa de Engenharia de Produção (em ordem alfabética): Andréia Moreira (PO), Claudete Lima, Diego Souza, Pedro Suevo e Roberta Arruda, que sempre resolveram as nossas necessidades acadêmicas;

Ao Programa de Engenharia de Produção da Universidade Federal do Rio de Janeiro, por ter fornecido a oportunidade de desenvolver os meus estudos;

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo suporte financeiro;

A minha família que tanto amo: Wal, João, Pedro e Chica, pelo suporte e incentivo contínuo;

A Emilia Matos, pelas pesquisas e pelo companheirismo nessa jornada;

Ao meu camarada Denis Silveira, pelo compartilhar das minhas agruras e pela amizade constante, não esquecendo a Eliane Loiola e a Juliana, que aturaram ambos;

Ao Leonardo Lima, pela amizade e aprendizado sobre a orientação de alunos;

Aos companheiros, que de alguma forma contribuíram nessa caminhada (em ordem alfabética): Andréa Bonifácio, Angelo Siqueira, Armando Gonçalves, Carla Oliveira, Carlos Eduardo, Juliana Bonfim, Lino Marujo, Marco Aurélio Leandro, Maria Aguieiras, Milena Estanislau e Ricardo Hamaoka;

As pessoas que amo muito e que sempre me incetivaram (em ordem alfabética): Almir Silveira, André Lopes (e Mike), Cecília Castelo Branco, Gloria Marins, Josélia Rocha, Luciana Pessoa, Luiza Franco, Maria Longo, Mercedes Barreto, Rita Cristina e Silvana Amodio.

Em Montréal, Canadá:

Ao Professor Pierre Hansen, que gentilmente aceitou orientar os meus estudos e pela presença constante nas minhas inquietações;

Aos Professores Gilles Caporossi e Silvain Perron, que me auxiliaram prontamente quando necessitei;

Especial agradecimento a Elivelton Bueno, amigo certo das horas incertas;

Ao suporte constante e amizade de Daniel Aloise, Caroline Rocha e Flávio Lima;

A Carole Dufour, Marie Perreault, que foram como anjos durante minha estadia;

Aos companheiros de jornada: Franck Belot, Hui Lin e Jin Jin.

"For what it's worth: it's never too late or, in my case, too early to be whoever you want to be. There's no time limit, stop whenever you want. You can change or stay the same, there are no rules to this thing. We can make the best or the worst of it. I hope you make the best of it. And I hope you see things that startle you. I hope you feel things you never felt before. I hope you meet people with a different point of view. I hope you live a life you're proud of. If you find that you're not, I hope you have the strength to start all over again."

The Curious Case of Benjamin Button

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

PQA: INVESTIGAÇÕES SOBRE A METAHEURÍSTICA VNS E SOBRE O USO DA VARIÂNCIA EM PROBLEMAS DE ISOMORFISMO DE GRAFOS

Valdir Agustinho de Melo

Setembro/2010

Orientadores: Paulo Oswaldo Boaventura Netto
Laura Silvia Bahiense da Silva Leite

Programa: Engenharia de Produção

Neste trabalho, apresentamos duas propostas a partir do Problema Quadrático de Alocação (PQA). Na primeira delas procuramos incorporar o uso de memória na metaheurística *Variable Neighborhood Search* (VNS), na tentativa de contribuir com resultados médios mais promissores para o PQA e na segunda proposta, o uso da variância do conjunto de soluções do PQA no estudo do Problema de Isomorfismo de Grafos (PIG) que, neste caso, foi modelado como um problema de PQA.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

QAP: INVESTIGATIONS ON THE VNS METAHEURISTICS AND ON THE USE OF ITS VARIANCE ON GRAPH ISOMORPHISM PROBLEMS

Valdir Agustinho de Melo

September/2010

Advisors: Paulo Oswaldo Boaventura Netto
Laura Silvia Bahiense da Silva Leite

Department: Production Engineering

In this work, we present two proposals for the Quadratic Assignment Problem (QAP). In the first proposal, we try to incorporate the use of memory in the metaheuristic Variable Neighborhood Search (VNS), in the attempt to contribute with more promising average results for QAP. In the second one, we look for invariant edge weight functions for a QAP instance built with two graphs composing the instances in order to try to find quantitative differences which would be associated with the absence of isomorphism.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Os avanços em processamento de cálculos na área de computação têm possibilitado melhorias e rapidez na solução dos problemas do dia-a-dia. Alguns desses problemas podem ser resolvidos em tempo polinomial, onde entradas de tamanho n, no pior caso, gastam um tempo computacional $O(n^k)$ para alguma constante k e, por isso, são considerados fáceis ou tratáveis, pertencendo a classe P (*Polynomial-time*). A questão é que nem todo problema pode ser resolvido em tempo polinomial, os quais pertencem a classe NP (*Nondeterministic Polynomial-time*). Segundo Cormen *et al* ((1)) são chamados NP-completos todos problemas para os quais ainda não foi descoberto um algoritmo que os resolva em tempo polinomial e de problemas NP-difícies aqueles que podem ser reduzidos a um problema cuja resolução pode ser feita em tempo polinomial. Comumente, são problemas onde se deseja maximizar ou minimizar uma função, cujas variáveis devem satisfazer ou não certas restrições. Tal função a ser otimizada pode ser contínua ou discreta: se ela for discreta, teremos um problema de Otimização Combinatória.

A maioria dos problemas de otimização discreta envolve um número grande, embora finito, de alternativas. Eles são encontrados em diferentes áreas de aplicação, onde é teoricamente possível enumerar todas as combinações de soluções e avaliar cada uma a respeito do objetivo esperado. A solução que apresentar o resultado mais favorável será certamente a ótima. Contudo, sob perspectiva prática, é inviável seguir esta estratégia, porque o número de possibilidades freqüentemente cresce exponencialmente com o tamanho do problema.

Encontrar soluções ótimas ou mesmo boas soluções aproximadas para esses problemas é um grande desafio, nem sempre fácil de ser vencido, mesmo com todo o aparato computacional disponível no momento. Esses problemas têm sido fonte de inspiração em pesquisas por métodos exatos e aproximados que possam "resolvê-los"em um tempo computacional admissível.

Um dos problemas que tem merecido grande atenção por parte dos pesquisadores, em vista do seu alto grau de complexidade computacional, é o Problema Quadrático

de Alocação, que será apresentado neste capítulo. Discutiremos algumas das suas formulações, os principais algoritmos exatos e heurísticos usados para sua resolução.

1.1 O Problema Quadrático de Alocação

Consideremos o problema de se alocar pares de atividades a pares de localidades, levando-se em conta os custos de se percorrer as distâncias entre as localidades e algum fluxo de unidades convenientemente definidas entre as atividades.

O Problema Quadrático de Alocação (PQA) consiste em encontrar uma alocação de custo mínimo das atividades às localidades, onde os custos são determinados pelo somatório dos produtos distância-fluxo.

O PQA ou Quadratic Assignment Problem (QAP), foi introduzido por Koopmans e Beckmann (2) como um modelo matemático relacionado a atividades econômicas. Segundo a maioria dos pesquisadores citados por Pardalos *et al* (3), o PQA pode ser visto como um problema de arranjo físico, o que viabiliza a sua utilização em importantes aplicações combinatórias, conhecidas como problemas de *lay-out*, que buscam a alocação ótima para um conjunto de instalações de uma empresa em diferentes locais previamente especificados para cada instalação. O *survey* sobre o PQA de Loiola *et al* (4) cita trabalhos que o utilizaram em planejamento de hospitais e na modelagem da localização de construções em um campus universitário. Nesse mesmo artigo, o PQA também aparece em diversas outras aplicações práticas, tais como a minimização da quantidade de ligações entre componentes de placas de circuitos eletrônicos; o desenvolvimento de um *framework* de decisões de alocação de uma nova facilidade (postos policiais, supermercados, escolas) que atenda a um dado conjunto de clientes; problemas de escalonamento de horários; definição do design de teclados e painéis de controle; aplicações em arqueologia, em análise de reações químicas, em análise estatística, em computação paralela e distribuída e em um problema relacionado a parques florestais. Conforme provado por Sahni e Gonzalez (5), o PQA é um problema NP-árduo (*NP-hard*). Mesmo dentro desta classe, sua complexidade computacional é muito elevada, uma vez que, em geral, instâncias de ordem superior a 40 não podem ser resolvidas em um tempo computacionalmente aceitável, o que tem atraído o interesse de diversos pesquisadores da área. Uma das instâncias mais tradicionais, a Nug30, foi proposta por Nugent *et al* em 1968 e somente em 2001 Anstreicher e Brixius (6), chegaram ao seu valor ótimo, através do desenvolvimento de um algoritmo *branch-and-bound*, cujo limite inferior se baseava em programação quadrática convexa.

Uma das primeiras e importantes contribuições teóricas sobre o PQA foi feita por Gilmore (7) e Lawler (8), respectivamente. Gilmore introduziu um limite inferior para o custo da solução ótima do PQA e Lawler, além de modificar o limite proposto

por Gilmore, apresentou uma formulação mais genérica para o problema. Esse novo limite ficou conhecido como Gilmore-Lawler e, até a presente data, é amplamente empregado pelos pesquisadores no estudo do PQA.

A partir da tese de doutorado de Abreu (9), que apresentou uma abordagem algébrica e combinatória para o problema, o Problema Quadrático de Alocação vem sendo estudado pelo Grupo de Grafos e Combinatória do Programa de Engenharia de Produção (PEP) da Coordenação de Pós-Graduação e Pesquisa de Engenharia (COPPE) da Universidade Federal do Rio de Janeiro (UFRJ), resultando em vários trabalhos publicados, tais como: Abreu e Boaventura (10), (11), Abreu *et al* (12), Querido *et al* (13), Vernet *et al* (14), Abreu *et al* (15), Abreu *et al* (16), Marins *et al* (17), Rangel *et al* (18), Rangel e Abreu (19), Boaventura e Loiola (20), Boaventura (21), Abreu *et al* (22), Boaventura (23), Marins *et al* (24), Loiola *et al* (25) e Loiola *et al* (4); e teses como: Querido (26), Moreira (27), Firmo (28), Loiola (29), Rangel (30), Marins (31) e Moreira (32).

1.2 Formulações do PQA

A formulação matemática para o PQA, inicialmente foi proposta por Koopmans e Beckmann (2) e pode ser vista como uma formulação de programação binária.

Tal formulação considera f_{ij} o fluxo entre as facilidades i e j e d_{kp} a distância entre as localidades k e p, cujo objetivo é calcular:

$$min \sum_{i,j=1}^{n} \sum_{k,p=1}^{n} f_{ij} d_{kp} x_{ik} x_{jp} \tag{1.1}$$

sujeito a

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

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

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

Se considerarmos o custo de alocação das atividades às localidades, uma formulação geral para instãncias do PQA de ordem n pode ser obtida através de três matrizes $F = [f_{ij}]$, $D = [d_{kp}]$ e $B = [b_{ik}]$ onde, respectivamente, as duas primeiras matrizes definem os fluxos entre as facilidades e as distâncias entre as localidades e, $[b_{ik}]$ como sendo os custos de alocação das facilidades às localidades. O problema

pode ser definido como:

$$min \sum_{i,j=1}^{n} \sum_{k,p=1}^{n} f_{ij} d_{kp} x_{ik} x_{jp} + \sum_{i,k=1}^{n} b_{ik} x_{ik} \tag{1.5}$$

sujeito a

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad 1 \leq j \leq n; \tag{1.6}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad 1 \leq i \leq n; \tag{1.7}$$

$$x_{ij} \in \{0, 1\} \qquad 1 \leq i, j \leq n. \tag{1.8}$$

Como o termo linear de 1.5 é fácil de ser resolvido, muitos autores o descartam.

Em 1963, Lawler (8) propôs uma versão mais geral de formulação matemática para PQA , onde os custos C_{ijkp}, não necessariamente correspondem ao produto de fluxos e distâncias. Os coeficientes C_{ijkp} descrevem o custo da alocação simultânea dos pares de facilidades i, j aos pares de localidades k, p. Esses custos são considerados por Koopmans e Beckman (2) como o produto de matrizes reais $n \times n$, onde $F = [f_{ij}]$ e $D = [d_{kp}]$ representam os fluxos entre as atividades i e j e as distancias entre às localidades k e p, respectivamente. Com isso, o problema pode ser expresso por:

$$min \sum_{i,j=1}^{n} \sum_{k,p=1}^{n} c_{ijkp} x_{ik} x_{jp} \tag{1.9}$$

sujeito a

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad 1 \leq j \leq n; \tag{1.10}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad 1 \leq i \leq n; \tag{1.11}$$

$$x_{ij} \in \{0, 1\} \qquad 1 \leq i, j \leq n. \tag{1.12}$$

As duas primeiras restrições, 1.10 e 1.11, obrigam à alocação de uma e somente uma atividade i a uma única localidade p.

Lawler (8) apresentou uma formulação linear para o PQA, introduzindo variáveis

binárias $y_{ijkp} = x_{ik}x_{jp}$, obtendo-se:

$$min \sum_{ijpq=1}^{n} c_{ijkp}y_{ijkp} \tag{1.13}$$

sujeito a

$$\sum_{i=1}^{n} x_{ik} = 1 \qquad p = 1, 2, ..., n; \tag{1.14}$$

$$\sum_{ijkp=1}^{n} y_{ijkp} = n^2; \tag{1.15}$$

$$\sum_{p=1}^{n} x_{ik} = 1 \qquad i = 1, 2, ..., n; \tag{1.16}$$

$$x_{ik} + x_{jp} - 2y_{ijkp} \geq 0, \qquad ijkp = 1, 2, ..., n. \tag{1.17}$$

onde

$$x_{ik} = \begin{cases} 1, & \text{se a atividade } i \text{ é alocada a } k; \\ 0, & \text{caso contrário.} \end{cases}$$

$$y_{ijkp} = \begin{cases} 1, & \text{se as unidades } i \text{ e } j \text{ são alocadas a } k \text{ e } p, \text{ simultaneamente;} \\ 0, & \text{caso contrário.} \end{cases}$$

Contudo, esta formulação não torna o PQA mais fácil, pois o número de variáveis é $(n^4 + n^2)$ e o de restrições é $(n^4 + 2^n + 1)$, o que dificulta uma formulação eficiente devido a natureza das restrições.

Numa visão grafo-teórica, uma instância do PQA simétrico, ou seja, onde F e D são matrizes simétricas, pode ser associada um par de cliques K_d e K_f, cujas arestas representam as distâncias (d_{ij}) e os fluxos (f_{kp}), respectivamente. Busca-se a permutação φ^* de vértices tal que a alocação dos vértices de uma clique sobre os da outra, gere uma alocação aresta $\times$ aresta, cujo somatório dos produtos das distâncias pelos fluxos seja o menor possível, o que corresponde a minimizar a função objetivo 1.19.

Gavett e Plyter (33) trabalharam com uma relaxação linear do problema. Com os vetores $F = [f_r]$ e $D = [d_s]$ tendo como componentes os respectivos valores das arestas de K_f e K_d, lexicograficamente ordenados, define-se a matriz Q, como $Q = FD^t = [\gamma_{rs}]$. Esta matriz tem a ordem do número de arestas, isto é, $N = n(n-1)/2$, sendo n o número de vértices. Na Figura 1.1, temos o exemplo de um grafo com

$n = 4$ e $N = 6$.

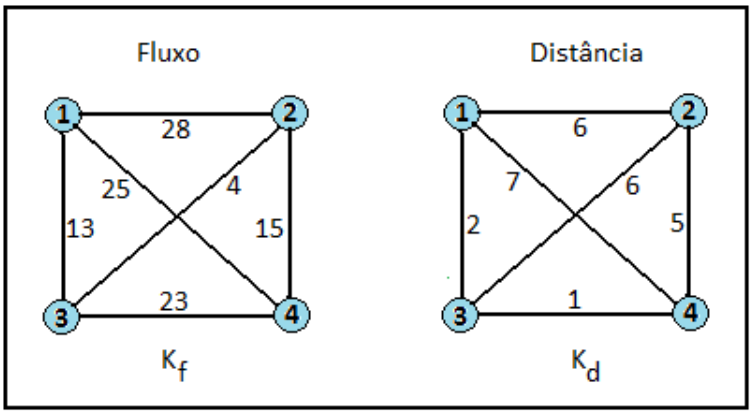

Figura 1.1: Instância de Gavett e Plyter

Com isso, temos para esta instância da Figura 1.1, os vetores de fluxo e distância da Figura 1.2:

F =	**28**	**13**	**25**	**4**	**15**	**23**
D =	**6**	**2**	**7**	**6**	**5**	**1**

Figura 1.2: Vetores de fluxo e distância para geração da matriz Q

que dão origem à matriz Q, representada na Figura 1.3:

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

Figura 1.3: Exemplo de uma matriz Q

Toda solução viável do PQA é uma solução para o problema relaxado. A recíproca, porém, não é verdadeira. O problema linear, relaxação do quadrático, tem $N!$ soluções, das quais apenas $n!$ são viáveis para o original. A bijeção:

$$\Psi(i, j) = (i - 1)n - [i(i + 1)/2] + j \tag{1.18}$$

associa os índices i e j das arestas das cliques aos índices r e s no problema relaxado.

Podem-se ordenar, por exemplo, as componentes de F em ordem não decrescente e as de D em ordem não crescente, obtendo-se respectivamente os vetores F^+ e D^-, e ainda a matriz $Q' = F^+ \times (D^-)^t$.

O traço (soma dos elementos da diagonal principal) da matriz Q' fornece um limite inferior para as instâncias do PQA, que no exemplo da Figura 1.1, corresponde a 389. Um limite superior é dado pela soma dos elementos da diagonal secundária de Q', que nesse caso é igual a 589. Usando a mesma instância do exemplo anterior temos os vetores F^+ e D^- na Figura 1.4, que dão origem a matriz Q', representada na Figura 1.5:

F+ =	**4**	**13**	**15**	**23**	**25**	**28**
D−=	**7**	**6**	**6**	**5**	**2**	**1**

Figura 1.4: Vetores de fluxo e distância para geração da matriz Q'

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

Figura 1.5: Exemplo de uma matriz Q'

Burkard e Stratman (34) propuseram uma formulação para o PQA genérico. Esta formulação considera que o conjunto de soluções de um PQA de ordem n pode ser associado ao conjunto de permutações dos elementos do conjunto $n = 1, ..., n$ para representar as alocações. Desta forma, as restrições 1.10 e 1.11 tornam-se desnecessárias. A função objetivo é representada simplesmente por:

$$\begin{array}{c} min \\ \varphi \in \Pi_n \end{array} \sum_{i,j=1}^{n} c_{ij\varphi(i)\varphi(j)} \tag{1.19}$$

onde π_n é o conjunto das permutações dos elementos de n, $(i) = k$ e $(j) = p$. Com esta representação das alocações por permutações, os custos na formulação de Koopmans e Beckmann (2) são dados pela expressão 1.20, a qual utilizamos nesta tese:

$$\begin{array}{c} min \\ \varphi \in \Pi_n \end{array} \sum_{i,j=1}^{n} f_{ij} d_{\varphi(i)\varphi(j)} \tag{1.20}$$

Outras formulações podem ser encontradas em Çela (35) ou em Loiola *et al* (4).

1.3 Métodos de Resolução

O crescimento exponencial do espaço de busca para as soluções das instâncias de muitos problemas de Otimização Combinatória, em função do crescimento das variáveis de decisão, muito comumente inviabiliza a procura de uma solução de valor ótimo através de algoritmos exatos, a partir de um limite que varia conforme o problema. Nesses casos, métodos aproximados, baseados em heurísticas ou metaheurísticas, são utilizados na tentativa de encontrar, rapidamente, soluções sub-ótimas de boa qualidade e têm sido objeto de estudos de diversos pesquisadores, principalmente a partir dos anos 90, como mostrado em Loiola *et al* (4).

Podemos dizer que heurísticas são técnicas que procuram soluções de boa qualidade a um custo computacional aceitável e que, geralmente, são desenvolvidas para um problema particular, podendo ser pouco eficientes quando aplicadas a outros problemas.

Tem-se acentuado a tendência para o uso de metaheurísticas, assunto que discutiremos no item 1.3.3.

1.3.1 Algoritmos Exatos

Uma grande variedade de algoritmos exatos tem sido proposta para resolver o PQA. Dentre os diferentes métodos utilizados, podemos citar os enumerativos como o *branch and bound*, e os métodos de planos de corte, introduzidos por Bazaraa e Sherali (36), ambos apresentando bons resultados, porém com uma convergência muito lenta, resolvendo, portanto, apenas pequenas instâncias do PQA, Kaufman e Broeckx (37), Bazaraa e Sherali (38), Burkard e Bonniger (39) e mais recentemente Miranda *et al* (40).

Uma combinação dos dois métodos citados é o *branch-and-cut*, que é uma variação proposta por Padberg e Rinaldi (41), aparecendo como uma estratégia alternativa de cortes que explora o politopo definido pelas soluções viáveis do problema. Nesse contexto têm-se descoberto novas propriedades básicas dos politopos que podem auxiliar no desenvolvimento de novos algoritmos: podemos citar a contribuição de Jünger e Kaibel (42–44), Padberg e Rijal (45), Kaibel (46) e Blanchard *et al* (47).

Basicamente, um algoritmo *branch and bound* utiliza-se de regras de corte e seleçãobaseadas na definição de limites inferiores e de uma estratégia de busca, sendo que um bom desempenho depende da qualidade do limite inferior gerado. Existem diversas referências disponíveis que utilizam esta abordagem, sendo as mais recentes, as contribuições de Mans *et al* (48), Bozer e Suk-Chul (49), Pardalos *et al* (50), Brüngger *et al* (51), Ball *et al* (52), Spiliopoulos e Sofianopoulou (53), Brixius e Anstreicher (54) e Hahn *et al* (55; 56). Nos últimos anos estão sendo muito utilizados procedimentos que combinam as técnicas de *branch and bound* com implementação

paralela, o que tem permitido progresso na resolução de instâncias maiores do PQA, onde podemos citar; Roucairol (57), Pardalos e Crouse (58), Mautor e Roucairol (59), Clausen e Perregaard (60) e Brungger *et al* (61), que solucionaram duas instâncias de Nugent *et al* encontradas na página QAPLIB (62).

Em vista do seu alto grau de dificuldade, até o presente momento instâncias do PQA, de ordem maior que 36, como muito das existentes na QAPLIB, ainda não foram resolvidas otimamente em um tempo computacional razoável. Podemos observar na referida página (62), por exemplo, que a solução ótima conseguida por Hahn para a instância Kra30b, foi conseguida com um tempo computacional equivalente a 182 dias, através do uso de uma única estação HP-3000, o que já foi um grande avanço se considerarmos que o tempo computacional necessário para resolver a instância Kra30b, por Anstreicher *et al*, foi equivalente a 2,7 anos, considerando a execução em uma estação análoga.

A instância Nug30, que foi proposta em 1968, foi resolvida em 2000, por Anstreicher *et al* (63), graças ao advento da meta-computação, técnica baseada na integração de computadores de diferentes plataformas integradas paralelamente.

1.3.2 Algoritmos Heurísticos

As heurísticas básicas para o PQA podem ser divididas em métodos construtivos e métodos de melhoria. Os métodos construtivos, introduzidos por Gilmore (7), em 1962, a cada iteração é atribuído um objeto a um local. Podem ser vistos como métodos gulosos já que, na iteração vigente, a escolha do próximo vértice é feita a partir da informação local dos vértices disponíveis. Métodos construtivos foram usados por Armour e Buffa (64), Buffa *et al* (65), Sarker *et al* (66; 67), Tansel e Bilen (68), Burkard (69), Arkin *et al* (70), Gutin e Yeo (71) e Yu e Sarker (72).

Como não há garantia de que somente os métodos construtivos produzam bons resultados médios, aplicam-se os métodos de melhoria sobre as soluções obtidas por eles, também conhecidos como algoritmos de busca local onde, a partir de uma solução viável, tais algoritmos procuram uma solução melhor na sua vizinhança, sendo o processo repetido até que nenhuma nova melhora seja obtida.

No final dos anos 90, surgem os procedimentos *multi-start*, que passaram também a ser usados para iniciar as heurísticas ou metaheurísticas. A cada iteração constrói-se uma solução viável, sem qualquer informação sobre as soluções geradas anteriormente. No entanto, para garantir o ótimo global, faz-se necessário chegar ao final do processo enumerativo, o que pode ser computacionalmente inviável. Uma possível solução para isso consiste em definir uma condição de parada, tal como: número máximo de iterações, tempo máximo de processamento, número máximo de iterações sem que haja uma melhora, etc. Nesta categoria, podemos encontrar

referências como Misevicius (73), Fleurent e Glover (74) , Misevicius e Riskus (75) e Feo e Resende (76).

1.3.3 Metaheurísticas

A primeira grande limitação dos métodos heurísticos clássicos reside no fato de que eles são, em sua maioria, dedicados a um problema específico. Pode-se citar, por exemplo, as heurísticas construtivas de 2-Tree e de Christofides e as heurísticas de melhoria de Lin-Kernighan e de Held-Karp, todas dedicadas ao Problema do Caixeiro Viajante. A segunda grande limitação se dá pela parada prematura em ótimos locais do problema.

A partir do final da década de 1980, surgem as primeiras metaheurísticas, o que facilitou a contrução de novos algoritmos, onde tais restrições ficaram menos visíveis.

Uma metaheurística consiste de um conjunto de regras (ou estratégias) genéricas que podem ser adaptadas a um dado problema, de acordo com suas características estruturais, gerando um algoritmo heurístico para a exploração do espaço de busca desse problema, com o intuito de tentar escapar de ótimos locais ainda distantes dos ótimos globais.

Sendo o PQA um dos mais difíceis problemas de otimização combinatória, o aparecimento das metaheurísticas aumentou o interesse dos pesquisadores no problema, já que resolvê-lo passou a ser uma maneira de testar a eficiência das metaheurísticas existentes. Em vista da sua dificuldade de resolução ótima, o PQA é considerado pelos especialistas como um bom *"benchmark"* para métodos de resolução, tanto exatos quanto heurísticos.

Algumas dessas metaheurísticas são baseadas em simulações dos processos naturais estudadas em conjunto com outro campo do conhecimento (metáforas), onde se destacam: *Simulated Annealing* (77), Algoritmos Genéticos (78) e Colônia de Formigas (79); e outras são baseadas diretamente em considerações teóricas e experimentais, tais como: *Scatter Search* (80), Busca Tabu (81; 82), GRASP - *Greedy Randomized Adaptive Search Procedures* (76), ILS - *Iterated Local Search* (83) e VNS - *Variable Neighbourhood Search* (84).

Em vista do alto grau de dificuldade do PQA, há uma tendência natural que existam mais trabalhos dedicados às heurísticas ou metaheurísticas do que a métodos exatos, o que pode ser constatado no *survey* de Loiola *et al* (4). Tal fato ocorreu, principalmente, a partir do início dos anos 90, como efeito do aparecimento das metaheurísticas, que impulsionaram novas pesquisas e métodos de resolução para os problemas de otimização combinatória.

Em 1996, Glover (85) propôs o *Path-Relinking* (PR), que não é uma metaheurística e sim um processo de pós-otimização, no qual somente um conjunto limi-

tado dos movimentos pode ser executado e onde somente os movimentos de melhora são permitidos, fazendo um balanço entre intensificação e diversificação.

Dentro das formas de uso das metaheurísticas se destacam os procedimentos híbridos, que são resultantes da composição de diferentes metaheurísticas. Uma justificativa para o seu uso é poder aproveitar as principais características dos métodos na criação de um novo algoritmo computacionalmente mais robusto. O uso de diversas propostas híbridas para o PQA e de diversas metaheurísticas é discutido e seus resultados são comparados e analisados em Maniezzo e Colorni (86) e em Taillard *et al* (87).

1.4 Características das instâncias do PQA

A biblioteca QAPLIB (62) contém cerca de 140 instâncias que são usadas freqüentemente por pesquisadores para comparar métodos. A ordem de grandeza dessas instâncias vai de 12 até 256 vértices, porém somente quatro delas são maiores que 100. Muitas dessas instâncias são relativamente bem resolvidas através de métodos heurísticos. Quase todos os métodos baseados em metaheurísticas são capazes de encontrar soluções com valores menores que 1% acima do melhor resultado conhecido das soluções. Métodos exatos têm dificuldades em provar a otimalidade das melhores soluções conhecidas para as instâncias contidas na QAPLIB. Algumas instâncias de tamanho menor que 30 ainda se encontram em aberto e até o presente momento nenhuma das instâncias de tamanho acima de 40 foi resolvida de maneira exata.

Segundo Drezner *et al* (88) e Taillard (89), alguns métodos heurísticos usados para resolver o PQA têm mostrado que sua eficiência depende fortemente do tipo de instância, para os quais foram utilizados. Com isso, um excelente método para um dado tipo de problema, pode ser ineficiente para outro. Conseqüentemente, se faz necessário adaptar o método ou usar um método apropriado para resolver um tipo de problema específico.

As instâncias do PQA podem ser divididas em cinco classes distintas, cujas as características são mostradas a seguir:

1.4.1 Classe 1 - Instâncias aleatórias com distâncias e fluxos uniformemente distribuídos

Os problemas deste tipo são gerados aleatoriamente através de uma distribuição uniforme. Isto faz com que eles sejam mais difíceis de resolver otimamente, mesmo sendo fácil de encontrar boas soluções. São amplamente usados na literatura porque grande parte deles somente foi resolvida de forma aproximada. Contudo, mesmo que ainda seja possível obter melhorias, mesmo que mínimas, Taillard (89) julga

que esse tipo de problema não seja muito interessante, já que todas as heurísticas recentes encontram boas soluções e o fato de que elas tenham melhorado o melhor valor conhecido para alguma solução, não constitui um prova de eficiência de um método, porém esta afirmação não é um consenso.

São exemplo, as instâncias e $Rouxx$ e $Taixxa$,onde xx sempre indica o tamanho da instância. Esse último conjunto de problemas é baseado da observação de que muitos métodos heurísticos disponíveis dependem de uma vizinhança da transposição, isto é, troca de duas facilidades. Para a sua construção, procurou-se por uma estrutura de problema para o qual o ótimo local (da vizinhança da troca) fosse relativamente distante e que a dificuldade das instâncias crescesse com o aumento do seu tamanho.

1.4.2 Classe 2 - Fluxos aleatórios em grades (*grids*)

As instâncias $Nugxx$ e $Skoxx$ são baseadas numa grade retangular, onde todos os nós não-adjacentes possuem peso zero. Desta maneira, uma troca dos pares da permutação ótima resultará em vários pares adjacentes que se tornarão não-adjacentes. Desse modo, o valor da função objetivo irá crescer rapidamente. A vizinhança do valor ótimo é constituída de soluções com valores da função objetivo muito acima desse, tornando-se muito difícil alcançar o ótimo a partir de uma solução na vizinhança desse.

1.4.3 Classe 3 - Problemas da vida real

Problemas da vida real são muito diferentes dos gerados aleatoriamente. A primeira observação que pode ser feita é que as matrizes de fluxos possuem um grande número de valores zerados e que eles não são uniformemente distribuídos. A segunda observação que pode ser feita pertence à estrutura do ótimo local, pois as permutações que correspondem aos ótimos locais privilegiam alguns locais para certas unidades. A seguir, estão agrupados problemas isolados que freqüentemente aparecem na literatura como aplicações práticas para o PQA.

1.4.4 Classe 4 - Instâncias aleatórias semelhantes aos problemas da vida real

Esta classe de problemas foi desenvolvida por Taillard (90), cuja proposta é um algoritmo que possibilite uma maneira simples de geração automática de problemas. Em tal procedimento de geração automática, os coeficientes das matrizes de fluxo e distância são inteiros, gerados aleatoriamente e uniformemente entre 0 e 99.

1.4.5 Classe 5 - Instâncias difíceis para as metaheurísticas

Em 2005, Drezner *et al* (88) propuseram uma nova classe de instâncias para o PQA, as $Drexx$ e $Taixxeyy$, bem como em 2000 Palubeckis (91), com as $Palxxx$. Tais instâncias são de difícil resolução para as heurísticas, principalmente para os mais recentes métodos que são construídos com a busca na vizinhança baseadas em troca de pares, já que apresentam ótimos locais com valores de função objetivo muito acima de 400% do ótimo, porém, são fáceis para os algoritmos exatos.

1.5 Objetivos deste trabalho

Muitos problemas de Otimização Combinatória de elevada complexidade encontram na Teoria dos Grafos processos de resolução aproximada capazes de produzir soluções sub-ótimas de boa qualidade dentro de um tempo computacional razoável. Nesses casos, métodos heurísticos baseados ou não em metaheurísticas são utilizados para encontrar tais resultados, onde se pode citar como exemplo o próprio PQA.

Como visto antes, em relação ao PQA, existem quatro questões de grande relevância a serem consideradas neste trabalho:

- Por ser um problema de fácil entendimento, porém de grande complexidade computacional, muitos pesquisadores o utilizam como um "*benchmark*"na validação de novas heurísticas;
- Assim como outros problemas combinatórios, o PQA pode modelar um grande número de aplicações práticas;
- Mesmo existindo muitos trabalhos dedicados ao VNS, a experiência disponível com esta metaheurística, no que concerne o PQA, segundo Loiola *et al* (4) é bastante reduzida, o que garante um campo de pesquisa com significativas oportunidades de inovação.
- Vários pesquisadores têm colaborado com diferentes abordagens na construção de estruturas internas de instâncias para PQA, o que dificulta ainda mais o desempenho dos algoritmos, como se observa em Stützle e Fernandes (92), pois faz com que diferentes algoritmos possam ser favoráveis a classes específicas de instâncias, o que amplia o interesse no desenvolvimento de novos algoritmos, sejam gerais, sejam mais adaptados a certas classes de instâncias.

Neste trabalho, apresentamos duas propostas de pesquisa relacionadas ao PQA. A primeira delas se refere a uma nova heurística e a segunda, ao uso da variância do conjunto de soluções do PQA no estudo do Problema de Isomorfismo de Grafos (PIG) que, nesse caso, foi modelado como um problema de PQA.

1.5.1 A heurística proposta

Nos últimos anos, algumas metaheurísticas foram propostas para problemas combinatórios e de otimização global. Partindo de uma solução inicial viável, gera-se uma seqüência de soluções intermediárias, na tentativa que haja uma melhora na função objetivo, o que geralmente conduz a bons resultados, não necessariamente ótimos.

Enquanto boas heurísticas são freqüentemente obtidas com alguma engenhosidade e vários parâmetros de ajuste, as razões do porquê elas conseguem bons resultados não são, em geral, conhecidas. A esse respeito, a situação é ainda mais obscura quando relacionada aos algoritmos híbridos. No entanto, conforme os autores, algumas propriedades desejáveis que visam garantir o interesse prático e teórico podem ser estabelecidas, as quais são apresentadas na lista a seguir:

- **Simplicidade**: a metaheurística deve ser baseada num princípio simples e claro, o qual deve ser amplamente aplicável;
- **Coerência**: todos os passos das heurísticas construídas para problemas particulares devem seguir os princípios básicos das metaheurísticas;
- **Eficiência**: heurísticas para um determinado problema devem prover soluções ótimas ou muito próximas do ótimo para todas, ou quase todas, dentre as instâncias disponíveis. Preferencialmente, elas devem encontrar soluções ótimas para quase todas as instâncias consideradas "benchmark";
- **Eficácia**: heurísticas para problemas particulares devem gastar um tempo computacional moderado para alcançar as soluções ótimas ou sub-ótimas;
- **Robustez**: o desempenho das heurísticas deve ser consistente sobre toda a variedade de instâncias, isto é, não apenas para um conjunto teste onde houve um ajuste fino e pior para as demais instâncias teste e em quase todas as execuçõesdo algoritmo numa dada instância;
- **Uso amigável**: heurísticas devem ser bem definidas, fácil de entender e, o mais importante, fáceis de usar. Isso implica que elas devem possuir o menor número de parâmetros possíveis, o ideal sendo que não haja parâmetro algum;
- **Inovação**: preferivelmente, os princípios das metaheurísticas e/ou de eficiência e eficácia de onde as heurísticas são produzidas devem conduzir para novos tipos de aplicações.

Em 1997, Hansen e Mladenovic̀ (84), propuseram a metaheurística VNS (Variable Neighborhood Search), que se baseia em uma sistemática troca de vizinhanças,

associada a um algoritmo aleatório na determinação de pontos iniciais da busca local.

Como será visto em detalhes no Capítulo 2, o esquema do VNS básico é muito simples e de fácil implementação. O algoritmo não faz uso de estruturas de memória para armazenar conhecimento, seja na construção da solução inicial, seja na mudança de vizinhança, seja na busca local.

Em relação ao VNS, nos últimos anos várias extensões têm sido propostas, principalmente as que visam auxiliar na solução de problemas com instâncias grandes (93). Em todas elas, um esforço significativo tem sido feito para preservar a simplicidade do esquema básico do VNS.

O uso de memória em metaheurísticas incorpora parte do modelo de multi-memória proposto em 1968 por Atkinson e Shiffrin (94), baseado na hipótese de como esse processo deve funcionar nos seres humanos. A memória de curto prazo, *short term memory*, é utilizada quando necessitamos de certas informações apenas em caráter imediato como, por exemplo, quando olhamos um número de telefone na agenda, fazemos a ligação e alguns segundos depois não nos lembramos mais do número que foi discado. Porém existem informações que guardamos na memória, podendo recordá-las depois de décadas: estas são associadas à memória de longo prazo, *long term memory*.

Nesta proposta, procuramos incorporar o uso de memória ao VNS, na tentativa de contribuir com resultados médios mais promissores para o PQA.

1.5.2 O estudo da Variância no Problema de Isomorfismo de Grafos (PIG)

Alguns problemas clássicos de Otimização Combinatória podem ser modelados via PQA, dentre eles podemos citar: Caixeiro Viajante, Empacotamento, Clique Maximal e o PIG.

Como descrito em Melo *et al* (95), dois grafos G_1 e G_2 são ditos isomorfos quando existe uma bijeção dos vértices de G_1 para os vértices de G_2 que preserva as arestas, ou seja, quando é possível permutar (ou re-rotular) os vértices de um dos grafos de maneira que um seja uma cópia exatamente igual ao outro.

Em termos práticos, o PIG é de grande importância em vários problemas, principalmente o de reconhecimento de padrões, como visto em Bunke e Shearer (96) e DePiero e Krout (97), porém, devido à alta complexidade computacional do PQA, a formulação desses problemas através dele torna-se muito menos eficiente do que outras formulações, como as de Programação Inteira. O caso do PIG é diferente, porque se trata de um problema cuja complexidade ainda não pôde ser estabelecida computacionalmente e que ainda não possui uma formulação muito eficiente via Pro-

gramação Inteira. Na literatura, é possível encontrar alguns estudos que fazem uso de métodos heurísticos aplicados ao PIG, tais como o VNS (95) e o GRASP (98).

Embora se possa determinar se dois grafos são ou não isomorfos através da resolução de uma instância PQA, esse processo envolve trabalhar com a complexidade do PQA. Por outro lado, a determinação da variância dos custos das soluções de uma instância PQA pode ser feita em tempo polinomial.

O capítulo 2 apresenta a metaheurística VNS e suas variantes. No capítulo 3 são estudados e desenvolvidos algoritmos a partir do VNS para resolução do PQA, utilizando-se como base os temas discutidos anteriormente. O capítulo 4 trata dos testes computacionais, detalhando as vizinhanças utilizadas e apresentando os resultados obtidos com a coleção de instâncias do PQA utilizadas e as avaliações dos resultados obtidos. No capítulo 5, descreve-se a segunda proposta deste trabalho que é o estudo da variância no PIG e no capítulo seguinte tem-se os testes realizados. Por último, apresentamos as conclusões, a proposta de pesquisa adicional, as referências bibliográficas e os anexos.

Capítulo 2

Variable Neighborhood Search

Em 1997, Hansen e Mladenovìc (84), propuseram uma metaheurística que se baseia em uma troca sistemática de vizinhança, associada a um algoritmo aleatório na determinação de pontos iniciais da busca local, chamada de Busca em Vizinhança Variável, conhecida na literatura em inglês como *Variable Neighborhood Search* (VNS). Contrariamente a outras metaheurísticas baseadas em métodos de busca local, VNS não segue uma trajetória, mas explora incrementalmente vizinhanças mais ou menos distantes da solução corrente, indo da solução atual para a nova se e somente se uma melhora ocorrer. Desta maneira, freqüentes características da solução atual são guardadas e utilizadas para a obtenção de soluções vizinhas promissoras, o que não ocorre com as heurísticas que utilizam apenas uma única estrutura de vizinhança. Esta unicidade pode restringir o desempenho da heurística e, caso a estrutura escolhida não seja adequada ao problema, a qualidade das soluções obtidas pode ser comprometida. Diferentemente de outras metaheurísticas, a estrutura do VNS básico e suas extensões são simples e requerem poucos, e às vezes nenhum parâmetro. Segundo Hansen e Mladenovìc (84), a vantagem do uso de várias vizinhanças é que o ótimo local, em relação a uma vizinhança, não é necessariamente o mesmo de outras: dessa forma, a procura deve continuar de uma maneira descendente (ou ascendente) até que a solução corrente seja um mínimo (ou máximo) local de todas as estruturas de vizinhanças pré-selecionadas.

O algoritmo básico do VNS pode ser visto na Figura 2.1:

A inicialização (*linha* 1) consiste na definição de um conjunto de vizinhanças e de um critério de parada e, ainda, da obtenção de uma solução inicial de forma totalmente aleatória. O algoritmo se inicia com a primeira vizinhança (*linha* 3), cuja condição de parada (*linha* 2), pode ser o tempo máximo de uso de CPU, o número máximo de iterações ou o número máximo de iterações entre duas melhoras ocorridas.

Uma iteração do VNS ocorre a partir da *linha* 4, enquanto a contagem de vizinhanças atual (k) for menor ou igual a vizinhança máxima (K_{max}):

Algoritmo 1 : *VNS Básico*

1 **Inicialização:**
Selecionar o conjunto de estruturas de vizinhanças N_k que serão usadas na busca;
Encontrar uma solução inicial x;
Escolher um critério de parada;
2 **Enquanto** (critério de parada não satisfeito) **faça**
3 $k \leftarrow 1$;
4 **Enquanto** ($k \leq K_{max}$) **faça**
5 **Agitação:**
Gerar aleatoriamente uma solução x' na *k-ésima* vizinhança de x ($x' \in N_k(x)$);
6 **Busca Local:**
Aplicar um método de busca local em x', obtendo um ótimo local x'';
7 **Mover ou não:**
Se o ótimo local é melhor que o atual **então**
8 mover para lá ($x \leftarrow x''$),
9 ir para a primeira vizinhança ($k \leftarrow 1$);
senão
10 passar para a próxima vizinhança ($k \leftarrow k + 1$);
fim Se
fim Enquanto
fim Enquanto

Figura 2.1: Algoritmo básico do VNS

- é feita uma *agitação* (*shaking*, na literatura em inglês) na solução x, ou seja, gera-se um ponto aleatório na vizinhança atual, correspondente a uma solução x'. A geração aleatória procura evitar evitar repetições, o que poderia ocorrer se alguma regra determinística fosse usada (*linha* 5);

- aplica-se uma busca local à solução x', gerando uma solução x'' (*linha* 6);

- se a solução x'' possuir um custo menor que a solução x (*linha* 7), passa-se a adotar x'' como sendo a solução atual x (*linha* 8), e há um retorno para a primeira vizinhança (*linha* 9);

- caso contrário, passa-se para a próxima vizinhança (*linha* 10).

2.1 Variantes do VNS

Além da versão básica do VNS, Hansen e Mladenovic (93) propuseram variantes, tais como o *Variable Neighborhood Descent*, *Reduced Variable Neighborhood Search* e *General Variable Neighborhood Search*, além de extensões, tais como *Skewed Variable Neighborhood Search* e *Parallel Variable Neighborhood Search*, cujas definições encontram-se a seguir.

2.1.0.1 *Variable Neighborhood Descent* (VND)

A *Variable Neighborhood Descent* (VND) ou Máxima Descida em Vizinhança Variável envolve também a substituição da solução atual pelo resultado da busca local, quando há uma melhora, porém a estrutura de vizinhança é trocada de forma determinística cada vez que se encontra um mínimo local. A solução resultante é um mínimo local em relação a todas as K_{max} estruturas de vizinhança exploradas. Existem outras versões do VND. A características principal do VND é sua monotonicidade o que, dependendo da solução inicial, das características da instância do problema e das estruturas de vizinhança, freqüentemente, limita-se a uma pequena região do espaço de busca.

2.1.1 *Reduced Variable Neighborhood Search* (RVNS)

A *Reduced Variable Neighborhood Search* (RVNS) ou Busca em Vizinhança Variável Reduzida procura escolher aleatoriamente um ponto na vizinhança atual, que poderá ser ou não um ótimo local. Se o seu valor é melhor que o atual, a busca é reiniciada a partir desse ponto, sem voltar para a primeira vizinhança como no VNS básico; caso contrário, busca-se um ponto aleatório na próxima vizinhança. Depois que todas as vizinhanças tenham sido consideradas, volta-se para a primeira, o que ocorre até que o critério de parada seja satisfeito, que usualmente é o máximo de tempo de execução desde a última melhora ou um número máximo de iterações.

Na prática, pode ser vista como a versão básica do VNS, porém com a supressão da busca local. Esta versão não é tão eficiente quanto às anteriores por ser totalmente aleatória, a vantagem está na sua aplicação a instâncias de problemas muito grandes, onde a busca local é muito custosa.

2.1.2 *General Variable Neighborhood Search* (GVNS)

A *General Variable Neighborhood Search* (GVNS) ou Busca em Vizinhança Variável Geral é uma fusão do VNS com o VND, ou seja, a busca local do VNS é feita utilizando-se o VND. Em alguns casos, tal junção pode apresentar resultados mais promissores que suas versões isoladas.

2.1.3 *Skewed Variable Neighborhood Search* (SVNS)

O VNS geralmente apresenta resultados tão bons ou melhores que os dos métodos de partida múltipla, mais ainda quando existem muitos ótimos locais. Muitos problemas possuem um agrupamento de ótimos locais, que facilitam a busca do ótimo global, porém pode acontecer que algumas instâncias possuam vários ótimos locais separados e distantes. Se, ao tentarmos melhorar a busca, considerarmos vizinhanças maiores, isto pode degenerar o VNS para um método de partida múltipla, já que a informação relacionada ao melhor ótimo local atual é "perdida". Para evitar tal situação, a *Skewed Variable Neighborhood Search* (SVNS) ou Busca em Vizinhança Variável Geral Enviesada introduz uma função $\rho(x, x'')$ que controla a distância entre as soluções produzidas (métrica dentro das estruturas de vizinhança).

2.1.4 *Parallel Variable Neighborhood Search* (PVNS)

A *Parallel Variable Neighborhood Search* (PVNS) ou Busca em Vizinhança Variável Geral Paralelizada, como visto na própria definição, é a versão paralelizada do VNS e que pode ser uma opção eficaz desde que a metaheurística paralela permita buscas múltiplas simultâneas atuando sobre o domínio do problema, realizadas em regiões diferentes. Isto acelera a busca, reduz o tempo computacional e pode fornecer uma solução final melhor qualidade.

2.1.5 *Algoritmo híbrido* ILS-RVNS

Mais recentemente, em 2010, Subramanian *et al* (99) propuseram a versão híbrida ILS-RVNS onde, a cada iteração do algoritmo, uma estrutura de vizinhança do VNS é selecionada de maneira aleatória.

2.2 Estruturas de Vizinhança

As heurísticas de busca local, em geral, possuem apenas uma estrutura de vizinhança, isto é, $K_{max} = 1$.

O VNS utiliza um conjunto pré-definido de estruturas de vizinhança N_k, ($k = 1, ..., K_{max}$), e com $N_K(x)$ o conjunto de soluções na k-ésima vizinhança de x.

Como descrito em (100), usualmente, cada nova vizinhança procura se distanciar ainda mais da solução inicial. Outra possibilidade é explorar as vizinhanças de maneira aninhada, isto é, onde a estrutura atual também contém a estrutura anterior.

Um grande desafio na escolha das estruturas de vizinhança é a definição da quantidade a ser utilizada num problema já que, quanto maior o seu número, maior será

o custo computacional o que, necessariamente, não traz uma melhora proporcional ao seu aumento. Isto ocorre porque, a cada iteração, serão verificadas K_{max}) vizinhanças.

Outro desafio é a seleção de movimentos interessantes e que sejam de baixo custo computacional. Sempre que possível, de ve-se procurar incorporar alguma informação sobre o problema em estudo, já que existem estruturas que se aplicam mais a certos problemas de Otimização Combinatória do que outros.

Para ilustrar melhor o conceito de vizinhança, mostramos na Figura 2.2, todas as vizinhanças possíveis para uma instância PQA $n = 4$ e *umatroca* onde, a partir de solução viável, o seu vizinho imediato é obtido através da troca do conteúdo de uma posição com outra posição imediatamente anterior ou posterior a esta:

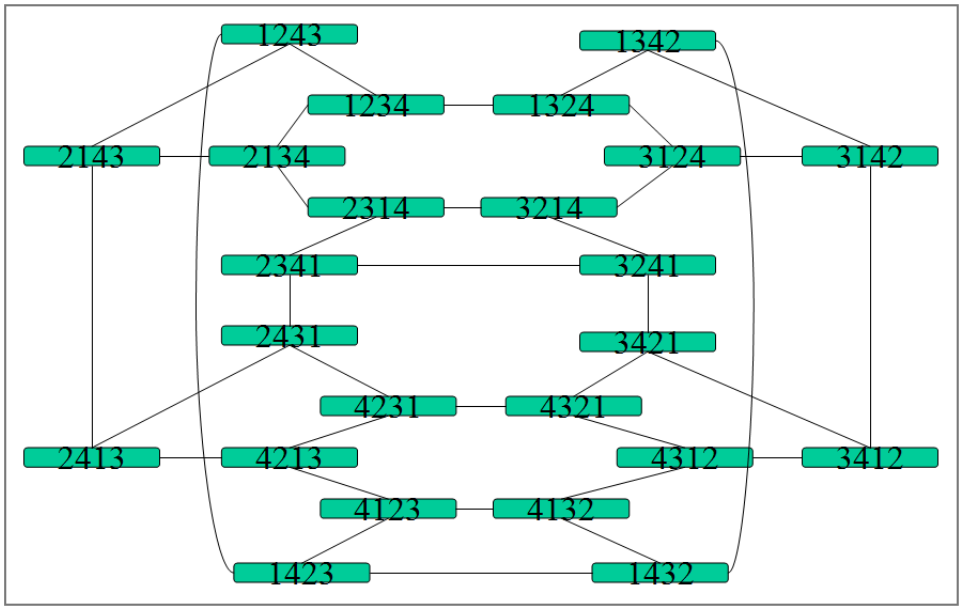

Figura 2.2: Todas as vizinhanças possíveis para $n = 4$ e $1 - troca$

Outro exemplo mais específico seria uma estrutura de vizinhança, muito utilizada no Problema do Caixeiro Viajante, que é a $r - Optimal$ ou $r - opt$, e que consiste em remover r arestas de vértices não adjacentes, produzindo r caminhos desconectados. Reconectam-se esses r caminhos de segundo um critério dado para produzir outro caminho, usando diferentes arestas daquelas que foram removidas. Desta maneira, a nova solução será diferente da anterior por exatamente r arestas. O custo de avaliação de cada vizinho é feito em $O(1)$ e cada iteração da busca local é $O(n^r)$. Em vista do alto custo computacional, geralmente se usa apenas o $2 - opt$.

Em 1991, Taillard (90) propôs uma busca tabu robusta para o PQA, cuja complexidade é $O(N)$, onde $N = n(n-1)/2$. Nesse artigo, partindo da localização ϕ, a vizinhança π da localização é obtida permutando as unidades r e s:

$$\begin{aligned}
\pi(k) &= \phi(k) \qquad \forall k \neq r, s \\
\pi(r) &= \phi(s) \\
\pi(s) &= \phi(r).
\end{aligned}$$

Se as matrizes são simétricas e com a diagonal principal nula (como ocorre no caso clássico), o valor do movimento, escrito $\triangle(\phi, r, s)$, é dado por:

$$\begin{aligned}
\triangle(\phi, r, s) &= \sum_{i=1}^{N}\sum_{j=1}^{N}(a_{ij}b_{\phi(i)\phi(j)} - a_{ij}b_{\pi(i)\pi(j)}) \\
&= 2.\sum_{k\neq r,s}(a_{sk} - a_{rk})(b_{\phi(s)\phi(k)} - b_{\phi(r)\phi(k)}).
\end{aligned}$$

Se as matrizes não são simétricas e/ou com a diagonal não nula, a expressão, ainda de complexidade $O(N)$, é um pouco mais complicada, como descrito em Burkard e Rendl (101). Contudo, se a troca das unidades r e s numa localização ϕ fornece a localização π, é possível (para u e v diferentes de r ou s) calcular o valor $\triangle(\phi, r, s)$ num tempo constante se for armazenado o valor de $\triangle(\phi, r, s)$ do passo anterior da busca tabu.

$$\begin{aligned}
\triangle(\pi, u, v) &= \triangle(\phi, u, v) + 2(a_{ru} - a_{rv} + a_{sv} - a_{su})(b_{\pi(s)\pi(u)} - b_{\pi(s)\pi(v)} + b_{\pi(r)\pi(v)} - \\
& b_{\pi(r)\pi(u)}).
\end{aligned}$$

Se u ou v é igual a r ou s, é possível calcular o valor de $\triangle(\phi, r, s)$ através do uso da expressão anterior. Conseqüentemente, a avaliação de toda vizinhança leva $O(N^2)$ operações, como foi mostrado em Frieze *et al* (102).

A avaliação rápida da qualidade dos movimentos é um fator importante, contribuindo para a eficiência da busca. Neste trabalho, usaremos como estratégia de busca local esta contribuição de Taillard.

Capítulo 3

Uso de memória no VNS básico

Como já visto anteriormente, o VNS se baseia num princípio simples: mudanças sistemáticas de vizinhanças e uso de uma busca local. Tal conceito vem sendo aplicado em vários problemas de Otimização, com bons resultados encontrados na literatura, o que pode ser visto em Hansen *et al* (103).

Conforme discutimos anteriormente, o PQA é um dos problemas mais difíceis da Otimização Combinatória, o que faz com que instâncias de ordem $n > 36$, em geral, não possam ser resolvidas num tempo computacional aceitável. Suas instâncias-teste, encontradas na QAPLIB, são de diferentes graus de dificuldade e, conforme Taillard (89), a eficiência dos métodos depende fortemente do tipo de problema que está sendo abordado. Portanto, um excelente método para um dado tipo de problema pode ser ineficiente para outro tipo. Além disso, poucos pesquisadores comparam seus métodos com outros em diferentes tipos de problemas. No seu trabalho, Taillard apresenta uma análise e discute particularidades de alguns problemas freqüentemente tratados na literatura.

Drezner *et al* (88) e Palubeckis (91) contribuiram com novas instâncias difíceis para o PQA, não disponibilizadas na QAPLIB, porém encontram-se na internet em (104) e (105), respectivamente.

Neste trabalho procuramos incorporar o uso de memória ao VNS, na tentativa de contribuir com resultados médios mais promissores para o PQA. Com esse propósito, as técnicas usadas neste trabalho recebem o nome de *Memorized Variable Neighborhood Search*, podendo também ser *Memorized VNS* ou simplesmente MVNS. O escopo desta contribuição será restrito à versão básica do VNS, não incluindo suas variantes, tais como o VND, RVNS e GVNS ou extensões, tais como SVNS e PVNS, que também podem ser encontradas em Hansen e Mladenovìc (93).

Importante ressaltar que, diferentemente de outras metaheurísticas, esta proposta não associa o controle de memória que irá guardar os movimentos ou vértices:

- a um parâmetro externo;
- à iteração atual;
- ao cálculo do custo associado a cada vértice, que é uma operação nem sempre fácil de manter atualizada;

e sim, procura fazer uso apenas das propriedades das vizinhanças do VNS, tentando também preservar a lista de propriedades desejáveis em uma metaheurística, encontrada em Hansen e Mladenovìc (93) e citada anteriormente, no final do Capítulo 1.

Um fato relevante entre as versões aqui propostas é que estas exigem pouca mudança no seu código, sendo que algumas delas aproveitam toda a estrutura de dados e os controles, minimizando o trabalho de programação, pois há um reaproveitamento de código, o que simplifica o teste de novas versões, mesmo que isso não indique uma redução no tempo de processamento. Para todas elas, foram consideradas duas variações de controle por memória, cujo controle se restringe sempre a iteração e vizinhança atual:

- **Controle por vértice** - se o vértice foi usado, sua posição não poderá ser trocada durante uma ou mais vezes;
- **Controle por movimento** - se um movimento de troca é feito, não poderá ser utilizado novamente durante uma ou mais vezes.

Uma justificativa para o uso do *controle por vértice* pode ser vista na Figura 3.1, pois sendo a vizinhança atual k par e $k > 1$, é possível que haja, quando da agitação da solução vigente, um retorno à solução da vizinhança imediatamente anterior $(k-1)$, também conhecida como agitação no VNS básico.

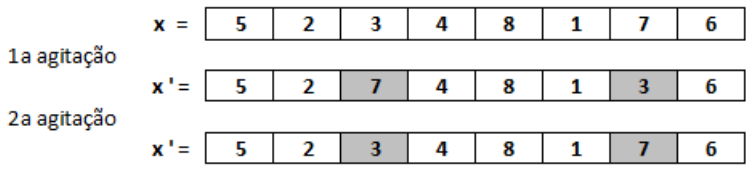

Figura 3.1: Exemplo de um possível retorno a solução anterior no VNS básico

Apesar de evitar um possível retorno a uma solução da vizinhança anterior, a versão *controle por vértice* tem o inconveniente de limitar as próximas escolhas, pois evita que apenas uma das posições usadas na vizinhança anterior, isoladamente, possa ser associada a outra posição qualquer na vizinhança atual. Tal restrição pode comprometer a eficiência do algoritmo, o que motivou a geração do *controle por movimento*. Na Figura 3.2, tem-se um exemplo de trocas sem repetição de

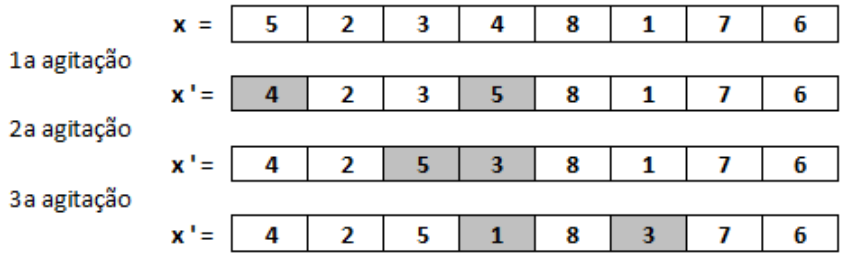

Figura 3.2: Exemplo de movimentos inibidos pelo controle por vértice

solução que seriam excluídas na versão de memória anterior, por fazerem uso da quarta posição do vetor solução.

O controle de memória inserido no VNS básico terá sua atuação **restrita** aos vértices ou movimentos candidatos a escolha na agitação da vizinhança atual, isto é, sua abrangência não afeta a busca local. A inserção desse controle acontece em dois pontos bem definidos do VNS básico, como pode ser visto na Figura 3.3:

(a) Memória da agitação feita na vizinhança atual

Procura auxiliar na condução da geração das próximas soluções, isto na fase de agitação da vizinhanca atual ($x \longrightarrow x'$). Tal controle usa o mesmo princípio da metaheurística Busca Tabu, Glover (81; 82), reduzindo assim o risco do efeito de ciclagem (possibilidade de retorno a uma solução anterior próxima a atual);

(b) Memória da melhora na vizinhança atual

O seu uso está associado apenas às melhoras ocorridas na fase de busca local ($f(x'') < f(x)$), ou seja, quando estas geram uma nova solução x'' melhor que a solução atual x, sendo o uso dessas informações preservado na geração das próximas vizinhanças. Isto equivale a um dos esquemas de cruzamento (*crossover*) da metaheurística Algoritmos Genéticos, Holland (78), onde se procura fazer com que as boas características dos pais sejam herdadas pelos filhos.

Esses controles são exercidos até que não haja mais possibilidade de escolhas aleatórias na agitação da vizinhança atual ou até que aconteça uma melhora na vizinhança atual, o que ocorrer primeiro. O controle é binário, isto é, quando um vértice ou posição de movimento estiver com valor um (1), ele será ignorado na próxima escolha aleatória das posições que irão compor a solução x' da vizinhança atual.

Procurou-se fazer uso do controle de memória no VNS básico em quatro situações bem definidas, as quais vão de um esquema bem simples, cujo controle é único, até um mais elaborado, com controle duplo:

Algoritmo 1 : *VNS Básico*

1 **Inicialização:**
Selecionar o conjunto de estruturas de vizinhanças N_k que serão usadas na busca;
Encontrar uma solução inicial x;
Escolher um critério de parada;
2 **Enquanto** (critério de parada não satisfeito) **faça**
3 $k \leftarrow 1$;
4 **Enquanto** ($k \leq K_{max}$) **faça**
(a) 5 **Agitação:**
Gerar aleatoriamente uma solução x' na *k-ésima* vizinhança de x ($x' \in N_k(x)$);
6 **Busca Local:**
Aplicar um método de busca local em x', obtendo um ótimo local x'';
7 **Mover ou não:**
Se o ótimo local é melhor que o atual **então**
(b) 8 mover para lá ($x \leftarrow x''$),
9 ir para a primeira vizinhança ($k \leftarrow 1$);
senão
10 passar para a próxima vizinhança ($k \leftarrow k + 1$);
fim Se
fim Enquanto
fim Enquanto

Figura 3.3: Locais de inserção de memória no VNS básico

- **Memória da ultima vizinhança** - cujo objetivo é tentar evitar um possível retorno à solução da vizinhança anterior, isso na construção de uma nova solução x';

- **Memória da agitação feita na vizinhança atual** - procura auxiliar na condução da geração das próximas soluções x', o que ocorre na fase de agitação da nova vizinhança. Na prática, representa a memória da última vizinhança ocorrendo durante uma ou mais vizinhanças;

- **Memória da melhora na vizinhança atual** - o seu uso foi associado apenas às melhoras ocorridas na fase de busca local, cujas informações foram preservadas para a geração das próximas vizinhanças;

- **Memória com controle duplo** - nas duas versões imediatamente anteriores, as variações de controle de memória propostas, isto é, o controle de memória da agitação feita na vizinhança atual e da melhora na vizinhança atual foram aplicados isoladamente, seja por vértices, seja por movimento. Verificou-se o comportamento dessas versões, quando utilizadas em conjunto.

Na Figura 3.4, é apresentada uma tabela com as legendas para as respectivas versões propostas neste trabalho, que serão detalhadas em seguida.

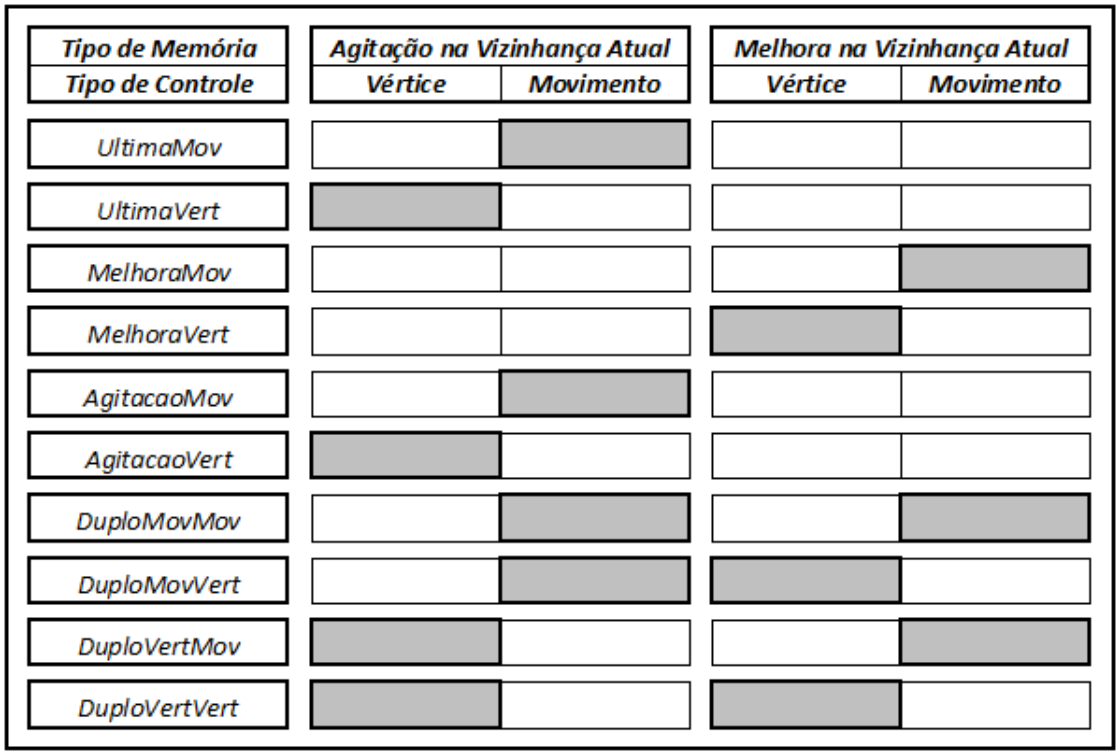

Figura 3.4: Legenda das versões propostas para o MVNS

Em todas as propostas, procurou-se sempre trabalhar com soluções iniciais construídas de maneira totalmente aleatória e com mesmo conjunto de sementes.

Por uma questão visual, nos algoritmos propostos a seguir, optou-se por mostrar apenas as linhas que sofreram alterações.

3.1 Memória da última vizinhança no VNS básico

Apresentam-se, a seguir, duas versões do uso de memória **apenas** da última vizinhança, sendo que a segunda versão procura fazer um controle mais amplo que a primeira. A abrangência dessas versões se dá como memória da agitação feita na vizinhança atual.

3.1.1 Memória da última vizinhança com controle por vértice *(UltimaVert)*

A primeira versão da memória da última vizinhança tem o seu controle por vértice, sendo chamada neste trabalho de *memória da última vizinhança com controle por vértice* ou apenas *UltimaVert*, cujo algoritmo é apresentado na Figura 3.5, a seguir:

Algoritmo 2 : *VNS Básico + UltimaVert*

```
•
•
•
5   Gerar aleatoriamente uma solução x' na k-ésima
    vizinhança de x (x' ∈ N_k(x)) usando vértices
    diferentes dos utilizados na vizinhança k − 1,
6   Guardar os vértices utilizados na vizinhança
    atual;
7   Aplicar um procedimento de busca local em x' para
    obter x'';
8   Se f(x'') < f(x) então
9     Inicializar o controle por vértices;
•
•
•
```

Figura 3.5: VNS básico + versão de memória *UltimaVert*

Seu controle é simples e feito através de duas variáveis de memória que guardam quais os vértices escolhidos aleatoriamente na agitação da vizinhança anterior.

A inicialização das variáveis de controle se faz atribuindo a elas o valor $n + 1$, o que ocorre na *(linha 1)*, ou sempre que houver uma melhora na vizinhança atual *(linha 9)*.

Na geração da solução x', isto é, agitação da vizinhança atual, verifica-se se uma das duas posições usadas na vizinhança anterior foi escolhida para a vizinhança atual. Caso isto tenha ocorrido, nova geração aleatória é feita, evitando que **ambos** os vértices da vizinhança anterior sejam utilizados na geração da vizinhança atual.

3.1.2 Memória da última vizinhança com controle por movimento *(UltimaMov)*

Como o próprio nome já sugere, a versão de *memória da última vizinhança com controle por movimento* ou *UltimaMov* procura fazer um controle por movimento. Para tal, bastou uma simples alteração na *linha 5* do algoritmo da Figura 3.5, gerando uma versão que somente impede o caso onde **ambas** as posições são iguais às utilizadas na vizinhança anterior, o que pode ser verificado no algoritmo da Figura 3.6, a seguir.

Algoritmo 3 : *VNS Básico + UltimaMov*

```
.
.
.
5   Gerar aleatoriamente uma solução x' na k-ésima
    vizinhança de x (x' ∈ N_k(x)) usando um movi-
    mento diferente do utilizado na vizinhança
    k − 1;
6   Guardar o movimento utilizado na vizinhança
    atual;
7   Aplicar um procedimento de busca local em x' para
    obter x'';
8   Se f(x'') < f(x) então
9     Inicializar o controle por movimento;
.
.
.
```

Figura 3.6: VNS básico + versão de memória *UltimaMov*

3.2 Memória da agitação feita na vizinhança atual no VNS básico

A inserção da memória na última vizinhança pode até trazer melhora na eficiência do VNS básico, mas sua abrangência envolve apenas duas vizinhanças consecutivas, isto é, a atual e a imediatamente anterior. Procurando ampliar esse controle, foi proposta a memória da agitação feita na solução atual, que é mais ampla porque sua atuação se estende por algumas trocas de vizinhanças através do uso de uma lista que guarda a caracterização dos últimos vértices ou movimentos feitos em trocas anteriores. Assim se introduz um conceito análogo ao que pode ser visto em algumas metaheurísticas que também não fazem uso de múltiplas partidas, tais como Algoritmos Genéticos, Busca Tabu e *Simulated Annealing*.

No caso do PQA, a idéia é evitar que as posições ou movimentos escolhidos aleatoriamente na troca de vizinhança atual não sejam utilizados durante algumas próximas vizinhanças sendo que, se nesse ínterim houver uma melhora da vizinhança atual, todo o controle atual será inicializado, por se tratar de uma região de busca diferente da atual, e por procurar preservar também a abrangência do VNS básico.

A seguir, são apresentadas duas opções de inserção **apenas** de memória da agitação feita na vizinhança atual, sendo que na primeira o controle é feito pelo vértice e a na segunda é feito pelo movimento.

3.2.1 Memória da agitação feita na vizinhança atual com controle por vértice *(Agitacao Vert)*

A inserção de memória da agitação na vizinhança atual com controle por vértice ou simplesmente *Agitacao Vert*, apresenta poucas alterações no VNS básico e é computacionalmente *"barata"*, com complexidade $O(n)$. Para isso, foi necessária a criação de duas listas, representadas por dois vetores de tamanho n e de uma função para a definição dos vértices candidatos à escolha aleatória da vizinhança atual. Esta versão é representada pelo algoritmo da Figura 3.7, a seguir.

Algoritmo 4 : *VNS Básico +* ***Agitacao Vert***

```
•
•
•
5   Gerar aleatoriamente uma solução x' na k-ésima
    vizinhança de x (x' ∈ N_k(x)) usando os vértices
    candidatos atuais;
6   Guardar os vértices utilizados na vizinhança
    atual;
7   Aplicar um procedimento de busca local em x' para
    obter x'';
8   Se f(x'') < f(x) então
9     Inicializar o controle por vértices;
•
•
•
```

Figura 3.7: VNS básico + versão de memória *Agitacao Vert*

No algoritmo da Figura 3.7, faz-se a inclusão de dois vetores. O primeiro vetor, aqui chamado de *Usado*, será:

- Inicializado com zeros;
- As escolhas (feitas aleatoriamente) das posições que irão compor a vizinhança atual recebem o valor um (1), criando um controle binário da utilização das posições usadas dos vértices;
- Como visto anteriormente, toda vez que acontecer uma melhora na vizinhança atual, o vetor será inicializado já que, nesse caso, uma nova solução suscita um novo controle, visto que a solução x é atualizada, recebendo todo o conteúdo da solução x''.

O segundo vetor, aqui chamado de *Candidato*, ficará responsável por guardar as posições que serão candidatas na escolha aleatória da vizinhança atual. Seu preenchimento e controle será feito a partir de uma função especifica, cujo algoritmo está descrito na Figura 3.8, a seguir.

Algoritmo 5 : *Função* ***CandidatosParaTroca()***

```
1  Inicialização:
   j ← 0;
2  Para (i ← 1 até n) faça
3    Se (Usado[i]) então
4      j ← j + 1;
5      Candidato[j] ← i;
     fim Se
   fim Para
6  Se (j < 3) então
7    Inicializar o controle de memória;
   fim Se
```

Figura 3.8: Função responsável pelos candidatos para troca

A função, que aqui é chamada de *CandidatosParaTroca()*, representada pelo algoritmo da Figura 3.8, é responsável por:

- Percorrer todo o primeiro vetor *(Usado)*, gravando as posições que forem iguais a zero no segundo vetor *(Candidato)*, isto é, que ainda não foram utilizadas no controle atual, laço que se inicia na *linha 2*, indo até a *linha 5*;

- Como no VNS básico a agitação *(shaking)* na vizinhança atual sempre corresponde à troca aleatória de duas posições, procurou-se preservar isso, inicializando o vetor *Usado* e preenchendo totalmente o vetor *Candidato* toda vez que o número de candidatos for menor que três, já que para dois candidatos não há aleatoriedade e para um candidato não há escolha. Isto, na prática, faz com que todos os vértices voltem a ser candidatos. Esta verificação é feita na *linha 6* e sua inicialização na *linha 7*;

3.2.2 Memória da agitação feita na vizinhança atual com controle por movimento *(AgitacaoMov)*

A inserção do controle de memória da versão *AgitacaoMov* traz consigo a restrição de que um vértice com status de "usado" ($Usado[i] \leftarrow 1$) numa vizinhança anterior, não possa ser associado a outro vértice na vizinhança atual, o que é semelhante ao ocorrido na versão *UltimaVert*, ambas apresentadas anteriormente. Dependendo do problema ou instância, tal restrição pode ser prejudicial na fase de busca local, pois pode conduzir a solução atual (x'') para uma região distante de um ótimo local.

Para que a memória inserida no VNS básico possa ser capaz de fazer um controle mais apurado, propôs-se a versão de *memória da agitação feita na vizinhança atual com controle por movimento*, ou apenas *AgitacaoMov*. Para isso, se fez necessário

um redimensionamento no tamanho das listas atuais e a criação de uma nova lista, o que pode ser observado no algoritmo da Figura 3.9.

Algoritmo 6 : *VNS Básico +* ***AgitacaoMov***

```
•
•
•
5   Gerar aleatoriamente uma solução x' na k-ésima
    vizinhança de x (x' ∈ N_k(x)) usando os movimen-
    tos candidatos atuais;
6   Guardar o movimento utilizado na vizinhança
    atual;
7   Aplicar um procedimento de busca local em x' para
    obter x'';
8   Se f(x'') < f(x) então
9     Inicializar o controle por movimento;
•
•
•
```

Figura 3.9: VNS básico + versão de memória *MelhoraMov*

Considerando as possibilidades de troca entre duas posições num grafo não direcionado, as listas *(Usado* e *Candidato)* passam a ser vetores de tamanho $N = n(n-1)/2$ (Seção 1.2). Dependendo do problema, deve-se considerar também que o uso dessa memória por movimento pode não ser viável computacionalmente, já que está associada ao número de arestas do grafo, o que equivale dizer que uma busca no vetor possui a complexidade de $O(N)$. Considerando que as instâncias do PQA que serão utilizadas neste trabalho têm, no máximo, na ordem de $n = 256$, algumas simétricas (grafos não direcionados), outras não, esse algoritmo utilizará listas de, no máximo, tamanho igual a 32640.

Esta versão se diferencia da anterior não apenas pelo tamanho dos vetores *(Usado* e *Candidato)*, mas também pela criação de um terceiro vetor de posições, aqui chamado de *Posicao*, de tamanho igual a N. Tal vetor *Posicao* é um recurso de estrutura de dados que permitirá a identificação, em $O(1)$, das coordenadas que aqui representam uma posição ou movimento, de ordem n, a partir de sua respectiva posicão no vetores de ordem N.

O seguinte exemplo (onde $n = 6$ e $N = 15$), da Figura 3.10 ilustra isso, pois, considerando que todas as posições estão disponíveis e representam os possíveis movimentos, qualquer uma das 15 posições será candidata. Se a escolha aleatória for o número 9, os vértices associados que terão seus conteúdos trocados na solução x' atual serão os vértices 2 e 6, respectivamente. Esse movimento será impedido de ocorrer novamente por $N - 3$ agitações, no máximo, já que existe uma preocupação de que a escolha seja sempre aleatória, como já explicado na função *CandidatosParaTroca()*.

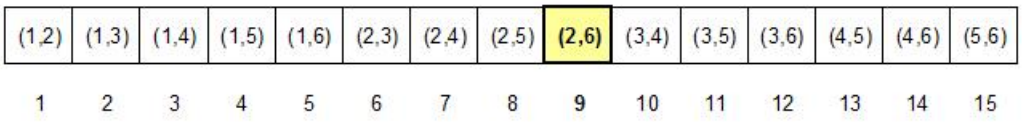

Figura 3.10: Exemplo do conteúdo do vetor de controle de posições *(Posicao)*

Porém, se a partir de duas posições de uma matriz, de ordem n, se deseja achar sua posição relativa num vetor de ordem N, isto é conseguido diretamente através da função *PosMatToVet()*. Cujo algoritmo é mostrado na Figura 3.11, a seguir.

Algoritmo 7 : *Função* ***PosicaoDaMatrizParaVetor()***

```
1  Se (i < j) então
2     pos ← (n * (i − 1) − (i * (i + 1)/2) + j;
   senão
3     pos ← (n * (j − 1) − (j * (ji + 1)/2) + i;
   fim Se
4  retorno(pos);
```

Figura 3.11: Função que retorna a posição relativa no vertor *Posicao*

O preenchimento do vetor *Posicao* com as posições relativas, conforme visto no algoritmo da Figura 3.9, deve ser feito na inicialização do algoritmo e não foi contemplado no mesmo. Dentro de todas as versões de controle de memória por movimento, o seu conteúdo foi usado apenas para consulta.

A diferença mais significativa dessa versão em relação à versão de troca de vizinhanças com controle da memória pelo vértice foi que:

- O vetor *Usado* é criado e inicializado, sempre, para o tamanho N;
- A partir da escolha aleatória de uma posição válida no vetor *Candidato*, podem-se acessar diretamente as posições i e j no vetor *Posicao* e, a partir daí, realizar a respectiva troca;
- O controle binário passou a ser sobre as posições usadas (*movimento*).

Vale a observação de que a função *CandidatoParaTroca()*, Figura 3.8 permanece a mesma, mudando apenas um dos parâmetros de entrada, que passa de tamanho n para N.

3.3 Uso de memória da melhora na vizinhança atual do VNS básico

O esquema de cruzamento (*crossover*) é um dos principais responsáveis pelo sucesso dos Algoritmos Genéticos, já que ele deve ser capaz de produzir uma nova solução viável através da combinação de boas características dos "*pais*". Ahuja *et al* (106) usaram os Algoritmos Genéticos para resolver o PQA onde, num dos esquemas de cruzamento, procuraram fazer com que as características comuns aos pais fossem herdadas pelos filhos, e para isso as trocas ocorrem somente nas posições que não são comuns. A Figura 3.12 ilustra um esquema de cruzamento que preserva as posições comuns.

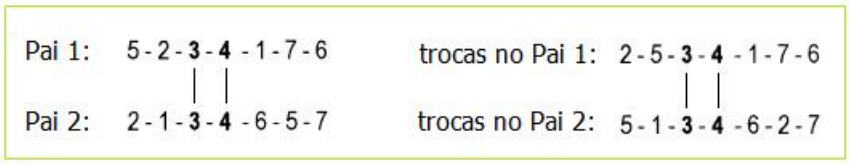

Figura 3.12: Exemplo de cruzamento preservando as posições comuns

Guardadas as devidas analogias, a versão de memória da melhora na vizinhança atual, aqui proposta, procura fazer com que a informação da troca que houve entre dois vértices ou movimento e que gerou uma nova solução x, seja "lembrada"durante algum tempo, o que corresponde ao seu não uso na geração das próximas vizinhanças. Isso é feito, deixando que ela se propague até que a memória atual seja totalmente utilizada, ou se encontre uma solução melhor x, ou que sejam percorridas todas as vizinhanças (K_{max}) do VNS, o que ocorrer primeiro.

3.3.1 Memória da melhora na vizinhança atual com controle por vértice (*MelhoraVert*)

Esta versão de memória, aqui chamada de memória da melhora da solução com controle por vértice ou apenas *MelhoraVert*, não faz uso da memória na mudança de vizinhança, e sim quando há uma melhora na solução x, após a busca local, apesar de usar a mesma estrutura de listas para seu controle, como pode ser visto no algoritmo da Figura 3.13, a seguir.

Sua parte específica se restringe a apenas *linha 9*, que é responsável pelo reforço da *"memória"* nas estruturas de vizinhança seguintes. Se houver uma melhora na solução x atual, procura-se evitar sua utilização na geração aleatória da próxima vizinhança, ver *linha 5*, o que acorre através da função *CandidatoParaTroca()*.

Algoritmo 8 : *VNS Básico + MelhoraVert*

```
   .
   .
   .
5      Gerar aleatoriamente uma solução x' na k-ésima
       vizinhança de x (x' ∈ N_k(x)) usando os vértices
       candidatos atuais;
6      Aplicar um procedimento de busca local em x' para
       obter x'';
7      Se f(x'') < f(x) então
8        Inicializar o controle por vértices;
9        Guardar os vértices usados na geração da
         nova solução;
   .
   .
   .
```

Figura 3.13: VNS básico + versão de memória *MelhoraVert*

3.3.2 Memória da melhora na vizinhança atual com controle por movimento (*MelhoraMov*)

Usando a mesma justificativa apresentada para a versão *MelhoraVert*, a versão de memória após melhora da solução com controle por movimento ou simplesmente *MelhoraMov*, faz uso das mesmas listas e estruturas de dados vistas nessa versão *MelhoraVert*. Esse algoritmo, porém, se apóia apenas no controle através do movimento feito, impedindo apenas que as respectivas posições sejam reutilizadas nas próximas vizinhanças, até que caiam no "*esquecimento*"o que, para os algoritmos aqui propostos, ocorre quando não houver mais a possibilidade de uma escolha aleatória *(linha 5)*, ou quando houver uma melhora na solução x' *(linha 8)*, como pode ser verificado no algoritmo da Figura 3.14.

3.4 Uso de memórias com controle duplo

Em todos os algoritmos anteriores os controles de memória, seja pelo vértice, seja pelo movimento, aconteceram isoladamente. Um caminho natural era fazer um estudo do seu impacto quando utilizadas em conjunto, estendendo o conceito neste trabalho para memória com controle duplo.

Para agrupar esses controles num mesmo algoritmo, usou-se a estrutura do controle por movimento, de tamanho N, para que esse possa conter também o controle por vértice, de tamanho n, já que $N > n$. Para isso, foi necessária a criação da função *FixaPosicaoVetor()*, cujo pseudocódigo encontra-se no algoritmo da Figura 3.15 onde, dado um vértice qualquer, possam ser alocados todos os movimentos possíveis para esse, simulando assim o controle por vértice, mesmo que a estrutura

Algoritmo 9 : *VNS Básico + **MelhoraMov***

```
•
•
•
5   Gerar aleatoriamente uma solução x' na k-ésima
    vizinhança de x (x' ∈ N_k(x)) usando os movimen-
    tos candidatos atuais;
6   Aplicar um procedimento de busca local em x' para
    obter x'';
7   Se f(x'') < f(x) então
8     Inicializar o controle por movimento;
9     Guardar o movimento usado na geração da
      nova solução;
•
•
•
```

Figura 3.14: VNS básico + versão de memória *MelhoraMov*

de dados represente sempre o movimento.

Algoritmo 10 : *Função **FixaPosicaoVetor()***

```
1  Para (i ← 1 até n) faça
2    pos ← PosicaoDaMatrizParaVetor();
3    Usado[pos] ← 1;
   fim Para
```

Figura 3.15: Função *FixaPosicaoVetor*

Como pode ser visto, esta função possui uma complexidade $O(n)$, percorrendo as posições que representam os possíveis movimentos de um vértice. Após achar sua posição relativa no vetor *Usado*, *linha 3* faz-se o registro de *"usado"*, isto na linha seguinte. O exemplo da Figura 3.16, simula o controle por vértices num vetor de movimentos, demonstrando todas as posições marcadas no vetor *Usado*, considerando como os vértices escolhidos: 2 e 6.

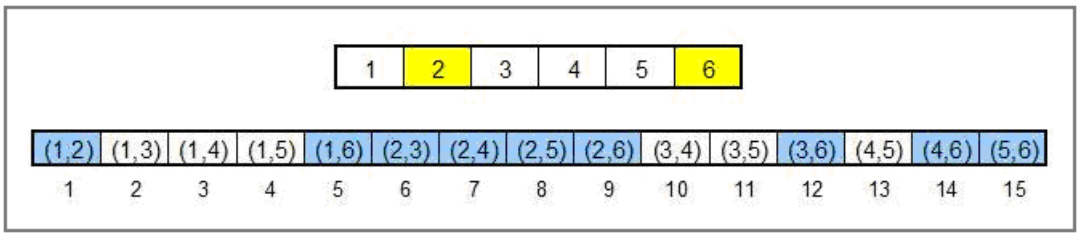

Figura 3.16: Simulação de controle por vértices num vetor de movimentos

Seguindo a mesma trajetória das versões anteriores, esta versão também se apoiou na simplicidade e reaproveitamento de código.

O controle dos candidatos para a escolha aleatória na vizinhança atual continua sendo feito pela função *CandidatoParaTroca()*, vista anteriormente no Algoritmo 3.8, que não sofreu alteração alguma. A marcação das posições já usadas no vetor *Usado* é que passa a ser feita nas duas situações já descritas no início desse capítulo: (a) sempre que for feita uma agitação na vizinhança atual; (b) sempre que houver uma melhora na vizinhança atual e que, como dito anteriormente, para contemplar os dois controles passa a guardar todos os movimentos, já que esse controle contém os vértices. Sendo assim, se o controle for pelo movimento, independente de onde for feito, deve-se inserir a seguinte linha, onde a variável r deve corresponder à sua posição no vetor Usado:

$Usado[r] = 1$;

Se o interesse for o controle pelo vértice, a função *FixaPosicaoVetor()*, que simula o controle por vértice no vetor Usado, deverá ser utilizada para cada um dos dois vértices que compõem o movimento, como no exemplo:

$FixaPosicaoVetor(Usado, vertice, n)$;

onde o parâmetro *vertice* deve representar cada vértice que compõe o movimento.

3.4.1 Memória com controle por vértice na agitação e na melhora da vizinhança atual (*DuploVertVert*)

A versão *DuploVertVert* como o próprio nome já a define, usa o controle por vértice em ambas as situações, isto é, na agitação feita na vizinhança atual e quando há uma melhora na vizinhança.

Como ambos os controles são feitos pelos vértices, a função *FixaPosicaoVetor()* é utilizada quatro vezes, marcando no vetor *Usado* todas as respectivas posições dos vértices que ela recebe como parâmetro de entrada e que, por simplificação, são referenciadas nas linhas 6 e 10 do algoritmo da Figura 3.17, a seguir.

3.4.2 Memória com controle por vértice na agitação e por movimento na melhora da vizinhança atual (*DuploVertMov*)

A versão *DuploVertMov* usa o controle de memória por vértice na agitação da vizinhança atual e o controle de memória por movimento quando há melhora na vizinhança atual.

A sua construção partiu do algoritmo 3.4.1 utilizado na versão anterior, onde foi necessária apenas a alteração da *linha 10*, para que pudesse passar a fazer o controle por movimento, o que pode ser conferido no algoritmo da Figura 3.18, a seguir.

Algoritmo 11 : *VNS Básico +* ***DuploVertVert***

```
.
.
.
5   Gerar aleatoriamente uma solução x' na k-ésima
    vizinhança de x (x' ∈ N_k(x)) usando os candidatos atuais;
6   Guardar os vértices utilizados na vizinhança atual;
7   Aplicar um procedimento de busca local em x' para obter x'';
8   Se f(x'') < f(x) então
9     Inicializar o controle de memória;
10    Guardar os vértices usados na geração da nova solução;
.
.
.
```

Figura 3.17: VNS básico + versão de memória *DuploVertVert*

Algoritmo 12 : *VNS Básico +* ***DuploVertMov***

```
.
.
.
5   Gerar aleatoriamente uma solução x' na k-ésima
    vizinhança de x (x' ∈ N_k(x)) usando os candidatos atuais;
6   Guardar os vértices utilizados na vizinhança atual;
7   Aplicar um procedimento de busca local em x' para obter x'';
8   Se f(x'') < f(x) então
9     Inicializar o controle de memória;
10    Guardar o movimento usado na geração da nova solução;
.
.
.
```

Figura 3.18: VNS básico + versão de memória *DuploVertMov*

3.4.3 Memória com controle por movimento na agitação e por vértice na melhora da vizinhança atual (*DuploMovVert*)

A versão *DuploMovVert* faz um controle contrário a versão *DuploVertMov* e como tal, fazendo uso de memória com controle por movimento na agitação da vizinhança atual e controle de memória por vértice quando há melhora na vizinhança.

Para sua construção, usou-se também como base a versão 3.4.1, onde foi ne-

cessária apenas a troca da *linha 6*. Isto pode ser visto no algoritmo da figura 3.4.3, a seguir.

Algoritmo 13 : *VNS Básico +* ***DuploMovVert***

```
•
•
•
5    Gerar aleatoriamente uma solução x' na k-ésima
     vizinhança de x (x' ∈ N_k(x)) usando os can-
     didatos atuais;
6    Guardar o movimento utilizado na vizinhança
     atual;
7    Aplicar um procedimento de busca local em x' para
     obter x'';
8    Se f(x'') < f(x) então
9      Inicializar o controle de memória;
10     Guardar os vértices usados na geração da
       nova solução;
•
•
•
```

Figura 3.19: VNS básico + versão de memória *DuploMovVert*

3.4.4 Memória com controle por movimento na agitação e na melhora da vizinhança atual (*DuploMovMov*)

A versão *DuploMovMov* faz o controle de memória pelo movimento para ambos os casos. Sua construção foi feita a partir da alteração da *linha 6* do algoritmo da versão 3.4.2, mas poderia ter utilizado o algoritmo da versão 3.4.3 onde, nesse caso, a alteração seria na *linha 10*. O resultado pode ser visto no algoritmo encontra-se da Figura 3.4.4.

Apenas para exemplificar que as versões propostas neste trabalho exigem pouco esforço para que sejam produzidas, outra possibilidade seria construir inicialmente as versões de memória com controle duplo e a partir delas, gerar as versões especificas de controle de memória na agitação e melhora da vizinhança atual, bastando apenas a remoção do conteúdo das linhas que fazem o controle não necessário para a versão que se está construindo.

Algoritmo 14 : *VNS Básico +* ***DuploMovMov***

```
•
•
•
5    Gerar aleatoriamente uma solução x' na k-ésima
     vizinhança de x (x' ∈ N_k(x)) usando os can-
     didatos atuais;
6    Guardar o movimento utilizado na vizinhança
     atual;
7    Aplicar um procedimento de busca local em x' para
     obter x'';
8    Se f(x'') < f(x) então
9       Inicializar o controle de memória;
10      Guardar o movimento utilizado na geração
        da nova solução;
•
•
•
```

Figura 3.20: VNS básico + versão de memória *DuploMovMov*

Capítulo 4

Testes Computacionais do uso de memória no VNS básico

Segundo Hansen e Mladenovìc (84), a escolha da solução inicial é importante para cada método de heurística de busca local. Porém, a qualidade da solução obtida através do VNS, em média, não depende significativamente da solução inicial. O número de iterações necessárias para se alcançar a solução ótima é muito sensível ao valor da semente e, portanto, para cada problema em particular, mas novamente não existe diferença significativa na média dos resultados. Como relatado anteriormente, K_{max} é o único parâmetro usado na versão básica do VNS. Nos nossos testes preliminares ($K_{max} = 2, 3, 5, 7$ e 10), o aumento do valor de K_{max} auxiliou nos resultados médios obtidos, porém se observou que tal crescimento apresenta uma saturação que, para os casos observados ocorreu, no máximo, com $K_{max} = 5$. A contrapartida é que quanto maior o valor de K_{max}, maior é o esforço computacional que o algoritmo necessita para realizar o mesmo numero de iterações já que, no pior caso, será preciso trocar K_{max} vizinhanças a cada iteração do algoritmo. Não foi feito um estudo sobre o impacto do valor de K_{max}, mas testes preliminares conduziram ao valor de $K_{max} = 3$, valor esse que foi usado para comparar com os resultados obtidos em outros trabalhos relacionados.

4.1 Estruturas de vizinhança utilizadas

O uso das estruturas de vizinhança no VNS básico procura auxiliar na busca por soluções melhores, cuja idéia consiste em explorar, sucessivamente, estruturas de vizinhança dispostas em ordem crescente e, como já visto no final do Capítulo 2, a escolha de tais estruturas traz dois grandes desafios: o tamanho das vizinhanças e a seleção de movimentos que sejam interessantes para o problema em estudo. Considerando isso e os nossos testes preliminares, optamos por fazer uso de **três**

estruturas de vizinhança, mostradas a seguir, onde é possível perceber claramente que o custo computacional aumenta a cada nova vizinhança, isto é, aumenta a medida que o valor de k aumenta.

Sempre que for necessário a geração de um valor entre um limite inferior e um limite superior será chamada a função *Aleatorio()*, que devolve um valor inteiro igual a um dos limites fornecidos ou compreendido entre eles.

Todas as vizinhanças fazem uso da função *CalculaValorDelta()*, que reproduz o que foi proposto por Taillard e também já visto no final do Capítulo 2, o que reduz sensivelmente o custo computacional do cálculo da possível troca. Tais trocas somente se efetivam se trazem alguma melhora na vizinhança atual.

4.1.1 Vizinhança 1

A partir da solução viável fornecida para esta vizinhança, procura-se, através de trocas totalmente aleatórias entre vértices diferentes, uma melhora na solução corrente. Na Figura 4.1, apresenta-se o algoritmo para a busca local na Vizinhança 1.

A escolha dos vértices utiliza a idéia sugerida por Taillard (90) em seu código do Tabu Robusto, que dispensa a verificação de que os vértices escolhidos aleatoriamente sejam iguais ou não: a escolha do primeiro vértice (i) será obtida através da função *Aleatorio()* que, nesse caso retornará um valor entre 1 e $n-1$, ver *linha 3*. Com isso, o segundo vértice (j) será um valor entre $i+1$ e n, *linha 4*.

Caso o custo da troca resulte em alguma melhora na solução atual, efetiva-se a troca, caso contrário, faz-se uma nova escolha aleatória até que o critério de parada seja alcançado, isto é, até que a variável de controle (x) seja igual à ordem da instância (n) ou que o valor ótimo ou o melhor valor conhecido até o momento (VOMVC) tenha sido alcançado.

4.1.2 Vizinhança 2

Também a partir de uma solução viável fornecida para esta vizinhança, procura-se, através de simples trocas sucessivas, uma melhora na solução corrente. Na Figura 4.2 apresenta-se o algoritmo para a busca local na Vizinhança 2.

Esta vizinhança difere da vizinhança anterior, primeiro porque a escolha aleatória do primeiro vértice (i) possui como candidatos todos os vértices da solução atual, ver *linha 3*, segundo porque para cada vértice escolhido (i) verifica-se a possibilidade de troca com todos os demais vértices, o que ocorre no laço compreendido entre as *linhas 5* e *10*.

Caso o custo de alguma troca resulte em alguma melhora na solução atual, efetiva-se a troca, caso contrário, faz-se uma nova escolha aleatória para o primeiro

```
Algoritmo 15 : Vizinhança 1
1   x ← 1;
2   Enquanto (x ≤ n) e (valorOtimo < valorAtual) faça
3     i ← Aleatorio(1,n-1);
4     j ← Aleatorio(i+1,n);
5     valorTroca ← CalculaValorDelta(i,j,n);
6     Se (valorAtual + valorTroca < valorAtual) então
7       TrocaPosicao(Solucao,i,j,n);
8       valorAtual ← valorAtual + valorTroca;
      fim Se
9     x ← x + 1;
    fim Enquanto
```

Figura 4.1: Algoritmo da estrutura de Vizinhança 1

vértice, até que o critério de parada seja alcançado, ou seja, até que a variável de controle (x) seja igual a ($n/2$) ou que o VOMVC tenha sido alcançado.

```
Algoritmo 16 : Vizinhança 2
1   x ← 1;
2   Enquanto (x ≤ n/2) e (valorOtimo < valorAtual) faça
3     i ← Aleatorio(1,n);
4     j ← 1;
5     Enquanto (j ≤ n) faça
6       valorTroca ← CalculaValorDelta(i,j,n);
7       Se (valorAtual + valorTroca < valorAtual) então
8         TrocaPosicao(Solucao,i,j,n);
9         valorAtual ← valorAtual + valorTroca;
        fim Se
10      j ← j + 1;
      fim Enquanto
11    x ← x + 1;
    fim Enquanto
```

Figura 4.2: Algoritmo da estrutura de Vizinhança 2

4.1.3 Vizinhança 3

Através de trocas sucessivas na solução viável fornecida para esta estrutura, procura-se uma melhora, de forma exaustiva. Para cada vértice da solução atual, verifica-se o custo da sua troca com cada um dos demais vértices. Por isso, trata-se de uma estrutura computacionalmente dispendiosa. Além de conter as vizinhanças anteriores, sua escolha como terceira vizinhança se justifica por sua utilização, pelo

algoritmo, ser menor que as vizinhanças anteriores.

Importante ressaltar que, sempre que houver uma melhora, as variáveis de controle, que também indicam os vértices candidatos a troca serão inicializada com valor 1, como pode ser visto nas *linhas 9* e *10*. Mesmo isso trazendo uma maior custo computacional ao algoritmo, tal inicialização se justifica porque uma melhora na solução não ocorre com tanta frequencia, e quando ocorre, testes preliminares nos indicam que tal inicialização produz resultados médios melhores, já que uma melhora suscita uma nova verificação em todos os vértices da solução atual.

Da mesma forma que nas vizinhanças anteriores, a solução atual somente será atualizada se for encontrada uma melhora. Na Figura 4.3 apresenta-se o algoritmo para a busca na Vizinhança 3.

Algoritmo 17 : *Vizinhança 3*

```
1  i ← 1;
2  Enquanto (i ≤ n) e (valorOtimo < valorAtual) faça
3     j ← 1;
4     Enquanto (j ≤ n) faça
5        valorTroca ← CalculaValorDelta(i,j,n);
6        Se (valorAtual + valorTroca < valorAtual) então
7           TrocaPosicao(Solucao,i,j,n);
8           valorAtual ← valorAtual + valorTroca;
9           i ← 1;
10          j ← 1;
         fim Se
11       j ← j + 1;
      fim Enquanto
12    i ← i + 1;
   fim Enquanto
```

Figura 4.3: Algoritmo da estrutura de Vizinhança 3

4.2 Instâncias do PQA usadas nos testes computacionais

As instâncias do PQA utilizadas neste trabalho foram divididas nas cinco classes apresentadas na seção 1.4. A Figura 4.4 mostra o conteúdo de cada uma das classes definidas.

A partir da Figura 4.4, é possível verificar o total de elementos de cada classe. A *classe 1* ficou com 47 instâncias, a *classe 2* com 30 instâncias, a *classe 3* com 38 instâncias, a *classe 4* com 16 e, por último, a *classe 5*, com 37 instâncias.

Instâncias - Classe 1					
chr12a	chr20a	tai12a	tai256c	tai40b	rou15
chr12b	chr20b	tai12b	tai25a	tai50a	rou20
chr12c	chr20c	tai150b	tai25b	tai50b	tho150
chr15a	chr22a	tai15a	tai30a	tai60a	tho30
chr15b	chr22b	tai15b	tai30b	tai60b	tho40
chr15c	chr25a	tai17a	tai35a	tai80a	wil100
chr18a	tai100a	tai20a	tai35b	tai80b	wil50
chr18b	tai100b	tai20b	tai40a	rou12	

Instâncias - Classe 2			
nug12	nug22	scr20	sko49
nug14	nug24	sko100a	sko56
nug15	nug25	sko100b	sko64
nug16a	nug27	sko100c	sko72
nug17	nug28	sko100d	sko81
nug18	nug30	sko100e	sko90
nug20	scr12	sko100f	
nug21	scr15	sko42	

Instâncias - Classe 3			
bur26a	esc16a	esc32b	had18
bur26b	esc16b	esc32c	had20
bur26c	esc16c	esc32d	kra30a
bur26d	esc16d	esc32e	kra30b
bur26e	esc16e	esc32g	kra32
bur26f	esc16g	esc32h	ste36a
bur26g	esc16h	esc64a	ste36b
bur26h	esc16i	had12	ste36c
els19	esc16j	had14	
esc128	esc32a	had16	

Instâncias - Classe 4	
lipa20a	lipa70a
lipa20b	lipa70b
lipa30a	lipa80a
lipa30b	lipa80b
lipa40a	lipa90a
lipa40b	lipa90b
lipa50a	
lipa50b	
lipa60a	
lipa60b	

Instâncias - Classe 5			
dre110	dre72	pal70	tai45e04
dre132	dre90	pal80	tai45e05
dre15	pal100	tai27e01	tai75e01
dre18	pal150	tai27e02	tai75e02
dre21	pal20	tai27e03	tai75e03
dre24	pal200	tai27e04	tai75e04
dre28	pal30	tai27e05	tai75e05
dre30	pal40	tai45e01	
dre42	pal50	tai45e02	
dre56	pal60	tai45e03	

Figura 4.4: Classes das instâncias

Todas as versões propostas partiram de soluções iniciais geradas aleatoriamente, utilizando o mesmo método de busca local descrito em Taillard (90) e um conjunto de dez execuções, onde cada uma delas foi inicializada com uma nova semente, de modo a garantir sua independência. As sementes foram sorteadas da lista de números primos de 1 a 2.000.000 propostos por Estany (107), onde as sementes utilizadas foram: 1635559, 1970641, 1014029, 1824607, 1354051, 993893, 1253171, 172190, 397127 e 343547.

As implementações foram feitas em Linguagem C e todos os testes executados num computador com processador Intel Core 2 Quad 2.4GHz com 4Gb de memória RAM, sob o Sistema Operacional Linux, distribuição openSUSE.

4.3 Critérios de desempenho das versões propostas

Para nossa análise, usamos um total de cinco critérios de comparação entre os conjuntos de algoritmos propostos:

a. Complemento do número de soluções ótimas obtidas;

b. Afastamento médio;

c. Indicador de qualidade;

d. Tempo médio de execução;

e. Complemento do tempo médio de estagnação.

O primeiro critério de comparação mostra o complemento do número de soluções ótimas obtidas nas 10 execuções independentes. Sendo assim definido o critério, os menores valores representam um melhor resultado.

O afastamento médio se consegue através da média das diferenças $(valorObtido - VOMVC)/VOMVC$, obtidos apenas nas execuções que não alcançaram o $VOMVC$, expresso em porcentagem.

O indicador de qualidade deve ser coerente com desempenho dos algoritmos, para isso, utilizou-se a função 4.1, onde $nSolOtm$ indica o número de soluções que alcançaram o $VOMVC$, $nSolNaoOtm$ indica o número de soluções que não alcançaram o $VOMVC$, e $erroMed$ indica o erro percentual das instâncias que não o alcancaram:

$$I(nsol, erro) = nSolNaoOtm * (erroMed\%) + nSolOtm^{-1} \tag{4.1}$$

Considerando que, em 10 execuções realizadas para uma instância, 8 soluções alcançaram o $VOMVC$ $(nSolOtm)$, então teremos: $I = 1/8 = 0,125$ e, consequentemente, 2 soluções apresentaram erro $(nSolNaoOtm)$ que, nesse exemplo, seria de $1,3\%$. Com isso, teríamos: $I = (2 * 1,3) + 0,125 = 2,725$. Agora, se o erro dessas 2 instâncias fosse de $21,6\%$, teríamos então: $I = (2 * 21,6) + 0,125 = 43,325$. Então, é fácil perceber que tal indicador diminui com o número de acertos e aumenta com o número de erros.

O tempo médio de execução considera apenas as instâncias que conseguiram alcançar o $VOMVC$, antes do critério de parada por tempo máximo de execução, aqui de 600 segundos, para cada execução.

O complemento do tempo médio de estagnação mostra a diferença entre o tempo máximo de execução e o tempo ocorrido na última melhora na solução antes de se alcançar o critério de parada. Quando o algoritmo consegue alcançar o $VOMVC$ esse valor é sempre igual a *zero*.

4.4 Métodos de comparação de desempenho das versões propostas

Realizar comparações entre métodos heurísticos distintos não é uma tarefa fácil, já que podem aparecer resultados muito discrepantes entre eles. Neste trabalho, procuramos fazer uso de dois métodos de comparação que são descritos a seguir.

4.4.1 Método *Condorcet*

Uma abordagem para essa situação foi proposta por Barbut (108) e utilizada por Abreu *et al* (22) e Moreira (32). Trata-se de realizar as avaliações por pares, procurando ordenar pares de resultados com a finalidade de obter uma medida para essas diferenças.

Considere uma população de w objetos $o_1, o_2, ..., o_{mm}$ e suas $mm!$ permutações. Cada permutação induz uma classificação $O_k, 1 \leq k \leq w!$, que significa uma relação de ordem. O objeto o_i é dito melhor ou mais fácil que o objeto o_j através da ordem O_k, quando o_i precede o_j em O_k (ou de maneira abreviada $o_i < o_j$). O par (o_i, o_j) é chamado de *par discordante* entre as ordens O_p e O_q se $o_i < o_j$ na ordem O_p e $o_j < o_i$ na ordem O_q. A distância entre O_p e O_q $(dist(O_p, O_q))$ é definida através do número de pares discordantes, entre os $C_{mm,2}$ possíveis pares. Podemos então definir o *erro relativo* de O_p em relação a O_q como sendo:

$$\varepsilon_{pq} = 100 * dist(O_p, O_q)/C_{w,2} \tag{4.2}$$

Tal técnica pode ser utilizada para comparar algoritmos, dois a dois, em relação a uma instância dada. No nosso caso, w será o número de algoritmos verificados, que estaremos comparando através de um total de z critérios de avaliação, cujos valores se refrem a uma instância dada, cada critério gerando uma ordem diferente.

A fim de evitar possíveis erros de digitação na contabilização dos resultados obtidos para a realização do método *Condorcet*, optou-se pelo desenvolvimento do algoritmo da Figura 4.5, cujas funções são explicadas com exemplos a seguir.

Algoritmo 18 : *Condorcet*

```
1  LeParametrosEntrada();
2  Para (cada instância não lida) faça
3     LeDadosInstancia();
4     OrdenaPorValor();
5     ComparaParesAlgoritmos();
6     ComparaParesCriteriosParesAlgoritmos();
7     GravaComparacaoInstancia();
   fim Para
```

Figura 4.5: Algoritmo cuja função é a comparação de outros algoritmos.

Na linha 1, encontra-se a função $LeParamentrosEntrada()$ que é responsável por obter os três parâmetros do algoritmo, que são: $(totInst)$ - total de instâncias; $(totAlgo)$ - total de algoritmos para comparação e $(totCrit)$ - total de critérios a serem comparados. A partir disso, é possível saber quantas iterações são executadas $(totInst)$, o que inicia na linha 2.

A função $LeDadosInstancia()$ é responsável por alocar, para a instância atual, uma matriz,$(totCrit\ X\ totAlgo)$, contendo as informações dos valores obtidos pelos algoritmos,$(totAlgo)$, nas respectivas linhas associadas aos critérios,$(totCrit)$, como pode ser visto no exemplo trivial da instância $Tai25a$ da Tabela 4.1. Notar que os números que precedem os valores, correspondem aos seus respectivos algoritmos.

	1	2	3	4	5
a	1 - 10.0000	2 - 9.0000	3 - 8.0000	4 - 10.0000	5 - 10.0000
b	1 - 0.5800	2 - 0.6800	3 - 0.5700	4 - 0.7100	5 - 0.5500
c	1 - 5.7600	2 - 7.1200	3 - 5.0400	4 - 7.1100	5 - 5.4500
d	1 - 0.0000	2 - 0.0900	3 - 176.3800	4 - 0.0000	5 - 0.0000
e	1 - 557.5400	2 - 564.9100	3 - 472.0000	4 - 481.5700	5 - 483.9400

Tabela 4.1: Instância Tai25a - Matriz inicial com os valores obtidos pelos algoritmos em cada critério avaliado.

A função $OrdenaPorValor()$ utiliza a matriz gerada pela função $LeDadosInstancia()$ e ordena os valores em ordem não decrescente. Após sua execução, a matriz inicial fica como no exemplo da Tabela 4.2.

	1	2	3	4	5
a	3 - 8.0000	2 - 9.0000	1 - 10.0000	4 - 10.0000	5 - 10.0000
b	5 - 0.5500	3 - 0.5700	1 - 0.5800	2 - 0.6800	4 - 0.7100
c	3 - 5.0400	5 - 5.4500	1 - 5.7600	4 - 7.1100	2 - 7.1200
d	1 - 0.0000	4 - 0.0000	5 - 0.0000	2 - 0.0900	3 - 176.3800
e	3 - 472.0000	4 - 481.5700	5 - 483.9400	1 - 557.5400	2 - 564.9100

Tabela 4.2: Instância Tai25a - Matriz com ordenação não descrescente por valor.

Após a execução da função $OrdenaPorValor()$, os valores ficam arranjados de modo que os melhores (menores) valores fiquem à esquerda, em todas as linhas. A partir da definição de uma relação para o conjunto de pares, onde $(+1, >), (-1, <), (0, =)$, podemos construir uma nova matriz de valores numéricos que expressa os resultados das comparações feitas para cada critério utilizado associado aos resultados dos testes dos algoritmos, como pode ser visto na Tabela 4.3, o que é feito através da função $ComparaParesAlgoritmos()$, linha 5 do algoritmo da Figura 4.5.

	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]
a	1	1	0	0	1	-1	-1	-1	-1	0
b	-1	1	-1	1	1	-1	1	-1	1	1
c	-1	1	-1	1	1	1	1	-1	-1	1
d	-1	-1	0	0	-1	1	1	1	1	0
e	-1	1	1	1	1	1	1	-1	-1	-1

Tabela 4.3: Instância Tai25a - Algoritmo de comparação Condorcet.

Para que se entenda melhor os valores obtidos na Tabela 4.3, vejamos os valores de algumas linhas na Tabela 4.2:

$linha\ a : 2(9.0000) < 4(10.0000)$, logo $[2,4] = -1$;
$linha\ b : 5(0.5500) < 1(0.5800)$, logo $[5,1] = -1$,
como usamos sempre $(i < j)$, então $[1,5] = +1$.
$linha\ d : 1(0.0000) = 4(0.0000)$, logo $[1,4] = [4,1] = 0$;

A etapa seguinte, executada pela função *ComparaParesCriteriosParesAlgoritmos*(), é responsável por uma comparação através da soma dos pares de critérios e dos pares de algoritmos, onde o resultado obtido, sendo igual a zero, indica uma discordância. Com isso, dois parâmetros com valores contrários (Ex: +1 e −1), obtidos a partir da função *ComparaParesAlgoritmos*(), terão um resultado igual a zero (0) o que indica, não sem motivo, sentidos opostos na comparação.

Importante observar, que se faz necessário distinguir os parâmetros com valores originais iguais a 0 (*zero*), cujo somatório chegará ao mesmo resultado dos valores discordantes (*zero*). Os casos onde isto ocorre foram indicados com o valor 9.

Por último, a função *ComparaParesCriteriosParesAlgoritmos*() acrescenta à matriz atual, duas linhas e duas colunas: a primeira, que possuirá o somatório dos valores iguais a *zero*, indica o número de discordância e a segunda, terá o percentual de erro relativo das sequências, visto em 4.2, e que pode ser conferido na Tabela 4.4.

	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	0	2	-1	1	2	-2	0	-2	0	1	3	30
[a,c]	0	2	-1	1	2	0	0	-2	-2	1	3	30
[a,d]	0	0	9	9	0	0	0	0	0	9	7	70
[a,e]	0	2	1	1	2	0	0	-2	-2	-1	3	30
[b,c]	-2	2	-2	2	2	0	2	-2	0	2	2	20
[b,d]	-2	0	-1	1	0	0	2	0	2	1	4	40
[b,e]	-2	2	0	2	2	0	2	-2	0	0	4	40
[c,d]	-2	0	-1	1	0	2	2	0	0	1	4	40
[c,e]	-2	2	0	2	2	2	2	-2	-2	0	2	20
[d,e]	-2	0	1	1	0	2	2	0	0	-1	4	40
Dist	4	4	2	0	4	6	4	4	6	2		
Perc	40	40	20	0	40	60	40	40	60	20		

Tabela 4.4: Instância Tai25a - Comparação entre pares de algoritmos e pares de critérios.

Para facilitar a visualização alterou-se a saída original da matriz gerada pela função *ComparaParesCriteriosParesAlgoritmos*(), onde se trocaram os valores iguais a *zero* por "1"e os demais por espaços, como pode ser visto na Tabela 4.5.

A última linha e coluna da Tabela 4.5 servem para a avaliação dos próprios indicadores. Se todos os pares indicadores apresentarem discordâncias muito elevadas, por exemplo: acima dos 75%, será conveniente um questionamento sobre a sua validade.

Para a instância $Tai25a$, utilizada como exemplo, pode-se observar que apenas o par (a,d) de critérios apresentou discordância mais elevada (70%). Os demais se

	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	1	-	-	-	-	-	1	-	1	-	3	30
[a,c]	1	-	-	-	-	1	1	-	-	-	3	30
[a,d]	1	1	-	-	1	1	1	1	1	-	7	70
[a,e]	1	-	-	-	-	1	1	-	-	-	3	30
[b,c]	-	-	-	-	-	1	-	-	1	-	2	20
[b,d]	-	1	-	-	1	1	-	1	-	-	4	40
[b,e]	-	-	1	-	-	1	-	-	1	1	4	40
[c,d]	-	1	-	-	1	-	-	1	1	-	4	40
[c,e]	-	-	1	-	-	-	-	-	-	1	2	20
[d,e]	-	1	-	-	1	-	-	1	1	-	4	40
Dist	4	4	2	0	4	6	4	4	6	2		
Perc	40	40	20	0	40	60	40	40	60	20		

Tabela 4.5: Instância Tai25a - Comparação simplificada entre pares de algoritmos e pares de critérios.

mostraram bastante concordantes, o que indica boa capacidade de avaliação dos critérios para os algoritmos aplicados a esta instância.

Situação praticamente análoga acontece quando observamos as colunas, excluindo apenas a coluna referente o par de algoritmos $[1, 5]$, que não apresentou discordância. Portanto, em relação a esta instância, os dois algoritmos se equivalem.

Mesmo que as últimas linha e coluna nos forneçam parâmetros de comparação, tais percentuais representam apenas um comparativo entre pares de algoritmos ou critérios. Para instâncias cuja saída apresenta uma matriz densa (como a $Tai25a$), fica difícil inferir uma causa que justifique esse resultado. No entanto, apresentamos no Apêndice A deste trabalho uma avaliação através desse método, para as Comparações 1 a 3 (ver adiante), em 10 instâncias, 2 de cada classe. São elas:

- Classe 1 $\Rightarrow$ Tai25a e Tai80b;
- Classe 2 $\Rightarrow$ Nug30 e Sko42;
- Classe 3 $\Rightarrow$ Esc32a e Kra30a;
- Classe 4 $\Rightarrow$ Lipa40b e Lipa70b;
- Classe 5 $\Rightarrow$ Dre90 e Tai75e05.

O critério de escolha das instâncias foi o diferente comportamento obtido por elas na execução dos algoritmos, para as sementes utilizadas.

4.4.2 Método *de Ordenação por Pesos* - MOP

Escolhemos propor uma nova técnica de avaliação baseada na atribuição de pesos, mais rápida, de melhor acompanhamento e que pudesse ser aplicada a todas as instâncias de uma classe.

Tal proposta de método de comparação faz uso de parte do método 4.5, onde usamos até a função , cujo resultado final forneceu o algoritmo de comparação nomeado por nós de *Método de Ordenação por Pesos*, ou simplesmente método de classificação *MOP*, cujo algoritmo 4.6, encontra-se a seguir:

Algoritmo 19 : Método de Ordenação por Pesos - MOP

```
1  LeParametrosEntrada();
2  Para (cada instância não lida) faça
3     LeDadosInstancia();
4     OrdenaPorValor();
5     RearranjaValoresIguais();
6     ReorganizaEspacos();
7     ConstroiNovaMatrizOrdem();
8     AchaPontuacao();
9     AcumulaPontuacaoInstancia();
10    GravaComparacaoInstancia();
   fim Para
11 GravaComparacaoGeral();
```

Figura 4.6: Algoritmo de comparação *Método de Ordenação por Pesos*.

Com a matriz gerada após a aplicação da função *OrdenaPorValor()*, que pode ser observada na Tabela 4.6, para cada critério utilizado, identificamos quais valores são iguais que, para esse exemplo, são identificados por uma cor diferente na matriz.

	1	2	3	4	5
a	3 - 8.0000	2 - 9.0000	1 - 10.0000	4 - 10.0000	5 - 10.0000
b	5 - 0.5500	3 - 0.5700	1 - 0.5800	2 - 0.6800	4 - 0.7100
c	3 - 5.0400	5 - 5.4500	1 - 5.7600	4 - 7.1100	2 - 7.1200
d	1 - 0.0000	4 - 0.0000	5 - 0.0000	2 - 0.0900	3 - 176.3800
e	3 - 472.0000	4 - 481.5700	5 - 483.9400	1 - 557.5400	2 - 564.9100

Tabela 4.6: Instância Tai25a - Matriz ordenada por valores - Original.

Após isto, fizemos uma alteração na matriz, vista na Tabela 4.6, através da função *RearranjaValoresIguais()*), cujo resultado pode ser conferido na Tabela 4.7. Com isso, valores iguais, mas que antes estavam em diferentes ordens passaram a compor a mesma posição, o que aconteceu com os algoritmos 1, 4 e 5, nos critérios a e d.

Caso passem a existir espaços entre valores após o rearranjo, esses são remanejados para a primeira posição vazia após os valores iguais e assim por diante, (função *EliminaEspacos()*). Como no caso dos algoritmos 2 e 3 na linha do critério d, que passaram da posição 4 e 5 (Tabela 4.8) para as posições 2 e 3 (Tabela 4.8), respectivamente .

	1	2	3	4	5
a	3 - 8.0000	2 - 9.0000	1 - 10.0000 4 - 10.0000 5 - 10.0000		
b	5 - 0.5500	3 - 0.5700	1 - 0.5800	2 - 0.6800	4 - 0.7100
c	3 - 5.0400	5 - 5.4500	1 - 5.7600	4 - 7.1100	2 - 7.1200
d	1 - 0.0000 4 - 0.0000 5 - 0.0000			2 - 0.0900	3 - 176.3800
e	3 - 472.0000	4 - 481.5700	5 - 483.9400	1 - 557.5400	2 - 564.9100

Tabela 4.7: Instância Tai25a - Matriz ordenada por valores - Após rearranjo de valores iguais.

	1	2	3	4	5
a	3 - 8.0000	2 - 9.0000	1 - 10.0000 4 - 10.0000 5 - 10.0000		
b	5 - 0.5500	3 - 0.5700	1 - 0.5800	2 - 0.6800	4 - 0.7100
c	3 - 5.0400	5 - 5.4500	1 - 5.7600	4 - 7.1100	2 - 7.1200
d	1 - 0.0000 4 - 0.0000 5 - 0.0000	2 - 0.0900	3 - 176.3800		
e	3 - 472.0000	4 - 481.5700	5 - 483.9400	1 - 557.5400	2 - 564.9100

Tabela 4.8: Instância Tai25a - Matriz ordenada por valores - Após eliminação de espaços.

Feito isso, a função *ConstroiNovaMatrizOrdem()*, como o próprio nome diz, constrói uma nova matriz, onde cada linha representa um algoritmo avaliado e cada coluna o somatório da ordem que o respectivo algoritmo obteve em cada critério. Isto fica mais claro se olharmos a matriz da Tabela 4.9.

	Ordem				
Algoritmos	1o.	2o.	3o.	4o.	5o.
1	1	0	3	1	0
2	0	2	0	1	2
3	3	1	1	0	0
4	1	1	1	1	1
5	2	1	2	0	0

Tabela 4.9: Instância Tai25a - Matriz com totalizações de ordenação de valor por algoritmo.

De posse da matriz da Tabela 4.9, propomos atribuir pesos para cada ordem, de modo a procurar quantificar sua relevância em cada algoritmo. Como um algoritmo que ficou em 1^o lugar deve possuir uma importância relativa maior que o algoritmo que ficou em 2^o lugar e maior ainda que o algoritmo que ficou em 3^o lugar e assim por diante, adotou-se um critério que pudesse nos auxiliar nesta tarefa, porém evitando resultados finais iguais.

Para isso, propomos a expressão 4.3, onde:

k $\longrightarrow$ valor da base (k$\geq$2);

w $\longrightarrow$ total de algoritmos;

i $\longrightarrow$ classe da instância;

j $\longrightarrow$ ordem do algoritmo;

O $\longrightarrow$ matriz (i, j).

$$Pontuacao_i = \sum_{j=1}^{w} O_{ij} * k^{w-j} \tag{4.3}$$

Isto é garantido pelo fato desta função ser injetiva sobre o conjunto dos naturais, o que nos dá a certeza de que cada conjunto de valores nela inserido produza um valor diferente para a função.

O valor de k, inteiro, deve ser ajustado de modo a fornecer resultados facilmente avaliáveis, ou seja, de modo que os pesos não tenham valores demasiadamente próximos. Após algumas tentativas, escolhemos $k = 3$.

Dentro do algoritmo MOP, é executada na linha 8 (*AchaPontuacao()*). Por último, acumula-se a pontuação da instâncias atual, (*AcumulaPontuacaoInstancia()*) e grava-se a comparação em disco, função *GravaComparacaoInstancia()*. Ao término do algoritmo, grava-se em disco o somatório das comparações de todas as instâncias da classe atual, (*GravaComparacaoGeral()*).

Tomando como exemplo a instância $Tai25a$, a Tabela 4.10 mostra para cada algoritmo o número de vezes de sua chegada na ordem indicada. Aplicando-se esses valores a (4.3), verificamos que o algoritmo 3 obteve o melhor resultado (279), ficando o algoritmo 2 com a pior pontuação (59):

Algoritmos	Ordem					Pontuação
	1o.	2o.	3o.	4o.	5o.	
1	1	0	3	1	0	111
2	0	2	0	1	2	59
3	3	1	1	0	0	279
4	1	1	1	1	1	121
5	2	1	2	0	0	207

Tabela 4.10: Instância Tai25a - Matriz com totalizações de ordenação de valor por algoritmo - método MOP.

É importante ressaltar que os resultados da aplicação desta função são consistentes apenas dentro de uma situação dada; as ordens obtidas em duas situações diferentes não são, nem coerentes, nem suas respectivas pontuações podem ser somadas.

4.5 Comparações entre os Algoritmos

As comparações feitas entre os algoritmos seguiram a ordem estabelecida na Figura 4.7.

Comparacao1 Controle Único Memória	Comparacao2 Controle Duplo Memória	Comparacao3 Outras Metaheurísticas	Comparacao4 Final
1 - VnsBasico	1 - VnsBasico	1 - VnsBasico	1 - Melhor Comparacao1
2 - AgitacaoMov	2 - DuploMovMov	2 - TabuTaillard	2 - Melhor Comparacao2
3 - AgitacaoVert	3 - DuploMovVert	3 - ILSStutz	3 - Melhor Comparacao3
4 - MelhoraMov	4 - DuploVertMov	4 - GraspRangel	
5 - MelhoraVert	5 - DuploVertVert		

Figura 4.7: Comparações entre os algoritmos.

Nos testes preliminares, as versões $UltimaMov$ e $UltimaVert$ apresentaram resultados muito inferiores às demais versões de controle único de memória, tendo sido por isso excluídas das comparações aqui apresentadas.

Na comparação 1 (*Comparacao1*), utilizou-se a versão básica de VNS com todas as versões de VNS com controle único de memória, descritas no início do Capítulo 3.

Na comparação 2 (*Comparacao2*), usou-se novamente a versão básica do VNS, comparando-a com com todas as versões de controle duplo de memória.

Na comparação 3 (*Comparacao3*), fez-se a comparação da versão básica do VNS com outras metaheurísticas.

Na comparação 4 (*Comparacao4*), foram incluídos os melhores algoritmos, conforme as três comparações anteriores.

Em todas as comparações realizadas, as instâncias foram divididas conforme indicado na Seção 4.2.

4.5.1 Testes de desempenho com controle único de memória

Tabelas semelhantes à Tabela 4.11 apresentam na primeira coluna o número e a denomização do algoritmo testado e, na segunda, o total de instâncias que alcançaram, *pelo menos uma vez*, o $VOMVC$. As quatro colunas seguintes detalham o total das instâncias onde não se alcançou o $VOMVC$ em *nenhuma das execuções*, dispostas em dois grupos, conforme o erro médio: igual ou menor que $2,0\%$, ou acima desse valor. Para ambos os casos, as colunas de quantidade (Qtd) mostram o total das instâncias sem acerto para as duas faixas de erro. Essas quantidades são, também, expressas em percentual (Perc(%)) nas colunas 4 e 6. Nas cinco colunas seguintes, apresentam-se respectivos critérios de comparação, primeiro o critério a $(10 - VOMVC)$, onde tem-se o total de execuções *que não alcançaram* o

$VOMVC$ e depois, em percentual, as médias obtidas nos critérios b $((afast, \%))$, c $(I(nsol, erro))$, d $((t, exec))$, e e $((t, estag))$.

Já as tabelas semelhantes à 4.12, apresentam na primeira coluna a classe das instâncias e, nas cinco colunas seguintes, os percentuais referentes à posição obtida para cada algoritmo no total dos cinco critérios (observar que a soma de cada linha é 100%).

4.5.1.1 Classe 1 - Instâncias Aleatórias com Distâncias e Fluxos Uniformemente Distribuídos (47 instâncias)

Nenhum dos algoritmos conseguiu resolver todas as instâncias da classe 1. Esta classe contém as instâncias Chr_{xx}, $Tai_{xx}a$, $Tai_{xx}b$, Rou_{xx}, Tho_{xx} e Wil_{xx}, envolvendo assim, de acordo com a experiência da literatura, instâncias de diversos níveis de dificuldade.

A versão básica do VNS foi a que apresentou o pior resultado quantitativo (ver Tabela 4.11), porém seu tempo médio de execução (penúltima coluna), foi melhor que o das versões com memória. Um provável motivo pode estar no aumento de tempo necessário para o controle de memória.

Em todos os casos, o erro médio ficou abaixo de $1,0\%$. Podemos citar as instâncias Tai80b e Tai25a, como exemplo de instâncias onde nem sempre se alcançou o $VOMVC$, o que as faz uma escolha interessante para o método *Condorcet*, a ser utilizado no Apêndice deste trabalho.

Classe 1	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	33	14	0,01	0	-	182	0,38	3,52	19,18	216,76
2 - AgitacaoMov	34	13	0,01	0	-	181	0,18	1,60	32,40	220,24
3 - AgitacaoVert	36	11	0,01	0	-	183	0,41	3,45	40,52	239,15
4 - MelhoraMov	35	12	0,01	0	-	173	0,37	3,38	25,60	196,10
5 - MelhoraVert	34	13	0,01	0	-	177	0,46	3,47	27,23	226,75

Tabela 4.11: Comparação 1 - Classe 1 - 47 instâncias

Nota-se que o resultado do Algoritmo 2 (*AgitacaoMov*) para o indicador de erro (nona coluna), inferior à metade do verificado nos outros algoritmos, é compatível com o baixo valor do seu afastamento médio (oitava coluna).

A função (4.3) foi aplicada, aqui, sobre os valores numéricos originais que geraram as porcentagens. Convém esclarecer que esse universo corresponde ao número de pares (instância, critério) que, para a Classe 1, tem cardinalidade 47 x 5 = 235.

A partir da Tabela 4.11 é possível observar que a pontuação final encontrada na Tabela 4.12 foi fortemente influenciada pelo critério e (complemento do tempo médio de estagnação). Isto indica que, para esta classe, mesmo quando o algoritmo

Classe - 1	Ordem de desempenho dos Algortimos - Controle Único					
(47 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	5o. (%)	Pontuação
1 - VnsBasico	62,98	8,94	12,77	8,51	6,81	12901
2 - AgitacaoMov	60,85	12,34	8,94	11,06	6,81	12649
3 - AgitacaoVert	62,13	13,19	11,91	7,23	5,53	12979
4 - MelhoraMov	68,09	11,06	10,21	6,38	4,26	**13933**
5 - MelhoraVert	62,98	13,19	11,06	5,53	7,23	13115

Tabela 4.12: Comparação 1 - Classe 1 - Classificação dos algoritmos

MelhoraMov não alcançou o $VOMVC$, na média, sua estagnação ocorreu mais próxima do critério de parada.

4.5.1.2 Classe 2 - Fluxos Aleatórios em Grades (30 instâncias)

Para a Classe 2, as Tabelas 4.13 e 4.14 mostraram-se bastante condizentes, e o critério c (Indicador de Qualidade) associado ao critério e (complemento do tempo médio de estagnação) foram decisivos na pontuação final. Nota-se que esta classe apresentou uma variação menor entre os erros médios dos algoritmos do que a Classe 1.

Classe 2	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	20	10	0,00	0	-	121	0,17	1,52	11,67	246,82
2 - AgitacaoMov	20	10	0,00	0	-	125	0,18	1,60	32,40	220,24
3 - AgitacaoVert	21	9	0,00	0	-	128	0,19	1,82	19,40	228,38
4 - MelhoraMov	20	10	0,00	0	-	130	0,20	1,87	30,35	246,89
5 - MelhoraVert	20	10	0,00	0	-	128	0,20	1,84	4,18	232,33

Tabela 4.13: Comparação 1 - Classe 2 - 30 instâncias

Seja quanto ao número de soluções ótimas, ou em relação às colunas de critério de avaliação (ver Tabela 4.13), as diferenças existentes entres os algoritmos não foram tão significativas, exceto pelo critério d (o tempo médio de execução) que, para o Algoritmo *MelhoraMov* resultou pior que os demais, porém isto só não foi tão impactante na pontuação da Tabela 4.14 em vista da ocorrência de um afastamento inferior ao dos demais algoritmos.

Classe - 2	Ordem de desempenho dos Algortimos - Controle Único					
(30 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	5o. (%)	Pontuação
1 - VnsBasico	68,00	11,33	8,67	4,00	8,00	8868
2 - AgitacaoMov	69,33	6,67	9,33	10,67	4,00	**8874**
3 - AgitacaoVert	62,00	15,33	11,33	6,67	4,67	8344
4 - MelhoraMov	62,67	6,67	8,00	12,00	10,67	8062
5 - MelhoraVert	66,00	7,33	12,67	7,33	6,67	8530

Tabela 4.14: Comparação 1 - Classe 2 - Classificação dos algoritmos

4.5.1.3 Classe 3 - Problemas da Vida Real (38 instâncias)

Todos os algoritmos conseguiram chegar ao $VOMVC$ das instâncias desses problemas, o que pode ser constatado através da Tabela 4.15. Como já citado anteriormente no Capítulo 3, uma justificativa para isso seria o fato de que as matrizes de fluxos possuem um grande numero de valores zerados e que eles não são nada uniformemente distribuídos.

Classe 3	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	38	0	-	0	-	17	0,21	0,89	7,34	74,14
2 - AgitacaoMov	38	0	-	0	-	7	0,09	0,28	16,73	43,68
3 - AgitacaoVert	38	0	-	0	-	7	0,21	0,38	11,02	78,18
4 - MelhoraMov	38	0	-	0	-	16	0,25	0,74	7,72	76,62
5 - MelhoraVert	38	0	-	0	-	6	0,19	0,38	16,52	44,05

Tabela 4.15: Comparação 1 - Classe 3 - 38 instâncias

Mesmo que os valores apresentados pelo Algoritmo 2 (*AgitacaoMov*) tenham sido quase todos superior aos demais algoritmos (Tabela 4.16), os critérios a e d foram foram decisivos para que esse algoritmo (*MelhoraMov*) obtivesse uma melhor pontuação nesta classe.

Pode-se observar que algumas versões apresentaram um desenpenho superior ao da versão básica do VNS, principalmente no que se refere aos critérios a e c. Isto indica que, para esta classe, o uso de memória trouxe benefícios visíveis ao VNS.

Classe - 3	Ordem de desempenho dos Algortimos - Controle Único					
(38 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	5o. (%)	Pontuação
1 - VnsBasico	81,05	7,37	3,68	4,21	3,68	12946
2 - AgitacaoMov	82,63	6,84	4,74	2,11	3,68	13168
3 - AgitacaoVert	81,05	5,79	7,37	4,21	1,58	12924
4 - MelhoraMov	80,53	6,32	7,37	4,74	1,05	12872
5 - MelhoraVert	84,74	7,37	2,63	3,16	2,11	**13486**

Tabela 4.16: Comparação 1 - Classe 3 - Classificação dos algoritmos

4.5.1.4 Classe 4 - Instâncias Aleatórias Semelhantes aos Problemas da Vida Real (16 instâncias)

A partir da Classe 4, (Tabela 4.17), encontramos instâncias cujo erro médio ficou acima dos 2% e que, de fato, para todos os algoritmos, ficou sempre em torno dos $19,60\%$. A diferença aparece nas instâncias que alcançaram o $VOMVC$, onde o melhor caso foi para o algoritmo (*MelhoraMov*).

Se analisarmos apenas os valores da segunda coluna, para os Algoritmos 2 e 4, na Tabela 4.17, encontremos valores mais interessantes para o Algoritmo 4 (*Melho-*

Classe 4	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	10	5	0,66	1	19,97	92	6,13	43,09	20,98	326,48
2 - AgitacaoMov	9	4	0,61	3	19,10	103	7,09	55,56	15,87	331,77
3 - AgitacaoVert	10	4	0,64	2	19,88	96	7,06	50,61	24,16	343,55
4 - MelhoraMov	11	4	0,60	1	19,98	97	7,11	51,47	37,35	345,17
5 - MelhoraVert	9	4	0,60	3	19,49	100	7,12	55,14	6,88	370,63

Tabela 4.17: Comparação 1 - Classe 4 - 16 instâncias

raMov), porém quando comparados ao valores obtidos na Tabela 4.18 o que ocorre é o contrário.

Isto ocorreu porque o critério d (tempo médio de execução) e o critério e (complemento do tempo médio de estagnação) apresentaram valores médios inferiores para o Algoritmo 2, o que nos leva a inferir que que, nesta classe, quando alcançou o VOMVC, o Algoritmo 2 o fez com menos iterações médias e, quando não alcançou, sua estagnação média foi mais próxima do critério de parada que a do Algoritmo 4.

Classe - 4	Ordem de desempenho dos Algortimos - Controle Único					
(16 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	5o. (%)	Pontuação
1 - VnsBasico	46,25	20,00	10,00	16,25	7,50	3546
2 - AgitacaoMov	51,25	11,25	15,00	7,50	15,00	**3702**
3 - AgitacaoVert	45,00	15,00	15,00	17,50	7,50	3396
4 - MelhoraMov	42,50	17,50	18,75	10,00	11,25	3300
5 - MelhoraVert	40,00	18,75	17,50	13,75	10,00	3164

Tabela 4.18: Comparação 1 - Classe 4 - Classificação dos algoritmos

4.5.1.5 Classe 5 - Instâncias Difíceis para as Metaheurísticas (37 instâncias)

Como o próprio nome da classe sugere, esta classe é constituída de instâncias de difícil resolução através das metaheurísticas. É interessante observar que, apesar disso, as Dre_{xx} são polinomiais com métodos exatos.

Os Algoritmos 2 e 3 obtiveram resultados melhores que os observados nos demais algoritmos. Uma razão para isso pode estar relacionada ao controle de memória que, para ambos os casos, foi inserido na fase da agitação da solução atual, o que pode ser corroborado pelo maior número instâncias onde se alcançou o $VOMVC$, (Tabela 4.19). Dentro desta hipótese, é possível também associar a esse controle de memória os menores percentuais de erro médio superiores a 2%.

Ainda na Tabela 4.19, um aspecto interessante a ser observado se refere às instâncias cujos erros médios ficaram abaixo dos 2%. Além desses valores ficarem muito próximos, as respostas foram idênticas para um subconjunto de instâncias. Uma inspeção mais detalhada mostrou que se trata, em todos os casos, das

Classe 5	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	18	8	0,23	11	105,72	279	119,95	853,96	18,92	420,36
2 - AgitacaoMov	21	8	0,23	8	88,97	255	126,51	832,46	31,07	412,89
3 - AgitacaoVert	21	8	0,23	8	95,30	263	124,21	833,37	39,78	388,94
4 - MelhoraMov	18	8	0,23	11	121,92	272	121,34	836,87	22,88	391,59
5 - MelhoraVert	18	8	0,23	11	152,19	268	119,13	833,94	41,46	396,17

Tabela 4.19: Comparação 1 - Classe 5 - 37 instâncias

instâncias: *Pal40*, *Pal50*, *Pal60*, *Pal70*, *Pal80*, *Pal100*, *Pal150* e *Pal200*. De fato, a experiência prévia com essas instâncias mostra que elas possuem muitos ótimos locais de boa qualidade, o que dificulta o atingimento de um ótimo global.

Para a Classe 5, a versão básica do VNS foi a que apresentou o resultado menos promissor, apesar do menor tempo de execução: quando comparada aos algoritmos 4 e 5 com controle de memória na melhora da solução atual (Tabela 4.19), pode-se observar que o erro desses algoritmos foi menor (na ordem de 2%). Porém, o critério responsável pela diferença na pontuação geral foi o critério e (indicador de qualidade), o que significa dizer que, para esta classe, os algoritmos com controle de memória, na média, alcançam mais vezes o $VOMVC$ e quando isto não aconteceu, menor foi o erro apresentado.

Classe - 5	Ordem de desempenho dos Algortimos - Controle Único					
(37 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	5o. (%)	Pontuação
1 - VnsBasico	47,03	16,76	13,51	10,27	12,43	8189
2 - AgitacaoMov	50,81	18,92	9,19	9,19	11,89	**8785**
3 - AgitacaoVert	50,27	18,38	10,81	11,35	9,19	8711
4 - MelhoraMov	49,73	15,14	17,30	14,59	3,24	8583
5 - MelhoraVert	46,49	22,16	14,05	11,89	5,41	8383

Tabela 4.20: Comparação 1 - Classe 5 - Classificação dos algoritmos

4.5.1.6 Todas as Classes - Comparação 1

Uma análise comparativa das formas de inserção de memória no VNS básico mostra que as versões com memória na agitação na vizinhança atual foram mais promissoras que as versões com memória na melhora na vizinhança atual. O Algoritmo 3 (*AgitacaoVert*) alcançou o $VOMVC$ em mais instâncias que os demais algoritmos propostos (Tabela 4.21) e, em relação à versão básica do VNS, tal melhora foi de aproximadamente $5,89\%$. Também o bom desempenho do Algoritmo 2 (*AgitacaoMov*) pode ser visto na Tabela 4.23, já que ficou em primeiro lugar em três classes de instâncias, perdendo apenas nas Classes 1 e 3.

Em uma análise comparativa dos dois tipos de controle, percebe-se através da Tabela 4.21, que o controle por movimento se mostrou mais robusto, pois indepen-

Comparacao1	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	119	37	0,18	12	25,14	691	25,37	180,59	15,61	256,91
2 - AgitacaoMov	122	35	0,17	11	21,61	671	26,81	178,30	25,69	245,76
3 - AgitacaoVert	126	32	0,18	10	23,04	677	26,41	177,93	26,97	255,64
4 - MelhoraMov	122	34	0,17	12	28,38	688	25,85	178,87	24,78	251,27
5 - MelhoraVert	119	35	0,17	14	34,34	679	25,42	178,95	19,25	253,99

Tabela 4.21: Comparação 1 - Todas as instâncias

dente do local de inserção (na agitação ou melhora feita na vizinhança atual) não houve variação no total de instâncias que alcançaram o $VOMVC$.

Já o controle de memória por vértice, mesmo encontrando o melhor resultado entre todos os algoritmos, na versão *AgitacaoVert*, mostrou uma variação considerável no total de instâncias que alcançaram o $VOMVC$ (ver 4.21).

A partir das pontuação obtida pelos algoritmos nas Tabelas 4.12, 4.14, 4.16, 4.18 e 4.20 contruímos uma matriz transposta, (Tabela 4.22), com sua classificação geral. Nela, os valores correspondem ao número do algoritmo e sua pontuação em cada classe.

Ordem Geral de Desempenho dos Algortimos - Controle Único					
Comparacao1	1o.	2o.	3o.	4o.	5o.
Classe - 1	1 - 12901	2 - 12649	3 - 12979	4 - 13933	5 - 13115
Classe - 2	1 - 8868	2 - 8874	3 - 8344	4 - 8062	5 - 8530
Classe - 3	1 - 12946	2 - 13168	3 - 12924	4 - 12872	5 - 13486
Classe - 4	1 - 3546	2 - 3702	3 - 3396	4 - 3300	5 - 3164
Classe - 5	1 - 8189	2 - 8785	3 - 8711	4 - 8583	5 - 8383

Tabela 4.22: Comparação 1 - Classificação Geral dos algoritmos

Assim como feito em 4.2, aplicamos a função $OrdenaPorValor()$, obtendo os valores da Tabela 4.23.

Ordem Geral de Desempenho dos Algortimos - Controle Único					
Comparacao1	1o.	2o.	3o.	4o.	5o.
Classe - 1	4 - 13933	5 - 13115	3 - 12979	1 - 12901	2 - 12649
Classe - 2	2 - 8874	1 - 8868	5 - 8530	3 - 8344	4 - 8062
Classe - 3	5 - 13486	2 - 13168	1 - 12946	3 - 12924	4 - 12872
Classe - 4	2 - 3702	1 - 3546	3 - 3396	4 - 3300	5 - 3164
Classe - 5	2 - 8785	3 - 8711	4 - 8583	5 - 8383	1 - 8189

Tabela 4.23: Comparação 1 - Classificação Geral Ordenada dos algoritmos

Como a função (4.3) garante que cada algoritmo receba uma pontuação diferente, cada um deles teve uma ordem distinta em cada classe.

O passo seguinte foi, para cada algoritmo da comparação atual, totalizá-lo em cada possível ordem. Exempliicando como isto foi feito, vamos utilizar a segunda

coluna da Tabela 4.23 que, como já explicado antes, contém o número e a pontuação do algoritmo que, nesse caso, se refere à primeira colocação em cada classe. Ao observarmos esta coluna, veremos que o algoritmo 2 (*AgitacaoMov*) ficou em primeiro lugar nas classes 2, 4 e 5, enquanto o algoritmo 4 (*MelhoraMov*) ficou em primeiro na classe 1 e o algoritmo 5 (*MelhoraVert*) ficou em primeiro na classe 3, cujos totais geraram a Tabela 4.24:

Desempenho Geral dos Algortimos - Controle Único					
Comparacao1	1o.	2o.	3o.	4o.	5o.
1 - VnsBasico		2	1	1	1
2 - AgitacaoMov	3	1			1
3 - AgitacaoVert		1	2	2	
4 - MelhoraMov	1		1	1	2
5 - MelhoraVert	1	1	1	1	1

Tabela 4.24: Comparação 1 - Classificação Geral por Algoritmos

Com o intuito de conseguirmos um comparativo mais preciso dos resultados obtidos nos algoritmos propostos e assim escolher a versão a ser incluída na próxima fase da avaliação, utilizamos os valores contidos na Tabela 4.24, aplicamos (4.3), obtendo o resultado expresso na Tabela 4.25 (onde as colocações estão expressas em porcentagem).

Todas Classes	Ordem de desempenho dos Algortimos - Controle Único					
Algoritmos	1o. (%)	2o. (%)	3o. (%)	4o. (%)	5o. (%)	Pontuação
1 - VnsBasico	-	40,00	20,00	20,00	20,00	67
2 - AgitacaoMov	60,00	20,00	-	-	20,00	**271**
3 - AgitacaoVert	-	20,00	40,00	40,00	-	51
4 - MelhoraMov	20,00	-	20,00	20,00	40,00	95
5 - MelhoraVert	20,00	20,00	20,00	20,00	20,00	121

Tabela 4.25: Comparação 1 - Classificação final dos algoritmos

Isto quer dizer, por exemplo, que dentre os algoritmos e classes verificados, o Algoritmo 2 (*AgitacaoMov*) ficou em primeiro lugar em 3 classes (60%) e que os algoritmos propostos *MelhoraMov* e *MelhoraVert* ficaram em primeiro lugar em apenas 1 classe (20%), cada um. Nesta comparação, nem a versão *AgitacaoVert*, nem a versão básica do VNS conseguiram ficar em primeiro lugar na pontuação geral.

O resultado encontrado na Tabela 4.25 aponta para as seguintes considerações sobre o uso **isolado** de memória no VNS:

- Quando não se alcançou o $VOMVC$, para erros médios abaixo de 2%, todas as versões apresentaram desempenho comparável, porém quando esses erros ficaram acima dos 2% (logo, nas instâncias mais difíceis) começaram a aparecer diferenças mais significativas.

- O uso de memória na agitação feita na vizinhança atual com controle por movimento (*AgitacaoMov*) mostrou-se mais consistente que o controle por vértice (*AgitacaoVert*) e o uso da memória na melhora na vizinhança atual (*MelhoraMov* e *MelhoraVert*) pois, na média, quando esse algoritmo alcançou o $VOMVC$, o fez com menos iterações, e quando isto não ocorreu, sua estagnação ficou mais próxima do critério de parada, o que justifica sua colocação na Tabela 4.25.

- Quanto à inserção do controle de memória, os resultados mostraram que, por ser menos restritivo, os controles por movimento (*AgitacaoMov* e *MelhoraMov*) auxiliaram, na média, no alcance de melhores resultados que os obtidos com o controle de memória por vértice (*AgitacaoVert* e *MelhoraVert*).

- A partir dos valores encontrados na Tabela 4.25, percebe-se que a escolha do tipo de controle na agitação da solução atual modifica muito o comportamento final dos algoritmos, já que o algoritmo *AgitacaoVert* mostrou-se muito diferente do *AgitacaoMov* nos resultados finais (nesse caso), muito aquém. O mesmo não se pode falar sobre as versões *MelhoraMov* e *MelhoraVert*, já que houveram interseções na maioria dos resultados obtidos nas ordens (em 4.25).

4.5.2 Testes de desempenho com controle duplo de memória

Neste ponto, iniciamos a comparação entre o algoritmo básico do VNS (*VnsBasico*), e todos os algoritmos com controle duplo de memória: *DuploMovMov*, *DuploMovVert*, *DuploVertMov* e *DuploVertVert*.

4.5.2.1 Classe 1 - Instâncias Aleatórias com Distâncias e Fluxos Uniformemente Distribuídos (47 instâncias)

Mesmo que as instâncias que não alcançaram o $VOMVC$ tenham ficado abaixo de 2%, percebe-se um aumento do erro médio em relação à versão básica do VNS, porém estas versões conseguiram alcançar mais vezes o $VOMVC$, ver critério *a* (Tabela 4.26).

Classe 1	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	33	14	0,01	0	-	182	0,38	3,52	19,18	216,76
2 - DuploMovMov	34	13	0,96	0	-	172	0,39	3,47	22,04	205,50
3 - DuploMovVert	34	13	1,05	0	-	180	0,49	3,75	23,73	244,35
4 - DuploVertMov	33	14	0,93	0	-	176	0,46	3,51	23,71	231,46
5 - DuploVertVert	33	14	0,95	0	-	174	0,44	3,54	18,51	218,04

Tabela 4.26: Comparação 2 - Classe 1 - 47 instâncias

Se, primeiramente, considerarmos os algoritmos que mais vezes alcançaram o $VOMVC$, teremos um conjunto de candidatos formados pelos algoritmos *DuploMovMov* e *DuploMovVert*, porém o que foi determinante para que o algoritmo *DuploMovMov* obtivesse uma melhor pontuação final foram os critérios a, c e e já que esse algoritmo obteve também um menor complemento do tempo médio de estagnação que os demais, o que pode ser visto na Tabela 4.26 e ratificado na 4.27.

O algoritmo *DuploVertVert* obteve menor tempo médio de execução que todos os demais algoritmos, o que lhe garantiu a segunda posição.

Classe - 1	Ordem de desempenho dos Algortimos - Controle Duplo					
(47 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	5o. (%)	Pontuação
1 - VnsBasico	64,26	10,21	10,64	8,94	5,96	13181
2 - DuploMovMov	68,51	8,51	10,21	8,94	3,83	**13869**
3 - DuploMovVert	60,85	14,89	4,68	11,49	8,09	12727
4 - DuploVertMov	62,13	13,19	11,49	5,53	7,66	12963
5 - DuploVertVert	67,66	10,21	10,21	5,53	6,38	13797

Tabela 4.27: Comparação 2 - Classe 1 - Classificação dos algoritmos

4.5.2.2 Classe 2 - Fluxos Aleatórios em Grades (30 instâncias)

Como também observado na Classe 1, nos algoritmos com controle duplo, as instâncias que não alcançaram o $VOMVC$ obtiveram erros médios bastante próximos entre elas, porém bem maiores que o da versão básica do VNS, (Tabela 4.28). O fato mais relevante nas versões com dois controles é que os algoritmos 2 e 3 permitiram alcançar uma unidade a mais do $VOMVC$ que os algoritmos 4 e 5.

Classe 2	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	20	10	0,00	0	-	121	0,17	1,52	11,67	246,82
2 - DuploMovMov	21	9	0,41	0	-	131	0,18	1,76	37,90	236,16
3 - DuploMovVert	21	9	0,40	0	-	126	0,18	1,67	23,76	233,16
4 - DuploVertMov	20	10	0,47	0	-	129	0,21	1,95	13,64	229,72
5 - DuploVertVert	20	10	0,41	0	-	129	0,18	1,73	12,72	227,70

Tabela 4.28: Comparação 2 - Classe 2 - 30 instâncias

Para a Classe 2, ao analisarmos a pontuação na Tabela 4.29 notamos que a versão básica do VNS foi melhor que todas as versões com controle duplo de memória. Isto se justifica pela Tabela 4.28, pois todos os valores encontrados para a versão básica do VNS, exceto o critério d, foram melhores que as outras versões.

4.5.2.3 Classe 3 - Problemas da Vida Real (38 instâncias)

O que pode ser visto na Tabela 4.30 é que, independente de inserção dupla ou não de memória no algoritmo básico do VNS, para as vizinhaças utilizadas neste trabalho,

Classe - 2	Ordem de desempenho dos Algortimos - Controle Duplo					
(30 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	5o. (%)	Pontuação
1 - VnsBasico	70,67	8,67	7,33	7,33	6,00	**9078**
2 - DuploMovMov	63,33	11,33	12,67	5,33	7,33	8360
3 - DuploMovVert	66,00	11,33	9,33	6,67	6,67	8644
4 - DuploVertMov	61,33	8,67	10,00	8,67	11,33	7994
5 - DuploVertVert	61,33	12,00	9,33	12,67	4,67	8128

Tabela 4.29: Comparação 2 - Classe 2 - Classificação dos algoritmos

todas as versões conseguiram chegar ao $VOMVC$ das instâncias dessa classe.

Classe 3	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	38	0	-	0	-	17	0,21	0,89	7,34	74,14
2 - DuploMovMov	38	0	-	0	-	17	0,23	1,01	8,97	75,57
3 - DuploMovVert	38	0	-	0	-	17	0,26	0,87	15,70	90,49
4 - DuploVertMov	38	0	-	0	-	11	0,24	0,70	8,74	62,40
5 - DuploVertVert	38	0	-	0	-	12	0,21	0,53	10,35	71,55

Tabela 4.30: Comparação 2 - Classe 3 - 38 instâncias

A partir da Tabela 4.31, observa-se que a versão *DuploVertVert* apresentou a melhor pontuação e que, pelos valores contidos na Tabela 4.30, podemos observar que, mesmo que ela tenha apresentado maior tempo médio de execução (critério d), seu indicador de qualidade (critério c) e complemento do tempo médio de estagnação (critério e) foram melhores.

Classe - 3	Ordem de desempenho dos Algortimos - Controle Duplo					
(38 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	5o. (%)	Pontuação
1 - VnsBasico	80,00	9,47	2,63	5,79	2,11	12880
2 - DuploMovMov	80,53	6,84	6,32	2,11	4,21	12872
3 - DuploMovVert	78,42	7,37	4,74	7,37	2,11	12574
4 - DuploVertMov	83,16	3,68	7,37	3,68	2,11	13138
5 - DuploVertVert	83,68	5,26	6,32	2,11	2,63	**13274**

Tabela 4.31: Comparação 2 - Classe 3 - Classificação dos algoritmos

4.5.2.4 Classe 4 - Instâncias Aleatórias Semelhantes aos Problemas da Vida Real (16 instâncias)

Como pode ser visto na Tabela 4.32, todos os algoritmos obtiveram erros médios muito próximos, tanto os que ficaram abaixo, quanto os que ficaram acima dos 2%.

Foi decisivo para a melhor pontuação do algoritmo *DuploVertMov* na Tabela 4.33 o critério e (complemento do tempo médio de estagnação) associado ao critéio d (tempo médio de execução).

O algoritmo *DuploMovMov*, mesmo não obtendo uma pontuação final melhor que a versão básica do VNS, mostrou resultados relativos interessantes, já que conseguiu

Classe 4	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	10	5	0,66	1	19,97	92	6,13	43,09	20,98	326,48
2 - DuploMovMov	10	4	0,60	2	19,86	100	7,12	50,05	16,53	359,16
3 - DuploMovVert	9	5	0,66	2	19,33	95	7,04	51,60	25,67	329,43
4 - DuploVertMov	9	5	0,66	2	19,13	98	6,05	53,69	11,78	287,06
5 - DuploVertVert	11	4	0,61	1	19,67	93	7,01	47,08	17,33	332,83

Tabela 4.32: Comparação 2 - Classe 4 - 16 instâncias

alcançar mais vezes o $VOMVC$ (critério a) com um tempo médio de execução menor (critério d).

O algoritmo *DuploVertVert* foi o único a alcançar o $VOMVC$ da instância *lipa*70*b*, (embora em uma única execução). Uma justificativa possível para tal desempenho seria o fato de que o duplo controle por vértice, nesse caso, possa ter contribuido na busca de uma região bem diferente das demais e mais promissora.

Classe - 4	Ordem de desempenho dos Algortimos - Controle Duplo					
(16 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	5o. (%)	Pontuação
1 - VnsBasico	46,25	16,25	15,00	11,25	11,25	3492
2 - DuploMovMov	40,00	25,00	18,75	8,75	7,50	3294
3 - DuploMovVert	45,00	10,00	13,75	13,75	17,50	3278
4 - DuploVertMov	51,25	11,25	17,50	10,00	10,00	**3722**
5 - DuploVertVert	50,00	13,75	16,25	16,25	3,75	3696

Tabela 4.33: Comparação 2 - Classe 4 - Classificação dos algoritmos

4.5.2.5 Classe 5 - Instâncias Difíceis para as Metaheurísticas (37 instâncias)

Um fato relacionado aos algoritmos de controle duplo de memória é que o erro médio acima dos 2%, ficou acima dos 16% da versão básica do VNS, (Tabela 4.34).

Classe 5	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	18	8	0,23	11	105,72	279	119,95	853,96	18,92	420,36
2 - DuploMovMov	19	7	0,22	11	132,27	270	121,65	837,91	22,37	396,48
3 - DuploMovVert	16	8	0,23	13	139,82	269	121,72	842,50	16,13	402,47
4 - DuploVertMov	17	8	0,23	12	123,56	267	121,84	832,89	23,67	409,99
5 - DuploVertVert	17	8	0,23	12	123,56	268	121,18	834,30	10,64	385,72

Tabela 4.34: Comparação 2 - Classe 5 - 37 instâncias

Há pouca variação entre os valores obtidos pelos critérios em relação aos diferentes algoritmos, contudo, os critérios responsáveis por conferir a maior pontuação ao *DuploVertVert* ,(Tabela 4.35), foram o menor tempo médio de execução (critério d) e e complemento do tempo médio de estagnação (critério e).

Classe - 5	Ordem de desempenho dos Algortimos - Controle Duplo					
(37 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	5o. (%)	Pontuação
1 - VnsBasico	44,32	20,54	13,51	11,35	10,27	7975
2 - DuploMovMov	45,41	19,46	9,19	14,05	11,89	8029
3 - DuploMovVert	50,27	15,68	11,35	11,35	11,35	8589
4 - DuploVertMov	47,57	16,22	13,51	12,43	10,27	8251
5 - DuploVertVert	51,35	16,22	18,38	9,19	4,86	**8871**

Tabela 4.35: Comparação 2 - Classe 5 - Classificação dos algoritmos

4.5.2.6 Todas as Classes - Comparação 2

A Tabela 4.36 representa os totais de 4.26, 4.28, 4.30, 4.32 e 4.34. A partir desta tabela, é possível verificar que o algoritmo (*DuploVertVert*) apresenta valores mais significativos, pois ganha dos demais algoritmos em quatro critérios, perdendo apenas para a versão básica do VNS, no critério b (afastamento médio), cuja diferença não é muito significativa.

Assim como na *Comparacao1*, optamos por uma análise mais detalhada de qual algoritmo deveria ser o escolhido para a *Comparacao3*, o que será feito a seguir.

Comparacao2	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	119	37	0,18	12	25,14	691	25,37	180,59	15,61	256,91
2 - DuploMovMov	122	33	0,44	13	30,43	690	25,91	178,84	21,56	254,57
3 - DuploMovVert	118	35	0,47	15	31,83	687	25,94	180,08	21,00	259,98
4 - DuploVertMov	117	37	0,46	14	28,54	681	25,76	178,55	16,31	244,13
5 - DuploVertVert	119	36	0,44	13	28,65	676	25,81	177,44	13,91	247,17

Tabela 4.36: Comparação 2 - Todas as instâncias

A Tabela 4.37 mostra a ordem geral dos algoritmos em relação às diferentes classes. Ela foi contruída a partir das Tabelas 4.27, 4.29, 4.31, 4.33 e 4.35 com os valores transpostos das pontuações de cada algoritmo.

Desempenho Geral dos Algortimos - Controle Duplo					
Comparacao2	1o.	2o.	3o.	4o.	5o.
Classe 1	1 - 13181	2 - 13869	3 - 12727	4 - 12963	5 - 13797
Classe 2	1 - 9078	2 - 8360	3 - 8644	4 - 7994	5 - 8128
Classe 3	1 - 12880	2 - 12872	3 - 12574	4 - 13138	5 - 13274
Classe 4	1 - 3492	2 - 3294	3 - 3278	4 - 3722	5 - 3696
Classe 5	1 - 7975	2 - 8029	3 - 8589	4 - 8251	5 - 8871

Tabela 4.37: Comparação 2 - Classificação Geral dos algoritmos

Com as informações contidas na Tabela 4.37, fizemos uso da função $OrdenaPorValor()$, gerando a Tabela 4.38.

As ordens de pontuação obtidas pelos algoritmos nas diferentes classes estão resumidas na Tabela 4.39. Aplicamos a função 4.3 a esses valores e obtivemos a

Desempenho Geral dos Algortimos - Controle Duplo					
Comparacao2	1o.	2o.	3o.	4o.	5o.
Classe 1	2 - 13869	5 - 13797	1 - 13181	4 - 12963	3 - 12727
Classe 2	1 - 9078	3 - 8644	2 - 8360	5 - 8128	4 - 7994
Classe 3	5 - 13274	4 - 13138	1 - 12880	2 - 12872	3 - 12574
Classe 4	4 - 3722	5 - 3696	1 - 3492	2 - 3294	3 - 3278
Classe 5	5 - 8871	3 - 8589	4 - 8251	2 - 8029	1 - 7975

Tabela 4.38: Comparação 2 - Classificação Geral dos algoritmos

Tabela 4.40 a seguir, cujo resultado confirma a melhor performance do algoritmo *DuploVertVert* (Tabela 4.36).

Desempenho Geral dos Algortimos - Controle Duplo					
Comparacao2	1o.	2o.	3o.	4o.	5o.
1 - VnsBasico	1		3		1
2 - DuploMovMov	1		1	3	
3 - DuploMovVert		2			3
4 - DuploVertMov	1	1	1	1	1
5 - DuploVertVert	2	2		1	

Tabela 4.39: Comparação 2 - Classificação Geral por Algoritmos

Todas Classes	Ordem de desempenho dos Algortimos - Controle Duplo					
Algoritmos	1o. (%)	2o. (%)	3o. (%)	4o. (%)	5o. (%)	Pontuação
1 - VnsBasico	20,00	-	60,00	-	20,00	327
2 - DuploMovMov	20,00	-	20,00	60,00	-	297
3 - DuploMovVert	-	40,00	-	-	60,00	171
4 - DuploVertMov	20,00	20,00	20,00	20,00	20,00	363
5 - DuploVertVert	40,00	40,00	-	20,00	-	**657**

Tabela 4.40: Comparação 2 - Classificação final dos algoritmos

Diferente da *Comparacao1*, o controle na agitação da solução, agora duplo, mostrou resultados muito diferentes entre eles (*DuploMovMov* e *DuploMovVert*) e, como visto na Tabela 4.40, com interseções nulas entre suas ordens.

Estse resultado sugere algumas considerações sobre o uso de memória no VNS com controle duplo:

- Entre as versões, percebe-se que o primeiro controle é predomimante para os resultados de erro médio acima de 2%, tempo de execução e tempo de estagnação, isto é, independentemente do segundo tipo de controle (movimento ou vértice), tais valores sofreram poucas alterações entre suas versões.

- Toda vez que se utilizou o controle por vértice (*DuploVertVert* e *DuploMovVert*), os tempos médios de execução foram menores que os observados na

versão equivalente com controle por movimento (*DuploMovMov* e *DuploMovVert*). Isto nos leva a inferir que o controle por vértice, na média, auxilia mais na convergência para uma região de busca mais promissora.

- Entre as quatro versões com controle duplo de memória, a que se mostrou mais robusta é a *DuploVertVert*. Embora o total de instâncias que alcançaram o $VOMVC$ tenha sido igual ao da versão básica do VNS, ela ganhou de todas as versões em quatro dos cinco critérios avaliados, o que a classifica para utilização na *Comparacao4*.

4.5.3 Testes de desempenho com outras metaheurísticas

Para este comparativo, escolhemos quatro metaheurísticas:

1. *VNS*. A mesma versão básica do VNS, que foi utilizada nas comparações anteriores.

2. *Busca Tabu*. Utilizamos a versão desenvolvida e disponibilizada na internet por Taillard (90), também conhecida como Tabu Robusto. A única alteração feita no código foi possibilitar o uso das nossas sementes.

3. *ILS*. A partir do trabalho feito por Stützle (109), desevolvemos uma aplicação, onde fizemos uso da nossa Vizinhança 3 (ver 4.1.3), como critério de busca local.

4. *GRASP*. A versão utilizada do GRASP nos foi gentilmente cedida por Rangel, fruto da sua tese de Doutorado (30).

4.5.3.1 Classe 1 - Instâncias Aleatórias com Distâncias e Fluxos Uniformemente Distribuídos (47 instâncias)

Mesmo que se observe no algoritmo 2 a existência de uma instância acima dos 2%, além de conseguir alcançar mais $VOMVC$, (Tabela 4.41), o algoritmo ganhou em quase todos os critérios de desempenho, (Tabela 4.42).

Classe 1	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	33	14	0,01	0	-	182	0,38	3,52	19,18	216,76
2 - TabuTaillard	36	10	0,58	1	9,51	141	0,33	3,37	24,29	91,17
3 - IlsStutzle	32	15	1,17	0	-	230	0,77	5,77	4,03	353,49
4 - GraspRangel	14	11	1,06	22	23,73	247	11,39	113,85	17,50	202,06

Tabela 4.41: Comparação 3 - Classe 1 - 47 instâncias

Classe - 1	Ordem de desempenho dos Algortimos - Outras Metaheurísticas				
(47 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	Pontuação
1 - VnsBasico	62,98	23,40	11,06	2,55	4575
2 - TabuTaillard	85,96	8,09	2,13	3,83	**5649**
3 - IlsStutzle	45,11	20,00	26,81	8,09	3493
4 - GraspRangel	35,74	22,55	17,87	23,83	2927

Tabela 4.42: Comparação 3 - Classe 1 - Classificação dos algoritmos

4.5.3.2 Classe 2 - Fluxos Aleatórios em Grades (30 instâncias)

Assim como na classe 1, os algoritmos 1 e 2 permaneceram nas mesmas posições, havendo uma alternancia de posições apenas nos algoritmos 3 e 4. Tal fato, pode ser explicado pelos critérios c e e.

Classe 2	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	20	10	0,00	0	-	121	0,17	1,52	11,67	246,82
2 - TabuTaillard	21	8	0,08	1	15,89	113	0,56	5,65	28,82	169,06
3 - IlsStutzle	18	12	0,55	0	-	153	0,32	2,71	3,82	365,46
4 - GraspRangel	16	13	1,31	1	3,49	152	2,21	2,12	25,08	186,88

Tabela 4.43: Comparação 3 - Classe 2 - 30 instâncias

Classe - 2	Ordem de desempenho dos Algortimos - Outras Metaheurísticas				
(30 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	Pontuação
1 - VnsBasico	58,00	28,67	12,00	1,33	2792
2 - TabuTaillard	85,33	8,67	3,33	2,67	**3592**
3 - IlsStutzle	46,00	11,33	32,00	10,67	2176
4 - GraspRangel	54,00	10,00	8,67	27,33	2402

Tabela 4.44: Comparação 3 - Classe 2 - Classificação dos algoritmos

4.5.3.3 Classe 3 - Problemas da Vida Real (38 instâncias)

O algoritmo 4 (*GraspRangel*) ficou muito distante dos demais algoritmos, (4.45), no alcance do $VOMVC$. Como o complemento do tempo médio de estagnação foi próxima do critério de parada, estima-se que, para estas instâncias, o algoritmo seja de convergência mais lenta.

Classe 3	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	38	0	-	0	-	17	0,21	0,89	7,34	74,14
2 - TabuTaillard	38	0	-	0	-	0	-	0,10	2,13	-
3 - IlsStutzle	38	0	-	0	-	39	0,36	2,00	1,18	124,29
4 - GraspRangel	26	5	0,51	7	-	65	1,23	11,22	27,51	72,24

Tabela 4.45: Comparação 3 - Classe 3 - 38 instâncias

Um fato interessante foi a facilidade com que o algoritmo 2 (*TabuTaillard*) conseguiu resolver todas as instâncias desta classe, (4.45), ratificado pela alta porcentagem

obtida na primeira posição da Tabela 4.46.

Classe - 3	Ordem de desempenho dos Algortimos - Outras Metaheurísticas				
(38 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	Pontuação
1 - VnsBasico	82,11	10,00	4,21	3,68	4414
2 - TabuTaillard	91,58	3,16	3,16	2,11	**4774**
3 - IlsStutzle	74,74	8,42	13,16	3,68	4060
4 - GraspRangel	54,74	23,68	9,47	12,11	3290

Tabela 4.46: Comparação 3 - Classe 3 - Classificação dos algoritmos

4.5.3.4 Classe 4 - Instâncias Aleatórias Semelhantes aos Problemas da Vida Real (16 instâncias)

Nesta classe, fica nítida a robustez do algoritmo 2 (*TabuTaillard*), já que foi o único algoritmo que alcançou o $VOMVC$ em todas as instâncias, (Tabela 4.47). Para os demais, houve uma grande diferença entre eles.

Classe 4	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	10	5	0,66	1	19,97	92	6,13	43,09	20,98	326,48
2 - TabuTaillard	16	0	-	0	-	17	0,07	0,74	49,84	60,02
3 - IlsStutzle	7	6	0,73	3	19,40	111	7,21	57,98	5,27	465,17
4 - GraspRangel	1	10	0,63	5	17,06	157	5,73	57,28	7,12	276,66

Tabela 4.47: Comparação 3 - Classe 4 - 16 instâncias

Mesmo que o algoritmo 3 tenha alcançado resultados melhores que o algoritmo 4, sua estagnação média (critério e) mostrou-se precoce, (Tabela 4.47).

Como verificado nas classes anterioes, o algoritmo 2 manteve-se na primeira colocação, sempre acima dos 80%, (Tabela 4.48).

Classe - 4	Ordem de desempenho dos Algortimos - Outras Metaheurísticas				
16 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	Pontuação
1 - VnsBasico	31,25	45,00	21,25	2,50	1052
2 - TabuTaillard	83,75	12,50	1,25	2,50	**1904**
3 - IlsStutzle	26,25	12,50	40,00	21,25	770
4 - GraspRangel	18,75	41,25	11,25	28,75	752

Tabela 4.48: Comparação 3 - Classe 4 - Classificação dos algoritmos

4.5.3.5 Classe 5 - Instâncias Difíceis para as Metaheurísticas (37 instâncias)

Como era de se esperar, por ser uma classe muito difícil para as metaheurísticas, nota-se uma pequena variação no valor do $VOMVC$ entre os três primeiros algoritmos, (Tabela 4.49) e, consequentemente, uma diminuição no percentual da primeira posição do algoritmo 2, (Tabela 4.50).

Classe 5	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	18	8	0,23	11	105,72	279	119,95	853,96	18,92	420,36
2 - TabuTaillard	19	8	0,23	10	41,52	201	12,08	116,46	54,38	255,40
3 - IlsStutzle	17	8	0,24	12	126,49	290	116,03	869,52	5,16	512,22
4 - GraspRangel	4	9	0,37	24	154,46	335	100,32	933,27	14,90	315,03

Tabela 4.49: Comparação 3 - Classe 5 - 37 instâncias

Fato que chama a atenção para o algoritmo 2 (*TabuTaillard*), na Tabela 4.50, é o erro médio acima dos 2% que, nesse caso ficou muito abaixo dos demais.

Classe - 5	Ordem de desempenho dos Algortimos - Outras Metaheurísticas				
(437 instâncias)	1o. (%)	2o. (%)	3o. (%)	4o. (%)	Pontuação
1 - VnsBasico	46,49	29,19	16,22	8,11	2913
2 - TabuTaillard	72,43	13,51	8,11	5,95	**3899**
3 - IlsStutzle	29,19	25,41	30,27	15,14	2077
4 - GraspRangel	29,19	26,49	18,38	25,95	2049

Tabela 4.50: Comparação 3 - Classe 5 - Classificação dos algoritmos

4.5.3.6 Todas as Classes - Comparação 3

O rendimento do algoritmo 2 (*TabuTaillard*) foi superior aos demais. Vale a pena ressaltar o baixo percentual de afastamento médio, (Tabela 4.51) que, consequentemente, refletiu no indice de qualidade (critério c).

Comparacao3	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - VnsBasico	119	37	0,18	12	25,14	691	25,37	180,59	15,61	256,91
2 - TabuTaillard	130	26	0,18	12	10,56	472	2,61	25,26	31,89	115,13
3 - IlsStutzle	112	41	0,54	15	29,18	823	24,94	187,60	3,89	364,12
4 - GraspRangel	61	48	0,78	59	39,75	956	24,18	241,50	18,42	210,57

Tabela 4.51: Comparação 3 - Todas as instâncias

Como feito na comparações anteriores, criamos a Tabela 4.52 a partir das Tabelas 4.42, 4.44, 4.46, 4.48 e 4.50. Mesmo que ainda não ordenada, é fácil perceber um desempenho constante dos algoritmos 2 (*TabuTaillard*) e 3 (*VnsBasico*).

Ordem de desempenho dos Algortimos - Outras Metaheurísticas				
Comparacao3	1o.	2o.	3o.	4o.
Classe - 1	1 - 4575	2 - 5649	3 - 3493	4 - 2927
Classe - 2	1 - 2792	2 - 3592	3 - 2176	4 - 2402
Classe - 3	1 - 4414	2 - 4774	3 - 4060	4 - 3290
Classe - 4	1 - 1052	2 - 1904	3 - 770	4 - 752
Classe - 5	1 - 2913	2 - 3899	3 - 2077	4 - 2049

Tabela 4.52: Comparação 3 - Classificação geral dos algoritmos

Após a aplicação da função $OrdenaValor()$ em 4.52, obteve-se, sem grande modificações, a Tabela 4.53:

Ordem de desempenho dos Algortimos - Outras Metaheurísticas				
Comparacao3	1o.	2o.	3o.	4o.
Classe - 1	2 - 5649	1 - 4575	3 - 3493	4 - 2927
Classe - 2	2 - 3592	1 - 2792	4 - 2402	3 - 2176
Classe - 3	2 - 4774	1 - 4414	3 - 4060	4 - 3290
Classe - 4	2 - 1904	1 - 1052	3 - 770	4 - 752
Classe - 5	2 - 3899	1 - 2913	3 - 2077	4 - 2049

Tabela 4.53: Comparação 3 - Classificação geral dos algoritmos

Feito isto, cada algoritmo da comparação atual foi totalizdo em cada possível ordem, gerando a Tabela 4.54:

Desempenho Geral dos Algortimos - Outras Metaheurísticas				
Comparacao3	1o.	2o.	3o.	4o.
1 - VnsBasico		5		
2 - TabuTaillard	5			
3 - IlsStutzle			4	1
4 - GraspRangel			1	4

Tabela 4.54: Comparação 3 - Classificação geral por Algoritmos

De posse da Tabela 4.54 aplicamos mais uma vez a função (4.3), obtendo a Tabela 4.55 a seguir, cujo resultado apenas reafima a robustez do *TabuTaillard*, (algoritmo 2).

Todas Classes	Ordem de desempenho dos Algortimos - Outras Metaheurísticas				
Algoritmos	1o. (%)	2o. (%)	3o. (%)	4o. (%)	Pontuação
1 - VnsBasico	-	100,00	-	-	45
2 - TabuTaillard	100,00	-	-	-	**135**
3 - IlsStutzle	-	-	80,00	20,00	13
4 - GraspRangel	-	-	20,00	80,00	7

Tabela 4.55: Comparação 3 - Classificação final dos algoritmos

Esse resultado sugere algumas considerações sobre as metaheurísticas utilizadas neste trabalho:

- O algoritmo 2 (*TabuTaillard*) foi utilizado nesse trabalho como *benchmark*, o que o habilita diretamente para a *Comparacao4*. Nossa intenção não foi obter um algoritmo que apresentasse um desempenho melhor que ele, mas que seus resultados servissem de parâmetro para as nossas comparações.

- Por ter ficado sempre em segundo lugar, o algoritmo *VnsBasico* mostrou um comportamento pouco dependente da classe de instâncias do PQA, isto é, em relação aos algoritmos *IlsStutzle* e *GraspRangel*.

- A estrutura de vizinhanca do ILS que, nesse caso, foi a mesma que utilizamos na vizinhança 3 do VNS, isoladamente, pode não ter sido suficiente para alcançar melhores resultados.

4.5.4 Testes de desempenho final

Para este comparativo, escolhemos os algoritmos que obtiveram as melhores pontuações nas Tabelas 4.25, 4.40, 4.55. São eles: o algoritmo *AgitacaoMov* (*Comparacao1*), *DuploVerVert* (*Comparacao2*) e *TabuTaillard* (*Comparacao3*).

4.5.4.1 Classe 1 - Instâncias Aleatórias com Distâncias e Fluxos Uniformemente Distribuídos (47 instâncias)

Mesmo que nenhum dos algoritmos tenha alcançado o $VOMVC$ para todas as instâncias, o algoritmo *TabuTaillard* teve um melhor desempenho que os outros dois, (Tabela 4.56).

Fato interessante ocorreu com o algoritmo *AgitacaoMov*, que apresentou um afastamento médio muito inferior aos demais, porém estagnou mais cedo.

Classe 1	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - AgitacaoMov	34	13	0,01	0	-	181	0,18	1,60	32,40	220,24
2 - DuploVertVert	33	14	0,95	0	-	174	0,44	3,54	18,51	218,04
3 - TabuTaillard	36	10	0,58	1	9,51	141	0,33	3,37	24,29	91,17

Tabela 4.56: Comparação 4 - Classe 1 - 47 instâncias

A Tabela 4.57 mostra que houve pouca diferença entre as versões com memória do VNS e o que foi determinante para a melhor colocação do *AgitacaoMov* foi o seu afastamento médio, como já citado.

Classe - 1	Desempenho dos Algortimos - Final			
(47 instâncias)	1o. (%)	2o. (%)	3o. (%)	Pontuação
1 - AgitacaoMov	59,15	24,26	16,60	1461
2 - DuploVertVert	60,85	25,53	13,62	1499
3 - TabuTaillard	89,36	7,23	3,40	**1949**

Tabela 4.57: Comparação 4 - Classe 1 - Classificação dos algoritmos

4.5.4.2 Classe 2 - Fluxos Aleatórios em Grades (30 instâncias)

Ambas as versões com controle de memória ficaram sempre com o erro médio abaixo dos 2%. Já o algoritmo *TabuTaillard*, obteve para uma única instância ($sko90$) um erro de $15,89\%$, (Tabela 4.58).

Nesta classe, a versão de controle único de memória (*AgitacaoMov*) começa a apresentar diferenças mais significativas em relação a versão com controle duplo (*DuploVertVert*), o que pode ser observado nos percetuais da Tabela 4.59.

Classe 2	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - AgitacaoMov	20	10	0,00	0	-	125	0,18	1,60	32,40	220,24
2 - DuploVertVert	20	10	0,41	0	-	129	0,18	1,73	12,72	227,70
3 - TabuTaillard	21	8	0,08	1	15,89	113	0,56	5,65	28,82	169,06

Tabela 4.58: Comparação 4 - Classe 2 - 30 instâncias

Classe - 2	Desempenho dos Algortimos - Final			
(30 instâncias)	1o. (%)	2o. (%)	3o. (%)	Pontuação
1 - AgitacaoMov	61,33	23,33	15,33	956
2 - DuploVertVert	60,00	17,33	22,67	922
3 - TabuTaillard	86,67	8,67	4,67	**1216**

Tabela 4.59: Comparação 4 - Classe 2 - Classificação dos algoritmos

4.5.4.3 Classe 3 - Problemas da Vida Real (38 instâncias)

Mesmo que todos os algoritmos tenham alcançado o $VOMVC$ com um desempenho satisfatório, (Tabela 4.61), o algoritmo *TabuTaillard* se mostrou mais robusto, principalmente pelo reduzido tempo médio de execução, (Tabela 4.60).

Classe 3	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - AgitacaoMov	38	0	-	0	-	7	0,09	0,28	16,73	43,68
2 - DuploVertVert	38	0	-	0	-	12	0,21	0,53	10,35	71,55
3 - TabuTaillard	38	0	-	0	-	0	-	0,10	2,13	-

Tabela 4.60: Comparação 4 - Classe 3 - 38 instâncias

Classe - 3	Desempenho dos Algortimos - Final			
(38 instâncias)	1o. (%)	2o. (%)	3o. (%)	Pontuação
1 - AgitacaoMov	85,79	7,89	6,32	1524
2 - DuploVertVert	81,05	16,32	2,63	1484
3 - TabuTaillard	92,63	1,58	5,79	**1604**

Tabela 4.61: Comparação 4 - Classe 3 - Classificação dos algoritmos

4.5.4.4 Classe 4 - Instâncias Aleatórias Semelhantes aos Problemas da Vida Real (16 instâncias)

Somente o algoritmo *TabuTaillard* conseguiu alcançar o $VOMVC$ em todas as instâncias, mesmo que tenha gasto um tempo médio de execução 3 vezes maior, aproximandamente, que as versões com controle de memória, (Tabela 4.62).

Esta classe mostrou dois grupos muito nítidos de desempenho, o que pode ser observado na Tabela 4.63 a seguir, com diferenças pouco significativas entre as versões com controle de memória, onde a distribuição entre as ordens ficou mais homogênea,

Classe 4	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - AgitacaoMov	9	4	0,61	3	19,10	103	7,09	55,56	15,87	331,77
2 - DuploVertVert	11	4	0,61	1	19,67	93	7,01	47,08	17,33	332,83
3 - TabuTaillard	16	0	-	0	-	17	0,07	0,74	49,84	60,02

Tabela 4.62: Comparação 4 - Classe 4 - 16 instâncias

o que não ocorreu com o algoritmo *TabuTaillard*, cujo percentual decresce de acordo com a ordem, apresentando uma grande variação entre o primeiro e terceiro lugar.

Classe - 4	Desempenho dos Algortimos - Final			
(16 instâncias)	1o. (%)	2o. (%)	3o. (%)	Pontuação
1 - AgitacaoMov	31,25	37,50	31,25	340
2 - DuploVertVert	28,75	42,50	28,75	332
3 - TabuTaillard	83,75	12,50	3,75	**636**

Tabela 4.63: Comparação 4 - Classe 4 - Classificação dos algoritmos

4.5.4.5 Classe 5 - Instâncias Difíceis para as Metaheurísticas (37 instâncias)

Para esta classe, o algoritmo (*DuploVertVert*) gastou pouco tempo de execução, no entanto seu erro médio acima de 2% superou em três vezes do algoritmo (Tabu-Taillard) que teve o maior tempo médio de execução nesta classe.

Dentre todas as classes esta foi a única vez que o algoritmo *TabuTaillard* alcançou o $VOMVC$ em menos instâncias que um concorrente seu, (Tabela 4.64). Mesmo assim, ele ficou com melhor pontuação, (Tabela 4.65), devido ao baixo índice de afastamento médio (critério b), em torno de 12%, enquanto os demais ficaram acima dos 120%. Isto afetou diretamente o índice de qualidade (critério c).

Classe 5	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Versão do Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - AgitacaoMov	21	8	0,23	8	88,97	255	126,51	832,46	31,07	412,89
2 - DuploVertVert	17	8	0,23	12	123,56	268	121,18	834,30	10,64	385,72
3 - TabuTaillard	19	8	0,23	10	41,52	201	12,08	116,46	54,38	255,40

Tabela 4.64: Comparação 4 - Classe 5 - 37 instâncias

Classe - 5	Desempenho dos Algortimos - Final			
(37 instâncias)	1o. (%)	2o. (%)	3o. (%)	Pontuação
1 - AgitacaoMov	39,46	45,41	15,14	937
2 - DuploVertVert	46,49	35,68	17,84	1005
3 - TabuTaillard	73,51	5,95	20,54	**1295**

Tabela 4.65: Comparação 4 - Classe 5 - Classificação dos algoritmos

4.5.4.6 Todas as Classes - Comparação 4

A Tabela 4.66 mostra uma certa vantagem para o algoritmo com controle único de memória (*AgitacaoMov*) em relação ao controle duplo de memória (*DuploVertVert*).

Comparacao4	Alcanc	ErroMed(Até 2%)		ErroMed(Acima 2%)		10-VOMVC	(afast,%)	I(nsol,erro)	(t,exec)	(t,estag)
Algoritmo	Qtd	Qtd	Perc (%)	Qtd	Perc (%)	Total	Média (%)	Média (%)	Média (%)	Média (%)
1 - AgitacaoMov	122	35	0,17	11	21,61	671	26,81	178,30	25,69	245,76
2 - DuploVertVert	119	36	0,44	13	28,65	676	25,81	177,44	13,91	247,17
3 - TabuTaillard	130	26	0,18	12	10,56	472	2,61	25,26	31,89	115,13

Tabela 4.66: Comparação 4 - Todas as instâncias

A partir da pontuação final de cada algoritmo em cada classe, isto é, fazendo uso das Tabelas 4.57, 4.59, 4.61, 4.63 e 4.65, construímos a Tabela 4.67.

Desempenho Geral dos Algortimos - Final			
Comparacao4	1o.	2o.	3o.
Classe - 1	1 - 1461	2 - 1499	3 - 1949
Classe - 2	1 - 956	2 - 922	3 - 1216
Classe - 3	1 - 1524	2 - 1484	3 - 1604
Classe - 4	1 - 340	2 - 332	3 - 1636
Classe - 5	1 - 937	2 - 1005	3 - 1295

Tabela 4.67: Comparação 4 - Classificação geral dos algoritmos

Em seguida, aplicamos a função $OrdenaValor()$ nesta tabela, obtendo a Tabela 4.68 a seguir:

Desempenho Geral dos Algortimos - Final			
Comparacao4	1o.	2o.	3o.
Classe - 1	3 - 1949	2 - 1499	1 - 1461
Classe - 2	3 - 1216	1 - 956	2 - 922
Classe - 3	3 - 1604	1 - 1524	2 - 1484
Classe - 4	3 - 1636	1 - 340	2 - 332
Classe - 5	3 - 1295	2 - 1005	1 - 937

Tabela 4.68: Comparação 4 - Classificação geral dos algoritmos

Cada um dos algoritmos pertencentes a esta *Comparacao4* foi totalizado pela sua posição (ordem) em 4.68, gerando a Tabela 4.69.

Desempenho Geral dos Algortimos - Final			
Comparacao4	1o.	2o.	3o.
1 - AgitacaoMov		3	2
2 - DuploVertVert		2	3
3 - TabuTaillard	5		

Tabela 4.69: Comparação 4 - Classificação geral por Algoritmos

A partir desses valores, usamos a função (4.3), obtendo a Tabela 4.55, que representa a pontuação final entre os melhores algoritmos das comparações anteriores.

Todas Classes	Desempenho dos Algortimos - Final			
Algoritmos	1o. (%)	2o. (%)	3o. (%)	Pontuação
1 - AgitacaoMov	-	60,00	40,00	11
2 - DuploVertVert	-	40,00	60,00	9
3 - TabuTaillard	100,00	-	-	45

Tabela 4.70: Comparação 4 - Classificação final dos algoritmos

Esse resultado sugere algumas considerações sobre o uso de memória no VNS:

- Inicialmente, nosso sentimento era que um controle duplo fosse, na média, mais vantajoso que um controle único, porém isso não se verificou sempre.
- O baixo tempo médio de processamento (critério d) do algoritmo *DuploVertVert*, menor inclusive que o da versão básica, (ver Tabela 4.36), indica que o controle isolado de memória por vértice, na média, não se mostrou tão robusto quanto o controle por movimento, que apresentou resultados médios mais consistentes. Porém, quando esse controle por vértice é duplo, seu uso torna-se convidativo se o interesse maior for o tempo médio de processamento.
- Caso o interesse seja obter resultados com menor erro médio e maior número de instâncias a alcançar o $VOMVC$, o algoritmo com controle único de memória *AgitacaoMov* se mostra mais interessante.
- Independentemente da motivação na escolha entre as duas versões com controle de memória, que obtiveram os melhores resultados médios entre todas as versões propostas neste trabalho, fica a certeza de que o seu uso amplia as possibilidades do VNS básico.

Capítulo 5

Aplicação das variâncias do PQA ao Problema de Isomorfismo de Grafos

O Problema de Isomorfismo em Grafos (PIG) se apresenta, habitualmente, sob duas formas diferentes, em vista do seu interesse teórico e das suas possíveis aplicações, tais como o reconhecimento de padrões, (96), (97). Os conceitos grafo-teóricos utilizados neste trabalho podem ser encontrados em Whitney (110) e Harary (111).

Vamos considerar dois grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ com os conjuntos de vértices rotulados de maneira independente. Dizemos que G_1 e G_2 são *isomorfos* se e somente se existe uma bijeção $\varphi : V_1 \longleftrightarrow V_2$ que preserve suas relações de adjacência. O problema mais geral aparece quando G_1 é um grafo e G_2 é um subgrafo de outro grafo, que seja pelo menos de mesma ordem e tamanho (número de arestas) de G_1. Um caso particular, que é estudado neste trabalho, é o do isomorfismo entre dois grafos de mesma ordem e tamanho. Boa parte dos pesquisadores acredita que o PIG é NP, porém até a presente data ainda não se pode afirmar se o problema é polinomial ou NP-completo, Garey e Johnson (112), Arvind e Thorán (113).

Dizemos que um parâmetro de um grafo é um *invariante* se ele possui o mesmo valor para todos os isomorfos desse grafo. Para grafos isomorfos, os invariantes mais prontamente disponíveis para considerar são a ordem n e o tamanho m, mas é evidente que o nosso interesse é encontrar um invariante cujo valor seja sempre diferente para dois grafos não isomorfos, envolvendo uma condição necessária e suficiente - o que não é, exatamente, o caso da ordem e do tamanho. Um invariante importante a considerar é a *sequência ordenada de graus*(SOG) associada a um grafo: porém, uma vez mais, dois grafos com a mesma SOG podem ser não isomorfos. De outro lado, dois grafos com diferentes SOG **são** não-isomorfos - e a SOG é fácil de calcular através de um algoritmo de ordenação eficiente como, por exemplo, o *Merge Sort*,

cuja complexidade é *O(n log n)*, Cormen *et al*, (1).

Um número significativo de recursos teóricos e computacionais tem sido aplicado ao estudo de isomorfismo em grafos. McKay (114) propôs um algoritmo específico; Bunke e Sharer (96) contribuiram com um artigo teórico definindo uma distância (*ad hoc*) entre dois grafos; Cross *et al* (115) usaram uma metaheurística e Depiero e Krout (97) usaram contagem de caminhos para aproximar subgrafos isomorfos; Ding e Huang (116) reorganizaram o grafo numa busca por um perímetro e uma matriz de adjacência canônica.

Utiliza-se uma instância construida com as matrizes de adjacência dos grafos em exame, procurando-se desta vez o máximo da função objetivo do PQA (5.1) encontramos esta função voltada para a minimização):

$$\underset{\varphi \in \Pi_n}{max} \sum_{i,j=1}^{n} f_{ij} d_{\varphi(i)\varphi(j)} \tag{5.1}$$

Este trabalho procura investigar o caso particular do PIG através da estrutura do PQA. Como visto no Capitulo 1, a base de dados utilizadas para as instâncias PQA é um par de matrizes (simétricas) F e D. Uma instância PQA pode ser construída com matrizes de adjacência de dois grafos simples de igual ordem e tamanho e, se procurarmos pelo solução de maior valor, iremos encontrar um valor $z^* = m$ se os grafos são isomorfos, ou $z^* < m$ se eles não o são. A existência de $z^* = m$ é uma condição necessária e suficiente para o isomorfismo, mas esse recurso subordina a solução do PIG à de um problema de complexidade sabidamente elevada.

Apesar do PQA ser NP-difícil, o cálculo da variância das instâncias do PQA é polinomial, Boaventura-Netto e Abreu (117), Abreu *et al* (4). Dado um par de grafos (G, H), podemos construir três instâncias, (G, H), (G, G) e (H, H) e calcular as três variâncias correspondentes. É evidente que, se G e H forem isomorfos, essas três variâncias serão iguais. Esta igualdade, no entanto, não é suficiente para caracterizar o isomorfsmo: é fácil encontrar contra-exemplos, tais como o par (grafo de Petersen, prisma pentagonal), da Figura 5.1:

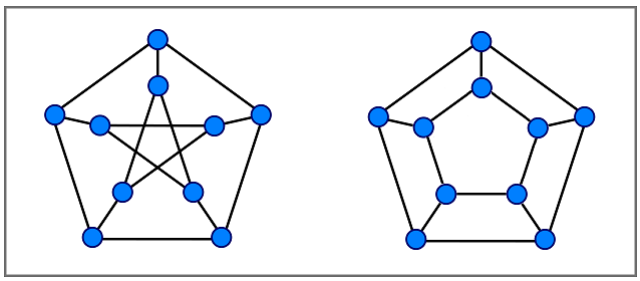

Figura 5.1: Grafo de Petersen e prisma pentagonal

Para superar esta dificuldade, definimos *funções de valor* para os conjuntos de

arestas do par de grafos em exame (a própria matriz de adjacência já define uma função de valor para as arestas mas, como acabamos de ver, é fácil encontrar exemplos nos quais ela não permite uma discriminação entre dois grafos não isomorfos). Estas funções, para esse fim, devem ser baseadas em propriedades da estrutura dos grafos (não relacionadas à rotulação dos vértices), o que vai garantir sua invariância em relação ao isomorfismo. Uma coloração dos vértices, por exemplo, não fornece uma função adequada, a menos que o grafo seja unicamente colorível, visto que a coloração obtida depende da indexação dos vértices. Logo, mesmo com dois grafos isomorfos haveria a possibilidade da obtenção de funções diferentes.

É de se esperar, embora não se tenha como apresentar uma prova, que uma função adequadamente definida permita que se encontrem diferenças entre dois grafos não isomorfos quaisquer: no exemplo, a função baseada na matriz de adjacência é invariante mas não permite essa discriminação dentro do esquema de cálculo citado anteriormente.

Este trabalho, portanto, é baseado em uma conjetura, que é a da existência, para todo par de grafos, de uma função de valor que permita defini-los como não isomorfos se esse for o caso. Por outro lado, esta técnica não permite afirmar diretamente o isomorfismo entre dois grafos, essa conclusão devendo ser atingida por complementaridade, ou seja, sujeita a erro, quando uma discriminação não for obtida através da aplicação de um conjunto adequado de funções de valor.

Os resultados mostrados nesta tese relacionam-se a grafos planares de até 3000 vértices. Duas funções de valor são aqui utilizadas:

- A primeira técnica é baseada na soma dos graus dos vértices que definem cada aresta. É de baixa complexidade, porém não de uso geral (não poderia ser aplicada ao par da Figura 5.1, que é de grafos regulares);
- A segunda técnica é de uso mais generalizado (podendo inclusive ser utilizada com grafos regulares) e se refere à contagem de percursos fechados entre dois vértices, que contenham uma dada aresta.

5.1 Base teórica

Como também observado no Capítulo 1, o PQA é composto por uma matriz de fluxos (F) e outra de distâncias (D). Para o estudo da variância no PIG consideraremos, como de hábito, que F e D são matrizes simétricas e iremos referenciar uma instância qualquer como $PQA(\boldsymbol{F,D})$.

5.1.1 Momentos Estatísticos das Instâncias PQA

Momentos estatísticos das instâncias PQA têm sido usados para comparar a dificuldade computacional com respeito aos algoritmos heurísticos. Esta questão tem sido estudada por Graves e Whinston (118), Herroeleven e Van Gils (119), Mautor e Roucairol (59) e Abreu *et al* (22).

Como já visto na Seção 1.2, para o PQA simétrico, podemos guardar todos os dados de uma instância com o auxílio de dois vetores F e D, de ordem $N = n(n-1)/2$, onde a posição ψ pode ser associada a posição (i, j) através da expressão 1.18, encontrada na página 6.

Podemos definir uma matriz de ordem N, $Q = FD^t$. Esta matriz contém cada parcela que compõe cada solução φ (permutação ordem n) do PQA. A função 1.20, encontrada na página 7, nos mostra que dados i e j $(i < j)$ com valores entre 1 e n, cada solução do PQA possui N parcelas, as quais cada solução correspondendo a um conjunto de posições independentes em Q. Para $n > 3$, o conjunto das $n!$ soluções de uma instância está estritamente contido no conjunto dessas $N!$ posições. Um problema baseado em Q é então uma *relaxação* do PQA original, onde nem todo conjunto de posições independentes de Q corresponde a uma solução viável do PQA. Iremos chamar tal relaxação de $PQA(\mathbf{Q})$.

É interessante observar que, o problema relaxado é fácilmente resolvido: basta ordenar F e D em ordens opostas ,como citado no exemplo da seção 1.2, página 3, obtendo-se por exemplo os vetores F^+ e D^- e. Pode-se então calcular a matriz $Q' = F^+(D^-)^t$, para uma dada instância do PQA (chamaremos à instância assim ordenada PQA(F^+,D^-)). A soma dos elementos da diagonal principal (o traço da matriz) constitui um limite inferior e a dos elementos da maior diagonal oposta, um limite superior para o conjunto de soluções (viáveis ou não) do PQA.

A média dos custos das soluções de uma instância $PQA(\boldsymbol{F,D})$, (118), (117), (120), (22), é:

$$\mu = S/N, \tag{5.2}$$

onde S é a soma de todos os elementos de Q.

A variância da instância relaxada $PQA(Q)$ é, (118):

$$\sigma_Q^2 = \frac{1}{N}(\sum_{i=1}^{N} f_i^2)(\sum_{i=1}^{N} d_j^2) + \frac{1}{N(N-1)}[\sigma^2(f_i)][\sigma^2(d_j)] - \mu^2 \quad (i, j = 1, ..., n) \tag{5.3}$$

A variância da instância $PQA(\boldsymbol{F,D})$, (117), (22), é:

$$\sigma_{F,D}^2 = (S_0 + S_1 + S_2)/n! - \mu^2, \tag{5.4}$$

onde

$$S_0 = 4(n-4)! \sum_{\cap_0} f_{ij} f_{rs} \sum_{\cap_0} d_{ij} d_{rs} \tag{5.5}$$

$$S_1 = (n-3)! \sum_{\cap_1} f_{ij} f_{rs} \sum_{\cap_1} d_{ij} d_{rs} \tag{5.6}$$

$$S_2 = 2(n-2)! \sum_{1 \leq i < j \leq n} f_{ij}^2 \sum_{1 \leq i < j \leq n} d_{ij}^2 \tag{5.7}$$

onde

$$| \cap_0 | = C_{n-2,2}, | \cap_1 | = 2(n-2), | \cap_2 | = 1, \tag{5.8}$$

Desse modo, a complexidade computacional da primeira variância é $O(n^2)$, e da segunda é $O(n^4)$ onde, em ambos os casos, consideramos F e D como sendo matrizes completas. Se trabalharmos com listas de adjacência poderemos escrever respectivamente $O(m)$ e $O(m^2)$ e, para grafos planares, $O(n)$ e $O(n^2)$, dado seu tamanho linear em relação à ordem.

5.1.2 Classes de instâncias relacionadas ao isomorfismo

Podemos considerar uma dada matriz Q' e escolher um $PQA(F^+, D^-)$ que possua Q como matriz de coeficientes e também uma classe de instâncias que possuam um mesmo Q' (sendo associadas assim com a mesma instância relaxada) e o mesmos limites inferior e superior baseados em Q' (sendo as matrizes Q e Q' como em 1.2). Então, se define, (22):

$$RelClass(F^-, D^+) = PQA(F,D)/(F^+)(D^-)^T = Q' \tag{5.9}$$

Uma instância PQA pode ser representada através de dois grafos completos K_F e K_D, respectivamente valorados pelas arestas através das funções de valor $w(K_F) = F$ e $w(K_D) = D$. Com isso, podemos dizer que dois grafos completos $K_n(V_1, E_1, F)$ e $K_n(V_2, E_2, D)$ são *w-isomorfos* se existir uma permutação $\alpha \in \Pi_n$, tal que, $(i,j) \in E_1 \iff (\alpha(i), \alpha(j)) \in E_2$ e $F(i,j) = D(\alpha(i), \alpha(j))$. Então, denotaremos esse isomorfismo como $w(K_F) \approx w(K_D)$. $PQA(F_1, D_1)$ e $PQA(F_2, D_2)$ serão isomorfos ($PQA(F_1, D_1) \approx PQA(F_2, D_2)$) se e somente se $w(K_{F_1}) \approx w(K_{F_2})$ e $w(K_{D_1}) \approx w(K_{D_2})$.

De outro lado, dizemos que $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ são isomorfos, se e somente se, existir uma função f que preserve as relações de adjacência sobre a permutação α de um dos seus conjuntos de vértices sobre o outro onde, para cada $(i,j) \in E1$, temos $f(\alpha(i), \alpha(j)) \in E_2$.

Um caso particular do que foi discutido é a função de valor dada pela matriz de adjacência para um grafo $G = (V, E), A = [a_{ij}]$, onde $a_{ij} = 1$ se $(i, j) \in E$ e $a_{ij} = 0$, caso contrário. Uma condição primária necessária para o isomorfismo entre dois grafos é que ambos possuam mesma ordem n e tamanho m.A definição de instâncias isomorfas, vista anteriormente, implica no seguinte teorema:

Teorema 5.1.1 (ABQG02)**.** Duas instâncias isomorfas $PQA(F_1, D_1)$ e $PQA(F_2, D_2)$ possuem o mesmo conjunto de soluções viáveis. ■

Se duas instâncias forem isomorfas, suas instâncias relaxadas também serão isomorfas, mas a recíproca não é verdadeira.

Corolário 5.1.2. Duas instâncias isomorfas possuem a mesma variância.

Demonstração: Imediata a partir do Teorema 5.1.1, uma vez que elas compartilham o mesmo conjunto de soluções viáveis. ■

A variância das soluções de uma instância PQA é então um *invariante*, no que diz respeito ao isomorfismo. No início desse capítulo apresentou-se um uso prático desta propriedade, porém isso pede uma discussão mais detalhada.

5.2 Metodologia

5.2.1 Comparação padrão e avaliação

Vimos que um meio prático para fazer a comparação entre dois grafos é construir três instâncias: a primeira, $PQA(G_1, G_2)$, composta com matrizes dos dois grafos e outras duas,$PQA(G_1, G_1)$ e $PQA(G_2, G_2)$, usando duas cópias da mesma matriz de cada grafo, o que nos permite realizar uma comparação padrão. Chamamos estas duas últimas instâncias de *instâncias associadas*, e suas variâncias de *variâncias associadas*. Um exame mais próximo das expressões 1.18 e 1.20 mostra que boa parte dos seus termos pode ser reutilizada na determinação dessas variâncias associadas, o que permite significativa economia de cálculo.

Para as instâncias relaxadas, o Corolário 5.1.2 é a única condição necessária: para possuírem a mesma variância, não é necessário que duas instâncias sejam isomorfas. A consequência disso, é que um número de pares de grafos com estruturas similares mostra variâncias iguais para as três instâncias apresentadas. Um exemplo seria o par de grafos presentes na Figura 5.1, página 79.

Tal como feito em Boaventura e Leite, (121), utilizamos funções de valor aplicadas às arestas de cada par de grafos, procurando maior capacidade de discriminação de não-isomorfos através de eventuais diferenças em suas estruturas. Das duas funções citadas acima, a de menor custo é a dada pela soma dos graus dos vértices

que definem cada aresta, $\boldsymbol{gr}(x, y) = d(x) + d(y)$. A segunda função é a mesma já aplicada a grafos regulares em (121), baseada no Teorema 5.2.1:

Teorema 5.2.1 (Festinger,(Bo06))**.** Se $G = (V, E)$ é um grafo, $A = [a_{ij}]$ é sua matriz de adjacência e $A^k = [a_{ij}^{(k)}]$ é a k-ésima potência de A, então o valor de $[a_{ij}^{(k)}]$ é igual ao número de k-caminhos i e $j \in V$. ■

Podemos notar que esses k-caminhos não são necessariamente elementares (isto é, eles podem repetir vértices), porém isso não compromete a utilidade da função para a aplicação que dela fazemos. Cabe ainda observar que, como aplicamos a técnica a pares de vértices que definem arestas, o que realmente estaremos contando será o número de $(k+1)-$percursos fechados que contém cada aresta. Utilizamos as potências 2 e 3, obtendo funções de valor que, face esta observação, denominamos $\boldsymbol{c3}(i, j) = a_{ij}^{(3)}$ e $\boldsymbol{c4}(i, j) = a_{ij}^{(4)}$, onde $a_{ij}^{(3)}$ e $a_{ij}^{(4)}$ são, respectivamente, elementos de A^3 e A^4, para $a_{ij} = 1$.

A função obtida é invariante com respeito ao isomorfismo porque ela depende somente da estrutura do grafo, a qual é a mesma para dois grafos isomorfos.

Como sustentação desta técnica, enunciamos a conjetura acima citada:

Conjectura 5.2.2. Para cada par de grafos (G_1, G_2) de mesma ordem e tamanho, existe uma função de valor invariante tal que, para (G_1 e G_2) não isomorfos, as instâncias $PQA(G_1, G_2)$, $PQA(G_1, G_1)$ e $PQA(G_2, G_2)$ irão possuir diferentes variâncias.

Um ponto interessante a considerar é que, se for possível provar esta conjetura para uma função de valor invariante polinomial aplicável a todo par de grafos, isto será equivalente a estabelecer a complexidade do PIG como sendo polinomial. Este trabalho, no entanto, não se propõe a isso.

5.2.2 Grafos Planares

Segundo Harary (111), Euler se tornou o pai da teoria e topologia dos grafos quando, em 1736, resolveu um famoso problema daquela época, chamado de o Problema das Pontes de Königsberg. Nesta cidade, que hoje chama-se Kaliningrado (Rússia), havia duas ilhas ligadas cada uma a outra e ambas às margens do rio Pregel através de sete pontes, como mostrado na Figura 5.2(a). O problema consiste em sair de uma das quatro áreas de terra, caminhar através de cada ponte, exatamente uma única vez, retornando ao ponto de origem. Alguém poderia facilmente tentar resolver este problema de maneira empírica, mas toda tentativa foi sem sucesso. Através da contribuição de Euler (122), provou-se que isso era impossível.

Para provar que o problema em insolúvel, Euler substituiu cada área de terra por um ponto, cada ponte por uma linha ligando os pontos correspondentes e, através

disso, produziu um "grafo". Isso é mostrado na Figura 5.2(b), onde os pontos são associados as quatro áreas de terra correspondentes. Mostrando que o problema é sem solução, é equivalente a mostrar que o grafo da Figura 5.2(c) não poderia ser atravessado em uma determinada maneira.

Ao invés de tratar uma situação específica, Euler generalizou o problema e desenvolveu um critério para um dado grafo ser "atravessável": o grafo deve ser conectado e cada ponto incidente deve possuir um número par de arestas.

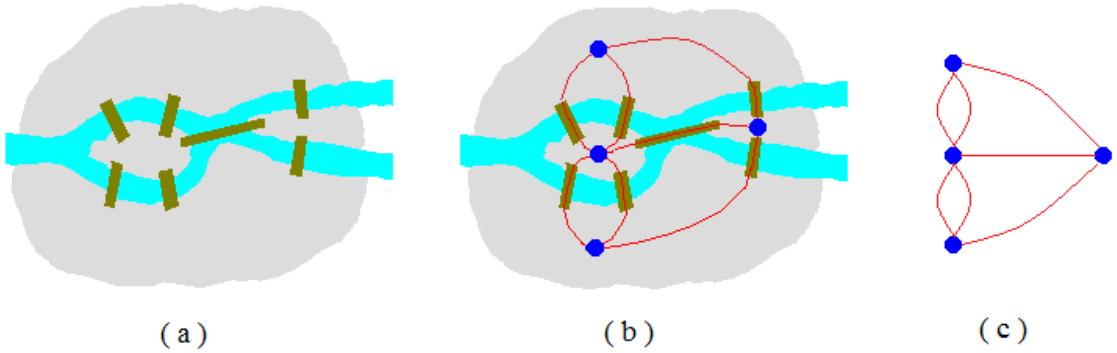

Figura 5.2: (a)Königsberg; (b)convertendo em um grafo; (c)grafo do problema.

A primeira contribuição sobre teoria topológica de grafos foi também de Euler, quando ele encontrou uma forma de relacionar o número dos vértices, arestas e faces de um poliedro. Porém, foi Kuratowski que encontrou um critério para um grafo ser planar. Também segundo Harary (111), outro pioneiro foi Whitney, que desenvolveu algumas importantes propriedades do inserção de grafos no plano.

Em 1988, numa publicação específica sobre grafos planares, Nishizeki e Chiba (123) comentaram que, nos últimos anos, os grafos planares têm atraído o interesse de muito pesquisadores e um número interessante de algoritmos e resultados sobre sua complexidade tem sido obtido.

Um grafo pode ser visto como um diagrama que consiste numa coleção de vértices juntos e arestas ligando certos pares de vértices. Um grafo planar é um diagrama particular que deve poder ser desenhado no plano sem que duas arestas sejam interceptadas geometricamente, exceto num vértice, onde ambas podem ser incidentes.

Considerando o exemplo descrito na Figura 5.4(a), que consiste em um grafo com 6 vértices e 12 arestas. Apesar do aspecto do primeiro esquema ele é planar, porque podemos desenhá-lo no plano localizando os vértices e colocando apropriadamente as arestas de maneira que não haja interseção entre duas delas, exceto nos vértices comuns. Como visto na Figura 5.4(a), o grafo possui duas interseções, que podem ser obeservadas nos dois círculos pontilhados. Contudo, isso pode ser evitado se, por exemplo, o vértice v_6 for deslocado para fora do quadrado $v_1v_2v_3v_4$ e, como mostrado na Figura 5.4(b), for feito o deslocamento das arestas (v_2, v_6) e (v_3, v_6) para fora do

quadrado. Desse modo, podemos afirmar que o grafo é planar.

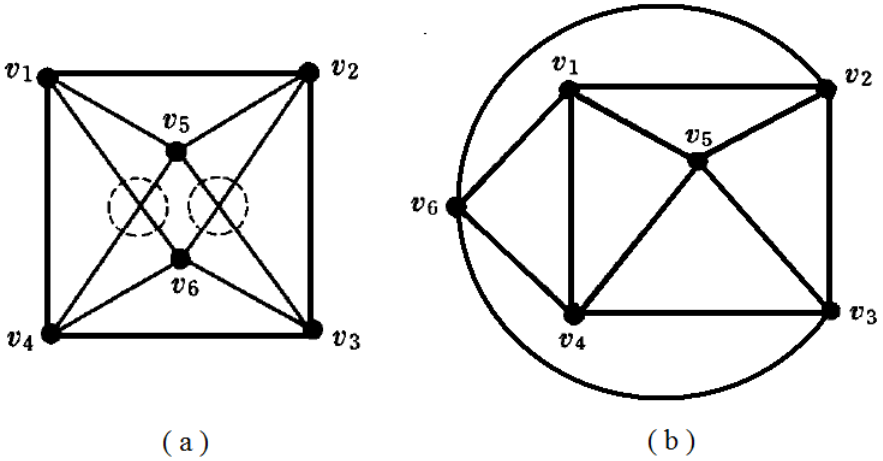

Figura 5.3: (a)um grafo G; (b)um grafo G fixado no plano.

Considerando o grafo da Figura 5.4(a), conhecido como um grafo completo K_5 de 5 vértices. Será ele planar? Se supusermos que sim, estaremos assumindo, sem perda de generalidade, que $v_1v_2v_3v_4v_5$ pode ser desenhado no plano como um pentágono regular. Por exemplo, se assumirmos que a aresta (v_1, v_3) permaneça no interior do pentágono e as arestas (v_2, v_5) e (v_2, v_4) possam ser desenhadas no exterior e, consequentemente, a aresta (v_3, v_5) possa ser desenhada no interior, como mostrada na Figura 5.5(a). Então, uma interseção deve ocorrer se a aresta (v_1, v_4) for desenhada, seja no interior ou no exterior. Com isso, é fácil perceber que K_5 não pode ser desenhado no plano sem que as arestas se sobreponham, o que nos leva a concluir que K_5 não é planar.

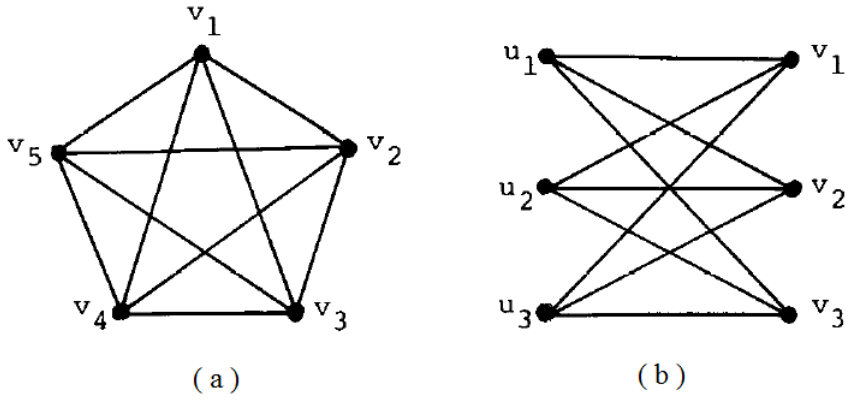

Figura 5.4: Grafos de Kuratowski: (a)grafo completo K_5; (b)grafo completo bipartido $K_{3,3}$.

Outro exemplo de grafo não planar é o grafo completo bipartido $K_{3,3}$, apresentado na Figura 5.4(b). Se assumirmos que a aresta (u_1, v_2) ficará no interior do hexágono $u_1v_1u_2v_2u_3, v_3$ e a aresta (v_1, u_3) no exterior, então, (u_2, v_3) não poderá ser desenhada

sem que produza uma interseção, o que pode ser visto na Figura 5.5(b). Isto nos leva a deduzir que $K_{3,3}$ também é não planar.

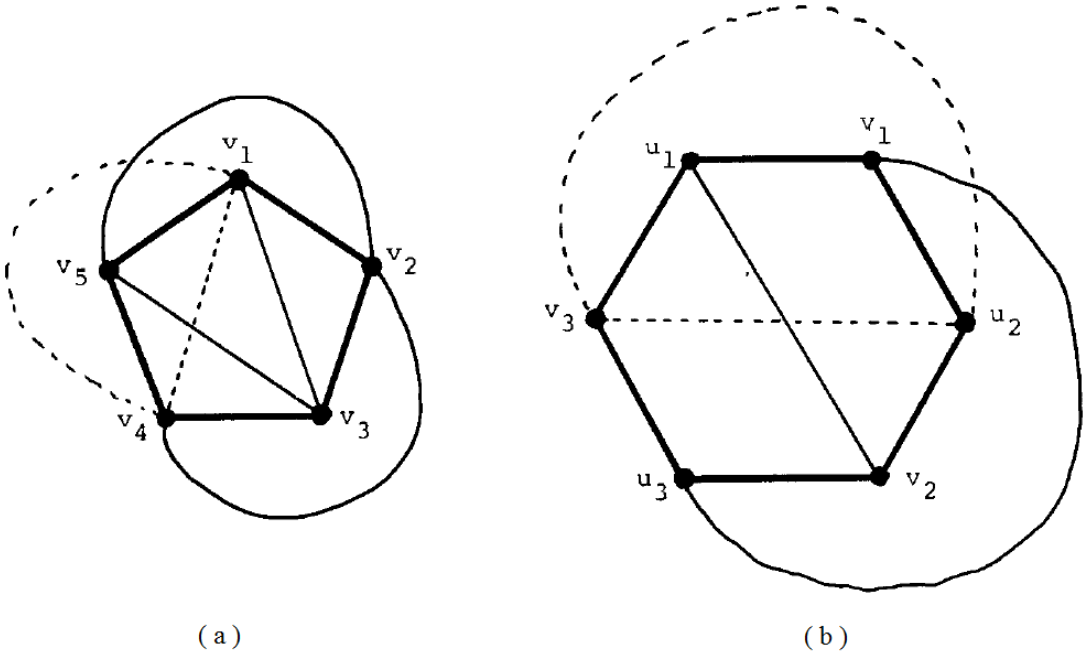

Figura 5.5: (a)grafo K_5 e (b)grafo $K_{3,3}$, parcialmente fixados no plano.

Nosso intuito é tentar encontrar pequenas diferenças em grafos planares o que, no futuro, poderá ser usado em reconhecimento de padrões. Mesmo que existam outras famílias de grafos que poderiam ser usadas com essa finalidade, optamos por trabalhar com as três classes a seguir:

- grafos (p, q) em grade com duas dimensões, com $n = pq$ vértices e $m = 2n - (p + q)$ arestas;

- a classe de grafos com duas dimensões que denominamos grafos (p, q) em grade triangular com janelas, obtida através da adição de uma diagonal em cada quadrado da grade e selecionando-se algumas casas para troca de direção das diagonais, as casas vizinhas não tendo diagonais. Esses grafos possuem $n = pq$ vértices e $m = 3n - 2(p + q) - 7$ arestas, no máximo;

- outra classe denominada roda quebrada de bicicleta (RQB), composta de grafos com um ciclo interior contendo p vértices, onde cada vértice está ligado por q arestas a outros tantos vértices pertencentes a um ciclo exterior por um feixe de arestas. Na ordem de contagem desse último ciclo, existe um vértice antes de cada feixe que não é ligado ao ciclo interior; por isso dizemos que a roda é "*quebrada*". Um $(p, q)-$RQB possui $p(q - 2)$ vértices e $2p(d + 1)$ arestas. A construção é baseada nos rótulos iniciando no ciclo interno e que continuam no ciclo externo.

Foram feitas trocas aleatórias de amostras em exemplos dessas classes de grafos, de modo a obter grafos com estruturas bastante próximas para serem utilizados nos testes. Os grafos quase-isomorfos obtidos possuem a mesma SOG dos originais, o que foi garantido pela aplicação dos seguintes critérios:

- para grafos (p, q) em *grade*, são selecionados aleatoriamente dois vértices adjacentes, ambos de grau 4, removendo-se a aresta que os liga e criando-se uma aresta entre dois vértices de grau 3,não vizinhos, escolhidos aleatoriamente na periferia do grafo. A Figura 5.6, traz um exemplo que ilustra uma troca ocorrida nesse tipo de grafo.

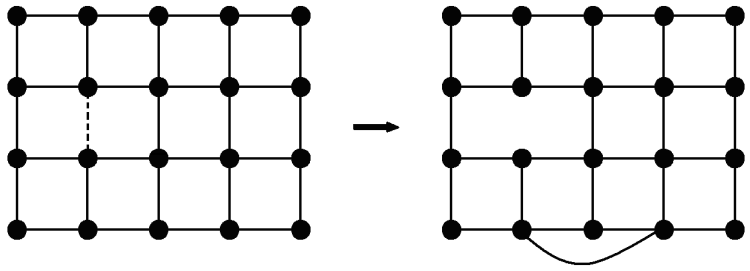

Figura 5.6: Exemplo de um grafo em grade

- para grafos (p, q) em *grade triangular com janelas*, é selecionada uma janela, trocando-se o sentido da diagonal da sua casa central. A Figura 5.7 traz um exemplo que ilustra uma troca ocorrida nesse tipo de grafo.

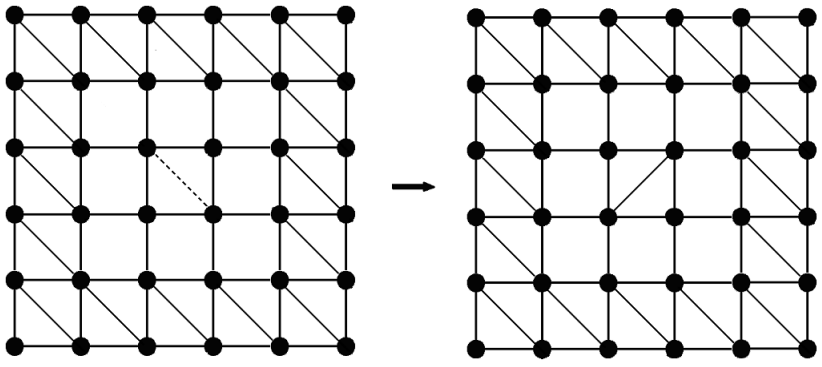

Figura 5.7: Exemplo de um grafo em grade triangular com janelas

- para grafos $(p, q)-$RQB, a última aresta do conjunto ligado a um vértice do ciclo interior é removida e desse vértice parte uma nova aresta que é ligada ao vértice anterior ao próximo conjunto de arestas. A Figura 5.8 traz um exemplo que ilustra uma troca ocorrida nesse tipo de grafo.

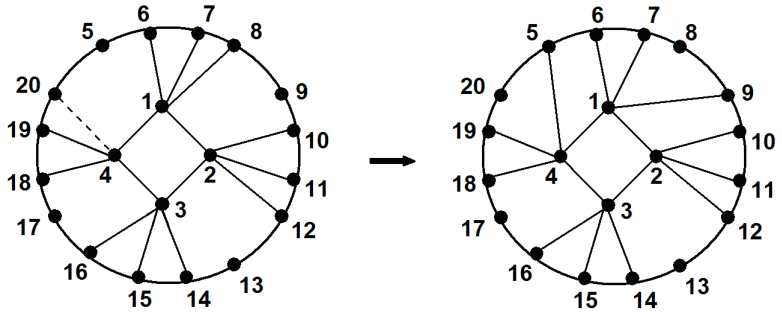

Figura 5.8: Exemplo de um grafo *roda quebrada de bicicleta*

Capítulo 6

Resultados obtidos para o PIG através da variância do PQA

6.1 Grafos Planares

Os resultados aqui discutidos envolvem as variâncias das instâncias originais e relaxadas. Estas últimas são mais fáceis de calcular, mas desde que $Relclass(F^-, D^+)$ inclui grafos não-isomorfos, uma igualdade destas variâncias não garante o isomorfismo sob a Conjetura 5.2.2, (página 84). Deve-se observar que essa conjetura não especifica a função que poderia discriminar dois grafos não isomorfos dados, mas com as instâncias originais ela diz que existe tal função; em relação às instâncias relaxadas, nada se pode afirmar, porém uma diferença nas variâncias será suficiente para indicar a ausência de isomorfismo. Portanto, é útil começar pelo cálculo dessas variâncias, já que isto pode ser feito rapidamente.

Para facilitar a visualização, as tabelas seguintes apresentam valores de desvios (expressões 6.1 e 6.2), expressos em porcentagens. Esses desvios correspondem às razões entre as variâncias das instâncias associadas, $(\sigma^2(PQA(G_1, G_1))$ e $(\sigma^2(PQA(G_2, G_2))$, e a variância da instância em relação à variância da instância construida com o par de grafos, ($\sigma^2(PQA(G_1, G_2))$),

$$R\%vars.Grafo1 = 100((\sigma(PQA(G_1, G_1))/\sigma^2(PQA(G_1, G_2)) - 1)), e \qquad (6.1)$$

$$R\%vars.Grafo2 = 100((\sigma^2(PQA(G_2, G_2))/\sigma^2(PQA(G_1, G_2)) - 1)). \qquad (6.2)$$

O tempo de processamento das variâncias foi obtido num computador com processador Intel Core 2 Quad $Q6600$, 3Gb de RAM, sob o Sistema Operacional Win-

dows XP e compilador Fortran *Powerstation* IV.

Para todos os grafos planares estudados, o padrão de troca das arestas foi o mesmo apresentado nas Figuras 5.6, 5.7 e 5.8.

Para definir os grafos em grade, usamos nove pares de dimensões (p, q), que vão de 30 até 3000 vértices, como mostrado (Tabela 6.1), com uma e quatro trocas de arestas para cada grafo, ambos usando as funções *c4* e ***gr***.

Os grafos em grade triangular com *janelas* foram construídos com 80 até 3000 vértices. Para esse tipo de grafo, não se usou o de ordem 30 pelo fato dele oferecer poucas opções para posicionar 4 janelas.

Para os grafos $(p, q)-$RQB, iniciou-se com $(4, 3)$ (20 vértices), indo-se até $(50, 20)$ (1100 vértices). A função usada foi *c4*.

Deixamos de informar os tempos computacionais, visto que os valores obtidos sempre foram inferiores a 0.05 segundos e que nenhuma execução para o programa de variâncias obteve tempo de processamento maior que 1 segundo.

O conjunto de testes correspondentes aos grafos em grade triangular com janelas (p, q) mostrados na Tabela 6.2 a seguir, usa a nomenclatura *Grdgw*, cujo significado é ***GR****id* ***D****ia****G****onal graphs with* ***W****indows*. De novo, aplicamos aqui as funções ***c4*** e soma de graus.

Esses grafos apresentaram um sério desafio para a validação da técnica. Aparentemente a criação das janelas em torno das mudanças nas arestas dilui seus efeitos na estrutura do grafo, as trocas na estrutura tendendo a ser simétricas. Como podemos ver, na maioria dos exemplos as variâncias relaxadas são incapazes de discriminar entre a tripla de instâncias e as relações com a variância original apresentam valores muito pequenos.

Nas Tabelas, apresentamos uma coluna de *densidade* planar $m/(3n - 6)$ em percentagem.

Os grafos $(p, q)-$RQB não responderam à função de valor do somatório de graus, tanto com as instâncias relaxadas quanto com as originais. A função de valor *c4*, ao contrário, permitiu uma boa discriminação entre seus quase-isomorfos.

6.2 Discussão sobre o efeito das trocas de aresta

Os grafos em grade e em grade triangular nos forneceram uma visão clara das razões por trás da sensitividade da técnica baseada no PQA. As Figuras 5.6, 5.7 e 5.8 mostram o efeito de uma troca simples de arestas sobre os valores de outras arestas, dentro da vizinhança dos vértices relacionados, quando consideramos a função de valor do somatório de graus.

Com os grafos em grade, mostrados na Figura 5.6, temos 12 valores trocados e, com grafos em grade triangular com janelas, ver Figura 5.7, temos 8 trocas. O efeito

sobre os valores da solução PQA podem ser melhor visualizados.

Os grafos $(p, q)-$RQB sofreram 2 trocas do somatório de graus no círculos interno e externos e as q arestas adjacentes entre os dois círculos trocaram também os seus valores. Temos então, $q + 4$ trocas de somatórios de graus. Diferentemente das famílias examinadas anteriormente, o valor do número de trocas aumenta com um parâmetro do grafo.

Alguns desses resultados foram incluídos em (124).

6.3 Tempo de Processamento das funções de peso

Ambas funções de peso propostas são polinomiais, $\boldsymbol{gr}(i, j)$ sendo $O(m)$ e $c_k(i, j)$ sendo $O(n^k)$ se o produto das matrizes é feito com as matrizes inteiras. Então, o primeiro é mais "barato"para calcular que as variâncias, e o segundo pode inclusive ser mais "caro". Através de um aperfeiçoamento de programação, conseguimos uma redução do tempo de processamento de $c_k(i, j)$, pelo uso de listas de adjacências, especialmente por se tratar de grafos com baixa densidade.

A Tabela 6.4 mostra alguns resultados de tempo de processamento, em segundos, para as funções de peso utilizadas nos testes.

É fácil de ver que o tempo de $\boldsymbol{gr}(i, j)$ cresce muito pouco em função da ordem dos grafos, enquanto $c_3(i, j)$ e $c_4(i, j)$ crescem rapidamente, seus tempos de execução sendo geralmente maiores do que aqueles exigidos pelo cálculo das variâncias. O aperfeiçoamento da programação deve poder contribuir para atenuar esse problema.

6.4 Um exemplo de isomorfismo

Como um exemplo de saída do programa construído para o isomorfismo, apresentamos o grafo $R30\_8\_1$ com função de valor c_4,(Boaventura e Leite, (121)) , junto com o seu valor calculado com os valores das duas variâncias, iguais para as 3 instâncias. O isomorfismo está caracterizado pela permutação usada, que diferencia as duas listas de adjacência (Figura 6.1).

Order (n)	Arquivos Grid_p_q / ql	Trocas		Arestas		Rel. % vars. Grafo 1		Rel. % vars. Grafo 2		Rel. % vars. Grafo 1		Rel. % vars. Grafo 2	
		N°	%	m	Pl. density m/(3n-6) %	RIx	Orig	RIx	Orig	RIx	Orig	RIx	Orig
						Função de Peso - C4				Função de Peso - Soma dos Graus			
30	Grid_6_5/5a	1	2041	49	58.3	96.219	955.764	-877.731	-872.167	0.19292	-0.12297	-0.19255	-0.12067
	Grid_6_5/5d	4	8.163			2.853.937	-2.220.754	2.768.456	-2.137.121	166.394	0.98460	-163.671	-0.78034
80	Grid_10_8/8a	1	0.704	142	60.7	172.931	-169.992	169.358	-166.416	0.10905	0.08625	-0.10893	-0.08579
	Grid_10_8/8d	4	2.820			963.043	-878.480	959.919	-875.805	-0.09098	-0.13811	-0.09106	0.13996
120	Grid_12_10/10a	1	0.459	218	61.6	173.391	-170.437	172.512	-169.579	0.08456	0.07242	-0.08449	-0.07222
	Grid_12_10/10d	4	1835			499.125	-475.405	493.396	-469.851	0.28809	0.24624	-0.28726	-0.24392
300	Grid_20_15/15a	1	0.177	565	63.2	0.38539	-0.38391	0.38341	-0.38194	0.02434	0.02274	-0.02433	-0.02273
	Grid_20_15/15d	4	0.708			178.344	-175.221	177.980	-174.865	0.09133	0.08555	-0.09125	-0.0854
600	Grid_30_20/20a	1	0.087	1150	64.1	0.21309	-0.21263	0.21351	-0.21305	0.00290	0.00283	-0.00290	-0.0283
	Grid_30_20/20d	4	0.348			0.86551	-0.85808	0.86569	-0.85826	0.03479	0.03366	-0.03478	-0.03364
1000	Grid_40_25/25a	1	0.052	1935	64.6	0.10715	-0.10704	0.10699	-0.10688	0.00678	0.00664	-0.00678	-0.00664
	Grid_40_25/25d	4	0.207			0.50917	-0.50659	0.50881	-0.50623	0.02544	0.02492	-0.02543	-0.02491
1500	Grid_50_30/30a	1	0.035	2840	63.2	0.07026	-0.07022	0.07020	-0.07015	0.00445	0.00438	-0.00445	-0.00438
	Grid_50_30/30d	4	0.141			0.29892	-0.29803	0.29871	-0.29782	0.01670	0.01646	-0.01670	-0.01646
2000	Grid_50_40/40a	1	0.025	3910	65.2	0.05209	-0.05206	0.05205	-0.05202	0.00330	0.00327	-0.00330	-0.00327
	Grid_50_40/40d	4	0.102			0.22147	-0.22098	0.22135	-0.22086	0.01240	0.01226	-0.01240	-0.01225
3000	Grid_60_50/50a	1	0.017	5890	65.5	0.03433	-0.03432	0.03431	-0.03430	0.00218	0.00216	-0.00216	-0.00216
	Grid_60_50/50d	4	0.068			0.15638	-0.15613	0.15640	-0.15615	-0.00163	-0.00166	0.00163	0.00166

Tabela 6.1: Resultados para instâncias PQA construídas de grafos quase isomorfos (p, q) grafos em grade (val *c4* e somatório de graus)

Order (n)	Arquivos Grid_p_q / ql	Trocas		Arestas		Rel. % vars. Grafo 1		Rel. % vars. Grafo 2		Rel. % vars. Grafo 1		Rel. % vars. Grafo 2	
		N°	%	m	Pl. density m/(3n-6) %	Rlx	Orig	Rlx	Orig	Rlx	Orig	Rlx	Orig
						Função de Peso - C4				Função de Peso - Soma dos Graus			
80	Grdgw_10_8/8a	1	0.500	200	85,5	-0.0063485	-0.0059707	0.0063489	0.0059722	-0.0094004	-0.0085337	0.0094013	0.0085370
	Grdgw_10_8/8d	4	2.235	179	75,8	-0.0098181	-0.0091661	0.0098181	0.0091688	-0.0130442	-0.0117241	0.0130459	0.0117295
120	Grdgw_12_10/10a	1	0.324	309	87,3	0.0000000	0.0000516	0.0000000	-0.0000516	0.0000000	0.0000950	0.000000	-0.0000950
	Grdgw_12_10/10d	4	1.379	290	81,9	0.0000000	-0.0001230	0.0000000	0.0001232	0.0000000	-0.0002089	0.0000000	0.0002089
300	Grdgw_20_15/15a	1	0.121	823	92	0.0000000	0.0000100	0.0000000	-0.0000100	0.0000000	0.0000168	0.0000000	-0.0000168
	Grdgw_20_15/15d	4	0.500	799	89,3	0.0000000	0.0000090	0.0000000	-0.0000090	0.0000000	0.0000160	0.0000000	-0.0000160
600	Grdgw_30_20/20a	1	0.059	1693	94,4	0.0000000	0.0000028	0.0000000	-0;0000028	0.0000000	0.0000045	0.0000000	-0.0000045
	Grdgw_30_20/20d	4	0.240	1669	93.0	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000045	0.0000000	-0.0000045
1000	Grdgw_40_25/25a	1	0.035	2863	95.6	0.0000000	0.0000010	0.0000000	-0.0000010	0.0000000	0.0000017	0.0000000	-0.0000017
	Grdgw_40_25/25d	4	0.141	2839	94.8	0.0000000	0.0000020	0.0000000	-0.0000020	0.0000000	0.0000025	0.0000000	-0.0000025
1500	Grdgw_50_30/30a	1	0.023	4253	94.7	0.0000000	0.0000005	0.000000	0.0000005	0.0000000	0.0000008	0.0000000	-0.0000008
	Grdgw_50_30/30d	4	0.094	4229	94.1	0.0000000	0.0000005	0.0000000	-0.0000005	0.0000000	0.0000008	0.0000000	-0.0000008
2000	Grdgw_50_40/40a	1	0.017	5813	97.0	0.0000000	0.0000003	0.0000000	-0.000003	0.0000000	0.0000004	0.0000000	-0.0000004
	Grdgw_50_40/40d	4	0.068	4989	96.6	0.0000000	0.0000003	0.0000000	-0.0000003	0.0000000	0.0000004	0.0000000	-0.0000004
3000	Grdgw_60_50/50a	1	0.011	8753	97.3	0.0000000	0.0000001	0.0000000	-0.0000001	0.0000000	0.0000002	0.0000000	-0.0000002
	Grdgw_60_50/50d	4	0.046	8729	97.0	0.0000000	0.0000001	0.0000000	-0.0000001	0.0000000	0.0000002	0.0000000	-0.0000002

Tabela 6.2: Resultados para instâncias PQA construídas de grafos quase isomorfos (p, q) grafos em grade triangular com janelas

Parâmetros	Ordem (n)	Arquivos	Trocas		Arestas		Rel. % vars. Grafo 1		Rel. % vars. Grafo 2	
p,q		Bbw/Bbw (trocas)	N°	%	m	Pl. Density m/3n-6 %	Rlx	Orig	Rlx	Orig
4, 3	20	Bbw_4_3/(1,3)	2	6.250	32	59.2	-2.303.799	-2.301.994	2.991.185	2.988.858
		Bbw_4_3/(2,4)					-2.255.674	-2.257.333	2.910.547	2.913.911
6, 4	36	Bbw_6_4/(1,2)	2	3.333	60	58.8	-204.402	-208.500	208.668	213.111
		Bbw_6_4/(1,3)					-400.473	-408.168	417.180	426.239
8, 6	64	Bbw_8_6/(1,5)	2	1.786	112	60.2	-218.505	-216.645	223.385	221.449
		Bbw_8_6/(1,3,5,7)	4	3.572			-427.526	-423.613	446.620	442.381
10, 8	100	Bbw_10_8/(1,5,9)	3	1.667	180	61.2	-144.869	-150.455	146.998	152.783
		Bbw_10_8/(2,4,6,8)	4	2.222			-192.224	-199.575	195.991	203.692
15, 10	180	Bbw_15_10/(3,5)	2	0.606	330	61,8	-0.47527	-0.49017	0.47754	0.49260
		Bbw_15_10/(4,6,8,10)	4	1.212			-0.94603	-0.97545	0.95506	0.98513
20, 12	280	Bbw_20_12/(5,10)	2	0.384	520	62.2	-0.27432	-0.28161	0.27507	0.28241
		Bbw_20_12/(6,12,18)	3	0.577			-0.41091	-0.42181	0.41261	0.42361
30, 15	510	Bbw_30_15/(5,15)	2	0.208	960	63.8	-0.13109	-0.13371	0.13126	0.13389
		Bbw_30_15/(7,14,21)	3	0.312			-0.19651	-0.20043	0.19690	0.20083
50, 20	1100	Bbw_50_20/(10,20,30)	3	0.142	2100	63.7	-0.07523	-0.07623	0.07529	0.07629
		Bbw_50_20/(15,25,35,45)	4	0.190			-0.10028	-0.10162	0.10038	0.10172

Tabela 6.3: Resultados para instâncias PQA construídas de grafos quase isomorfos (p, q)-RQB com função de valor ***c4***

p	q	Ordem,n	Somatório de Grau	*Contagem do caminho c3*	*Contagem do caminho c4*
10	8	80	< 0,01	< 0,01	< 0,01
12	10	120	< 0,01	< 0,01	0,01
20	15	300	0,01	0,14	0,21
30	20	600	0,01	1,43	2,01
40	25	1000	0,01	5,08	7,12
50	30	1500	0,02	13,97	24,41
50	40	2000	0,02	30,17	57,74
60	50	3000	0,02	91,40	194,37

Tabela 6.4: Alguns tempos de processamento (seg) para as funções de peso

```
Reg30_8_1_c3:      30
 1    1    1    1    1    1    1    2    2    2    2    2    2    2    2    3    3    3    3    3    4    4    4    4    4    4    4
 5    5    5    5    5    6    6    6    6    6    6    6    7    7    7    7    7    7    8    8    8    8    8    8    9    9    9
 9    9   10   10   10   10   11   11   11   12   12   12   12   12   13   13   13   13   13   13   14   14   14   14   14   15   15
15   16   17   17   18   18   18   19   19   19   20   20   20   20   21   21   22   22   22   22   24   26   26   27

 3    9   11   14   16   23   28    5   10   12   17   20   21   24   30    4   10   11   20   28   13   14   17   20   23   28   29
 8   16   24   25   30    9   14   18   21   26   27   30   12   18   21   23   24   25   16   20   25   26   29   30   10   16   21
24   30   11   16   17   28   17   22   25   13   15   16   23   30   17   18   19   23   25   27   18   19   27   28   29   23   26
29   23   19   23   22   25   27   21   22   28   22   24   26   27   24   28   24   25   27   28   26   29   30   29

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

Permutation for the isomorph:
(13 2 7 19 1 17 29 18 24 27 26 3 21 15 12 14 9 22 28 20 8 16 25 23 4 6 5 10 11 30)

Reg30_8_1iso1_c3:      30
 1    1    1    1    1    1    2    2    2    2    2    2    2    3    3    3    3    3    3    4    4    4    4    4    4    5    5
 5    5    5    5    5    6    6    6    6    6    6    6    7    7    7    7    7    7    8    8    8    8    8    9    9    9    9
 9    9    9   10   10   10   10   10   10   11   11   11   11   12   13   13   13   13   13   13   14   14   14   14   15   15   15   15
16   16   16   16   16   17   17   17   18   18   19   19   19   20   21   21   21   22   23   23   24   24   25   26

 2    4   14   18   23   30    3    8    9   20   23   27   30   12   14   21   25   29   30   16   18   21   22   26   29   11   15
16   17   20   21   22   11   12   17   18   20   23   30   10   13   19   20   26   27   10   17   23   24   28   29   19   21   25
26   27   28   13   15   16   19   27   28   12   15   18   19   25   14   15   24   25   26   18   24   25   27   17   19   22   28
20   22   23   26   28   22   24   30   20   30   20   21   25   23   22   25   28   29   24   29   27   30   29   27

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

Variances for the three test instances: relaxed 170.1935335, originals 173.3623220.
R% Graphs 1 and 2: 0.00000
```

Figura 6.1: Um exemplo de saída de grafo com seu isomorfo

Capítulo 7

Conclusões e propostas de pesquisa adicional

7.1 Uso de memória no VNS básico

7.1.1 Conclusões

Foi introduzido neste trabalho o uso de memória com o VNS, na tentativa de contribuir com resultados médios mais promissores para o PQA. Com este propósito, as técnicas usadas neste trabalho receberam o nome de *Memorized Variable Neighborhood Search*, podendo também ser chamado de *Memorized VNS* ou simplesmente MVNS, cujo escopo foi restrito à sua versão básica.

Esta proposta procurou fazer uso apenas das propriedades das vizinhanças do VNS, não associando o controle de memória a um parâmetro externo, ou à iteração atual, ou ao cálculo do custo associado a cada vértice. Procurou-se também preservar a lista de propriedades desejáveis em uma metaheurística.

Numa análise comparativa entre os testes de desempenho com controle único de memória, verificou-se que o controle por movimento se mostrou mais robusto, pois independentemente do local de inserção (na agitação ou melhora da solução atual) não houve variação no total de instâncias que alcançaram o $VOMVC$. Já o controle de memória por vértice mostrou uma variação considerável entre suas duas versões.

Isto nos levar a concluir que o tipo de controle de memória (vértice ou movimento) pode ser decisivo na qualidade dos resultados obtidos, já que os controles por movimento auxiliaram, na média, no alcance de melhores resultados que os obtidos com o controle de memória por vértice.

Ainda sobre o controle único de memória, verificou-se que a inserção de memória na agitação na solução atual foi mais promissora, já que conduziu a melhores resultados médios que a inserção na melhora da solução atual.

Quando o controle de memória é duplo, o controle por vértice mostrou-se mais

interessante já que, sempre que foi inserido, os tempos médios de execução foram menores e as estagnações médias mais próximas do critério de parada, que os observados na versão equivalente com controle por movimento.

Com os todos os resultados obtidos neste trabalho, verificamos que, para o nosso problema em estudo (PQA) e vizinhanças escolhidas (VNS), o uso de memória no VNS trouxe benefícios à sua versão básica e que não existe um controle (único ou duplo) melhor que o outro. Ambos possuem vantagens e desvantagens.

Se o interesse for uma versão com menor gasto computacional, a versão com duplo controle por vértice se mostrou mais interessante. Acreditamos que o que motivou isso tenha sido a grande restrição imposta pelo controle por vértice, já que, embora diminua as possibilidades de escolha aleatória , na agitação, pode ter sido benéfico para a busca local, direcionando a busca para uma região mais promissora.

Caso o intuito seja uma versão menos susceptível a variações, nossa sugestão é a versão com controle único memória por movimento. Por ser menos restritiva, permite uma maior varredura na região de busca, o que pode trazer benefícios ao problema em estudo.

7.1.2 Propostas de pesquisa adicional

Como trabalho futuro, propomos para as versões de algoritmos que conseguiram os melhores resultados neste trabalho:

- Avaliar o comportamento dos algoritmos para outros problemas de Otimização Combinatória;
- Formular algoritmos análogos tomando como base outras versões do VNS e do VND;
- Híbridar os algoritmos aqui propostos com algumas metaheurísticas da literatura;
- Tentar superar os resultados médios obtidos pelo algoritmo *TabuTaillard*;
- Paralelizar os algoritmos propostos, onde todos os processos façam uso do mesmo controle de memória.

7.2 O estudo da variância no Problema de Isomorfismo de Grafos

7.2.1 Conclusões

Também foi introduzido neste trabalho um estudo da variância no Problema de Isomorfismo de Grafos, através da estrutura do PQA.

Apesar do PQA ser NP-difícil, o cálculo da variância das instâncias do PQA é polinomial. Como o cálculo da variância não é suficiente para garantir que dois grafos sejam isomorfos, fizemos uso de duas funções de valor para os conjuntos de arestas do par de grafos. Estas funções foram baseadas em propriedades da estrutura dos grafos, garantindo assim a sua invariância em relação ao isomorfismo e, portanto, a das variâncias PQA nelas baseadas. Todo o trabalho é, de fato, baseado na Conjetura 5.2.2 referente ao uso das funções de valor.

A primeira função é a soma dos graus dos vértices que definem cada aresta e a segunda, representa o calculo dos k-caminhos de uma matriz de adjacência. Para a segunda função, a avaliação obtida mostrou-se uma invariante com respeito ao isomorfismo, que nos levou a propor a seguinte Conjetura: Para cada par de grafos de mesma ordem e tamanho, existe uma avaliação invariante tal que, para não isomorfos, as instâncias irão possuir diferentes variâncias.

Para avaliar nossa proposta, fizemos uso de três classes de grafos planares. A primeira delas foi a de grafos em grade, cujos resultados foram significativos para ambas as funções. A segunda classe corresponde a grafos em grade com janelas. Para esta classe, na maioria dos exemplos as variâncias relaxadas foram incapazes de discriminar entre a tripla das instâncias e as relações com a variância original apresentaram valores muito pequenos. Uma justificativa para isso é a estrutura peculiar do grafo, como discutido. A última classe, formada por grafos RQB, não respondeu à função *gr*, porém a função de valor *c4* permitiu uma boa discriminação entre seus quase isomorfos.

7.2.2 Propostas de pesquisa adicional

Como conseguimos uma grande redução de tempo de processamento ao fazermos uso de listas de adjacência, nossa intenção como trabalho futuro sobre o estudo da variância no Problema de Isomorfismo de Grafos é:

- Avaliar o resultado para grafos planares de maior ordem, em princípio procurando atingir ordens de 250.000 ou maiores, visando a possibilidade da construção de uma ferramenta para detecção de pequenas diferenças em imagens. Um trabalho nesta direção é (125);

- Passar a utilizar compiladores para 64 bits, de modo a reduzir substancialmente os tempos de processamento facilitando assim o seu uso em intâncias de ordens elevadas.

Referências Bibliográficas

[1] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., et al., *Introduction to Algorithms*. The MIT Press, 2002.

[2] KOOPMANS, T. C., BECKMANN, M., *Assignment problems and the locations of economics activities*. Econometrica, vol. 25, pp 53-76, 1957.

[3] PARDALOS, P. M., RENDL, F., WOLKOWICZ, H., *The quadratic assignment problem: a survey and recent developments*. DIMACS Series Discr. Math. Theor. Comp. Sci., vol 16, pp 1-42, 1994.

[4] LOIOLA, E. M., ABREU, N. M. M., BOAVENTURA-NETTO, P. O., et al., *A survey for the quadratic assignment problem*. European Journal of Operational Research v. 176, p. 657-690, 2007.

[5] SAHNI, S., GONZALES, T., *P-complete approximation problems*. Journal of the Association for Computing Machinery 23, 555-565, 1976.

[6] ANSTREICHER, K. M., BRIXIUS, N. W., *A new bound for the quadratic assignment problem based on convex quadratic programming*. Mathematical Programming 89 (3)), 341-357, 2001.

[7] GILMORE, P. C., *Optimal and suboptimal algorithms for the quadratic assignment problem*. SIAM Journal on Applied Mathematics 10, 305-313, 1962.

[8] LAWLER, E. L., *The quadratic assignment problem*. Management Science 9, 586-599, 1963.

[9] ABREU, N. M. M., *Um Estudo Algébrico e Combinatório do Problema Quadrático de Alocação segundo Koopmans e Beckmann*. Tese de D.Sc., COPPE/UFRJ, 1984.

[10] ABREU, N. M. M., BOAVENTURA-NETTO, P. O., *Sobre a Viabilidade de Soluções do Problema Quadrático de Alocação*. III CLAIO, Santiago do Chile. Anais do III CLAIO. Santiago: Universidad de Chile. p. 141-156, 1986.

[11] ABREU, N. M. M., BOAVENTURA-NETTO, P. O., *The quadratic assignment problem:Permutation ordering and inversions*. AMSE Rev., 10(3), 21-52, 1989.

[12] ABREU, N. M. M., QUERIDO, T. M., GUIMARÃES, E. M., *The Traveling Salesman Problem as an Instance the Quadratic Assignment Problem*. Advances in Modelling and Simulation, v. 26, n. 2, p. 1-22, 1991.

[13] QUERIDO, T. M., ABREU, N. M. M., BOAVENTURA-NETTO, P. O., *Rede de Soluções do Problema Quadrático de Alocação*. In: XXVI SBPO, Florianópolis. Anais do XXVI SBPO. Rio de Janeiro: SOBRAPO. v. único. p. 770-774, 1994.

[14] VERNET, O., RODRIGUES, R. M. N. D., ABREU, N. M. M., *Reticulados de Permutações*. Relatórios Técnico, EP-03/95 Série P.O., Programa de Engenharia de Produção - COPPE/UFRJ, Brasil, 1995.

[15] ABREU, N. M. M., QUERIDO, T. M., BOAVENTURA-NETTO, P. O., *RedInv-SA: A simulated annealing for the quadratic assignment problem*. RAIRO, Operations Research, 33(3), 249-273, 1999.

[16] ABREU, N. M. M., BOAVENTURA-NETTO, P. O., GOUVEA, E. F., et al., *On the Use of Variance in Complexity Measures for the Quadratic Assignment Problem*. INFORMS Cincinnati Spring Meeting, Cincinnati. ICSM Program. Cincinnati: INFORMS. v. único. p. 78-78, 1999.

[17] MARINS, M. T., ABREU, N. M. M., BOAVENTURA-NETTO, P. O., *Instâncias Aparentadas do Problema Quadrático de Alocação*. In: XXXI SBPO, Juiz de Fora. Anais do XXXI SBPO. Rio de Janeiro: SOBRAPO, 1999. v. único. p. 1062-1070, 1999.

[18] RANGEL, M. C., ABREU, N. M. M., BOAVENTURA-NETTO, P. O., *Grasp para o pqa: Um limite de aceitação para soluções iniciais*. Pesquisa Operacional, 20(1), 45-58, 2000.

[19] RANGEL, M. C., ABREU, N. M. M., *Ordenações parciais nos conjuntos das soluções dos problemas de alocação linear e quadrático*. Pesquisa Operacional, 23(2), 265-284, 2003.

[20] BOAVENTURA-NETTO, P. O., LOIOLA, E. M., *Uma Estrutura de Vizinhança para o Problema Quadrático de Alocação*. In: XXXIII Simpósio Brasileiro de Pesquisa Operacional (XXXIII SBPO), Campos de Jordão. Anais do XXXIII SBPO. Niterói/RJ: Instituto Doris Ferraz de Aragon (ILTC) v. 1. p. 1288-1297, 2001.

[21] BOAVENTURA-NETTO, P. O., *A Structure-Based Heuristic for the Quadratic Assignment Problem*. In: ECCO XV, 2002, Lugano. Proceedings of ECCO XV. Lugano: Università di Lugano, 2002.

[22] ABREU, N. M. M., BOAVENTURA-NETTO, P. O., QUERIDO, T. M., et al., *Classes of quadratic assignment problem instances: isomorphism and difficulty measure using a statistical approach*. Discrete Applied Mathematics, 124(1-3), 103-116, 2002.

[23] BOAVENTURA-NETTO, P. O., *Um estudo sobre a construção de um algorítmo VNS para o Problema Quadrático da Alocação*. In: XII CLAIO, 2004, Havana. XII CLAIO - Programa y actas de resumenes. Havana : Universidad de La Habana. v. 1. p. 37-37, 2004.

[24] MARINS, M. T., ABREU, N. M. M., JURKIEWISCZ, S., *Automorphism of groups and quadratic assignment problem*. In: Congreso Latino-Íberoamericano de Investigation Operativa, Havana. Memorias del Claio. Universidad de La Havana. v. 1. p. 651-656, 2004.

[25] LOIOLA, E. M., ABREU, N. M. M., BOAVENTURA-NETTO, P. O., *Uma revisão comentada das abordagens do problema quadrático de alocação*. Pesquisa Operacional, Rio de Janeiro, v. 24, n. 1, p. 73-110, 2004.

[26] QUERIDO, T. M., *Simulated annealing no grafo das inversões do problema quadrático de alocação*. Tese de D.Sc., Programa de Engenharia de Produção, COPPE/UFRJ, Rio de Janeiro, Brasil, 1994.

[27] MOREIRA, A. S. T., *Aplicação de uma técnica branch-and-bound a um modelo reduzido para o problema do caixeiro-viajante*. Dissertação de M.Sc., Programa de Engenharia de Produção, COPPE/UFRJ, Rio de Janeiro, Brasil, 1998.

[28] FIRMO, A. L. M., *Geradores de instâncias do problema quadrático de alocação*. Dissertação de M.Sc., Programa de Engenharia de Produção, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 1999.

[29] LOIOLA, E. M., *Um algoritmo com parâmetros estatísticos para o PQA*. Dissertação de M.Sc., Programa de Engenharia de Produção, COPPE/UFRJ, Rio de Janeiro, Brasil, 2000.

[30] RANGEL, M. C., *Contribuições Algébricas ao Problema Quadrático de Alocação*. Tese de D.Sc., Programa de Engenharia de Produção - COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2000.

[31] MARINS, M. T., *O uso de Automorfismo de Grafos no Problema Quadrático de Alocação*. Tese de D.Sc., Programa de Engenharia de Produção, COPPE/UFRJ, Rio de Janeiro, Brasil, 2001.

[32] MOREIRA, A. S. T., *Algoritmos híbridos grasp-busca tabu, utilizando a estrutura da matriz de Picard-Queyranne, para o problema quadrático de alocação*. Tese de D.Sc., Programa de Engenharia de Produção, COPPE/UFRJ, Rio de Janeiro, Brasil, 2006.

[33] GAVETT, J. W., PLYTER, N. V., *The optimal assignment of facilities to locations by branch and bound*. Operations Research, Vol. 14, pp. 210-232, 1966.

[34] BURKARD, R. E., STRATMAN, R. H., *Numerical investigations on quadratic assignment problem*. Naval Research Logistics Quarterly 25, 129-140, 1978.

[35] ÇELA, E., *The quadratic assignment problem: Theory and algorithms*. In: Du, D.Z., Pardalos, P. (Eds.), Combinatorial Optimization. Kluwer Academic publishers, Dordrecht, 1998.

[36] BAZARAA, M. S., SHERALI, H. D., *Benders' partitioning scheme applied to a new formulation of the quadratic assignment problem*. Naval Research Logistics Quarterly 27, 29-41, 1980.

[37] KAUFMAN, L., BROECKX, F., *An algorithm for the quadratic assignment problem using Bender's decomposition*. European Journal of Operation Research 2, 204-211, 1978.

[38] BAZARAA, M. S., SHERALI, H. D., *On the use of exact and heuristic cutting plane methods for the quadratic assignment problem*. Journal of the Operational Research Society 33, 991-1003, 1982.

[39] BURKARD, R. E., BONNIGER, T., *A heuristic for quadratic Boolean programs with applications to quadratic assignment problems*. European Journal of Operation Research 13, 374-386, 1983.

[40] MIRANDA, G., LUNA, H. P. L., MATEUS, G. R., et al., *A performance guarantee heuristic for electronic components placement problems including thermal effects*. Computers and Operations Research 32, 2937-2957, 2005.

[41] PADBERG, M. W., RINALDI, G., *A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems*. SIAM Review 33, 60-100, 1991.

[42] JÜNGER, M., KAIBEL, V., *On the SQAP-polytope*. SIAM Journal on Optimization 11 (2), 444-463, 2000.

[43] JÜNGER, M., KAIBEL, V., *The QAP-polytope and the star transformation*. Discrete Applied Mathematics 111 (3), 283-306, 2001.

[44] JÜNGER, M., KAIBEL, V., *Box-inequalities for quadratic assignment polytopes*. Mathematical Programming 91 (1), 175-197, 2001.

[45] PADBERG, W., RIJAL, P., *Location, Scheduling, Design and Integer Programming*. Kluwer Academic publishers, Boston, 1996.

[46] KAIBEL, V., *Polyhedral combinatorics of quadratic assignment problems with less objects than locations*. Lecture Notes in Computer Science 1412, 409-422, 1998.

[47] BLANCHARD, A., ELLOUMI, S., FAYE, A., et al., *A cutting algorithmfor the quadratic assignment problem*. INFOR 41 (1), 35-49, 2003.

[48] MANS, B., MAUTOR, T., ROUCAIROL, C., *A parallel depth first search branch and bound algorithm for the quadratic assignment problem*. European Journal of Operational Research 81, 617-628, 1995.

[49] BOZER, Y. A., SUK-CHUL, R., *A branch and bound method for solving the bidirectional circular layout problem*. Applied Mathematical Modeling 20 (5),342-351, 1996.

[50] PARDALOS, P. M., RAMAKRISHNAN, K. G., RESENDE, M. G. C., et al., *Implementation of a variance reduction-based lower bound in a branch-and-bound algorithm for the quadratic assignment problem*. SIAM Journal on Optimization 7 (1), 280-294, 1997.

[51] BRUNGGER, A., MARZETTA, A., CLAUSEN, J., et al., *Solving large-scale QAP problems in parallel with the search library ZRAM*. Journal of Parallel and Distributed Computing 50 (1-2), 157-169, 1998.

[52] BALL, M. O., KAKU, B. K., VAKHUTINSKY, A., *Network-based formulations of the quadratic assignment problem*. European Journal of Operational Research 104 (1), 241-249, 1998.

[53] SPILIOPOULOS, K., SOFIANOPOULOU, S., *An optimal tree search method for the manufacturing systems cell formation problem*. European Journal of Operational Research 105 (3), 537-551, 1998.

[54] BRIXIUS, N. W., ANSTREICHER, K. M., *Solving quadratic assignment problems using convex quadratic programming relaxations*. Optimization Methods and Software 16, 49-68, 2001.

[55] HAHN, P. M., HIGHTOWER, W. L., JOHNSON, T. A., et al., *Tree elaboration strategies in branch and bound algorithms for solving the quadratic assignment problem*. Yugoslavian Journal of Operational Research 11 (1), 41-60, 2001.

[56] HAHN, P. M., HIGHTOWER, W. L., JOHNSON, T. A., et al., *A level-2 reformulation-linearization technique bound for the quadratic assignment problem*. Working Paper 01-04, Systems Engineering Department, University of Pennsylvania, 2001.

[57] ROUCAIROL, C., *A parallel branch and bound algorithm for the quadratic assignment problem*. Discrete Applied Mathematics 18, 211-225, 1987.

[58] PARDALOS, P., CROUSE, J., *A parallel algorithm for the quadratic assignment problem*. In: Proceedings of the Supercomputing Conference 1989. ACM Press, pp. 351-360, 1989.

[59] MAUTOR, T., ROUCAIROL, C., *A new exact algorithm for the solution of quadratic assignment problems*. Discrete Applied Mathematics 55, 281-293, 1994.

[60] CLAUSEN, J., PERREGAARD, M., *Solving large quadratic assignment problems in parallel*. Computational Optimization and Applications 8, 111-127, 1997.

[61] BRUNGGER, A., MARZETTA, A., CLAUSEN, J., et al., *Joining forces in solving large-scale quadratic assignment problems*. In: Proceedings of the 11th International Parallel Processing Symposium IPPS. IEEE Computer Society Press, pp. 418-427, 1997.

[62] BURKARD, R. E., ÇELA, E., KARISCH, S. E., et al., *QAPLIB Home Page*. Disponível em: http://www.seas.upenn.edu/qaplib/, Acesso em janeiro de 2009, 2009.

[63] ANSTREICHER, K. M., BRIXIUS, N. W., GOUX, J.-P., et al., *Solving large quadratic assignment problems on computational grids*. Mathematical Programming 91 (3), 563-588, 2002.

[64] ARMOUR, G. C., BUFFA, E. S., *Heuristic algorithm and simulation approach to relative location of facilities*. Management Science 9 (2), 294-309, 1963.

[65] BUFFA, E. S., ARMOUR, G. C., VOLLMANN, T. E., *Allocating facilities with CRAFT*. Harvard Business Review 42 (2), 136-158, 1964.

[66] SARKER, B. R., WILHELM, W. E., HOGG, G. L., et al., *Backtracking of jobs in one-dimensional machine location problems*. European Journal of Operational Research 85 (3), 593-609, 1995.

[67] SARKER, B. R., WILHELM, W. E., HOGG, G. L., *One-dimensional machine location problems in a multi-product flowline with equidistant locations*. European Journal of Operational Research 105 (3), 401-426, 1998.

[68] TANSEL, B. C., BILEN, C., *Move based heuristics for the unidirectional loop network layout problem*. European Journal of Operational Research 108 (1), 36-48, 1998.

[69] BURKARD, R. E., *Locations with spatial interactions: The quadratic assignment problem*. In: Mirchandani, P.B., Francis, R.L. (Eds.), Discrete Location Theory. John Wiley and Sons, pp. 387-437, 1991.

[70] ARKIN, E. M., HASSIN, R., SVIRIDENKO, M., *Approximating the maximum quadratic assignment problem*. Information Processing Letters 77 (1), 13-16, 2001.

[71] GUTIN, G., YEO, A., *Polynomial approximation algorithms for TSP and QAP with a factorial domination number*. Discrete Applied Mathematics 119 (1-2), 107-116, 2002.

[72] YU, J., SARKER, B. R., *Directional decomposition heuristic for a linear machine-cell location problem*. European Journal of Operational Research 149 (1), 142-184, 2003.

[73] MISEVICIUS, A., *A new algorithm for the quadratic assignment problem*. Information Technology and Control 5, 39-44, 1997.

[74] FLEURENT, C., GLOVER, F., *Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory*. INFORMS Journal on Computing 11, 189-203, 1999.

[75] MISEVICIUS, A., RISKUS, A., *Multistart threshold accepting: Experiments with the quadratic assignment problem. Solving Certain Large Instances of the Quadratic*. Information Technology and Control 12, 31-39., 1999.

[76] FEO, T. A., RESENDE, M. G. C., *Greedy randomized adaptive search procedures*. Journal of Global Optimization 6, 109-133, 1995.

[77] KIRKPATRICK, S., GELATT, C. D., VECCHI, M. P., *Optimization by simulated annealing*. Science 220, 671-680, 1983.

[78] HOLLAND, J. H., *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[79] DORIGO, M., MANIEZZO, V., COLORNI, A., *Ant System: Optimization by a colony of cooperating agents*. IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1):29-41, 1996.

[80] GLOVER, F., *Heuristics for integer programming using surrogate constraints*. Decision Science 8, 156-166, 1977.

[81] GLOVER, F., *Tabu search-Part I*. ORSA Journal on Computing 1, 190-206, 1989.

[82] GLOVER, F., *Tabu search-PartII*. ORSA Journal on Computing 2, 4-32, 1989.

[83] LOURENÇO, H. R., MATRIN, O. C., STÜTZLE, T., *Iterated Local Search*. Handbook of Metaheuristics, Kluwer Academic Publishers, 2003.

[84] HANSEN, P., MLADENOVIĊ, N., *Variable neighborhood search*. Computers and Operations Research 24, 1097-1100, 1997.

[85] GLOVER, F., *Tabu search and adaptive memory programing - Advances, applications and challenges*. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, Interfaces in Computer Science and Operations Research, pages 1-75. Kluwer, 1996.

[86] MANIEZZO, V., COLORNI, A., *Algodesk: An experimental comparison of eight evolutionary heuristics applied to the quadratic assignment problem*. European Journal of Operational Research 81 (1), 188-204, 1995.

[87] TAILLARD, E., GAMBARDELLA, L. M., GENDREAU, M., et al., *Adaptive memory programming: A unified view of metaheuristics*. European Journal of Operational Research 135 (1), 1-16, 2001.

[88] DREZNER, Z., HAHN, P. M., TAILLARD, E. D., *Recent Advances for the Quadratic Assignment Problem with Special Emphasis on Instances that are Difficult for Meta-Heuristic Methods*. Annals of Operations Research 139, 65-94, 2005.

[89] TAILLARD, E., *Comparison of Iterative Searches for the Quadratic Assignment Problem*. Centre de Recherche sur les Tansports, Publication no 989, 1994.

[90] TAILLARD, E., *Robust taboo search for the quadratic assignment problem*. Parallel Computing 17, 443-455, 1991.

[91] PALUBECKIS, G., *An Algorithm for Construction of Test Cases for the Quadratic Assignment Problem*. Informatica. vol.11, n.3, 281-296, 2000.

[92] STÜTZLE, T., FERNANDES, S., *New benchmark instances for the QAP and the experimental analysis of algorithms*. Lecture Notes in Computer Science 3004, 199-209, 2004.

[93] HANSEN, P., MLADENOVIĊ, N., *Developments of Variable Neighborhood Search*. Les Cahiers du GERAD, G-2001-24, 2001.

[94] ATKINSON, R. C., SHIFFRIN, R. M., *Human memory: A proposed system and its control processes*. In K.W. Spence and J.T. Spence (Eds.), The psychology of learning and motivation, vol. 8. London: Academic Press, 1968.

[95] MELO, V. A., SILVEIRA, D. S., LEITE, L. S. B. S., et al., *Uma Heurística para o Reconhecimento de Grafos Isomorfos via o Problema Quadrático de Alocação*. XXXIX Simpósio Brasileiro de Pesquisa Operacional - SBPO, Fortaleza, Ceará, 2007.

[96] BUNKE, H., SHEARER, K., *A graph distance metric based on the maximal common subgraph.*. Pattern Recognition Letters, 19, 3-4, 255-259, 1998.

[97] DEPIERO, F., KROUT, D., *An algorithm using length-r paths to approximate subgraph isomorphism*. Pattern Recognition Letters 24, 33-46, 2003.

[98] LEE, L., RANGEL, M. C., BOERES, M. C. S., *Isomorfismo de Grafos como o Problema Quadrático de Alocação*. Anais do XXXIX SBPO, 1601-1612, Fortaleza, 2007.

[99] SUBRAMANIAN, A., MERCHE, M., MUNHOZ, P. L. A., et al., *Uma heurística baseada em Iterated Local Search para o Problema de Roteamento de Veículos com Múltiplos Depósitos*. XLII SBPO, Bento Gonçalves. Anais do XLII SBPO, 2010.

[100] ILIĊ, A., UROSEVIĊ, D., BRIMBERG, J., et al., *A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem*. European Journal of Operational Research 206, 289-300, 2010.

[101] BURKARD, R. E., RENDL, F., *A thermodynamically motivated simulation procedure for combinatorial optimization problems*. EJOR 17 (1984) 169-174, 1984.

[102] FRIEZE, A. M., YADEGAR, J., EL-HORBATY, S., et al., *Algorithms for assignment problems on an array processor*. Parallel Comput 11, 151-162, 1989.

[103] HANSEN, P., MLADENOVIĊ, N., PEREZ, J. A. M., *Variable Neighborhood Search*. European Journal of Operational Research 191, 593-595, 2008.

[104] DREZNER, Z., HAHN, P. M., TAILLARD, E. D., *QAP Instances - Drezner/Hahn/Taillard*. Disponível em: http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html, Acesso em janeiro de 2010, 2010.

[105] PALUBECKIS, G., *QAP Instances - Palubeckis*. Disponível em: http://www.soften.ktu.lt/ gintaras/qproblem.html, Acesso em janeiro de 2010, 2010.

[106] AHUJA, R. K., ORLIN, J. B., TIWARI, A., *A greedy genetic algorithm for the quadratic assignment problem*. Computers & Operations Research 27, 917-934, 2000.

[107] ESTANY, C. P., *Números Primos*. Disponível em: http://pinux.info/primos/, Acesso em janeiro de 2010, 2010.

[108] BARBUT, C. C. P., *Automorphismes du permutoèdre et votes de Condorcet*. Math. Inform. Sci. Hum. 28e (111) 73-82, 1990.

[109] STÜTZLE, T., *Iterated local search for the quadratic assignment problem*. European Journal of Operational Research,174(3), 1519-1539, 2006.

[110] WHITNEY, H., *Congruent graphs and the connectivity of graphs*. The Johns Hopkins University Press,Vol. 54, No. 1, pp. 150-168, 1932.

[111] HARARY, F., *Graph theory*. Addison-Wesley, Reading, Massachussetts, 1971.

[112] GAREY, M. R., JOHNSON, D. S., *Computers and intractability: a guide to NP-completedness*. W.H. Freeman, 1979.

[113] ARVIND, V., THORAN, J., *Isomorphism testing: perspective and open problems*. The Computational Complexity Column. Bulletin of the European Association for Theoretical Computer Science. 86, 66-84, 1985.

[114] MCKAY, B., *Practical graph isomorphism*. Congressus Numerantium 30 (1) 45-87, 1981.

[115] CROSS, A. D. J., WILSON, R. C., HANCOCK, E. R., *Inexact graph matching using genetic search*. Pattern Recognition, 30, n. 6, 953-970, 1997.

[116] DING, H., HUANG, Z., *Isomorphism identification of graphs: especially for the graphs of kinematic chains*. Mechanism and Machine Theory 44, 122-139, 2009.

[117] BOAVENTURA-NETTO, P. O., ABREU, N. M. M., *Quadratic assignment problem: a polynomial expression for the variance of the solutions*. Actas de Resumenes Extendidos, 408-413, I ELIO/Optima 97, Concepción, Chile, 1997.

[118] GRAVES, G. W., WHINSTON, A. B., *An algorithm for the quadratic assignment problem*. Management Science 17, 453-471, 1970.

[119] GRAVES, G. W., WHINSTON, A. B., *On the use of flow dominance in complexity measures for facility layout problems*. Interational Jornal of Production Research 23, 97-108, 1985.

[120] ANGEL, R., ZISSIMOPOULOS, V., *On the quality of the local search for the quadratic assignment problem*. Discrete Applied Mathematics 82, 15-25, 1998.

[121] BOAVENTURA-NETTO, P. O., LEITE, L. S. B. S., *O uso das variâncias do pqa na caracterização do isomorfismo de grafos*. XXXI Congresso Nacional de de Matemática Aplicada e Computacional - CNMAC - Pará, 2008.

[122] EULER, L., *Solutio problematis ad geometriam situs pertinentis.*. Commend. Academiae Sci. I. Petropolitanae 8, 128-140, 1736.

[123] NISHIZEKI, T., CHIBA, N., *Planar Graphs: Theory and Algorithms*. Elsevier Science Publishers, Amsterdam, Holanda, 1988.

[124] MELO, V. A., BOAVENTURA-NETTO, P., HAHN, P., et al., *Graph isomorphism and QAP variances*. Studia Informatica Universalis 8 (2), 209-234, Paris, França, 2010.

[125] MELO, V. A., BOAVENTURA-NETTO, P., *Investigations on the planar graph isomorphism through QAP variances (abstract)*. Technical Sessions, ALIO-INFORMS 2010 International Meeting, p. 113, Buenos Aires, Argentina, 2010.

Apêndice A

Exemplos de aplicação do método Condorcet

Fazendo uso do método Condorcet, descrito em 4.4.1 (página 47), mostramos a seguir sua aplicação em 2 exemplos para cada tipo de classe de instância usada neste trabalho.

Como este método foi aplicado apenas em um subconjunto de instâncias dos três primeiros tipos de comparações (*comparacao1, comparacao2, comparacao3*) e não como uma avaliação completa, falta uma totalização que possa fornecer resultados adicionais, como em Moreira (32).

A.1 Testes de desempenho com controle único de memória

A.1.1 Classe 1 - Instâncias Aleatórias com Distâncias e Fluxos Uniformemente Distribuídos (47 instâncias)

Controle Único - Classe 1

Tai25a	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	1	-	-	-	-	-	1	-	1	-	3	30
[a,c]	1	-	-	-	-	1	1	-	-	-	3	30
[a,d]	1	1	-	-	1	1	1	1	1	-	7	70
[a,e]	1	-	-	-	-	1	1	-	-	-	3	30
[b,c]	-	-	-	-	-	1	-	-	1	-	2	20
[b,d]	-	1	-	-	1	1	-	1	-	-	4	40
[b,e]	-	-	1	-	-	1	-	-	1	1	4	40
[c,d]	-	1	-	-	1	-	-	1	1	-	4	40
[c,e]	-	-	1	-	-	-	-	-	-	1	2	20
[d,e]	-	1	-	-	1	-	-	1	1	-	4	40
Dist	4	4	2	0	4	6	4	4	6	2		
Perc	40	40	20	0	40	60	40	40	60	20		

Tai80b	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	1	-	-	-	-	-	-	-	-	1	10
[a,c]	-	1	-	-	-	-	-	-	-	-	1	10
[a,d]	-	1	1	-	1	1	-	-	1	1	6	60
[a,e]	-	1	-	-	1	-	-	-	-	-	2	20
[b,c]	-	-	-	-	-	-	-	-	-	-	0	0
[b,d]	-	-	1	-	1	1	-	1	1	1	6	60
[b,e]	-	-	-	-	1	-	-	-	-	-	1	10
[c,d]	-	-	1	-	1	1	-	1	1	1	6	60
[c,e]	-	-	-	-	1	-	-	-	-	-	1	10
[d,e]	-	-	1	-	-	1	-	1	1	1	5	50
Dist	0	4	4	0	6	4	0	3	4	4		
Perc	0	40	40	0	60	40	0	30	40	40		

Tabela A.1: Comparação 1 - Classe 1 - Método Condorcet

Na Classe 1 de instâncias, para o controle único, observa-se que as maiores diferenças entre os algoritmos encontram-se entre as versões propostas com memória. Na instância de menor ordem (Tai25a), observa-se que isto ocorre principalmente com pares de algoritmos 2 e 4 e 3 e 5, mais precisamente com os pares de tipo de controle (vértice ou movimento). Para esta instância, observa-se que os algoritmos 1 e 5 são equivalentes, frente a todos os pares de critérios.

Em relação aos critérios de avaliação, percebe-se que todas as comparações envolvendo o critério d (tempo médio de execução) com os demais critérios apresentaram uma maior discriminação. Neste caso, a menor discordância deve corresponder ao melhor caso.

Para $Tai80b$ os algoritmos 1, 2 e 5 mostram equivalência, a maior discordância ocorre entre 2 e 3.

A.1.2 Classe 2 - Fluxos Aleatórios em Grades (30 instâncias)

Controle Único - Classe 2

Nug30	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	-	-	-	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	1	-	-	1	1	1	1	1	-	1	7	70
[a,e]	1	-	-	-	-	1	1	-	-	-	3	30
[b,c]	-	-	-	-	-	-	-	-	-	-	0	0
[b,d]	-	-	-	-	-	-	-	-	-	-	0	0
[b,e]	-	-	-	-	-	-	-	-	-	-	0	0
[c,d]	1	-	-	1	1	1	1	1	-	1	7	70
[c,e]	1	-	-	-	-	1	1	-	-	-	3	30
[d,e]	-	-	-	1	1	-	-	1	-	1	4	40
Dist	4	0	0	3	3	4	4	3	0	3		
Perc	40	0	0	30	30	40	40	30	0	30		

Sko42	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	1	-	1	-	-	-	-	-	-	2	20
[a,c]	-	-	-	1	-	-	-	-	-	-	1	10
[a,d]	-	1	1	1	1	1	1	1	-	1	8	80
[a,e]	-	1	1	1	1	-	1	-	-	-	5	50
[b,c]	-	1	-	-	-	-	-	-	-	-	1	10
[b,d]	-	-	1	-	1	1	-	1	1	1	6	60
[b,e]	-	-	1	-	1	-	-	-	1	-	3	30
[c,d]	-	1	1	-	1	1	1	1	1	1	8	80
[c,e]	-	1	1	-	1	-	1	-	1	-	5	50
[d,e]	-	-	-	-	-	1	-	1	-	1	3	30
Dist	0	6	6	4	6	4	4	4	4	4		
Perc	0	60	60	40	60	40	40	40	40	40		

Tabela A.2: Comparação 1 - Classe 2 - Método Condorcet

Ainda na comparação 1, porém na classe 2, nota-se para ambas as intâncias que o critério *d* (tempo médio de execução) mostrou uma grande discordância em relação aos critérios a (10 - VOMVC) e c (erro). Para as duas instâncias (como em geral para a Classe 2), o VNS básico já produziu bons resultados, os aperfeiçoamentos feitos não tendo produzido melhorias significativas.

A.1.3 Classe 3 - Problemas da Vida Real (38 instâncias)

Controle Único - Classe 3

Esc32b	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	-	-	-	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	-	-	-	-	-	-	-	-	-	-	0	0
[a,e]	-	-	-	-	-	-	-	-	-	-	0	0
[b,c]	-	-	-	-	-	-	-	-	-	-	0	0
[b,d]	-	-	-	-	-	-	-	-	-	-	0	0
[b,e]	-	-	-	-	-	-	-	-	-	-	0	0
[c,d]	-	-	-	-	-	-	-	-	-	-	0	0
[c,e]	-	-	-	-	-	-	-	-	-	-	0	0
[d,e]	-	-	-	-	-	-	-	-	-	-	0	0
Dist	0	0	0	0	0	0	0	0	0	0		
Perc	0	0	0	0	0	0	0	0	0	0		

Kra30a	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	1	-	1	-	-	1	-	-	-	-	3	30
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	1	1	1	1	-	1	-	1	-	1	7	70
[a,e]	1	-	-	1	-	1	-	-	-	1	4	40
[b,c]	1	-	1	-	-	1	-	-	-	-	3	30
[b,d]	-	1	-	1	-	-	1	1	-	1	5	50
[b,e]	-	-	1	1	-	-	-	-	-	1	3	30
[c,d]	1	1	1	1	-	1	1	1	-	1	8	80
[c,e]	1	-	-	1	-	1	-	-	-	1	4	40
[d,e]	-	1	1	-	-	-	1	1	-	-	4	40
Dist	6	4	6	6	0	6	3	4	0	6		
Perc	60	40	60	60	0	60	30	40	0	60		

Tabela A.3: Comparação 1 - Classe 3 - Método Condorcet

Na classe 3, com a instância $Esc32b$ não foi possível identificar nada relevante, porém com a instância $Kra30a$, novamente se vê uma discrepância entre o critério d e os demais.

As maiores discordâncias foram entre 1 e 2, 1 e 4, 1 e 5, 2 e 4 e 4 e 5 (onde 1 figura 3 vezes e 4 também). De fato, na avaliação geral da Classe 2, estes algoritmos obtiveram menor peso, enquanto 2 e 5 foram os melhores. A técnica não encontrou, para esta instância, discordância entre 2 e 3, nem entre 2 e 5.

A.1.4 Classe 4 - Instâncias Aleatórias Semelhantes aos Problemas da Vida Real (16 instâncias)

Controle Único - Classe 4

121

Lipa40b	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	-	-	-	1	1	1	1	-	1	5	50
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	1	1	1	1	1	1	1	-	-	1	8	80
[a,e]	-	-	-	-	1	1	1	1	-	1	5	50
[b,c]	-	-	-	-	1	1	1	1	-	1	5	50
[b,d]	1	1	1	1	-	-	-	1	-	-	5	50
[b,e]	-	-	-	-	-	-	-	-	-	-	0	0
[c,d]	1	1	1	1	1	1	1	-	-	1	8	80
[c,e]	-	-	-	-	1	1	1	1	-	1	5	50
[d,e]	1	1	1	1	-	-	-	1	-	-	5	50
Dist	4	4	4	4	6	6	6	6	0	6		
Perc	40	40	40	40	60	60	60	60	0	60		

127

Lipa70b	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	-	-	-	1	1	-	-	-	-	2	20
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	-	1	-	1	1	1	1	-	1	1	7	70
[a,e]	-	1	1	-	1	1	-	-	-	-	4	40
[b,c]	-	-	-	-	1	1	-	-	-	-	2	20
[b,d]	-	1	-	1	-	-	1	-	1	1	5	50
[b,e]	-	1	1	-	-	-	-	-	-	-	2	20
[c,d]	-	1	-	1	1	1	1	-	1	1	7	70
[c,e]	-	1	1	-	1	1	-	-	-	-	4	40
[d,e]	-	-	1	1	-	-	1	-	1	1	5	50
Dist	0	6	4	4	6	6	4	0	4	4		
Perc	0	60	40	40	60	60	40	0	40	40		

Tabela A.4: Comparação 1 - Classe 4 - Método Condorcet

Como observado nas classes anteriores, o critério d mais uma vez é o principal responsável pela discriminação entre todos so critérios. Em relação aos algoritmos, para ambas as instâncias o algoritmo 2 (*AgitacaoMov*) mostrou maior discriminação entre os demais. Curiosamente, porém, não houve discordância entre ele e o VNS básico (Algoritmo 1), para esta instância; por outro lado, nota-se que foram os dois algoritmos de melhor resultado na avaliação por pesos.

A.1.5 Classe 5 - Instâncias Difíceis para as Metaheurísticas (37 instâncias)

Controle Único - Classe 5

dre90	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	-	-	-	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	-	-	-	-	-	-	-	-	-	-	0	0
[a,e]	-	-	-	-	-	-	-	-	-	-	0	0
[b,c]	-	-	-	-	-	-	-	-	-	-	0	0
[b,d]	-	-	-	-	-	-	-	-	-	-	0	0
[b,e]	1	-	-	-	1	1	1	-	-	-	4	40
[c,d]	-	-	-	-	-	-	-	-	-	-	0	0
[c,e]	1	-	-	-	1	1	1	-	-	-	4	40
[d,e]	-	-	-	-	-	-	-	-	-	-	0	0
Dist	2	0	0	0	2	2	2	0	0	0		
Perc	20	0	0	0	20	20	20	0	0	0		

tai75e05	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	-	-	-	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	1	1	1	1	-	-	-	-	-	-	4	40
[a,e]	-	1	-	-	-	-	-	-	-	-	1	10
[b,c]	-	-	-	-	-	-	-	-	-	-	0	0
[b,d]	1	1	1	1	-	-	-	-	-	-	4	40
[b,e]	-	1	-	-	1	-	-	1	1	1	5	50
[c,d]	1	1	1	1	-	-	-	-	-	-	4	40
[c,e]	-	1	-	-	1	-	-	1	1	1	5	50
[d,e]	1	-	1	1	-	-	-	-	-	-	3	30
Dist	4	6	4	4	2	0	0	2	2	2		
Perc	40	60	40	40	20	0	0	20	20	20		

Tabela A.5: Comparação 1 - Classe 5 - Método Condorcet

Para a classe 5, o critério e foi o determinante, principalmente para a instância $Dre90$. Para a instância $Tai74e05$, também aparece o critério d. Tal critério mostrou-se predominante em todas as instâncias observadas para o controle único de memória.

Para $Dre90$ o algoritmo 2, que obteve o melhor peso na outra avaliação, apresenta discordâncias apenas com o 3, que foi o segundo colocado. Neste caso, a dificuldade das instâncias se combina à variedade do seu comportamento, tornando difícil qualquer comparação baseada apenas em duas instâncias.

Já a instância $Tai75e05$ permite uma melhor discriminação: o VNS básico (Algoritmo 1) apresenta discordâncias em relação aos demais em diversos pares de critérios, sem que esses pares discordem muito entre eles. Isto deve corresponder ao seu desempenho claramente mais fraco para esta classe.

A.2 Testes de desempenho com controle duplo de memória

A.2.1 Classe 1 - Instâncias Aleatórias com Distâncias e Fluxos Uniformemente Distribuídos (47 instâncias)

Controle Duplo - Classe 1

tai25a	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	1	-	1	-	-	-	1	-	1	4	40
[a,c]	-	-	-	-	-	-	-	1	-	1	2	20
[a,d]	-	1	-	1	1	-	1	1	-	1	6	60
[a,e]	-	-	-	-	-	-	-	-	-	-	0	0
[b,c]	-	1	-	1	-	-	-	-	-	-	2	20
[b,d]	-	-	-	-	1	-	1	-	1	-	3	30
[b,e]	1	1	1	1	-	1	-	1	1	1	8	80
[c,d]	-	1	-	1	1	-	1	-	1	-	5	50
[c,e]	1	-	1	-	-	1	-	1	1	1	6	60
[d,e]	-	1	-	1	1	-	1	1	-	1	6	60
Dist	2	6	2	6	4	2	4	6	4	6		
Perc	20	60	20	60	40	20	40	60	40	60		

tai80b	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	-	-	-	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	-	-	-	-	-	-	-	-	-	-	0	0
[a,e]	-	-	-	-	-	-	-	-	-	-	0	0
[b,c]	-	-	-	-	-	-	-	-	-	-	0	0
[b,d]	-	-	-	-	-	-	-	-	-	-	0	0
[b,e]	-	-	-	-	-	-	-	-	1	-	1	10
[c,d]	-	-	-	-	-	-	-	-	-	-	0	0
[c,e]	-	-	-	-	-	-	-	-	1	-	1	10
[d,e]	-	-	-	-	-	-	-	-	-	-	0	0
Dist	0	0	0	0	0	0	0	0	2	0		
Perc	0	0	0	0	0	0	0	0	20	0		

Tabela A.6: Comparação 2 - Classe 1 - Método Condorcet

Na classe 1 do controle duplo, percebe-se para a instância $Tai25a$, que apenas os critérios a e e não apresentaram discordâncias, isto para qualquer um dos algoritmos. De fato, no conjunto da classe se observa uma proporcionalidade aproximada entre o número de soluções não $VOMVC$ e o tempo médio de estagnação.

Os algoritmos 3 e 4, mais fracos para as instâncias desta classe, apresentaram pequena discordância.

Para a instância $Tai80b$, houve comparatibilidade apenas entre os algoritmos 3 e 5, critérios b e e e c e e.

Tal como a avaliação por pesos em geral, estas instâncias mostram pequena discordância entre os algoritmos, que foram aproximadamente equivalentes frente a estas instâncias.

A.2.2 Classe 2 - Fluxos Aleatórios em Grades (30 instâncias)

Controle Duplo - Classe 2

nug30	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	-	-	-	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	1	-	-	-	1	1	1	-	-	-	4	40
[a,e]	1	-	1	-	1	1	1	1	-	1	7	70
[b,c]	-	-	-	-	-	-	-	-	-	-	0	0
[b,d]	-	-	-	-	-	-	-	-	-	-	0	0
[b,e]	-	-	-	-	-	-	-	-	-	-	0	0
[c,d]	1	-	-	-	1	1	1	-	-	-	4	40
[c,e]	1	-	1	-	1	1	1	1	-	1	7	70
[d,e]	-	-	1	-	-	-	-	1	-	1	3	30
Dist	4	0	3	0	4	4	4	3	0	3		
Perc	40	0	30	0	40	40	40	30	0	30		

sko42	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	1	1	-	1	-	-	1	-	-	1	5	50
[a,c]	-	1	-	1	-	-	1	-	-	1	4	40
[a,d]	1	1	1	-	1	-	-	-	-	-	4	40
[a,e]	1	1	1	1	1	-	-	1	-	-	6	60
[b,c]	1	-	-	-	-	-	-	-	-	-	1	10
[b,d]	-	-	1	1	1	-	1	-	-	1	5	50
[b,e]	-	-	1	-	1	1	1	1	-	1	6	60
[c,d]	1	-	1	1	1	-	1	-	-	1	6	60
[c,e]	1	-	1	-	1	1	1	1	-	1	7	70
[d,e]	-	-	-	1	-	1	-	1	-	-	3	30
Dist	6	4	6	6	6	3	6	4	0	6		
Perc	60	40	60	60	60	30	60	40	0	60		

Tabela A.7: Comparação 2 - Classe 2 - Método Condorcet

Na classe 2, para a instância $Nug30$ os critérios a e c, quando associados ao critério e foram responsáveis pelas maiores discordâncias.

Em ambas as instâncias, o método não demonstrou discordâncias entre os algoritmos 3 ($DuploMovVert$) e 5 ($DuploVertVert$), o que nos faz sugerir que, para esta classe, o segundo controle, (por vértice), talvez seja predominante na atuação do primeiro controle.

A.2.3 Classe 3 - Problemas da Vida Real (38 instâncias)

Controle Duplo - Classe 3

esc32b	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	-	-	-	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	-	-	-	-	-	-	-	-	-	-	0	0
[a,e]	-	-	-	-	-	-	-	-	-	-	0	0
[b,c]	-	-	-	-	-	-	-	-	-	-	0	0
[b,d]	-	-	-	-	-	-	-	-	-	-	0	0
[b,e]	-	-	-	-	-	-	-	-	-	-	0	0
[c,d]	-	-	-	-	-	-	-	-	-	-	0	0
[c,e]	-	-	-	-	-	-	-	-	-	-	0	0
[d,e]	-	-	-	-	-	-	-	-	-	-	0	0
Dist	0	0	0	0	0	0	0	0	0	0		
Perc	0	0	0	0	0	0	0	0	0	0		

kra30a	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	1	1	-	-	1	1	-	1	-	-	5	50
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	1	1	-	1	-	1	1	1	1	1	8	80
[a,e]	1	1	-	-	1	1	-	1	-	-	5	50
[b,c]	1	1	-	-	1	1	-	1	-	-	5	50
[b,d]	-	-	1	1	1	-	1	-	1	1	6	60
[b,e]	-	-	-	-	-	-	-	-	-	-	0	0
[c,d]	1	1	1	1	-	1	1	1	1	1	9	90
[c,e]	1	1	-	-	1	1	-	1	-	-	5	50
[d,e]	-	-	1	1	1	-	1	-	1	1	6	60
Dist	6	6	3	4	6	6	4	6	4	4		
Perc	60	60	30	40	60	60	40	60	40	40		

Tabela A.8: Comparação 2 - Classe 3 - Método Condorcet

A instância $Esc32b$ é de baixa dificuldade entre as demais Esc_{xx} de mesma ordem. Os algoritmos mostraram desempenho equivalente frente a todos os critérios.

A instância $Kra30a$ apresentou uma distribuição mais homogênea entre os algoritmos, ficando o critério d responsável pelas maiores discordancias entre os critérios.

Em particular o critério c (indicador de qualidade) mostrou forte discordância em relação ao tempo de execução (d) o que, neste caso, foi altamente influenciado pelo critério a (afastamento médio).

A.2.4 Classe 4 - Instâncias Aleatórias Semelhantes aos Problemas da Vida Real (16 instâncias)

Controle Duplo - Classe 4

lipa40b	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	-	-	-	-	-	1	-	-	-	1	10
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	1	1	-	1	-	1	-	1	-	1	6	60
[a,e]	-	-	-	-	-	-	1	-	1	-	2	20
[b,c]	-	-	-	-	-	-	1	-	-	-	1	10
[b,d]	1	1	-	1	-	1	1	1	-	1	7	70
[b,e]	-	-	-	-	1	-	-	-	1	-	2	20
[c,d]	1	1	-	1	-	1	-	1	-	1	6	60
[c,e]	-	-	-	-	1	-	1	-	1	-	3	30
[d,e]	1	1	-	1	1	1	1	1	1	1	9	90
Dist	4	4	0	4	3	4	6	4	4	4		
Perc	40	40	0	40	30	40	60	40	40	40		

lipa70b	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	-	-	-	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	1	1	1	1	1	1	-	-	1	1	8	80
[a,e]	1	1	1	1	1	-	-	-	1	1	7	70
[b,c]	-	-	-	-	-	-	-	-	-	-	0	0
[b,d]	1	1	1	1	1	1	1	-	1	1	9	90
[b,e]	1	1	1	1	1	-	1	-	1	1	8	80
[c,d]	1	1	1	1	1	1	1	-	1	1	9	90
[c,e]	1	1	1	1	1	-	1	-	1	1	8	80
[d,e]	-	-	-	-	-	1	-	-	-	-	1	10
Dist	6	6	6	6	6	4	4	0	6	6		
Perc	60	60	60	60	60	40	40	0	60	60		

Tabela A.9: Comparação 2 - Classe 4 - Método Condorcet

O critério d mais uma vez demonstra discordância entre os outros critérios na classe 4.

Para a instância *Lipa*70*b*, como o critério b (afastamento médio) influencia diretamente o critério c (indicador de qualidade), nos levar a supor que, o fato do algoritmo ter gasto mais tempo de execução (critério d), não indicou, necessariamente, que o afastamento médio (critério e), fosse menor. Outra possível causa, talvez seja a dificuldade maior desta instância em relação a *Lipa*40*b*, o que faz com que sua estagnação foi mais prematura.

A.2.5 Classe 5 - Instâncias Difíceis para as Metaheurísticas (37 instâncias)

Controle Duplo - Classe 5

dre90	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	-	-	-	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	-	-	-	-	-	-	-	-	-	-	0	0
[a,e]	-	-	-	-	-	-	-	-	-	-	0	0
[b,c]	-	-	-	-	-	-	-	-	-	-	0	0
[b,d]	-	-	-	-	-	-	-	-	-	-	0	0
[b,e]	-	1	-	1	1	-	-	1	1	-	5	50
[c,d]	-	-	-	-	-	-	-	-	-	-	0	0
[c,e]	-	1	-	1	1	-	-	1	1	-	5	50
[d,e]	-	-	-	-	-	-	-	-	-	-	0	0
Dist	0	2	0	2	2	0	0	2	2	0		
Perc	0	20	0	20	20	0	0	20	20	0		

tai75e05	[1,2]	[1,3]	[1,4]	[1,5]	[2,3]	[2,4]	[2,5]	[3,4]	[3,5]	[4,5]	Dist	Perc
[a,b]	-	-	-	-	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	-	-	-	-	0	0
[a,d]	1	1	1	1	-	-	-	-	-	-	4	40
[a,e]	-	-	1	-	-	-	-	-	-	-	1	10
[b,c]	-	-	-	-	-	-	-	-	-	-	0	0
[b,d]	1	1	1	1	-	-	-	-	-	-	4	40
[b,e]	-	-	1	-	-	1	-	1	1	1	5	50
[c,d]	1	1	1	1	-	-	-	-	-	-	4	40
[c,e]	-	-	1	-	-	1	-	1	1	1	5	50
[d,e]	1	1	-	1	-	-	-	-	-	-	3	30
Dist	4	4	6	4	0	2	0	2	2	2		
Perc	40	40	60	40	0	20	0	20	20	20		

Tabela A.10: Comparação 2 - Classe 5 - Método Condorcet

A classe 5, que é a mais difícil para as metaheurísticas, apresentou discordância apenas entre o critério e e o indicador de qualidade (critério c) que, por estar diretamente associado ao critério b, também apresentou discordância na instância $Dre90$. No entanto, de modo geral os critérios se apresentaram bastante concordantes. Os algoritmos com duplo controle também se mostraram concordantes, as maiores divergências tendo ocorrido entre eles e o VNS básico.

Tais obsevações demonstram que, para esta classe de instância, as variações de controle duplo de memória, não trouxeram diferenças tão significativas de desempenho.

A.3 Testes de desempenho com outras meta-heurísticas

A.3.1 Classe 1 - Instâncias Aleatórias com Distâncias e Fluxos Uniformemente Distribuídos (47 instâncias)

Outras Metaheurísticas - Classe 1

tai25a	[1,2]	[1,3]	[1,4]	[2,3]	[2,4]	[3,4]	Dist	Perc
[a,b]	-	1	-	-	-	-	1	16,67
[a,c]	-	1	-	-	-	-	1	16,67
[a,d]	1	1	-	1	1	-	4	66,67
[a,e]	-	1	-	-	-	1	2	33,33
[b,c]	-	-	-	-	-	-	0	0
[b,d]	1	-	-	1	1	-	3	50
[b,e]	-	-	1	-	-	1	2	33,33
[c,d]	1	-	-	1	1	-	3	50
[c,e]	-	-	1	-	-	1	2	33,33
[d,e]	1	-	1	1	1	1	5	83,33
Dist	4	4	3	4	4	4		
Perc	40	40	30	40	40	40		

tai80b	[1,2]	[1,3]	[1,4]	[2,3]	[2,4]	[3,4]	Dist	Perc
[a,b]	-	1	-	1	-	-	2	33,33
[a,c]	-	-	-	1	-	-	1	16,67
[a,d]	1	1	-	-	1	-	3	50
[a,e]	-	1	-	1	-	1	3	50
[b,c]	-	1	-	-	-	-	1	16,67
[b,d]	1	-	-	1	1	-	3	50
[b,e]	-	-	-	-	-	1	1	16,67
[c,d]	1	1	-	1	1	-	4	66,67
[c,e]	-	1	-	-	-	1	2	33,33
[d,e]	1	-	-	1	1	1	4	66,67
Dist	4	6	0	6	4	4		
Perc	40	60	0	60	40	40		

Tabela A.11: Comparação 3 - Classe 1 - Método Condorcet

Um fato interessante ocorrido com a instância $Tai25a$ foi o fato de que entre os algoritmos 1 e 3 predominou o critério a, entre 1 e 4, foi o critério e. Isto indica que o critério a, entre 1 e 3 é o inverso ao critério e, entre 1 e 4.

Como nos demais casos observados, os critério d e e ocorreram em maior número de discrepâncias que os demais.

A.3.2 Classe 2 - Fluxos Aleatórios em Grades (30 instâncias)

Outras Metaheurísticas - Classe 2

nug30	[1,2]	[1,3]	[1,4]	[2,3]	[2,4]	[3,4]	Dist	Perc
[a,b]	-	1	1	-	-	-	2	33,33
[a,c]	-	-	-	-	-	-	0	0
[a,d]	-	1	1	1	1	-	4	66,67
[a,e]	-	1	1	-	-	-	2	33,33
[b,c]	-	1	1	-	-	-	2	33,33
[b,d]	-	-	-	1	1	-	2	33,33
[b,e]	-	-	-	-	-	1	1	16,67
[c,d]	-	1	1	1	1	-	4	66,67
[c,e]	-	1	1	-	-	1	3	50
[d,e]	-	-	-	1	1	-	2	33,33
Dist	0	6	6	4	4	2		
Perc	0	60	60	40	40	20		

sko42	[1,2]	[1,3]	[1,4]	[2,3]	[2,4]	[3,4]	Dist	Perc
[a,b]	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	0	0
[a,d]	1	-	-	-	-	-	1	16,67
[a,e]	-	-	1	-	-	1	2	33,33
[b,c]	-	-	-	-	-	-	0	0
[b,d]	1	-	-	-	-	-	1	16,67
[b,e]	-	-	1	-	-	1	2	33,33
[c,d]	1	-	-	-	-	-	1	16,67
[c,e]	-	-	1	-	-	1	2	33,33
[d,e]	1	-	1	-	-	1	3	50
Dist	4	0	4	0	0	4		
Perc	40	0	40	0	0	40		

Tabela A.12: Comparação 3 - Classe 2 - Método Condorcet

Na comparação 3, para a instância $Nug30$, os únicos algoritmos que não apresentaram discordância foram 1 e 3. Os pares de critérios (a,d) e (c,d) (envolvendo número de soluções $VOMVC$, erro e tempo de processamento) foram os mais divergentes.

Para a instância $Sko42$ não houve divergência com três pares de algoritmos: 1 e 3, 2 e 3 e 2 e 4.

Em ambas instâncias, os critérios a e c não foram discordantes entre si, o que sugere que, para esta classe, o critério b tenha sido menos expressivo na composição do indicador de qualidade (critério c).

A.3.3 Classe 3 - Problemas da Vida Real (38 instâncias)

Outras Metaheurísticas - Classe 3

esc32b	[1,2]	[1,3]	[1,4]	[2,3]	[2,4]	[3,4]	Dist	Perc
[a,b]	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	0	0
[a,d]	-	1	1	1	1	-	4	66,67
[a,e]	-	-	-	-	-	1	1	16,67
[b,c]	-	-	-	-	-	-	0	0
[b,d]	-	1	1	1	1	-	4	66,67
[b,e]	-	-	-	-	-	1	1	16,67
[c,d]	-	1	1	1	1	-	4	66,67
[c,e]	-	-	-	-	-	1	1	16,67
[d,e]	-	1	1	1	1	1	5	83,33
Dist	0	4	4	4	4	4		
Perc	0	40	40	40	40	40		

kra30a	[1,2]	[1,3]	[1,4]	[2,3]	[2,4]	[3,4]	Dist	Perc
[a,b]	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	0	0
[a,d]	-	1	1	1	-	1	4	66,67
[a,e]	-	-	1	-	-	1	2	33,33
[b,c]	-	-	-	-	-	-	0	0
[b,d]	-	1	1	1	-	1	4	66,67
[b,e]	-	-	1	-	-	1	2	33,33
[c,d]	-	1	1	1	-	1	4	66,67
[c,e]	-	-	1	-	-	1	2	33,33
[d,e]	-	1	-	1	-	-	2	33,33
Dist	0	4	6	4	0	6		
Perc	0	40	60	40	0	60		

Tabela A.13: Comparação 3 - Classe 3 - Método Condorcet

Nesta classe, ambas as instâncias ($Esc32b$ e $Kra30a$) foram dominadas pelo critério d, sendo que os algoritmos 1 e 2 não apresentaram discrepância alguma entre eles. De fato, o desempenho foi bom para todos os algoritmos, menos o 4.

Na instância $Kra30a$, observa-se que o algoritmo 2 (*TabuTaillard*) mostrou discordância apenas com o algoritmo 3 (*IlsStutzle*). O motivo mais provável talvez seja o baixíssmo tempo de execução médio (critério d) obtido por este segundo algoritmo.

A.3.4 Classe 4 - Instâncias Aleatórias Semelhantes aos Problemas da Vida Real (16 instâncias)

Outras Metaheurísticas - Classe 4

lipa40b	[1,2]	[1,3]	[1,4]	[2,3]	[2,4]	[3,4]	Dist	Perc
[a,b]	-	-	-	-	-	1	1	16,67
[a,c]	-	-	-	-	-	1	1	16,67
[a,d]	-	1	1	-	-	-	2	33,33
[a,e]	-	-	-	-	-	1	1	16,67
[b,c]	-	-	-	-	-	-	0	0
[b,d]	-	1	1	-	-	1	3	50
[b,e]	-	-	-	-	-	-	0	0
[c,d]	-	1	1	-	-	1	3	50
[c,e]	-	-	-	-	-	-	0	0
[d,e]	-	1	1	-	-	1	3	50
Dist	0	4	4	0	0	6		
Perc	0	40	40	0	0	60		

lipa70b	[1,2]	[1,3]	[1,4]	[2,3]	[2,4]	[3,4]	Dist	Perc
[a,b]	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	0	0
[a,d]	-	-	-	-	-	-	0	0
[a,e]	-	-	-	-	-	-	0	0
[b,c]	-	-	-	-	-	-	0	0
[b,d]	-	-	-	-	-	-	0	0
[b,e]	-	-	-	-	-	-	0	0
[c,d]	-	-	-	-	-	-	0	0
[c,e]	-	-	-	-	-	-	0	0
[d,e]	-	-	-	-	-	-	0	0
Dist	0	0	0	0	0	0		
Perc	0	0	0	0	0	0		

Tabela A.14: Comparação 3 - Classe 4 - Método Condorcet

Para *Lipa*40*b*, novamente o critério *d* foi mais relevante, porém não apresentando diferença alguma entre os algoritmos 1 e 2, 2 e 3 e 2 e 4.

A instância *Lipa*70*b* não mostrou discordância alguma entre os algoritmos e critério observados. O que se verificou é que as posições obtidas em cada critério, para cada algoritmo, não se alteraram quando ordenadas, cuja ordem dos algoritmos foi: 2, 1, 3 e 4, respectivamente.

A.3.5 Classe 5 - Instâncias Difíceis para as Metaheurísticas (37 instâncias)

Outras Metaheurísticas - Classe 5

dre90	[1,2]	[1,3]	[1,4]	[2,3]	[2,4]	[3,4]	Dist	Perc
[a,b]	-	-	-	-	-	-	0	0
[a,c]	-	-	-	-	-	-	0	0
[a,d]	-	-	-	-	-	-	0	0
[a,e]	-	-	-	-	-	-	0	0
[b,c]	-	-	-	-	-	-	0	0
[b,d]	-	-	-	-	-	-	0	0
[b,e]	-	-	-	1	1	1	3	50
[c,d]	-	-	-	-	-	-	0	0
[c,e]	-	-	-	1	1	1	3	50
[d,e]	-	-	-	-	-	-	0	0
Dist	0	0	0	2	2	2		
Perc	0	0	0	20	20	20		

tai75e05	[1,2]	[1,3]	[1,4]	[2,3]	[2,4]	[3,4]	Dist	Perc
[a,b]	1	-	1	1	-	1	4	66,67
[a,c]	1	-	1	1	-	1	4	66,67
[a,d]	1	-	1	1	-	-	3	50
[a,e]	1	-	1	1	-	1	4	66,67
[b,c]	-	-	-	-	-	-	0	0
[b,d]	-	1	-	-	-	1	2	33,33
[b,e]	-	-	-	-	-	-	0	0
[c,d]	-	1	-	-	-	1	2	33,33
[c,e]	-	-	-	-	-	-	0	0
[d,e]	-	1	-	-	-	1	2	33,33
Dist	4	3	4	4	0	6		
Perc	40	30	40	40	0	60		

Tabela A.15: Comparação 3 - Classe 5 - Método Condorcet

Para a isntância $Dre90$, nenhum dos algoritmos um mostrou desempenho melhor diferente em relação ao algoritmo 1, fato somente observado, através do critério e, nos demais algoritmos.

Na instância $Tai75e05$ o critério a preveleceu entre os algoritmos 1 e 2, 1 e 4, 2 e 3 e 3 e 4, o que pela dificuldade da instância gerou grande diferença entre o número de instâncias que alcançou o $VOMVC$.

Ao final, o que se percebe é que, independentemente de classe ou comparação, para as instâncias mais díficeis ($n \geq 70$) da classe, os critérios d (tempo médio de execução) e e (complemento do tempo médio de estagnação) sempre estão presentes e predominam nas discordâncias.

Por último, a análise dos resultados obtidos por este método pode se mostrar díficil, ao se observarem duas instâncias, dada a grande diferença de performance entre os algoritmos; isto porque, sendo o método *Condorcet* semi-qualitativo, ele não discrimina diferenças pequenas de grandes, considerando apenas a presença de diferenças e do sentido delas.