Universidade Federal Fluminense

MÁRIO MESTRIA

Metaheurísticas Híbridas para a Resolução do Problema do Caixeiro Viajante com Grupamentos

NITERÓI

2011

MÁRIO MESTRIA

Metaheurísticas Híbridas para a Resolução do Problema do Caixeiro Viajante com Grupamentos

Algoritmos e Otimização

Orientadores:

Luiz Satoru Ochi
Simone de Lima Martins

Universidade Federal Fluminense

NITERÓI

2011

Metaheurísticas Híbridas para a Resolução do Problema do Caixeiro Viajante com Grupamentos

Mário Mestria

Tese de Doutorado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Doutor em Computação.

Aprovada por:

Prof. Luiz Satoru Ochi / IC-UFF (Presidente)

Profa. Simone de Lima Martins / IC-UFF

Prof. Fábio Protti / IC-UFF

Prof. Virgílio José Martins Ferreira Filho / COPPE-UFRJ

Profa. Márcia Helena Costa Fampa / COPPE-UFRJ

Prof. Carlos Alberto de J. Martinhon / IC-UFF

Niterói, 13 de janeiro de 2011.

Dedico a minha esposa Edna e aos meus filhos Felipe e Thais.

Agradecimentos

A Deus por ter chegado até este momento e aos meus amigos do IC, em especial a Tiago Neves, André Renato, Jacques, Diego Brandão, Jonivan, Warley, Copetti, Renato, Sanderson, Cristiano, Marcelo Zamith, Cristiane, Idálmis, Luciana Gonçalves, Ádria, Anand, Puca, Renata e Marcus Guelpeli e a tantos outros que no momento não me recordo. Aos meus orientadores Luiz Satoru Ochi e Simone de Lima Martins. Ao Rafael e Carlos da equipe de suporte. As secretárias da Pós, Ângela Regina de M. C. Dias e Teresa Cristina da Silva Cancela e a nossa ex-secretária Maria de Freitas. Aos Professores Aura Conci, Alexandre Plastino e Célio Albuquerque. Aos Professores da UFES, Arlindo Gomes de Alvarenga, Orivaldo de Lira Tavares e Mário Sarcinelli Filho. Ao Professor Geraldo Robson Mateus da UFMG. Ao professores Jean-Yves Potvin, *Département d'informatique et de recherche opérationnelle, Université de Montréal*, pelo material bibliográfico e a David S. Johnson - *AT&T Labs - Research*, pelos códigos para gerar instâncias do tipo 2. Ao professor Ye Cheng e ao Chao Ding, *Department of Industrial Engineering, Tsinghua University, Beijing*, pelas especificações detalhadas de seus trabalhos. A minha família, o meu pai Nicola Mestria (*In Memoriam*), a minha mãe Assumpta Maria Cypriano Mestria, minhas duas irmãs Argenzia e Zenaide, a minha sobrinha Bianca e meus sobrinhos Henrique e Eduardo. Aos Professores do IFES, Reginaldo, Guilherme, Tedesco, Cabelino, Pinotti, Alan, Ailton, Érika e ao Rodrigues.

Resumo

Este trabalho propõe diversas heurísticas para resolver o Problema do Caixeiro Viajante com Grupamentos (PCVG). O PCVG é uma generalização do Problema do Caixeiro Viajante (PCV), em que os vértices são divididos em grupos e todos os vértices de cada grupo devem ser visitados de forma contígua. As abordagens desenvolvidas foram baseadas nas metaheurísticas GRASP e *Iterated Local Search* (ILS). Uma versão corresponde ao GRASP tradicional, três incluem módulos de Reconexão de Caminhos (RC), três combinam a RC e o Método de Descida em Vizinhança Variável (*VND*), enquanto outras duas heurísticas híbridas combinam VND com ILS. As heurísticas propostas foram testadas em instâncias com até 2000 vértices e número de grupos variando de dois a 150 vértices. Os resultados computacionais mostraram que as heurísticas híbridas que utilizam ILS e VND apresentam os melhores resultados. O desempenho dos algoritmos propostos foi comparado com um algoritmo exato usando o *software* CPLEX e um Algoritmo Genético da literatura.

Palavras-chave: Metaheurísticas, Algoritmos com Memória Adaptativa, Heurísticas Híbridas, Problema do Caixeiro Viajante com Grupamentos.

Abstract

This work proposes several heuristics to solve the Clustered Traveling Salesman Problem (CTSP). The CTSP is a generalization of the Traveling Salesman Problem (TSP) in which the vertices are partitioned into clusters and all vertices of each cluster must be visited continuously. The developed approaches were based on the GRASP metaheuristic and Iterated Local Search (ILS). One version corresponds to the traditional GRASP, three includes Path Relinking (PR) modules, three combines the PR and the Variable Neighborhood Descent (VND), while other two hybrid heuristics combines the VND and ILS. The proposed heuristics were tested in instances with up to 2000 vertices and clusters varying between two to 150 vertices. The computational results showed that the hybrid heuristics using ILS and VND achieve the best results. The performance of the proposed algorithms was compared with an exact algorithm by using the CPLEX software and a Genetic Algorithm of literature.

Keywords: Metaheuristics, Adaptive Memory Algorithms, Hybrid Heuristics, Clustered Traveling Salesman Problem.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviaturas e Siglas

PCV	:	Problema do Caixeiro Viajante;
PCVG	:	Problema do Caixeiro Viajante com Grupamento;
PCVB	:	Problema do Caixeiro Viajante com *BackHauls*;
GRASP	:	Greedy Randomized Adaptive Search Procedures;
VNS	:	Variable Neighborhood Search;
VND	:	Variable Neighborhood Descent;
ILS	:	Iterated Local Search;
GENI	:	Generalized Insertion Procedure;
US	:	Unstringing and Stringing;
GENIUS	:	GENI+US;
IMPP	:	Inserção mais Proxima com as Arestas inter-grupos Penalizadas;
IMPnP	:	Inserção mais Proxima com as Arestas inter-grupos não Penalizadas;
VND-P	:	VND combinado com a heurística construtiva IMPP;
VND-nP	:	VND combinado com a heurística construtiva IMPnP;
ILS-VND-P	:	ILS com busca local VND e heurística construtiva IMPP;
ILS-VND-nP	:	ILS com busca local VND e heurística construtiva IMPnP;
VMPP	:	Vizinho mais Próximo com as Arestas Penalizadas;
VMPnP	:	Vizinho mais Próximo com as Arestas não Penalizadas;
IMPP	:	Inserção mais Próxima com as Arestas Penalizadas;
IMPnP	:	Inserção mais Próxima com as Arestas não Penalizadas;
IMBP	:	Inserção mais Barata com as Arestas Penalizadas;
IMBnP	:	Inserção mais Barata com as Arestas não Penalizadas;
GENIV	:	GENI e vértice como elemento do GRASP escolhido aleatório;
GENIP	:	GENI e a posição como elemento do GRASP escolhido aleatório;
AG	:	Algoritmo Genético;
RA	:	Recombinação de Arestas;
LRC	:	Lista Restrita de Candidatos;
CE	:	Conjunto Elite;
RC	:	Reconexão de Caminhos;
DPA	:	Distribuição de Probabilidade Acumulativa;

Capítulo 1

Introdução

O Problema do Caixeiro Viajante (PCV) é um problema muito estudado na área de Otimização Combinatória com diversos trabalhos desenvolvidos para sua resolução (DANTZIG; FULKERSON; JOHNSON, 1954), (BELLMORE; NEMHAUSER, 1968), (HELD; KARP, 1970), (HELD; KARP, 1971), (LIN; KERNIGHAN, 1973), (LAWLER et al., 1985), (REINELT, 1994), (FREDMAN et al., 1995), (JOHNSON; McGEOCH, 1997), (JOHNSON; McGEOCH, 2002) e (APPLEGATE; COOK; ROHE, 2003). Suas aplicações encontram-se nas áreas de sistemas de manufatura, transportes, escalonamento de tarefas, problemas de coletas e entregas de produtos, roteiros de veículos, localizações de facilidades, roteamentos, redes de telecomunicações, dentre outros.

Uma generalização do PCV é conhecida como o Problema do Caixeiro Viajante com Grupamentos (PCVG). Este problema surgiu quando foi estudado um problema prático envolvendo roteamento automático em um sistema de armazenagem (CHISMAN, 1975).

Com a mesma terminologia utilizada para o PCV que associa cada vértice do grafo a uma cidade, pode-se definir o PCVG da seguinte forma. O conjunto de cidades é particionado em k grupos disjuntos e não vazios e o caixeiro viajante deve sair de uma cidade origem, visitar todas as cidades uma única vez antes de retornar a sua cidade origem, e todas as cidades de cada grupo devem ser visitadas contiguamente. Obviamente quando o número de grupos for igual a um (1), o PCVG se reduz ao PCV. Desta forma pode-se afirmar que o PCVG também pertence à classe $\mathcal{NP}$-difícil.

Especificamente o PCVG pode ser subdividido em dois subproblemas: o primeiro é encontrar a ordem de visitas das cidades em cada grupo e o segundo a ordem de visita dos grupos. Os dois subproblemas deverão ser resolvidos simultaneamente e estes não são independentes.

O PCVG aparece na literatura em duas versões. Na primeira mais explorada, supõe-se que a ordem de visitas aos grupos já é previamente definida. No segundo caso, para o qual existe reduzido número de trabalhos, essa ordem de visitas não é definida previamente e esta escolha é deixada para o algoritmo. A segunda versão é mais complexa, pois na busca da melhor sequência de visitas às cidades de cada grupo, deve-se também analisar a melhor ordem de visitas aos grupos que correspondem a encontrar os vértices de entrada e saída de cada grupo. Assim, é necessário achar simultaneamente uma ordem de visitas das cidades em cada grupo e a ordem de visita dos grupos de modo a minimizar a distância total percorrida. A tese tem como objetivo apresentar algoritmos heurísticos para este segundo caso aplicado ao PCVG.

Algumas aplicações modeladas como PCVG são encontradas em roteamento automático para armazéns e planejamento da produção (LOKIN, 1979), sistemas de manufaturas (LAPORTE; LOPES; SOUMIS, 1998), despacho de veículos de emergência (WEINTRAUB et al., 1999), desfragmentação de discos rígidos (LAPORTE; PALEKAR, 2002), na fase de pós-otimização para problemas *timetabling* (BALAKRISHNAN; LUCENA; WONG, 1992) e em transações comerciais envolvendo supermercados, lojas e fornecedores de mercadorias (GHAZIRI; OSMAN, 2003).

A motivação para trabalhar com o PCVG deve-se ao fato que este problema é uma variante do PCV com diferentes aplicações reais e a maioria dos trabalhos relacionados na literatura partem do princípio de que já existe definida uma ordem de visita aos grupos (ANILY; BRAMEL; HERTZ, 1999), (GENDREAU; LAPORTE; POTVIN, 1994), (LAPORTE; POTVIN; QUILLERET, 1996), (POTVIN; GUERTIN, 1995), informação esta, que geralmente não é conhecida em muitos problemas reais. Este trabalho trata o caso geral, onde o PCVG não considera nenhuma ordem de visitas dos grupos. Para este caso são encontrados na literatura métodos utilizando Algoritmos Genéticos (DING; CHENG; HE, 2007) e (POTVIN; GUERTIN, 1996) e Algoritmos α-aproximados (ARKIN; HASSIN; KLEIN, 1997) e (GUTTMANN-BECK et al., 2000). Um outro aspecto incentivador do presente trabalho é o fato das heurísticas existentes para o PCVG serem todas bastante simples, ou são heurísticas clássicas de construção e/ou busca local, ou metaheurísticas básicas sem incorporar módulos que tem se mostrado fundamentais no desempenho final destes algoritmos tais como a Reconexão de Caminhos (RC) (RESENDE; RIBEIRO, 2002) e métodos de buscas locais mais sofisticadas (HANSEN; MLADENOVIĆ, 2003). Observa-se que a inclusão destes módulos em algumas metaheurísticas (BASTOS; OCHI; MACAMBIRA, 2005) e (SILVA et al., 2007) têm melhorado bastante seu desempenho. Estas estratégias de nosso conhecimento ainda não foram exploradas para a resolução do PCVG.

Dentre as metaheurísticas existentes na literatura, uma que tem se destacado é "Greedy Randomized Adaptive Search Procedures - GRASP" (FEO; RESENDE, 1995), (PITSOULIS; RESENDE, 2002), (RESENDE; RIBEIRO, 2002) e (RESENDE et al., 2008). GRASP é um algoritmo iterativo no qual cada iteração consiste de duas fases: uma de construção e outra de busca local. Na fase de construção, uma solução viável é construída, e na fase de busca local sua vizinhança é investigada até um mínimo local ser encontrado. A melhor solução obtida ao final de um determinado número de iterações é retornada como a solução do algoritmo.

Neste trabalho, coloca-se como um dos objetivos verificar o impacto ao se incluir na metaheurística GRASP, módulos de memória adaptativa através da reconexão de caminhos (GLOVER; LAGUNA; MARTÍ, 2000), (GLOVER; LAGUNA; MARTÍ, 2004) e (MARTÍ; LAGUNA; GLOVER, 2006) e/ou uma busca local mais eficiente utilizando conceitos da metaheurística *Variable Neighborhood Search* (VNS) (HANSEN; MLADENOVIĆ, 2003). Neste contexto foram desenvolvidas sete versões utilizando a metaheurística GRASP. A primeira versão é um GRASP tradicional. Nas demais versões, um módulo de busca intensiva que utiliza conceitos de memória adaptativa através do uso de reconexão de caminhos é incorporado ao GRASP.

Uma outra metaheurística que vem se mostrando eficiente para obter soluções aproximadas de boa qualidade para problemas de otimização combinatória é o método *Iterated Local Search* (ILS) (LOURENÇO; MARTIN; STÜTZLE, 2002). Sua estrutura canônica é formada por quatro módulos principais: construção da solução, busca local, perturbação e critério de aceitação. Destaque das características do ILS são sua simplicidade, facilidade de implementação e robustez (LOURENÇO; MARTIN; STÜTZLE, 2002).

Na literatura encontra-se o uso do ILS para uma vasta gama de problemas desde acadêmicos até aplicações comerciais, industriais e prestação de serviços (CODENOTTI et al., 1996), (STÜTZLE, 1998), (SOUZA et al., 2005), (STÜTZLE, 2006), (CORDEAU; LAPORTE; PASIN, 2008), (DONG; HUANG; CHEN, 2008), (ERDOGAN; CORDEAU; LAPORTE, 2008), (HASHIMOTO; YAGIURA; IBARAKI, 2008), (IBARAKI et al., 2008) e (SUBRAMANIAN; CABRAL, 2008). Uma das grandes vantagens do ILS está na estrutura modular de cada componente. Assim, podem-se desenvolver módulos independentes e incorporar versões mais sofisticadas a cada um com a finalidade de se obter soluções de melhor qualidade. O sucesso do ILS depende fundamentalmente do projeto de cada módulo (LOURENÇO; MARTIN; STÜTZLE, 2002).

Neste trabalho desenvolveram-se para o PCVG diversos métodos de buscas locais,

estratégias de perturbação e tipos de critério de aceitação que foram incluídos nos módulos da metaheurística ILS em várias formas para verificar a qualidade da solução final produzida. Neste sentido, duas versões foram desenvolvidas utilizando a metaheurística ILS.

A contribuição principal deste trabalho é o fato de se utilizar as metaheurísticas GRASP com memória adaptativa e ILS pela primeira vez para o PCVG. Outras contribuições decorrentes são: apresentar diversas heurísticas construtivas e de buscas locais para geração de soluções do PCVG e criar um conjunto de instâncias de domínio público. Neste ponto observa-se que nos trabalhos de (JONGENS; VOLGENANT, 1985) até (LAPORTE; POTVIN; QUILLERET, 1996) existem regras para criar as instâncias, mas as mesmas não estão disponibilizadas. O conjunto de instâncias geradas para o PCVG são instâncias euclidianas e simétricas construídas de seis formas diferentes.

No Capítulo 2 apresenta-se a definição do problema, descrevem-se os trabalhos relacionados existentes, as formulações matemáticas para o PCVG e as técnicas para solucionar o PCVG com a formulação Dantzig/Chisman (DANTZIG; FULKERSON; JOHNSON, 1954) e (CHISMAN, 1975). Os Capítulos 3 e 4 descrevem as heurísticas propostas para o PCVG. Os resultados computacionais, as análises de desempenho das heurísticas, as comparações entre as heurísticas e dos métodos penalizados e não-penalizados são apresentados no Capítulo 5. Finalmente no Capítulo 6 descrevem-se as conclusões e sugestões dos trabalhos futuros para esta tese.

Capítulo 2

Problema do Caixeiro Viajante com Grupamentos

2.1 Definição do Problema

O PCV é um dos problemas mais estudados na área de otimização combinatória e consiste em achar um ciclo partindo de uma cidade (origem) percorrendo todas as outras cidades em uma única vez sem retornar a qualquer uma delas e retornando a cidade de origem (ciclo hamiltoniano), enquanto simultaneamente minimiza alguma função de custo. Este problema pode ser representado na estrutura de um grafo não direcionado, completo e ponderado (LAWLER et al., 1985).

O PCV é um problema $\mathcal{NP}$-difícil (GAREY; JOHNSON, 1979). Neste contexto, para o PCV não são conhecidos algoritmos que o resolvam em tempo polinomial. Admitindo que um conjunto de vértices represente as cidades, necessita-se construir um ciclo passando por todos os vértices uma única vez. Quantas soluções são possíveis nesta sequência de vértices? A resposta no caso simétrico é que o número total de soluções possíveis é $n!/2$, onde n é o total de vértices.

O Problema do Caixeiro Viajante com Grupamentos (PCVG) é uma extensão do PCV. O PCV é um caso particular do PCVG, onde existe somente um grupo ou analogamente um vértice em cada grupo. Por ser uma generalização do PCV torna-o também $\mathcal{NP}$-difícil (GUTTMANN-BECK et al., 2000) e (DING; CHENG; HE, 2007).

Uma descrição do PCVG (LAPORTE; PALEKAR, 2002) na estrutura de um grafo é dado a seguir: seja $G{=}(V, A)$ um grafo completo, direcionado, simétrico e ponderado com

um conjunto de vértices $V = \{v_1, v_2, ..., v_n\}$ e um conjunto de arcos $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ em ambas as direções com mesmo peso. O conjunto de vértices V é particionado em m grupos $V_1, V_2, ..., V_m$, onde: $V = \bigcup_{i=1}^{m} V_i$, $\forall\ i$ e $V_i \cap V_j = \emptyset$ para todos i e j, $i \neq j$. Assumindo que um custo não-negativo c_{ij} é associado com o arco $(v_i, v_j) \in A$, o PCVG consiste em determinar um ciclo hamiltoniano de custo mínimo em G, tal que os vértices de cada grupo são visitados contiguamente, denominado de PCVG não-orientado.

O custo c_{ij} é a distância euclidiana que corresponde, neste trabalho, a distância geométrica d_{ij} entre dois vértices v_i e v_j no plano bidimensional. Detalhes do cálculo da distância d_{ij} podem ser visto no **Apêndice A**. Neste contexto, instâncias euclidianas significam que quando os métodos utilizam estas instâncias para resolução do PCVG fazem o uso das distâncias euclidianas. Visto que as instâncias abordadas neste trabalho são euclidianas e simétricas, a rigor temos o Problema do Caixeiro Viajante com Grupamentos Euclidiano e Simétrico.

Uma variante do PCVG pode ser destacada quando uma sequência de visita aos grupos já está pré-definida, mais precisamente, será uma permutação na ordem de visita dos grupos $V_{i_1}, V_{i_2}, ..., V_{i_m}$, no qual chama-se PCVG orientado, que também pertence também a classe $\mathcal{NP}$-difícil (ANILY; BRAMEL; HERTZ, 1999).

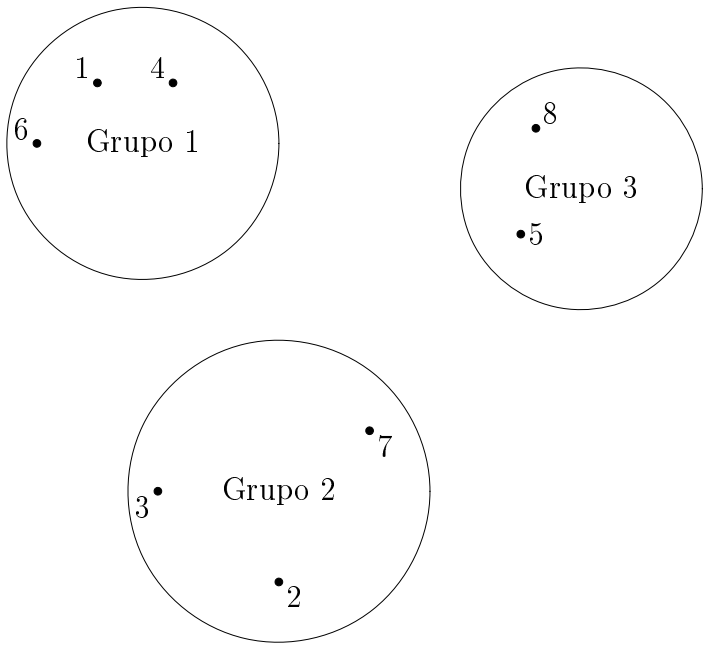


Figura 2.1: Exemplo para o PCVG.

A partir deste momento denota-se o PCVG não-orientado, simétrico e euclidiano, simplesmente como PCVG. Para exemplificar o PCVG, mostra-se na Figura 2.1 os vértices

e os grupos a que estes pertencem. O Grupo 1 contém os seguintes vértices: 1, 4 e 6. O Grupo 2 os vértices 2, 3 e 7 e o Grupo 3 os vértices 5 e 8.

Uma solução viável para o exemplo dada na Figura 2.1 é mostrado pela Figura 2.2, onde a solução para o PCVG forma um ciclo hamiltoniano que passa por todos os vértices, visitando todos os vértices de cada grupo contiguamente.

As arestas (linha cheia) (4,1) e (1,6) mostram as arestas do Grupo 1, para o Grupo 2 (3,2) e (2,7) e a aresta (5,8) no Grupo 3. Por último, as linhas tracejadas mostram as ligações entre os grupos.

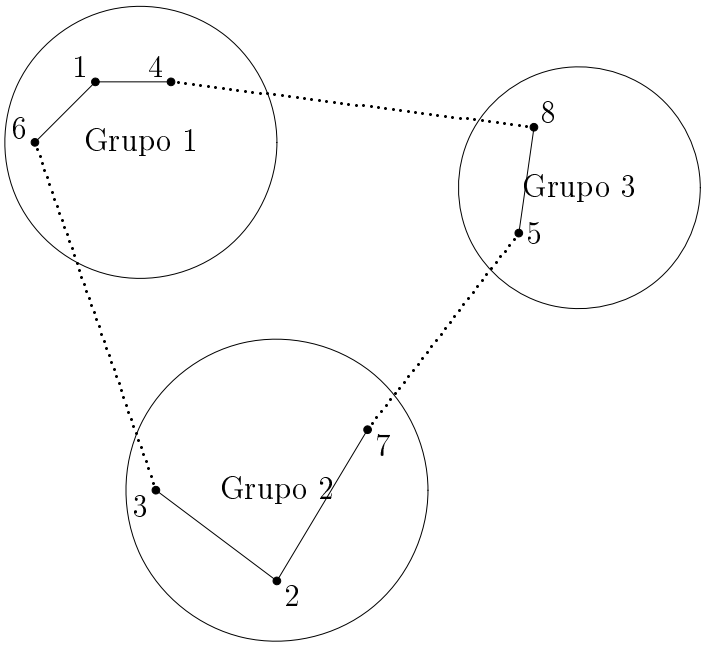


Figura 2.2: Uma solução viável do PCVG da Figura 2.1.

Uma outra solução viável utilizando o mesmo exemplo dada na Figura 2.1, mas de custo maior é mostrado pela Figura 2.3.

As Figuras 2.2 e 2.3 mostram que a escolha dos vértices extremos de entrada e de saída para cada grupo é um fator importante para encontrar soluções de melhor qualidade para o PCVG. Através deste exemplo é demonstrado que a dificuldade do PCVG aumenta, porque não é conhecido estes vértices extremos de cada grupo, principalmente na versão onde não é pré-definida a visita dos grupos. Pois, nesta versão mais complexa para o PCVG, os algoritmos devem buscar a melhor sequência de visitas aos vértices de cada grupo e também analisar a melhor sequência de visitas aos grupos.

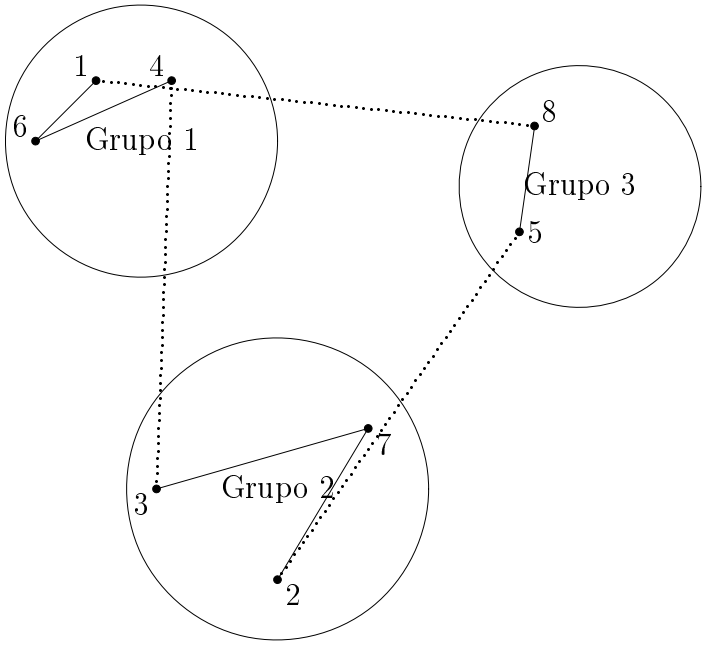


Figura 2.3: Uma solução viável do PCVG da Figura 2.1 de custo maior.

2.2 Trabalhos Relacionados

O PCVG foi proposto por Chisman (1975) para resolver um problema real de roteamento automático em sistemas de armazenagem. Neste exemplo particular de sistemas de armazenagem, as ordens de despacho de produtos devem ser respeitadas e estas ordens contêm várias subordens, mais conhecidas como *bills-of-lading*. *Bill-of-lading* (BL) é um documento emitido por uma transportadora a um transportador (carregador), descrevendo e confirmando a recepção de produtos para o transporte e especificando as condições de entrega. Nos sistemas de armazenagem, cada BL estabelece os produtos a serem despachados contendo o número de estoque destes produtos. Um veículo motorizado deve despachar através do armazém coletando os produtos estocados especificados em cada BL. A restrição nos sistemas de armazenagem especifica que uma BL deve ser completamente despachada antes de a próxima BL ser iniciada. A ordem de coleta dos produtos estocados especificados em cada BL e a ordenação das BLs é simultaneamente otimizada.

Este tipo de logística foi modelado como um PCVG, onde os vértices representam as localizações dos estoques dos produtos, as arestas representam as distâncias a serem percorridas pelo veículo motorizado e os grupos representam as *bills-of-lading*. O problema é transformado em um PCV adicionando-se uma penalidade constante de valor M aos custos das arestas que ligam quaisquer dois vértices pertencentes a grupos diferentes.

Usando um algoritmo *branch-and-bound* o PCV resultante foi resolvido, mas somente para instâncias contendo no máximo 10 vértices.

Como todas as arestas intergrupos são penalizadas por um valor igual M, isto não afeta as escolhas das arestas que serão mais atraentes entre os grupos. O valor M utilizado foi igual a $10 * max\{c_{ij}\}$, no qual c_{ij} são os custos entre as arestas i e j, $\forall i, j$. O enlace entre os grupos ocorrerá devido a um ciclo hamiltoniano ser necessário para o PCV. Na solução final o custo da função objetivo deverá ser corrigido eliminando-se o valor M de penalização. Como as arestas intragrupos não foram penalizadas, a solução do algoritmo produzido em cada grupo não é afetado. Em (CHISMAN, 1975) mostra-se a restrição que deverá ser adicionada numa formulação do PCV para manter a visita dos vértices contígua em cada grupo. Ao utilizar esta restrição, não há necessidade de se penalizar as arestas intergrupos para resolver o PCVG através de um algoritmo *branch-and-bound*.

Limites inferiores utilizando Relaxação Lagrangeana foram desenvolvidos em (JONGENS; VOLGENANT, 1985) para o PCVG. O método para obtenção do limite inferior baseou-se na árvore geradora mínima desenvolvida primeiro para o PCV (HELD; KARP, 1971) e (HELD; KARP, 1970). Foram criados vários conjuntos de instâncias variando o número de vértices de 80 a 150 com diferentes números de grupos. Os valores médios dos limites inferiores alcançaram *gap* igual a 0,35%.

Em (LAPORTE; POTVIN; QUILLERET, 1996) uma Busca Tabu combinada com uma fase de diversificação usando um Algoritmo Genético foi proposta para o PCVG, mas a ordem de visita dos grupos já era definida *a priori*. O algoritmo parte de um conjunto de soluções geradas aleatoriamente. A cada solução individual gerada aplica-se a Busca Tabu. A seguir, constrói-se um conjunto de soluções selecionadas entre as melhores produzidas pela Busca Tabu e removem-se as repetidas. Neste conjunto estão os genitores para a próxima geração de filhos. O conjunto de soluções filhos é construído da seguinte forma: caso o número de soluções seja ímpar, a solução pior é descartada; a seguir escolhe-se aleatoriamente um par entre as soluções pais e cada par selecionado é processado pela Recombinação de Arestas (RA) (WHITLEY, 1989), a fim de se obterem novos filhos. O processo se repete até que um número de soluções filhos diferentes sejam produzidas para completar uma população do Algoritmo Genético; as etapas são novamente aplicadas iniciando pela Busca Tabu sobre as soluções filhos geradas, ao menos que o número máximo de gerações seja atingida.

Um Algoritmo Genético (AG) básico (sem componentes que realizam uma busca mais intensiva, tais como: reconexão de caminhos e/ou *Variable Neighborhood Descent*) foi

proposto para o PCVG por Potvin e Guertin (1996). O AG utilizou a reprodução através do operador de seleção de classificação linear descritos em (BAKER, 1985) e (BLICKLE; THIELE, 1995) e a Recombinação de Arestas. O algoritmo tratou de forma independente o roteamento intergrupos e intragupos, dando prioridade a solução intergrupos e em seguida a solução intragrupos. Ainda que este algoritmo resolva o PCVG para o caso geral, o AG foi comparado somente com uma heurística clássica de construção e busca local denominada GENIUS (GENDREAU; HERTZ; LAPORTE, 1992), desenvolvida para o PCV. Além disso, nesta comparação com a heurística GENI (etapa construtiva do GENIUS) utilizou-se uma única vizinhança de tamanho fixo igual a cinco, limitando o desempenho desta heurística.

No artigo apresentado por Ding, Cheng e He (2007) um outro AG, também básico, usa o critério de visitação dos vértices de cada grupo de forma aleatória sem estabelecer uma relação com as distâncias entre os vértices. A escolha da ordem de visita de cada grupo também é aleatória. O AG foi dividido em dois níveis chamados de nível baixo e nível alto. No nível mais baixo, o AG acha um ciclo hamiltoniano para cada grupo. No nível mais alto, o algoritmo escolhe aleatoriamente uma aresta a ser excluída do ciclo de cada grupo e simultaneamente determina a sequência de visita de todos os grupos de forma aleatória. A desvantagem desta metodologia está na resolução do problema de forma separada. Primeiro, a metodologia resolve a sequência dos vértices de cada grupo e depois a visita de cada grupo. Neste trabalho, somente uma forma de gerar instâncias do PCVG foi realizada e poucos testes computacionais foram executados para a avaliação do AG.

No trabalho de (VIEYRA, 1999) uma metaheurística híbrida combinando AG com Colônia de Formigas (AG-CF) foi desenvolvida. O AG-CF inicia sua população de indivíduos definindo primeiro a sequência dos grupos, depois a sequência dos vértices em cada grupo. Os resultados do AG-CF foram comparados com os melhores valores obtidos em (GENDREAU; LAPORTE; POTVIN, 1994) e (OCHI; RABELLO, 1996). Nos testes realizados de um total de 12 instâncias o algoritmo conseguiu sete melhores resultados. Isto reforça a potencialidade dos métodos híbridos para obter soluções aproximadas ao PCVG. O trabalho em (VIEYRA, 1999) sugere, dentre as propostas futuras, utilizar reconexão de caminhos combinadas com outras metaheurísticas, por exemplo, o AG.

Gendreau, Hertz e Laporte (1996) estudaram o PCVG para um caso particular, onde o número de grupos é igual a dois e com a ordem pré-definida para visitar os grupos. Este problema é conhecido como PCV com *BackHauls* (PCVB). Ele consiste de conjuntos de

vértices, mais conhecidos como consumidores, divididos em dois grupos: um chamado de *linehauls*, representado por L, e outro *backhauls*, B. O problema consiste em partindo de um vértice inicial v_0, conhecido como depósito, visitar primeiro os consumidores contidos em *linehauls*, depois os consumidores em *backhauls* e retornar ao depósito. Para tratar o PCVB utilizaram várias heurísticas tais como: GENI e US (GENDREAU; HERTZ; LAPORTE, 1992), Método de Inserção mais Barata (REINELT, 1994) e *Or*-Optimal (OR, 1976), combinando-as em várias alternativas. A heurística composta por GENI+US aplicada ao L (grupo contendo os consumidores *linehauls*), B (os consumidores *backhauls*) e v_0 (depósito) em conjunto obteve os melhores resultados para instâncias com número de vértices até 300, mas seu tempo computacional foi maior que todas as outras heurísticas construídas. A segunda melhor heurística considerando a qualidade da solução utiliza a Inserção mais Barata combinada com a US. Para instâncias maiores com 500 e 1000 vértices a primeira heurística (GENI+US) produziu resultados próximos a todas outras heurísticas.

Um procedimento para resolver o PCVB baseado na metaheurística VNS, que consiste de trocas sistemáticas de vizinhanças de busca local, foi apresentado em (MLADENOVIĆ; HANSEN, 1997). A metaheurística VNS obteve melhoria em média de 0,4% na qualidade da solução comparado com os resultados da GENIUS, mas com um aumento de 30% no tempo computacional. Para todas as instâncias, a heurística VNS obteve melhores resultados. Cabe ressaltar que a heurística VNS utilizou a GENI para construção inicial das soluções e como busca local a US. Pode-se observar que a heurística VNS consiste na execução de várias etapas da GENIUS.

O trabalho em (GHAZIRI; OSMAN, 2003) utilizou redes neurais para resolver o PCVB. Os autores Ghaziri e Osman (2003) desenvolveram um algoritmo baseado no Mapa Auto-Organizado de Kohonen (MAOK) (KOHONEN, 2001). Uma variante do MAOK com busca local foi desenvolvida utilizando a heurística 2-Optimal na fase de pós-otimização. Comparações dos algoritmos foram realizadas com dois grupos de instâncias. O primeiro grupo é composto de 100 instâncias testes e todas com 50 vértices, porém o número de vértices em cada grupo varia na relação $\rho = |B/V|$, onde B é o número de vértices no grupo *backhauls* e V é o total de vértices. Os valores de ρ foram distribuídos no conjunto $\Omega = \{0.1, 0.2, 0.3, 0.4, 0.5\}$. Para estas instâncias os resultados dos algoritmos MAOK e MAOK-2-opt foram comparados com o algoritmo *branch-and-bound* (B&B) (FISCHETTI; TOTH, 1992), sendo que o algoritmo B&B foi executado no mesmo *hardware* no qual foram realizados os testes dos algoritmos MAOK e MAOK-2-opt. A qualidade da solução foi comparada através do $gap_{li_{B\&B}}$ que é calculado em relação ao limite inferior dado pelo

algoritmo B&B. O $gap_{li_{B\&B}}$ médio do MAOK foi igual a 7,95%, do MAOK-2-opt igual a 7,63% e do algoritmo B&B em 7,56%. Em relação ao esforço computacional o tempo médio de MAOK foi igual a 2,24s, de MAOK-2-opt com 2,29s e de B&B igual a 41,48.

O segundo grupo contém um conjunto de instâncias com número de vértices variando de 100 a 1000 com a mesma distribuição dos vértices em cada grupo dada pela relação $\rho \in \Omega$. Nesta comparação o desempenho médio do MAOK-2-opt foi melhor que a GENIUS (GENDREAU; HERTZ; LAPORTE, 1996), mas obteve desempenho inferior ao VNS (MLADENOVIĆ; HANSEN, 1997). O tempo computacional, normalizado num computador Sun SPARC 10 (MLADENOVIĆ; HANSEN, 1997), do MAOK e MAOK-2-opt é maior do que o tempo despendido pela GENIUS e pelo VNS.

Os algoritmos conhecidos como algoritmos α-aproximados são definidos, a seguir:

Definição 1 *Um algoritmo A com solução $x_A(I)$ é α-aproximado para um problema de minimização se, e somente se, qualquer que seja a instância I de tamanho n, o valor da solução obtida é no máximo α vezes o valor da solução ótima $x^*(I)$. Mais especificamente, o parâmetro α é conhecido como fator de aproximação do algoritmo A.*

Através desta definição, se A é α-aproximado, então $x_A(I) \leq \alpha x^*(I)$, para problemas de minimização. Em problemas de minimização, naturalmente, deve-se ter $\alpha \geq 1$. Observe ainda que, quando mais α se aproxima de 1, melhor a qualidade da solução obtida pelo algoritmo α-aproximado.

Os algoritmos α-aproximados para o PCVG com diferentes variantes são encontrados em (ANILY; BRAMEL; HERTZ, 1999), (ARKIN; HASSIN; KLEIN, 1997), (GENDREAU; LAPORTE; HERTZ, 1997), (GENDREAU; LAPORTE; POTVIN, 1994) e (GUTTMANN-BECK et al., 2000) que utilizam adaptações de (CHRISTOFIDES, 1976) e (HOOGEVEEN, 1991). A maioria destes algoritmos necessita estabelecer em cada grupo os vértices de entrada e de saída e uma ordem pré-definida de visita dos grupos.

A metodologia destes algoritmos é seguinte: cada algoritmo constrói primeiro as árvores geradoras mínimas em cada grupo. Em seguida faz um emparelhamento mínimo para o conjunto de vértices de grau ímpar. Na etapa seguinte combina as árvores geradoras com o emparelhamento produzindo um ciclo euleriano. No final converte o ciclo euleriano em um hamiltoniano. Observa-se que estes algoritmos resolvem o problema intergrupos e intragrupos independentes. Uma desvantagem destes algoritmos α-aproximados é de não permitir construir ciclos quando a ordem de visita dos grupos não é especificada.

Nesta categoria de algoritmos α-aproximados que utilizam uma ordem pré-definida de

visita dos grupos, encontra-se um algoritmo com fator de aproximação igual a 5/3 (ANILY; BRAMEL; HERTZ, 1999), um outro com fator de 3,5 (ARKIN; HASSIN; KLEIN, 1997), um que estabelece o fator de aproximação igual a dois (GENDREAU; LAPORTE; POTVIN, 1994) e por último um com fator igual a 3/2 (GENDREAU; LAPORTE; HERTZ, 1997).

No artigo de (GUTTMANN-BECK et al., 2000) descreveram-se soluções para os algoritmos α-aproximados ao PCVG dependendo das especificações dos vértices extremos de entrada e saída de cada grupo. O fator de aproximação foi de 9/5 quando os vértices extremos são especificados e existe a liberdade de escolha para qualquer um dos vértices ser o de entrada e o outro de saída. O fator foi de 21/11 quando os vértices de entrada e saída são especificados e de 37/14 quando somente o vértice de entrada é especificado. Por último o fator foi de 11/4 para o caso quando não são especificados os vértices extremos.

Observa-se que o maior fator de aproximação (11/4) ocorreu quando não são especificados os vértices de entrada e saída. Isto mostra a dificuldade de se obter soluções aproximadas quando trata-se do PCVG no caso geral, onde não é especificada a ordem do grupo e ainda necessita-se encontrar a ordem de visita dos vértices em cada grupo.

2.3 Formulações Matemáticas

Diversas formulações do Problema do Caixeiro Viajante podem ser encontradas em (DANTZIG; FULKERSON; JOHNSON, 1954), (MILLER; TUCKER; ZEMLIN, 1960), (FOX; GAVISH; GRAVES, 1980), (CLAUS, 1984), (LAWLER et al., 1985), (ARTHANARI; USHA, 2000), (PATAKI, 2003) e (ORMAN; WILLIAMS, 2004). Muitas destas, podem ser adaptadas para o Problema do Caixeiro Viajante com Grupamento. Para isto, basta incorporarem-se as restrições que mantém os vértices dos grupos a serem visitados de uma forma contígua. Em (CHISMAN, 1975) especificam-se as restrições a serem incorporadas.

Uma formulação para o PCVG, descrita a seguir, utiliza os trabalhos de (CHISMAN, 1975) e (DANTZIG; FULKERSON; JOHNSON, 1954) no qual chama-se "Dantzig/Chisman". Esta formulação é um problema de programação inteira `0-1` através de um grafo completo, direcionado, simétrico e ponderado $G = (V, A)$, onde $V = \{v_1, v_2, ..., v_n\}$ representa o conjunto de vértices e A o conjunto de arcos, sendo $(v_i, v_j) \in A$, $v_i, v_j \in V$ e $i \neq j$:

$$\text{Min} \quad z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.1}$$

s.a.

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad \forall i \in V \tag{2.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad \forall j \in V \tag{2.3}$$

$$\sum_{i,j \in S} x_{ij} \leq \mid S \mid -1, \qquad \forall S \subset V, \mid S \mid \geq 1 \tag{2.4}$$

$$\sum_{i \in V_k} \sum_{j \in V_k} x_{ij} = \mid V_k \mid -1, \quad \forall V_k \subset V, \mid V_k \mid \geq 1, k = 1, ..., m \tag{2.5}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{2.6}$$

A função objetivo 2.1 requer que se minimize o custo total percorrido pelo caixeiro viajante. As restrições 2.2 garantem que de cada cidade i só se pode sair para uma única cidade j. As restrições 2.3 significam que para cada cidade j há uma única origem em i. Estes dois conjuntos de restrições implicam que somente duas arestas podem ser conectadas a cada vértice formando um conjunto de ciclos possivelmente disjuntos. As restrições 2.4 garantem não haver subciclos desconexos da origem na solução.

As restrições 2.5 indicam que a soma das variáveis x_{ij} na partição do grupo V_k, $k = 1, ..., m$, sendo m o número total de grupos, alcançam o valor da cardinalidade $\mid V_k \mid$ deste grupo menos 1, forçando em conjunto com as restrições 2.2 e 2.3, a formação de um caminho dentro deste grupo. As variáveis binárias x_{ij} assumem o valor igual a 1 se a cidade i for conectada a cidade j na solução. Caso contrário, será igual a 0. Esta formulação envolve $2^n + 2n - 1 + m$ restrições e n^2 variáveis inteiras.

Uma outra formulação para o PCVG utilizando programação inteira `0-1` está descrita em Chisman (1975) e em Miller, Tucker e Zemlin (1960), e é denominada "Miller/Chisman". Seja um caixeiro viajante partindo da origem da cidade 1 sem perda de generalidade. A seguir, mostra-se a formulação dada por (CHISMAN, 1975) e (MILLER; TUCKER; ZEMLIN, 1960):

$$\text{Min} \qquad z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{2.7}$$

s.a.

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad \forall i \in V \tag{2.8}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad \forall j \in V \tag{2.9}$$

$$u_i - u_j + (n-1)x_{ij} \leq (n-2), \quad 2 \leq i \neq j \leq n \tag{2.10}$$

$$\sum_{i \in V_k}\sum_{j \in V_k} x_{ij} = \mid V_k \mid -1, \qquad \forall V_k \subset V, \mid V_k \mid \geq 1, k = 1, ..., m \tag{2.11}$$

$$u_i \geq 0 \qquad 2 \leq i \leq n \tag{2.12}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{2.13}$$

A função objetivo 2.7 é idêntica à função 2.1 e as restrições 2.8, 2.9 e 2.13 são idênticas às restrições 2.2, 2.3 e 2.6, respectivamente. As restrições 2.10 garantem não haver subciclo na solução e que o caixeiro viajante parte da cidade de número 1 (vértice origem). As variáveis u_i, $i = 2, ..., n$ representam a sequência das cidades i a serem visitadas tendo a relação u_j - $u_k \leq$ -1, $\forall j, k \in V$ sendo k e j cidades contíguas na solução, $x_{jk} = 1$.

As restrições 2.10 utilizam variáveis contínuas u_i que representam o número de arcos menos uma unidade (1) entre o vértice origem e o vértice i. São necessários num ciclo que começa na cidade 1 e termina na última cidade a ser visitada, no máximo $(n-1)$ arcos. Se uma cidade j for visitada após a cidade k, a diferença entre os valores das variáveis contínuas (u_k e u_j) é da ordem de uma unidade (1). Assim, a diferença entre quaisquer duas cidades, exceto a cidade origem, adquire um *limite superior* igual a $(n-2)$. Este limite superior é mostrado nas restrições 2.10. Observe que no caso trivial com x_{ij}=0, as restrições 2.10 são satisfeitas utilizando este limite superior.

A Figura 2.4 mostra um exemplo para as restrições 2.10, com os dados da Figura 2.2. Nesta solução para o PCVG os valores das variáveis u_i são: u_6=0, u_3=1, u_2=2, u_7=3, u_5=4, u_8=5 e u_4=6. Se observarmos o vértice 7 e percorrermos o ciclo no sentido horário na Figura 2.4, verifica-se que existem as arestas a, b, c e d até se chegar no vértice 1. O total de arestas menos 1 é igual ao valor da variável u_7=3. Para qualquer par de vértices consecutivos j e k no ciclo da Figura 2.4 verifica-se que a diferença entre as variáveis $|u_j$ - $u_k|$ é igual a 1.

As restrições 2.11 forçam um caminho dentro de cada grupo, V_k, $k = 1, ..., m$, indicado por suas variáveis x_{ij}, igualmente às restrições 2.5. Esta formulação envolve $n^2-n+2+m$

restrições e n^2 variáveis inteiras e $n-1$ variáveis contínuas.

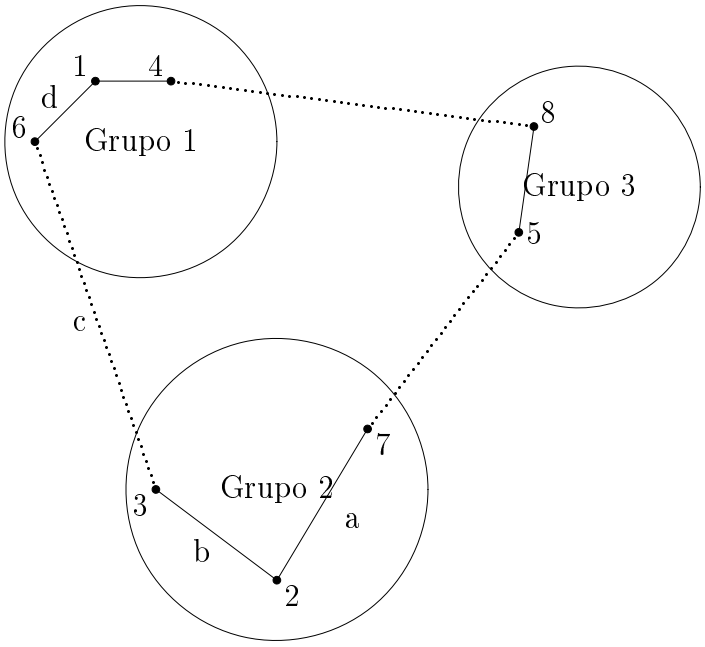


Figura 2.4: Formação do ciclo hamiltoniano evitando os subciclos desconexos da origem para o PCVG.

Apesar de ambas as formulações matemáticas de Dantzig/Chisman e Miller/Chisman encontrarem soluções ótimas ou limites inferiores para o PCVG, a diferença entre estas duas formulações está na forma de tratar as restrições que evitam subciclos desconexos da origem na solução. Na formulação matemática de Dantzig/Chisman, a medida que cresce o número de vértices, as restrições que evitam subciclos crescem na proporção exponencial. Isto conduz a formulação de Dantzig/Chisman não ser viável solucioná-la diretamente. Enquanto que no modelo Miller/Chisman, se houver um incremento no número de vértices, as restrições que evitam subciclos serão incrementadas na proporção quadrática.

Nos testes computacionais realizados neste trabalho utilizando a formulação Dantzig/Chisman não foi possível armazenar todas as restrições de eliminação de subciclos (restrições 2.4) a partir de um certo número n_k de vértices. Nos testes computacionais, o valor n_k foi igual a 20 para um conjunto de instâncias mostradas na seção 5.1. Em todos os resultados computacionais apresentados no Capítulo 5, onde foram alcançados soluções ótimas ou bons limitantes inferiores ao PCVG, utilizou-se a formulação Miller/Chisman com $n_k > 20$.

Uma forma para solucionar o PCVG com a formulação Dantzig/Chisman é através de

um algoritmo *branch-and-cut*. A estratégia para o PCVG via algoritmo *branch-and-cut* será semelhante a utilizada ao PCV. Para uma descrição breve, no início deste algoritmo as restrições 2.2, 2.3, 2.5 são armazenadas, a integralidade das restrições 2.6 são relaxadas e planos de corte iterativamente são adicionados até se encontrarem soluções ótimas ou bons limitantes inferiores para o PCVG. Estes planos de cortes eventualmente utilizarão alguns subconjuntos das restrições 2.4.

Além disso, o algoritmo *branch-and-cut* encontra os limites inferiores e utiliza os limites superiores calculados por heurísticas. Nas fases iterativas mais adiante deste algoritmo *branch-and-cut* ocorrerá a fixação de algumas variávies inteiras através de heurísticas primais, nos quais podem realizar cálculos por custos reduzidos ou por implicações lógicas. Em seguida o algoritmo *branch-and-cut* fará a busca sobre os nós da árvore do algoritmo *branch-and-bound*. No algoritmo *branch-and-bound* ocorre a fase de ramificação em duas opções: por variáveis ou por restrições. A primeira opção de ramificação fixa o valor de cada variável em 0 ou 1 e a segunda opção são adicionadas uma restrição de desigualdade $ax \leq b'$ num ramo da árvore e em outro ramo a desiguladade $ax \geq b''$, onde x representa um vetor coluna de variávies na forma x_{ij}, sendo $i, j = 1, ..., n$, a representa um vetor linha de coeficientes a_{ij} associados a estas variáveis e os valores de b' e b'' representam constantes.

Nesta segunda opção os limites inferiores dos subproblemas são de melhor qualidade, mas o gerenciamento da árvore de decisão é mais difícil dado que a primeira opção somente é necessária armazenar os valores das variáveis até a etapa corrente do algoritmo (FERREIRA; WAKABAYASHI, 1996). Depois da ramificação do algoritmo *branch-and-bound* inicia a fase de relaxação linear novamente e novos cortes são adicionados. O processo termina quando o algoritmo *branch-and-cut* encontra uma solução ótima ou é interrompido, no caso de instâncias de grande porte, imprimindo o limite inferior.

Uma propriedade a ser verificada em algoritmos *branch-and-cut* é a existência de *tailing off* que são desigualdades de separação de alguns pontos gerando pequenas melhorias e sobrecarregando muito o sistema. Neste caso em vez de separar estes pontos, é preciso ir direto para a ramificação.

Um dos grandes problemas do algoritmo *branch-and-bound* é o *gap* de integralidade definido entre a solução relaxada e a solução inteira. Para reduzir este *gap* um dos primeiros planos de cortes desenvolvidos foram introduzidos por Gomory e Hu (1961). Para problemas de cobertura de conjuntos este tipo de corte é eficiente, mas para diversos outros problemas, como por exemplo o PCVG, estes cortes reduzem muito pouco o *gap*

de integralidade. Cabe ressaltar que no *solver* comercial CPLEX (2009), os cortes de Gomory já estão incorporados para solucionar problemas de programação linear inteira.

Um dos avanços na programação linear inteira envolvendo a teoria dos poliedros foram as classes de restrições ou desigualdades que poderão ser adicionadas ao algoritmo *branch-and-bound* nos quais frequentemente definam faces ou desigualdades válidas do problema. As facetas são inequações válidas que interceptam o poliedro inteiro, definido pelas restrições do problema, em uma face com uma dimensão menor do que a dimensão do poliedro inteiro. A situação ideal para um algoritmo *branch-and-bound* é a adição de cortes que representam facetas do fecho convexo das soluções inteiras viáveis (SHERALI; DRISCOLL, 2000). Um estudo mais amplo sobre poliedros pode ser encontrado em (FERREIRA; WAKABAYASHI, 1996).

Capítulo 3

Heurísticas para o PCVG

3.1 Heurísticas Construtivas

Existem diversas heurísticas construtivas clássicas para o PCV e dentre elas: vizinho mais próximo, inserção e baseadas em árvore geradora (REINELT, 1994). Para o PCVG, estas heurísticas devem ser adaptadas para se construir soluções viáveis.

3.1.1 Heurística do Vizinho mais Próximo

A heurística do vizinho mais próximo, descrita em Bellmore e Nemhauser (1968) é explicada a seguir para o PCV. Dado um grafo completo e simétrico, a *primeira etapa* é iniciar a solução com um vértice aleatório; na *segunda etapa*, procura-se o vértice mais próximo do último vértice adicionado, que ainda não foi escolhido, e ligam-se esses dois vértices; quando todos os vértices estiverem adicionados, adicione uma aresta conectando o vértice inicial e o último vértice adicionado. A complexidade deste algoritmo é $O(n^2)$, sendo n o número de cidades. Uma variante deste algoritmo seria procurar o vértice mais próximo dos vértices extremos adicionados para a próxima inserção. A complexidade desta variante, também é $O(n^2)$. Estes algoritmos podem ser executados escolhendo-se um vértice de cada vez com a finalidade de minimizar o efeito de escolha do vértice inicial. Neste caso, a complexidade torna-se $O(n^3)$.

As adaptações da heurística do vizinho mais próximo ao PCVG são descritas a seguir pelos Algoritmos 1 e 2. No Algoritmo 1, as arestas que ligam os vértices de diferentes grupos são penalizadas e passa-se a tratar o PCVG como um PCV. No Algoritmo 2, resolve-se o PCVG diretamente sem a necessidade das arestas intergrupos serem penalizadas.

O procedimento `penalizar_arestas( )` (passo 5 do Algoritmo 1) é utilizado para

alterar os valores das arestas inter-grupos. A alteração penaliza os custos entre os vértices que pertecem a grupos distintos somando um valor constante igual a L, onde $L >> \max\{c_{ij}\}$. No Algoritmo 1, o conjunto de vértices $V = \{v_1, v_2, ..., v_n\}$ é definido para representar os elementos pertencentes a solução S.

Algoritmo 1: Algoritmo Vizinho mais Próximo com as Arestas Penalizadas

1: Escolher um vértice inicial aleatório v_k;
2: $S \leftarrow v_k$;
3: Inicializar o conjunto de candidatos $C \leftarrow V \backslash \{v_k\}$;
4: Faça $v_p \leftarrow v_k$;
5: `penalizar_arestas( );`
6: **enquanto** $C \neq \emptyset$ **faça**
7: Avaliar os custos incrementais $c(v_i)$, $v_i \in C$ e computar as distâncias de todos os vértices contidos em C em relação ao v_p;
8: $c_{min} \leftarrow min\{c(v_i) : v_i \in C\}$;
9: $c_{max} \leftarrow max\{c(v_i) : v_i \in C\}$;
10: $LRC \leftarrow \{v_i \in C : c(v_i) \leq c_{min} + \alpha * (c_{max} - c_{min})\}$;
11: Selecionar um elemento v_{k+1} da LRC aleatório;
12: $S \leftarrow S \cup \{v_{k+1}\}$; (conectar v_p a v_{k+1})
13: Faça $v_p \leftarrow v_{k+1}$;
14: Atualizar o conjunto candidatos $C \leftarrow C \backslash \{v_{k+1}\}$;
15: **fim enquanto**
16: Retornar S

No Algoritmo 1, a solução é inicializada escolhendo-se aleatoriamente um vértice (passos 1-2). Em seguida, o conjunto de candidatos é inicializado com todos os vértices que não estão na solução parcial e as arestas inter-grupos são penalizadas, passo 5. Este conjunto de candidatos é avaliado pelos custos incrementais dos vértices, computando as distâncias destes vértices contidos no conjunto em relação ao último vértice conectado à solução parcial. O menor valor do custo incremental c_{min} e o maior c_{max} são retidos para calcular a Lista Restrita de Candidatos (LRC). A LRC (versão randomizada) é construída para posterior uso em metaheurísticas que geram várias soluções iniciais. Os detalhes da construção da LRC estão descritos na seção 4.1.1. A seguir, seleciona-se um elemento vigente mais próximo aleatoriamente da LRC que é conectado ao vértice extremo da solução parcial, passo 12. Atualiza-se o conjunto de candidatos e se repete o processo (passos 7 até 14) até que o conjunto de candidatos esteja vazio. No final a solução S é retornada, passo 16.

No Algoritmo 2 soluções são construídas escolhendo-se primeiro um vértice aleatório (vértice inicial) e avaliando-se em seguida todos os vértices que ainda restaram, mas pertencentes ao mesmo grupo do vértice inicial. Quando todos os vértices do grupo já

foram selecionados e estão contidos na solução parcial, passa-se a escolher um vértice entre os grupos que ainda não foram selecionados a ser introduzido na solução parcial e o processo se repete.

O Algoritmo 2 diferencia-se do Algoritmo 1 na forma de se inicializar o conjunto de candidatos e por não penalizar as arestas intergrupos. O conjunto de candidatos a ser inicializado é tal que se existem vértices pertencentes ao grupo do vértice extremo (grupo corrente), consideram-se somente estes vértices para compor o conjunto. Quando todos os vértices do grupo corrente forem inseridos na solução parcial, todos os vértices que ainda não pertencem à solução parcial são considerados candidatos.

Algoritmo 2: Algoritmo Vizinho mais Próximo com as Arestas não Penalizadas

1: Escolher um vértice inicial aleatório v_k;
2: $S \leftarrow v_k$;
3: Inicializar o conjunto de candidatos $C \leftarrow V \backslash \{v_k\}$;
4: Faça $v_p \leftarrow v_k$;
5: **enquanto** $C \neq \emptyset$ **faça**
6: **se** Existir vértices pertencentes ao mesmo grupo G_i do vértice v_p **então**
7: avaliar os custos incrementais $c(v_i)$, $v_i \in G_i$ e computar as distâncias dos vértices deste grupo em relação ao v_p;
8: $V \leftarrow G_i$;
9: **senão**
10: Avaliar os custos incrementais $c(v_i)$, $v_i \in C$ e computar as distâncias de todos os vértices contidos em C em relação ao v_p;
11: $V \leftarrow C$;
12: **fim se**
13: $c_{min} \leftarrow min\{c(v_i) : v_i \in V\}$;
14: $c_{max} \leftarrow max\{c(ei) : v_i \in V\}$;
15: $LRC \leftarrow \{v_i \in V : c(v_i) \leq c_{min} + \alpha * (c_{max} - c_{min})\}$;
16: Selecionar um elemento v_{k+1} da LRC aleatório;
17: $S \leftarrow S \cup \{v_{k+1}\}$; (conectar v_p a v_{k+1})
18: Faça $v_p \leftarrow v_{k+1}$;
19: Atualizar o conjunto candidatos $C \leftarrow C \backslash \{v_{k+1}\}$;
20: **fim enquanto**
21: Retornar S

3.1.2 Heurística de Inserção mais Próxima

As heurísticas de inserção geralmente necessitam de três níveis de decisão: um ciclo inicial, a escolha do próximo vértice a ser inserido na solução e a posição de inserção desse novo vértice no ciclo atual. A inserção do vértice pode ser escolhida de várias formas: vértice mais próximo, vértice mais distante, vértice que conduz ao ciclo mais barato (inserção

mais barata) e vértice aleatório.

Algoritmo 3: Algoritmo Inserção mais Próxima com as Arestas Penalizadas

1: Escolher três vértices iniciais v_l, v_{l+1} e v_{l+2}, sendo o primeiro aleatório e os outros dois serão os vizinhos mais próximos;
2: $S \leftarrow \{v_l, v_{l+1}, v_{l+2}\}$;
3: Inicializar o conjunto de candidatos $C \leftarrow V \backslash \{v_l, v_{l+1}, v_{l+2}\}$;
4: `penalizar_arestas( );`
5: **enquanto** $C \neq \emptyset$ **faça**
6: Encontrar os vértices mais próximos contidos em C em relação aos vértices pertencentes ao ciclo já formado, calculando o custo incremental $c(v_k)$;
7: $c_{min} \leftarrow min\{c(v_k) : v_k \in C\}$;
8: $c_{max} \leftarrow max\{c(v_k) : v_k \in C\}$;
9: $LRC \leftarrow \{v_k \in C : c(v_k) \leq c_{min} + \alpha * (c_{max} - c_{min})\}$;
10: Selecionar um elemento v_k da LRC aleatório;
11: $S \leftarrow S \cup \{v_k\}$; (conectar v_k aos vértices do ciclo (i e $i+1$), cujo custo (*price*) $p(v_k)$=$\{p_{i,k}$+$p_{k,i+1}$-$p_{i,i+1}\}$ seja mínimo e atualizar o ciclo);
12: Atualizar o conjunto candidatos $C \leftarrow C \backslash \{v_k\}$;)
13: **fim enquanto**
14: Retornar S

O Algoritmo 3 e o Algoritmo 4 foram desenvolvidos para o PCVG utilizando-se o conceito da heurística de inserção mais próxima. O primeiro, Algoritmo 3, penaliza as arestas intergrupos como no Algoritmo 1 tratando o PCVG como um PCV. O segundo, Algoritmo 4, segue a mesma metodologia do Algoritmo 2 na formação do conjunto de candidatos. Primeiro escolhem-se os vértices que irão compor o conjunto de candidatos sendo aqueles que pertencem ao mesmo grupo corrente dos vértices contidos na solução parcial. Quando todos os vértices do grupo corrente já estão na solução parcial, todos os outros vértices restantes que ainda não foram selecionados são considerados candidatos.

No início do Algoritmo 3 são escolhidos três vértices, sendo o primeiro aleatório e os dois outros usando o conceito de vizinho mais próximo. Em seguida, os vértices pertencentes ao conjunto de candidatos são avaliados pelos custos incrementais (passo 6), computando-se as distâncias destes vértices contidos no conjunto em relação aos vértices da solução parcial. O valor de c_{min}, menor custo incremental e o valor de c_{max}, maior custo incremental são armazenados para calcular a LRC. A seguir, seleciona-se aleatoriamente um vértice da LRC que é conectado aos vértices do ciclo da solução parcial, onde o custo de inserção seja mínimo. Atualiza-se o conjunto de candidatos e se repetem os passos 6 até 12. O algoritmo termina quando o conjunto de candidatos estiver vazio.

O Algoritmo 4 não penaliza as arestas intergrupos e a formação do conjunto de candidatos é diferente da utilizada no Algoritmo 3. Neste caso, o conjunto de candidatos

Algoritmo 4: Algoritmo Inserção mais Próxima com as Arestas não Penalizadas

1: Escolher três vértices iniciais v_l, v_{l+1} e v_{l+2}, sendo o primeiro aleatório e os outros dois serão os vizinhos mais próximos;
2: $S \leftarrow \{v_l, v_{l+1}, v_{l+2}\}$;
3: Inicializar o conjunto de candidatos $C \leftarrow V \backslash \{v_l, v_{l+1}, v_{l+2}\}$;
4: **enquanto** $C \neq \emptyset$ **faça**
5: **se** Existir vértices em C pertencentes ao grupo G_i dos vértices no ciclo **então**
6: Encontrar os vértices mais próximos contidos em relação aos vértices pertencentes ao ciclo já formado, calculando o custo incremental $c(v_k)$;
7: $V \leftarrow G_i$;
8: **senão**
9: Encontrar os vértices mais próximos contidos em C em relação aos vértices pertencentes ao ciclo já formado, calculando o custo incremental $c(v_k)$;
10: $V \leftarrow C$;
11: **fim se**
12: $c_{min} \leftarrow min\{c(v_k) : v_k \in V\}$;
13: $c_{max} \leftarrow max\{c(v_k) : v_k \in V\}$;
14: $LRC \leftarrow \{v_k \in V : c(v_k) \leq c_{min} + \alpha * (c_{max} - c_{min})\}$;
15: Selecionar um elemento v_k da LRC aleatório;
16: $S \leftarrow S \cup \{v_k\}$; (conectar v_k aos vértices do ciclo (i e $i+1$), cujo custo (*price*) $p(v_k)$=$\{p_{i,k}$+$p_{k,i+1}$-$p_{i,i+1}\}$ seja mínimo e atualizar o ciclo - se $v_k \in G_i$ pelo menos i ou $i+1$ devem $\in G_i$);
17: Atualizar o conjunto candidatos $C \leftarrow C \backslash \{v_k\}$;
18: **fim enquanto**
19: Retornar S

é construído de forma semelhante ao Algoritmo 2. Se existem vértices pertencentes ao grupo corrente dos vértices do ciclo atual (solução parcial), consideram-se somente estes vértices para compor o conjunto de candidatos. Quando todos os vértices pertencentes ao grupo corrente do ciclo atual forem inseridos, consideram-se todos os vértices que ainda não pertence à solução parcial como candidatos. Os passos de 5 até 11 do Algoritmo 4 mostra a forma de se elaborar o conjunto candidatos.

3.1.3 Heurística de Inserção mais Barata

A Heurística de Inserção mais Barata (IMB) se diferencia da Inserção mais Próxima (IMP) pelo fato de IMB efetuar os passos de: (i) quem será inserido e (ii) onde será inserido, num único passo. Isto é, na IMB, determina-se para cada vértice fora da solução parcial, o incremento que o vértice irá causar caso seja inserido na melhor posição do ciclo atual.

Os Algoritmos 5 e 6 descritos a seguir, constroem soluções para o PCVG usando uma adequação da heurística de inserção mais barata desenvolvida para o PCV.

Algoritmo 5: Algoritmo Inserção mais Barata com as Arestas Penalizadas

1: Escolher três vértices iniciais v_l, v_{l+1} e v_{l+2}, sendo o primeiro aleatório e os outros dois serão os vizinhos mais próximos;
2: $S \leftarrow \{v_l, v_{l+1}, v_{l+2}\}$;
3: Inicializar o conjunto de candidatos $C \leftarrow V \backslash \{v_l, v_{l+1}, v_{l+2}\}$;
4: `penalizar_arestas( );`
5: **enquanto** $C \neq \emptyset$ **faça**
6: Calcular o custo incremental sobre o ciclo, neste caso para diferenciar das distâncias, representa-se pelo *price*, $p(v_k)$=$min\{p_{i,k}$+$p_{k,i+1}$-$p_{i,i+1}\}$, onde $k \in C$ e as arestas $(i, i+1)$ pertencem ao ciclo já formado;
7: $p_{min} \leftarrow min\{p(v_k) : v_k \in C\}$;
8: $p_{max} \leftarrow max\{p(v_k) : v_k \in C\}$;
9: $LRC \leftarrow \{v_k \in C : p(v_k) \leq p_{min} + \alpha * (p_{max} - p_{min})\}$;
10: Selecionar um elemento v_k da LRC aleatório;
11: $S \leftarrow S \cup \{v_k\}$; (conectar v_k aos vértices do ciclo i e $i+1$ e atualizar o ciclo);
12: Atualizar o conjunto candidatos $C \leftarrow C \backslash \{v_k\}$;
13: **fim enquanto**
14: Retornar S

Usando a mesma analogia do Algoritmo 1 e do Algoritmo 3, o Algoritmo 5 penaliza as arestas intergrupos. No grafo onde não são penalizadas as arestas intergrupos, o Algoritmo 6 trabalha primeiro com a escolha dos vértices que compõem o conjunto de candidatos escolhendo somente aqueles que pertencem ao mesmo grupo corrente dos vértices contidos na solução parcial. Depois que todos os vértices do grupo corrente forem selecionados, o conjunto de candidatos considera todos os outros vértices restantes que ainda não pertencem à solução parcial.

No Algoritmo 5 três vértices são escolhidos para formar o ciclo inicial, sendo o primeiro aleatório e os dois outros usando o conceito do vizinho mais próximo. Em seguida, os vértices que irão pertencer ao conjunto de candidatos são avaliados pelos custos incrementais (passo 6) sobre a solução parcial, computando todas as inserções dos vértices que não pertencem à solução parcial entre todos os pares de dois vértices consecutivos no ciclo atual. Os valores de c_{min} e de c_{max} são armazenados para calcular a LRC. A seguir, seleciona-se um vértice da LRC aleatoriamente que é conectado aos vértices do ciclo na solução parcial. Atualiza-se o conjunto de candidatos e repetem-se os passos 6 até 12 até que o algoritmo termine quando o conjunto de candidatos estiver vazio.

A forma de compor o conjunto de candidatos do Algoritmo 6 é diferente do Algoritmo 5 porque no Algoritmo 6 as arestas intergrupos não são penalizadas. A forma de compor o conjunto de candidatos é definida a seguir. Se existem vértices pertencentes ao grupo corrente, consideram-se somente estes vértices. Quando todos os vértices do grupo

Algoritmo 6: Algoritmo Inserção mais Barata com as Arestas não Penalizadas

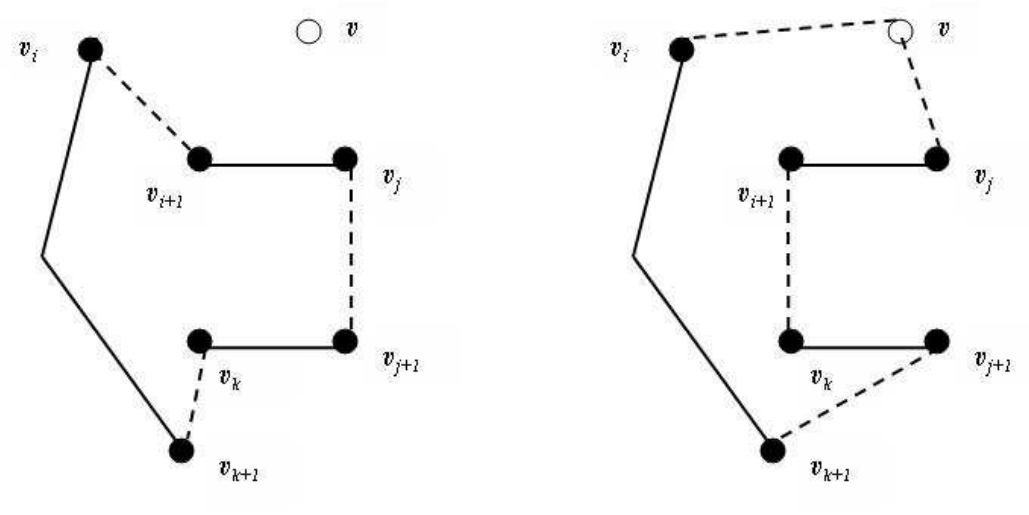
1: Escolher três vértices iniciais v_l, v_{l+1} e v_{l+2}, sendo o primeiro aleatório e os outros dois serão os vizinhos mais próximos;
2: $S \leftarrow \{v_l, v_{l+1}, v_{l+2}\}$;
3: Inicializar o conjunto de candidatos $C \leftarrow V \backslash \{v_l, v_{l+1}, v_{l+2}\}$;
4: **enquanto** $C \neq \emptyset$ **faça**
5: **se** Existir vértices em C pertencentes ao grupo G_i dos vértices do ciclo **então**
6: Calcular o custo incremental, *price*, $p(v_k)$=$min\{p_{i,k}$+$p_{k,i+1}$-$p_{i,i+1}\}$ sobre o ciclo, onde $k \in G_i$ e as arestas $(i, i+1)$ pertencem ao ciclo já formado ou pelo menos i ou $i+1$ devem $\in G_i$;
7: $V \leftarrow G_i$;
8: **senão**
9: Calcular o custo incremental, *price*, $p(v_k)$=$min\{p_{i,k}$+$p_{k,i+1}$-$p_{i,i+1}\}$ sobre o ciclo, onde $k \in C$ e as arestas $(i, i+1)$ pertencem ao ciclo já formado, neste caso $i \in G_m$ e $i+1 \in G_n$, com $m \neq n$;
10: $V \leftarrow C$;
11: **fim se**
12: $p_{min} \leftarrow min\{p(v_k) : v_k \in V\}$;
13: $p_{max} \leftarrow max\{p(v_k) : v_k \in V\}$;
14: $LRC \leftarrow \{v_k \in V : p(v_k) \leq p_{min} + \alpha * (p_{max} - p_{min})\}$;
15: Seleciona um elemento v_k da LRC aleatório;
16: $S \leftarrow S \cup \{v_k\}$; (conectar v_k aos vértices do ciclo (i e $i+1$) e atualizar o ciclo);
17: Atualizar o conjunto candidatos $C \leftarrow C \backslash \{v_k\}$;
18: **fim enquanto**
19: Retornar S

corrente forem inseridos na solução parcial, todos os vértices que ainda não pertencem a solução parcial são considerados candidatos. Os passos 5 até 11 do Algoritmo 6 detalham como construir o conjunto de candidatos. A forma para calcular o custo incremental utilizada no Algoritmo 6, a escolha do vértice da LRC e a atualização da solução parcial são semelhantes ao Algoritmo 5.

3.1.4 Heurística GENI

A heurística GENI (GENDREAU; HERTZ; LAPORTE, 1992) desenvolvida originalmente para o PCV é um método de construção de soluções baseado na inserção generalizada de vértices, cuja principal característica é que o vértice v será inserido entre dois outros que não precisam necessariamente serem consecutivos no ciclo parcial. Depois da inserção, os dois vértices passam a ser adjacentes deste novo vértice v a ser inserido. A inserção de vértices pode ser realizada de dois modos, conforme é ilustrado nas Figuras 3.1 e 3.2. Para cada modo, o método considera a orientação do ciclo em sentido horário e anti-horário. Por exemplo, sejam os vértices v_i, v_j e v_k com $v_i \neq v_j$ e $v_i \neq v_k$. Uma orientação do ciclo

no sentido horário pode ser um ciclo $C_{h_{i,j,k}}$ começando por um arco que liga o vértice v_i ao vértice v_j, o vértice v_j ao vértice v_k e depois o vértice v_k ao vértice v_i fechando este ciclo. Para a orientação do ciclo no sentido anti-horário $C_{ah_{i,k,j}}$, podemos ter um ciclo visitando os vértices v_i, v_k e v_j, respectivamente. Para cada vértice v_m considera-se o vértice v_{m-1} como o vértice antecessor e o vértice v_{m+1} como o sucessor.



(a) Ciclo antes da inserção do vértice v.

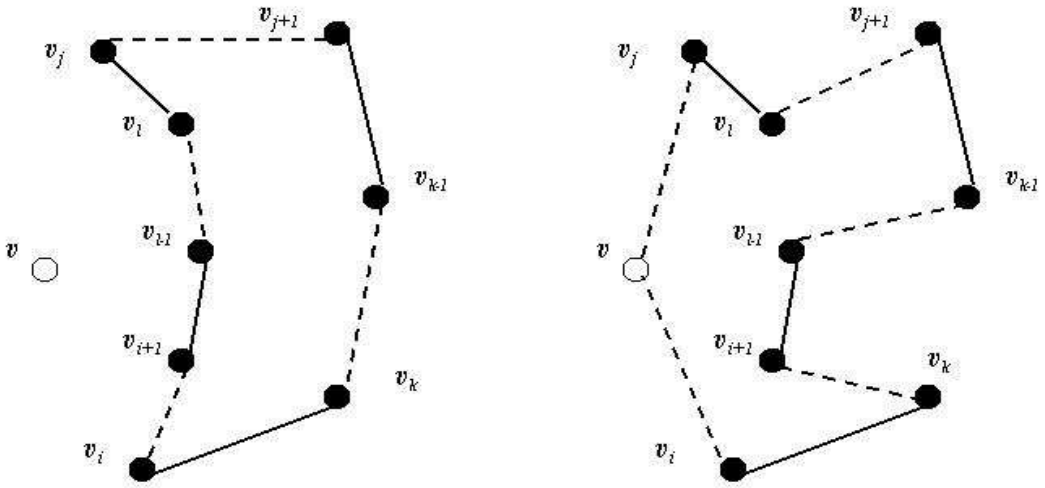
(b) Ciclo depois da inserção do vértice v.

Figura 3.1: Inserção do Tipo I da GENI.

Para a Inserção do Tipo I consideram-se os vértices $v_k \neq v_i$, $v_k \neq v_j$ e v_k um vértice na direção de v_j para v_i. A inserção do vértice v resulta nas retiradas dos arcos (v_i, v_{i+1}), (v_j, v_{j+1}) e (v_k, v_{k+1}) e as inserções dos arcos (v_i, v), (v, v_j), (v_{i+1}, v_k) e (v_{j+1}, v_{k+1}). Estas exclusões e inserções implicam que os caminhos $(v_{i+1}, \ldots, v_j)$ e $(v_{j+1}, \ldots, v_k)$ são reversos. As Figuras 3.1(a) e 3.1(b) ilustram, respectivamente, o ciclo antes da inserção do vértice v e depois da inserção.

A Inserção do Tipo II considera $v_k \neq v_j$, $v_k \neq v_{j+1}$, $v_l \neq v_i$ e $v_l \neq v_{i+1}$, v_k na direção de v_j para v_i e v_l na direção de v_i para v_j. O vértice v inserido no ciclo resulta nas exclusões dos arcos (v_i, v_{i+1}), (v_{l-1}, v_l) , (v_j, v_{j+1}) e (v_{k-1}, v_k) e os arcos (v_i, v), (v, v_j), (v_l, v_{j+1}), (v_{k-1}, v_{l-1}) e (v_{i+1}, v_k) serão inseridos. Os caminhos $(v_{i+1}, \ldots, v_{l-1})$ e $(v_l, \ldots, v_j)$ são reversos. As Figuras 3.1(a) e 3.1(b) ilustram a Inserção do Tipo II da GENI para o vértice v.

Desde que o número de possibilidades de escolha para os vértices v_i, v_j, v_k e v_l é da ordem $O(n^4)$, a GENI aqui proposta busca somente um subconjunto destes vértices e estabelece para o vértice v uma vizinhança de tamanho p, representada por $V_p(v)$. A vizinhança $V_p(v)$ consiste de um conjunto de vértices com cardinalidade p mais próximos



(a) Ciclo antes da inserção do vértice v. (b) Ciclo depois da inserção do vértice v.

Figura 3.2: Inserção do Tipo II da GENI.

Algoritmo 7: Algoritmo GENI

1: Construir um ciclo inicial selecionando-se três vértices aleatórios;
2: Inicializar as p vizinhanças de todos os vértices, $V_p(v_i)$, $i = 1, ..., n$;
3: **enquanto** Todos os vértices não pertencem ao ciclo **faça**
4: Selecionar um vértice v não pertencente ao ciclo formado;
5: Construir o novo ciclo considerando os dois modos de inserções e as duas orientações;
6: Atualizar as novas p vizinhanças de todos os vértices considerando que v agora pertence ao novo ciclo;
7: **fim enquanto**
8: Retornar o Ciclo;

ao vértice v com respeito ao custo c_{ij}. Geralmente o valor de p não deve ser grande. Assim, para um dado valor de p, são selecionados v_i e $v_j \in V_p(v)$, $v_k \in V_p(v_{i+1})$ e $v_l \in V_p(v_{j+1})$. O procedimento da GENI, no Algoritmo 7, considera também todas as inserções de v entre dois vértices consecutivos v_i e v_{i+1} e com $v_i \in V_p(v)$.

Mostram-se a seguir, o Algoritmo 8 e o Algoritmo 9 utilizando a heurística GENI adaptada ao PCVG. No Algoritmo 8, inicialmente escolhem-se para o conjunto candidato os vértices que pertencem ao mesmo grupo corrente dos vértices do ciclo já formado. Se todos os vértices do grupo corrente pertencem à solução parcial, os outros vértices restantes que ainda não foram selecionados serão considerados candidatos. O Algoritmo 8 considera como elementos da LRC os próprios "vértices". O Algoritmo 9 escolhe um vértice aleatório para formar o conjunto candidato e considera como elementos da LRC todas as "posições" deste vértice. O Algoritmo 9 penaliza as arestas intergrupos transformando

Algoritmo 8: Algoritmo usando GENI e escolhendo o elemento do conjunto candidato o *vértice* do mesmo grupo dos vértices do ciclo formado

1: Escolher três vértices iniciais v_l, v_{l+1} e v_{l+2}, sendo o primeiro aleatório e os outros dois serão os vizinhos mais próximos;
2: $S \leftarrow \{v_l, v_{l+1}, v_{l+2}\}$;
3: Inicializar o conjunto de candidatos $C \leftarrow V \backslash \{v_l, v_{l+1}, v_{l+2}\}$;
4: **enquanto** $C \neq \emptyset$ **faça**
5: **se** Existir vértices em C pertencentes ao grupo G_i dos vértices do ciclo **então**
6: Calcular o custo incremental $c(v_k)$ dos vértices que pertencem ao mesmo grupo dos vértices do ciclo já formado usando a Heurística GENI nos dois tipos de inserção e nas duas orientações para cada tipo;
7: $V \leftarrow G_i$;
8: **senão**
9: Calcular o custo incremental $c(v_k)$ dos vértices não pertencentes ao ciclo em relação ao ciclo já formado usando a Heurística GENI nos dois tipos de inserção e nas duas orientações para cada tipo;
10: $V \leftarrow C$;
11: **fim se**
12: $c_{min} \leftarrow min\{c(v_k) : v_k \in V\}$;
13: $c_{max} \leftarrow max\{c(v_k) : v_k \in V\}$;
14: $LRC \leftarrow \{v_k \in V : c(v_k) \leq c_{min} + \alpha * (c_{max} - c_{min})\}$;
15: Seleciona um elemento (vértice) v_k da LRC aleatório;
16: $S \leftarrow S \cup \{v_k\}$; (conectar v_k aos vértices do ciclo já formado usando a posição, o tipo e a orientação escolhida na fase anterior e atualizar o ciclo);
17: Atualizar o conjunto candidatos $C \leftarrow C \backslash \{v_k\}$;
18: **fim enquanto**
19: Retornar S

o PCVG ao PCV. O Algoritmo 9 implementa o algoritmo GENI original (Algoritmo 7) quando o valor de α for igual à zero.

No Algoritmo 8 três vértices são escolhidos, sendo o primeiro aleatório e os outros os vizinhos mais próximos ao primeiro. Em seguida, os vértices pertencentes ao conjunto de candidatos são avaliados pelos custos incrementais (passo 6) ou (passo 9). O cálculo do custo incremental do conjunto de candidatos (vértices não pertencentes ao ciclo) em relação ao vértices do ciclo já formado (ciclo atual) utilizam a Heurística GENI nos dois tipos de inserção e nas duas orientações para cada tipo. Para gerar a LRC, o valor de c_{min}, menor custo incremental e de c_{max}, maior custo incremental são armazenados. A seguir, seleciona-se um vértice da LRC aleatoriamente que é conectado aos vértices do ciclo atual usando a posição, o tipo e a orientação, onde estes parâmetros foram determinados pelo vértice escolhido da LRC. Atualiza o conjunto de candidatos e repetem-se os passos 5 a 17 até que todos os vértices sejam escolhidos.

Algoritmo 9: Algoritmo usando GENI com as arestas penalizadas e escolhendo o elemento do conjunto candidato a *posição*, (GENI original com α=0)

1: Escolher três vértices iniciais v_l, v_{l+1} e v_{l+2}, sendo o primeiro aleatório e os outros dois serão os vizinhos mais próximos;
2: $S \leftarrow \{v_l, v_{l+1}, v_{l+2}\}$;
3: Inicializar o conjunto de candidatos $C \leftarrow V \backslash \{v_l, v_{l+1}, v_{l+2}\}$;
4: `penalizar_arestas( )`;
5: **enquanto** $C \neq \emptyset$ **faça**
6: Selecionar um vertice v_k aleatório do conjunto de candidatos C;
7: Para cada posição, p_i, calcular os custos incrementais, $c_{p_i}(v_k)$, de inserção do vértice v_k no ciclo já formado usando a Heurística GENI nos dois tipos de inserção e nas duas orientações para cada tipo;
8: $c_{p_{min}} \leftarrow min\{c_{p_i}(v_k) : p_i \in C\}$;
9: $c_{p_{max}} \leftarrow max\{c_{p_i}(v_k) : p_i \in C\}$;
10: $LRC \leftarrow \{p_i \in C : c_{p_i}(v_k) \leq c_{p_{min}} + \alpha * (c_{p_{max}} - c_{p_{min}})\}$;
11: Seleciona um elemento (posição) p_i da LRC aleatório;
12: $S \leftarrow S \cup \{v_k\}$; (posicionar o vértice v_k em relação ao ciclo já formado usando o tipo, a orientação e a "posição" escolhida na fase anterior e atualizar o ciclo);
13: Atualizar o conjunto candidatos $C \leftarrow C \backslash \{v_k\}$;.
14: **fim enquanto**
15: Retornar S

No Algoritmo 9 três vértices são escolhidos, sendo o primeiro selecionado aleatoriamente e os outros os vizinhos mais próximos ao primeiro. Em seguida, seleciona-se um vértice aleatoriamente do conjunto de candidatos (todos os vértices não pertencentes à solução parcial). Através deste vértice serão avaliados todos os custos incrementais de todas as "posições" que poderão pertencer ao ciclo já formado considerando-se todas as vizinhanças de tamanho p do vértice escolhido, usando-se a Heurística GENI nos dois tipos de inserção e nas duas orientações para cada tipo. O passo 7 mostra como realizar os cálculos dos custos incrementais. O menor custo incremental $c_{p_{min}}$ e o maior $c_{p_{max}}$ são considerados para gerar a LRC. Seleciona-se uma "posição" da LRC aleatória e conecta-se o vértice ao ciclo atual. Atualiza-se o conjunto de candidatos e repetem-se os passos 6 até 13. O algoritmo termina quando todos os vértices estiverem presentes no ciclo.

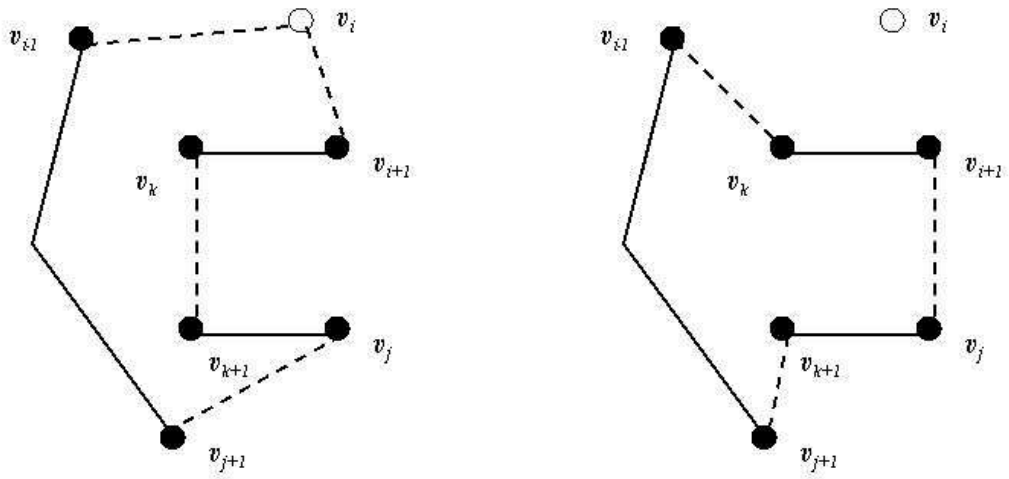
3.2 Heurísticas de Busca Local

Descrevem-se nesta seção, as propostas de algoritmos de busca local para o PCVG.

3.2.1 Heurística *Unstringing/Stringing*

A heurística *Unstringing/Stringing* (US) descrita em (GENDREAU; HERTZ; LAPORTE, 1992) foi aplicada para o Problema do Caixeiro Viajante numa fase de pós-otimização da heurística GENI e consiste na remoção de um vértice de um ciclo viável e inserção deste vértice novamente. A US pode ser aplicada não somente ao procedimento GENI, mas a qualquer um que gere um ciclo completo viável. A US é composta de dois procedimentos: *Stringing* que é idêntico aos passos 4 e 5 do Algoritmo 7 (GENI) e *Unstringing* que consiste na remoção de um vértice, conforme ilustram as Figuras 3.3 e 3.4. A remoção dos vértices realizada pela *Unstringing* considera também as duas orientações no sentido horário e anti-horário. Para a reinserção de um vértice v considera-se a vizinhança de todos os vértices, exceto o próprio, na fase de reconstrução. Mesmo que o ciclo anterior fosse construído pelo método *Stringing*, os vértices disponíveis na vizinhança da fase de construção em que este vértice v foi considerado provavelmente seriam diferentes.

Na Remoção do Tipo I do procedimento *Unstringing* considera $v_j \in V_p(v_{i+1})$, $v_k \in V_p(v_{i-1})$ e v_k no caminho $(v_{i+1}, \ldots, v_{j-1})$. Os arcos (v_{i-1}, v_i), (v_i, v_{i+1}) , (v_k, v_{k+1}) e (v_j, v_{j+1}) serão excluídos e os arcos (v_{i-1}, v_k), (v_{i+1}, v_j) e (v_{k+1}, v_{j+1}) serão conectados. Os caminhos $(v_{i+1}, \ldots, v_k)$ e $(v_{k+1}, \ldots, v_j)$ são reversos. A Remoção do Tipo I é ilustrada nas Figuras 3.3(a) e 3.3(b), onde mostra-se o ciclo antes da remoção do vértice v_i e a nova formação do ciclo, desconsiderando-se este vértice v_i.

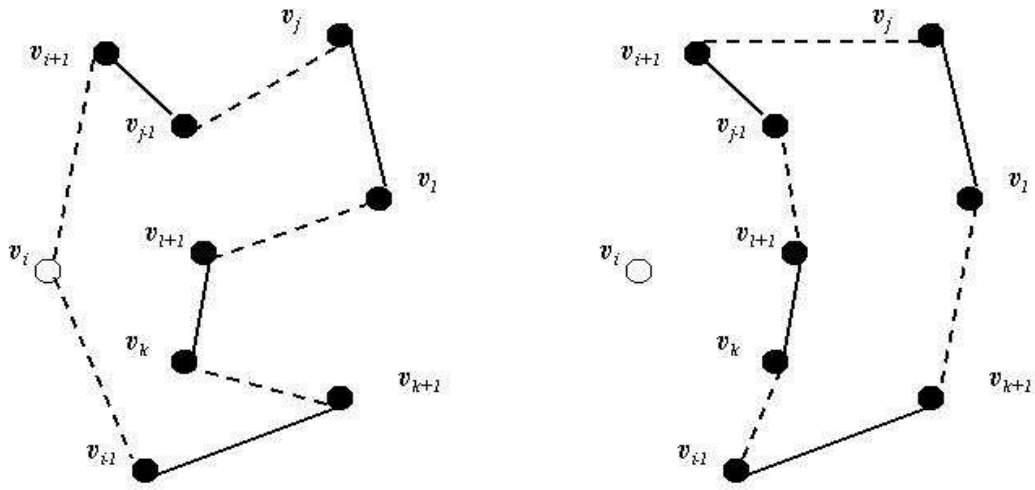


(a) Ciclo com o vértice v_i. (b) Ciclo depois da remoção do vértice v_i.

Figura 3.3: Remoção do Tipo I da *Unstringing*.

A Remoção do Tipo II considera $v_j \in V_p(v_{i+1})$, $v_k \in V_p(v_{i-1})$ e $v_l \in V_p(v_{k+1})$. O vértice v_k estará no caminho $(v_{j+1}, \ldots, v_{i-2})$ e v_l no caminho$(v_j, \ldots, v_{k-1})$. Nesta

remoção, excluem-se os arcos (v_{i-1}, v_i), (v_i, v_{i+1}) , (v_{j-1}, v_j), (v_l, v_{l+1}) e (v_k, v_{k+1}) e os arcos (v_{i-1}, v_k), (v_{l+1}, v_{j-1}), (v_{i+1}, v_j) e (v_l, v_{k+1}) são inseridos. Os caminhos $(v_{i+1}, ... , v_{j-1})$ e $(v_{l+1}, ... , v_k)$ são reversos. As Figuras 3.4(a) e 3.4(b) ilustram a Remoção do Tipo II da *Unstringing* com vértice v_i no ciclo e depois de sua remoção, respectivamente.



(a) Ciclo com o vértice v_i.

(b) Ciclo depois da remoção do vértice v_i.

Figura 3.4: Remoção do Tipo II da *Unstringing*.

O procedimento da *Unstringing/Stringing* é descrito no Algoritmo 10, com T_{min} o melhor ciclo de custo c_{min} a ser retornado pelo algoritmo e n o total de vértices. Considerar um ciclo viável, T_{atual} com custo c_{atual} e v_i o vértice i a ser retirado, como dados de entrada do Algoritmo 10. No início um ciclo viável, com seu custo, é considerado o melhor até o momento, passos 1 a 2. Em seguida é retirado o vértice v_1 do ciclo usando-se as duas formas de remoção da *Unstringing* e é inserido considerando-se as duas formas de inserção da *Stringing*. Em ambas as formas de remoção e inserção são consideradas as duas orientações (horário e anti-horário), passo 5. Se o custo for melhor, o processo de US é mantido sobre o vértice v_1, passos 8 a 12. Caso contrário o procedimento US é aplicado no vértice v_2, passos 13 a 15 e o procedimento US se repete no passo 5 até que todos os vértices sejam analisados. No final o Algoritmo 10 retorna o ciclo T_{min} e seu custo c_{min}, passo 17. Observa-se que caso não haja melhoria no custo do ciclo mínimo T_{min}, o procedimento US continua a atuar sobre a solução do ciclo T_{nov} gerada e considerada a atual solução T_{atual}. Isto significa que se a melhor solução gerada, representada no ciclo T_{nov}, por esta vizinhança não for a melhor de todas as soluções, esta não poderá ser descartada. É possível realizar novas buscas em outras vizinhanças sobre o ciclo T_{nov}.

Muitos métodos como *Simulated Annealing*, Busca Tabu e *Iterated Local Search* adotam esta estratégia de aceitar a solução corrente gerada para a próxima iteração mesmo

Algoritmo 10: Fase *Unstringing/Stringing* do GENI

1: T_{min}=T_{atual};
2: c_{min}=c_{atual};
3: i=1;
4: **enquanto** $i \leq n$ **faça**
5: Inicializar o procedimento US com o vértice v_i no ciclo T_{atual}, considerando as duas formas de remoção e inserçãodos vértices. Em cada caso considerar duas orientações. Seja T_{nov} o novo ciclo obtido com custo c_{nov};
6: T_{atual}=T_{nov};
7: c_{atual}=c_{nov};
8: **se** $T_{atual} < T_{min}$ **então**
9: T_{min}=T_{atual};
10: c_{min}=c_{atual};
11: i=1;
12: **fim se**
13: **se** $T_{atual} \geq T_{min}$ **então**
14: i=i+1;
15: **fim se**
16: **fim enquanto**
17: Retornar o ciclo T_{min} com custo c_{min};

que a solução corrente não atualize a melhor solução encontrada pelo método até a etapa presente.

A GENI e a US constroem soluções a cada etapa com refinamento local. O critério de seleção do valor do tamanho da vizinhança p da GENI e da US está relacionado ao compromisso entre qualidade da solução e tempo computacional. Caso o tempo computacional seja importante, uma solução produzida pela GENI com vizinhança p_G é mais vantajosa do que a solução da US com vizinhança p_{US}, sendo $p_G > p_{US}$. Em Gendreau, Hertz e Laporte (1992), a GENI com p_G igual a sete produz melhor resultado do que a US com p_{US} igual a dois e três para uma instância com 500 vértices. Além disso, o resultado da solução da US com p_{US} igual a sete para esta instância difere somente em 1% da solução produzida pela GENI com p_G=p_{US}. O tempo computacional médio da US neste caso foi aproximadamente 4.5 maior que o tempo da GENI. Resultados semelhantes foram observados com as instâncias com 100, 200, 300 e 400 vértices. Para a instância P532, ver (GENDREAU; HERTZ; LAPORTE, 1992), o valor da solução produzida pela *Unstringing/Stringing* ficou 1,2% acima do valor da solução da Busca Tabu de (FIECHTER, 1994).

Observa-se que nos passos de 8 a 12 do Algoritmo 10 se houver melhoria no custo do ciclo atual, c_{atual}, em relação ao custo mínimo, c_{min}, o algoritmo prossegue com o ciclo

atual, T_{atual}, e reinicia a partir do vértice v_1, o primeiro vértice deste ciclo. Isto para instância com grande número de vértices levaria o algoritmo a ter um custo computacional muito alto. Pode-se modificar este algoritmo fazendo o custo da melhor solução ser atualizado, mas o vértice a ser escolhido na próxima etapa ser incrementado (PAULA, 2001). Sendo assim, os passos de 8 a 12 passam a ser:

$$\begin{aligned} T_{min} &= T_{atual} \\ c_{min} &= c_{atual} \\ i &= i + 1 \end{aligned} \tag{3.1}$$

Esta modificação na heurística será denotada por (USM), acrônimo de *US/Modified*. O Algoritmo 10 reescrito é mostrado no Algoritmo 11 que foi melhorado proporcionando menor esforço computacional. Observa-se no Algoritmo 11, nos passos 8 a 11, que mesmo a solução encontrada sendo melhor, o próximo vértice será analisado.

Algoritmo 11: Algoritmo USM

1: T_{min}=T_{atual};
2: c_{min}=c_{atual};
3: $k = 1$;
4: **enquanto** $k \leq n$ **faça**
5: Aplica-se os procedimentos *Unstringing* e *Stringing* sobre o ciclo T_{atual}, com o vértice v_k, considerando em cada caso os dois tipos de operações e dois tipos de orientações (horário e anti-horário):
6: Seja, T_{iter} o ciclo obtido com o custo c_{iter};
7: T_{atual}=T_{iter} e c_{atual}=c_{iter};
8: **se** $c_{atual} < c_{min}$ **então**
9: T_{min}=T_{atual}, c_{min}=c_{atual};
10: **fim se**
11: k=k+1;
12: **fim enquanto**
13: O melhor ciclo é T_{min} com seu custo igual a c_{min}.

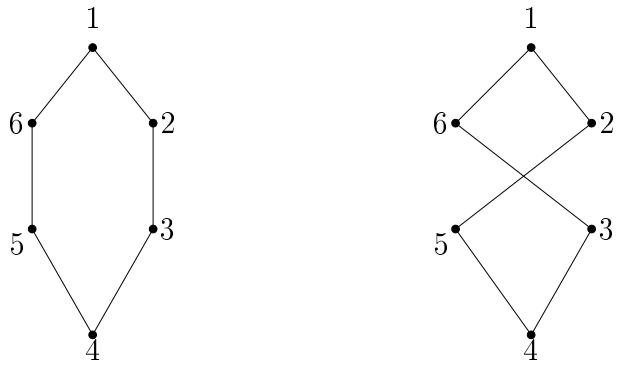
3.2.2 Heurística *k*-Optimal

As heurísticas k-Optimal consistem em substituir k arestas não adjacentes num ciclo, representando uma solução do PCV ou PCVG, combinando-as em todas as possibilidades e modificando-se a sequência dos vértices não adjacentes no ciclo. Estas heurísticas possibilitam a criação de ciclos com custos menores do que o ciclo inicial analisado. A idéia é excluir k arestas não adjacentes do ciclo produzindo k novos caminhos desconectados. Desta forma, é possível se reconectar os k caminhos em outros ciclos diferentes usando as

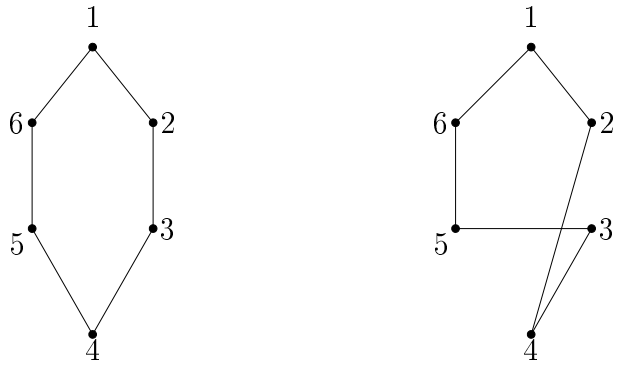
arestas removidas. Assim, novos ciclos são exatamente diferentes por k arestas. Quanto maior o valor de k maior será a probabilidade de se atingir uma solução ótima. No entanto à medida que se aumenta o valor de k, aumenta-se o custo computacional.

A opção de usar a heurística 2-Optimal (k=2) é utilizada por ser relativamente eficiente e de baixo custo computacional, $O(n^2)$. Em Bentley (1990) e Muyldermans et al. (2005) são utilizados as heurísticas 2-Optimal e 3-Optimal.

A Figura 3.5 ilustra para a 2-Optimal a situação de duas arestas não adjacentes sendo retiradas e reconectadas para a formação de um novo ciclo. Para cada aresta retirada, devem-se buscar todas as outras que possibilitem uma nova formação no ciclo. Na 3.5(a) são retiradas as arestas a_{23} e a_{56} e são reconectadas formando um novo ciclo. Na 3.5(b) são retiradas as arestas a_{23} e a_{45} para formar um outro ciclo. Para uma completa formação de todos os ciclos fixando-se a aresta a_{23}, faltaria a retirada da aresta a_{61}. Deve-se na heurística 2-Optimal varrer as arestas fixando uma e escolhendo todas as outras arestas viáveis.



(a) Recombinação das arestas a_{23} e a_{56}.



(b) Recombinação das arestas a_{23} e a_{45}.

Figura 3.5: Exemplos de reconexões de arestas na heurística 2-Optimal.

Capítulo 4

Metaheurísticas para o PCVG

O PCVG pertence à classe $\mathcal{NP}$-difícil limitando com isso o uso exclusivo de métodos exatos. Neste contexto, a resolução para o PCVG proposta neste trabalho utilizará metaheurísticas. A grande maioria das metaheurísticas utilizam heurísticas de construção e de busca local. Na literatura a maioria dos problemas de otimização combinatória estão sendo tratados por metaheurísticas tais como: Busca Tabu, Algoritmos Genéticos, Colônia de Formigas, GRASP, *Variable Neighborhood Search*, *Iterated Local Search*, entre outras (BLUM; ROLI, 2003) e (GLOVER; KOCHENBERGER, 2003).

Para resolver o PCVG propõe-se utilizar a metaheurística GRASP que tem sido aplicada para solucionar diversos problemas de otimização combinatória com sucesso (FESTA; RESENDE, 2002). GRASP é um método fácil de implementar, usa conceitos simples, robusto pois soluções de boa qualidade são obtidas independente das instâncias e eficaz por produzir soluções com tempo computacional relativamente baixo.

Uma metaheurística muitas vezes usada como busca local do GRASP é a *Variable Neighborhood Descent* (VND) (HANSEN; MLADENOVIĆ, 2003), (CHAVES; LORENA, 2005) e (HERNÁNDEZ-PÉREZ; RODRÍGUEZ-MARTÍN; SALAZAR-GONZÁLEZ, 2009). A estrutura do VND não é complexa, de fácil implementação e seus componentes de vizinhanças podem ser alterados aumentando ou diminuindo sua complexidade para ter compromisso entre qualidade de solução e tempo computacional. Assim, propõe-se neste trabalho, utilizar o VND como módulo de busca local a ser incorporado nas heurísticas propostas para o PCVG.

Uma outra metaheurística promissora para problemas de natureza combinatória é a *Iterated Local Search* (ILS) (LOURENÇO; MARTIN; STÜTZLE, 2002). Os fundamentos da metaheurística ILS foram introduzidos por (MARTIN; OTTO; FELTEN, 1992) e (MARTIN;

OTTO, 1996), que aplicaram ao PCV usando instâncias euclidianas. ILS possui princípios simples, é de implementação fácil e possibilita o armazenamento de informações históricas da estrutura da solução ao longo do algoritmo. A metaheurística ILS também será utilizada para solucionar o PCVG.

O uso de memória é incontestavelmente um componente essencial para qualquer pesquisa que merece ser chamada de *inteligente* (GLOVER, 1996). Para utilizar a memória nos procedimentos heurísticos necessitamos ter um conjunto de elementos de boa qualidade denominados de soluções elites, conjunto de referência, conjunto de soluções, conjunto elite, estruturadas de dados especiais ou atributos das soluções (GLOVER, 1996), (TAILLARD et al., 2001), (RESENDE; RIBEIRO, 2005), (GLOVER; LAGUNA; MARTÍ, 2004), (MARTÍ; LAGUNA; GLOVER, 2006) e (RESENDE et al., 2008). Neste trabalho será referenciado o conjunto de elementos de boa qualidade armazenado em memória somente de *conjunto elite*.

Durante as buscas realizadas por metaheurísticas pode ser armazenado em memória um conjunto elite e através deste conjunto são escolhidas soluções nos quais são aplicadas a Reconexão de Caminhos para permitir integrar intensificação e diversificação e gerar novos caminhos entre estas soluções selecionadas explorando o espaço de busca desta vizinhança (MARTÍ; LAGUNA; GLOVER, 2006).

Nos trabalhos de (GLOVER; LAGUNA, 1997) e (HERTZ; TAILLARD; WERRA, 2008) definem a memória de longo prazo como uma das formas de explorar a memória. Uma das estratégias para uso deste tipo de memória é através de um procedimento que constrói soluções ao longo das iterações dos algoritmos heurísticos. Este conjunto elite permite explorar com mais eficiência as estratégias de intensificação e diversificação. Em (KULTUREL-KONAK et al., 2004) explorou a memória de longo prazo como método de busca para encontrar soluções de boa qualidade em vários problemas de otimização combinatória. A metodologia baseada em memória mostrou-se a que obteve melhor desempenho.

Os quatros elementos necessários numa **memória adaptativa**, segundo (GLOVER, 1996), são: recenticidade, frequência, influência e lógica. Destaca-se para este trabalho o elemento recenticidade que se refere a trabalhar com as soluções elites produzidas recentemente e o elemento influência que permite trocar a qualidade e a estrutura atual do conjunto elite.

Os autores Taillard, Gambardella, Gendreau e Potvin (2001) também compartilham a idéia do uso de memória a ser incorporada numa metaheurística e vão além em estabelecer que diversas metaheurísticas possam ser enquadradas numa versão unificada no qual chamaram de **Programação de Memória Adaptativa**. Estes autores observaram

que um grande número de métodos eficientes para solucionar problemas de otimização combinatória compartilha características em comum, tais como:

1. Um conjunto de elite ou uma estrutura de dados especiais que agregam as particularidades das soluções produzidas pelas buscas é memorizado;
2. Uma solução provisória é construída usando os dados em memória;
3. A solução provisória é melhorada usando um algoritmo de busca local ou uma metaheurística mais sofisticada;
4. A nova solução é incluída na memória ou é usada para atualizar as estruturas de dados que memorizam a história da busca.

O uso de memória adaptativa foi um recurso utilizado por diversos trabalhos para melhorar a qualidade da solução produzida pelas heurísticas (GLOVER, 1996), (TAILLARD et al., 2001), (AHMADI; OSMAN, 2005), (MARTÍ; LAGUNA; GLOVER, 2006), (GENDREAU et al., 2006),(OLIVERA; VIERA, 2007), (ARMENTANO; FILHO, 2007), (NEVES, 2007) e (TARANTILIS; KIRANOUDIS, 2007).

Neste trabalho, propõem-se utilizar memória adaptativa através do uso da técnica de reconexão de caminhos (MARTÍ; LAGUNA; GLOVER, 2006), cuja característica principal será a construção e a atualização dinâmica de um conjunto elite (CE).

4.1 GRASP - *Greedy Randomized Adaptive Search Procedure*

A metaheurística GRASP é um método iterativo, multipartida no qual cada iteração consiste de duas fases: uma de construção e outra de busca local (FEO; RESENDE, 1995), (PITSOULIS; RESENDE, 2002) e (RESENDE; RIBEIRO, 2002). Na fase de construção, o GRASP constrói uma solução viável, cuja vizinhança é investigada até um mínimo local ser achado durante a fase de busca local. A melhor solução durante as iterações é retornada como solução final.

O pseudocódigo apresentado no Algoritmo 12 ilustra os principais blocos de um procedimento GRASP para problemas de minimização (RESENDE; RIBEIRO, 2002). O Algoritmo 12, adaptado de (RESENDE; RIBEIRO, 2002), tem como parâmetros de entrada, o número máximo de iterações `Max_Iter` executado pelo GRASP e a semente (`seed`) utilizada como valor inicial para o gerador de números aleatórios.

Algoritmo 12: Procedimento *GRASP*(`Max_Iter`, `seed`)

```
1: Ler_Dados_Entrada( );
2: para k de 1 até Max_Iter faça
3:    S ← Construcao_Aleatoria_Gulosa(seed);
4:    S ← Busca_Local(S);
5:    Atualizar_Solucao(S, S*);
6: fim para
7: Retornar S*;
```

O pseudocódigo da fase de construção é ilustrado pelo Algoritmo 13. A cada iteração dessa fase, o conjunto de elementos candidatos é formado por todos os elementos que podem ser incorporados à solução parcial sem destruir a sua viabilidade. A seleção do próximo elemento a ser incorporado é determinada pela avaliação de todos os elementos candidatos através de uma função de avaliação gulosa.

Essa função gulosa usualmente representa, num problema de minimização, o aumento incremental na função custo devido à incorporação desse elemento dentro da solução construída (aspecto guloso do algoritmo). A avaliação dos elementos por essa função conduz a criação da Lista Restrita de Candidatos (LRC) formada por um subconjunto dos elementos melhor avaliados. O elemento a ser incorporado à solução parcial é selecionado aleatoriamente da LRC (aspecto probabilístico do algoritmo). O elemento selecionado é incorporado à solução parcial, a lista de candidatos é atualizada e os custos incrementais são reavaliados (aspecto adaptativo do algoritmo).

Algoritmo 13: Procedimento *Construcao_Aleatoria_Gulosa*(`seed`)

```
1: S ← ∅;
2: Avaliar os custos incrementais dos elementos candidatos;
3: enquanto S não está completa faça
4:    Construir a Lista Restrita de Candidatos (LRC);
5:    Selecionar um elemento s da LRC aleatório;
6:    S ← S ∪ s;
7:    Reavaliar os custos incrementais;
8: fim enquanto
9: Retornar S;
```

As soluções geradas pela construção aleatória gulosa do GRASP não representam necessariamente ótimos locais. A fase de busca local visa tentar melhorar a solução construída. Um algoritmo de busca local trabalha em um modo iterativo substituindo sucessivamente a solução corrente por uma solução melhor na vizinhança da solução corrente. O algoritmo termina quando nenhuma solução melhor é encontrada na vizinhança.

O pseudocódigo de um algoritmo de busca local básico partindo da solução S construída na primeira fase do GRASP com uma vizinhança $N(S)$ é descrito pelo Algoritmo 14, adaptado de (RESENDE; RIBEIRO, 2002).

Algoritmo 14: *Procedimento Busca_Local*(S)

1: **enquanto** S não é ótima local **faça**
2: Achar $S' \in N(S)$ com $f(S') \leq f(S)$;
3: $S \leftarrow S'$;
4: **fim enquanto**
5: Retornar S;

4.1.1 Construção da Lista Restrita de Candidatos

Uma característica atraente do GRASP é a facilidade para a implementação e a existência de poucos parâmetros a serem ajustados. GRASP possui basicamente dois parâmetros principais, um relacionado ao critério de parada e outro à cardinalidade da LRC. O critério de parada pode ser determinado pelo número máximo de iterações ou um tempo limite de processamento ou ainda por um valor alvo a ser encontrado.

O parâmetro α relacionado a construção da LRC usada na primeira fase do GRASP é descrito a seguir. Considere o problema como sendo de minimização sem perda de generalidade e um conjunto V, $V = \{v_1, v_2, ..., v_n\}$, composto de elementos a serem inseridos numa solução. Designa-se por $c(v_i)$ o custo incremental associado com a incorporação do elemento $v_i \in V$ dentro da solução construída. Representa-se por c^{min} e c^{max}, o menor e o maior custo incremental, respectivamente.

Algoritmo 15: Procedimento *Construcao_Aleatoria_Gulosa*(α, `seed`)

1: $S \leftarrow \emptyset$;
2: Iniciar o conjunto de candidatos: $C \leftarrow V$;
3: Avaliar os custos incrementais $c(v_i)$ para todos $v_i \in C$;
4: **enquanto** $C \neq \emptyset$ **faça**
5: $c^{min} \leftarrow min\{c(v_i)|v_i \in C\}$;
6: $c^{max} \leftarrow max\{c(v_i)|v_i \in C\}$;
7: LRC $\leftarrow \{v_i \in C|c(v_i) \leq c^{min} + \alpha(c^{max} - c^{min})\}$;
8: Selecionar um elemento s da LRC aleatório;
9: $S \leftarrow S \cup \{s\}$;
10: Atualizar o conjunto candidatos C;
11: Reavaliar os custos incrementais $c(v_i)$ para todos $v_i \in C$;
12: **fim enquanto**
13: Retornar S;

A LRC é construída com elementos $v_i \in V$ com os melhores custos incrementais $c(v_i)$. Essa lista pode ser limitada pelo número de elementos ou por sua qualidade. No primeiro caso, a LRC é construída de l elementos com os melhores custos incrementais, onde l é um parâmetro. No segundo caso, considera-se a LRC associada com um parâmetro limiar $\alpha \in [0, 1]$. A LRC é formada por todos os elementos viáveis $v_i \in V$, os quais podem ser inseridos dentro da solução parcial construída sem destruir a viabilidade e cuja qualidade é superior ao valor limiar. Assim, verifica-se: $\{c(v_i) \in [c^{min}, c^{min} + \alpha(c^{max} - c^{min})]\}$.

O caso $\alpha = 0$ corresponde ao algoritmo puramente guloso, enquanto $\alpha = 1$ equivale ao algoritmo totalmente aleatório. O pseudocódigo do Algoritmo 15, adaptado de (RESENDE; RIBEIRO, 2002), é um refinamento do pseudocódigo de construção aleatória gulosa apresentado no Algoritmo 13 mostrando que o parâmetro α controla os pesos de inserção gulosa e de aleatoriedade no algoritmo.

A versão reativa do GRASP tenta determinar dinamicamente os valores de α a serem utilizados. A seguir, a estratégia da escolha do parâmetro α de modo reativo (PRAIS; RIBEIRO, 2000a) e (PRAIS; RIBEIRO, 2000b) é descrita sucintamente. Esta estratégia procura incorporar um mecanismo de aprendizado usando os valores das soluções produzidas ao longo das iterações. Neste caso, o valor de α não será fixo. Seja, $\Psi = \{\alpha_1, ..., \alpha_m\}$, o conjunto dos possíveis valores para α de tamanho m. A probabilidade inicial associada com a escolha de cada valor α_i será igual a p_i=$1/m$, com $i = 1, ..., m$. Seja também z^* a melhor solução encontrada até o momento e z_{m_i} o valor médio de todas as soluções encontradas usando $\alpha = \alpha_i, i = 1, ..., m$. As probabilidades de seleção serão periodicamente reavaliadas, com p_i=$q_i/\sum_{j=1}^{m} q_j$, q_i=z^*/z_{m_i} para $i = 1, ..., m$. O valor de cada q_i será maior quando o valor médio z_{m_i} diminuir. Os maiores valores de q_i correspondem aos valores mais adequados para os α_i. As probabilidades associadas com esses valores mais *apropriados* aumentam à medida que os valores de α são reavaliados.

4.2 Reconexão de Caminhos

A reconexão de caminhos (RC) (*Path Relinking*) foi originalmente proposta por Glover (1996) como uma estratégia de intensificação que explora trajetórias conectando soluções elite obtidas pela Busca Tabu e *Scatter Search*. O uso da reconexão de caminhos com o GRASP foi primeiro proposto por Laguna e Martí (1999) como uma estratégia de intensificação. Partindo de uma ou mais soluções elites, a reconexão de caminhos permite que outras soluções sejam geradas e exploradas percorrendo-se novos caminhos no espaço

de solução. Isso é efetuado selecionando-se os movimentos que introduzem atributos contidos nas soluções guias nas soluções base (PIÑANA et al., 2004).

Para gerar os caminhos, necessita-se selecionar movimentos que partindo de uma solução base introduzem progressivamente atributos da solução guia à solução base de modo a reduzir a diferença entre as soluções. Os critérios para selecionar a solução base e a guia são intercambiáveis (GLOVER; LAGUNA; MARTÍ, 2000) e (GLOVER; LAGUNA; MARTÍ, 2004). Basicamente duas estratégias são usadas com o uso da reconexão de caminhos com o GRASP:

1. Aplicada numa etapa de pós-otimização para todos os pares de soluções elite;
2. Aplicada num processo de intensificação para cada ótimo local obtido depois da fase de busca local.

Na segunda estratégia, a reconexão de caminhos é aplicada ao pares (x_1, x_2) de soluções extremos, onde x_1 é a solução obtida usualmente por uma busca local e x_2 é uma das soluções elite (*conjunto das k melhores soluções obtidas até o momento pelo GRASP*) escolhida aleatoriamente deste conjunto construído ao longo das iterações (RESENDE; RIBEIRO, 2002). A cardinalidade deste conjunto é limitado a um valor máximo (`Max_Elite`).

Uma forma de construir o conjunto de soluções elite (RESENDE; RIBEIRO, 2002), quando o GRASP é utilizado, é descrito a seguir. Originalmente o conjunto é vazio. Cada solução ótima local, obtida pela busca local é candidata a ser inserida no conjunto, caso seja diferente de todas as outras soluções já pertencentes ao conjunto. Se o número de soluções atingir o valor `Max_Elite` e a solução candidata for melhor do que a pior contida no conjunto, esta será substituída. Se o conjunto não está preenchido a solução candidata é inserida dentro deste. Em (RESENDE; WERNECK, 2004) verificou-se que ao aplicar a reconexão de caminhos para um problema de otimização combinatória deve-se considerar não somente a qualidade da solução a ser inserida no conjunto de soluções elite, mas também a diversidade das soluções. Empiricamente os autores Resende e Werneck (2004) observaram que ao aplicar a reconexão de caminhos aos pares de soluções, os resultados apurados não obtiveram sucesso quando as soluções eram muito similares. Um longo caminho percorrido entre os pares de soluções realizado pela reconexão de caminhos, permite maior probabilidade para que um mínimo local seja encontrado.

A Figura 4.1, adaptada de (BOUDIA; LOULY; PRINS, 2007), ilustra um possível caminho percorrido da solução x_1 para x_2 (linha sólida), onde não foram encontradas soluções melhores do que x_1 e x_2 e o caminho da solução x_2 para x_1 (linha pontilhada), no qual

foram encontradas duas melhores soluções. A melhor solução encontrada ao longo dessa

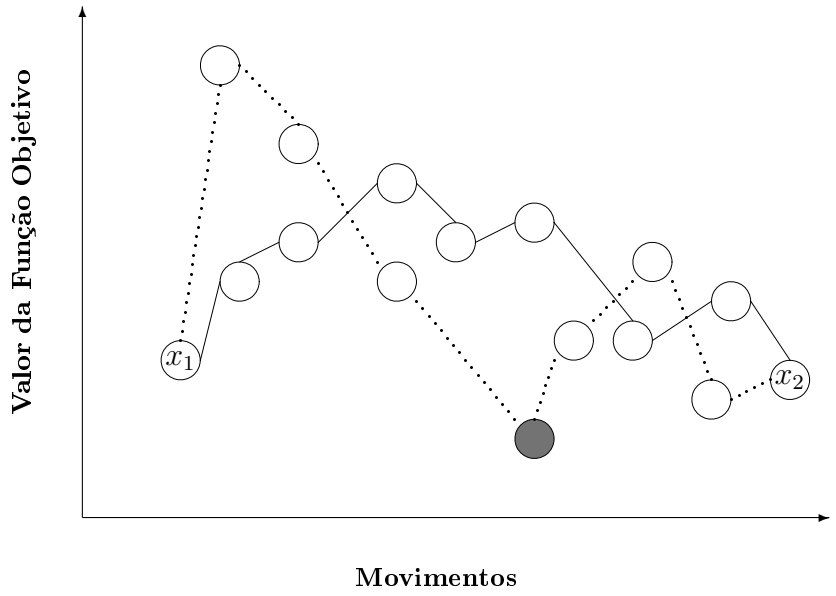


Figura 4.1: Movimentos causados pela reconexão de caminhos, adaptado de (BOUDIA; LOULY; PRINS, 2007).

trajetória (círculo preto) é considerada como solução candidata para ser inserida no conjunto elite. Várias alternativas têm sido considerada em diversas implementações, tais como:

1. Não aplicar a reconexão de caminhos em toda iteração GRASP, mas somente periodicamente;
2. Explorar duas trajetórias diferentes, uma com x_1 considerada como a solução base e a outra com x_2 como solução base;
3. Explorar somente uma trajetória, partindo de x_1 ou x_2;
4. Não permitir a trajetória completa, mas somente parte dela, truncando a reconexão de caminhos.

Todas essas alternativas apresentam compromissos entre tempo computacional e qualidade da solução. Constata-se que caso a solução base seja escolhida como a melhor entre as duas soluções extremos x_1 e x_2, as melhores soluções encontradas no caminho estarão todas mais próximas à solução base do que à solução guia. Assim, a reconexão de caminhos pode parar antes de se atingir o fim da trajetória (RESENDE; RIBEIRO, 2002).

O Algoritmo 16 mostra a reconexão de caminhos realizada entre duas soluções elites, solução base S_b e solução guia S_g. O procedimento $Novos\_Sub\_Conj(S_k, S_j)$ estabelece os subconjuntos novos entre as soluções S_k e S_j definidos pelas diferenças entre estas duas soluções. O Algoritmo 16 estabelece no passo 1 que S_b será a solução inicial (S_{ini}). Em seguida, o Algoritmo 16 analisa todas as diferenças existentes entre a solução base e a solução guia (passos 3 a 9). Depois de escolhido o par de soluções, S_{ini} e S_g, percorre-se o caminho de S_{ini} a S_g e uma diferença entre S_{ini} a S_g é retornada pela solução S_{int} (solução intermediária) através do procedimento `sel_diferencas`(S_{ini}, S_g, S_{int}). A solução S_{int} será a solução inicial da próxima etapa, passo 5. A melhor de todas as soluções intermediárias nas etapas da reconexão de caminhos será a solução de saída, passos 6 à 8.

Algoritmo 16: Reconexão de Caminhos(S_b, S_g)

1: $S_{ini} \leftarrow S_b$;
2: $S_{rc} \leftarrow S_b$;
3: **enquanto** $Novos\_Sub\_Conj(S_{ini}, S_g) \neq \emptyset$ **faça**
4: `sel_diferencas`(S_{ini}, S_g, S_{int});
5: $S_{ini} \leftarrow S_{int}$;
6: **se** S_{ini} for melhor que S_{rc} **então**
7: $S_{rc} \leftarrow S_{ini}$;
8: **fim se**
9: **fim enquanto**
10: Retornar a solução S_{rc};

4.2.1 Implementações das Versões de Reconexão de Caminhos

Propõem-se neste trabalho, quatro versões de reconexão de caminhos a serem incorporadas às diferentes heurísticas GRASP e a possível ampliação destas versões em outras metaheurísticas. Uma das principais características para realização da reconexão de caminhos é a construção de um conjunto elite (CE). Em todas as versões escolhe-se a solução base, cujo custo da solução seja o menor entre as duas soluções extremos. A solução de pior qualidade será a solução guia.

A cardinalidade do conjunto elite é formada por um número fixo pré-definido de soluções e a estratégia para escolher as soluções que irão pertencer ao conjunto elite é a seguinte. Se o conjunto elite estiver incompleto, a solução gerada na iteração do GRASP simplesmente é inserida neste. Se o conjunto elite já estiver completo, a solução gerada na iteração do GRASP é inserida no conjunto elite caso seu custo da função objetivo seja menor que o custo da solução de pior qualidade do conjunto elite e a solução apresente uma diferença mínima na sua estrutura para cada solução presente no CE. Depois de

verificada as duas condições, necessita-se saber qual será a solução que deve ser retirada do conjunto elite. A solução que irá sair deverá ter em relação à solução que entra no conjunto elite um custo maior e uma menor diferença na estrutura entre as duas soluções. A idéia da construção do conjunto elite é ter um conjunto de soluções de boa qualidade.

A primeira versão proposta para a RC realiza uma busca mais intensiva, onde a reconexão é feita entre cada par de soluções elite (RC1). Caso surjam novas soluções nesta recombinação que atualizem o CE, uma nova etapa será realizada entre as novas soluções obtidas. O processo termina quando o CE não for mais atualizado. Este tipo de reconexão de caminhos é aplicada na fase de pós-otimização, depois das iterações do GRASP. A segunda (RC2) é uma versão mais leve onde a reconexão é feita apenas entre a solução da iteração atual do GRASP com uma solução elite extraída do CE. A solução escolhida do CE deverá ter em relação à solução da iteração atual a maior diferença entre suas estruturas. A terceira (RC3) é semelhante a RC1, mas é ativada durante as iterações do GRASP sempre que o CE for totalmente renovado. A RC3 é realizada em cada iteração do GRASP somente quando mudam todas as soluções do CE. A quarta (RC4) é semelhante a RC3, mas sua ativação ocorre toda vez que um percentual de soluções novas é inserido no conjunto elite. Pretende-se escolher um percentual fixo para ativação da reconexão de caminhos RC4 durante as iterações do GRASP. Para isto será feito uma análise deste percentual e definido *a posteriori* no Capítulo 5.

4.3 Método de Descida em Vizinhança Variável

Várias metaheurísticas híbridas utilizam como busca local o Método de Descida em Vizinhança Variável (VND-*Variable Neighborhood Descent*) (HANSEN; MLADENOVIĆ, 2003) para solucionar problemas de otimização combinatória tais como: Problema do Caixeiro Viajante com Coleta de Prêmios (CHAVES; LORENA, 2005; GOMES; DINIZ; MARTINHON, 2000), *Traveling Purchaser Problem* (SILVA; DRUMMOND; OCHI, 2000; DRUMMOND et al., 2002) e Problema do Caixeiro Viajante com Coleta e Entrega de uma Comodidade (HERNÁNDEZ-PÉREZ; RODRÍGUEZ-MARTÍN; SALAZAR-GONZÁLEZ, 2009). Para o PCVG, propõe-se verificar o impacto de se incluir nas metaheurísticas módulos de buscas locais mais eficientes utilizando os conceitos do VND. Neste contexto foram desenvolvidas várias estruturas de vizinhanças para realizar uma fase de busca local baseada no VND.

O Algoritmo 17 descreve o VND proposto em Hansen e Mladenović (2001). No Algoritmo 17 define-se o conjunto de vizinhanças N'_k, k=1, 2, ..., k'_{max} e uma solução inicial x.

Algoritmo 17: *Variable Neighborhood Descent*

repita
  $k \leftarrow 1$
  enquanto $k \leq k'_{max}$ **faça**
    Encontrar a melhor vizinhança x' de x, $(x' \in N'_k(x))$;
    se $f(x') < f(x)$ **então**
      $x \leftarrow x'$;
      $k \leftarrow 1$;
    senão
      $k \leftarrow k + 1$;
    fim se
  fim enquanto
até nenhum melhoramento é obtido

Para realizar o Método de Descida em Vizinhança Variável propõem-se nesta tese, quatro tipos de estruturas de vizinhanças. A primeira N'_1 é obtida através de um movimento de deslocamento do vértice semente no ciclo de uma posição para outra, 1-*Shift*. Para definir a segunda estrutura N'_2 precisa-se definir dois tipos de procedimentos: *Drop* e *Cheap*. O procedimento *Drop* varre o ciclo de uma solução e retira deste o vértice que proporcionará a maior economia com sua retirada. Este tipo de procedimento proposto tem analogia à Heurística das Economias estabelecida por Clarke e Wright (1964) para o PCV.

Algoritmo 18: *Drop*

para k **de** 1 **até** n **faça**
  g_k=c_{a_k,v_k} + c_{v_k,s_k} - c_{a_k,s_k} (computar todos os ganhos de economia para cada vértice v_k);
fim para
$\bar{k} = argmax\{g_k\colon k \in G\}$;
Remover $v_{\bar{k}}$ do ciclo;

O procedimento *Cheap* insere um vértice não pertencente a um ciclo parcial entre dois vértices adjacentes pertencentes ao ciclo, cuja inserção permita um custo adicional mínimo a solução. O ciclo construído após a inserção do vértice deverá permanecer viável. Esse procedimento está relacionado à heurística de construção inserção mais barata desenvolvida para o PCV (REINELT, 1994).

Diante destas duas definições de procedimentos pode-se estabelecer a segunda estrutura de vizinhança N'_2, 1-*DropCheap*. Uma sequência de procedimentos *Drop* e *Cheap* é realizada sobre uma solução até que nenhum melhoramento seja encontrado. Estruturas de vizinhanças semelhantes a do tipo 1-*DropCheap* são vistas em (SILVA; DRUMMOND; OCHI, 2000), (GOMES; DINIZ; MARTINHON, 2000) e (DRUMMOND et al., 2002). O Algo-

Algoritmo 19: *Cheap*

1: **para** i **de** 1 **até** m **faça**
2: $h(k)_i$=$c_{i,k}$ + $c_{k,i+1}$ - $c_{i,i+1}$, (computar todos os custos de inserção do vértice v_k entre os vértices i e $i+1$ no ciclo C);
3: **fim para**
4: $\bar{k} = argmin\{h(k)_i\colon i \in H\}$;
5: Inserir $v_{\bar{k}}$ no ciclo C entre os vértices i e $(i+1)$;

ritmo 18 mostra o procedimento *Drop* considerando-se: $V = \{v_1, v_2, ..., v_n\}$ o conjunto de vértices de um ciclo; g_k o ganho da economia pela retirada do vértice v_k do conjunto V; a_k e s_k o antecessor e sucessor, respectivamente do vértice v_k; e o conjunto de ganhos g_k para cada vértice k, $G = \{g_1, g_2, ..., g_n\}$.

O Algoritmo 19 apresenta o procedimento *Cheap* com as seguintes definições: v_k o vértice a ser inserido; n o total de vértices; o conjunto de vértices $V = \{v_1, v_2, ..., v_n\}$ e $W = \{v_1, v_2, ..., v_m\}$, sendo W=$(V \backslash v_k)$; C o ciclo formado pelos vértices em W; $h(k)_i$ o custo de inserção do vértice v_k entre os vértices i e $(i+1)$ em C; e $H = \{h(k)_1, h(k)_2, ..., h(k)_m\}$ o conjunto de custos $h(k)_i$ de inserção do vértice k entre os vértices i e $(i+1)$.

Algoritmo 20: *Variable Neighborhood Descent* para o PCVG

1: $k \leftarrow 1$
2: **enquanto** $k \leq 4$ **faça**
3: **se** $k = 1$ **então**
4: $x^{'} \leftarrow$ 1-*Shift(x)*
5: **senão se** $k = 2$ **então**
6: $x^{'} \leftarrow$ 1-*DropCheap(x)*
7: **senão se** $k = 3$ **então**
8: $x^{'} \leftarrow$ 2-*Swap(x)*
9: **senão**
10: $x^{'} \leftarrow$ 2-*Optimal(x)*
11: **fim se**
12: **se** $f(x^{'}) < f(x)$ **então**
13: $x \leftarrow x^{'}$;
14: $k \leftarrow 1$;
15: **senão**
16: $k \leftarrow k + 1$;
17: **fim se**
18: **fim enquanto**
19: Retornar x;

A terceira vizinhança $N_3^{'}$ é obtida através de uma permuta entre os vértices de um mesmo grupo (*cluster*), 2-*Swap*. A 2-*Swap* foi aplicada em outras variantes do PCV (DRUMMOND et al., 2002) e (HERNÁNDEZ-PÉREZ; RODRÍGUEZ-MARTÍN; SALAZAR-

GONZÁLEZ, 2009). A quarta vizinhança N_4' utiliza à heurística 2-*Optimal* (REINELT, 1994) para trocar as arestas intragrupos e as intergrupos. Nesta quarta vizinhança quando são retiradas as arestas intragrupos e é recontruída uma nova solução, somente a sequência dos vértices dentro do grupo é alterada e quando as arestas intergrupos são trocadas, somente a sequência dos grupos é alterada. Diante das definições anteriores, o método de descida VND proposto a ser incluído nas metaheurísticas como módulo de busca local é descrito pelo Algoritmo 20.

4.4 Heurísticas GRASP para o PCVG

Seis heurísticas GRASP foram desenvolvidas para o PCVG. A primeira versão (G1) utiliza a estrutura do GRASP tradicional, onde em cada iteração executa-se uma fase de construção de uma solução e uma fase de busca local. Na etapa de construção do G1, utiliza-se o método de Inserção mais Próxima com as arestas penalizadas, cujos nós extremos pertençam a subconjuntos V_i distintos sendo o valor de α calculado pela versão reativa e na busca local o G1 utiliza o método 2-Optimal desenvolvido originalmente para o PCV.

A metaheurística GRASP tem obtido bons resultados para diferentes problemas de otimização, mas um dos gargalos deste método na sua forma tradicional (G1) é a ausência de memória entre suas iterações. Ou seja, a cada nova iteração uma nova solução é obtida sem utilizar informação das soluções obtidas em iterações anteriores.

Neste trabalho, utiliza-se memória adaptativa no GRASP através do uso e atualização dinâmica de um conjunto elite (CE) que supre os dados de entrada para a técnica de reconexão de caminhos (*path relinking*). As reconexões de caminhos (RC1, RC2, RC3 e RC4) utilizadas nos GRASP (G2, G3, G4, G5 e G6') foram definidas previamente na seção 4.2.1 e em todas estas versões se utiliza o mesmo construtivo do G1.

Durante a execução do GRASP é gerado e atualizado um CE que armazena as num_eli melhores soluções distintas encontradas pelo GRASP. Tal procedimento equivale a usar uma memória que armazene no CE informações relevantes encontradas em iterações passadas. Como este conjunto sofre atualizações sempre que uma nova solução de boa qualidade é encontrada entre as iterações do GRASP, esta *memória é também adaptativa*. Cabe ressaltar que em algumas heurísticas desenvolvidas neste trabalho após a finalização das iterações do GRASP é realizado a reconexão de caminhos entre todos os pares de solução do CE, *fase de pós-otimização*. Este CE pode sofrer atualização nas

soluções armazenadas em memória, o que torna nesta fase de pós-otimização, a *memória continuar a ser adaptativa.*

A segunda versão (G2) introduz em G1 a RC1 ao final da execução de G1. A terceira versão (G3) incorpora ao G1 a reconexão de caminhos (RC2) a cada iteração de G1. A quarta heurística (G4) introduz a RC2 durante a execução das iterações GRASP e ao final do GRASP ativa-se a RC1. A quinta heurística (G5) utiliza todos os módulos do G3, exceto o método 2-Optimal que é substituído pelo VND, e a reconexão de caminhos RC2 é substituída pela RC3. A sexta versão (G6') utiliza a RC4 durante as iterações do GRASP.

Algoritmo 21: Algoritmo GRASP para o PCVG.

1: Início;
2: $iterprob \leftarrow 1$; (incremento para atingir o valor num_prob)
3: $iter \leftarrow 1$; (incremento para atingir o valor num_iter)
4: **enquanto** $iter <= num\_iter$ **faça**
5: α= `calcularalfa( )`; (calcular alfa pela versão reativa)
6: $S' \leftarrow Construcao\_Aleatoria\_Gulosa$;
7: $S' \leftarrow Busca\_Local\ (S')$;
8: **se** $iter > num\_eli$ **então**
9: $S' \leftarrow Rec\_Cam(U^{elite}, S')$;
10: **fim se**
11: **se** S' for melhor que S^* **então**
12: $S^* \leftarrow S'$;
13: **fim se**
14: $U^{elite} \leftarrow Max\_Sol\_Elite(S', iter)$; (conjunto de soluções elites)
15: **se** $iterprob = maxprob$ **então**
16: `atualizardisalfas( )`; (atualizar a distribuição de probabilidade dos alfas)
17: $iterprob \leftarrow 0$;
18: **fim se**
19: iterprob++;
20: iter++;
21: **fim enquanto**
22: $S^{pos} \leftarrow Pos\_Otim(U^{elite})$;
23: **se** S^{pos} for melhor que S^* **então**
24: $S^* \leftarrow S^{pos}$;
25: **fim se**
26: Retornar Solução S^*;
27: Fim.

As versões G2, G3 e G4 utilizam na busca local o método 2-Optimal. Em G5 e G6', o método 2-Optimal foi substituído pela metaheurística VND (HANSEN; MLADENOVIĆ, 2003). Para o VND utilizam-se quatro tipos de estruturas de vizinhanças definidas anteriormente na seção 4.3. A primeira N_1' denominada de 1-*Shift*, a segunda N_2' (1-*DropCheap*),

a terceira N_3' (2-*Swap*) e a quarta N_4' utiliza à heurística 2-Optimal.

Nesta proposta de trabalho, adotam-se as seguintes estratégias ao VND para reduzir o esforço computacional decorrente do uso das estruturas de vizinhanças. Quando se utilizam as vizinhanças N_1' e N_3', adota-se a estratégia *first improvement* (ou seja, na primeira melhora ocorre o movimento para esta solução aprimorante) (HANSEN; MLADENOVIĆ, 2003), enquanto quando utilizam-se N_2' e a N_4', adota-se o método *best improvement* (analisam-se todas as soluções vizinhas e ocorre o movimento para a melhor).

As heurísticas GRASP (G1, G2, G3, G4, G5 e G6') para o PCVG estão representadas no Algoritmo 21 com todos os componentes: construção da solução, busca local e reconexão de caminhos. Para o Algoritmo 21 considera-se: S^* a melhor solução; num_prob o n^o máximo de iterações para mudar a distribuição de probabilidade dos alfas; num_iter o n^o máximo de iterações; e num_eli o n^o máximo da cardinalidade do conjunto elite; O procedimento $Max\_Sol\_Elite(S', iter)$ (Algoritmo 22) representa a construção do conjunto de soluções elites e o $Rec\_Cam(U^{elite}, S')$ representa a reconexão de caminhos durante as iterações do GRASP (RC2, RC3 e RC4).

A reconexão de caminhos ao final do GRASP está representada pelo procedimento $Pos\_Otim(U^{elite})$ que simboliza a fase de pós-otimização (RC1). O Algoritmo 21 mostra ainda os procedimentos `calcularalfa( )`, utilizado para selecionar o alfa a ser utilizado pela versão reativa e `atualizardisalfas( )`, para atualizar a distribuição de probabilidade dos alfas.

GRASP	$Busca\_Local$	$Rec\_Cam(U^{elite}, S')$	$Pos\_Otim(U^{elite})$
G1	2-Optimal	—	—
G2	2-Optimal	—	RC1
G3	2-Optimal	RC2	—
G4	2-Optimal	RC2	RC1
G5	VND	RC3	—
G6'	VND	RC4	—

Tabela 4.1: Heurísticas GRASP com seus procedimentos.

A Tabela 4.1 descreve resumidamente as heurísticas GRASP para o PCVG com os procedimentos de busca local e de reconexão de caminhos utilizados no Algoritmo 21.

O Algoritmo 22 descreve o procedimento para construção do conjunto elite. Os parâmetros de entrada são: S' a solução do GRASP e num é uma variável que contabiliza a ordem no qual as soluções são geradas. O retorno deste algoritmo é o conjunto de soluções elites (U^{elite}=$\{S_1, ..., S_{maxelite}\}$). Ainda para o Algoritmo 22 necessita-se definir os parâmetros: $maxelite$ o n^o máximo de soluções do conjunto elite; $mindif$ a mínima

diferença permitida entre duas soluções; $valordif$ o valor da diferença entre as soluções; $cont$ contador das soluções, cuja diferença mínima foi alcançada; e $dif(S_i, S_j)$ o cálculo da diferença entre a solução S_i e S_j.

Ainda, para o Algoritmo 22, caso o conjunto elite (CE) esteja incompleto, a solução (S') é inserida neste (passos de 1 a 4). Caso o CE estiver completo, a solução a ser inserida (S') no conjunto deverá ter o seu custo da função objetivo $f(S')$ menor do que o custo da solução de pior qualidade ($S_{maxelite}$) do CE e deve apresentar uma diferença mínima ($mindif$) nos seus elementos para cada solução presente no conjunto U^{elite}=$\{S_1, ..., S_{maxelite}\}$ (passos de 5 a 12).

Algoritmo 22: $Max\_Sol\_Elite(S', num)$ - Construção do Conjunto Elite

1: **se** $num <= maxelite$ **então**
2: $U^{elite} \leftarrow U^{elite} \cup \{S'\}$;
3: Reordenar U^{elite}; (reordenar o conjunto elite)
4: **senão**
5: **se** $f(S') < f(S_{maxelite})$ **então**
6: $cont \leftarrow 0$;
7: **para** j **de** 1 **até** $maxelite$ **faça**
8: Seleciona S_j em U^{elite};
9: **se** $dif(S', S_j) > mindif$ **então**
10: $cont++$;
11: **fim se**
12: **fim para**
13: **se** $cont$=$maxelite$ **então**
14: $valordif \leftarrow dif(S', S_{maxelite})$;
15: $S_{out} \leftarrow S_{maxelite}$;
16: **para** j **de** 1 **até** $maxelite$ **faça**
17: Seleciona S_j em U^{elite};
18: **se** $(f(S') < f(S_j))$ **and** $(dif(S', S_j) < valordif)$ **então**
19: $valordif \leftarrow dif(S', S_j)$;
20: $S_{out} \leftarrow S_j$; (solução a ser retirada do conjunto elite)
21: **fim se**
22: **fim para**
23: $U^{elite} \leftarrow U^{elite} \setminus \{S_{out}\}$;
24: $U^{elite} \leftarrow U^{elite} \cup \{S'\}$;
25: Reordenar U^{elite};
26: **fim se**
27: **fim se**
28: **fim se**
29: Retornar o conjunto U^{elite};

Após as duas condições terem sido atendidas, necessita-se encontrar a solução a ser retirada S_{out} do conjunto elite. A solução que irá ser retirada deve ter em relação à solução

a ser inserida no CE um custo maior e a menor diferença ($valordif$) nos elementos entre estas duas soluções (passos 13 a 26). O Algoritmo 22 no final retorna o conjunto U^{elite}, passo 29.

4.5 ILS - *Iterated Local Search*

As características da metaheurística ILS são simplicidade, eficiência, facilidade de implementação e robustez, (LOURENÇO; MARTIN; STÜTZLE, 2002). Na literatura encontra-se o uso do ILS para problemas acadêmicos até aplicações comerciais, industriais e prestação de serviços (CODENOTTI et al., 1996), (STÜTZLE, 1998), (SOUZA et al., 2005), (STÜTZLE, 2006), (CORDEAU; LAPORTE; PASIN, 2008), (DONG; HUANG; CHEN, 2008), (ERDOGAN; CORDEAU; LAPORTE, 2008), (HASHIMOTO; YAGIURA; IBARAKI, 2008), (IBARAKI et al., 2008), (SUBRAMANIAN; CABRAL, 2008), (BRITO et al., 2010), (SOUZA et al., 2010) and (SUBRAMANIAN et al., 2010).

A principal característica do ILS estava presente no algoritmo desenvolvido por (MARTIN; OTTO, 1996) no qual combinava *Simulated Annealing* com métodos de busca local, processo de *cadeia de Markov*. ILS foi tratado para diversos problemas com outras formas e nomes distintos, por exemplo, *large-step Markov chains* (MARTIN; OTTO; FELTEN, 1991) e (MARTIN; OTTO; FELTEN, 1992), *chained local optimization* (MARTIN; OTTO, 1996), mas que na sua essência possuía módulos contidos no ILS atual. Sua estrutura canônica atual é formada por quatro módulos principais: construção da solução, busca local, perturbação e critério de aceitação. Cada módulo pode ser tratado de forma independente possibilitando maior facilidade em sua atualização. É claro que o sucesso do ILS depende exclusivamente do projeto de cada módulo.

ILS busca por soluções em várias bacias de atrações de mínimos locais que são regiões onde um conjunto de soluções é mapeado para ótimos locais sobre um módulo de busca local (LOURENÇO; MARTIN; STÜTZLE, 2002). Para acessar estas bacias é importante o processo de perturbação. A partir de certo número de iterações, as perturbações não conseguem alcançar todas as bacias de atrações e o ILS entra em estagnação. Assim, é necessário incorporar novas estratégias para acessar estas bacias. Uma das estratégias de permitir que as perturbações possam alcançar tais bacias de atrações seria reiniciarmos o ILS com soluções novas. Uma outra estratégia para escapar da estagnação seria trabalhar com um conjunto de soluções, característica típica dos métodos evolutivos.

A maneira como ILS explora o espaço de soluções de ótimos locais é definido a se-

guir: primeiro aplica-se uma perturbação à solução corrente S que permite um estado intermediário S'. Sobre S', aplica-se uma busca local obtendo-se S''. Se a solução S'' for considerada pelo critério de aceitação, esta será a próxima solução a sofrer a perturbação, caso contrário é retornada a solução S (LOURENÇO; MARTIN; STÜTZLE, 2002). O resultado disto é que o ILS realiza um caminho de busca estocástica. As perturbações ocorridas sobre as soluções não deverão ser tão pequenas implicando em poucas soluções novas permanecendo na mesma bacia de atração e não deverão ser grandes permitindo bacias de atrações aleatórias. Como a cada iteração é realizada uma varredura por várias bacias de atrações via perturbação, podem ocorrer ciclos neste processo. Neste sentido, deve-se evitar que estes ciclos possam ocorrer. Para isto as perturbações deverão ter elementos aleatórios ou serem adaptativas.

Algoritmo 23: Procedimento *Iterated Local Search.*

1: Início;
2: $S^0 \leftarrow Construcao\_Aleatoria\_Gulosa$;
3: $S \leftarrow Busca\_Local(S^0)$;
4: $S^* \leftarrow S$
5: **repita**
6: $S' \leftarrow Perturbacao(S,história)$;
7: $S'' \leftarrow Busca\_Local(S')$;
8: $S \leftarrow Criterio\_Aceitacao(S,S'',história)$;
9: **se** S for melhor que S^* **então**
10: $S^* \leftarrow S$;
11: **fim se**
12: **até** critério de parada não satisfeito
13: Retornar Solução S^*;
14: Fim.

Como as perturbações dependem sensivelmente das soluções obtidas nas etapas anteriores guiadas pelo critério de aceitação, pode-se dizer que ILS incorpora mecanismo de memória. Mas, a perturbação e o critério de aceitação poderão não depender das soluções geradas nas etapas anteriores e nem dos elementos incorporados nas soluções. Assim, o ILS ficará sem o uso da "história" e sem o uso de memória. Esta alternativa não é recomendável, pois maiores sucessos são obtidos pelas metaheurísticas que incorporam mecanismo de memória. ILS em síntese dependerá do sucesso da perturbação, do critério de aceitação e das amostragens de várias soluções obtidas por buscas locais.

Uma das grandes vantagens do ILS está na estrutura modular de cada componente. Assim, podem-se desenvolver os módulos independentes e incorporar complexidades maiores a cada um com a finalidade de tentar obter soluções de alta qualidade. ILS torna-se muito promissor quando comparado a métodos de partidas aleatórias. ILS é descrito

pelo Algoritmo 23 baseada no *framework* discutido em (LOURENÇO; MARTIN; STÜTZLE, 2002). No Algoritmo 23, a solução S^* representa a melhor solução encontrada e o módulo *Construcao_ Aleatoria_ Gulosa* representa a construção de solução. No Algoritmo 23 são mostrados ainda, os módulos perturbação, busca local, e critério de aceitação, passos 6, 7 e 8, respectivamente.

4.5.1 Heurísticas ILS para o PCVG

Para implementar heurísticas ILS aplicada ao PCVG foram desenvolvidos diversos métodos de buscas locais, de perturbações e critério de aceitação, conforme módulos presentes no Algoritmo 23. Para os métodos de buscas locais foram propostos a 2-Optimal, 3-Optimal e o VND. Um tipo de perturbação utilizada no PCV com sucesso foi o movimento *double-bridge* (MARTIN; OTTO; FELTEN, 1992). Este tipo de perturbação remove quatro arestas na solução e reconstrói uma nova solução substituindo as arestas removidas por outras arestas para obter um movimento não sequencial nas novas arestas inseridas.

Para o PCVG a perturbação utilizada no Algoritmo 23 foi realizada sob três formas: a primeira denominada *Pert1* utiliza o movimento *double-bridge* aplicado às arestas que interligam grupos distintos. São selecionadas quatro arestas aleatórias, mas que conectam as extremidades de grupos diferentes. No final desta perturbação a solução contém uma nova sequência de grupos, sem alterar os vértices internos de cada grupo. A segunda (*Pert2*) realiza uma troca aleatória entre dois grupos sem alterar os vértices internos dos grupos. A terceira (*Pert3*) realiza o movimento *double-bridge* aplicado aos vértices em um grupo escolhido aleatório. A sequência dos grupos não é alterada, somente a sequência dos vértices do grupo escolhido é modificada.

O primeiro critério de aceitação (*CritAceit1*) construído na etapa seguinte da perturbação foi escolhido entre as duas soluções ótimas locais, S e S'', descritas anteriormente pelo Algoritmo 23. A solução que tiver o menor custo será escolhida. Então, pode-se definir o primeiro critério como:

$$CritAceit1(S, S'', história) = \begin{cases} S & \text{se} \quad c(S) < c(S''), \\ S'' & \text{caso contrário,} \end{cases} \tag{4.1}$$

Observa-se que no primeiro critério de aceitação (*CritAceit1*) existe a incorporação de *história recente* no algoritmo devido às duas soluções ótimas (S e S''). Neste caso, a história recente considera todos os vértices de cada solução S e S'' que produz o menor

custo entre estas duas soluções. Mas, no critério *CritAceit1* não há nenhuma relação ligada à *história remota* do algoritmo realizada até o momento de decisão da escolha entre as duas soluções ótimas locais. Para incorporar história remota no segundo critério de aceitação, admite-se uma outra solução a ser incorporada.

A solução a ser admitida neste segundo critério é a solução S^*, a melhor solução até a etapa corrente do Algoritmo 23. Para o segundo critério de aceitação (*CritAceit2*) a solução que tiver o menor custo entre a três soluções (S, S'', S^*) será escolhida como solução para a etapa seguinte da perturbação. Assim, temos:

$$CritAceit2(S, S'', história) = \begin{cases} S & \text{se} \quad c(S) < c(S'') \quad \text{e} \quad c(S) < c(S^*), \\ S'' & \text{se} \quad c(S'') < c(S) \quad \text{e} \quad c(S'') < c(S^*), \\ S^* & \text{caso contrário,} \end{cases} \tag{4.2}$$

Capítulo 5

Resultados Computacionais

Neste capítulo são descritos os resultados dos testes computacionais efetuados para a solução do PCVG. Serão mostrados resultados utilizando a formulação matemática da literatura proposta por Chisman (1975), as heurísticas de construção, as diversas versões GRASPs desenvolvidas e as heurísticas implementadas utilizando-se *Iterated Local Search*.

Encontram-se na literatura métodos exatos e heurísticos para o PCVG, no entanto não se tem conhecimento de nenhuma biblioteca pública com instâncias do problema na sua versão genérica sem fixação da ordem de visitas aos grupos. Desta forma, tornou-se necessária a criação de um conjunto de instâncias para o PCVG genérico para avaliar os métodos aqui propostos, sendo estas instâncias disponíveis através de acesso em <ftp://ftp.cefetes.br/Teses_Dissertacoes/MarioMestria/instances/>.

Foram geradas instâncias de seis tipos: (*tipo 1*) - adaptação de instâncias do PCV (TSPLIB95, 2007) agrupadas usando o algoritmo de clusterização *k-means*; (*tipo 2*) - adaptação de instâncias disponíveis na literatura para o PCV, agrupando-se os vértices em torno de um centro geométrico, com o total de centros geométricos igual a $n/100$ (n - número de vértices) e as coordenadas são escolhidas no intervalo $[0,10^6)$ (JOHNSON; McGEOCH, 2002); (*tipo 3*) - as instâncias são geradas através da interface Concorde (APPLEGATE et al., 2007); (*tipo 4*) - as instâncias são geradas usando *layout* conforme proposto em Laporte, Potvin e Quilleret (1996); (*tipo 5*) - instâncias semelhantes às do *tipo 2*, mas geradas com parâmetros diferentes dos realizados em (JOHNSON; McGEOCH, 2002); (*tipo 6*) - instâncias do PCV (TSPLIB95, 2007), onde a área plana retangular que compõe a instância é dividida em diversos quadriláteros e cada quadrilátero corresponde a um grupo.

Os detalhes de como são formadas as instâncias e os formatos dos arquivos são apresentados no **Apêndice A**. Estas seis formas diferentes de instâncias contidas neste conjunto,

sem fixação da ordem de visitas aos grupos, para o PCVG são instâncias euclidianas e simétricas. Os resultados computacionais apresentados nas seções seguintes foram obtidos para este conjunto de instâncias para verificar quais das heurísticas desenvolvidas produzem melhores resultados para o PCVG.

5.1 Resultados utilizando as Formulações Matemáticas

Devido a inexistência de biblioteca de instâncias para o PCVG, e também para melhor avaliar o desempenho das heurísticas aqui propostas, para instâncias de pequeno e médio porte nesta tese foi usada formulação matemática do PCVG e o *software* CPLEX (CPLEX, 2009) com a finalidade de encontrar soluções ótimas de parte das instâncias consideradas. Neste sentido, foram utilizadas as formulações (2.1 a 2.6) e (2.7 a 2.13), descritas na seção 2.3 proposta por Chisman (1975). Na formulação 2.1 a 2.6 de Dantzig/Chisman suas restrições crescem exponencialmente na proporção de 2^n, enquanto que na formulação 2.7 a 2.13 de Miller/Chisman suas restrições crescem na proporção polinomial (n^2). Para as instâncias do *tipo 5*, *1* e *6* mostradas a seguir, o CPLEX Paralelo, versão 11.2, foi executado num computador com 4 núcleos Intel Core 2 Quad, 2.83 GHz com 8 GB de RAM num sistema operacional Linux Ubuntu versão 4.3.2-1. A partir de agora o CPLEX Paralelo, versão 11.2, será mencionado simplesmente por CPLEX.

#nós	$\#V_i$	**Instância**	Dantzig/Chisman		Miller/Chisman	
			Valor	Tempo (s)	Valor	Tempo (s)
10	2	i-10-2-1	3973	< 1	3973	< 1
11	2	i-11-2-1	3861	< 1	3861	< 1
12	2	i-12-2-1	3839	< 1	3839	< 1
13	2	i-13-2-1	4471	< 1	4471	< 1
14	3	i-14-3-1	4287	1	4287	< 1
15	3	i-15-3-1	4393	3	4393	< 1
16	4	i-16-4-1	5022	9	5022	< 1
17	4	i-17-4-1	5251	24	5251	< 1
18	4	i-18-4-1	5167	58	5167	< 1
19	4	i-19-4-1	5100	147	5100	< 1
20	4	i-20-4-1	5044	351	5044	< 1

Tabela 5.1: Valores dos custos e tempos alcançados pelas duas formulações utilizando instâncias do *tipo 5*.

Na Tabela 5.1 são mostrados as comparações do tempo computacional medido em segundos demandado pelo CPLEX usando as duas formulações: Dantzig/Chisman e Miller/Chisman. As duas formulações alcançaram valores ótimos. Na primeira coluna da Tabela 5.1 é mostrado o número de vértices, na segunda o número de grupos, na ter-

ceira coluna as instâncias do *tipo 5*, na quarta e quinta colunas o valor encontrado no custo da função objetivo e o tempo computacional, respectivamente usando a formulação Dantzig/Chisman. Nas duas últimas colunas apresentam-se, o valor e o tempo computacional usando a formulação Miller/Chisman. Em ambas formulações os valores dos tempos computacionais alcançados pelo CPLEX, para algumas instâncias, foram bem pequenos e menores do que um segundo (1s), sendo representado na Tabela 5.1 por (< 1).

A Tabela 5.1 mostra que a formulação Dantzig/Chisman demanda um tempo maior para encontrar a solução ótima do que a formulação de Miller/Chisman. Observa-se que o tempo passa de menos de um segundo para 351 segundos quando varia-se o número de vértices das instâncias de 10 para 20. Já na formulação de Miller/Chisman o tempo para encontrar a solução é sempre menor que um segundo. A partir de instâncias com número de vértices maior que 20 não foi possível encontrar soluções no modelo de Dantzig/Chisman, devido não ser viável armazenar o problema na memória utilizando a formulação (2.1 a 2.6), principalmente devido à restrição 2.4.

Escolhe-se o modelo de Miller/Chisman que possibilitou encontrar soluções através do CPLEX sem ocorrer estouro de memória. Foram realizados testes computacionais a diversos *tipos* de instâncias utilizando a formulação 2.7 a 2.13 sem penalizar as arestas intergrupos, denominada simplesmente de Miller/Chisman e a formulação 2.7 a 2.10 mais as restrições 2.12 e 2.13 com as arestas intergrupos penalizadas, denominada de Miller/Chisman/Pen.

A seguir, é mostrada na Tabela 5.2 a comparação entre os tempos computacionais usando as formulações dos modelos Miller/Chisman e Miller/Chisman/Pen demandado pelo CPLEX para diversos tipos de instâncias. Na primeira coluna da Tabela 5.2 é mostrado o número de vértices, na segunda o número de grupos, na terceira coluna as instâncias, na quarta coluna o *tipo* de instância e nas duas últimas colunas os tempos computacionais obtidos pelo CPLEX em segundos com a formulação sem penalizar as arestas inter-grupos e com a formulação que penaliza as arestas inter-grupos, respectivamente. Os valores alcançados pelo CPLEX na função objetivo foram iguais em ambas formulações para todas as instâncias e estão mostrados nas Tabelas 5.3, 5.4 e 5.5.

#nós	$\#V_i$	Instâncias	Tipo	Tempo sem Pen.(s)	Tempo com Pen.(s)
51	5	5-eil51	1	12	13
51	10	10-eil51	1	74	70
51	15	15-eil51	1	2	1

(*Continua na próxima página.*)

#nós	$\#V_i$	Instâncias	Tipo	Tempo sem Pen.(s)	Tempo com Pen.(s)
52	5	5-berlin52	1	202	152
52	10	10-berlin52	1	89	56
52	15	15-berlin52	1	76	91
70	15	15-st70	1	884	605
76	5	5-eil76	1	84	91
76	10	10-eil76	1	254	188
76	15	15-eil76	1	50	35
76	5	5-pr76	1	99	301
76	10	10-pr76	1	238	286
76	15	15-pr76	1	262	163
99	10	10-rat99	1	651	800
99	25	25-rat99	1	351	343
99	50	50-rat99	1	2798	2305
100	25	25-kroA100	1	3514	367
100	50	50-kroA100	1	948	543
100	50	50-kroB100	1	2579	704
101	25	25-eil101	1	709	150
101	50	50-eil101	1	275	357
105	50	50-lin105	1	1577	2801
tempo médio (*tipo 1*) (s)				714,91	**473,73**
30	5	i-30-5-17	5	1	1
30	7	i-30-7-17	5	3	2
30	10	i-30-10-17	5	1	1
45	5	i-45-5-18	5	15	8
45	7	i-45-7-18	5	81	115
45	10	i-45-10-18	5	60	99
60	5	i-60-5-21	5	527	829
60	7	i-60-7-21	5	711	176
60	10	i-60-10-21	5	643	1220
65	5	i-65-5-21	5	1112	535
65	10	i-65-10-21	5	118	143
70	5	i-70-5-21	5	1994	250

(*Continua na próxima página.*)

#nós	$\#V_i$	Instâncias	Tipo	Tempo sem Pen.(s)	Tempo com Pen.(s)
70	10	i-70-10-21	5	701	2446
tempo médio (*tipo 5*) (s)				459,00	**448,08**
51	4	4-eil51-2x2	6	8	8
51	9	9-eil51-3x3	6	8	8
51	12	12-eil51-3x4	6	1	1
51	16	16-eil51-4x4	6	9	11
51	20	20-eil51-4x5	6	9	14
51	25	25-eil51-5x5	6	1	5
52	4	4-berlin52-2x2	6	226	210
52	6	6-berlin52-2x3	6	204	329
52	8	8-berlin52-2x4	6	343	488
52	10	10-berlin52-2x5	6	840	375
70	12	12-st70-3x4	6	2960	2167
70	20	20-st70-4x5	6	545	1733
76	4	4-eil76-2x2	6	110	280
76	9	9-eil76-3x3	6	204	165
76	12	12-eil76-3x4	6	65	45
76	16	16-eil76-4x4	6	26	35
76	20	20-eil76-4x5	6	48	28
76	25	25-eil76-5x5	6	28	27
76	4	4-pr76-2x2	6	3482	1751
76	9	9-pr76-3x3	6	197	200
76	12	12-pr76-3x4	6	34	42
76	15	15-pr76-3x5	6	430	3465
76	18	18-pr76-3x6	6	1019	931
99	25	25-rat99-5x5	6	1000	992
99	42	42-rat99-6x7	6	2521	2048
100	30	30-kroB100-5x6	6	368	561
101	9	9-eil101-3x3	6	1441	703
101	36	36-eil101-6x6	6	522	858
tempo médio (*tipo 6*) (s)				**594,61**	624,29
tempo médio total (s)				608,64	**535,35**

Tabela 5.2: Tempos alcançados pelo CPLEX.

Das 22 instâncias do *tipo 1*, 14 obtiveram tempos computacionais menores quando utilizam a formulação que penaliza as arestas inter-grupos, comparados aos tempos computacionais na formulação que não penaliza. O tempo computacional médio quando não se penalizam as arestas ficou em 714,91s e quando se penalizam em **473,73s**. Para as instâncias do *tipo 5*, seis obtiveram tempos computacionais menores quando utilizam a formulação que não penaliza as arestas em um total de 13, comparados aos tempos computacionais na formulação que penaliza. Dois tempos computacionais foram iguais em ambas formulações. O tempo computacional médio quando não se penalizam as arestas ficou em 459,00s e quando se penalizam em 448,08s. Os tempos médios para as instâncias do *tipo 5* foram bem próximos.

#nós	$\#V_i$	Instância	Valor	***gap*** (%)	***status***
30	5	i-30-5-17	5193	0,01	sol. ótima
30	7	i-30-7-17	7940	0,01	sol. ótima
30	10	i-30-10-17	6772	0,00	sol. ótima
45	5	i-45-5-18	6879	0,01	sol. ótima
45	7	i-45-7-18	7478	0,01	sol. ótima
45	10	i-45-10-18	8258	0,01	sol. ótima
60	5	i-60-5-21	7824	0,01	sol. ótima
60	7	i-60-7-21	8623	0,01	sol. ótima
60	10	i-60-10-21	9317	0,01	sol. ótima
65	5	i-65-5-21	7680	0,01	sol. ótima
65	7	i-65-7-21	8732	0,10	sol. encontrada
65	10	i-65-10-21	9740	0,01	sol. ótima
70	5	i-70-5-21	7793	0,01	sol. ótima
70	7	i-70-7-21	9013	0,07	sol. encontrada
70	10	i-70-10-21	9746	0,01	sol. ótima
75	5	i-75-5-22	7327	1,06	sol. encontrada
75	10	i-75-10-22	9278	0,40	sol. encontrada
90	5	i-90-5-33	6678	0,58	sol. encontrada
90	10	i-90-10-33	8727	0,31	sol. encontrada
120	5	i-120-5-46	7890	1,23	sol. encontrada
120	10	i-120-10-46	9734	0,21	sol. encontrada

Tabela 5.3: Valores da função objetivo para as instâncias do *tipo 5*

Nas instâncias do *tipo 6*, de um total de 28, 13 instâncias obtiveram tempos computacionais menores ao utilizar a formulação que não penaliza as arestas inter-grupos. Em três instâncias os tempos computacionais foram iguais. O tempo computacional médio quando não se penalizam as arestas ficou em **594,61s** e quando se penalizam em 624,29s. Observa-se que para as instâncias do *tipo 6* a penalização aplicada nas arestas inter-grupos não é uma estratégia promissora. Considerando todas as instâncias da Tabela 5.2, o tempo computacional médio total ficou em 608,64s ao utilizar a formulação que não penaliza as arestas inter-grupos e de **535,35s** quando se penalizam.

Na Tabela 5.3 apresentam-se os resultados obtidos para as instâncias do *tipo 5* geradas variando o número de vértices de 30 a 120 e o número de grupos de 5 a 10 vértices. Na Tabela 5.3 é mostrado na primeira coluna o numero de vértices, na segunda o número de grupos, na terceira a instância, na quarta o valor alcançado pelo CPLEX, na quinta o percentual do *gap* calculado pelo CPLEX e na última coluna o *status* da solução. Para a instância i-65-7-21 com 65 vértices, para a instância i-70-7-21 com 70 vértices e todas as instâncias contendo 75, 90 e 120 vértices não foram encontradas soluções ótimas. Para as demais, o CPLEX alcançou soluções ótimas. O tempo máximo estabelecido ao CPLEX para encontrar soluções ótimas ou não, foi de 7200 segundos.

#nós	$\#V_i$	Instâncias	Valor
51	5	5-eil51	437
51	10	10-eil51	440
51	15	15-eil51	437
52	5	5-berlin52	7991
52	10	10-berlin52	7896
52	15	15-berlin52	8049
70	5	5-st70	695
70	10	10-st70	691
70	15	15-st70	692
76	5	5-eil76	559
76	10	10-eil76	561
76	15	15-eil76	565
76	5	5-pr76	108590
76	10	10-pr76	109538
76	15	15-pr76	110678
99	10	10-rat99	1238
99	25	25-rat99	1269
99	50	50-rat99	1249
100	25	25-kroA100	21917
100	50	50-kroA100	21453
100	10	10-kroB100	22440
100	50	50-kroB100	22355
101	25	25-eil101	663
101	50	50-eil101	644
105	25	25-lin105	14438
105	50	50-lin105	14379
105	75	75-lin105	14521

Tabela 5.4: Valores da função objetivo para as instâncias do *tipo 1*

Na Tabela 5.4 apresentam-se as instâncias do *tipo 1* com o número de vértices na faixa 51 a 105 e número de grupos variando entre cinco e 75 vértices. Na primeira coluna da Tabela 5.4 é mostrado o número de vértices, na segunda coluna o número de grupos, na terceira as instâncias e na quarta o valor obtido pelo CPLEX. Para todas instâncias o

CPLEX atingiu soluções ótimas.

#nós	$\#V_i$	Instâncias	Valor
51	4	4-eil51-2x2	429
51	9	9-eil51-3x3	445
51	12	12-eil51-3x4	441
51	16	16-eil51-4x4	438
51	20	20-eil51-4x5	441
51	25	25-eil51-5x5	437
52	4	4-berlin52-2x2	8232
52	6	6-berlin52-2x3	8236
52	8	8-berlin52-2x4	8412
52	10	10-berlin52-2x5	9267
70	6	6-st70-2x3	706
70	9	9-st70-3x3	703
70	12	12-st70-3x4	701
70	16	16-st70-4x4	714
70	20	20-st70-4x5	718
76	4	4-eil76-2x2	557
76	9	9-eil76-3x3	560
76	12	12-eil76-3x4	562
76	16	16-eil76-4x4	562
76	20	20-eil76-4x5	562
76	25	25-eil76-5x5	557
76	4	4-pr76-2x2	112071
76	6	6-pr76-2x3	112573
76	9	9-pr76-3x3	109123
76	12	12-pr76-3x4	109164
76	15	15-pr76-3x5	115960
76	18	18-pr76-3x6	117406
99	25	25-rat99-5x5	1377
99	42	42-rat99-6x7	1318
100	28	28-kroA100-4x7	25372
100	25	25-kroB100-5x5	24626
100	30	30-kroB100-5x6	25244
101	9	9-eil101-3x3	659
101	36	36-eil101-6x6	658
105	2	2-lin105-2x1	14627
105	16	16-lin105-4x4	16516

Tabela 5.5: Valores da função objetivo para as instâncias do *tipo 6*

A Tabela 5.5 mostra as soluções ótimas alcançadas pelo CPLEX para as instâncias do *tipo 6*. O número de vértices mostrado na primeira coluna da Tabela 5.5 variaram de 51 a 105 vértices e o número de grupos (segunda coluna) entre quatro e 42 vértices. Na terceira coluna desta tabela são mostradas as instâncias e na quarta coluna o valor obtido pelo CPLEX.

Os testes computacionais, usando a formulação Miller/Chisman e Miller/Chisman/Pen, realizados nas instâncias do *tipo 1*, *5* e *6* alcançaram valores ótimos num total de 87 instâncias. Somente em oito casos as instâncias não atingiram a solução ótima dentro do tempo estabelecido. Toda estas instâncias (Tabelas 5.1, 5.3, 5.4 e 5.5) são de pequeno porte com até 120 vértices. Para os outros tipos de instâncias não foram realizados testes computacionais utilizando estas formulações, pois as instâncias do *tipo 2*, *3* e *4* contém acima de 1000, 300 e 200 vértices, respectivamente. Isto inviabiliza encontrar soluções ótimas para estes últimos tipos de instâncias usando o *software* CPLEX executado no *hardware* descrito acima.

Para as instâncias do *tipo 2*, *3* e *4* foram realizados testes computacionais através do CPLEX utilizando-se o modelo de Miller/Chisman/Pen para se encontrar bons limites inferiores para estas instâncias, com número maior e igual a 200 vértices. Estes testes serão mostrados na seção 5.3, assim como os testes computacionais para obter limites inferiores para as instâncias de porte maior do *tipo 1*, *5* e *6* contendo número de vértices acima de 100.

Em observações realizadas nas instâncias, onde se conhece o valor ótimo, verifica-se que nem sempre os vértices conectando arestas inter-grupos são os vértices mais próximos. Uma outra observação encontrada em soluções ótimas, refere-se que alguns pares de grupos não serão conectados através de suas arestas inter-grupos, porque a melhor sequência de visita dos grupos não passa por tais grupos. Isto será útil nas técnicas heurísticas para encontrar soluções ao PCVG. Pode-se afirmar que ao buscar uma solução para o PCVG, a fixação da sequência de grupos neste caso não será uma técnica promissora. A Figura 5.1 ilustra estes dois pontos observados.

A Figura 5.1 mostra a instância i-30-5-17 com seus vértices (pontos) de cada grupo e as ligações das arestas inter-grupos (linhas cheias) num plano cartesiano. Estas arestas inter-grupos foram resultados da solução ótima alcançada pelo CPLEX. Observa-se que o grupo 3 se conecta ao grupo 5, o grupo 5 ao grupo 2, o grupo 2 ao 1, o grupo 1 ao 4 e finalmente o grupo 4 se conecta ao grupo 3. Observa-se que o grupo 3 não se conecta aos grupos 1 e 2. Mesma analogia pode ser feita para o grupo 5 que não se conecta aos grupos 1 e 4. Entre os grupos 1 e 5 existem dois vértices bem próximos, cuja aresta que poderia conectá-los não faz parte da solução ótima. As coordenadas cartesianas (x, y) deste dois vértices são: para o grupo 1 igual a (380, 526) e para o grupo 5 (413, 532).

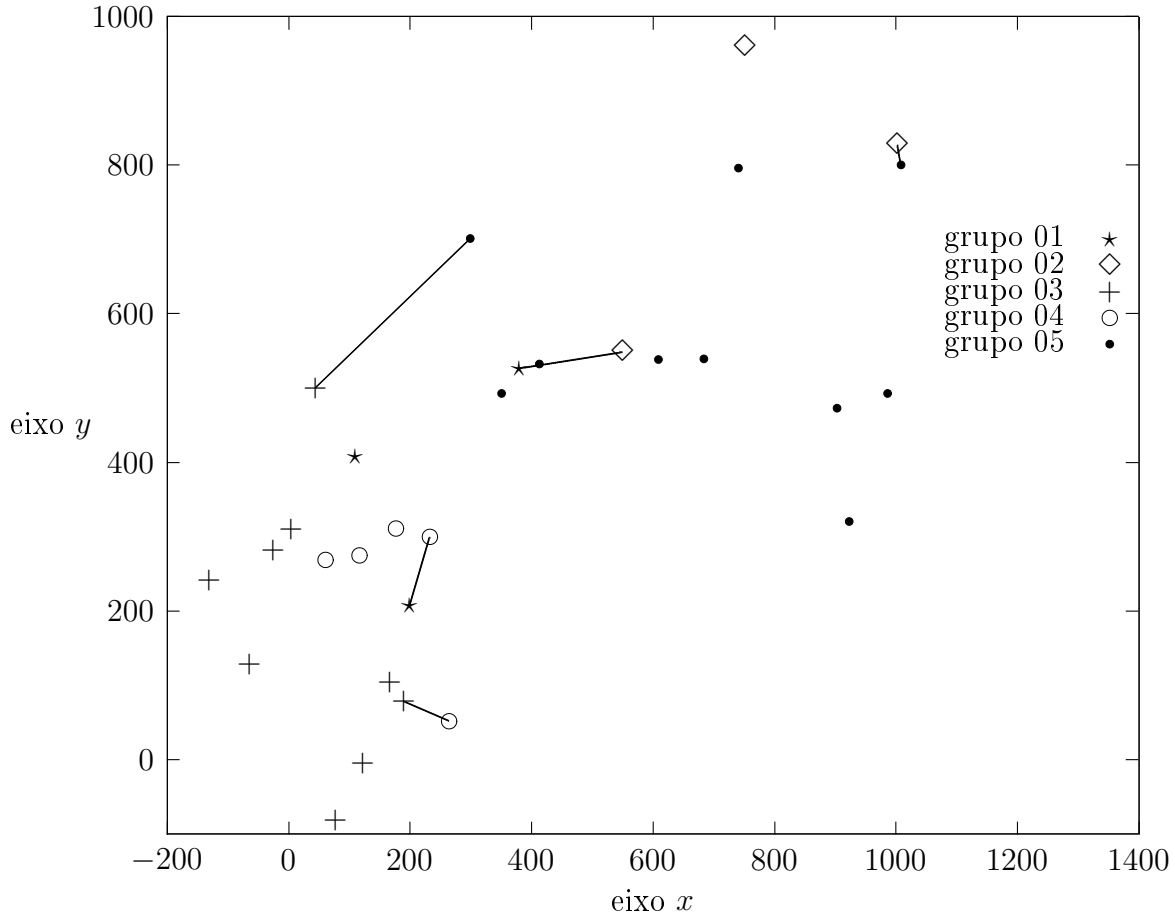


Figura 5.1: Instância i-30-5-17 com trinta vértices e cinco grupos.

5.2 Heurísticas Construtivas

A Tabela 5.6 resume as heurísticas construtivas utilizadas em cada algoritmo. Para verificar a potencialidade destes algoritmos deve-se comparar empiricamente o desempenho de cada um através de experimentos. O método adotado utiliza somente a fase de construção do GRASP e executam-se os algoritmos, obtendo seus melhores valores e valores médios. Em seguida, escolhem-se algumas instâncias para realizar a Distribuição de Probabilidade Acumulativa (DPA) (AIEX; RESENDE; RIBEIRO, 2007). Para realizar a comparação dos algoritmos foram utilizadas instâncias do *tipo 5* variando o número de vértices e de grupos. Este *tipo* de instância foi escolhida ao acaso.

A Tabela 5.7 mostra o *gap* dos valores médios para os diversos algoritmos construtivos. Na primeira coluna apresenta-se a instância e da segunda coluna a nona são apresentados *gaps* dos algoritmos em relação ao melhor valor médio (ZC^*_{med}) alcançado pelos algoritmos construtivos, cujos valores estão na última coluna. Este *gap* é representado por gap_c e

calculado da seguinte forma:

$$gap_c = \frac{vc - mv}{mv}, \tag{5.1}$$

onde, vc é valor médio alcançado pelo algoritmo construtivo e mv o melhor valor médio encontrado entre todos os algoritmos construtivos descritos na Tabela 5.6.

Algoritmo Construtivo	Tipo de Heurística
VMPP	Vizinho mais Próximo com as Arestas Penalizadas
VMPnP	Vizinho mais Próximo com as Arestas não Penalizadas
IMPP	Inserção mais Próxima com as Arestas Penalizadas
IMPnP	Inserção mais Próxima com as Arestas não Penalizadas
IMBP	Inserção mais Barata com as Arestas Penalizadas
IMBnP	Inserção mais Barata com as Arestas não Penalizadas
GENIV	GENI - vértice como elemento do GRASP escolhido aleatório
GENIP	GENI - posição como elemento do GRASP escolhido aleatório

Tabela 5.6: Nomenclatura dos algoritmos construtivos associados às Heurísticas.

A Tabela 5.8 mostra o desempenho em relação aos melhores valores dos construtivos. A descrição das colunas é idêntica a da Tabela 5.7, exceto que na última coluna apresenta-se o melhor valor (ZC^*) alcançado pelos algoritmos construtivos. O gap_c mostrado na Tabela 5.8 é calculado através da equação 5.1, onde vc agora é o melhor valor alcançado pelo algoritmo construtivo e mv o melhor valor encontrado entre todos os algoritmos construtivos.

Valores Médios									
Instância	VMPP	VMPnP	IMPP	IMPnP	IMBP	IMBnP	GENIV	GENIP	ZC^*_{med}
i-30-5-17	0,33	0,05	**0,00**	**0,00**	0,01	**0,00**	0,17	0,32	5200,13
i-45-5-18	0,45	0,09	**0,00**	**0,00**	**0,00**	**0,00**	0,12	0,29	7061,34
i-60-5-21	0,57	0,12	**0,00**	**0,00**	**0,00**	**0,00**	0,09	0,28	7914,53
i-90-5-33	0,93	0,14	**0,00**	**0,00**	**0,00**	0,01	0,11	0,32	6891,91
i-120-5-46	1,06	0,16	**0,00**	**0,00**	**0,00**	0,01	0,09	0,30	7912,29
i-150-5-54	1,39	0,25	**0,00**	0,01	**0,00**	0,02	0,12	0,33	7591,06
i-180-10-63	1,04	0,20	**0,00**	0,01	0,01	0,01	0,09	0,29	10512,60
i-210-10-78	1,24	0,17	**0,00**	**0,00**	**0,00**	0,02	0,10	0,32	10529,49
i-240-10-83	1,26	0,21	**0,00**	0,02	**0,00**	0,03	0,10	0,31	11181,74
i-270-10-92	1,42	0,23	**0,00**	**0,00**	**0,00**	**0,00**	0,08	0,31	10890,44
i-300-10-109	1,61	0,22	**0,00**	**0,00**	**0,00**	0,01	0,08	0,30	10444,81

Tabela 5.7: Os gap_cs dos valores médios obtidos pelos algoritmos construtivos.

Todos os resultados apresentados nas Tabelas 5.7 e 5.8 foram obtidos executando-se 10 vezes cada algoritmo construtivo para cada instância num total de 880 execuções. Pelos valores das Tabelas 5.7 e 5.8 serão desconsiderados os algoritmos construtivos VMPP e

Melhores Valores									
Instância	VMPP	VMPnP	IMPP	IMPnP	IMBP	IMBnP	GENIV	GENIP	ZC^*
i-30-5-17	0,26	0,05	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	5191,67
i-45-5-18	0,45	0,07	0,01	0,01	**0,00**	0,01	0,01	0,01	6966,38
i-60-5-21	0,56	0,09	**0,00**	0,01	0,01	0,01	0,02	0,02	7829,64
i-90-5-33	0,92	0,11	**0,00**	0,01	0,01	0,01	0,04	0,05	6805,61
i-120-5-46	1,04	0,17	**0,00**	0,02	0,01	0,01	0,03	0,05	7744,99
i-150-5-54	1,40	0,26	**0,00**	0,02	0,02	0,03	0,05	0,05	7433,81
i-180-10-63	1,04	0,19	0,01	0,02	**0,00**	0,02	0,05	0,04	10271,77
i-210-10-78	1,23	0,16	**0,00**	**0,00**	**0,00**	**0,00**	0,05	0,05	10417,12
i-240-10-83	1,21	0,18	**0,00**	**0,00**	**0,00**	0,03	0,06	0,05	11049,07
i-270-10-92	1,40	0,22	**0,00**	**0,00**	**0,00**	**0,00**	0,02	0,04	10820,87
i-300-10-109	1,65	0,22	**0,00**	0,02	0,01	0,02	0,07	0,05	10173,74

Tabela 5.8: Gap_c dos melhores valores obtidos pelos algoritmos construtivos.

VMPnP pelo desempenho inferior em relação aos outros construtivos.

Estes métodos construtivos produzem uma solução diferente a cada iteração pois são métodos com componentes aleatórios e a comparação entre estes métodos iterativos deve considerar a qualidade da solução produzida e o esforço computacional exigido (TAILLARD, 2001). Para realizar a comparação utilizam-se análises gráficas das distribuições de probabilidade empírica acumulativa aplicada às diversas instâncias (AIEX; RESENDE; RIBEIRO, 2002), (STÜTZLE; HOOS, 2002), (AIEX; RESENDE; RIBEIRO, 2007) e (HOOS; STÜTZLE, 2007).

Foram consideradas três instâncias com número de vértices diferentes. A escolha destas três instâncias foram:

1. A instância com 30 vértices, `i-30-5-17`, onde a solução ótima é encontrada num tempo de 9,35 segundos usando um método exato (CPLEX).

2. A instância com 150 vértices, `i-150-5-54`, onde a melhor solução conhecida foi alcançada pelo construtivo IMPP em um tempo médio de 96,3 segundos.

3. A instância com 300 vértices, `i-300-10-109`, cuja solução encontrada pelo construtivo IMPP teve um tempo médio de 715,57 segundos.

Para cada instância foram estabelecidos dois alvos considerando os valores produzidos pelos algoritmos construtivos (IMPP, IMPnP, IMBP, IMBnP, GENIV e GENIP). O maior destes valores foi considerado um alvo de acesso fácil (*alvo_fácil*). Um dos menores valores dos construtivos, mas que todos possam teoricamente alcançar, foi considerado alvo difícil

(*alvo_difícil*). A Tabela 5.9 mostra as instâncias, seus alvos e o melhor valor conhecido obtido pelos construtivos.

Instância	melhor valor	alvos	
		alvo_fácil	*alvo_difícil*
i-30-5-17	5191,67	5433,60	5196,16
i-150-5-54	7433,81	9367,57	7814,54
i-300-10-109	10173,74	12408,47	10887,86

Tabela 5.9: Valores dos alvos a serem alcançados pelos algoritmos construtivos.

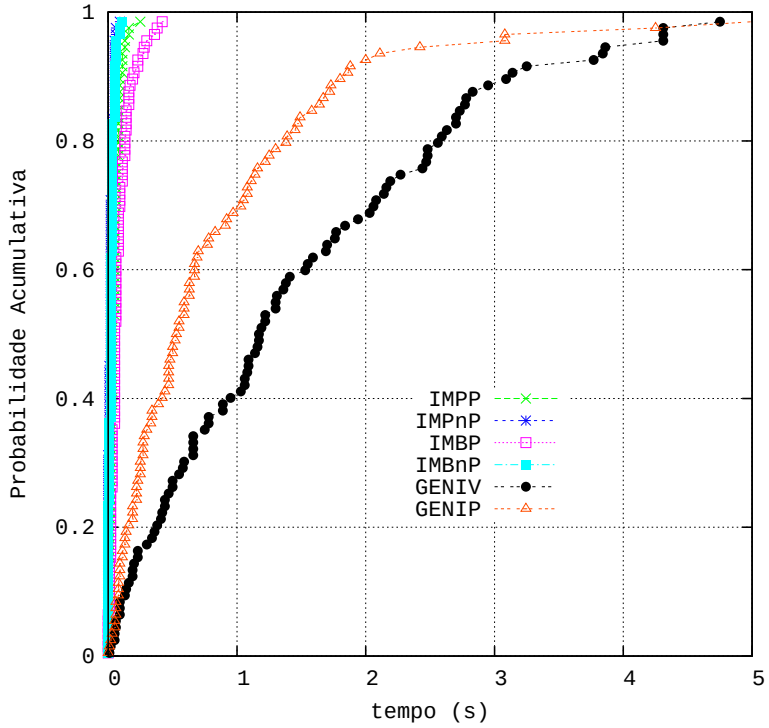


Figura 5.2: DPA dos construtivos para *alvo_fácil*, i-30-5-17.

Para realizar a DPA os algoritmos construtivos foram executados 100 vezes para cada instância da Tabela 5.9 até atingir um determinado alvo, utilizando-se *seed* diferente em cada execução, tornando-as independentes. Como existem três instâncias, dois alvos e seis algoritmos construtivos foram realizados um total de 3600 execuções. Para cada conjunto de execução de uma instância/alvo foi estabelecido um tempo limite. Na instância `i-30-5-17` foi estabelecido um tempo limite de cinco segundos, na instância `i-150-5-54` foi de 10 segundos e na instância `i-300-10-109` de 50 segundos.

No gráfico da Figura 5.2 observa-se o desempenho dos construtivos para a instância i-30-5-17 (*alvo_fácil*). Os construtivos IMPP, IMPnP, IMBP e IMBnP, alcançam o alvo em menos de um segundo. Enquanto, os construtivos GENIV e GENIP alcançam o alvo dentro de cinco segundos.

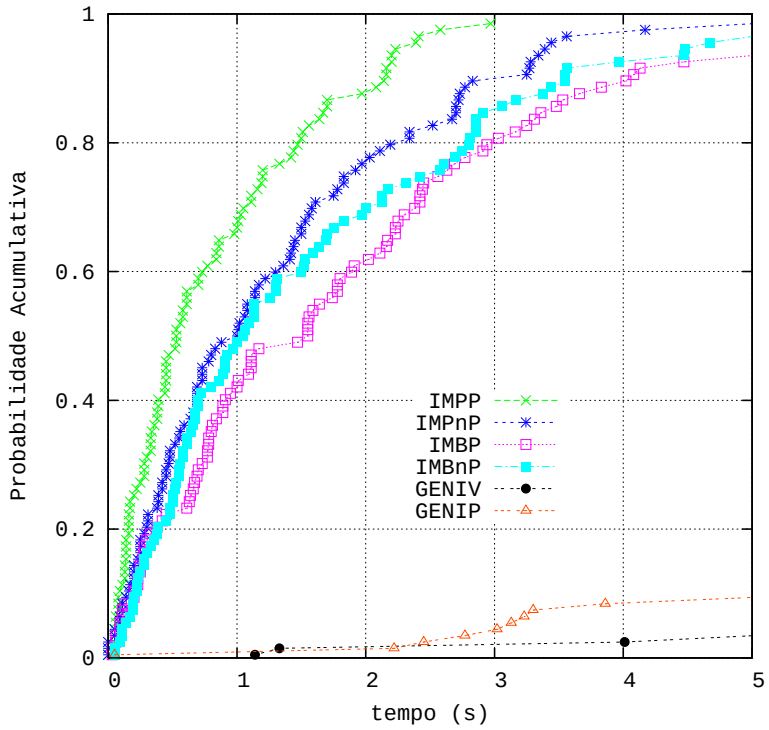


Figura 5.3: DPA dos construtivos para *alvo_difícil*, i-30-5-17.

Para a instância i-30-5-17 (*alvo_difícil*) há uma maior distinção entre os algoritmos construtivos. O gráfico da Figura 5.3 mostra que o construtivo IMPP possui probabilidade de 100% de alcançar o alvo em até três segundos, enquanto que IMPnP, IMBP e IMBnP só conseguem após cinco segundos. GENIV e GENIP necessitam de um tempo maior para alcançar o alvo. GENIP mostra um desempenho ligeiramente melhor do que GENIV.

Para a instância i-150-5-54 (*alvo_fácil*) não há uma diferença significativa entre os construtivos IMPP, IMPnP, IMBP e IMBnP. No gráfico da Figura 5.4, verifica-se que o algoritmo construtivo GENIP alcança o alvo em menos de quatro segundos com desempenho melhor que o GENIV.

No gráfico da Figura 5.5 para a instância i-150-5-54 (*alvo_difícil*) são mostrados o desempenhos dos construtivos e percebe-se uma melhor distinção entre estes. O construtivo IMPP tem 80% de probabilidade de alcançar a solução em oito segundos, sendo que a probabilidade do IMBP alcançar o alvo é menor do que 60%. O IMPnP e IMBnP mostram desempenhos parecidos e bem melhores do que o GENIV e GENIP.

Na instância i-300-10-109 (*alvo_fácil*), os construtivos IMPP, IMPnP, IMBP e IMBnP apresentam distribuições de probabilidades próximas e alcançam a solução alvo em tempo menor do que cinco segundos (Figura 5.6). Percebe-se que probabilidade de alcançar o alvo do GENIP é extremamente maior do que GENIV.

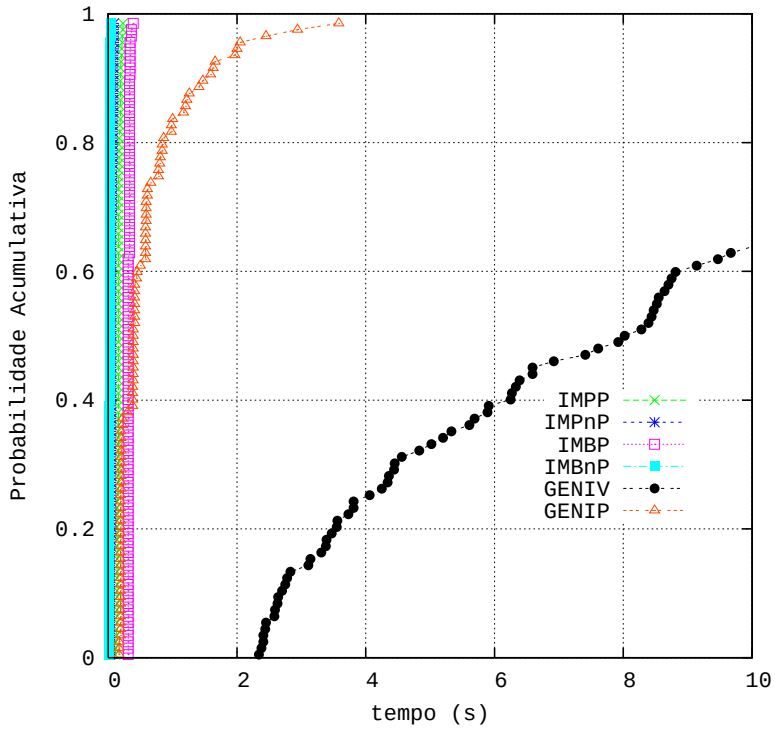


Figura 5.4: DPA dos construtivos para *alvo_fácil*, i-150-5-54.

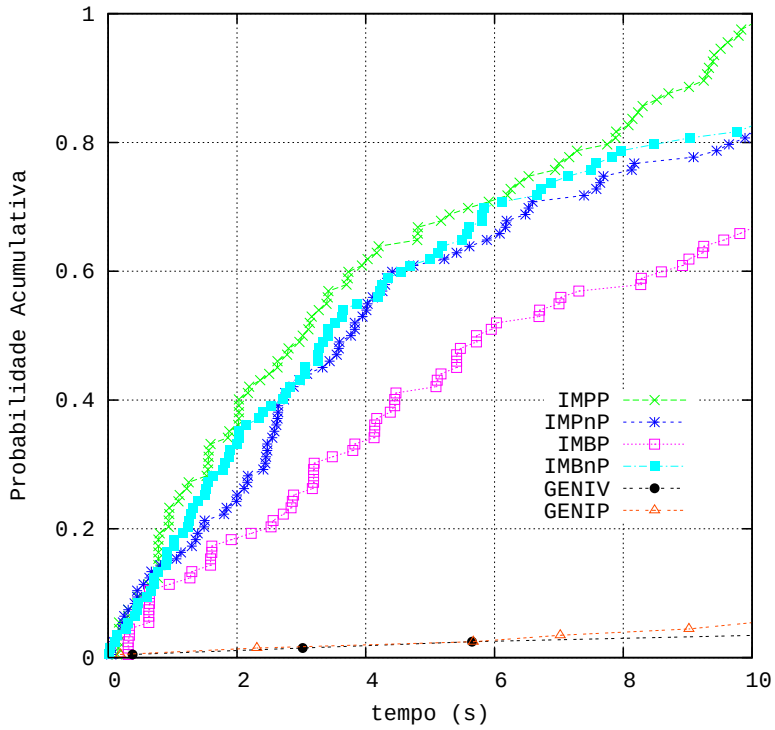


Figura 5.5: DPA dos construtivos para *alvo_difícil*, i-150-5-54.

No gráfico da Figura 5.7 observa-se a probabilidade dos algoritmos construtivos de alcançarem a solução alvo para a instância i-300-10-109 (*alvo_difícil*). Em menos de

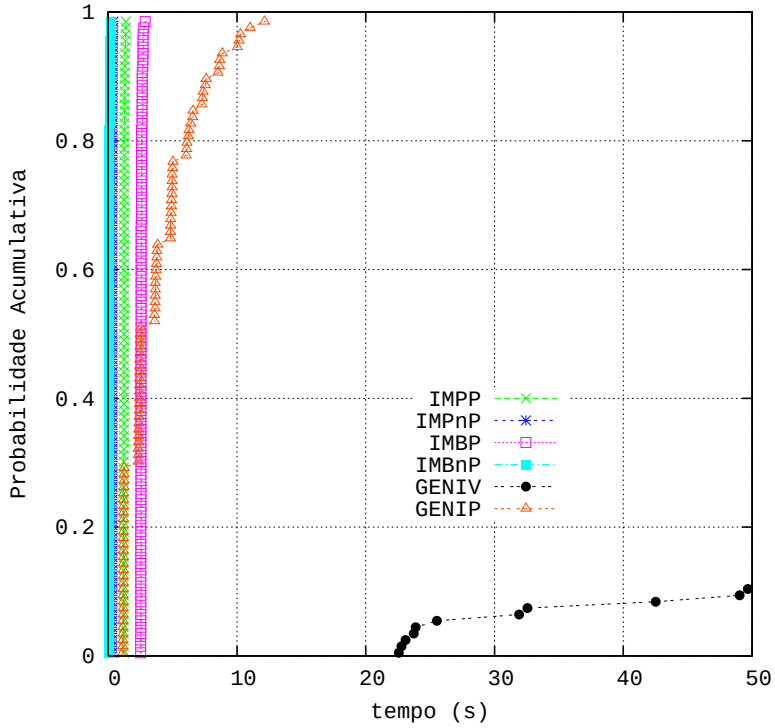


Figura 5.6: DPA dos construtivos para *alvo_fácil*, i-300-10-109.

10 segundos, a probabilidade de alcançar o alvo do IMBnP é 100%. Em 10 segundos, IMPnP tem probabilidade de mais de 90% para alcançar o alvo, o IMPP com mais de 70% e o IMBP com 40%. GENIV e GENIP obtiveram desempenho inferior aos outros construtivos. A ordem de desempenho dos construtivos é mostrada na Tabela 5.10.

Instância	alvo	Ordem de Desempenho
i-30-5-17	*alvo_fácil*	IMPnP, IMBnP, IMPP, IMBP, GENIP, GENIV
	alvo_difícil	IMPP, IMPnP, IMBnP, IMBP, GENIP, GENIV
i-150-5-54	*alvo_fácil*	IMBnP, IMPnP, IMPP, IMBP, GENIP, GENIV
	alvo_difícil	IMPP, IMBnP, IMPnP, IMBP, GENIP, GENIV
i-300-10-109	*alvo_fácil*	IMBnP, IMPnP, IMPP, IMBP, GENIP, GENIV
	alvo_difícil	IMBnP, IMPnP, IMPP, IMBP, GENIP, GENIV

Tabela 5.10: Desempenho realizado pelos algoritmos construtivos.

Os valores de α utilizados para os construtivos usam a estratégia reativa (PRAIS; RIBEIRO, 2000a) e (PRAIS; RIBEIRO, 2000b), onde foram distribuídos em valores discretos, α_i, $i = 0, 1, ..., 10$, com $\alpha_i = i * 0,1$. O número máximo de iterações utilizadas pelos construtivos foi igual a 600 e o valor para atualizar periodicamente as probabilidades p_i, $i = 0, ..., 10$ dos α_i foi igual a 100. Os algoritmos foram executados no Windows XP Service Pack 2, processador Intel Pentium 4 (2,4 GHz) com 1,250 GB de RAM e desenvolvido em linguagem C, compilado em Dev-C++ 4.9.9.2, software sobre Licença de Documentação Livre GNU (LAPLACE et al., 2005).

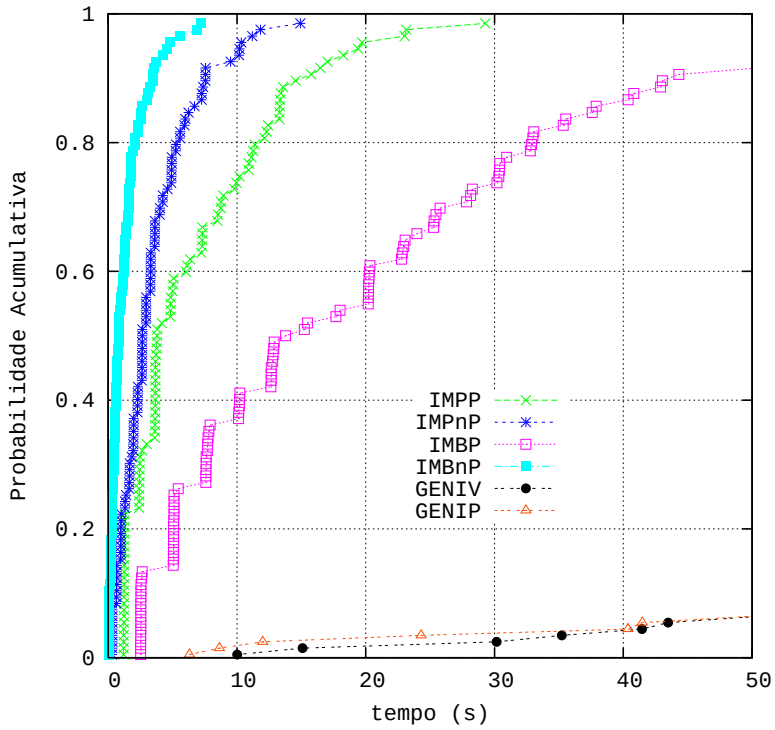


Figura 5.7: DPA dos construtivos para *alvo_ difícil*, i-300-10-109.

Diante dos resultados das Tabelas 5.7, 5.8 e 5.10 observa-se que os algoritmos construtivos de melhor desempenho para as instâncias do *tipo 5* foram na ordem IMPP, IMBP, IMPnP e IMBnP ponderando as três tabelas.

A seguir mostramos testes computacionais sobre um conjunto de instâncias selecionadas de cada *tipo* utilizando os algoritmos construtivos através da heurística de inserção mais próxima (IMP) e do método de descida em vizinhança variável (VND) combinado com a IMP. O VND realiza a fase de busca local aplicada após a construção da solução via heurística IMP.

A Tabela 5.11 mostra os *gaps* alcançados pelas heurísticas IMPP, IMPnP e pelo método VND utilizando a heurística IMPP, denominado de VND-P, além do VND combinado com a IMPnP (VND-nP). Foram escolhidas várias instâncias de diversos *tipos* variando o número de vértices entre 200 e 1000 e o número de grupos entre cinco e 144 vértices. Na primeira coluna da Tabela 5.11 são mostradas as instâncias, na segunda coluna o *tipo*, nas quatro colunas seguintes os *gaps* dos melhores valores alcançados pelas heurísticas IMPP, IMPnP, VND-P e VND-nP, respectivamente. Nas últimas colunas os *gaps* dos valores médios.

Instâncias	**Tipo**	Melhores Valores				Valores Médios			
		IMPP	IMPnP	VND-P	VND-nP	IMPP	IMPnP	VND-P	VND-nP
10-gil262	1	1,14	3,46	0,83	**0,00**	0,87	8,19	**0,00**	5,13
25-gil262	1	1,80	4,18	0,04	**0,00**	0,96	7,61	**0,00**	3,60
50-gil262	1	3,45	5,53	1,18	**0,00**	1,27	6,66	**0,00**	2,87
10-lin318	1	1,33	2,17	**0,00**	1,29	2,64	4,41	**0,00**	1,92
25-lin318	1	6,74	6,32	0,92	**0,00**	3,04	4,74	**0,00**	1,27
50-lin318	1	3,41	5,33	**0,00**	0,62	3,35	3,84	**0,00**	0,17
10-pcb442	1	2,70	3,07	0,47	**0,00**	1,53	2,72	**0,00**	0,12
25-pcb442	1	4,45	3,83	**0,00**	1,17	1,57	3,29	**0,00**	0,88
50-pcb442	1	2,69	3,74	0,09	**0,00**	1,98	3,14	**0,00**	0,89
gap_h méd.(*tipo 1*)		3,08	4,18	0,39	**0,34**	1,91	4,96	**0,00**	1,87
C1k.0	2	2,19	3,27	**0,00**	0,81	0,22	4,61	**0,00**	2,78
C1k.1	2	2,66	4,07	**0,00**	1,46	0,17	6,06	**0,00**	2,52
C1k.2	2	2,27	2,70	0,80	**0,00**	0,64	3,67	**0,00**	1,23
C1k.3	2	1,95	3,02	**0,00**	0,27	0,52	6,72	**0,00**	2,96
C1k.4	2	2,05	3,03	**0,00**	0,25	0,32	5,70	**0,00**	2,54
gap_h méd.(*tipo 2*)		2,23	3,22	**0,16**	0,56	0,37	5,35	**0,00**	2,41
300-6	3	1,48	3,45	**0,00**	0,49	0,87	5,88	**0,00**	3,34
350-6	3	1,25	**0,00**	0,10	0,10	0,35	4,58	**0,00**	3,40
400-6	3	1,05	2,53	**0,00**	0,11	1,07	3,45	**0,00**	1,83
450-6	3	2,37	2,96	0,79	**0,00**	0,90	3,39	**0,00**	1,26
500-6	3	3,44	3,61	**0,00**	0,70	1,11	3,43	**0,00**	1,35
550-20	3	2,41	3,61	1,07	**0,00**	0,67	6,60	**0,00**	3,05
600-20	3	1,34	3,06	**0,00**	0,26	0,62	6,19	**0,00**	3,23
650-20	3	1,64	4,08	0,91	**0,00**	0,82	5,89	**0,00**	2,49
700-20	3	2,43	4,22	**0,00**	0,81	0,78	6,42	**0,00**	3,46
750-25	3	2,93	4,37	1,60	**0,00**	0,76	4,94	**0,00**	2,01
gap_h méd.(*tipo 3*)		2,03	3,19	0,45	**0,25**	0,80	5,08	**0,00**	2,54
200-4-a	4	1,32	1,97	0,13	**0,00**	0,64	2,29	**0,00**	2,02

(*Continua na próxima página.*)

Instâncias	**Tipo**	Melhores Valores				Valores Médios			
		IMPP	IMPnP	VND-P	VND-nP	IMPP	IMPnP	VND-P	VND-nP
200-4-h	4	2,42	1,01	0,90	**0,00**	1,17	2,98	**0,00**	1,59
200-4-z	4	0,79	2,17	**0,00**	0,41	0,95	3,30	**0,00**	1,64
200-4-x1	4	1,88	1,84	0,60	**0,00**	1,10	2,84	**0,00**	1,32
200-4-x2	4	0,93	0,63	**0,00**	0,69	1,21	2,61	**0,00**	1,57
600-8-a	4	1,56	1,78	**0,00**	0,71	0,30	2,19	**0,00**	1,76
600-8-h	4	2,08	2,53	**0,00**	0,96	1,27	1,55	0,23	**0,00**
600-8-z	4	4,49	2,84	0,93	**0,00**	2,21	2,38	0,84	**0,00**
600-8-x1	4	3,14	2,02	0,37	**0,00**	1,71	1,90	0,67	**0,00**
600-8-x2	4	4,94	3,04	0,83	**0,00**	2,00	2,09	0,66	**0,00**
gap_h méd.(*tipo 4*)		2,35	1,98	0,38	**0,28**	1,26	2,41	**0,24**	0,99
300-5-108	5	1,26	2,62	0,92	**0,00**	0,23	5,15	**0,00**	2,64
300-10-109	5	3,28	3,33	1,76	**0,00**	0,35	3,94	**0,00**	2,00
300-15-110	5	2,21	4,25	**0,00**	0,08	0,60	7,43	**0,00**	5,31
300-20-111	5	0,97	3,54	**0,00**	0,01	0,64	7,29	**0,00**	4,49
300-25-112	5	2,62	4,11	0,54	**0,00**	0,61	5,19	**0,00**	3,53
500-5-304	5	0,57	3,39	**0,00**	0,02	0,56	5,14	**0,00**	3,13
500-10-305	5	2,74	2,51	1,60	**0,00**	0,31	5,87	**0,00**	3,22
500-15-306	5	1,99	4,12	**0,00**	0,32	0,55	7,57	**0,00**	4,98
500-20-307	5	1,98	4,77	**0,00**	0,32	0,41	4,90	**0,00**	2,58
500-25-308	5	2,54	2,56	0,92	**0,00**	0,74	5,37	**0,00**	3,14
gap_h méd.(*tipo 5*)		2,02	3,52	0,57	**0,07**	0,50	5,78	**0,00**	3,50
9-eil101-3x3	6	3,15	5,26	**0,00**	1,65	0,67	7,92	**0,00**	5,12
25-eil101-5x5	6	3,22	4,98	**0,00**	0,44	2,03	7,09	**0,00**	1,87
36-eil101-6x6	6	4,93	2,99	1,49	**0,00**	3,23	5,53	0,18	**0,00**
9-a280-3x3	6	3,40	4,97	1,84	**0,00**	1,84	4,77	**0,00**	1,61
25-a280-5x5	6	4,24	4,91	0,06	**0,00**	2,93	4,50	**0,00**	1,45
42-a280-6x7	6	6,03	6,94	**0,00**	3,00	2,55	4,88	**0,00**	1,36
49-rat783-7x7	6	4,36	3,20	**0,00**	1,21	3,55	2,15	0,37	**0,00**
100-rat783-10x10	6	6,38	8,15	**0,00**	0,56	3,10	4,70	**0,00**	1,37
144-rat783-12x12	6	6,87	10,58	**0,00**	1,57	3,18	5,50	**0,00**	2,52

(*Continua na próxima página.*)

Instâncias	**Tipo**	Melhores Valores				Valores Médios			
		IMPP	IMPnP	VND-P	VND-nP	IMPP	IMPnP	VND-P	VND-nP
gap_h méd.(*tipo 6*)		4,73	5,77	**0,38**	0,94	2,56	5,23	**0,06**	1,70
***gap*$_h$ médio total**		*2,74*	*3,63*	*0,41*	***0,38***	*1,28*	*4,74*	***0,06***	*2,16*

Tabela 5.11: Gap_h dos valores alcançados pelas heurísticas.

O percentual de *gap* do *melhor valor* expresso na Tabela 5.11 é cálculado conforme a equação 5.2 descrita abaixo:

$$gap_h = 100 * \left(\frac{vh - mv}{mv} \right), \tag{5.2}$$

onde, vh é o melhor valor encontrado pela heurística mostrada na Tabela 5.11 e mv é o melhor valor alcançado entre as heurísticas IMPP, IMPnP, VND-P e VND-nP. Para o percentual de *gap* do *valor médio*, vh é o valor médio encontrado pela heurística e mv é o melhor valor médio alcançado entre as heurísticas IMPP, IMPnP, VND-P e VND-nP.

Para cada execução das heurísticas foram utilizadas 200 iterações com o parâmetro α na versão reativa que gerencia o tamanho da LRC na etapa de construção. Os valores de α variam no intervalo de [0,1]. O valor para atualizar periodicamente as probabilidades dos αs foi igual a 50. Os algoritmos das heurísticas IMPP, IMPnP, VND-P e VND-nP foram codificados em linguagem de programação C e executados sobre o sistema operacional Linux Ubuntu versão 4.3.2-1. O computador utilizado foi Intel Core 2 Quad, 2.83 GHz com 8 GB de RAM, com quatro núcleos, mas somente utilizou-se um núcleo. As heurísticas IMPP, IMPnP, VND-P e VND-nP da Tabela 5.11 foram executadas dez vezes e seus tempos médios são mostrados na Tabela 5.12. Na primeira coluna da Tabela 5.12 são mostradas as instâncias, na segunda coluna o *tipo* e nas quatro últimas colunas os tempos médios medidos em segundos alcançados pelas heurísticas IMPP, IMPnP, VND-P e VND-nP, respectivamente.

As heurísticas VND-P e VND-nP alcançaram melhores resultados do que as heurísticas IMPP e IMPnP. O gap_h médio total de IMPP foi igual a 2,74, de IMPnP com o valor de 3,63, de VND-P com 0,41 e VND-nP com ***0,38*** (Tabela 5.11). Observe que o gap_h médio total de cada heurística (VND-P e VND-nP) apresenta pouca diferença. Ao fazer uma análise em particular para cada tipo de instância sobre as heurísticas que realizam busca local (VND-P e VND-nP), verifica-se que: para o *tipo 1*, o gap_h médio de VND-P ficou em 0,39 e VND-nP em ***0,34***, para o *tipo 2*, o gap_h médio de VND-P obteve o valor de ***0,16*** e VND-nP com 0,56 e para o *tipo 3*, o gap_h médio de VND-P ficou em 0,45 e VND-nP em

0,25. Para as instâncias do *tipo 4*, *5* e *6*, os gap_hs médio de VND-P são 0,38, 0,57 e ***0,38***, respectivamente. Para o VND-nP os gap_hs médio das instâncias do *tipo 4*, *5* e *6* ficaram em ***0,28***, ***0,07*** e 0,94, respectivamente. Verifica-se que para as instâncias do *tipo 1*, *3*, *4* e *5*, a heurística VND-nP alcançou melhores resultados do que a VND-P, enquanto que para as instâncias do *tipo 2* e *6*, VND-P se sobressaiu em relação ao VND-nP.

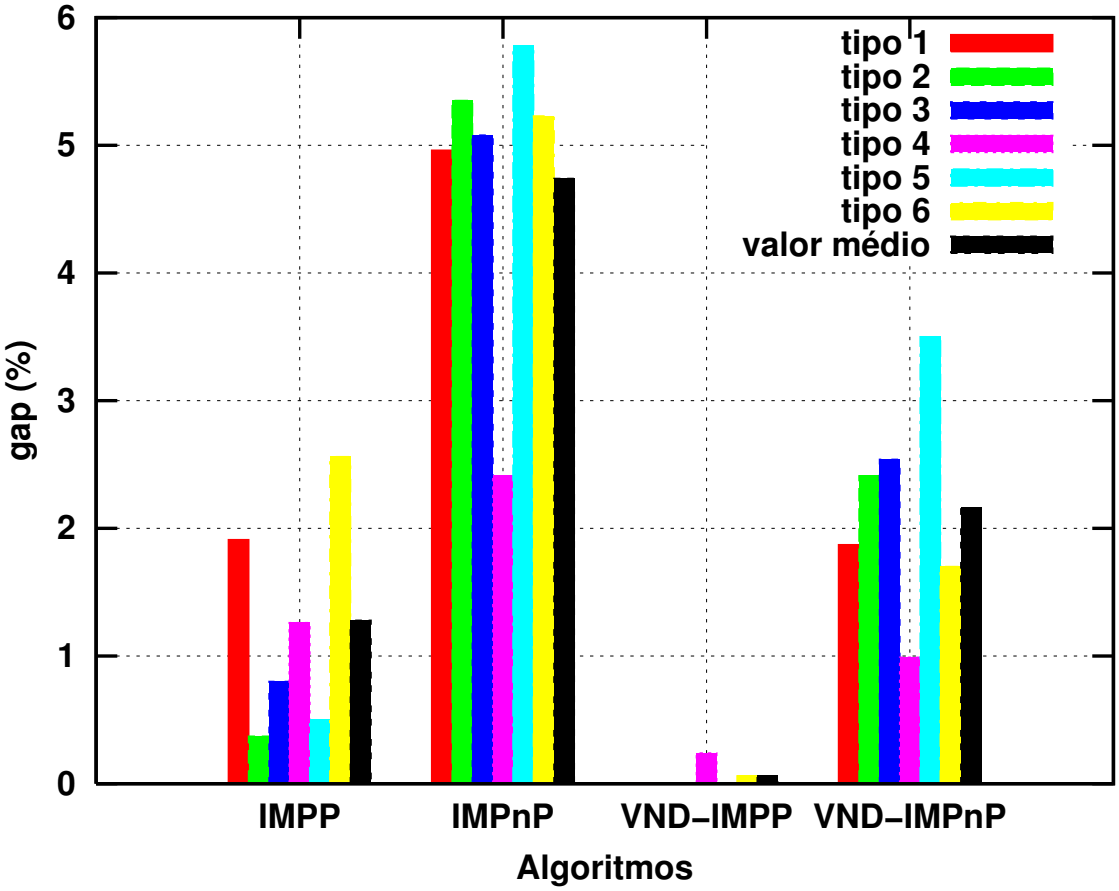


Figura 5.8: Gap_h médio para cada *tipo* de instância alcançados pelos algoritmos: IMPP, IMPnP, VND-IMPP e VND-IMPnP, sobre os valores médios.

O desempenho dos algoritmos construtivos (IMPP e IMPnP) e com buscas locais (VND-IMPP e VND-IMPnP) para os valores médios são mostrados na Figura 5.8 através do gap_h médio (eixo y) para cada *tipo* de instância. Observa-se que os algoritmos (eixo x) que utilizaram penalização mostram-se com melhores resultados na média do que os algoritmos que não aplicaram a estratégia de penalização. Os dados apresentados na Figura 5.8 foram extraídos da Tabela 5.11. Por exemplo, para cada algoritmo e para as instâncias do *tipo 1* (primeiros dados em barras no gráfico na cor **vermelha**), o gap_h médio dos valores médios do algoritmo IMPP foi igual a 1,91%, de IMPnP com o valor de 4,96%, de VND-IMPP com ***0,00%*** e VND-IMPnP com 1,87%. Observe que o gap_h médio do algoritmo VND-IMPP foram todos iguais a ***zero***, exceto para as instâncias do

tipo 4 e do *tipo 6*, cujos valores foram iguais a 0,24% e 0,06%, respectivamente. Na Figura 5.8 também são mostrados em barras, na cor **preta**, os valores médios alcançados pelos algoritmos para todos os tipos de instâncias representando o gap_h médio individual de cada algoritmo (*dados extraídos da última linha da Tabela 5.11*).

Instâncias	**Tipo**	Tempos Médios (segundos)			
		IMPP	IMPnP	VND-P	VND-nP
10-gil262	1	10,6	10,0	18,2	25,9
25-gil262	1	10,6	11,1	19,8	30,3
50-gil262	1	10,6	13,5	22,1	36,6
10-lin318	1	23,8	14,2	45,6	45,3
25-lin318	1	16,9	16,1	52,5	57,1
50-lin318	1	16,7	19,6	60,3	66,9
10-pcb442	1	45,2	39,6	125,9	132,2
25-pcb442	1	45,2	43,0	128,1	172,9
50-pcb442	1	45,2	49,8	135,0	191,3
C1k.0	2	1097	1029	3335	5199
C1k.1	2	1107	1030	3273	8947
C1k.2	2	1126	1068	3146	7059
C1k.3	2	1134	1037	2670	7518
C1k.4	2	1096	1030	3202	6945
300-6	3	14,4	13,7	25,0	37,7
350-6	3	22,9	61,3	43,9	86,0
400-6	3	31,4	30,9	67,3	83,1
450-6	3	47,1	41,6	91,7	116,3
500-6	3	62,1	54,2	142,2	243,5
550-20	3	89,7	74,3	182,5	473,1
600-20	3	119,5	91,1	236,5	443,2
650-20	3	148,2	119,0	322,6	624,2
700-20	3	184,2	149,7	400,3	774,7
750-25	3	234,1	195,3	493,5	1002,8
200-4-a	4	5,0	3,9	8,7	7,4
200-4-h	4	4,5	4,1	7,8	8,6
200-4-z	4	4,6	4,1	8,6	8,3

(*Continua na próxima página.*)

Instâncias	**Tipo**	Tempos Médios (segundos)			
		IMPP	IMPnP	VND-P	VND-nP
200-4-x1	4	4,7	4,0	8,1	8,2
200-4-x2	4	4,8	3,9	8,6	8,5
600-8-a	4	121,9	87,3	247,8	211,4
600-8-h	4	155,0	87,2	300,0	346,8
600-8-z	4	111,2	87,2	310,1	327,7
600-8-x1	4	111,2	90,8	307,9	281,0
600-8-x2	4	111,3	86,7	316,4	314,6
300-5-108	5	15,8	15,4	26,8	42,6
300-10-109	5	16,3	15,0	28,6	37,3
300-15-110	5	16,0	15,2	32,8	46,1
300-20-111	5	16,3	15,8	31,3	46,4
300-25-112	5	15,9	16,3	30,8	43,6
500-5-304	5	70,5	72,8	162,6	249,9
500-10-305	5	72,7	67,1	142,8	283,9
500-15-306	5	73,5	67,7	150,0	249,7
500-20-307	5	70,3	70,9	152,7	259,9
500-25-308	5	70,1	71,9	155,4	234,9
9-eil101-3x3	6	0,7	0,7	1,1	1,2
25-eil101-5x5	6	0,8	1,1	1,4	2,1
36-eil101-6x6	6	0,7	0,9	1,8	2,7
9-a280-3x3	6	12,2	9,6	23,5	27,2
25-a280-5x5	6	12,2	11,7	30,2	30,7
42-a280-6x7	6	12,2	12,8	31,7	36,9
49-rat783-7x7	6	220,9	369,3	1328,2	1361,6
100-rat783-10x10	6	311,7	436,6	2130,2	2490,2
144-rat783-12x12	6	237,3	474,0	1935,9	2934,8
valor médio total (s)		162,62	***159,36***	**493,60**	947,46

Tabela 5.12: Tempos médios das heurísticas.

O tempo médio total de IMPP igual a 162,62s e de IMPnP com o valor de ***159,36s*** mostrados na Tabela 5.12 apresentam pouca diferença entre estes. Isto mostra que em termos de esforço computacional para construir soluções, ambas heurísticas são promissoras.

Numa análise mais profunda verificamos que nas instâncias do *tipo 2*, *3* e *4*, a heurística de construção IMPnP demanda menor esforço computacional quando comparada a heurística IMPP. Já para instância do *tipo 6*, a heurística de construção IMPP gasta menos tempo computacional do que IMPnP. Para as instâncias do *tipo 1* e *5*, ambas heurísticas IMPP e IMPnP mostram tempos computacionais médios bem próximos.

O tempo médio total de **493,60s** (Tabela 5.12) depreendido pelo método VND-P é menor do que VND-nP, cujo valor foi de 947,46s, devido ao fato que a heurística VND-P realiza menos trabalho na busca local sobre a solução produzida pelo construtivo IMPP. Isto acontece porque a solução gerada pela heurística IMPP é de melhor qualidade do que a solução gerada pela heurística IMPnP. O tempo computacional médio do VND-nP foi menor do que o do VND-P somente em sete instâncias do total de 53.

5.3 Heurísticas GRASP

Para analisar o desempenho empírico das heurísticas GRASP propostas (MESTRIA; OCHI; MARTINS, 2009a) e (MESTRIA; OCHI; MARTINS, 2009b), os resultados de G1, G2, G3, G4, G5 e G6' foram comparados, quando possível, com a solução ótima, ou limites inferiores obtidos pelo método exato usando a formulação matemática da literatura proposta em Chisman (1975) e o software CPLEX 11.2. A Tabela 5.13 resume os tipos de GRASP propostos para o PCVG com as técnicas utilizadas para reconexão de caminhos e busca local. As versões GRASP utilizaram a heurística IMPP na fase de construção da solução.

Nomenclatura	Tipo de GRASP
G1	GRASP tradicional sem Reconexão de Caminhos e 2-Optimal
G2	GRASP com Reconexão de Caminhos 1 (RC1) e 2-Optimal
G3	GRASP com Reconexão de Caminhos 2 (RC2) e 2-Optimal
G4	GRASP com RC1 e RC2 e 2-Optimal
G5	GRASP com Reconexão de Caminhos 3 (RC3) e VND
G6'	GRASP com RC4 e VND

Tabela 5.13: Versões do GRASP aplicado ao PCVG.

Os principais parâmetros utilizados nas heurísticas são descritos a seguir. O valor para penalizar os custos das arestas (L) foi de 10*max$\{c_{ij}\}$. A expressão max$\{c_{ij}\}$ significa o maior custo entre todos os custos das arestas (v_i,v_j). O valor de 10*max$\{c_{ij}\}$ adotado para o parâmetro L não ocasionou nenhum problema para os algoritmos gerar as soluções. Todas as soluções geradas neste trabalho foram soluções viáveis. O parâmetro α para gerenciar a LRC na etapa de construção do GRASP utiliza a versão reativa, onde os valores

de α variam no intervalo de [0,1]. O número de iterações do GRASP foi fixado em 200. A cardinalidade do conjunto de soluções elites, *num_ eli*, foi igual a 10 (estabelecido após experimentos preliminares). Devido à natureza aleatória do GRASP, foram executados dez (10) vezes cada heurística aqui proposta para cada instância.

Instâncias	Id.	#nós	$\#V_i$	Tipo	CPLEX		
					Valor	Lim. Inf.	(%)
10-lin318	I_1	318	10	1	531931	526559,70	1,01
10-pcb442	I_2	442	10	1	546157	536478,37	1,77
25-eil101	I_3	101	25	6	**23671**	23668,63	0,01
144-rat783	I_4	783	144	6	916103	913715,52	0,26
300-20-111	I_5	300	20	5	310590	308627,83	0,63
500-25-308	I_6	500	25	5	367509	364114,62	0,92
300-6	I_7	300	6	3	**8956**	8916,41	0,44
700-20	I_8	700	20	3	41615	41274,00	0,82
C1k.0	I_9	1000	10	2	133638625	131360206,30	1,70
C1k.1	I_{10}	1000	10	2	130563491	128540131,50	1,55
200-4-h	I_{11}	200	4	4	**62835**	62391,08	0,71
600-8-z	I_{12}	600	8	4	132767	128083,87	3,53
						gap_{li} médio =	1,11

Tabela 5.14: Instâncias com seus identificadores: número de nós, número de grupos, tipo e valores encontrados pelo CPLEX

O CPLEX foi executado num computador com 4 núcleos Intel Core 2 Quad, 2.83 GHz com 8 GB de RAM. O sistema operacional utilizado foi Linux Ubuntu versão 4.3.2-1. As heurísticas GRASP foram codificadas em linguagem de programação C e executadas no mesmo computador descrito anteriormente, mas utilizando somente um núcleo.

A Tabela 5.14 mostra as instâncias (coluna 1) com os seus identificadores (coluna 2), seguido do número de vértices (nós), número de grupos e tipo. Na Tabela 5.14 são mostradas as soluções, os limites inferiores gerados pelo software CPLEX e na última coluna mostra-se o percentual do gap_{li} (%) entre valor da solução e limite inferior calculado da seguinte forma:

$$gap_{li} = 100 * \left(\frac{best - li}{best + \epsilon} \right), \tag{5.3}$$

onde, *best* é o melhor valor encontrado pelo CPLEX, *li* o limite inferior dado pelo CPLEX e ϵ igual 10^{-10}.

Devido às longas iterações ocorridas no CPLEX para a maioria das instâncias, um tempo limite de 25200s foi determinado para sua execução. Para a instância I_3, a solução ótima foi encontrada pelo CPLEX em 442s. Todas as outras execuções utilizaram 25200s, sendo, portanto abortadas antes de se encontrar ou confirmar uma solução ótima.

Instâncias	Id.	#nós	$\#V_i$	Tipo	Melhores Valores	
					Custo	Método
10-lin318	I_1	318	10	1	43918	G5
10-pcb442	I_2	442	10	1	55850	G5
25-eil101	I_3	101	25	6	671	CPLEX
300-20-111	I_5	300	20	5	13820	G2
200-4-h	I_{11}	200	4	4	10795	CPLEX
600-8-z	I_{12}	600	8	4	19872	G4

Tabela 5.15: Valores alcançados pelas heurísticas e pelo CPLEX para um subconjunto de instâncias.

Antes de mostrar os resultados das heurísticas G1, G2, G3, G4, G5 e G6', apresenta-se a metodologia utilizada para determinar o percentual fixo para ativação da reconexão de caminhos RC4 na heurística G6'. Para isto foram realizados testes computacionais sobre um conjunto de instâncias. Os percentuais analisados foram de 10% a 90% do total de soluções do conjunto elite. Isto corresponde respectivamente, a ativar a RC4 desde a introdução de uma solução até nove soluções novas. Nos resultados apresentados a seguir, representa-se por p o número de soluções novas introduzidas no conjunto elite. Exemplo: p=1 significa que quando uma solução nova aparecer no conjunto elite a RC4 será ativada. A justificativa para elaborar esta versão vem dos resultados da RC3 incorporada ao G5 que foi competitivo em relação aos GRASP (G1, G2, G3 e G4). Cabe ressaltar que a RC3 é ativada quando 10 soluções novas são inseridas no conjunto elite. Mais precisamente quando o conjunto elite torna-se totalmente novo.

A seguir mostram-se os resultados que conduziram a escolha da heurística G6' (representação da heurística G6 com p=1) para comparação com as heurísticas G1, G2, G3, G4 e G5. Para verificar o comportamento do G6 e sem perda de generalidade foi escolhido um subconjunto de instâncias extraídas da Tabela 5.14 e mostrados na Tabela 5.15. A Tabela 5.15 mostra na primeira coluna as instâncias, na segunda coluna os identificadores, na terceira o número de nós, na quarta e quinta colunas o total de grupos e tipo, respectivamente. Nas duas últimas colunas são apresentados os custos e os métodos (G2, G4, G5 e CPLEX) que alcançaram os melhores valores.

Na Tabela 5.16 são mostrados os percentuais dos *gaps* dos melhores valores alcançados pela heurística G6 quando comparados aos melhores valores conhecidos e expressos pela Tabela 5.15. O cálculo do *gap* usa a equação 5.2 onde, vh é o melhor valor alcançado pela heurística G6 e mv o melhor valor encontrado pelo método mostrado na Tabela 5.15.

O valor de p na Tabela 5.16 mostra o número de soluções novas no conjunto elite considerada para ativação da reconexão de caminhos (RC4). Para cada valor de p a

heurística G6 foi executada 10 vezes. Foram realizados um total de 540 execuções. O melhor gap_h médio ocorreu com ***p=1, cujo valor foi de 1,10%***. O segundo melhor valor foi 1,24%, p=6. Para a instância $\boldsymbol{I_1}$ **com** $\boldsymbol{p}$ **igual a um**, a heurística G6 foi melhor que G5 e para a instância $\boldsymbol{I_5}$ **com** $\boldsymbol{p}$ **igual a quatro e cinco**, G6 foi melhor que G2. Cabe ressaltar que a heurística G5 utilizou a RC3 com o valor de p igual a 10. Em todas as outras instâncias e valores de p, exceto os descritos anteriormente, G6 não obteve os melhores resultados.

Id.	p=1	p=2	p=3	p=4	p=5	p=6	p=7	p=8	p=9
I_1	**-0,48**	0,90	0,61	0,78	0,82	1,24	0,48	0,93	0,68
I_2	1,28	1,04	1,85	1,49	2,04	1,55	1,62	0,68	1,62
I_3	0,60	1,19	1,19	1,94	0,89	1,19	1,49	0,89	1,94
I_5	0,54	0,25	1,17	**-0,12**	**-0,34**	0,65	0,51	1,22	0,45
I_{11}	3,07	3,64	3,08	2,65	4,36	2,16	3,70	4,07	3,97
I_{12}	1,61	1,63	1,46	2,02	2,00	0,62	2,02	1,49	2,45
gap_h médio=	**1,10**	1,44	1,56	1,46	1,63	**1,24**	1,64	1,55	1,85

Tabela 5.16: Gap_h dos melhores valores de G6 em relação as demais heurísticas e ao CPLEX.

Id.	p=1	p=2	p=3	p=4	p=5	p=6	p=7	p=8	p=9
I_1	**-0,48**	0,90	0,61	0,78	0,82	1,24	0,48	0,93	0,68
I_2	1,28	1,04	1,85	1,49	2,04	1,55	1,62	0,68	1,62
I_3	0,00	0,59	0,59	1,33	0,30	0,59	0,89	0,30	1,33
I_5	0,27	**-0,02**	0,90	**-0,38**	**-0,61**	0,38	0,24	0,95	0,18
I_{11}	0,48	1,04	0,50	0,07	1,74	**-0,41**	1,09	1,45	1,36
I_{12}	0,57	0,59	0,42	0,98	0,96	**-0,40**	0,98	0,46	1,40
gap_h médio=	**0,35**	0,69	0,81	0,71	0,88	**0,49**	0,88	0,79	1,10

Tabela 5.17: Gap_h dos melhores valores de G6 em relação a G5.

Uma comparação mais próxima entre G5 e G6 com o mesmo subconjunto de instâncias da Tabela 5.15 foi realizada. Os melhores valores alcançados por G5 são mostrados na Tabela 5.18. Na Tabela 5.17 são mostrados os *gaps* dos melhores valores alcançados pela heurística G6 em relação aos valores de G5 (Tabela 5.18). O percentual do *gap* é idêntico ao gap_h da equação 5.2, exceto que mv é o melhor valor encontrado por G5 mostrado na Tabela 5.18. O melhor gap_h médio ocorreu com ***p=1, cujo valor foi de 0,35%***. O segundo melhor valor alcançado foi de 0,49% com p=6. Para as instâncias $\boldsymbol{I_1}$ **com** $\boldsymbol{p}$**=1,** $\boldsymbol{I_5}$ **(**$\boldsymbol{p}$**=2, 4 e 5),** $\boldsymbol{I_{11}}$ **(**$\boldsymbol{p}$**=6) e** $\boldsymbol{I_{12}}$ **(**$\boldsymbol{p}$**=6)**, G6 foi melhor que G5.

Apesar de ter sido escolhido somente um subconjunto de instâncias para as análises, os resultados mostrados nas Tabelas 5.16 e 5.17 já mostram uma tendência no comportamento da escolha do percentual de soluções elites para ativação da RC4. Os gap_hs de G6

variam com o valor de p. O valor de p escolhido para heurística G6' foi igual a um.

Instâncias	G5
10-lin318	43918
10-pcb442	55850
25-eil101	675
300-20-111	13857
200-4-h	11073
600-8-z	20077

Tabela 5.18: Melhores valores de G5 para um subconjunto de instâncias.

Id.	CPLEX	Melhores Valores					
		G1	G2	G3	G4	G5	G6'
I_1	1,01	1,14	0,80	1,02	0,79	0,74	**0,71**
I_2	1,77	1,22	0,71	1,10	0,70	**0,66**	0,78
I_3	**0,01**	0,09	0,05	0,06	0,06	0,03	0,03
I_4	0,26	0,21	0,14	0,21	0,14	**0,13**	0,14
I_5	0,63	0,58	**0,45**	0,54	0,48	0,47	0,48
I_6	0,92	0,71	0,57	0,70	0,56	**0,55**	0,60
I_7	**0,44**	0,75	0,53	0,62	0,52	0,49	0,47
I_8	0,82	0,67	0,59	0,64	0,59	**0,41**	0,62
I_9	1,70	1,61	1,34	1,63	**1,42**	1,43	1,45
I_{10}	1,55	1,31	1,05	1,29	**1,03**	1,07	1,09
I_{11}	**0,71**	1,74	1,09	1,59	1,05	1,15	1,24
I_{12}	3,53	2,07	1,54	2,00	**1,38**	1,54	1,65
gap_{li} médio	1,11	1,01	0,74	0,95	**0,72**	**0,72**	0,77

Tabela 5.19: Os melhores valores obtidos pelas heurísticas GRASP com 200 iterações.

A Tabela 5.19 mostra os resultados referentes às melhores soluções obtidas pelas heurísticas GRASP comparadas ao CPLEX. Na primeira coluna apresentam-se os identificadores das instâncias descritos na Tabela 5.14, na segunda coluna os gap_{li} encontrados pelo CPLEX e nas seis últimas colunas são mostrados os *gaps* das heurísticas GRASP (gap_{li}), calculado pela equação 5.3, exceto que *best* é o melhor valor encontrado pela heurística GRASP. Compara-se a melhor solução de cada método com os limites inferiores (Tabela 5.14). Neste contexto o gap_{li} médio do CPLEX ficou em 1,11% (isso porque em muitas instâncias o CPLEX não conseguiu convergir para um ótimo no tempo limite estabelecido), G1 em 1,01%, G2 em 0,74%, G3 em 0,95%, ***G4 em 0,72%***, ***G5 em 0,72%*** e G6' em 0,77%. O CPLEX obteve a melhor solução (solução ótima ou não) em três num total de 12 instâncias, G2 e G6' obtiveram uma solução cada, G4 obteve três melhores soluções e G5 quatro melhores soluções.

A Tabela 5.20 mostra as soluções médias obtidas pelas heurísticas GRASP (10 execu-

Id.	Valores Médios					
	G1	G2	G3	G4	G5	G6'
I_1	1,75	1,26	1,37	1,13	**1,05**	1,18
I_2	1,94	1,32	1,49	1,19	**1,10**	1,35
I_3	0,35	0,27	0,21	0,18	**0,04**	0,21
I_4	0,28	0,20	0,25	0,19	**0,17**	0,19
I_5	0,84	0,66	0,71	**0,62**	0,63	0,69
I_6	0,92	0,73	0,81	**0,70**	0,74	0,79
I_7	1,08	0,89	0,89	**0,79**	0,81	0,86
I_8	0,83	0,71	0,75	0,68	**0,51**	0,73
I_9	1,88	1,60	1,74	**1,59**	**1,59**	1,73
I_{10}	1,54	1,26	1,41	1,24	**1,23**	1,37
I_{11}	3,07	2,42	2,33	**2,03**	2,12	2,35
I_{12}	3,16	2,24	2,61	**2,15**	2,24	2,49
gap_{li} médio	1,47	1,13	1,21	1,04	**1,02**	1,16

Tabela 5.20: Os valores médios obtidos pelas heurísticas GRASP com 200 iterações.

ções de cada instância) comparando-as com os limites inferiores (Tabela 5.14). A descrição da Tabela 5.20 é semelhante a da Tabela 5.19. O gap_{li} de G1 ficou em 1,47%, de G2 em 1,13%, de G3 em 1,21%, de G4 em 1,04%, de ***G5 igual a 1,02%*** e G6' em 1,16%.

Na Tabela 5.21 são mostrados os tempos médios medidos em segundos das heurísticas GRASP. Observa-se que G1 e G3 são aquelas que consomem o menor tempo médio, fato que era esperado, pois estas não utilizam a RC1 ao final do GRASP. Apesar de G2, G4, G5 e G6' apresentarem tempos médios maiores que G1 e G3, estes tempos ainda são bem menores que os tempos exigidos pelo software CPLEX, exceto para as instâncias I_9 e I_{10} quando utiliza-se a heurística G5.

Id.	Tempos Médios (s)					
	G1	G2	G3	G4	G5	G6'
I_1	57,05	158,87	87,48	184,75	496,4	238,4
I_2	151,01	451,11	227,67	495,25	1385,7	623,4
I_3	2,48	3,72	3,95	6,43	8,4	6,1
I_4	750,30	3718,32	1133,96	4132,28	17507,0	6370,1
I_5	51,68	121,30	75,65	142,81	308,7	146,2
I_6	225,93	720,47	331,15	822,49	2303,8	1076,9
I_7	49,44	99,47	75,97	119,72	283,5	193,3
I_8	593,80	1618,11	881,57	1837,78	4605,0	2857,4
I_9	1762,68	7592,92	2608,65	8749,56	39790,0	7720,1
I_{10}	1765,84	9421,46	2616,59	10297,08	31141,0	7938,4
I_{11}	15,75	33,95	25,07	42,37	76,9	50,4
I_{12}	364,78	1533,07	554,9	1652,09	4765,0	1568,7

Tabela 5.21: Os tempos médios das heurísticas GRASP (em segundos).

De acordo com os resultados apresentados até o momento, onde o critério de parada é o número de iterações e utilizando o mesmo número para todas as heurísticas, pode-se observar que G4 e G5 obtiveram os melhores resultados, mas exigiram tempos computacionais maiores que G1 e G3. A versão G6' na maioria das instâncias gastou um tempo maior que G4. Realizaram-se novos testes computacionais colocando como critério de parada um tempo máximo idêntico para as seis heurísticas. Foi utilizado um tempo limite de 2 horas para o CPLEX e 720 segundos para as heurísticas GRASP, porque, para cada instância, as heurísticas são executadas 10 vezes e a melhor solução encontrada nestas execuções é retornada. Somente a execução da instância I_3 não foi limitada por duas horas, devido à solução ótima ser encontrada antes.

Os novos valores obtidos pelo CPLEX com este novo critério de parada são mostrados na Tabela 5.22. Na primeira coluna apresentam-se os identificadores das instâncias descritos pela Tabela 5.14, na segunda coluna o valor alcançado pelo CPLEX, na terceira coluna o limite inferior e na última coluna o gap_{li}.

CPLEX - Nova Execução			
Id.	Valor	Lim. Inf.	(%)
I_1	534640	526412,07	1,54
I_2	547152	536478,33	1,95
I_3	**23671**	23668,63	0,01
I_4	916174	913715,52	0,27
I_5	311286	308595,45	0,86
I_6	367586	364108,13	0,95
I_7	8969	8915,18	0,60
I_8	41638	41274,00	0,87
I_9	134025123	131354923,50	1,99
I_{10}	130750874	128540131,50	1,69
I_{11}	63429	62244,84	1,87
I_{12}	132897	127901,75	3,76
		gap_{li} médio =	1,36

Tabela 5.22: Valores da nova execução do CPLEX com limite de duas horas.

Considerando este novo critério de parada o gap_{li} médio do CPLEX ficou em 1,36% (Tabela 5.22), de G1 em 0,98%, de G2 em 0,83%, de G3 em 0,94%, G4 igual a 0,82%, ***G5 com 0,78%*** e G6' com 0,83% (Tabela 5.23). A descrição da Tabela 5.23 é idêntica a da Tabela 5.19. G5 obteve novamente o melhor desempenho obtendo as melhores soluções em oito de um total de doze instâncias, G4 com cinco e G6' com 3 melhores soluções, G2 e o CPLEX obtiveram uma solução melhor para cada método.

Observa-se desta forma a importância de se usar reconexão de caminhos intensiva

Id.	CPLEX	Melhores Valores					
		G1	G2	G3	G4	G5	G6'
I_1	1,54	0,97	0,80	0,93	**0,76**	**0,76**	0,80
I_2	1,95	1,12	0,73	0,99	**0,66**	0,70	0,82
I_3	**0,01**	0,04	0,04	0,05	0,03	**0,01**	**0,01**
I_4	0,27	0,21	0,19	0,21	0,20	**0,14**	**0,14**
I_5	0,86	0,55	0,45	0,51	**0,43**	**0,43**	0,45
I_6	0,95	0,69	0,61	0,66	0,58	**0,56**	0,59
I_7	0,60	0,67	0,51	0,60	**0,49**	0,53	**0,49**
I_8	0,87	0,67	0,63	0,65	0,64	**0,43**	0,63
I_9	1,99	1,63	1,61	1,64	1,60	**1,40**	1,57
I_{10}	1,69	1,28	1,27	1,31	1,27	**1,13**	1,16
I_{11}	1,87	1,68	1,28	1,60	**1,19**	1,36	1,30
I_{12}	3,76	2,23	**1,80**	2,14	1,96	1,86	1,94
gap_{li} médio	1,36	0,98	0,83	0,94	0,82	**0,78**	0,83

Tabela 5.23: Gap_{li}s dos melhores valores das heurísticas GRASP com limite de tempo.

ao final do GRASP (G2 e G4) e a reconexão de caminhos RC3 e RC4 em conjunto ao método de descida baseado no VND nos GRASPs G5 e G6'. Nesta segunda bateria de testes, observa-se que apesar de G2, G4, G5 e G6' exigirem mais tempo por iteração, estas necessitam de menos iterações para atingir bons limites superiores de um valor ótimo.

Id.	Valores Médios					
	G1	G2	G3	G4	G5	G6'
I_1	1,78	1,18	1,21	1,18	**1,09**	1,35
I_2	1,93	1,24	1,36	1,22	**1,09**	1,41
I_3	0,35	0,22	0,14	0,18	**0,06**	0,32
I_4	0,28	0,23	0,25	0,24	**0,17**	0,19
I_5	0,85	0,64	**0,63**	**0,63**	0,64	0,78
I_6	0,92	**0,73**	0,78	**0,73**	0,75	0,79
I_7	1,09	0,82	**0,78**	**0,78**	0,83	1,00
I_8	0,82	0,71	0,75	0,72	**0,50**	0,73
I_9	1,88	1,76	1,79	1,76	**1,61**	1,75
I_{10}	1,54	1,44	1,47	1,42	**1,26**	1,46
I_{11}	3,30	2,37	**2,01**	2,30	2,52	3,21
I_{12}	3,32	2,43	2,70	2,54	**2,34**	2,62
gap_{li} médio	1,51	1,15	1,16	1,14	**1,07**	1,30

Tabela 5.24: Gap_{li}s dos valores médios das heurísticas GRASP com limite de tempo.

A Tabela 5.24 ilustra o desempenho médio das heurísticas utilizando um tempo limite como critério de parada. A descrição da Tabela 5.24 é idêntica a da Tabela 5.20. O gap_{li} médio do G1 ficou em 1,51%, de G2 em 1,15%, de G3 em 1,16%, de G4 em 1,14%, de G5 igual a 1,07% e G6' com 1,30%. Novamente os resultados mostram a superioridade da

versão G5 em relação às outras. No entanto ressalta-se o bom comportamento médio das seis heurísticas aqui propostas mostrando que todas elas possuem uma robustez necessária para a sua confiabilidade prática.

Instâncias	Id.	#nós	$\#V_i$	Tipo	Melhores Valores		Valores Médios	
					G1	G4	G1	G4
49-pcb1173	I_{13}	1173	49	6	0,40	**0,00**	2,94	**0,00**
100-pcb1173	I_{14}	1173	100	6	0,54	**0,00**	2,87	**0,00**
144-pcb1173	I_{15}	1173	144	6	0,08	**0,00**	2,68	**0,00**
10-nrw1379	I_{16}	1379	10	6	0,66	**0,00**	2,33	**0,00**
12-nrw1379	I_{17}	1379	12	6	0,26	**0,00**	2,96	**0,00**
1500-10-503	I_{18}	1500	10	5	0,32	**0,00**	1,03	**0,00**
1500-20-504	I_{19}	1500	20	5	0,06	**0,00**	1,36	**0,00**
1500-50-505	I_{20}	1500	50	5	0,03	**0,00**	0,65	**0,00**
1500-100-506	I_{21}	1500	100	5	0,52	**0,00**	1,19	**0,00**
1500-150-507	I_{22}	1500	150	5	0,01	**0,00**	0,43	**0,00**
2000-10-a	I_{23}	2000	10	4	0,86	**0,00**	0,53	**0,00**
2000-10-h	I_{24}	2000	10	4	0,25	**0,00**	1,11	**0,00**
2000-10-z	I_{25}	2000	10	4	0,17	**0,00**	1,29	**0,00**
2000-10-x1	I_{26}	2000	10	4	0,91	**0,00**	0,49	**0,00**
2000-10-x2	I_{27}	2000	10	4	0,19	**0,00**	1,64	**0,00**
				gap_h médio =	0,35	**0,00**	1,57	**0,00**

Tabela 5.25: Comparação entre G1 e G4 para instâncias de grande porte.

A eficiência de se inserir reconexão de caminhos (RC2) durante as iterações e o uso de reconexão de caminhos (RC1) no final do GRASP (G4) foi comparado com o GRASP tradicional (G1) utilizando uma nova bateria de testes em instâncias de grande porte. A estratégia para a escolha das instâncias foi realizada da seguinte forma: fixa-se o número de vértices e varia-se o número de grupos (instâncias do tipo 5 e 6) ou o *layout* dos grupos (instâncias do tipo 4). Como as instâncias são de portes maiores é estabelecido um limite de 1080 segundos para cada execução de G1 e G4. As execuções das heurísticas GRASP são realizadas 10 vezes num total de três horas para cada instância. A Tabela 5.25 mostra as instâncias escolhidas para comparar G1 com G4 com seus identificadores, número de nós, número de grupos, tipo e os valores encontrados expresso em *gaps*. O percentual do *gap* dos melhores valores é calculado usando a equação 5.2, onde vh é o melhor valor alcançado pela heurística (G1 ou G4) e mv o melhor valor encontrado entre G1 e G4. Para o percentual do *gap* dos valores médios, vh é o valor médio alcançado pela heurística (G1 ou G4) e mv o melhor valor médio encontrado entre G1 e G4.

Observa-se que novamente a introdução dos módulos de reconexão de caminhos (RC1) e (RC2) em G4 é fundamental para encontrar soluções de boa qualidade. De fato na Tabela

5.25, o gap_h médio dos melhores valores encontrados por G1 é 0,35% e de G4 igual à zero. Nos valores médios, o gap_h médio de G1 foi igual a 1,57% e de G4 permanece igual à zero.

A seguir verifica-se o desempenho da introdução de módulos de reconexão de caminhos (RC1 e RC2) utilizando-se análises gráficas das distribuições de probabilidade acumulativa sobre as instâncias (AIEX; RESENDE; RIBEIRO, 2007). O desempenho analisado foi de G1 e G4. Foram consideradas três instâncias com número de vértices diferentes, dadas a seguir: a instância I_{11} (200-4-h), onde a melhor solução foi encontrada num tempo de 7200s usando o método exato CPLEX, cujo valor foi de 62835, a instância I_4 (144-rat783), onde a melhor solução conhecida (914965) foi alcançada pelo G4 em um tempo médio de 4132,28s e a instância I_{10} (C1k.1), onde o melhor valor de 129872457 foi obtido por G4 num tempo médio de 10297,08. A Tabela 5.26 apresenta os tempos médios das instâncias e a melhor solução encontrada.

Id.	Tempos Médios (s)		Melhores Valores
	G1	G4	
I_{11}	15,75	42,37	62835 (CPLEX)
I_4	750,30	4132,28	914965 (G4)
I_{10}	1765,84	10297,09	129872457 (G4)

Tabela 5.26: Os melhores valores alcançados e tempos médios medidos em segundos de G1 e G4.

A cada instância foram estabelecidos dois alvos: um alvo mais difícil denominado de *alvo_ difícil* e um mais fácil denominado de *alvo_fácil*. Os alvos para I_{11} são: (*alvo_ difícil*) igual a 63600 (1,21% acima do melhor valor) e (*alvo_fácil*) com 63800 (1,54% do melhor valor). Para I_4: (*alvo_ difícil*) com valor de 916400 (0,15% da melhor solução encontrada) e (*alvo_fácil*) com 916700 (0,19% acima da melhor solução). Para I_{10} o (*alvo_ difícil*) foi estabelecido com o valor de 130280000 (0,31% da melhor solução encontrada) e (*alvo_fácil*) o valor de 130400000 (0,41% acima da melhor solução).

Id.	*alvo_ difícil*		*alvo_fácil*		Melhores Valores		Valores Médios	
	Valor	(%)	Valor	(%)	G1	G4	G1	G4
I_{11}	63600	1,21	63800	1,54	63493	63050	64368,21	63685,84
I_4	916400	0,15	916700	0,19	915651	914965	916324,00	915468,18
I_{10}	130280000	0,31	130400000	0,41	130250305	129872457	130545650	130160400

Tabela 5.27: Valores de alvos para as instâncias.

O limite de tempo para alcançarem os alvos foram 85s para a instância I_{11} e 3600s para a I_4 e I_{10}, tempo o suficiente para os alvos executarem a fase de pós-otimização do G4. Foram executados 100 vezes para cada instância/alvo com *seed* diferente o que tornam estas execuções independentes. A Tabela 5.27 mostram os alvos escolhidos e a

porcentagem em relação a melhor solução encontrada (Tabela 5.26), junto aos valores médios e melhores valores do G1 e G4.

No gráfico da Figura 5.9 observa-se o desempenho de G1 e G4 para a instância I_{11}. O G1 tem a probabilidade de 100% de alcançar o *alvo_fácil* em 21s, enquanto o G4 alcança em 7s com a mesma probabilidade.

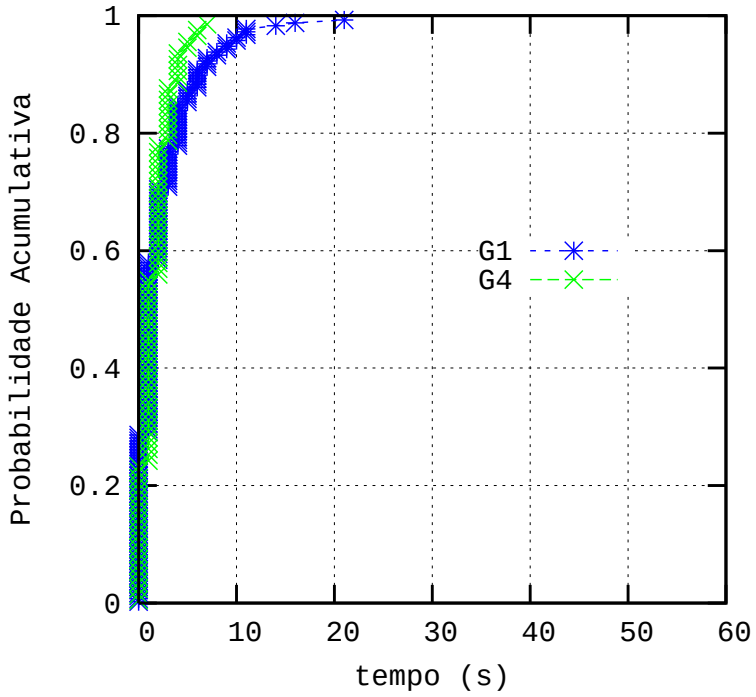


Figura 5.9: DPA realizada sobre I_{11} para *alvo_fácil*.

A Figura 5.10 apresenta-se o desempenho de G1 e G4 para a instância I_{11} e *alvo_difícil*. A probabilidade do G4 alcançar este alvo com 100% ocorre em 35s, enquanto que o G1 em 35s alcança somente 75% do alvo. O G1 alcançou o tempo limite estabelecido (85s) em duas execuções.

No gráfico da Figura 5.11 é mostrado o desempenho de G1 e G4 para a instância I_4, *alvo_fácil*. A probabilidade de 100% de G1 alcançar o alvo ocorre em 99s, enquanto o G4 em 72s. A Figura 5.12 apresenta-se o desempenho de G1 e G4 para a instância I_4, (*alvo_difícil*). Para este alvo em 1250s o G4 apresenta a probabilidade de 100%, enquanto que G1 atinge o alvo com 100% de probabilidade em 3001s.

A Figura 5.13 apresenta para instância I_{10} e *alvo_fácil* a distribuição de probabilidade acumulativa de G1 e G4. O G4 alcança o alvo em 433s com probabilidade de 100%, enquanto que G1 em 360s.

A distribuição de probabilidade acumulativa de G1 e G4 realizada para a instância I_{10}

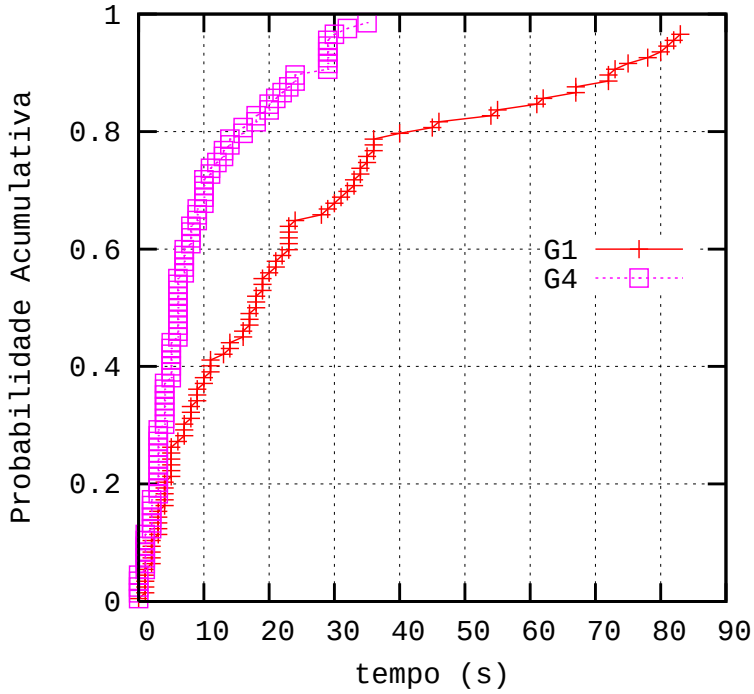


Figura 5.10: DPA realizada sobre I_{11} para *alvo_difícil*.

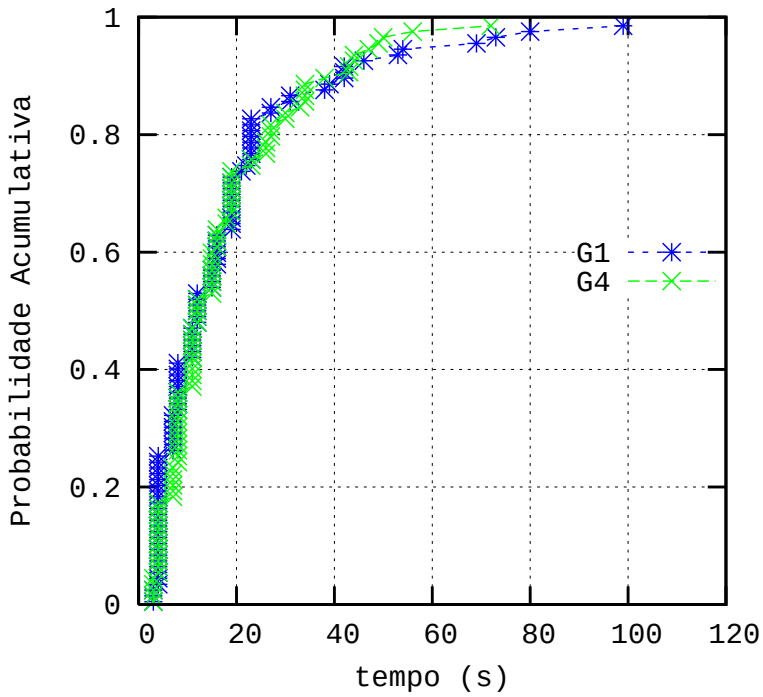


Figura 5.11: DPA realizada sobre I_4 para o *alvo_fácil*.

(*alvo_difícil*) é mostrada na Figura 5.14. Em 3478s, G4 consegue alcançar 95% do alvo, enquanto que G1 com 3488s só consegue 46%. G4 alcançou o tempo limite estabelecido (3600s) somente em três execuções e G1 em 53 execuções.

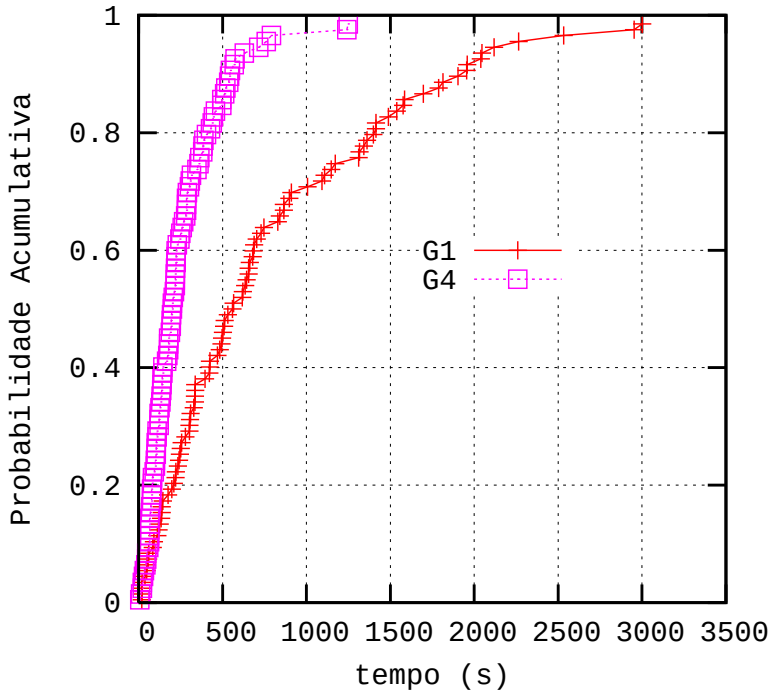


Figura 5.12: DPA realizada sobre I_4 para o *alvo_difícil*.

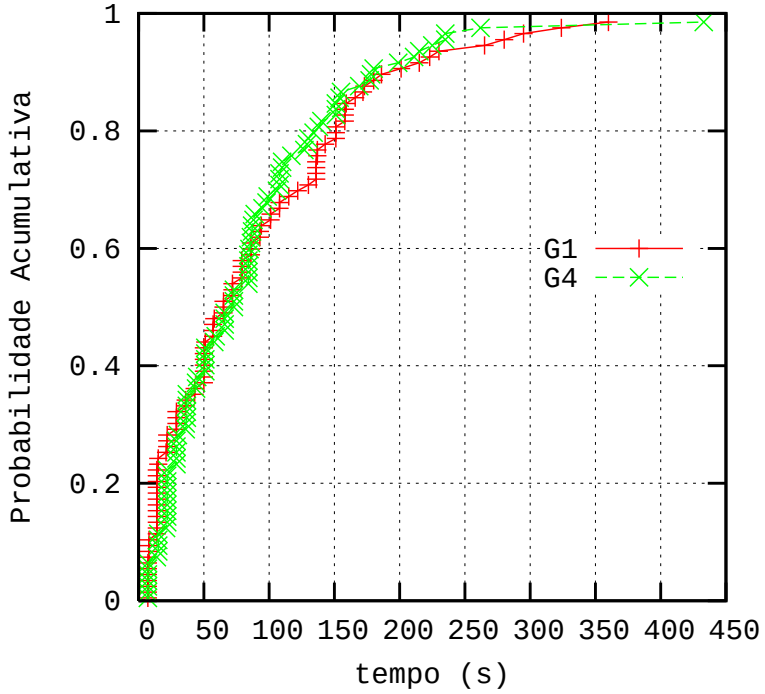


Figura 5.13: DPA realizada sobre I_{10} para o *alvo_fácil*.

Nas distribuições de probabilidades mostradas nas Figuras 5.9 a 5.14 observa-se que G4 apresenta comportamento melhor do que G1 com probabilidades maiores de alcançar os alvos, principalmente quando o alvo torna-se mais difícil. Isto reforça a importância da inserção dos módulos de reconexão de caminhos (RC1) e (RC2) no GRASP (G4) para conseguir soluções de melhor qualidade do que as obtidas com a versão tradicional G1 e

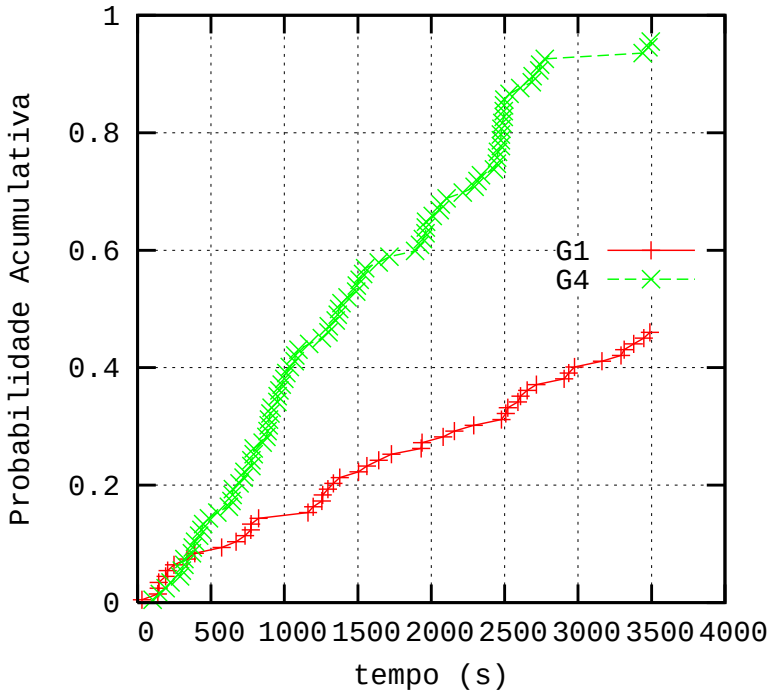


Figura 5.14: DPA realizada sobre I_{10} para o *alvo_ difícil*.

tempos computacionais razoáveis.

As comparações entre as heurísticas GRASP (G1, G2, G3, G4, G5 e G6') e o CPLEX foram mostradas nas Tabelas5.19, 5.20, 5.23 e 5.24. Na comparação da heurística G6 variando o valor de p com G5 os resultados são bem próximos e a favor de G5, somente em alguns casos G6 superou G5. Ao ampliar o número de testes entre G6' e G5 com dois tipos de critério de parada (200 iterações e limite de tempo), G5 mostra-se claramente como a melhor opção. Observa-se a necessidade de se armazenar um conjunto de soluções elite de boa qualidade para alcançar resultados promissores. Os resultados obtidos pelas heurísticas G2, G4 e G5 confirmam esta necessidade.

Novos testes foram efetuados apenas entre G4, G5 e o método exato CPLEX restritos às instâncias menores, mas sem limitação de tempo para o CPLEX. Na primeira coluna da Tabela 5.28 são mostradas as instâncias, na segunda coluna encontra-se o valor obtido pelo CPLEX, na terceira coluna o tempo demandado pelo CPLEX em segundos t_{CPLEX}, na quarta o *gap* do melhor valor obtido por G4(%) em relação ao valor obtido pelo CPLEX (gap_{hc} - calculado pela equação 5.4), na quinta o tempo médio de execução t_{G4}(s) de G4 medido em segundos. Na sexta e sétima colunas são mostradas o gap_{hc} G5(%) e o tempo médio de execução t_{G5}(s) do G5.

O percentual do gap_{hc} é calculado da seguinte forma:

$$gap_{hc} = 100 * \left(\frac{vh - vcplex}{vcplex} \right), \tag{5.4}$$

onde, vh é o melhor valor alcançado pela heurística (G4 ou G5) e $vcplex$ o valor da solução ótima encontrada pelo CPLEX.

Instâncias	CPLEX	t_{CPLEX}(s)	G4(%)	t_{G4}(s)	G5(%)	t_{G5}(s)
5-eil51	437	12,31	**0,00**	1,00	**0,00**	1,50
10-eil51	440	74,38	**0,00**	1,00	**0,00**	1,30
15-eil51	437	2,04	**0,00**	1,00	**0,00**	1,40
5-berlin52	7991	201,80	**0,00**	1,20	**0,00**	1,80
10-berlin52	7896	89,17	**0,00**	1,10	**0,00**	1,60
15-berlin52	8049	75,93	**0,00**	1,10	**0,00**	1,60
5-st70	695	13790,11	**0,00**	2,30	**0,00**	3,40
10-st70	691	4581,00	**0,00**	2,00	**0,00**	2,50
15-st70	692	883,50	**0,00**	2,00	**0,00**	2,90
5-eil76	559	83,70	0,36	2,70	0,54	3,80
10-eil76	561	254,30	0,53	2,40	0,71	3,20
15-eil76	565	49,66	0,35	2,50	0,53	3,60
5-pr76	108590	99,29	**0,00**	2,70	**0,00**	5,50
10-pr76	109538	238,13	**0,00**	2,20	**0,00**	4,00
15-pr76	110678	261,94	0,15	2,30	0,15	4,40
10-rat99	1238	650,67	**0,00**	4,90	**0,00**	9,70
25-rat99	1269	351,15	0,63	4,70	**0,00**	9,80
50-rat99	1249	2797,58	0,72	4,90	0,80	10,50
25-kroA100	21917	3513,57	**0,00**	4,70	**0,00**	8,60
50-kroA100	21453	947,55	**0,00**	5,20	**0,00**	9,60
10-kroB100	22440	4991,44	0,16	4,80	**0,00**	9,30
50-kroB100	22355	2579,22	1,33	5,20	0,54	9,50
25-eil101	663	709,45	1,36	4,60	1,21	7,80
50-eil101	644	275,33	1,09	5,40	1,09	9,10
25-lin105	14438	6224,55	**0,00**	5,10	**0,00**	10,50
50-lin105	14379	1577,21	1,08	5,70	**0,00**	12,40
75-lin105	14521	15886,77	0,59	6,40	0,23	14,10
valores médios =	-	2266,73	0,25	3,3	***0,21***	6,05

Tabela 5.28: Comparação do CPLEX com G4 e G5 para instâncias do *tipo 1* de pequeno porte.

As heurísticas G4 e G5 foram executadas com número de iterações fixo igual a 200. Para todas as instâncias de pequeno porte, o CPLEX encontrou soluções ótimas. G4 encontrou soluções ótimas em 15 de um total de 27 instâncias e o gap_{hc} médio sobre todas as instâncias nos valores obtidos pelo G4 em relação ao ótimo foi de 0,25% com tempo médio geral de 3,3s (Tabela 5.28). Enquanto que G5 alcançou 18 soluções ótimas com tempo médio de 6,05s. O gap_{hc} médio de G5 foi igual a ***0,21%***. Este resultado reforça o

potencial de G5 em encontrar soluções melhores em tempo computacional viável.

Instâncias	Id.	#nós	$\#V_i$	Tipo	CPLEX		
					Valor	Lim. Inf.	(%)
i-50-gil262	J_1	262	50	1	135529	135374,68	0,11
10-lin318	J_2	318	10	1	534640	526412,07	1,54
10-pcb442	J_3	442	10	1	547152	536478,33	1,95
C1k.0	J_4	1000	10	2	134025123	131354923,50	1,99
C1k.1	J_5	1000	10	2	130750874	128540131,50	1,69
C1k.2	J_6	1000	10	2	144341485	141501445	1,97
300-6	J_7	300	6	3	8969	8915,18	0,60
400-6	J_8	400	6	3	9117	9021,51	1,05
700-20	J_9	700	20	3	41638	41274,00	0,87
200-4-h	J_{10}	200	4	4	63429	62244,84	1,87
200-4-x1	J_{11}	200	4	4	**60797**	60242,96	**0,91**
600-8-z	J_{12}	600	8	4	132897	127901,75	3,76
600-8-x2	J_{13}	600	8	4	132228	127901,75	3,27
300-5-108	J_{14}	300	5	5	68361	67128,93	1,80
300-20-111	J_{15}	300	20	5	311286	308595,45	0,86
500-15-306	J_{16}	500	15	5	196001	193522,8	1,26
500-25-308	J_{17}	500	25	5	367586	364108,13	0,95
25-eil101	J_{18}	101	25	6	**23671**	23668,63	**0,01**
42-a280	J_{19}	280	42	6	130043	129560,53	0,37
144-rat783	J_{20}	783	144	6	916174	913715,52	0,27
						gap_{li} médio =	1,36

Tabela 5.29: Instâncias com seus identificadores (J_k), número de nós, número de grupos, tipo e valores alcançados pelo CPLEX com limite de duas horas.

Diante dos resultados mostrados até o momento verifica-se que o G5 apresenta desempenho melhor quando comparado aos demais GRASPs desenvolvidos. Os testes anteriores realizados com os GRASPs utilizaram na fase de construção o algoritmo baseado na Inserção mais Próxima com as arestas inter-grupos penalizadas (IMPP). Para verificar se o método IMPP é sempre eficaz para todo o conjunto de instâncias geradas em seus diversos *tipos*, realizam-se novos testes computacionais e escolhe-se o construtivo IMPnP para comparação com o IMPP, porque a metodologia de construir soluções parciais é equivalente. O objetivo destes testes é verificar se a penalização das arestas inter-grupos é sempre a melhor alternativa do que a não penalização nos diversos tipos de instâncias. As heurísticas G5-IMPP e G5-IMPnP serão executadas cada uma 10 vezes com limite de 720 segundos em cada execução (duas horas no total). Foram utilizados nos testes computacionais as instâncias mostradas na Tabela 5.29. A descrição das colunas da Tabela 5.29 é idêntica a da Tabela 5.14. Na Tabela 5.29 mostram-se os valores obtidos pelo CPLEX com limite de tempo estabelecido em duas horas.

Id.	Melhores Valores		Valores Médios	
	G5-IMPP	G5-IMPnP	G5-IMPP	G5-IMPnP
J_1	**0,07**	0,08	**0,28**	0,32
J_2	**0,76**	0,81	**1,08**	1,64
J_3	**0,70**	0,76	**1,08**	1,63
J_4	**1,38**	1,47	**1,58**	1,94
J_5	1,12	**1,07**	**1,24**	1,60
J_6	**1,48**	**1,48**	**1,82**	1,83
J_7	**0,52**	0,53	**0,83**	1,32
J_8	0,82	**0,79**	**1,39**	1,53
J_9	**0,42**	0,63	**0,50**	0,82
J_{10}	**1,35**	1,36	**2,46**	3,45
J_{11}	1,04	1,04	**2,97**	3,21
J_{12}	1,83	**1,67**	**2,29**	2,63
J_{13}	1,83	**1,81**	2,81	**2,70**
J_{14}	1,47	**1,33**	**2,18**	2,46
J_{15}	**0,43**	0,47	**0,64**	1,02
J_{16}	**0,95**	1,01	**1,33**	1,49
J_{17}	**0,56**	0,57	**0,74**	0,90
J_{18}	**0,01**	0,04	**0,06**	0,36
J_{19}	**0,17**	**0,17**	**0,45**	0,48
J_{20}	**0,14**	0,15	**0,17**	0,24
gap_{li} médio	**0,85**	0,86	**1,30**	1,58

Tabela 5.30: Gap_{li} dos valores das heurísticas G5 com limite de tempo.

Na Tabela 5.30 mostra-se a comparação entre G5-IMPP e G5-IMPnP considerando as instâncias da Tabela 5.29 com limite de tempo. A primeira coluna da Tabela 5.30 mostra os identificadores das instâncias, na segunda coluna o gap_{li}, calculado pela equação 5.3, do melhor valor encontrado por G5-IMPP e na terceira o gap_{li} do melhor valor encontrado por G5-IMPnP. Nas duas últimas colunas são mostrados os gap_{li}s dos valores médios alcançados por G5-IMPP e G5-IMPnP. O gap_{li} médio dos melhores valores de G5-IMPP ficou em ***0,85%*** e de G5-IMPnP igual a 0,86%. O gap_{li} do CPLEX para estas instâncias com o critério de parada estabelecido de duas horas foi igual 1,36%. G5-IMPP obteve o melhor desempenho obtendo as melhores soluções em quatorze de um total de vinte instâncias, G5-IMPnP com sete e CPLEX com duas.

A Tabela 5.30, também ilustra o desempenho médio das heurísticas G5-IMPP e G5-IMPnP utilizando como critério de parada o limite de tempo. O gap_{li} médio dos valores médios de G5-IMPP foi igual a ***1,30%*** e de G5-IMPnP em 1,58%. Estes resultados mostram a uma leve superioridade da versão G5-IMPP em relação a G5-IMPnP. No entanto, ressalta-se o bom comportamento médio das duas heurísticas propostas mostrando que

ambas possuem uma robustez necessária para fins práticos.

5.4 Heurísticas ILS

5.4.1 Análise do Algoritmo ILS para o PCVG

Nesta seção são apresentados análises sobre as diversas formas de inserção dos módulos de perturbação, tais como *Pert1*, *Pert2* e *Pert3*, dos módulos de critérios de aceitação *CritAceit1* e *CritAceit2* e dos diversos métodos de buscas locais. Estes módulos foram descritos na seção 4.5.1. A calibração dos parâmetros de ajuste do ILS para o PCVG, também será mostrada.

Algoritmo 24: Algoritmo ILS-VND.

```
1: Início;
2: iterprob ← 1;
3: S^0 ← Construcao_Aleatoria_Gulosa;
4: S ← VND(S^0);
5: S* ← S
6: para i=1 to iterext faça
7:    α= calcularalfa( );
8:    S^0 ← Construcao_Aleatoria_Gulosa;
9:    S ← VND(S^0);
10:   para i=1 to maxiter faça
11:      S' ← Pert3(S);
12:      S'' ← VND(S');
13:      S ← CritAceit2(S, S'',história);
14:      se S for melhor que S* então
15:         S* ← S;
16:      fim se
17:   fim para
18:   se iterprob = maxprob então
19:      atualizardisalfas( );
20:      iterprob ← 0;
21:   fim se
22:   iterprob++;
23: fim para
24: Retornar Solução S*;
25: Fim.
```

Os resultados computacionais apresentados neste capítulo mostram as análises sobre as formas de perturbações, os critérios de aceitação e as buscas locais e de acordo com os testes, a heurística ILS-VND obteve melhor desempenho e está representada pelo Algoritmo 24. Neste algoritmo, o número máximo de iterações para mudar a distribuição

de probabilidade dos alfas é representado pelo parâmetro *maxprob*, o parâmetro *iterprob* representa o incremento para atingir o valor *maxprob*, o *iterext* denota o número máximo de iterações externas e o *maxiter* o número máximo de iterações internas.

No Algoritmo 24, a solução S^0 representa uma solução construída pelo módulo *Construcao_Aleatoria_Gulosa*, no qual é utilizada para iniciar o algoritmo e para gerar soluções diversificadas a cada iteração mais externa. A solução S corresponde a solução corrente no qual é aplicada uma perturbação, a solução S' é a solução que está num estado intermediário, onde é aplicada uma busca local obtendo a solução S'' e a solução S^* representa a melhor solução encontrada. No Algoritmo 24 são mostrados ainda os módulos: de perturbação (*Pert3*), de busca local (*VND*) e *CritAceit2* (critério de aceitação). Estes módulos foram os que permitiram as heurísticas utilizando o método ILS encontrar as melhores soluções para o PCVG. O procedimento `calcularalfa( )` seleciona o alfa (α) na versão reativa a ser utilizado pelo módulo construtivo e o procedimento `atualizardisalfas( )` atualiza a distribuição de probabilidade dos alfas, sempre que *iterprob* atingir seu valor máximo igual a *maxprob*.

No Algoritmo 24, no passo 3, inicialmente é obtido uma solução S^0 e, no passo 7, é selecionado um valor para α através de uma distribuição de probabilidade. Para cada iteração externa do algoritmo, passos 6 a 23, uma solução nova S^0 é construída no passo 8 e, no passo 9, a busca local é executada pelo VND obtendo a solução S. Para cada iteração interna do algoritmo, passos 10 a 17, a perturbação é aplicada gerando a solução S' (passo 11) e a busca local é executada de novo pelo VND produzindo a solução S'', passo 12. No passo 13, é executado o critério de aceitação e a melhor solução S^* é atualizada no passo 15. No passo 19, os alfas são atualizados pela distribuição de probabilidade na versão reativa e finalmente, no passo 24, a melhor solução S^* é retornada.

As análises apresentadas foram realizadas pelo método de inferência nos dados. Neste contexto necessita-se escolher um conjunto de instâncias para testes. Este conjunto de instâncias foi escolhido de forma a ter no mínimo uma instância de cada *tipo* e é mostrado na Tabela 5.31, subconjunto da Tabela 5.14. Na primeira coluna apresenta-se a instância, na segunda o identificador da instância, na terceira o número de vértices (# nós), na quarta coluna o número de grupos (# V_i) e na quinta o tipo de instância. Nas últimas três colunas apresenta-se os melhores valores alcançados pelos algoritmos (G4, G5, G6' ou CPLEX) e o tempo de execução em segundos.

Com os módulos definidos na seção 4.5.1 e com um conjunto de instâncias escolhidas para testes realiza-se a execução do Algoritmo ILS. O critério de parada para estes testes

Instâncias	Id.	#nós	$\#V_i$	Tipo	Melhores Valores Conhecidos		
					Valor	Algoritmos	Tempo(s)
10-lin318	I_1	318	10	1	43705	G6'	238,40[1]
10-pcb442	I_2	442	10	1	55850	G5	1385,70[1]
25-eil101	I_3	101	25	6	671	CPLEX	442,00
300-20-111	I_5	300	20	5	13731	G4	720,00[1]
500-25-308	I_6	500	25	5	16629	G5	2303,80[1]
300-6	I_7	300	6	3	796	CPLEX	25200,00
C1k.1	I_{10}	1000	10	2	12724157	G4	10297,08[1]
200-4-h	I_{11}	200	4	4	10795	CPLEX	25200,00
600-8-z	I_{12}	600	8	4	19872	G4	1652,09[1]

[1]Tempo Médio

Tabela 5.31: Melhores valores conhecidos pelos algoritmos para as instâncias testes.

foi fixado pelo número de iterações internas (*iter*) igual a 1000 e uma única solução inicial aleatória gulosa construída. No módulo *Construcao_Aleatoria_Gulosa* utilizou-se o algoritmo construtivo baseado na Inserção mais Próxima com as arestas inter-grupos Penalizadas (IMPP). Como existe somente uma forma de consttruir a solução inicial e uma busca local pré-definida (2-Optimal) realizaram-se os testes para definir a melhor opção para a perturbação e o critério de aceitação. Fixou-se o critério de aceitação igual *CritAceit1* e verifica-se as soluções produzidas pela busca local.

Neste sentido percebe-se que a forma de realizar a perturbação sempre influencia na solução final. A perturbação por ordem de desempenho da melhor para a pior foi: *Pert3*, *Pert2* e *Pert1*. Com a perturbação definida avaliaram-se os dois critérios de aceitação. O segundo critério (*CritAceit2*) permitiu ótimos locais de melhor qualidade. Os testes utilizando os tipos de pertubação e critérios de aceitação foram avaliados através da impressão das soluções finais produzidas pelo Algoritmo ILS.

Nestes testes foi observado que o algoritmo sofre de estagnação, a melhor solução não é atualizada a partir de um certo número de iterações. Para resolver isto, pode-se construir uma população de soluções e aplicar iterativamente a sequência canônica de perturbação, busca local e critério de aceitação ou através de uma forma alternativa, onde o algoritmo inicia com soluções novas construídas. A cada solução nova realiza-se a sequência canônica dos módulos: perturbação, busca local e critério de aceitação. Escolheu-se a última opção de iniciar o algoritmo com soluções novas possibilitando diversificação nas soluções. As soluções são construídas sempre que o algoritmo necessitar romper a estagnação. Agora fica a pergunta, para produzir a solução final de melhor qualidade, quantas soluções novas (iterações externas) e quantas iterações internas são necessárias para sequência canônica?

Para responder esta pergunta, escolhem-se quatro combinações de iterações internas e de soluções novas . Estas soluções novas serão reiniciadas pelo algoritmo depois de executadas as iterações internas da sequência canônica dos módulos. A primeira combinação fixa em uma única solução nova e 1000 iterações internas. Na segunda, escolhem-se 10 soluções novas e 100 iterações internas. Na terceira, com 40 soluções novas e 25 iterações internas. Por último, a combinação de 100 soluções novas e 10 iterações internas.

# soluções novas (# *starts*)	# iterações internas (# *iter*)	total de soluções obtidas
1	1000	1001
10	100	1010
40	25	1040
100	10	1100

Tabela 5.32: Combinações de soluções novas (# *starts*) e de iterações internas (# *iter*).

A Tabela 5.32 detalha as combinações: na primeira coluna apresenta-se o número de soluções novas, mais precisamente quantos *starts* o algoritmo irá realizar. Na segunda coluna o número de iterações internas (*iter*) nas quais se realizam a perturbação, busca local e o critério de aceitação e na terceira coluna o total de soluções obtidas pela combinação de soluções novas e de iterações internas. Estas combinações permitem que os testes sobre as instâncias tenham um total de soluções obtidas bem próximas (última coluna da Tabela 5.32) para a comparação ser mais precisa. Com isto, os tempos de execuções serão praticamente iguais. Para gerar os histogramas e aumentar a confiabilidade nas análises dos dados consideram-se somente as 1001 soluções iniciais obtidas em cada combinação.

Com os melhores valores conhecidos das instâncias dado pela Tabela 5.31 e as combinações da Tabela 5.32 executa-se o Algoritmo ILS com o construtivo IMPP, busca local 2-Optimal, perturbação *Pert3* e critério de aceitação *CritAceit2*. Para cada instância armazena-se o número de soluções com seus respectivos valores expressos em *gaps*. Para cada solução, o cálculo do *gap* é definido pela relação da diferença entre a solução alcançada pelo Algoritmo ILS menos o *melhor valor* sobre o *melhor valor*. O *melhor valor* está estabelecido na Tabela 5.31.

A Figura 5.15 ilustra o histograma do número de soluções para cada *gap* utilizando a instância 10-lin318. Por exemplo, para 1 *start*, 99 soluções alcançaram o valor de *gap* igual a 0,17% e para 40 *starts* com *gap* igual 0,1%, 193 soluções foram obtidas. Observa-se nesta instância que com o aumento no número de *starts*, os *gaps* das soluções obtidas melhoram. Verifica-se que para 1 *start* o *gap* mínimo foi de 0,15% e 10 *starts* foi de 0,10%. ***Com 40 e 100 starts o gap mínimo alcançou 0,05%***. Outra observação que se constata no Algoritmo ILS está relacionada à qualidade das soluções médias produzidas.

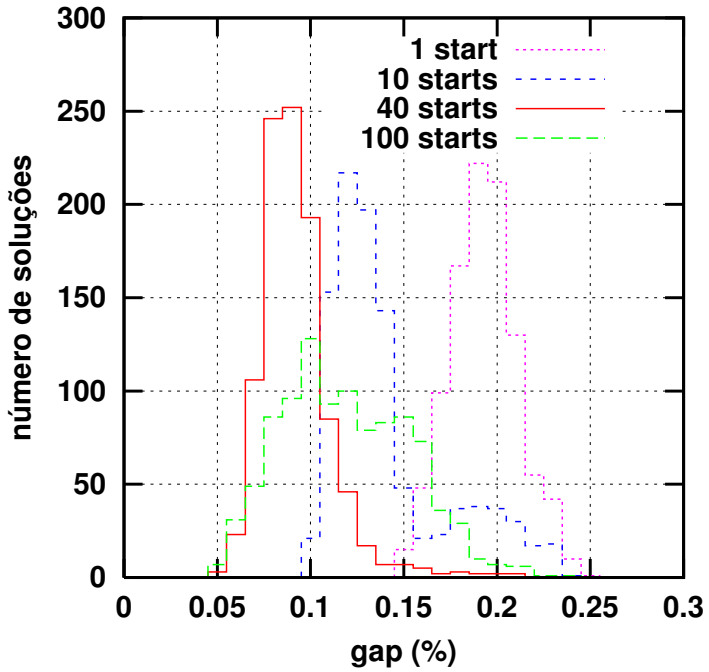


Figura 5.15: Número de soluções a cada *gap* para as diversas combinações, instância: 10-lin318.

Para fazer esta análise, definiu-se $F_m(s)$ uma função que calcula o *gap* médio das soluções geradas quando se fixa o número de *starts* (s). Assim, podem-se verificar os valores médios alcançados com os diversos valores de s. Para a instância 10-lin318 o valor de $\boldsymbol{F_m(40)}$ ***foi de 0,09%*** e $F_m(100)$ igual a 0,12%. Significa-se que com 40 *starts* o Algoritmo ILS produz soluções médias melhores do que com 100 *starts*. Isto acontece porque com 100 *starts* obtém-se várias soluções novas construídas e estas soluções sofrerão poucas perturbações e buscas locais, pois serão executadas poucas iterações do Algoritmo ILS.

A Figura 5.16 ilustra o histograma para a instância 10-pcb442. Novamente observa-se que com o aumento do número de *starts*, os valores dos *gaps mínimos* das soluções obtidas diminuem. Por exemplo, para 1 *start* o *gap* mínimo foi de 0,11%, para 10 *starts* foi de 0,08%, ***40 starts foi 0,06%*** e 100 starts o menor *gap* alcançou 0,07%. O valor de $\boldsymbol{F_m(40)}$ ***foi de 0,09%*** e $F_m(100)$ igual a 0,11%.

Para a instância C1k.1, o histograma é mostrado na Figura 5.17. Para 1 *start* existem 56 soluções com *gap* mínimo igual a 0,07%, para 10 *starts* sete soluções com *gap* mínimo de 0,04%, para 40 *starts* 33 soluções com *gap* mínimo 0,04% e para ***100 starts e gap mínimo de 0,04 existem 113 soluções***. O valor de $F_m(1)$ foi de 0,08%, o valor de

$F_m(10)$ de 0,06% e os valores de $\boldsymbol{F_m(40)}$ *e* $\boldsymbol{F_m(100)}$ *são iguais a 0,05%*.

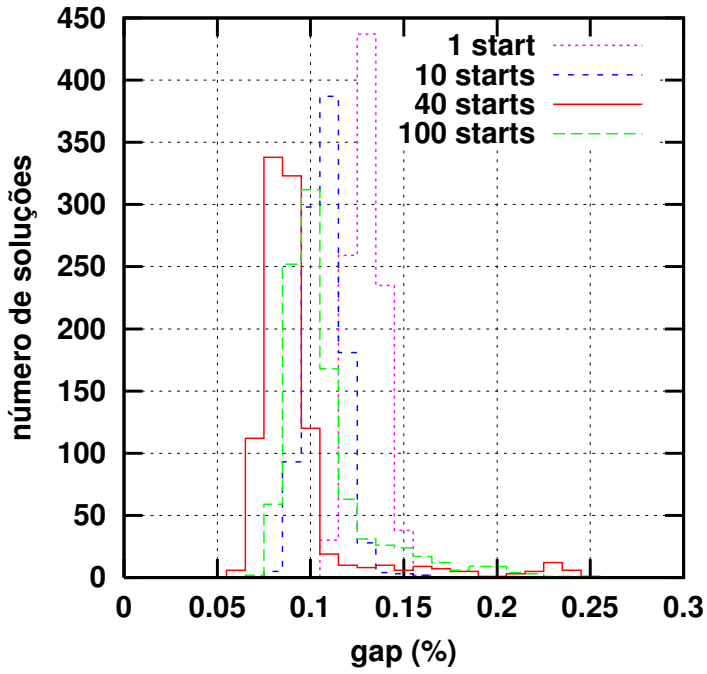


Figura 5.16: Número de soluções a cada *gap* para as diversas combinações, instância: 10-pcb442.

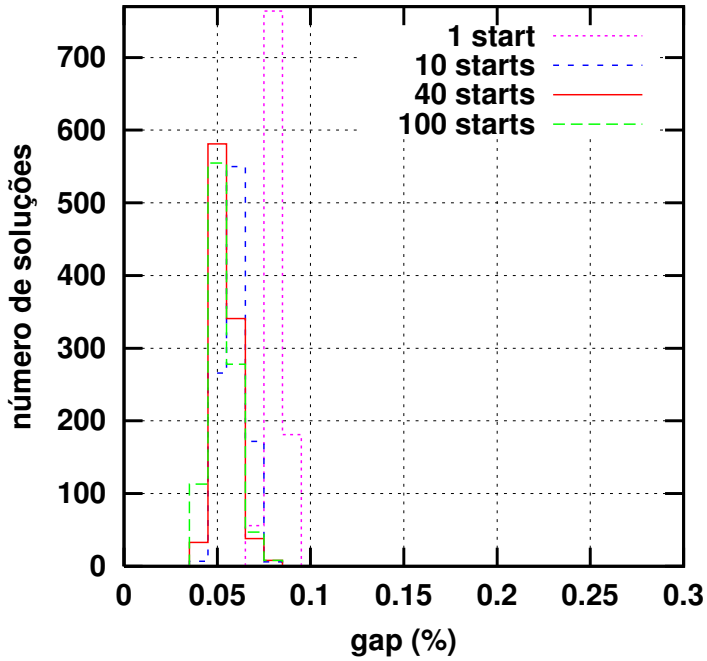


Figura 5.17: Número de soluções a cada *gap* para as diversas combinações, instância: C1k.1.

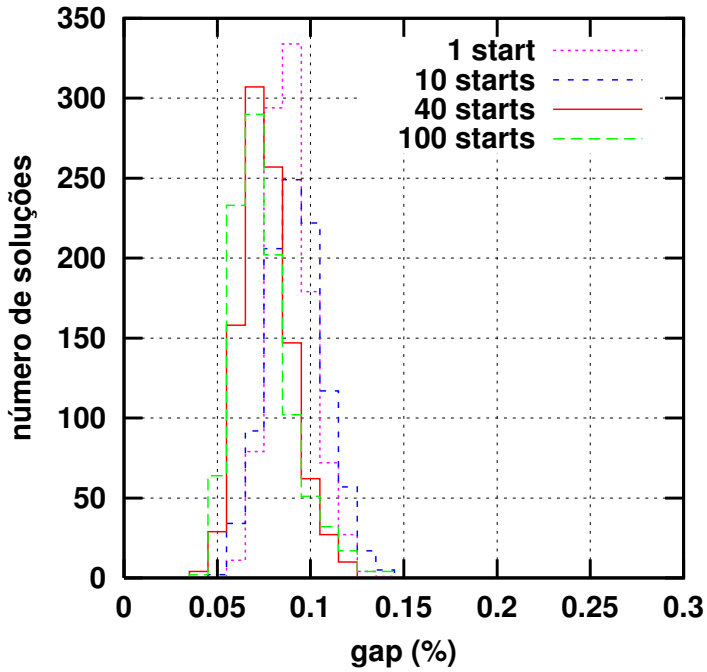


Figura 5.18: Número de soluções a cada *gap* para as diversas combinações, instância: 300-6.

O histograma da instância 300-6 é mostrado na Figura 5.18. Nesta instância com 1 *start* e *gap* mínimo de 0,06% foram obtidas 11 soluções, para 10 *starts* duas soluções com *gap* mínimo de 0,02%, ***para 40 starts quatro soluções com gap mínimo 0,04%*** e 100 *starts* e *gap* mínimo de 0,04% foram duas soluções. O valor de $F_m(1)$ foi de 0,09%, o valor de $F_m(10)$ de 0,09%, o valor de $F_m(40)$ igual 0,08% e $\boldsymbol{F_m(100)}$ ***obteve o valor de 0,07%***.

Diante das descrições expostas, escolher a combinação de 40 *starts* e 25 iterações internas (*iter*=25) para o Algoritmo ILS mostra-se promissor. Assim, o número de soluções novas foi fixado em 40 (*iterext*=40), mas o número de iterações internas foi aumentado para 35. Aumentando o número de iterações permite-se ao Algoritmo ILS realizar mais etapas de intensificação. O Algoritmo ILS redefinido com *iterext*=40 e *iter*=35 é descrito pelo Algoritmo 25, onde a solução S^* representa melhor solução encontrada, *maxprob* o número máximo de iterações para recalcular a distribuição de probabilidade dos alfas, *iterprob* o incremento para atingir o valor *maxprob*, *iterext* o máximo de iterações externa e *iter* o máximo de iterações internas.

Para avaliar se a introdução de outros módulos de buscas locais melhora o desempenho do Algoritmo ILS proposto, realizaram-se novos testes com as instâncias descritas pela Tabela 5.31. A Tabela 5.33 mostra os melhores valores obtidos para os diversos módulos de busca local. Na primeira coluna apresentam-se os identificadores das instâncias, na

Algoritmo 25: Algoritmo ILS

```
1: Início;
2: iterprob ← 1;
3: S^0 ← Construcao_Aleatoria_Gulosa;
4: S ← Busca_Local(S^0);
5: S* ← S
6: para i=1 to iterext faça
7:    α= calcularalfa( );
8:    S^0 ← Construcao_Aleatoria_Gulosa;
9:    S ← Busca_Local(S^0);
10:   para i=1 to iter faça
11:      S' ← Pert3(S);
12:      S'' ← Busca_Local(S');
13:      S ← CritAceit2(S,S'',história);
14:      se S for melhor que S* então
15:         S* ← S;
16:      fim se
17:   fim para
18:   se iterprob = maxprob então
19:      atualizardisalfas( );
20:      iterprob ← 0;
21:   fim se
22:   iterprob++;
23: fim para
24: Retornar Solução S*;
25: Fim.
```

segunda o *gap* dos melhores valores obtidos pela busca local 2-Optimal comparado com os melhores valores (Tabela 5.31) alcançados pelos algoritmos, na terceira e quarta colunas os *gaps* dos melhores valores alcançados pela 3-Optimal e pelo método de descida VND, respectivamente. Observa-se que o valor de *gap* negativo em negrito indica que a opção pelo método de descida VND apresenta bons resultados e que somente em uma instância o Algoritmo ILS com busca local VND não alcançou o valor ótimo dado pelo CPLEX na Tabela 5.31. O *gap* médio dos melhores valores da busca local 2-Optimal ficou em 3,92%, da 3-Optimal em 2,47% e ***do VND igual a -1,16%***.

A Tabela 5.34 mostra os *gaps* dos valores médios alcançados pelo Algoritmo ILS com seus métodos de busca local utilizados. A descrição da Tabela 5.34 é idêntica à Tabela 5.33. O *gap* médio da 2-Optimal foi 7,82%, da 3-Optimal de 5,97% e do ***VND de 0,43%***. Em três instâncias o valor médio dado pelo Algoritmo ILS com VND foi melhor que o melhor valor dado pelos algoritmos descritos na Tabela 5.31.

Pelas Tabelas 5.33 e 5.34 torna-se claro a vantagem do uso modular do ILS e a pos-

Id.	Melhores Valores do ILS		
	2-Optimal	3-Optimal	VND
I_1	3,21	1,06	**-1,19**
I_2	5,82	3,72	**-0,99**
I_3	3,87	3,13	1,49
I_5	2,71	1,11	**-1,17**
I_6	2,07	0,32	**-1,12**
I_7	2,89	1,63	**-1,76**
I_{11}	6,08	4,23	**-0,30**
I_{12}	4,74	4,56	**-4,22**
gap médio	3,92	2,47	**-1,16**

Tabela 5.33: *Gap* dos melhores valores do ILS para os diversos métodos de buscas locais.

Id.	Valores Médios do ILS		
	2-Optimal	3-Optimal	VND
I_1	7,93	5,44	0,02
I_2	8,97	7,10	0,69
I_3	6,18	4,92	3,29
I_5	5,85	2,64	1,07
I_6	3,83	2,48	**-0,29**
I_7	6,31	5,39	**-0,34**
I_{11}	13,86	10,93	1,61
I_{12}	9,64	8,85	**-2,61**
gap médio	7,82	5,97	**0,43**

Tabela 5.34: *Gap* dos valores médios do ILS para os diversos métodos de buscas locais.

sibilidade de aumentar a complexidade dos módulos de busca local sem grandes conhecimentos do problema de otimização combinatória a ser tratado. Para cada busca local e instância, o número de execuções (*nexec*) do Algoritmo ILS foi igual a 10, totalizando 240 execuções.

Uma análise do esforço computacional necessário para a execução do Algoritmo ILS utilizando diversos métodos de busca local foi realizada comparando os tempos do Algoritmo ILS (Tabela 5.35) com os tempos depreendidos pelos algoritmos da Tabela 5.31. Na Tabela 5.35 é exposto na primeira coluna os identificadores das instâncias, na segunda os tempos médios depreendidos pelo Algoritmo ILS com busca local 2-Optimal, na terceira coluna os tempos médios com a 3-Optimal, na quarta os tempos médios com o VND e na última coluna os tempos dos Algoritmos descritos na Tabela 5.31. Os valores em negrito na Tabela 5.35 mostram que o tempo computacional do Algoritmo ILS foi menor do que o tempo dos Algoritmos da Tabela 5.31. Observa-se que somente em duas situações o tempo computacional do Algoritmo ILS foi maior.

Id.	Tempos Médios do ILS (s)			Tempo (s)
	2-Optimal	3-Optimal	VND	Algoritmos (Tabela 5.31)
I_1	**90,40**	**112,40**	387,1	238,40
I_2	**238,00**	**354,00**	**1027,90**	1385,70
I_3	**3,60**	**7,00**	**15,50**	442,00
I_5	**108,00**	**129,80**	**286,10**	720,00
I_6	**343,00**	**479,40**	**1130,50**	2303,80
I_7	**77,30**	**97,50**	**264,00**	25200,00
I_{11}	**24,50**	**40,30**	**148,10**	25200,00
I_{12}	**592,40**	**684,80**	3461,80	1652,09

Tabela 5.35: Tempos do Algoritmo ILS e dos Algoritmos descritos na Tabela 5.31.

Para comprovar se não é um exagero limitar o número de iterações internas em 35 e se o número de execuções é adequado, registram-se os valores da iteração e da execução, onde aconteceram as melhores soluções (Tabela 5.33). Em particular, a Tabela 5.36 mostra estes valores quando se utiliza o Algoritmo ILS com método de busca local VND (ILS-VND). Observa-se que para a instância I_2 a melhor solução aconteceu justamente nos valores máximos de *iter* e *nexec*. Na instância I_1 o melhor valor ocorreu na décima execução do ILS-VND e na instância I_{12} o melhor valor ocorreu na iteração de número 34, bem próximo de seu valor máximo. O valor médio do número de iterações foi de 17,9 e do número de execuções igual a 6,1. Pelos melhores valores alcançados pelo ILS-VND mostrados na Tabela 5.33, pelos valores médios da Tabela 5.34 e os valores da Tabela 5.36, conclui-se que executar o ILS-VND com os parâmetros do número máximo de iterações internas igual a 35 e do número de execuções igual a 10 são promissores.

Id.	ILS-VND	
	n^o iteração	n^o execução
I_1	4	**10**
I_2	**35**	**10**
I_3	18	4
I_5	9	8
I_6	24	1
I_7	15	4
I_{11}	4	5
I_{12}	34	7
valor médio	17,9	6,1

Tabela 5.36: Valores das iterações internas e das execuções onde ocorreram as melhores soluções do ILS-VND.

Com limite de tempo de duas horas verifica-se se o número de iterações internas e o número de execuções são adequados para o ILS-VND. Assim, registram-se seus valores,

mostrados na Tabela 5.37, onde a melhor solução foi alcançada pelo Algoritmo ILS-VND (Tabela 5.39). Observa-se que para as instâncias I_2, I_6, I_7, I_8, I_9 e I_{10}, as melhores soluções aconteceram com valores das iterações internas iguais a 25, 30, 28, 23, 23 e 34, respectivamente. Para as instâncias I_6, I_9, I_{10} e I_{12} o número da execução atingiu o valor máximo e nas instâncias I_2 e I_8 atingiram os valores 8 e 9, respectivamente. O valor médio do número de iterações internas foi de 17,5 e do número de execuções igual a 5,9. Com o critério de parada definido pelo limite de tempo para execução do ILS-VND, constata-se como anteriormente que *iter* com 35 e *nexec* igual a 10 são bons valores para estes dois parâmetros.

Id.	ILS-VND (limite no tempo)	
	n^o iteração	n^o execução
I_1	6	2
I_2	25	8
I_3	1	4
I_4	17	3
I_5	12	2
I_6	30	**10**
I_7	28	2
I_8	23	9
I_9	23	**10**
I_{10}	34	**10**
I_{11}	4	1
I_{12}	7	**10**
valor médio	17,5	5,9

Tabela 5.37: Valores das iterações internas e das execuções onde ocorreram as melhores soluções do ILS-VND com limite de tempo de duas horas.

Nesta seção uma análise realizada sobre o Algoritmo ILS para o PCVG mostra que é possível obter soluções de boa qualidade em tempo computacional razoável quando se utiliza o Algoritmo ILS com o método de Descida em Vizinhança Variável (VND). As análises mostram ainda que o Algoritmo ILS na sua forma padrão leva a um processo de estagnação das soluções não permitindo melhoria ao longo das iterações. Para solucionar isto, se incorporam novas soluções construídas durante as etapas do Algoritmo ILS. Uma grande vantagem do ILS refere-se à possibilidade de se utilizar mecanismos de memória ao longo das iterações. Os melhores resultados desenvolvidos pelos algoritmos para problemas de otimização combinatória incorpora em sua estrutura, a memória, fato observado pelos experimentos reportados para o GRASP (G4 e G5). O Algoritmo ILS desenvolvido para o PCVG incorporou o mecanismo de memória no critério de aceitação em que este verificou a "história" das melhores soluções ocorridas ao longo do algoritmo. O critério de

aceitação utilizado pelo Algoritmo ILS ao PCVG utilizou até a etapa corrente do algoritmo o custo da melhor solução. Várias formas de implementações dos módulos presentes no Algoritmo ILS como perturbações e métodos de busca local foram analisados, assim como os parâmetros relacionados ao número de iterações e ao número de soluções novas necessárias para suprir estagnação. Neste sentido, para ter uma qualidade superior aos GRASP (G1, G2, G3, G4, G5 e G6') verifica-se que a combinação do método heurístico ILS com o método de busca local VND (ILS-VND) permitiu melhores resultados, ver (Tabelas 5.38, 5.39 e 5.40).

5.4.2 Resultados do Algoritmo ILS-VND

Nesta seção, serão mostrados os resultados computacionais do Algoritmo ILS-VND descrito na seção 5.4.1 para o PCVG, cujos parâmetros ajustados foram: total de soluções novas geradas iguais a 40 ($iterext$=40 - número máximo de iterações externas) e número máximo de iterações internas igual a 35 ($iter$=35). As descrições do processador, sistema operacional e linguagem de programação para os testes do Algoritmo ILS-VND e para as análises da seção 5.4.1 são idênticas às utilizadas pelas heurísticas GRASP na seção 5.3.

A Tabela 5.38 mostra os resultados obtidos para os testes realizados com o Algoritmo ILS-VND e o método exato CPLEX, restritos às instâncias menores sem limitação de tempo para o CPLEX. O Algoritmo ILS-VND foi executado 10 vezes.

Na primeira coluna da Tabela 5.38 encontram-se as instâncias, na segunda o valor obtido pelo CPLEX, na terceira, o tempo demandado pelo CPLEX em segundos t_{CPLEX}(s) e na quarta o gap_{hc} (equação 5.4) do melhor valor obtido por ILS-VND (%) em relação ao valor obtido pelo CPLEX. Na quinta coluna o tempo médio de execução do ILS-VND $t_{ILS-VND}$(s) medido em segundos. Na última coluna, é mostrada a perturbação utilizada pelo ILS-VND. A razão para executar a perturbação *Pert2* em algumas instâncias deve-se ao fato que estas não têm vértices suficientes nos grupos para realizar a perturbação *Pert3*, pois deve-se ter oito vértices no mínimo em um grupo para realizar esta perturbação. A *Pert2* foi realizada em 11 instâncias do total de 27. A necessidade do algoritmo realizar a *Pert2* torna-se importante mesmo em instâncias de grande porte contendo muitos grupos e não tendo vértices o suficiente em cada grupo para realizar a perturbação *Pert3*. Isto pode acontecer em situações práticas e reais.

Para todas as instâncias de pequeno porte (Tabela 5.38), o CPLEX encontrou soluções ótimas. ILS-VND encontrou soluções ótimas em 21 de um total de 27 instâncias e o gap_{hc} médio total dos valores obtidos pelo ILS-VND em relação ao ótimo foi de 0,10% com

tempo médio igual a 11,37s.

Instâncias	CPLEX	t_{CPLEX}(s)	ILS-VND(%)	$t_{ILS-VND}$(s)	perturbação
5-eil51	437	12,31	**0,00**	5,20	Pert3
10-eil51	440	74,38	**0,00**	4,40	Pert2
15-eil51	437	2,04	**0,00**	4,90	Pert2
5-berlin52	7991	201,80	**0,00**	2,90	Pert3
10-berlin52	7896	89,17	**0,00**	2,50	Pert3
15-berlin52	8049	75,93	**0,00**	4,30	Pert3
5-st70	695	13790,11	**0,00**	5,40	Pert3
10-st70	691	4581,00	**0,00**	6,60	Pert3
15-st70	692	883,50	**0,00**	5,40	Pert3
5-eil76	559	83,70	**0,00**	6,70	Pert3
10-eil76	561	254,30	0,36	5,50	Pert3
15-eil76	565	49,66	**0,00**	6,90	Pert3
5-pr76	108590	99,29	**0,00**	8,20	Pert3
10-pr76	109538	238,13	**0,00**	8,40	Pert3
15-pr76	110678	261,94	0,49	10,00	Pert3
10-rat99	1238	650,67	**0,00**	13,40	Pert3
25-rat99	1269	351,15	**0,00**	23,80	Pert2
50-rat99	1249	2797,58	**0,00**	23,40	Pert2
25-kroA100	21917	3513,57	**0,00**	18,40	Pert2
50-kroA100	21453	947,55	**0,00**	19,90	Pert2
10-kroB100	22440	4991,44	0,16	9,40	Pert3
50-kroB100	22355	2579,22	**0,00**	17,70	Pert2
25-eil101	663	709,45	0,45	21,20	Pert2
50-eil101	644	275,33	1,09	21,10	Pert2
25-lin105	14438	6224,55	**0,00**	8,10	Pert3
50-lin105	14379	1577,21	**0,00**	21,50	Pert2
75-lin105	14521	15886,77	0,15	21,90	Pert2
valores médios	-	2266,73	0,10	11,37	-

Tabela 5.38: Comparação do ILS-VND com o CPLEX para instâncias do *tipo 1* de pequeno porte.

Fazendo uma comparação entre os resultados obtidos pelo ILS-VND mostrados na Tabela 5.38 com os obtidos pelas heurísticas G4 e G5 mostrados na Tabela 5.28 verifica-se que o ILS-VND obteve os melhores resultados. O ILS-VND encontrou 21 soluções ótimas contra 18 do G5 e 15 do G4. Os gap_{hc}s médios foram 0,10%, 0,21% e 0,25% para ILS-VND, G5 e G4, respectivamente.

Novos testes para o ILS-VND utilizando um limite de tempo foram realizados com instâncias maiores para comparar com o CPLEX e com as diversas heurísticas GRASP (G1, G2, G3, G4, G5 e G6'). Foi colocado um tempo limite de 2 horas para o CPLEX e 720 segundos para as heurísticas GRASP e ILS-VND (pois as heurísticas são executadas

Id.	CPLEX	Melhores Valores						
		G1	G2	G3	G4	G5	G6'	ILS-VND
I_1	1,54	0,97	0,80	0,93	0,76	0,76	0,80	**0,74**
I_2	1,95	1,12	0,73	0,99	0,66	0,70	0,82	**0,56**
I_3	**0,01**	0,04	0,04	0,05	0,03	**0,01**	**0,01**	0,02
I_4	0,27	0,21	0,19	0,21	0,20	**0,14**	**0,14**	0,16
I_5	0,86	0,55	0,45	0,51	**0,43**	**0,43**	0,45	0,47
I_6	0,95	0,69	0,61	0,66	0,58	0,56	0,59	**0,51**
I_7	0,60	0,67	0,51	0,60	0,49	0,53	0,49	**0,24**
I_8	0,87	0,67	0,63	0,65	0,64	**0,43**	0,63	0,51
I_9	1,99	1,63	1,61	1,64	1,60	1,40	1,57	**1,32**
I_{10}	1,69	1,28	1,27	1,31	1,27	1,13	1,16	**0,89**
I_{11}	1,87	1,68	1,28	1,60	1,19	1,36	1,30	**0,90**
I_{12}	3,76	2,23	1,80	2,14	1,96	1,86	1,94	**1,06**
gap_{li} médio	1,36	0,98	0,83	0,94	0,82	0,78	0,83	**0,61**

Tabela 5.39: Gap_{li} do CPLEX comparados aos melhores valores das heurísticas com limite de tempo.

10 vezes). Somente a execução da instância I_3 não foi limitada por duas horas, pois a solução ótima foi encontrada antes pelo CPLEX. A Tabela 5.39 mostra os *gaps* dos melhores valores alcançados pelas heurísticas e pelo CPLEX. Os valores em negrito mostram a melhor solução alcançada. Na primeira coluna da Tabela 5.39 encontram-se os identificadores das instâncias descritos na Tabela 5.14, na segunda o gap_{li} (%) do CPLEX e nas sete últimas colunas os *gaps* dos melhores valores de cada heurística, calculado pela equação 5.3 e expresso por gap_{li}. ***O ILS-VND alcançou o maior número de melhores soluções, oito no total de 12 com gap médio de 0,61%.***

Na Tabela 5.40 são mostrados os valores médios alcançados pelas heurísticas com critério limite de tempo. Na primeira coluna encontram-se os identificadores das instâncias (idênticos aos da Tabela 5.39) e nas sete últimas colunas, o gap_{li} de cada instância obtida nos valores médios das heurísticas. Os valores em negrito mostram onde o desempenho médio foi melhor. O ILS-VND obteve o melhor desempenho médio para todas as instâncias, exceto para instância I_8. Os valores médios do ILS-VND obtiveram ***gap médio de 0,75%***.

Na Tabela 5.40 observa-se que em algumas instâncias os *gaps* dos valores médios do ILS-VND superam os *gaps* dos melhores valores de algumas das heurísticas GRASP (Tabela 5.23). Por exemplo, na instância I_1 o *gap* do valor médio do ILS-VND é igual **0,82%**, enquanto que o *gap* do melhor valor de G1 é igual 0,97% e de G3 foi de 0,93%. Na instância I_2, o *gap* do ILS-VND é igual a **0,79%** e os *gaps* de G1 e G3 são 1,12% e

0,99%, respectivamente.

Id.	Valores Médios						
	G1	G2	G3	G4	G5	G6'	ILS-VND
I_1	1,78	1,18	1,21	1,18	1,09	1,35	**0,82**
I_2	1,93	1,24	1,36	1,22	1,09	1,41	**0,79**
I_3	0,35	0,22	0,14	0,18	0,06	0,32	**0,04**
I_4	0,28	0,23	0,25	0,24	0,17	0,19	**0,16**
I_5	0,85	0,64	0,63	0,63	0,64	0,78	**0,54**
I_6	0,92	0,73	0,78	0,73	0,75	0,79	**0,59**
I_7	1,09	0,82	0,78	0,78	0,83	1,00	**0,35**
I_8	0,82	0,71	0,75	0,72	**0,50**	0,73	0,55
I_9	1,88	1,76	1,79	1,76	1,61	1,75	**1,54**
I_{10}	1,54	1,44	1,47	1,42	1,26	1,46	**1,07**
I_{11}	3,30	2,37	2,01	2,30	2,52	3,21	**1,17**
I_{12}	3,32	2,43	2,70	2,54	2,34	2,62	**1,31**
gap_{li} médio	1,51	1,15	1,16	1,14	1,07	1,30	**0,75**

Tabela 5.40: Gap_{li} dos valores médios das heurísticas com limite de tempo.

Uma comparação entre ILS-VND-IMPP (ILS-VND) e G5-IMPP foi realizada estabelecendo um alvo a ser atingido por ambas heurísticas. A estratégia utilizada executou 10 vezes cada instância, alvo e heurística considerando um tempo limite. Como as duas heurísticas foram comparadas nas Tabelas 5.39 e 5.40 num tempo de duas horas (720 segundos a cada execução), foi estabelecido o tempo limite de quatro horas (1440 segundos a cada execução) para a heurística atingir o alvo.

O tempo limite para a instância I_3 foi considerado de 890 segundos (89s a cada execução), porque que o CPLEX encontrou a solução ótima em menos de duas horas e a comparação entre ILS-VND-IMPP e G5-IMPP estabelecido nas Tabelas 5.39 e 5.40 foi realizado num tempo total de 445 segundos para as 10 execuções. O alvo a ser atingido foi o melhor valor encontrado entre as duas heurísticas e estão mostrados na Tabela 5.41.

A Figura 5.19 mostra os *gaps* de ILS-VND-IMPP e G5-IMPP das melhores soluções encontradas estabelecendo o alvo e o tempo limite. No eixo x são mostrados os identificadores das instâncias da Tabela 5.39 e no eixo y os *gaps* de ambas as heurísticas. Valores abaixo e inclusive sobre a linha tracejada com descrição **"alvo"** mostrados na Figura 5.19 significam que as heurísticas atingiram o alvo. Observa-se que G5-IMPP atingiu quatro alvos e ILS-VND-IMPP atingiu oito alvos.

A Tabela 5.41 mostra o tempo atingido para encontrar o alvo para cada instância utilizando as heurísticas ILS-VND-IMPP e G5-IMPP. Os valores em negrito mostram que as heurísticas alcançaram o alvo antes do tempo limite. Na primeira coluna da Tabela 5.41

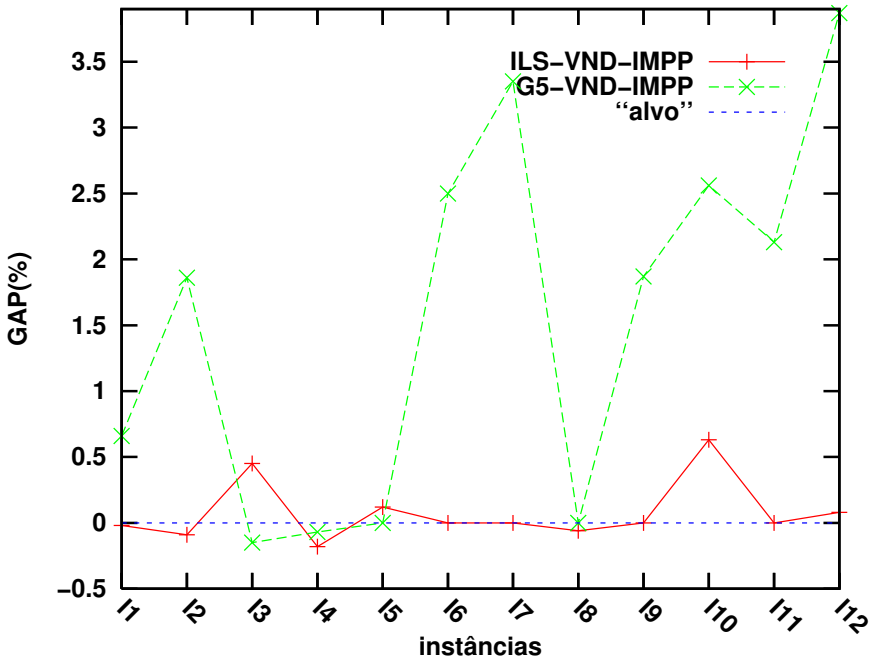


Figura 5.19: *Gaps* de ILS-VND-IMPP e G5-IMPP alcançados para encontrar o alvo.

é mostrado os identificadores das instâncias, na segunda o melhor valor encontrado entre as duas heurísticas (Tabela 5.39) e considerado como alvo a ser alcançado, na terceira a heurística que encontrou o melhor valor e nas últimas duas colunas os tempos alcançados pelas heurísticas expressos em segundos para atingirem os alvos.

Id.	Valor	Heurística	Tempos (s)	
			ILS-VND-IMPP	G5-IMPP
I_1	43709	ILS-VND-IMPP	**964**	14400
I_2	55388	ILS-VND-IMPP	**1501**	14400
I_3	672	G5-IMPP	890	**212**
I_4	10673	G5-IMPP	**858**	**6424**
I_5	13737	G5-IMPP	14400	**3039**
I_6	16461	ILS-VND-IMPP	**2805**	14400
I_7	777	ILS-VND-IMPP	**3676**	14400
I_8	1723	G5-IMPP	**427**	**3557**
I_9	12704785	ILS-VND-IMPP	**8801**	14400
I_{10}	12530603	ILS-VND-IMPP	14400	14400
I_{11}	10763	ILS-VND-IMPP	**139**	14400
I_{12}	19260	ILS-VND-IMPP	14400	14400

Tabela 5.41: Tempos alcançados por ILS-VND-IMPP e G5-IMPP para encontrar o alvo.

Como podemos observar pela Figura 5.19 e Tabela 5.41 o ILS-VND-IMPP encontra mais alvos do que o G5-IMPP e quando ambos encontram o alvo, ILS-VND-IMPP atinge num tempo menor, (ver as duas situações nas instâncias I_4 e I_8 da Tabela 5.41). Con-

siderando as Tabelas 5.39, 5.40 e 5.41 e a Figura 5.19 conclui-se que ILS-VND-IMPP apresenta melhor desempenho do que G5-IMPP.

Instâncias	CPLEX	t_{CPLEX}(s)	ILS-VND-nP(%)	$t_{ILS-VND-nP}$(s)
5-eil51	437	12,31	**0,00**	3,80
10-eil51	440	74,38	**0,00**	5,50
15-eil51	437	2,04	**0,00**	5,00
5-berlin52	7991	201,80	0,11	4,50
10-berlin52	7896	89,17	0,52	5,10
15-berlin52	8049	75,93	**0,00**	4,00
5-st70	695	13790,11	**0,00**	6,90
10-st70	691	4581,00	0,43	3,70
15-st70	692	883,50	0,87	6,40
5-eil76	559	83,70	**0,00**	5,10
10-eil76	561	254,30	1,07	8,60
15-eil76	565	49,66	0,88	6,80
5-pr76	108590	99,29	**0,00**	6,30
10-pr76	109538	238,13	**0,00**	8,30
15-pr76	110678	261,94	**0,00**	9,70
10-rat99	1238	650,67	**0,00**	12,60
25-rat99	1269	351,15	**0,00**	21,40
50-rat99	1249	2797,58	**0,00**	18,60
25-kroA100	21917	3513,57	**0,00**	21,70
50-kroA100	21453	947,55	**0,00**	21,70
10-kroB100	22440	4991,44	0,91	12,40
50-kroB100	22355	2579,22	**0,00**	22,40
25-eil101	663	709,45	1,06	21,10
50-eil101	644	275,33	0,16	20,80
25-lin105	14438	6224,55	1,38	16,70
50-lin105	14379	1577,21	**0,00**	22,60
75-lin105	14521	15886,77	**0,00**	23,70
valores médios =	-	2266,73	0,27	12,05

Tabela 5.42: Comparação do ILS-VND-nP com o CPLEX para instâncias do *tipo 1* de pequeno porte.

Em todos os resultados apresentados pela heurística ILS, as soluções foram construídas através do IMPP. O construtivo IMPP penaliza as arestas intergrupos e resolve o PCVG transformando-o em um PCV. Uma outra forma de resolver o PCVG é selecionar primeiro todos os vértices de cada grupo para construir soluções parciais e em seguida escolher os vértices do próximo grupo até que todos os grupos sejam selecionados. Neste sentido, as soluções do PCVG serão construídas sem penalizar as arestas intergrupos. Para este propósito, utiliza-se o construtivo IMPnP para comparação com o IMPP, porque o método utilizado para construir soluções parciais (inserção mais próxima) são semelhantes. O

construtivo IMPnP foi inserido no ILS-VND, visto que o desempenho do ILS-VND foi melhor do que outras heurísticas (G1, G2, G3, G4, G5 e G6'). Os tipos de instâncias para realizar a comparação estão nas Tabelas 5.29 e 5.38.

Na Tabela 5.42 são mostrados os valores obtidos pelo Algoritmo ILS-VND utilizando o algoritmo construtivo baseado na Inserção mais Próxima com as arestas inter-grupos não Penalizadas (IMPnP), denominado de ILS-VND-nP. As descrições na Tabela 5.42 são idênticas a da Tabela 5.38, exceto que aparece na quarta coluna o gap_{hc} (equação 5.4) do melhor valor obtido por ILS-VND-nP(%) em relação ao valor obtido pelo CPLEX e na quinta coluna o tempo médio de execução $t_{ILS-VND-nP}$(s), medido em segundos. O tipo de perturbação utilizada em cada instância é idêntica a da Tabela 5.38. O CPLEX encontrou soluções ótimas em todas as 27 instâncias e o ILS-VND-nP conseguiu 17 soluções ótimas. O gap_{hc} médio total dos valores obtidos pelo ILS-VND-nP em relação ao ótimo foi de 0,27% com tempo médio igual a 12,05s.

Observa-se que o tempo médio de 12,05s do ILS-VND-nP na Tabela 5.42 foi um pouco maior do que o tempo médio de 11,37s do ILS-VND-P (ILS-VND com o algoritmo construtivo IMPP - Tabela 5.38). Isto ocorre porque em IMPnP, a Lista de Candidatos que auxilia a elaboração da solução parcial considera em cada etapa todos os vértices pertencentes ao grupo dos vértices contidos na solução parcial. Em oposição, IMPP considera somente os k vértices mais próximos pertencentes ou não ao mesmo grupo dos vértices da solução parcial já construída.

Para as instâncias da Tabela 5.42, o gap_{hc} médio de 0,27% do ILS-VND-nP foi maior em relação ao do ***ILS-VND-P com 0,11%***. Além disso, o gap_{hc} médio do ILS-VND-nP ultrapassou o gap_{hc} médio do G4 com 0,25% e do G5 com 0,21% (Tabela 5.28).

A Tabela 5.43 mostra a comparação entre os melhores valores do ILS-VND-P e do ILS-VND-nP com limite de tempo para instâncias da Tabela 5.29. Foi estabelecido o tempo limite de duas horas (720s a cada execução). Na primeira coluna da Tabela 5.43 apresentam-se os identificadores das instâncias descritos na Tabela 5.29, na segunda e terceira colunas os *gaps* do ILS-VND-P e do ILS-VND-nP, respectivamente. O ILS-VND-P alcançou onze melhores soluções, enquanto que o ILS-VND-nP alcançou treze melhores soluções. O CPLEX alcançou uma única solução melhor. O gap_{li} médio do ***ILS-VND-P foi igual a 0,65%***, do ILS-VND-nP ficou em 0,67% e do CPLEX em 1,36%.

Na Tabela 5.44, os valores dos gap_{li}s dos valores médios do ILS-VND-P são comparados aos valores do ILS-VND-nP. A descrição das colunas na Tabela 5.44 é semelhante a da Tabela 5.43. O gap_{li} médio dos valores médios do ***ILS-VND-P alcançou o valor***

Id.	Melhores Valores	
	ILS-VND-P	ILS-VND-nP
J_1	0,11	**0,10**
J_2	**0,73**	**0,73**
J_3	0,56	**0,46**
J_4	**1,30**	1,45
J_5	**0,88**	0,99
J_6	1,29	**1,24**
J_7	0,24	**0,21**
J_8	**0,33**	**0,33**
J_9	0,51	**0,43**
J_{10}	**0,89**	**0,89**
J_{11}	**0,55**	1,13
J_{12}	**1,05**	1,17
J_{13}	1,43	**1,04**
J_{14}	1,03	**1,01**
J_{15}	**0,46**	0,52
J_{16}	**0,80**	0,86
J_{17}	0,51	**0,47**
J_{18}	0,02	0,04
J_{19}	**0,12**	**0,12**
J_{20}	**0,16**	**0,16**
gap_{li} médio	**0,65**	0,67

Tabela 5.43: Gap_{li} dos melhores valores do ILS-VND-P comparado ao ILS-VND-nP.

de 0,80% e do ILS-VND-nP foi igual a 0,86%.

Pelos resultados apresentados nas Tabelas 5.42, 5.43 e 5.44 constata-se que o algoritmo construtivo IMPP quando utilizado para construir soluções para o Algoritmo ILS-VND permite alcançar resultados melhores, mas bem próximos do algoritmo construtivo IMPnP. Neste sentido, verifica-se que ao tratar o PCVG transformando-o em PCV é uma alternativa eficaz para conseguir bons resultados. Cabe ressaltar que várias instâncias do *tipo 1* e as instâncias do *tipo 3* alcançaram melhores resultados utilizando o construtivo IMPnP. Nas instâncias do *tipo 1* e do *tipo 3* os vértices estão agrupados bem próximos uns dos outros e relativamente distantes dos vértices de grupos distintos. Isto facilita a construir soluções parciais de qualidade em torno dos vértices de cada grupo, o que torna o IMPnP mais eficaz para estes tipos de instâncias.

Id.	Valores Médios	
	ILS-VND-P	ILS-VND-nP
J_1	**0,15**	**0,15**
J_2	**0,82**	0,83
J_3	0,79	**0,65**
J_4	**1,52**	1,81
J_5	**1,06**	1,28
J_6	1,46	**1,41**
J_7	0,35	**0,31**
J_8	0,52	**0,46**
J_9	0,55	**0,51**
J_{10}	**1,16**	1,29
J_{11}	**0,92**	1,85
J_{12}	**1,30**	1,56
J_{13}	1,79	**1,38**
J_{14}	1,23	**1,18**
J_{15}	**0,54**	0,59
J_{16}	1,02	**0,98**
J_{17}	0,59	**0,54**
J_{18}	**0,04**	0,09
J_{19}	**0,13**	0,21
J_{20}	**0,16**	0,18
gap_{li} médio	**0,80**	0,86

Tabela 5.44: Gap_{li} dos valores médios do ILS-VND-P comparado ao ILS-VND-nP.

5.4.3 Considerações Finais do ILS para o PCVG

Vários testes com diferentes tipos de perturbações, métodos de busca local, critérios de aceitação e formas diferentes de construir soluções foram realizados. Os diversos ILS propostos podem ser considerados como algoritmos híbridos pois foram incorporados componentes de outras metaheurísticas tais como o GRASP através de sua fase construtiva e do VNS através do método de descida em vizinhança variável. Observou-se que as formas híbridas permitem melhor desempenho comparado às formas canônicas de cada metaheurística. Destacam-se como formas híbridas neste trabalho: a heurística denominada de G4 (GRASP utilizando reconexão de caminhos RC1 e RC2), G5 (GRASP com reconexão de caminhos RC3 e método VND) e ILS-VND-IMPP (fase de construção do GRASP utilizando IMPP e método VND). Existem ainda as formas híbridas G6 (GRASP com reconexão de caminhos RC4 e método VND) e ILS-VND-IMPnP (fase de construção do GRASP utilizando IMPnP e método VND).

Na implementação do ILS-VND-IMPP, os resultados obtidos foram de qualidade superior aos obtidos pelos algoritmos GRASP (G1, G2, G3, G4, G5 e G6'). Na comparação

entre ILS-VND-IMPP e ILS-VND-IMPnP, este último conseguiu número maior de soluções melhores. Entretanto na comparação entre os melhores valores e valores médios, o ILS-VND-IMPP se sobressaiu ao ILS-VND-IMPnP tanto para instâncias de porte maior quanto para instâncias de porte menor, onde se conhecem os valores ótimos destas últimas instâncias. O ILS-VND-IMPP quando comparado ao melhor GRASP (G5-IMPP) mostra desempenho superior para os melhores valores alcançados e valores médios, além de permitir atingir um número maior de alvos em tempos menores, quando o critério é atingir soluções alvos. O bom desempenho do ILS-VND-IMPP acontece devido o balanceamento entre a diversificação, dada por várias soluções novas construídas e a intensificação, realizada pelo método de busca local. O uso de memória do módulo referente ao critério de aceitação que armazena sempre a melhor solução encontrada até a etapa corrente do algoritmo foi fundamental para o sucesso do ILS-VND-IMPP.

5.5 Comparações entre as Heurísticas Propostas com um Algoritmo Genético

As heurísticas propostas foram comparadas com Algoritmo Genético (AG) encontrado na literatura desenvolvido pelos autores Ding, Cheng e He (2007) que resolve o PCVG sem a pré-fixação da sequência dos grupos visitados. O AG foi implementado neste trabalho pois não foi possível acesso ao código original e seguiu todas as sugestões de parâmetros dado pelos autores (CHENG; DING, 2010).

O AG é denominado de (*Two-Level Genetic Algorithm - TLGA*) (DING; CHENG; HE, 2007) e constrói soluções para o PCVG em dois níveis, um denominado de nível baixo e outro de nível alto. No nível baixo um método evolutivo utilizando operadores de cruzamento e mutação produz um ciclo T_k, k=1,...,m, em cada grupo utilizando uma população de p_b cromossomos, denominado de `Lower_Level(`T_k`,`p_b`)`. No nível baixo os operadores de cruzamento e mutação são aplicados somentes aos vértices e não operam sobre os grupos. No nível alto uma sequência dos grupos (V_1, V_2, ..., V_m) é gerada aleatoriamente. Em seguida, um caminho P_k sobre o ciclo T_k para cada grupo é encontrado quebrando o ciclo T_k através da escolha de dois vértices aleatórios consecutivos. Um vértice *start* v_{s_k} representa o início do caminho no grupo k e um outro *end* v_{e_k} o fim. O ciclo hamiltoniano C para o PCVG é encontrado designando cada caminho P_k ao grupo correspondente V_k, iniciando o caminho em cada grupo pelo vértice v_{s_k} e finalizando no vértice v_{e_k}. O algoritmo TLGA gera uma população inicial de p_a cromossomos nos quais

cada um representa um ciclo C para o PCVG.

Algoritmo 26: Algoritmo Genético (TLGA), adaptado do fluxograma de (DING; CHENG; HE, 2007)

```
1: Lower_Level(T_k, p_b);
2: Populacao_Inicial;
3: S* ← S_0;
4: iter=1;
5: miter=1;
6: enquanto (iter − miter) ≤ maxger faça
7:    Avaliar na população P={1, 2, ..., p_a} a aptidão de cada cromossomo;
8:    Selecionar um par de pai (S_Pi,S_Pj) em P para a próxima geração;
9:    se random(1,0) < CF então
10:      S_Fi ← Crossover(S_Pi, S_Pj)
11:      S_Fj ← Crossover(S_Pj, S_Pi)
12:      se S_Fi for melhor que S_Fj então
13:         S_AG ← S_Fi;
14:      senão
15:         S_AG ← S_Fj;
16:      fim se
17:   fim se
18:   se random(1,0) < MF_1 então
19:      S_AG ← Mutacao_Grupos(S_Fi, S_Fj)
20:   fim se
21:   se random(1,0) < MF_21 então
22:      para k de 1 até numgrupos faça
23:         se random(1,0) < MF_22 então
24:            S_AG ← Mutacao_Genes(k)
25:         fim se
26:      fim para
27:   fim se
28:   Atualizar a População P com a solução S_AG;
29:   se S_AG for melhor que S* então
30:      S* ← S_AG;
31:      miter = iter;
32:   fim se
33: fim enquanto
34: Retornar S*;
```

Em cada geração do TLGA, todos os ciclos C são avaliados por uma função de aptidão. Dois ciclos C (par de pais) são escolhidos por uma *escala de aptidão* e são processados por operadores genéticos de cruzamento e de mutação. A população é atualizada pelo ciclo hamiltoniano final executado pelas fases do TLGA (cruzamento e mutação) e uma nova geração do algoritmo é reiniciada. O TLGA termina se após um número máximo consecutivos de gerações ($maxger$), a melhor solução da população não é atualizada.

O Algoritmo 26 mostra o Algoritmo Genético TLGA, adaptado do fluxograma de (DING; CHENG; HE, 2007). O procedimento `Populacao_Inicial` (geração 0) constrói uma população inicial de p_a cromossomos. O parâmetro *iter* representa o número da geração, o *miter* recebe o valor de *iter* sempre que o algoritmo TLGA encontrar uma solução melhor do que a atual, P=$\{1,\ 2,\ ...,\ p_a\}$ representa a população com p_a cromossomos e *numgrupos* representa o total de grupos. A função `random(1,0)` gera um número aleatório no intervalo [0,1]. A melhor solução é representada por S^*, a solução S_{AG} significa a solução produzida numa geração do algoritmo TLGA depois do cruzamento e mutações e S_0 a solução inicial.

Cada par de pais (S_{Pi},S_{Pj}), com $i \neq j$, representam os genitores escolhidos aleatoriamente conforme uma *escala de aptidão*, que realiza primeiro o cruzamento seguido do procedimento de mutação nos filhos (S_{Fi},S_{Fj}), ambos aplicados somente aos grupos. Após esta etapa, é realizado a mutação nos genes de cada grupo no ciclo hamiltoniano do PCVG. Os procedimentos `Crossover(`S_{Pi}`,`S_{Pj}`)` e `Mutacao_Grupos(`S_{Fi}`,`S_{Fj}`)` operam sobre as sequências dos grupos, modificando o cromossomo, sem alterar a sequência dos vértices em cada grupo. Os genes em cada grupo k sofrem uma mutação `Mutacao_Genes(`k`)` alterando seus caminhos e modificando somente a sequência dos genes, sem alterar a sequência dos grupos.

Os dados a seguir, para implementar o algoritmo TLGA (AG_I), foram obtidos em (CHENG; DING, 2010). O tamanho da população p depende do número de vértices da instâncias e varia tanto no nível mais baixo p_b, como no nível mais alto p_a. Para 442 vértices foi designado p_b=50 e p_a=80, para 783 vértices p_b=60 e p_a=90 e para 1000 vértices, p_b=70 e p_a=100. A *escala de aptidão* utilizada foi o método de amostragem estocástica uniforme (*stocastic universal sampling*). Os valores utilizados pelos operadores de cruzamento e mutação são: CF=0,76, MF_1=0,06, MF_{21}=0,13 e MF_{22}=0,5. O valor de *maxger* também depende do tamanho das instâncias e foi estabelecido em 100, 110 e 120, para o número de vértices 442, 783 e 1000, respectivamente. A atualização da população é realizada pelos melhores cromossomos. O operador utilizado para realizar o cruzamento e a mutação nos grupos foi *order crossover*. Os autores Cheng e Ding (2010) mencionaram que utilizaram também o operador *partial mapped crossover* para o TLGA, mas que não perceberam nenhuma diferença no desempenho do algoritmo.

Os resultados computacionais, apresentados a seguir, comparando o TLGA com as heurísticas desenvolvidas foram obtidos com o mesmo *software* e *hardware* descritos anteriormente na seção 5.3, (seção Heurísticas GRASP).

Instâncias	TLGA (%)	t_{TLGA}(s)	G5-IMPP(%)	$t_{G5-IMPP}$(s)	ILS-VND-P(%)	$t_{ILS-VND-P}$(s)
5-eil51	8,47	**0,4**	**0,00**	1,50	**0,00**	5,20
10-eil51	2,73	**0,4**	**0,00**	1,30	**0,00**	4,40
15-eil51	7,78	**0,4**	**0,00**	1,40	**0,00**	4,90
5-berlin52	1,44	**0,6**	**0,00**	1,80	**0,00**	2,90
10-berlin52	10,18	**0,4**	**0,00**	1,60	**0,00**	2,50
15-berlin52	14,69	**0,4**	**0,00**	1,60	**0,00**	4,30
5-st70	0,86	**1,6**	**0,00**	3,40	**0,00**	5,40
10-st70	1,88	**1,0**	**0,00**	2,50	**0,00**	6,60
15-st70	7,23	**0,8**	**0,00**	2,90	**0,00**	5,40
5-eil76	3,94	**1,8**	0,54	3,80	**0,00**	6,70
10-eil76	9,45	**1,2**	0,71	3,20	**0,36**	5,50
15-eil76	2,48	**1,2**	0,53	3,60	**0,00**	6,90
5-pr76	1,78	**2,0**	**0,00**	5,50	**0,00**	8,20
10-pr76	1,02	**1,2**	**0,00**	4,00	**0,00**	8,40
15-pr76	5,95	**1,2**	**0,15**	4,40	0,49	10,00
10-rat99	6,70	**3,4**	**0,00**	9,70	**0,00**	13,40
25-rat99	23,48	**2,4**	**0,00**	9,80	**0,00**	23,80
50-rat99	37,15	**2,4**	0,80	10,50	**0,00**	23,40
25-kroA100	5,69	**2,2**	**0,00**	8,60	**0,00**	18,40
50-kroA100	22,23	**2,8**	**0,00**	9,60	**0,00**	19,90
10-kroB100	2,49	**3,8**	**0,00**	9,30	0,16	9,40
50-kroB100	25,36	**2,2**	0,54	9,50	**0,00**	17,70
25-eil101	6,33	**2,2**	1,21	7,80	**0,45**	21,20
50-eil101	20,34	**2,2**	**1,09**	9,10	**1,09**	21,10
25-lin105	19,39	**2,0**	**0,00**	10,50	**0,00**	8,10
50-lin105	26,29	**3,2**	**0,00**	12,40	**0,00**	21,50
75-lin105	56,62	**4,6**	0,23	14,10	**0,15**	21,90
valores médios	12,29	**1,8**	0,21	6,05	**0,10**	11,37

Tabela 5.45: Comparação do TLGA com G5-IMPP e ILS-VND-P utilizando como critério de parada as iterações das heurísticas.

Os primeiros testes foram efetuados entre G5-IMPP, ILS-VND-P e TLGA, restritos às instâncias menores, mostrados na Tabela 5.45. Nestes testes foram utilizados como critério de parada o número de iterações das heurísticas. Na heurística G5-IMPP o número de iterações foi igual a 200, no ILS-VND-P para o número de iterações internas *iter* foi igual a 35 e o parâmetro *iterext* (número de iterações externas) igual a 40 e no TLGA: o número máximo consecutivos de gerações, onde a melhor solução da população não é atualizada (*maxger*) foi igual 10 com p_b=5 e p_a=10. O número de execuções para cada instância foi de 10 para G5-IMPP e ILS-VND-P e cinco para TLGA (valor utilizado pelos autores Ding, Cheng e He (2007)).

Na primeira coluna da Tabela 5.45 são mostradas as instâncias, na segunda coluna

encontra-se o gap_{hc} do melhor valor obtido por TLGA(%). Na terceira coluna o tempo médio demandado pelo TLGA em segundos t_{TLGA}, na quarta o gap_{hc} do melhor valor obtido por G5-IMPP(%), na quinta o tempo médio de execução $t_{G5-IMPP}$(s) de G5-IMPP medido em segundos. Na sexta e sétima colunas são mostradas o gap_{hc} ILS-VND-P (%) e o tempo médio de execução $t_{ILS-VND-P}$(s) do ILS-VND-P. O percentual do gap_{hc} é calculado de acordo a equação 5.4, onde os valores das soluções ótimas são apresentados na Tabela 5.38.

Observa-se que na Tabela 5.45 o TLGA produz soluções de baixa qualidade quando comparado às heurísticas G5-IMPP e ILS-VND-P. O gap_{hc} médio das soluções produzidas pelo TLGA ficou em 12,29%, enquanto G5-IMPP alcançou 0,21% e ***ILS-VND-P com 0,10%***. O TLGA nunca alcançou uma solução ótima. O tempo médio total avaliado sobre todas as instâncias do TLGA foi melhor do que G5-IMPP e ILS-VND-P alcançando ***1,8s***, G5-IMPP ficou em 6,05s e ILS-VND-P com 11,37s. Mesmo que o tempo médio do TLGA seja menor, a opção pela heurística G5-IMPP torna-se mais promissora, porque aglutina pequeno esforço computacional com soluções de alta qualidade.

Nos resultados apresentados até o momento, onde o critério de parada é o número de iterações, observa-se que G5-IMPP e ILS-VND-P obtiveram os melhores resultados, mas exigiram tempos computacionais maiores que TLGA, mesmo para instâncias de pequeno porte. Novos testes computacionais para instâncias maiores foram executados. Para estas instâncias, os parâmetros do TLGA são: se o número de vértices for menor ou igual a 105, utiliza-se p_a=10, se o número de vértices for maior do que 105 e menor do que 442, utiliza-se p_a=80 e para instâncias com vértices maior do que 442 e menor do que 783 vértices, usa-se p_a=90. A *escala de aptidão*, os valores utilizados pelos operadores de cruzamento CF e mutação (MF_1, MF_{21} e MF_{22}) e demais parâmetros padrões são descritos por (CHENG; DING, 2010).

O critério de parada estabelecido para as três heurísticas neste novo teste foi definido por um tempo máximo de 720 segundos a cada execução, porque, para cada instância, as heurísticas são executadas 10 vezes. Mesmo para o TLGA que no algoritmo original executa cinco vezes, nessa nova bateria de testes, o algoritmo foi executado 10 vezes. A melhor solução encontrada e a solução média nestas execuções são retornadas. Somente a execução da instância J_{18} não foi limitada por duas horas, devido à solução ótima ser encontrada antes pelo CPLEX. Os resultados foram comparados com os limites inferiores encontrados pelo CPLEX descrito na Tabela 5.29. O CPLEX foi executado no tempo limite de duas horas.

Id.	Melhores Valores		
	TLGA (%)	G5-IMPP(%)	ILS-VND-P(%)
J_1	0,12	**0,07**	0,11
J_2	1,08	0,76	**0,73**
J_3	9,07	0,70	**0,56**
J_4	26,32	1,38	**1,30**
J_5	25,84	1,12	**0,88**
J_6	23,61	1,48	**1,29**
J_7	0,81	0,52	**0,24**
J_8	4,71	0,82	**0,33**
J_9	6,99	**0,42**	0,51
J_{10}	1,72	1,35	**0,89**
J_{11}	1,46	1,04	**0,55**
J_{12}	33,71	1,83	**1,05**
J_{13}	33,34	1,83	**1,43**
J_{14}	1,61	1,47	**1,03**
J_{15}	0,56	**0,43**	0,46
J_{16}	7,93	0,95	**0,80**
J_{17}	4,39	0,56	**0,51**
J_{18}	0,05	**0,01**	0,02
J_{19}	0,14	0,17	**0,12**
J_{20}	2,70	**0,14**	0,16
gap_{li} médio	9,31	0,85	**0,65**

Tabela 5.46: Gap_{li} dos melhores valores de TLGA, G5-IMPP e ILS-VND-P com limite de tempo.

Os melhores valores obtidos pelas heurísticas com o critério de parada limitado pelo tempo são mostrados na Tabela 5.46, onde na primeira coluna apresentam-se os identificadores das instâncias. Na segunda coluna o gap_{li} alcançado pelo TLGA, onde este *gap* é calculado pela equação 5.3. Na terceira coluna o gap_{li} alcançado por G5-IMPP e na última coluna o gap_{li} de ILS-VND-P.

Na Tabela 5.46, observa-se que mesmo estabelecendo tempos limites iguais para as três heurísticas, o algoritmo TLGA não consegue melhorar a qualidade da solução para o PCVG. O TLGA continua produzindo soluções de baixa qualidade comparado às heurísticas G5-IMPP e ILS-VND-IMPP, onde o gap_{li} médio ficou em 9,31%, enquanto G5-IMPP alcançou 0,85% e ***ILS-VND-P com 0,65%***. Para algumas instâncias, o algoritmo TLGA obteve gap_{li} próximos de G5-IMPP e ILS-VND-P. Por exemplo, para a instância J_{18} o gap_{li} de TLGA foi de 0,05%, enquanto G5-IMPP e ILS-VND-P obtiveram 0,01% e 0,02%, respectivamente. Para as instâncias J_1, J_{15} e J_{19}, o gap_{li} de TLGA alcançou 0,12%, 0,56% e **0,14%**, respectivamente, enquanto que G5-IMPP obteve para J_1 o valor de 0,07%, para J_{15} igual a 0,43% e para J_{19} o valor 0,17%. O valor em negrito para a instância J_{19} mos-

Id.	Valores Médios		
	TLGA (%)	G5-IMPP(%)	ILS-VND-P(%)
J_1	0,32	0,28	**0,15**
J_2	6,27	1,08	**0,82**
J_3	16,86	1,08	**0,79**
J_4	27,26	1,58	**1,52**
J_5	26,85	1,24	**1,06**
J_6	24,72	1,82	**1,46**
J_7	4,72	0,83	**0,35**
J_8	15,01	1,39	**0,52**
J_9	8,22	**0,50**	0,55
J_{10}	4,34	2,46	**1,16**
J_{11}	4,01	2,97	**0,92**
J_{12}	38,66	2,29	**1,30**
J_{13}	38,57	2,81	**1,79**
J_{14}	6,60	2,18	**1,23**
J_{15}	1,84	0,64	**0,54**
J_{16}	11,15	1,33	**1,02**
J_{17}	6,08	0,74	**0,59**
J_{18}	0,33	0,06	**0,04**
J_{19}	0,82	0,45	**0,13**
J_{20}	3,28	0,17	**0,16**
gap_{li} médio	12,30	1,30	**0,80**

Tabela 5.47: Gap_{li} dos valores médios de TLGA, G5-IMPP e ILS-VND-P com limite de tempo.

tra que TLGA foi melhor do que G5-IMPP. Para ILS-VND-P o gap_{li} da instância J_1 foi igual a 0,11%, para J_{15} igual a 0,46% e J_{19} em 0,12%. As melhores soluções sempre são encontradas por G5-IMPP ou ILS-VND-P. Com o critério de parada limitado pelo tempo, a opção pela heurística ILS-VND-P torna-se promissora quando comparada ao TLGA.

Os valores médios obtidos pelas heurísticas com o critério de parada limitado pelo tempo são mostrados na Tabela 5.47. A descrição da Tabela 5.47 é semelhante a da Tabela 5.46. O gap_{li} médio dos valores médios de TLGA ficou em 12,30%, enquanto G5-IMPP alcançou 1,30% e ***ILS-VND-P com 0,80%***. Para todas as instâncias da Tabela 5.47, o gap_{li} do algoritmo TLGA foi sempre maior do que G5-IMPP ou ILS-VND-P. Em algumas instâncias os gap_{li}s dos valores médios dos custos das soluções encontradas pelo algoritmo TLGA são menores do que 1%. Por exemplo, o TLGA obteve gap_{li} dos valores médios para as instâncias J_1, J_{18} e J_{19}, iguais a 0,32%, 0,33% e 0,82%, respectivamente,

5.5.1 Análises entre as Heurísticas Propostas e o TLGA

Em (DING; CHENG; HE, 2007), para avaliar o TLGA foram realizados testes agrupando as instâncias do PCV (TSPLIB95, 2007) através de clusterização. A avaliação do melhor custo do ciclo de cada solução produzida pelo TLGA (Tabela 3 de (DING; CHENG; HE, 2007)) foi calculada em relação ao custo ótimo da instância do PCV correspondente e obteve a taxa média de 1,30.

Devido que os autores Ding, Cheng e He (2007) não possibilitaram o envio das instâncias originais utilizadas, foram geradas instâncias clusterizadas conforme o padrão descrito em seus trabalhos. Estas instâncias foram mostradas nas Tabelas 5.45 e 5.46. Considerando estas instâncias clusterizadas geradas, a taxa média dos custos das melhores soluções avaliadas pela implementação do TLGA ficou em 1,18. Para as instâncias da Tabela 5.45 a taxa média ficou em 1,15. Ao analisar a taxa de todas as instâncias da Tabela 5.29 na relação valor alcançado pelo TLGA sobre o limite inferior encontrado pelo CPLEX para as instâncias do PCVG, a taxa média ficou em 1,12 para os melhores valores da Tabela 5.46 e de 1,17 para os valores médios da Tabela 5.47. Isto mostra a confiabilidade dos resultados apresentados pelas Tabelas 5.45, 5.46 e 5.47.

Ao ampliar a comparação entre os melhores valores de TLGA com G5-IMPnP, para instâncias de porte maior com limite de tempo no critério de parada, verifica-se que o gap_{li} médio dos melhores valores de G5-IMPnP ficou em 0,86% (Tabela 5.30), contra 9,31% para o TLGA (Tabela 5.46). Para os valores médios, o gap_{li} médio de G5-IMPnP ficou em 1,58% (Tabela 5.30) e 12,30% para o TLGA (Tabela 5.47).

Considerando ainda as instâncias de porte maior executadas com limite de tempo, o gap_{li} médio dos melhores valores de ILS-VND-nP ficou em 0,67% (Tabela 5.43) e 9,31% para o TLGA (Tabela 5.46). Para os valores médios, o gap_{li} médio de ILS-VND-nP ficou em 0,86% (Tabela 5.44) e para o TLGA em 12,30% (Tabela 5.47).

Em instâncias de pequeno porte com critério de parada estabelecido pelas iterações, o gap_{hc} médio de TLGA ficou em 12,29% com tempo médio 1,8s (Tabela 5.45), G4 (G4-IMPP) apresentou o gap_{hc} médio em 0,25% com tempo médio de 3,3s (Tabela 5.28) e ILS-VND-nP obteve o gap_{hc} médio de 0,27% com tempo médio de 12,05s (Tabela 5.42). O tempo computacional médio do G4 é menor do que o tempo depreendido pelo ILS-VND-nP. O TLGA obtém vantagem no esforço computacional em relação ao G4, mas os resultados apresentados pelo G4 são de qualidade superior ao TLGA.

Ao comparar TLGA com as heurísticas propostas (G4-IMPP, G5-IMPP, G5-IMPnP,

ILS-VND-P e ILS-VND-nP), verifica-se que as soluções produzidas pelas heurísticas propostas são de melhor qualidade tanto para instâncias de pequeno porte, quanto para as instâncias de porte maior. No caso de se necessitar encontrar soluções com menor esforço computacional, ainda é possível escolher uma heurística proposta para competir ao TLGA. Neste caso particular penalizar as arestas inter-grupos ou não, ao construir soluções para diversos tipos de instâncias do PCVG, através das heurísticas propostas, não interferem muito no desempenho dos algoritmos nesta comparação com o TLGA.

O desempenho inferior do TLGA é ocasionado por vários fatores: memória pouca explorada, ausência de busca local, nenhuma fase de intensificação nas soluções produzidas pela população inicial e pequena taxa de mutação realizada nos grupos. A população inicial é gerada por um conjunto de soluções diferentes, mas não de qualidade. A introdução de Reconexão de Caminhos, VND ou um outro tipo de busca local e exploração de memória podem melhorar a qualidade das soluções produzidas pelo TLGA para o PCVG.

5.6 Métodos Penalizados e não-Penalizados

A maioria dos testes realizados para as heurísticas desenvolvidas para o PCVG utiliza dois métodos diferentes para construir soluções. O primeiro método utilizado foi o algoritmo construtivo de Inserção mais Próxima que penaliza as arestas inter-grupos (IMMP) e o segundo, o algoritmo construtivo de Inserção mais Próxima que não penaliza as arestas inter-grupos (IMMnP). As descrições e características de ambos algoritmos construtivos foram mostrados na seção 3.1. Diversas heurísticas apresentadas (VND, G5 e ILS-VND) para o PCVG utilizaram os dois algoritmos construtivos. Na formulação exata, utilizando o CPLEX para solucionar o PCVG, pode-se também penalizar (CPLEX-P) ou não (CPLEX-nP) as arestas inter-grupos, detalhes dos resultados podem ser vistos na seção 5.1.

Para analisar os resultados das heurísticas e das formulações exatas que penalizam ou não as arestas inter-grupos, classificam-se como Métodos Penalizados (P), aqueles que utilizam a penalização inter-grupos e de Métodos não-Penalizados (nP) os que não penalizam. Dentre os Métodos Penalizados destacam-se: IMPP, VND-IMPP, G5-IMPP, ILS-VND-IMPP e CPLEX-P. Para os Métodos não-Penalizados, apresenta-se: IMPnP, VND-IMPnP, G5-IMPnP, ILS-VND-IMPnP e CPLEX-nP.

A Tabela 5.48 mostra para cada tipo de instância uma comparação do desempenho de cada método (Penalizado ou não-Penalizado). Foram realizados um total de 233 experi-

mentos. Para cada experimento realizado com uma instância foi atribuído um valor igual a um (1) se o método obteve melhor desempenho, zero (0) se obteve pior desempenho e meio (1/2) a cada método se o desempenho for igual em ambos. Para os métodos construtivos, VND, G5 e ILS-VND considera-se como melhor desempenho, o *gap* da melhor solução encontrada por estes métodos. Para o CPLEX foi considerado o menor tempo computacional pois ambos sempre encontram a solução ótima para as instâncias de pequeno porte. Os dados para calcular os resultados apresentados na Tabela 5.48 foram extraídos das Tabelas: 5.2, 5.11, 5.30, 5.38, 5.42 e 5.43.

Tipo	IMPP	IMPnP	VND-P	VND-nP	G5-P	G5-nP	ILS-P	ILS-nP	CPLEX-P	CPLEX-nP
1	**77,8**	22,2	33,3	**66,7**	**100,0**	0,0	**63,0**	37,0	36,4	**63,6**
2	**100,0**	0,0	**80,0**	20,0	50,0	50,0	**66,7**	33,3	-	-
3	**90,0**	10,0	**55,0**	45,0	**66,7**	33,3	16,7	**83,3**	-	-
4	40,0	**60,0**	40,0	**60,0**	37,5	**62,5**	**62,5**	37,5	-	-
5	**90,0**	10,0	50,0	50,0	**75,0**	25,0	50,0	50,0	**53,8**	46,2
6	**77,8**	22,2	**66,7**	33,3	**83,3**	16,7	**66,7**	33,3	**51,8**	48,2

Tabela 5.48: Desempenho dos Métodos Penalizados e não-Penalizados para cada *tipo* de instância.

Na primeira coluna da Tabela 5.48 apresenta-se o *tipo* de instância analisada, na segunda coluna a soma dos valores (em porcentagem) obtidos pelo algoritmo construtivo IMPP para todas as instâncias deste tipo, na terceira a soma dos valores para o algoritmo construtivo IMPnP, na quarta e quinta colunas as somas dos valores dos método VND com IMPP (VND-P) e do VND com IMPnP (VND-nP), respectivamente. Na sexta coluna a soma dos valores obtidos para G5-IMPP (G5-nP), na sétima a soma obtida pelo G5-IMPnP (G5-nP) e na oitava e nona colunas os valores de somas do ILS-VND-IMPP (ILS-P) e ILS-VND-IMPnP (ILS-nP), respectivamente. Por fim, nas duas últimas colunas os valores de soma obtidos para a formulação exata utilizado o CPLEX com penalização (CPLEX-P) e sem penalização (CPLEX-nP). Na Tabela 5.48 algumas porcentagens não foram calculadas, marcadas por "-", devido aos experimentos não terem sido realizados com CPLEX a este conjunto específico de instâncias. Não existem instâncias geradas de pequeno porte para este conjunto específico.

Observa-se na Tabela 5.48 para as instâncias do *tipo 2*, *5* e *6* que os Métodos Penalizados sempre ganham. Na maioria dos casos, as instâncias do *tipo 3* mostram-se porcentagens melhores com os Métodos Penalizados, exceto ao utilizar o algoritmo ILS-nP, já as instâncias do *tipo 4* com os não-Penalizados. Para as instâncias do *tipo 1* em algumas situações mostram-se com desempenho melhor para alguns Métodos Penalizados e em outras siutações aos Métodos não-Penalizados.

Na Tabela 5.49 mostra-se o desempenho global de todos os experimentos realizados com os Métodos Penalizados e não-Penalizados para os vários *tipos* de instâncias. As análises foram realizadas sobre o total de experimentos para cada *tipo* de instância. Os valores em porcentagens apresentados na Tabela 5.49 considera para seu cálculo todos os experimentos individuais de cada método (Penalizado ou não-Penalizado) somando todos os valores (0, 1/2 ou 1) e dividindo pelo total de experimentos. A primeira coluna da Tabela 5.49 mostra os *tipos* de instâncias, a segunda o desempenho expresso em porcentagem para os Métodos Penalizados (P) e na terceira coluna para os Métodos não-Penalizados (nP).

Tipo	P	nP
1	**54,3**	45,7
2	**78,1**	21,9
3	**65,4**	34,6
4	42,9	**57,1**
5	**63,4**	36,6
6	**61,5**	38,5

Tabela 5.49: Desempenho global da penalização e da não-penalização aos diferentes *tipos* de instâncias.

Através da Tabela 5.49 constata-se que para as instâncias do *tipo 2*, *3*, *5* e *6* obtém-se melhor desempenho na maioria dos casos quando é realizado a penalização entre as arestas inter-grupos. Para as instâncias do *tipo 4* o melhor desempenho acontece quando não é realizada a penalização. Nestas instâncias, os vértices não são dispersos e o número de vértices em cada grupo é constante. Para instâncias do *tipo 1*, considerando as Tabelas 5.48 e 5.49, não se pode afirmar qual é a melhor estratégia. As instâncias do *tipo 1* são clusterizadas permitindo que a distribuição dos vértices realize-se em torno de diversos centros, sendo que cada centro representa um grupo. Os resultados para algumas instâncias do *tipo 1* apresenta melhor desempenho quando as arestas inter-grupos não são penalizadas, para os diversos métodos heurísticos e formulações exatas.

Observa-se nos resultados dos Métodos Penalizados e dos não-Penalizados que para diversas instâncias, os algoritmos (heurísticos e métodos exatos) apresentam-se melhor desempenho quando a estratégia que *não penaliza as arestas inter-grupos* é aplicada. Assim, pode-se afirmar que a penalização não é uma estratégia boa, portanto existem várias instâncias, principalmente as do *tipo 1*, *tipo 3* e do *tipo 4* que necessitam utilizar algoritmos não-Penalizados.

5.7 Comentário Final

O trabalho abordou o PCVG, na versão genérica, onde a sequencia dos grupos não é pré-definida. Uma comparação entre os algoritmos propostos, para o PCVG na versão genérica, e um Algoritmo Genético TLGA (DING; CHENG; HE, 2007) disponível na literatura foi realizada. Cabe ressaltar que o TLGA não utiliza busca local e nem explora memória.

Outras comparações, também foram realizadas entre os algoritmos propostos e com uma formulação matemática. Em (MANIEZZO; STÜTZLE; VOSS, 2010) cita que se nenhum algoritmo está disponível na literatura para um problema de natureza combinatória, uma comparação poderá ser realizada entre o *algoritmo proposto* com um método de programação linear ou inteira, no qual o método obtém *bounds* para a função objetivo. Além disso, se existe elementos estocásticos no *algoritmo proposto*, no mínimo uma comparação com um procedimento de partida aleatória simples poderá ser executada, para mostrar que o desempenho do *algoritmo proposto* é melhor do que o procedimento de partida aleatória.

As comparações dos *algoritmos propostos* foram realizadas com limites inferiores obtidos pelo CPLEX através da formulação matemática de Miller/Chisman e com o GRASP tradicional, que é melhor do que um método de partida aleatória, pois utiliza uma fase construtiva no qual cada elemento incorporado na solução parcial é avaliado por uma função gulosa e ainda existe no GRASP tradicional uma fase de busca local.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho foi realizado um estudo elaborado de heurísticas e metaheurísticas aplicadas ao PCVG através de métodos construtivos, de buscas locais, de memória adaptativa utilizando a reconexão de caminhos e de métodos híbridos. Várias execuções das versões GRASP, ILS, Algoritmos Genéticos da literatura e formulações matemáticas usando *software* CPLEX com as instâncias geradas foram realizadas. Diversos tipos de reconexão de caminhos (RC1, RC2, RC3 e RC4) e de métodos de descida em vizinhança variável para compor uma busca mais intensiva e dos módulos do *Iterated Local Search* (ILS), como perturbação e critérios de aceitação de soluções foram avaliados.

Métodos heurísticos para o PCVG foram desenvolvidos e as primeiras comparações foram realizadas entre o GRASP tradicional (G1-IMPP) com as versões que utilizam à memória adaptativa (G2-IMPP, G3-IMPP, G4-IMPP, G5-IMPP e G6'-IMPP) através de módulos de reconexão de caminhos. Os resultados computacionais na forma empírica mostram que a adição de módulos com memória adaptativa nas heurísticas GRASP melhora bastante o desempenho em relação ao G1-IMPP.

Em particular, novos testes com instâncias de porte maior, contendo número de vértices entre 1173 a 2000, e análises de distribuição de probabilidade foram realizados entre G1-IMPP e G4-IMPP comprovando a importância de incluir memória adaptativa ao GRASP tradicional. Verificou-se que o uso conjugado de memória com busca local mais efetiva (VND) no GRASP (G5-IMPP) tende a produzir algoritmos mais eficientes.

Uma segunda etapa de comparação foi realizada entre a heurística híbrida ILS-VND-IMPP e as versões GRASP, mostrando que ILS-VND-IMPP obteve melhor desempenho. O ILS-VND-IMPP quando comparado ao melhor GRASP (G5-IMPP) se sobressaiu, permitindo melhor aproveitamento para os melhores valores e valores médios. Ao realizar

testes envolvendo soluções alvos, o ILS-VND-IMPP atingiu maior número de alvos em tempos menores do que (G5-IMPP). Uma nova heurística GRASP com memória adaptativa (G5-IMPnP) utilizando o algoritmo construtivo IMPnP foi desenvolvida para analisar a não penalização das arestas inter-grupos. Neste contexto, verificou-se através de resultados computacionais que para algumas instâncias, o G5-IMPnP se sobressaiu em relação ao G5-IMPP.

Uma terceira etapa foi realizada para comparar ILS-VND-IMPP e ILS-VND-IMPnP, onde este último conseguiu maior número de soluções melhores. Entretanto, na comparação entre os gap_{li}s médios dos melhores valores e dos valores médios, o ILS-VND-IMPP obteve melhor desempenho do que ILS-VND-IMPnP, tanto para instâncias de porte maior, quanto para instâncias de porte menor. Os fatores para um bom desempenho do ILS-VND-IMPP está relacionado ao balanceamento entre a diversificação e a intensificação, aliado ao uso de memória no módulo critério de aceitação.

As soluções ótimas e os limites inferiores, utilizando a formulação exata com o *software* CPLEX, foram obtidos para diversas instâncias geradas para o PCVG, nos quais foram realizadas comparações com as heurísticas apresentadas neste trabalho (G4-IMPP, G5-IMPP, G5-IMPnP, ILS-VND-IMPP e ILS-VND-IMPnP). Na maior parte das instâncias de pequeno porte testadas as heurísticas apresentadas alcançaram soluções ótimas. Ao estabelecer limites de tempo para o CPLEX, os gap_{li}s alcançados pelas heurísticas, medidos em relação ao limite inferior do CPLEX, para as instâncias de porte maior foram melhores do que os obtidos pela formulação exata, em sua grande maioria.

As melhores heurísticas propostas (G4-IMPP, G5-IMPP, G5-IMPnP, ILS-VND-IMPP e ILS-VND-IMPnP) foram comparadas a um Algoritmo Genético da literatura, numa etapa final, produzindo estas heurísticas soluções de melhor qualidade tanto para instâncias de pequeno porte, quanto para as instâncias de porte maior.

Para o nosso conhecimento, esta é a primeira vez que o GRASP com memória adaptativa e o ILS é utilizado para a solução do PCVG genérico, onde não é previamente fixada a ordem de visitas aos grupos. Os resultados apresentados por este trabalho mostram que as formas híbridas obtiveram melhor aproveitamento comparado às formas canônicas de cada metaheurística. Destacam-se como heurísticas híbridas neste trabalho: G4-IMPP com RC1 e RC2; G5-IMPP e G5-IMPnP, ambos com RC3 e o método VND; e ILS-VND-IMPP e ILS-VND-IMPnP com o algoritmo construtivo IMPP (uma das fases do GRASP) e o método VND.

Ao analisar os Métodos Penalizados (IMPP, VND-IMPP, G5-IMPP, ILS-VND-IMPP

e CPLEX-P) com os Métodos não-Penalizados (IMPnP, VND-IMPnP, G5-IMPnP, ILS-VND-IMPnP e CPLEX-nP) verificou-se que para as instâncias do *tipo 2*, *5* e *6*, obtém-se melhor desempenho quando as arestas inter-grupos são penalizadas. Em diversas heurísticas e em formulações exatas, para as instâncias do *tipo 1*, os melhores resultados foram encontrados quando não se aplica a penalização nas arestas inter-grupos. Para as instâncias do *tipo 3*, na maioria dos casos, os algoritmos que utilizaram a estratégia de penalização ocasionou melhor desempenho, exceto ao utilizarem o algoritmo ILS-nP. Já nas instâncias do *tipo 4* é preferível construir soluções sem penalizar as arestas inter-grupos. Nas instâncias do *tipo 1*, *3* e *4*, os vértices não são dispersos e em cada grupo os vértices são agrupados com melhor distribuição geométrica em torno de um centro.

Assim, verifica-se que para diversas instâncias a resolução do PCVG através de algoritmos heurísticos ou métodos exatos com estratégia de não penalização das arestas inter-grupos torna-se promissora. Pode-se afirmar que a penalização não é uma estratégia adequada para estas instâncias, principalmente para as instâncias do *tipo 1*, *3* e *4* e para algumas instâncias do *tipo 2*, *5* e *6* .

Os resultados computacionais realizados neste trabalho utilizaram instâncias euclidianas e simétricas, mas as heurísticas que foram apresentadas podem ser adaptadas para produzir soluções com instâncias de outra natureza, como por exemplo, instâncias contendo matrizes de custos para todos os vértices associados. Neste caso, somente necessita ler os custos sem precisar calcular as distâncias euclidianas.

Como resultado desta tese envolvendo o GRASP tradicional (G1-IMPP), as versões GRASP (G2-IMPP, G3-IMPP e G4-IMPP) com memória adaptativa (RC1 e/ou RC2) e os métodos híbridos, o GRASP com a reconexão de caminhos RC3 e VND (G5-IMPP e G5-IMPnP) e o *Iterated Local Search* com VND (ILS-VND-IMPP e ILS-VND-IMPnP), foram geradas três publicações, até o momento, descritas a seguir:

- Mestria, M., Ochi, L. S., Martins, S. L. GRASP com Memória Adaptativa para o Problema do Caixeiro Viajante com Grupamentos. In *XXIX Encontro Nacional de Engenharia de Produção* (Salvador, BA, 2009), vol. 1, Anais do XXIX ENEGEP, ABEPRO, RJ, (CD-ROM).

- Mestria, M., Ochi, L. S., Martins, S. L. Desenvolvimento e Análise Experimental de Heurísticas GRASP para o Problema do Caixeiro Viajante com Grupamentos. In *IX Congresso Brasileiro de Redes Neurais/Inteligência Computacional (IX CBRN)* (Ouro Preto, MG, 2009), vol. 1, Anais do IX CBRN/Inteligência Computacional, SBRN, RJ, (CD-ROM), ISSN 2177-1200.

- Mestria, M., Ochi, L. S., Martins, S. L. Heurísticas Híbridas para o Problema do Caixeiro Viajante com Grupamentos. In *XXX Encontro Nacional de Engenharia de Produção* (São Carlos, SP, 2010), vol. 1, Anais do XXX ENEGEP, ABEPRO, RJ, (CD-ROM).

Trabalhos futuros incluem a adaptação dos módulos desenvolvidos nesta tese como Reconexão de Caminhos (RC) e VND para outras metaheurísticas tais como os Algoritmos Evolutivos e *Variable Neighborhood Search* (VNS), bem como outras formas híbridas envolvendo as heurísticas GRASP, RC, VND e ILS combinando-as em diversas estratégias.

Novas alternativas poderão ser estudadas para utilizar os algoritmos construtivos com penalização nas arestas e sem penalização, ajustando-os com as buscas locais (2-Optimal, 3-Optimal e VND) e/ou reconexão de caminhos (RC1, RC2, RC3 e RC4) para formar módulos que se adaptem às futuras metaheurísticas a serem implementadas.

Outras fontes de pesquisas futuras envolvem: o desenvolvimento de um algoritmo *branch-and-cut* para o PCVG utilizando a formulação matemática de Dantzig/Chisman, adaptação das heurísticas desenvolvidas neste trabalho para realizar testes com instâncias não euclidianas e por último um estudo mais profundo para o PCVG assimétrico, ainda não contemplado pela literatura.

Referências

AHMADI, S.; OSMAN, I. H. Greedy random adaptive memory programming search for the capacitated clustering problem. *European Journal of Operational Research*, v. 162, n. 1, p. 30–44, 2005.

AIEX, R.; RESENDE, M.; RIBEIRO, C. Probability Distribution of Solution Time in GRASP: An Experimental Investigation. *Journal of Heuristics*, v. 8, n. 3, p. 343–373, 2002.

AIEX, R.; RESENDE, M.; RIBEIRO, C. TTT plots: a Perl Program to Create Time-to-Target Plots. *Optimization Letters*, v. 1, n. 4, p. 355–366, 2007.

ANILY, S.; BRAMEL, J.; HERTZ, A. A 5/3-approximation Algorithm for the Clustered Traveling Salesman Tour and Path Problems. *Operations Research Letters*, v. 24, n. 1-2, p. 29–35, 1999.

APPLEGATE, D.; BIXBY, R.; CHVÁTAL, V.; COOK, W. Concorde TSP Solver. In: . School of Industrial and Systems Engineering, Georgia Tech: William Cook, 2007. Disponível em: <http://www.tsp.gatech.edu/concorde/index.html>. Acesso em: 22/12/2007.

APPLEGATE, D.; COOK, W.; ROHE, A. Chained Lin-Kernighan for Large Traveling Salesman Problems. *INFORMS Journal on Computing*, INFORMS, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, EUA, v. 15, n. 1, p. 82–92, 2003.

ARKIN, E. M.; HASSIN, R.; KLEIN, L. Restricted Delivery Problems on a Network. *Networks*, v. 29, n. 4, p. 205–216, 1997.

ARMENTANO, V. A.; FILHO, M. F. de F. Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based GRASP approach. *European Journal of Operational Research*, v. 183, n. 1, p. 100–114, 2007.

ARTHANARI, T.; USHA, M. An Alternate Formulation of the Symmetric Traveling Salesman Problem and its Properties. *Discrete Applied Mathematics*, v. 98, p. 173–190, 2000.

BAKER, J. E. Adaptive selection methods for genetic algorithms. In: *Proceedings of the 1st International Conference on Genetic Algorithms*. Mahwah, NJ, EUA: J. J. Grefenstette, editor, Lawrence Erlbaum Associates, Inc., 1985. p. 101–111. ISBN 0-8058-0426-9.

BALAKRISHNAN, N.; LUCENA, A.; WONG, R. T. Scheduling examinations to reduce second-order conflicts. *Computers & Operations Research*, v. 19, n. 5, p. 353–361, 1992.

BASTOS, L. de O.; OCHI, L. S.; MACAMBIRA, E. M. Grasp with path relinking for the sonet ring assignment problem. In: *HIS '05: Proceedings of the Fifth International Conference on Hybrid Intelligent Systems*. Washington, DC, EUA: IEEE Computer Society, 2005. p. 239–244. ISBN 0-7695-2457-5.

BELLMORE, M.; NEMHAUSER, G. L. The Traveling Salesman Problem: A Survey. *Operations Research*, v. 16, n. 3, p. 538–558, 1968.

BENTLEY, J. L. Experiments on traveling salesman heuristics. In: *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, São Francisco, Califórnia, EUA. Filadélfia, PA, EUA: Society for Industrial and Applied Mathematics, 1990. p. 91–99. ISBN 0-89871-251-3.

BLICKLE, T.; THIELE, L. *A Comparison of Selection Schemes Used in Genetic Algorithms*. Relatório Técnico, n. 11, versão 2 (2ª edição). Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurique, Suíça, 1995.

BLUM, C.; ROLI, A. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, ACM, Nova York, NY, EUA, v. 35, n. 3, p. 268–308, 2003.

BOUDIA, M.; LOULY, M.; PRINS, C. A Reactive GRASP and Path Relinking for a Combined Production-Distribution Problem. *Computers & Operations Research*, v. 34, n. 11, p. 3402–3419, 2007.

BRITO, J.; OCHI, L. S.; MONTENEGRO, F.; MACULAN, N. An iterative local search approach applied to the optimal stratification problem. *International Transactions in Operational Research*, Blackwell Publishing Ltd, v. 17, n. 6, p. 753–764, 2010.

CHAVES, A. A.; LORENA, L. A. N. Hybrid Algorithms with Detection of Promising Areas for the Prize Collecting Travelling Salesman Problem. In: *Proceedings of the Fifth International Conference on Hybrid Intelligent Systems (HIS '05)*. Washington, DC, EUA: IEEE Computer Society, 2005. p. 49–54. ISBN 0-7695-2457-5.

CHENG, Y.; DING, C. Comunicação Particular. Department of Industrial Engineering, Tsinghua University, Beijing (Pequim), China, 2010.

CHISMAN, J. A. The Clustered Traveling Salesman Problem. *Computers & Operations Research*, v. 2, n. 2, p. 115–119, 1975.

CHRISTOFIDES, N. *Worst-case Analysis of a new Heuristic for the Travelling Salesman Problem*. Relatório Técnico, n. 388. Graduate School of Industrial Administration, Carnegie-Mellon University, 1976.

CLARKE, G.; WRIGHT, J. Scheduling of Vehicles from a Central Depot to a Number of Delivering Points. *Operations Research*, v. 12, n. 4, p. 568–581, 1964.

CLAUS, A. A New Formulation for the Travelling Salesman Problem. *SIAM J. ALG. DISC. Math*, v. 5, n. 1, p. 21–25, 1984.

CODENOTTI, B.; MANZINI, G.; MARGARA, L.; RESTA, G. Perturbation: An Efficient Technique for the Solution of Very Large Instances of the Euclidean TSP. *INFORMS Journal on Computing*, v. 8, n. 2, p. 125–133, 1996.

CORDEAU, J.-F.; LAPORTE, G.; PASIN, F. An iterated local search heuristic for the logistics network design problem with single assignment. *International Journal of Production Economics*, v. 113, n. 2, p. 626–640, 2008.

CPLEX. ILOG CPLEX 11.2 User's Manual and Reference Manual, 2009. In: . ILOG S.A.: [s.n.], 2009. Disponível em: <http://www.ilog.com>. Acesso em: 06/01/2009.

DANTZIG, G.; FULKERSON, R.; JOHNSON, S. Solution of a Large-Scale Traveling-Salesman Problem. *Journal of the Operations Research Society of America*, v. 2, p. 393–410, 1954.

DING, C.; CHENG, Y.; HE, M. Two-Level Genetic Algorithm for Clustered Traveling Salesman Problem with Application in Large-Scale TSPs. *Tsinghua Science and Technology*, v. 12, n. 4, p. 459–465, 2007.

DONG, X.; HUANG, H.; CHEN, P. An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Computers & Operations Research*, doi:10.1016/j.cor.2008.04.001, p. 1–6, 2008.

DRUMMOND, L. M. de A.; VIANNA, L. S.; SILVA, M. B. da; OCHI, L. S. Distributed Parallel Metaheuristics based on GRASP and VNS for Solving the Traveling Purchaser Problem. In: *Proceedings of the Ninth International Conference on Parallel and Distributed Systems*. [S.l.: s.n.], 2002. v. 1, p. 257–263.

ERDOGAN, G.; CORDEAU, J.-F.; LAPORTE, G. The pickup and delivery traveling salesman problem with first-in-first-out loading. *Computers & Operations Research*, doi:10.1016/j.cor.2008.05.005, p. 1–9, 2008.

FEO, T.; RESENDE, M. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, v. 6, n. 2, p. 109–133, 1995.

FERREIRA, C.; WAKABAYASHI, Y. Combinatória Poliédrica e Planos-de-Corte Faciais. In: *X Escola de Computação*. UNICAMP, Campinas, SP, Brasil: [s.n.], 1996.

FESTA, P.; RESENDE, M. GRASP: An annotated bibliography. In: RIBEIRO, C. C.; HANSEN, P. (Ed.). *Essays and surveys in metaheuristics*. [S.l.]: Kluwer Academic Publishers, 2002. p. 325–367.

FIECHTER, C. N. A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics*, v. 51, n. 3, p. 243–267, 1994.

FISCHETTI, M.; TOTH, P. An Additive Bounding Procedure for the Asymmetric Travelling Salesman Problem. *Mathematical Programming*, Springer-Verlag Nova York, Inc., Secaucus, NJ, EUA, v. 53, n. 2, p. 173–197, 1992.

FOX, K. R.; GAVISH, B.; GRAVES, S. C. An n-Constraint Formulation of the (Time-Dependent) Traveling Salesman Problem. *Operations Research*, v. 28, n. 4, p. 1018–1021, 1980.

FREDMAN, M. L.; JOHNSON, D. S.; McGEOCH, L. A.; OSTHEIMER, G. Data structures for traveling salesmen. *Journal of Algorithms*, v. 18, n. 3, p. 432–479, 1995.

GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Nova York, NY, EUA: W. H. Freeman & Co., 1979. ISBN 0716710447.

GENDREAU, M.; GUERTIN, F.; POTVIN, J.-Y.; SEGUIN, R. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, v. 14, n. 3, p. 157–174, 2006.

GENDREAU, M.; HERTZ, A.; LAPORTE, G. New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. *Operations Research*, v. 40, n. 6, p. 1086–1094, 1992.

GENDREAU, M.; HERTZ, A.; LAPORTE, G. The Traveling Salesman Problem with Backhauls. *Computers & Operations Research*, v. 23, n. 5, p. 501–508, 1996.

GENDREAU, M.; LAPORTE, G.; HERTZ, A. An Approximation Algorithm for the Traveling Salesman Problem with Backhauls. *Operational Research*, v. 45, n. 4, p. 639–641, 1997.

GENDREAU, M.; LAPORTE, G.; POTVIN, J. Y. *Heuristics for the Clustered Traveling Salesman Problem*. Relatório Técnico, n. CRT-94-54. Centre de recherche sur les transports, Universidade de Montreal, Montreal, Canadá, 1994.

GHAZIRI, H.; OSMAN, I. H. A Neural Network for the Traveling Salesman Problem with Backhauls. *Computers & Industrial Engineering*, v. 44, n. 2, p. 267–281, 2003.

GLOVER, F. Tabu Search and Adaptive Memory Programing - Advances, Applications and Challenges. In: BARR, R. S.; HELGASON, R. V.; KENNINGTON, J. L. (Ed.). *Interfaces in Computer Science and Operations Research*. Dordrecht, MA, EUA: Kluwer Academic Plublishres, 1996. p. 1–75.

GLOVER, F.; KOCHENBERGER, G. A. *Handbook of Metaheuristics*. Boston, EUA: Kluwer Academic Publishers, 2003. (International Series in Operations Research & Management Science, v. 57).

GLOVER, F.; LAGUNA, F. *Tabu Search*. Norwell, MA, EUA: Kluwer Academic Publishers, 1997. ISBN 079239965X.

GLOVER, F.; LAGUNA, M.; MARTÍ, R. Fundamentals of Scatter Search and Path Relinking. *Control and Cybernetics*, v. 29, n. 3, p. 653–684, 2000.

GLOVER, F.; LAGUNA, M.; MARTÍ, R. Scatter Search and Path Relinking: Foundations and Advanced Designs. In: ONWUBOLU, G. C.; BABU, B. V. (Ed.). *New Optimization Techniques in Engineering*. Heidelberg: Springer-Verlag, 2004. p. 87–100.

GOMES, L. M.; DINIZ, V. B.; MARTINHON, C. A. An Hybrid GRASP+VNS Metaheuristic for the Prize Collecting Traveling Salesman Problem. In: *Anais do XXXII Simpósio Brasileiro de Pesquisa Operacional (SBPO)*. [S.l.: s.n.], 2000. p. 1656–1666.

GOMORY, R. E.; HU, T. C. Multi-Terminal Network Flows. *Journal of the Society for Industrial and Applied Mathematics*, SIAM, v. 9, n. 4, p. 551–570, 1961.

GUTTMANN-BECK, N.; HASSIN, R.; KHULLER, S.; RAGHAVACHARI, B. Approximation Algorithms with Bounded Performance Guarantees for the Clustered Traveling Salesman Problem. *Algorithmica*, v. 28, n. 4, p. 422–437, 2000.

HANSEN, P.; MLADENOVIĆ, N. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, v. 130, n. 3, p. 449–467, 2001.

HANSEN, P.; MLADENOVIĆ, N. Variable Neighborhood Search. In: GLOVER, F. W.; KOCHENBERGER, G. A. (Ed.). *Handbook of Metaheuristics*. Boston, Dordrecht: London: Kluwer Academic Publisher, 2003. p. 145–184.

HASHIMOTO, H.; YAGIURA, M.; IBARAKI, T. An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. *Discrete Optimization*, v. 5, n. 2, p. 434–456, 2008.

HELD, M.; KARP, R. M. The Traveling-Salesman Problem and Minimum Spanning Trees. *Operations Research*, v. 18, n. 6, p. 1138–1162, 1970.

HELD, M.; KARP, R. M. The Traveling-Salesman Problem and Minimum Spanning Trees: Part II. *Mathematical Programming*, Springer Berlin / Heidelberg, v. 1, n. 1, p. 6–25, 1971.

HERNÁNDEZ-PÉREZ, H.; RODRÍGUEZ-MARTÍN, I.; SALAZAR-GONZÁLEZ, J. J. A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Operations Research*, v. 36, n. 5, p. 1639 – 1645, 2009.

HERTZ, A.; TAILLARD, E.; WERRA, D. de. *A Tutorial on Tabu Search*. EPFL, Département de Mathématiques, MA-Ecublens, CH-1015 Lausanne, 2008. Disponível em: <http://www.cs.colostate.edu/ whitley/CS640/hertz92tutorial.pdf>. Acesso em: 12 de julho de 2008.

HOOGEVEEN, J. A. Analysis of Christofides' Heuristic: Some Paths are more Difficult than Cycles. *Operations Research Letters*, v. 10, n. 5, p. 291–295, 1991.

HOOS, H. H.; STÜTZLE, T. Towards a Characterisation of the Behaviour of Stochastic Local Search Algorithms for SAT. *Artificial Intelligence*, Elsevier Science Publishers Ltd., Essex, UK, v. 112, n. 1-2, p. 213–232, 1999.

HOOS, H. H.; STÜTZLE, T. Empirical Analysis of Randomized Algorithms. In: GONZALEZ, T. F. (Ed.). *Approximation Algorithms and Metaheuristics*. [S.l.: s.n.], 2007.

IBARAKI, T.; IMAHORI, S.; NONOBE, K.; SOBUE, K.; UNO, T.; YAGIURA, M. An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. *Discrete Applied Mathematics*, v. 156, n. 11, p. 2050–2069, 2008.

JOHNSON, D. S.; McGEOCH, L. A. The Traveling Salesman Problem: A Case Study in Local Optimization. In: AARTS, E. H. L.; LENSTRA, J. K. (Ed.). *Local Search in Combinatorial Optimization*. Londres: John Wiley and Sons, 1997. p. 215–310.

JOHNSON, D. S.; McGEOCH, L. A. Experimental Analysis of Heuristics for the STSP. In: GUTIN, G.; PUNNEN, A. (Ed.). *The Traveling Salesman Problem and its Variations*. Dordrecht, Holanda: Kluwer Academic Plublishres, 2002. p. 369–443.

JONGENS, K.; VOLGENANT, T. The Symmetric Clustered Traveling Salesman Problem. *European Journal of Operational Research*, v. 19, n. 1, p. 68–75, 1985.

KANUNGO, T.; MOUNT, D. M.; NETANYAHU, N. S.; PIATKO, C. D.; SILVERMAN, R.; Y.WU, A. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Transactions Pattern Analysis Machine Intelligence*, IEEE Computer Society, Washington, DC, EUA, v. 24, n. 7, p. 881–892, 2002.

KOHONEN, T. *Self-Organizing Maps*. Heidelberg, Berlim: 501 páginas, Springer Verlag, 2001. (Series in Information Sciences, v. 30).

KULTUREL-KONAK, S.; NORMAN, B. A.; COIT, D. W.; SMITH, A. E. Exploiting tabu search memory in constrained problems. *INFORMS Journal on Computing*, INFORMS, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, EUA, v. 16, n. 3, p. 241–254, 2004.

LAGUNA, M.; MARTÍ, R. Grasp and Path Relinking for 2-Layer Straight Line Crossing Minimization. *INFORMS Journal on Computing*, INFORMS, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, v. 11, n. 1, p. 44–52, 1999.

LAPLACE, C.; LAI, H.; MANDRAVELLOS, Y.; BERG, M. The Dev-C++ Resource Site. In: . [s.n.], 2005. Disponível em: <http://www.bloodshed.net/dev/index.html>. Acesso em: 13/09/2007.

LAPORTE, G.; LOPES, L.; SOUMIS, F. Optimal Sequencing Rules for Some Large-Scale Flexible Manufacturing Problems Under the Manhattan and Chebychev Metrics. *International Journal of Flexible Manufacturing Systems*, v. 10, p. 27–42, 1998.

LAPORTE, G.; PALEKAR, U. Some Applications of the Clustered Travelling Salesman Problem. *Journal of the Operational Research Society*, v. 53, n. 9, p. 972–976, 2002.

LAPORTE, G.; POTVIN, J.-Y.; QUILLERET, F. A Tabu Search Heuristic using Genetic Diversification for the Clustered Traveling Salesman Problem. *Journal of Heuristics*, v. 2, n. 3, p. 187–200, 1996.

LAWLER, E. L.; LENSTRA, J. K.; KAN, A. H. G. R.; SHMOYS, D. B. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Nova York, NY, EUA: John-Wiley and Sons, Ltda, 1985. 463 p.

LIN, S.; KERNIGHAN, B. W. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, v. 21, n. 2, p. 498–516, 1973.

LOKIN, F. C. J. Procedures for Travelling Salesman Problems with Additional Constraints. *European Journal of Operational Research*, v. 3, n. 2, p. 135–141, 1979.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated Local Search. In: GLOVER, F.; KOCHENBERGER, G. (Ed.). *Handbook of Metaheuristics*. [S.l.]: Kluwer Academic Publishers, Boston, 2002. p. 321–353.

MANIEZZO, V.; STÜTZLE, T.; VOSS, S. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*. [S.l.]: Springer-Verlag, 2010. 270 p. (Annals of Information Systems).

MARTÍ, R.; LAGUNA, M.; GLOVER, F. Principles of Scatter Search. *European Journal of Operational Research*, v. 169, n. 2, p. 359–372, 2006.

MARTIN, O.; OTTO, S. W.; FELTEN, E. W. Large-Step Markov Chains for the Traveling Salesman Problem. *Complex Systems*, v. 5, p. 299–326, 1991.

MARTIN, O.; OTTO, S. W.; FELTEN, E. W. Large-Step Markov Chains for the TSP Incorporating Local Search Heuristics. *Operations Research Letters*, v. 11, n. 4, p. 219–224, 1992.

MARTIN, O. C.; OTTO, S. W. Combining Simulated Annealing with Local Search Heuristics. *Annals of Operations Research*, Springer Netherlands, v. 63, n. 1, p. 57–75, 1996.

MESTRIA, M.; OCHI, L. S.; MARTINS, S. L. Desenvolvimento e Análise Experimental de Heurísticas GRASP para o Problema do Caixeiro Viajante com Grupamentos. In: *IX Congresso Brasileiro de Redes Neurais/Inteligência Computacional (IX CBRN)*. Ouro Preto, MG, Brasil, 2009: Anais do IX CBRN/Inteligência Computacional, SBRN, RJ (CD-ROM), 2009. v. 1. ISSN 2177-1200.

MESTRIA, M.; OCHI, L. S.; MARTINS, S. L. GRASP com Memória Adaptativa para o Problema do Caixeiro Viajante com Grupamentos. In: *XXIX Encontro Nacional de Engenharia de Produção*. Salvador, BA: Anais do XXIX ENEGEP, ABEPRO, RJ (CD-ROM), 2009. v. 1.

MESTRIA, M.; OCHI, L. S.; MARTINS, S. L. Heurísticas Híbridas para o Problema do Caixeiro Viajante com Grupamentos. In: *XXX Encontro Nacional de Engenharia de Produção*. São Carlos, SP: Anais do XXX ENEGEP, ABEPRO, RJ (CD-ROM), 2010. v. 1.

MESTRIA, M.; OCHI, L. S.; MARTINS, S. L. Manual de Utilização das Instâncias para o Problema do Caixeiro Viajante com Grupamentos. Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo, Vitória, ES, 2010. Disponível em: <ftp://ftp.cefetes.br/Teses_Dissertacoes/MarioMestria/instances/>.

MICROSOFT. Funcionalidades do Utilitário de Desfragmentação do Disco Rígido do Windows Vista. 2008. Disponível em: <http://support.microsoft.com/kb/942092/pt>. Acesso em: 19/06/2008.

MILLER, C. E.; TUCKER, A. W.; ZEMLIN, R. A. Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM*, ACM Press, Nova York, NY, EUA, v. 7, n. 4, p. 326–329, 1960.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. *Computers and Operations Research*, Elsevier Science Ltd., Oxford, UK, UK, v. 24, n. 11, p. 1097–1100, 1997.

MUYLDERMANS, L.; BEULLENS, P.; CATTRYSSE, D.; OUDHEUSDEN, D. V. Exploring variants of 2-opt and 3-opt for the general routing problem. *Operational Research*, v. 53, n. 6, p. 982–995, 2005.

NEVES, T. A. *Heurísticas com Memória Adaptativa Aplicadas ao Problema de Roteamento e Scheduling de Sondas de Manutenção*. 81 p. Dissertação (Mestrado) — Mestrado em Computação, Universidade Federal Fluminense, Niterói, RJ, 2007.

OCHI, L. S.; RABELLO, P. G. A new hybrid genetic algorithm for the clustered traveling salesman problem. In: *Procedings of the VIII CLAIO. Latin-Ibero-American Congress on Operations Research and System Engineering*. Rio de Janeiro, RJ: ALIO/SOBRAPO, 1996. v. 3, p. 1160–1165.

OLIVERA, A.; VIERA, O. Adaptive memory programming for the vehicle routing problem with multiple trips. *Computers & Operations Research*, v. 34, n. 1, p. 28–47, 2007.

OR, I. *Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Blood Banking*. Tese (Doutorado) — Department of Industrial Engineering and Management Science, Northwestern University, Evanston, EUA, 1976.

ORMAN, A.; WILLIAMS, H. *A Survey of Different Integer Programming Formulations of the Travelling Salesman Problem*. Relatório Técnico, n. LSEOR 04.67. Department of Operational Research, London School of Economics and Political Science, Londres, Grã Bretanha, 2004.

PATAKI, G. Teaching Integer Programming Formulations Using the Traveling Salesman Problem. *SIAM REVIEW*, v. 45, n. 1, p. 116–123, 2003.

PAULA, M. M. V. *Heurísticas e Metaheurísticas para o Problema do Caixeiro Viajante com Grupamentos*. 106 p. Dissertação (Mestrado) — Mestrado em Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, 2001.

PIÑANA, E.; PLANA, I.; CAMPOS, V.; MARTÍ, R. GRASP and Path Relinking for the Matrix Bandwidth Minimization. *European Journal of Operational Research*, v. 153, n. 1, p. 200–210, 2004.

PITSOULIS, L.; RESENDE, M. Greedy Randomized Adaptive Search Procedures. In: P.M.PARDALOS; M.G.C.RESENDE (Ed.). *Handbook of Applied Optimization*. [S.l.]: Oxford University Press, 2002. p. 168–181.

POTVIN, J.-Y.; GUERTIN, F. *A Genetic Algorithm for the Clustered Traveling Salesman Problem with an A Priori Order on the Clusters*. Relatório Técnico, n. CRT-95-06. Centre de recherche sur les transports, Universidade de Montreal, Montreal, Canadá, 1995.

POTVIN, J.-Y.; GUERTIN, F. The Clustered Traveling Salesman Problem: A Genetic Approach. In: OSMAN, I. H.; KELLY, J. (Ed.). *Meta-heuristics: Theory & Applications*. Norwell, MA, EUA: Kluwer Academic Plublishres, 1996. cap. 37, p. 619–631.

PRAIS, M.; RIBEIRO, C. Parameter Variation in GRASP Procedures. *Investigación Operativa*, v. 9, p. 1–20, 2000.

PRAIS, M.; RIBEIRO, C. Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment. *INFORMS Journal on Computing*, v. 12, p. 164–176, 2000.

REINELT, G. *The Traveling Salesman: Computational Solutions for TSP Applications*. Heidelberg, Berlim: 213 pp, Springer Verlag, 1994. (Lecture Notes in Computer Science, v. 840).

RESENDE, M.; RIBEIRO, C. Greedy Randomized Adaptive Search Procedures. In: GLOVER, F.; KOCHENBERGER, G. (Ed.). *Handbook of Metaheuristics*. [S.l.]: Kluwer Academic Publishers, 2002. p. 219–249.

RESENDE, M.; RIBEIRO, C. GRASP with Path-Relinking: Recent Advances and Applications. In: IBARAKI, T.; NONOBE, K.; YAGIURA, M. (Ed.). *Metaheuristics: Progress as Real Problem Solvers*. [S.l.]: Springer Verlag, 2005. p. 29–63.

RESENDE, M.; WERNECK, R. F. A Hybrid Heuristic for the p-Median Problem. *Journal of Heuristics*, v. 10, p. 59–88, 2004.

RESENDE, M.; WERNECK, R. F. A Hybrid Multistart Heuristic for the Uncapacitated Facility Location Problem. *European Journal of Operational Research*, v. 174, p. 54–68, 2006.

RESENDE, M. G.; MARTÍ, R.; GALLEGO, M.; DUARTE, A. GRASP and Path Relinking for the Max-Min Diversity Problem. *Computers & Operations Research*, doi:10.1016/j.cor.2008.05.011, 2008.

SHERALI, H. D.; DRISCOLL, P. J. Evolution and state-of-the-art in integer programming. *Journal of Computational and Applied Mathematics*, Elsevier Science Publishers B. V., Amesterdã, Países Baixos, v. 124, n. 1-2, p. 319–340, 2000.

SILVA, G. C.; ANDRADE, M. R.; OCHI, L. S.; MARTINS, S. L.; PLASTINO, A. New heuristics for the maximum diversity problem. *Journal of Heuristics*, Kluwer Academic Publishers, Hingham, MA, EUA, v. 13, n. 4, p. 315–336, 2007.

SILVA, M. B. da; DRUMMOND, L. M. de A.; OCHI, L. S. Metaheurística GRASP/VNS para a solução de Problemas de Otimização Combinatória. In: *Anais do XXXII Simpósio Brasileiro de Pesquisa Operacional (SBPO)*. [S.l.: s.n.], 2000. v. 1, p. 352–360.

SOUZA, M. J. F.; MINE, M. T.; SILVA, M. S. A.; OCHI, L. S.; PONTES, R. C. V. Programação de Jogos da Primeira Divisão do Campeonato Brasileiro de Futebol por meio da Metaheurística *Iterated Local Search*. In: *Anais do VIII Simpósio de Pesquisa Operacional e Logística da Marinha - VIII SPOLM - CASNAV*. Rio de Janeiro, RJ: [s.n.], 2005. v. 1, p. 1–14.

SOUZA, M. J. F.; MINE, M. T.; SILVA, M. S. A.; OCHI, L. S.; SUBRAMANIAN, A. A hybrid heuristic, based on Iterated Local Search and GENIUS, for the Vehicle Routing Problem with Simultaneous Pickup and Delivery. *Submetido a International Journal of Logistics Systems Management*, Inderscience Publishers, 2010. ISSN 1742-7967.

STÜTZLE, T. *Applying Iterated Local Search to the Permutation Flow Shop Problem*. Relatório Técnico, n. AIDA-98-04. Department of Computer Science, University of Technology, Darmstadt, Alemanha, 1998.

STÜTZLE, T. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, v. 174, n. 3, p. 1519–1539, 2006.

STÜTZLE, T.; HOOS, H. H. Analyzing the Run-Time Behaviour of Iterated Local Search for the TSP. In: HANSEN, P.; RIBEIRO, C. (Ed.). *Essays and Surveys on Metaheuristics*. Norwell, MA: Kluwer Academic Publishers, 2002, (Operations Research/Computer Science Interfaces Series). p. 589–612.

SUBRAMANIAN, A.; CABRAL, L. A. F. An ILS Based Heuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Limit. In: HEMERT, J. I. van; COTTA, C. (Ed.). *Proceedings of the Eighth European Conference on Evolutionary Computation in Combinatorial Optimisation - EvoCOP*. Nápoles, Itália: Springer Verlag, 2008. (Lecture Notes in Computer Science, v. 4972), p. 135–146.

SUBRAMANIAN, A.; DRUMMOND, L. M. A.; BENTES, C.; OCHI, L. S.; FARIAS, R. A parallel heuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery. *Computers Operations Research*, v. 37, n. 11, p. 1899–1911, 2010.

TAILLARD, E. D. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, v. 17, p. 443–455, 1991.

TAILLARD, E. D. Comparison of Non-Deterministic Iterative Methods. In: *MIC' 2001 - 4th Metaheuristic International Conference*. Porto, Portugal: [s.n.], 2001. v. 1, p. 273–276.

TAILLARD, E. D.; GAMBARDELLA, L. M.; GENDREAU, M.; POTVIN, J.-Y. Adaptive Memory Programming: A Unified View of Metaheuristics. *European Journal of Operational Research*, v. 135, n. 1, p. 1–16, 2001.

TARANTILIS, C.; KIRANOUDIS, C. A flexible adaptive memory-based algorithm for real-life transportation operations: Two case studies from dairy and construction sector. *European Journal of Operational Research*, v. 179, n. 3, p. 806–822, 2007.

TSPLIB95. Traveling Salesman Problem. In: . Heidelberg, Alemanha: Gerhard Reinelt, Universität Heidelberg, Institut für Informatik, 2007. Disponível em: <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html>. Acesso em: 17/09/2007.

VERHOEVEN, M. G. A.; AARTS, E. H. L. Parallel local search. *Journal of Heuristics*, v. 1, n. 1, p. 43–65, 1995.

VIEYRA, P. W. P. *Uma Metaheurísticas Genética Não Convencional e Ant Colony Systems para resolver o Problema do Caixeiro Viajante com Grupamentos*. 142 p. Dissertação (Mestrado) — Mestrado em Computação, Universidade Federal Fluminense, Niterói, RJ, 1999.

WEINTRAUB, A.; ABOUD, J.; FERNANDEZ, C.; LAPORTE, G.; RAMIREZ, E. An Emergency Vehicle Dispatching System for an Electric Utility in Chile. *Journal of the Operational Research Society*, v. 50, p. 690–696, 1999.

WHITLEY, D. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In: *Proceedings of the 3rd International Conference on Genetic Algorithms*. San Francisco, CA, EUA: J. D. Schaffer, editor, Morgan Kaufmann Publishers Inc., 1989. p. 116–123. ISBN 1-55860-066-3.

APÊNDICE A – Tipos de Instâncias

Este apêndice pretende orientar a utilização da biblioteca de instâncias disponíveis para o Problema do Caixeiro Viajante com Grupamentos (PCVG). Várias instâncias do Problema do Caixeiro Viajante (PCV) da biblioteca TSPLIB foram adaptadas para o PCVG. Para a biblioteca destinada ao PCVG foram geradas vários tipos de instâncias. A cada tipo de instância, uma descrição da forma de construí-la é realizada. O formato do arquivo correspondente para cada tipo é descrito e exemplos são mostrados.

Ao pesquisar o Problema do Caixeiro Viajante com Grupamentos (PCVG) observa-se que desde o artigo de (JONGENS; VOLGENANT, 1985) até o trabalho de (LAPORTE; POTVIN; QUILLERET, 1996) e as instâncias do Problema do Caixeiro Viajante com *BackHauls* (PCVB) (GENDREAU; HERTZ; LAPORTE, 1996), existe uma regra para criar as instâncias e não a disponibilidade destas. Neste sentido, cria-se um conjunto de instâncias de domínio público, onde adota-se a versão genérica sem fixação da ordem de visitas aos grupos. Todas as formas de instâncias geradas são instâncias euclidianas e simétricas. Através do endereço **ftp://ftp.cefetes.br/Teses_Dissertacoes/MarioMestria/instances/**, pode-se acessar as instâncias.

As instâncias geradas foram divididas em seis tipos. Duas formas de instâncias utilizam o trabalho de (JOHNSON; McGEOCH, 2002) disponível na literatura para o PCV: em um tipo agrupam-se os vértices em torno de um centro geométrico, com o total de centros geométricos igual a $n/100$ (n - número de vértices) e as coordenadas são escolhidas no intervalo $[0,10^6)$, denominada de *tipo 2* e no outro tipo as instâncias são geradas semelhantemente às do *tipo 2*, mas geradas com parâmetros diferentes, onde o total de centros geométricos é calculado pela relação n/m, sendo n o número de vértices e $m \in \mathbb{N}^*$ um parâmetro a ser escolhido. As coordenadas são definidas no intervalo $[0,10^3)$ e denomina-se esta instância de *tipo 5*. Uma terceira forma de gerar instâncias é através da interface *Concorde* (APPLEGATE et al., 2007) e denomina-se esta instância de *tipo 3*. Na quarta forma, as instâncias são geradas usando *layout* conforme proposto por (LAPORTE; POTVIN; QUILLERET, 1996) e ainda são introduzidos dois novos *layouts*, estas instâncias são

denominadas de *tipo 4*. Por último, duas outras formas de instâncias geradas são adaptações das instâncias do PCV (TSPLIB95, 2007). Em uma, os vértices são agrupados usando o algoritmo de clusterização *k-means*, denominada de *tipo 1* e em outra a área plana retangular que compõe os vértices da instância é dividida em diversos quadriláteros e cada quadrilátero corresponde a um grupo, denomina-se de instância do *tipo 6* .

Para cada tipo de instância existem arquivos gerados que armazenam o vértice i, as coordenadas x_i e y_i e o grupo a que o vértice pertence g_i. Todas as instâncias geradas utilizam distâncias euclidianas para o cálculo entre dois vértices i e j. O cálculo da distância euclidiana (d_{ij}) é dada por:

$$\begin{aligned} x_d &= x_i - x_j \\ y_d &= y_i - y_j \\ d_{ij} &= nint(\sqrt{x_d * x_d + y_d * y_d}), \end{aligned} \tag{A.1}$$

onde *nint* é o inteiro mais próximo.

Os nomes dos arquivos de todos os tipos de instância são constituídos de palavras representando campos da seguinte forma: (*campo*1-*campo*2-*campo*3-*campo*4.*campo*5). Os significados de cada campo são descritos a seguir:

1.*campo*1, contém a palavra ou letra: *instancia* para os *tipos 2*, *3* e *5*, *instanciaigual* para o *tipo 4* ou *i* para os *tipos 1* e *6*;

2.*campo*2, número de vértices n para instâncias do *tipo 3*, *4* e *5*, número de grupos para instâncias do *tipo 1* e *6* ou Clk.m para instância *tipo 2*, onde lk representa o total de vértices e $m \in \mathbb{N}$ a identificação de cada instância;

3.*campo*3, número de grupos para instância do *tipo 3*, *4* e *5*, nome de identificação na biblioteca TSPLIB para as instâncias *tipo 1* e *tipo 6* ou nulo para instância *tipo 2*;

4.*campo*4, *seed* para instância do *tipo 5*, *layout* para *tipo 4*, dimensão da área plana retangular para *tipo 6* ou nulo para *tipo 1*, *2* e *3*;

5.*campo*5, mostra a extensão:

i) *txt*, para instâncias do *tipo 1*, *2*, *4*, *5* e *6*;

ou

ii) *tsp*, para instâncias do *tipo 3*;

A.1 Instâncias geradas através de (JOHNSON; McGEOCH, 2002)

Para as instâncias do *tipo 2* os vértices são agrupados aleatoriamente em torno de um centro geométrico e o número de grupos é definido pela razão $n/100$. Assim, define-se o número de centro geométricos como sendo o total de grupos. As coordenadas do centros geométricos são geradas no intervalo $[0,10^6)$. Para cada um dos vértices é escolhido aleatoriamente um centro e duas variáveis com distribuição normal (Gaussiana), representando as coordenadas x e y dos vértices num plano cartesiano. Para cada uma das variáveis é multiplicada por $10^6/\sqrt{n}$, arredondando-as e adicionadas às coordenadas correspondentes aos seus centros designados. As novas coordenadas são de novo arredondadas ao inteiro mais próximo. A Figura A.1 ilustra um arquivo contendo os dados de uma instância do *tipo 2*.

```
NAME : portcgen-1000-10009
COMMENT : portcgen N=1000, seed=10009
TYPE : TSP
DIMENSION : 1000
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
1 909444 143969 7
2 515900 293384 8
3 749939 228922 3
...
...
997 389024 488295 10
998 319675 476919 2
999 893759 631434 1
1000 861328 323685 9
```

Figura A.1: Exemplo de um arquivo de uma instância do *tipo 2* com 1000 vértices distribuídos por 10 grupos, `instanciaC1k.9.txt`

A seguir, descreve-se como são realizadas a geração para as instâncias do *tipo 5* que são

semelhantes a do *tipo 2*, mas geradas com parâmetros diferentes. Seja, n o total de vértices nas instâncias. Escolhem-se n/m centros para os grupos com as coordenadas escolhidas uniformemente no intervalo $[0, 10^3)$, $m \in \mathbb{N}^*$. O número de grupos não é definido *a priori* e torna-se um parâmetro a ser escolhido pelovalor m. No final o arquivo contém em cada linha o vértice i, as coordenadas x_i e y_i, com $i = 1, 2, ..., n$ e o grupo que este vértice pertence g_j, com $j = 1, 2, ..., n/m$. A Figura A.2 mostra um arquivo de uma instância do *tipo 5* gerada com 1000 vértices e 10 grupos.

```
NOME : instancia-1000-10-45
COMENTARIO : geracaotamgruposdif N=1000, FATOR_de_GRUPO=100, valoraleatorio=45
TIPO : PCVG
DIMENSAO VERTICES : 1000
DIMENSAO GRUPO : 10
TIPO_PESOS_ARESTAS : EUC_2D
SECAO_COORDENADAS_VERTICES: vertice x y grupo
1 742 243 10
2 590 19 8
3 152 298 9
4 719 321 10
...
...
997 379 846 7
998 491 479 3
999 423 509 3
1000 733 336 10
```

Figura A.2: Exemplo de um arquivo de uma instância do *tipo 5* com 1000 vértices distribuídos por 10 grupos, `instancia-1000-10-45.txt`.

A Figura A.3 mostra a distribuição dos vértices nos eixos x e y para a instância `instancia-1000-10-45.txt` com os vértices centrados em torno do centro geométrico de seu grupo. Pode-se observar nesta instância o total de 10 grupos gerados.

Para a instância (`instancia-1000-10-45.txt`), a relação dos vértices por grupo é demostrada na Tabela A.1.

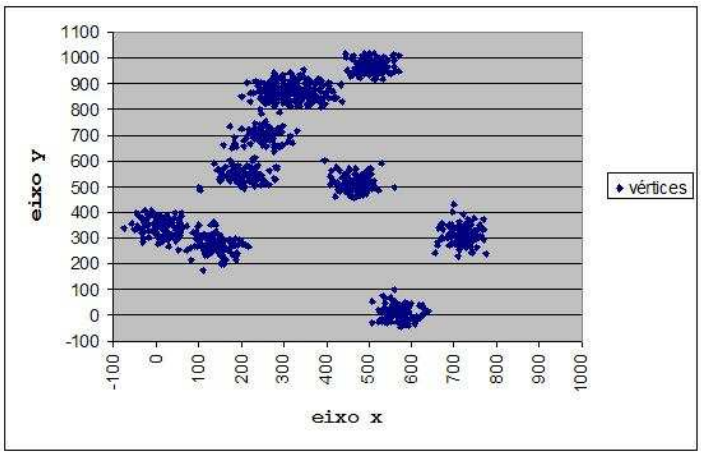


Figura A.3: Os vértices distribuídos nos eixos x e y em torno do centro geométrico da instância: `instancia-1000-10-45.txt`.

Grupo	Total de Vértices
1	98
2	100
3	96
4	103
5	90
6	107
7	86
8	104
9	109
10	107

Tabela A.1: Distribuição dos vértices de `instancia-1000-10-45.txt` a cada grupo.

A.2 Instâncias geradas utilizando (APPLEGATE et al., 2007)

As instâncias do *tipo 3* são geradas através do *software Concorde* (APPLEGATE et al., 2007). Este *software* é utilizado para resolver o PCV simétrico e alguns problemas de otimização em redes. Seu código é escrito em linguagem de programação C padrão ANSI e está disponível para pesquisas acadêmicas. Utiliza-se a interface do *Concorde* para gerar as intâncias do *tipo 3* ao PCVG.

Os vértices de cada grupo são gerados muito próximos uns dos outros e distantes de outros vértices distintos a este grupo. As coordenadas nesta interface são geradas sobre uma área retangular, cujo valores são números reais. Cada valor das coordenadas são arredondados para números inteiros mais próximo. A Figura A.4 mostra uma instância (`instancia-700-20.tsp`) gerada com 700 vértices e dividida em 20 grupos.

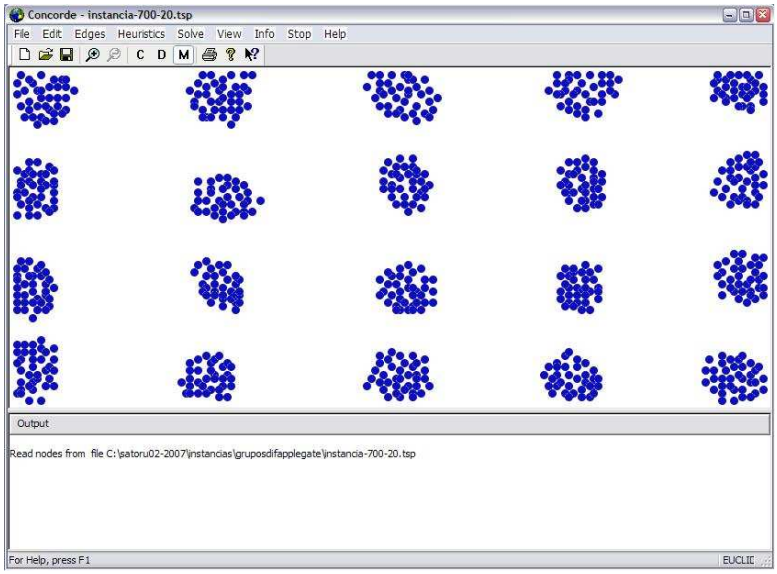


Figura A.4: Interface do *Concorde* gerando a instância: `instancia-700-20.tsp`

```
COMMENT: new type=CTSP by Mário Mestria
NAME: instancia700
TYPE: TSP
COMMENT: localizacoes=700
COMMENT: grupos=20
DIMENSION: 700
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 -48 100 1
2 -50 98 1
3 -52 96 1
. . .
. . .
698 114 25 20
699 117 26 20
700 122 26 20
```

Figura A.5: Exemplo de um arquivo de uma instância do *tipo 3* com 700 vértices distribuídos em 20 grupos, `instancia-700-20.tsp`.

A instância (`instancia-700-20.tsp`) da Figura A.5 mostram as coordenadas x_i e y_i dos vértices i=1, 2,..., 700 e a distribuição dos grupos g_j a que estes pertencem, j=1, 2,..., 20.

A.3 Instâncias geradas utilizando o trabalho de (LAPORTE; POTVIN; QUILLERET, 1996)

As instâncias do *tipo 4* utilizam os trabalhos de (GENDREAU; LAPORTE; POTVIN, 1994), (POTVIN; GUERTIN, 1996) e (LAPORTE; POTVIN; QUILLERET, 1996) que descrevem três tipos diferentes de *layout* para gerar as instâncias.

Neste tipo de instância, as coordenadas x_i e y_i, $i = 1, 2, ..., n$, onde n representa o total dos vértices, são geradas aleatoriamente sobre um plano cartesiano com distribuição uniforme sobre um intervalo $[0, N]$. Para cada coordenada x_i e y_i correspondendo a um vértice, existe um grupo associado representado por G_j, com $j = 1, 2, ..., m$. Em cada *layout* neste tipo de instância, o número de vértices em cada grupo é constante. Foram gerados vários tipos de diagramas que serão descritos a seguir, cujo valores de m foram iguais a 4, 8, 10 e 20. Cabe ressaltar que os valores de m para as instâncias geradas por (GENDREAU; LAPORTE; POTVIN, 1994) e (LAPORTE; POTVIN; QUILLERET, 1996) foram restritos a 4 e 10 e a ordem de visitas do grupos são definidas *a priori* e seus três tipos de diagramas utilizados são descritos a seguir:

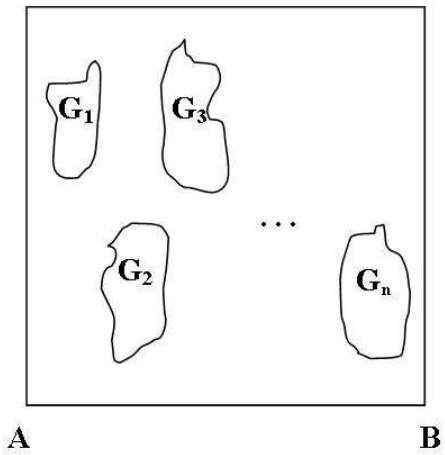


Figura A.6: Diagrama Aleatório

- Aleatório (A): Os vértices são gerados aleatoriamente numa área quadrada dispostas entre os pontos [A, B]^2, de acordo a uma distribuição uniforme. Então, os vértices

são designados aleatoriamente aos grupos, na ordem nos quais são gerados. A Figura A.6 demonstra a idéia deste diagrama.

•Horário (H): Os vértices pertencem a área quadrada $[A, B]^2$ dividida entre retângulos de tamanhos iguais e cada retângulo corresponde a um grupo. Os grupos são marcados seguindo um diagrama no sentido horário, Figura A.7. Os vértices são gerados aleatoriamente para cada retângulo, de acordo a uma distribuição uniforme.

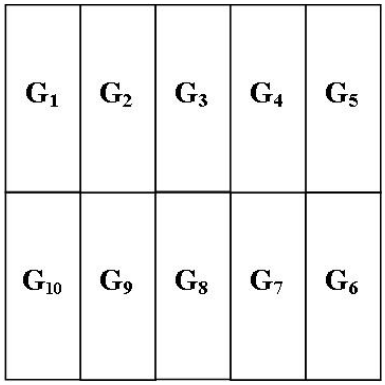


Figura A.7: Diagrama no Sentido Horário

•Zigzag (Z): Utiliza a mesma idéia do diagrama H, mas os vértices são divididos em grupos marcados seguindo um diagrama em *zigzag*, Figura A.8.

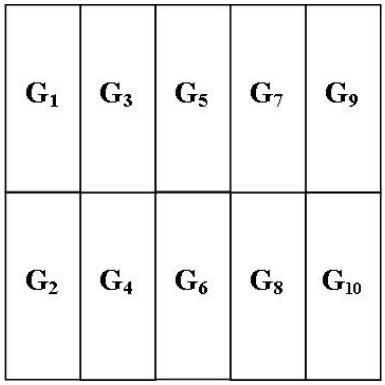


Figura A.8: Diagrama em Zigzag

No trabalho de (POTVIN; GUERTIN, 1996) a ordem de visitas dos grupos não foi definida inicialmente e faz parte da solução gerada pelo algoritmo. Os valores de m para as instâncias geradas, também permaneceram em 4 e 10. Utilizam-se neste trabalho os diagramas aleatório e horário, descritos acima, para realizar testes computacionais.

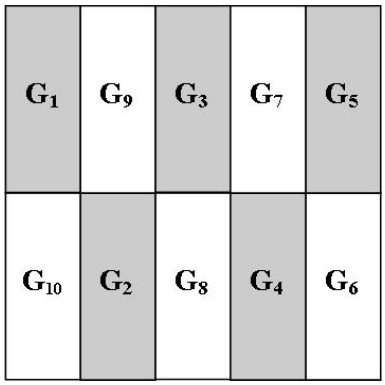


Figura A.9: Diagrama em Xadrez do tipo 1

Além dos três diagramas com *layout* (A), (H) e (Z) descritos anteriormente, foram introduzidos para as instâncias do *tipo 4* mais dois novos diagramas para testes computacionais, onde o número de vértices contidos em cada grupo permanecem com valores fixos e iguais. A seguir são descritos os dois tipos:

- Xadrez do tipo 1 (X1): A área quadrada, os retângulos e a distribuição uniforme dos vértices são semelhantes aos diagramas A, H e Z. É introduzido uma nova sequência de grupo ao diagrama. Este diagrama lembra um tabuleiro do jogo de xadrez, Figura A.9.

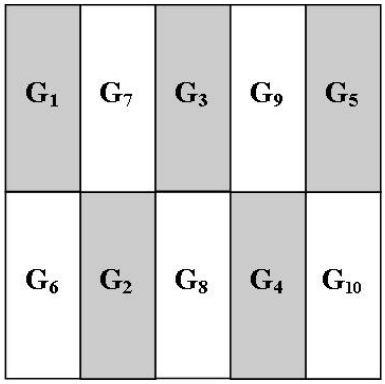


Figura A.10: Diagrama em Xadrez do tipo 2

- Xadrez do tipo 2 (X2): Semelhante ao X1, mas com a sequência do grupo modificada. Observa a disposição deste diagrama na Figura A.10, que os grupos G_i, i=1, 2, 3, 4 e 5 são mantidos iguais ao do diagrama *xadrez do tipo 1*. Os outros grupos mudam de posição.

Para todos os tipos de *layout* (A), (H), (Z), (X1) e (X2) foram geradas instâncias variando o número de vértices e número de grupos. Os grupos apresentam valores iguais

a quatro, oito, dez e vinte. A Figura A.11 mostra um arquivo de uma instância com 400 vértices e quatro grupos num diagrama horário.

```
400
1 269 44 4
2 137 428 4
3 452 13 4
4 445 254 4
5 268 33 4
6 423 474 4
...
...
395 923 733 2
396 918 625 2
397 504 512 2
398 624 555 2
399 833 709 2
400 864 996 2
```

Figura A.11: Exemplo de um arquivo de instância (`instanciaigual-400-4-h.txt`) do *tipo 4* com 400 vértices, quatro grupos e diagrama horário.

A.4 Instâncias geradas utilizando a biblioteca TSPLIB

Uma biblioteca de instâncias para o PCV (TSPLIB95, 2007) foi adaptada para gerar instâncias para o PCVG. Neste trabalho, duas formas de adaptação foram realizadas: a primeira nas instâncias do *tipo 1*, os vértices foram agrupados usando o algoritmo de clusterização *k-means* (KANUNGO et al., 2002) e na segunda, a área plana retangular que compõe os vértices da instância é dividida em diversos quadriláteros e cada quadrilátero corresponde a um grupo, instância do *tipo 6*. As coordenadas das instâncias para o PCV são mantidas fixas a ambas instâncias do PCVG.

Por exemplo, na instância do *tipo 1* do PCVG adaptada da instância `eil76.tsp` do PCV, as posições x_i e y_i dos vértices i=1, 2,..., 76 são mantidas. Após a adaptação a

instância torna-se `i-5-eil76.tsp.txt` com cinco grupos e é apresentada num arquivo na Figura A.12. Neste arquivo para esta instância do *tipo 1* são introduzidos os grupos g_i aos vértices i.

```
NAME : i-5-eil76.tsp.txt
COMENTARIO: k-means 'algkmeans' aplicado a instancia: eil76.tsp
TIPO : PCVG
DIMENSAO VERTICES : 76
DIMENSAO GRUPO : 5
TIPO_PESOS_ARESTAS: EUC_2D
SECAO_COORDENADAS_VERTICES: vertice x y grupo
1 22 22 4
2 36 26 4
3 21 45 2
4 45 35 5
5 55 20 3
6 33 34 4
...
...
71 59 5 3
72 35 60 1
73 27 24 4
74 40 20 3
75 40 37 5
76 40 40 5
```

Figura A.12: Exemplo de um arquivo de instância (`i-5-eil76.tsp.txt`) do *tipo 1* com 76 vértices e cinco grupos.

As Figuras A.13 e A.14 apresentam as instâncias do *tipo 1* num plano cartesiano mostrando as distribuições dos vértices de cada grupo e seus centróides. A instância da Figura A.13 contém 318 vértices e 10 grupos e a instância da Figura A.14 contém 442 vértices e 10 grupos.

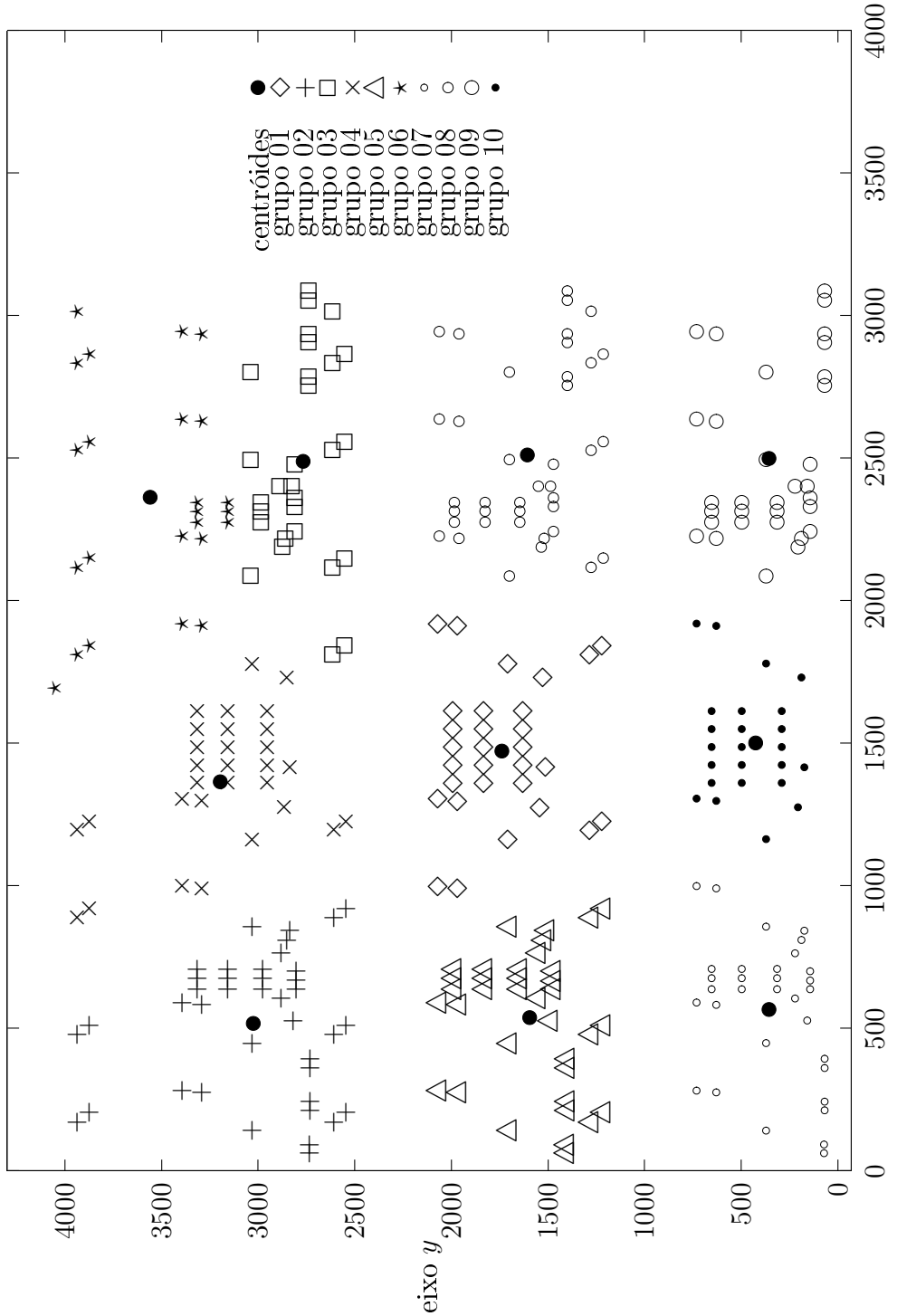


Figura A.13: Instância lin318.tsp agrupadas por *k-means* com 10 grupos.

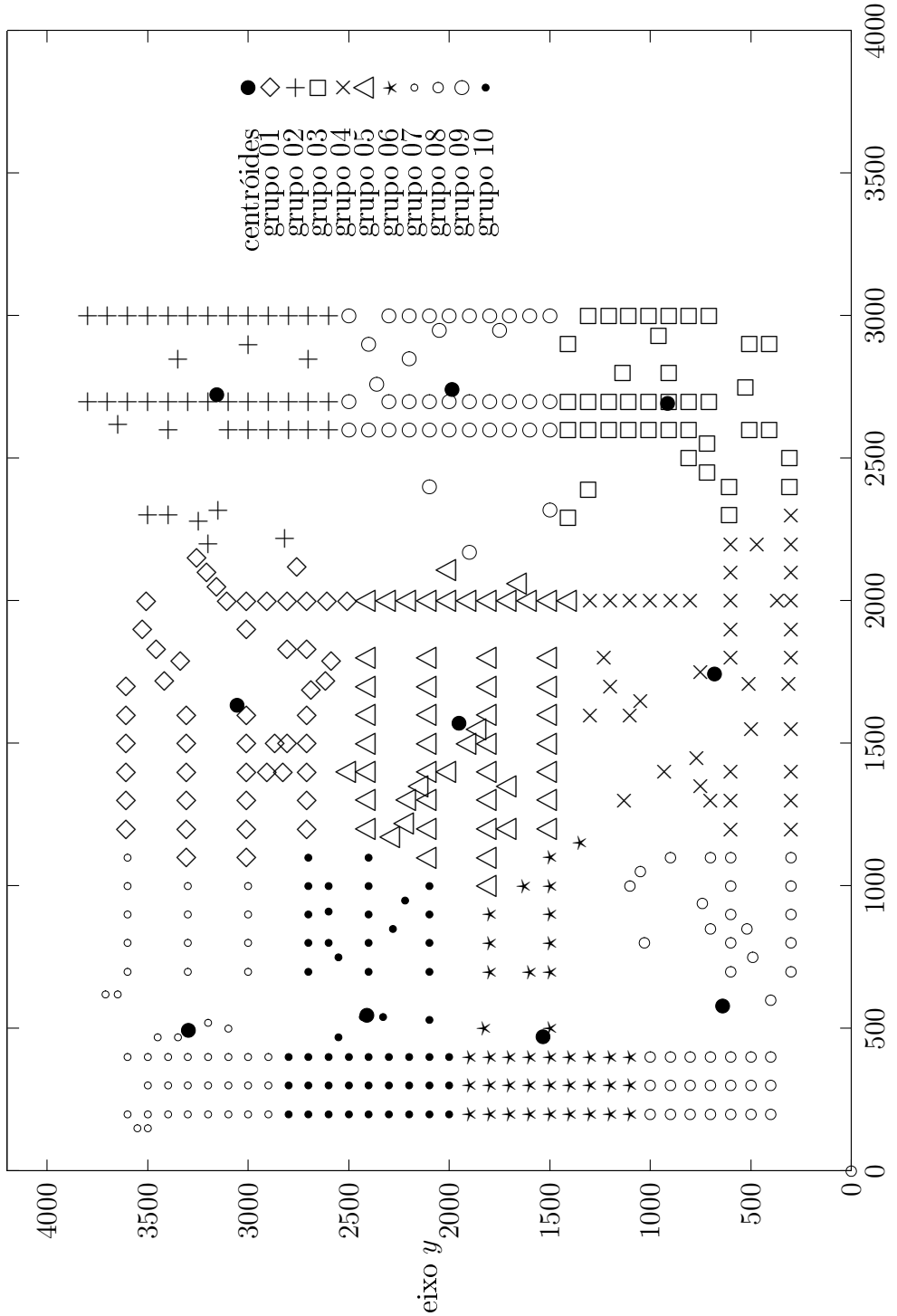


Figura A.14: Instância pcb442.tsp agrupadas por *k-means* com 10 grupos.

A Figura A.15 mostra um arquivo contendo dados da instância do *tipo 6*. Na maioria das instância do *tipo 6*, a divisão na horizontal é igual na vertical. Na instância da Figura A.15 apresenta seis divisões na horizontal e seis divisões na vertical com um total de 36 grupos.

```
NOME: i-36-pcb442.tsp-6x6.txt
COMENTARIO: quadrantes aplicado a instancia: pcb442.tsp
TIPO : PCVG
DIMENSAO VERTICES : 442
DIMENSAO GRUPO : 36
TIPO_PESOS_ARESTAS : EUC_2D
SECAO_COORDENADAS_VERTICES: vertice x y grupo
442
1 200 400 31
2 200 500 31
3 200 600 31
4 200 700 25
5 200 800 25
...
...
437 2300 3500 5
438 2320 3150 11
439 530 2100 14
440 2550 710 30
441 750 490 32
442 0 0 31
```

Figura A.15: Exemplo de uma instância (`i-36-pcb442.tsp-6x6.txt`) do *tipo 6* com 442 vértices e 36 grupos.

Existem instâncias do *tipo 6* que a divisão na horizontal é diferente da vertical. A Figura A.16 mostra a instância `i-12-pr76.tsp-3x4.txt` que apresenta três divisões na horizontal e quatro na vertical com o total de 12 grupos.

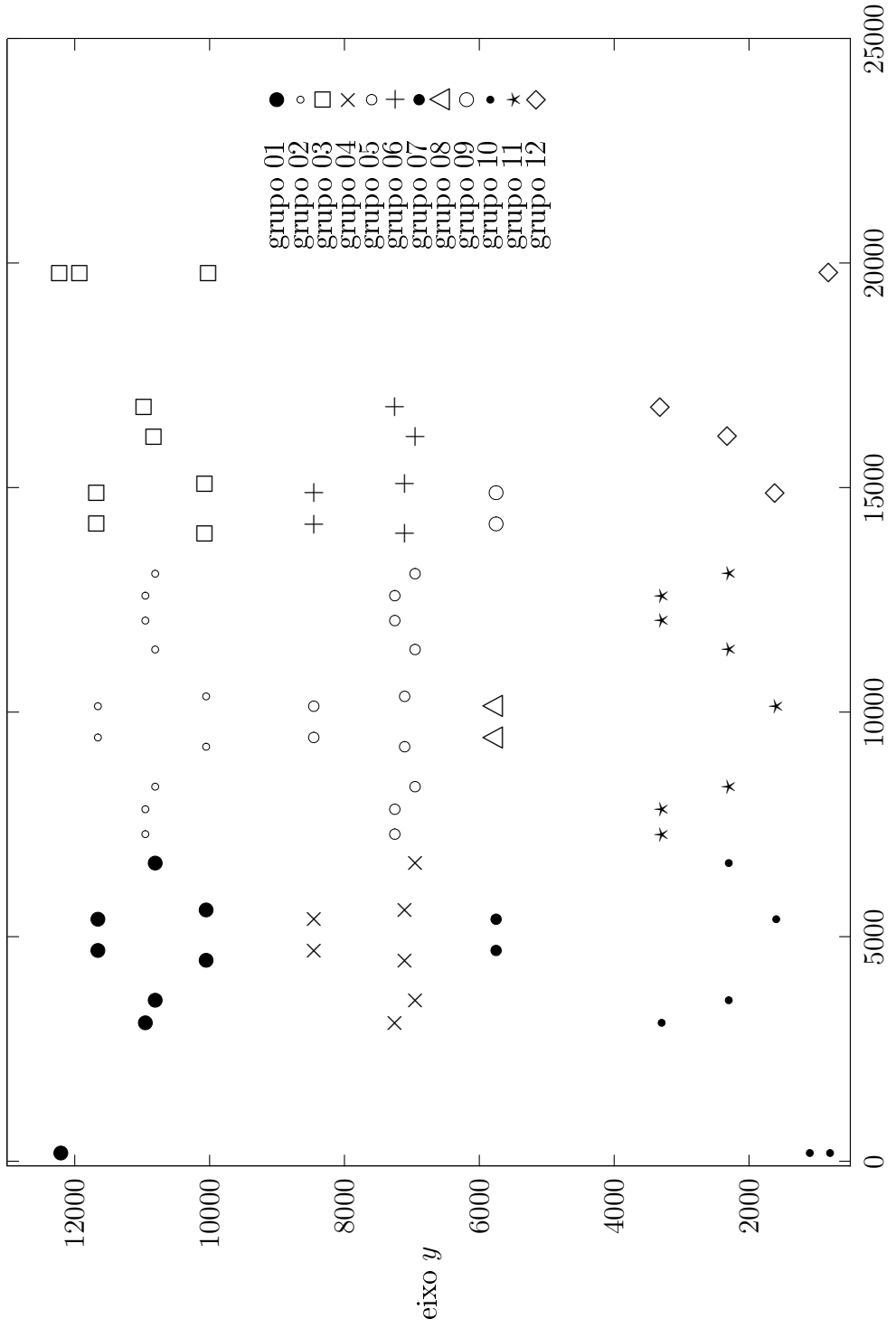


Figura A.16: Instância pr76.tsp dividida em 12 grupos.